

Fys4150

Project 5

Peter Killingstad and Karl Jacobsen

<https://github.com/kaaaja/fys4150>

November 26, 2017

Note to instructors regarding Github repository

If the above Github-link does not work, it is either because you have not yet accepted our invite to the repository, or you have not yet provided us with an e-mail address available at Github so that we can invite you. The Github user you will be invited from is "kaaaja". If the latter applies to you, please send us an e-mail with an e-mail address available in Github or your Github username so that we can send you an invite. Our e-mail addresses: peter.killingstad@hotmail.com, karljaco@gmail.com.

Abstract

Numerical finite difference schemes displaying the truncation error for the 1D (Forward Euler, Backward Euler and Crank-Nicolson) and the 2D (explicit and implicit) diffusion equation are derived and simulated. All schemes have truncation error that is 2nd order in space. Except Crank-Nicolson, which is 2nd order in time, all schemes are 1st order in time. Except for Forward Euler, all schemes are shown to be unconditionally stable. The Forward Euler scheme is found to be stable for $\Delta t/\Delta x^2 < 1/2$, in the 1D case, and stable for $\Delta t/h^2 < 1/4$ in the quadratic ($h = \Delta x = \Delta y$) 2D case. Analytical solutions for the 1D and the 2D diffusion equation are derived. All 1D schemes are implemented through the θ -rule. Linear system formulations are derived for all schemes. In 1D a more efficient version of the original Thomas algorithm is derived. The general fixed point iteration scheme, with conditions for convergence and FLOP count, is presented and related to the implemented Jacobi algorithm and the implemented Gauss-Seidel algorithm. These iterative methods are shown to be very efficient compared to direct schemes like Gaussian elimination. Gauss-Seidel is shown to be more efficient than Jacobi, Gauss-Seidel needing only 2/3 of the iterations of Jacobi. Parallelization by use of Open MP is used and shown to give a speed up of 4. It is shown that numerical evaluation of the analytical solution is far more demanding than the evaluation of the numerical schemes. Gaussian quadrature is applied for integrations. The simulations confirm that Forward Euler is sensitive around the stability limit, both in the 1D and the 2D case. In 1D, Crank-Nicolson outperforms the other schemes.

1 Introduction

Partial differential equations (PDEs) describes most physical processes. Analytical solutions to PDEs are only known in a few special cases. Numerical solutions of PDEs are indispensable in modelling of physical processes. In this report we will derive the most used schemes for one of the most used PDEs, the diffusion equation. We will develop finite difference schemes for solution of the 1D and the 2D diffusion equation. For the 1D case we derive and implement the Forward Euler scheme, the Backward Euler scheme and the Crank-Nicolson scheme. For the 2D case we derive and implement an explicit scheme and an implicit scheme.

Numerical schemes involve approximations, and this implies errors. Knowledge about a numerical method's error is of great importance. All schemes we derive, will be derived from Taylor expansions, implying that we get terms representing the order of the truncation errors (the errors that is made by leaving out terms in the Taylor series).

Numerical instability is another type of error that may occur due to approximations. A numerical scheme can give unphysical results, being unstable by e.g. producing unphysical oscillations. We perform Neuman stability analysis on all the derived schemes, identifying which schemes that are conditionally stable and what the stability requirements are.

Calculation of errors can be done in many ways. A very good method to calculate errors, is to compare the numerical solutions to analytical solutions. In our special case, we are so lucky that there exists known analytical solutions. Knowledge about how to derive these analytical solutions in itself is important. We derive, in full detail, the analytical solutions for both the 1D and 2D case.

The word "error" always comes up when discussing numerical schemes, and it is very often mentioned along with another word: "efficiency". A scheme can produce as small an error as possible, but it is worth little if it cannot produce the results in a reasonable amount of time. Hence methods improving the efficiency of the schemes are very central when dealing with numerical simulations. Our schemes can be, and are, written as linear systems. Typically for PDEs, these systems involve large and sparse matrices. Standard methods like Gaussian elimination for solving linear system becomes very time consuming. Alternatives are needed.

We improve the efficiency of the 1D schemes by utilizing that the schemes, when written as the general θ -scheme, are tridiagonal linear systems. The Thomas algorithm, which is much more efficient than e.g. Gaussian elimination and LU-decomposition, can be used for these systems. We further improve the efficiency of the Thomas algorithm by taking into account that all coefficients in the coefficient matrix of the linear system are constant.

For the 2D implicit scheme, the matrix is no longer tridiagonal, so the Thomas algorithm cannot be used anymore. Other efficient alternatives to direct methods like Gaussian elimination and LU-decomposition are necessary. Iterative methods is one answer. We set up a general iterative method, the fixed point iteration method, and present convergence conditions and FLOP counts for this method. It is seen that this method quickly has the potential of being much more efficient than e.g. Gaussian elimination. Further the Jacobi iteration method and the Gauss-Seidel iteration method are related to the fixed point iteration method, and then implemented to the implicit 2D scheme.

Improving the efficiency further, we parallelize parts of our code, using open MP. A timing analysis reveals that evaluation of the analytical solution takes up considerably more time than the numerical schemes! As a result, we parallelize the analytical solution. We also parallelize that Jacobi solver.

One reason for the high computational demands from evaluation of the analytical solution, is that there is a double integral that needs to be evaluated many times. In order to speed up the evaluation of this integral, we apply Gaussian quadrature with Legendre polynomials. Gaussian quadrature is known to be much more efficient compared to standard integration rules like e.g. Simpson's rule.

In the next section the analytical solution is derived, before we turn to derivation and error analysis of the numerical schemes. Finally results and conclusions follows.

2 Theory

This section contains derivations of analytical solutions and derivation and error analysis of the numerical schemes. But first, the physical problems are presented.

2.1 1D problem

The one-dimensional problem is written as

$$\frac{\partial^2 u(x, t)}{\partial x^2} = \frac{\partial u(x, t)}{\partial t}, \quad t > 0, \quad x \in (0, L) \quad (1a)$$

$$u(x, 0) = 0, \quad 0 < x < 1, \quad (1b)$$

$$u(0, t) = 0, \quad t > 0 \quad (1c)$$

$$u(L, t) = 1, \quad t > 0, \quad (1d)$$

where the length has been scaled by the length of the x -domain, L .

The problem consists of non-homogeneous boundary conditions, and it is not trivial to find a closed form solution. The problem can be seen as a physical problem such as a temperature gradient in a rod or flow between two infinite flat plates, where the fluid is initially at rest and the plate at $x = 1$ is given a sudden movement.

2.2 2D problem

In the 2D case, we choose to model a laminar flow inside an infinite tunnel with a finite quadratic cross section. We only model a single cross section, the problem being identical for all cross sections because of the homogeneity assumption in the flow direction.

There will be no slip boundary conditions on all four walls. The floor and the sides will be fixed, while the roof will be moving in the flow direction. With these boundary conditions we have zero velocity on the floor and the side walls, while we have a non-zero velocity in the flow direction. The figure below shows our problem.

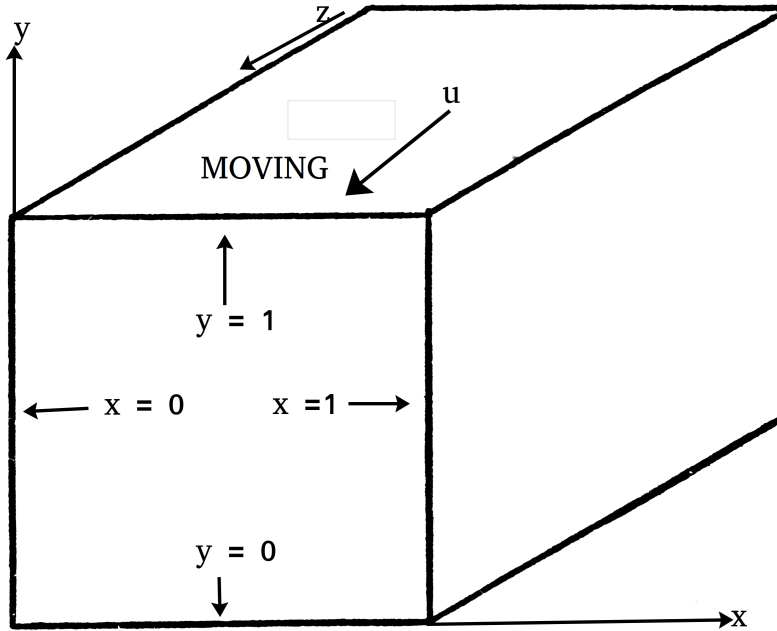


Figure 1: 2D problem sketch

With our choice of coordinate system, positive flow represents flow out of the paper.

We choose $u(x, y = 1) = 1$, and $u(x = 0, y) = u(x = 1, y) = u(x, y = 0) = 0$.

We model the flow with the heat equation. This equation can be derived from the equations of incompressible flow, the Navier-Stokes equations, by assuming laminar flow and a flow profile of type $u = u(x, y)$:

$$\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \nabla^2 \mathbf{u} \quad (2a)$$

$$w(x, y)_t + (w \frac{\partial}{\partial z})w(x, y) \stackrel{\text{Zero pressure gradient}}{=} (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})w(x, y) \quad (2b)$$

$$w(x, y)_t = (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2})w(x, y) \quad (2c)$$

Central assumptions behind the resulting heat equation is that there is no external pressure gradient and that the flow is laminar.

2.3 Analytical solutions

2.3.1 Analytical solution 1D

From physical observations we know that as time goes, i.e. when $t \rightarrow \infty$, the problem coincides with its surroundings and becomes steady. It is reasonable to introduce a solution consisting of the sum of two parts, a steady-state solution, and a transient solution that depends on the initial conditions:

$$u(x, t) = U(x) + V(x, t) \quad (3)$$

where $U(x)$ is the steady-state solution and $V(x, t)$ is the transient solution.

Since the steady-state problem is not changing in time, i.e. $\partial_t = 0$, it is written in a compact form as:

$$U_{xx} = 0, \quad x \in (0, 1) \quad (4a)$$

$$U(0) = 0 \quad (4b)$$

$$U(L) = 1 \quad (4c)$$

For the transient part of the problem we have the problem written in compact form as:

$$V_t = V_{xx}, \quad t > 0, \quad x \in (0, 1) \quad (5a)$$

$$V(x, 0) = -U(x), \quad 0 < x < 1 \quad (5b)$$

$$V(0, t) = 0, \quad t > 0 \quad (5c)$$

$$V(1, t) = 0, \quad t > 0 \quad (5d)$$

The steady state solution is solved by integrating (4a) twice

$$\begin{aligned} \int \frac{\partial^2 u(x, t)}{\partial x^2} dx &= A \\ \int \frac{\partial u(x, t)}{\partial x} dx &= \int A dx + B \end{aligned}$$

and we end up with the general solution $U(x) = Ax + B$. Applying the boundary conditions at $x = 0$ and $x = 1$ we get that

$$U(x) = x \quad (6)$$

The transient problem (5a) has homogeneous boundary conditions, and we can solve it by separation of variables. We start by making the ansatz that $V(x, t) = T(t)X(x)$ so that we can rewrite (5a) as:

$$XT' = X''T \quad (7)$$

Reordering the equation we get the following:

$$\frac{T'}{T} = \frac{X''}{X} = k, \quad (8)$$

where k is an unknown constant. The reason we can insert k into the above equation, is that the sides depends on different variables, and so equality between the sides is only achieved if both sides equals a constant. We have homogenous BCs, and for this to be achieved when we solve the X -part of the above equation, we must have $k < 0$. As can be seen later, with $k \geq 0$ we would not be able to satisfy the homogenous BCs. For computational ease, we set $k = -\lambda^2$, where $\lambda > 0$.

Starting with the t dependent part, we get

$$\frac{T'}{T} = -\lambda^2 \quad (9)$$

$$T' = -\lambda^2 T \quad (10)$$

The ODE can be solved by separation of variables

$$\frac{1}{T} \frac{dT}{dt} = -\lambda^2 \quad (11a)$$

$$\Rightarrow \frac{1}{T} dT = -\lambda^2 dt \quad (11b)$$

$$\int \frac{1}{T} dT = - \int \lambda^2 dt \quad (11c)$$

$$\Rightarrow \ln(T) = -\lambda^2 t + A \quad (11d)$$

The t dependent term is then given by

$$T = Ae^{-\lambda^2 t} \quad (12)$$

We see that our assumption $\lambda > 0$ makes sense also for the T -solution, since we do not get blow-up as $t \rightarrow \infty$.

For the x dependent term we get

$$X'' = -\lambda^2 X \quad (13a)$$

$$\Rightarrow X'' + \lambda^2 X = 0 \quad (13b)$$

We use an ansatz $X = e^{\alpha x}$:

$$\alpha^2 e^{\alpha x} + \lambda^2 e^{\alpha x} = 0 \quad (14a)$$

$$\Rightarrow \alpha^2 = -\lambda^2 \quad (14b)$$

$$\Rightarrow \alpha = \pm \sqrt{-\lambda^2} \quad (14c)$$

Since we have $\lambda > 0$, we get our wanted trigonometric solution

$$X = Be^{-i\lambda x} + Ce^{i\lambda x}, \quad (15)$$

which can be written as

$$X(x) = B \cos(\lambda x) + C \sin(\lambda x) \quad (16)$$

,

where the constants have changed.

Applying the boundary conditions we get that

$$X(0) = B \cos(0) + C \sin(0) = 0 \quad (17)$$

$$X(1) = B \cos(\lambda) + C \sin(\lambda) = 0 \quad (18)$$

From the first boundary condition where $x = 0$ we must have that $B = 0$. We are looking for non-trivial solutions of the problem, so for $x = 1$ we must find the values that gives $C \sin(\lambda) = 0$. We know that $\sin(n\pi) = 0$ for $n = 1, 2, \dots$, which lets us determine that $\lambda = n\pi$.

Summing up, we have the following particular solutions for the transient part:

$$V_n(x, t) = A_n e^{-(n\pi)^2 t} \sin(n\pi x), n = 1, 2, \dots, \infty \quad (19a)$$

Since all particular solutions satisfy the PDE, and the BCs are homogenous, all linear combinations of the particular solutions will also satisfy our problem, giving the general solution of the transient problem

$$\sum_{n=1}^{\infty} a_n e^{-(n\pi)^2 t} \sin(n\pi x) \quad (20)$$

The coefficients a_n is found by applying the initial condition (IC)

$$-U(x) = \sum_{n=1}^{\infty} a_n \sin(n\pi x) \quad (21)$$

To obtain an expression for the coefficients a_n we use that the functions $\sin(n\pi x)$ is orthogonal to each other in the sense that

$$\int_0^1 \sin(n\pi x) \sin(m\pi x) dx = \begin{cases} 0 & \text{if } m \neq n \\ 1/2 & \text{if } m = n \end{cases} \quad (22)$$

Using the above statement, multiplying 21 by $\sin(m\pi x)$ and then integrating over x over the domain, we get

$$\begin{aligned} -\int_0^1 U(x) \sin(m\pi x) dx &= a_m \int_0^1 \sum_{n=1}^{\infty} \sin(m\pi x) \sin(n\pi x) dx = a_m \int_0^1 \sin^2(m\pi x) dx \\ &\Rightarrow -\int_0^1 U(x) \sin(m\pi x) dx = \frac{a_m}{2} \\ &\Rightarrow a_m = -2 \int_0^1 U(x) \sin(m\pi x) dx \end{aligned} \quad (23)$$

Solving for a_n , applying the steady state solution (6), yields

$$\begin{aligned} a_n = -2 \int_0^1 x \sin(m\pi x) dx &= \left[\frac{2}{(m\pi)^2} \sin(m\pi x) + \frac{2x}{m\pi} \cos(m\pi x) \right]_0^1 \\ &\Rightarrow a_n = \frac{2}{m\pi} (-1)^m \end{aligned} \quad (24)$$

Putting the coefficient from (24) into the fundamental solutions (20) we end up with the following solution for the transient problem:

$$V(x, t) = 2 \sum_{n=1}^{\infty} \frac{(-1)^n}{n\pi} e^{-(n\pi)^2 t} \sin(n\pi x) \quad (25)$$

Now we get an expression for the whole problem by putting combining the steady-state solution (6) and the transient solution (25) such that we obtain

$$u(x, t) = U(x) + V(x, t) = x + 2 \sum_{n=1}^{\infty} e^{-(n\pi)^2 t} \frac{(-1)^k}{n\pi} \sin(n\pi x) \quad (26)$$

2.3.2 Analytical solution 2D

In the two-dimensional case the differential equation becomes

$$\frac{\partial^2 u(x, y, t)}{\partial x^2} + \frac{\partial^2 u(x, y, t)}{\partial y^2} = \frac{\partial u(x, y, t)}{\partial t}, \quad t > 0, \quad x, y \in (0, 1) \quad (27a)$$

$$u(x, y, 0) = 0, \quad x, y \in (0, 1) \quad (27b)$$

$$u(0, y, t) = 0, \quad t > 0 \quad (27c)$$

$$u(1, y, t) = 0, \quad t > 0 \quad (27d)$$

$$u(x, 0, t) = 0, \quad t > 0 \quad (27e)$$

$$u(x, 1, t) = 1, \quad t > 0 \quad (27f)$$

The boundary conditions is extended in such a way that the physical problem can be that of a flow within a box with infinite plates in the streamwise direction (z), where the fluid is initially at rest and the plate at $y = 1$ is given a sudden movement. We have "no slip" boundary conditions i.e. the fluid has zero movement on all boundaries except at $y = 1$.

To obtain a closed form solution for the two-dimensional, we use the same argument as for the one-dimensional problem, i.e. we split the solution into a steady-state solution and a transient solution.

We end up with a solution defined as:

$$u(x, y, t) = U(x, y) + V(x, y, t) \quad (28)$$

where $U(x, y)$ is the steady-state solution and $V(x, y, t)$ is the transient solution.

The steady-solution is then written in a compact form as:

$$U_{xx} + U_{yy} = 0, \quad x, y \in [0, 1] \quad (29a)$$

$$U(0, y) = 0 \quad (29b)$$

$$U(1, y) = 0 \quad (29c)$$

$$U(x, 0) = 0 \quad (29d)$$

$$U(x, 1) = 1 \quad (29e)$$

For the transient problem we have the problem defined as:

$$V_t = V_{xx} + V_{yy}, \quad t > 0, \quad x \in [0, 1] \quad (30a)$$

$$V(x, 0) = -U(x, y), \quad x, y \in [0, 1] \quad (30b)$$

$$V(0, y, t) = 0, \quad t > 0 \quad (30c)$$

$$V(1, y, t) = 0, \quad t > 0 \quad (30d)$$

$$V(x, 0, t) = 0, \quad t > 0 \quad (30e)$$

$$V(x, 1, t) = 0, \quad t > 0 \quad (30f)$$

For the steady-state problem we have a 2D Laplacian equation, which we can solve by separation of variables. We use the ansatz that $U(x, y) = X(x)Y(y)$, so that we end up with:

$$YX'' + XY'' = 0$$

$$\frac{X''}{X} = -\frac{Y''}{Y} = -\beta^2$$

Due to the independence of the terms on each side of the equation, we can solve the equations separately, each equation becoming

$$\frac{X''}{X} = -\beta^2$$

$$\frac{Y''}{Y} = \beta^2$$

The sign on the right hand side in both of the above equations is determined by the fact that in the x dependent term we have homogeneous boundaries, while in the y dependent term we have non-homogeneous. By the same reasoning as in the 1D-case, homogenous BCs in the x -direction, we get $\beta > 0$.

For the x dependent term we get

$$X'' = -\beta^2 X$$

$$\Rightarrow X'' + \beta^2 X = 0,$$

which has a similar solution as that of the transient x dependent term in the one-dimensional case (15). With our choice of β , we get the wanted trigonometric form

$$X(x) = A \cos(\beta x) + B \sin(\beta x).$$

Applying the boundary conditions, we end up with the term

$$X_n(x) = B_n \sin(n\pi x), \tag{31}$$

where $X_n(x)$ is the family of particular solutions.

The y dependent term can be rewritten as

$$y'' = \beta^2 Y \tag{32a}$$

$$\Rightarrow Y'' - \beta^2 Y = 0 \tag{32b}$$

We are again using an ansatz so that $Y = e^{\gamma y}$. This leaves us with the term

$$\gamma^2 e^{\gamma y} - \beta^2 e^{\gamma y} = 0 \tag{33a}$$

$$\Rightarrow \gamma^2 = \beta^2 \tag{33b}$$

$$\Rightarrow \gamma = \pm \beta, \tag{33c}$$

and we end up with the following expression

$$Y_n(y) = C_n e^{-n\pi y} + D_n e^{n\pi y} \tag{34}$$

$$\Rightarrow C_n \cosh(n\pi y) + D_n \sinh(n\pi y) \tag{35}$$

for the family of particular solutions.

The fundamental solutions of the steady-state is the given by

$$U(x, y) = \sum_{n=1}^{\infty} X(x)Y(y) = \sum_{n=1}^{\infty} B_n \sin(n\pi x) \left(C_n \cosh(n\pi y) + D_n \sinh(n\pi y) \right) \tag{36}$$

Applying the the boundary of $U(x, 0)$ to the equation above we get that C_n must be zero. We are then left with the expression

$$U(x, y) = \sum_{n=1}^{\infty} c_n \sin(n\pi x) \sinh(n\pi y) \quad (37)$$

Applying the last boundary $U(x, 1) = 1$ we must have that

$$1 = \sum_{n=1}^{\infty} c_n \sin(n\pi x) \sinh(n\pi) \quad (38)$$

setting $d_n = c_n \sinh(n\pi)$ we get that

$$1 = \sum_{n=1}^{\infty} d_n \sin(n\pi x) \quad (39)$$

Using again that the functions $\sin(n\pi x)$ is orthogonal (22), as in the 1D case, multiplying the above equation with $\sin(m\pi x)$ and then integrating over x over the domain, we get that

$$d_n = 2 \int_0^1 \sin(m\pi x) dx = \frac{2}{m\pi} (1 - (-1)^m) \quad (40a)$$

$$\Rightarrow c_n = \frac{2}{m\pi \sinh(m\pi)} (1 - (-1)^m). \quad (40b)$$

Putting the coefficients c_n back into the expression for the fundamental solutions (37), we end up with the following expression for the steady-state

$$U(x, y) = \frac{2}{\pi} \sum_{m=1}^{\infty} \frac{\sin(m\pi x) \sinh(m\pi y)}{\pi \sinh(m\pi)} (1 - (-1)^m) \quad (41a)$$

$$\Rightarrow U(x, y) = \frac{4}{\pi} \sum_{m=1}^{\infty} \frac{\sin((2m-1)\pi x) \sinh((2m-1)\pi y)}{\pi \sinh((2m-1)\pi)}. \quad (41b)$$

For the transient problem we once again use the anzats that $V(x, y, t) = X(x)Y(y)T(t)$ in order to write the problem as

$$XYT' = YTX'' + XTY'' \quad (42a)$$

$$\Rightarrow \frac{T'}{T} = \frac{X''}{X} + \frac{Y''}{Y} = -k^2. \quad (42b)$$

Starting by solving the t dependent term, which is solved in the same way as for the one-dimensional term (11a), we end up with the expression

$$T = Ae^{-k^2 t}. \quad (43)$$

For the x dependent part we have

$$\frac{X''}{X} = -\frac{Y''}{Y} - k^2 \quad (44a)$$

$$\Rightarrow \frac{X''}{X} = -\gamma^2, \quad (44b)$$

where

$$-\gamma^2 = \frac{Y''}{Y} - k^2. \quad (45)$$

(44a) has the same solution form as the x dependent transient one-dimensional term (15), as well as the steady state two-dimensional term (31), so we end up with

$$X(x) = B \cos(\gamma x) + C \sin(\gamma x) \quad (46)$$

For the above equation to hold on the boundaries, we get $B = 0$ and $\gamma = n\pi, n = 1, 2, \dots, \infty$, so we end up with

$$X_n(x) = c_n \sin(n\pi x) \quad (47)$$

We are left with the y dependent term, which is written as

$$\frac{Y''}{Y} = k^2 - \gamma^2 \quad (48a)$$

Solving for Y we again use the anzats $Y = e^{\alpha y}$ and obtain the term

$$\alpha^2 e^{\alpha y} + (\gamma^2 - k^2) e^{\alpha y} = 0 \quad (49a)$$

$$\alpha^2 + \gamma^2 - k^2 = 0 \quad (49b)$$

$$\Rightarrow \alpha = \pm \sqrt{k^2 - \gamma^2} \quad (49c)$$

We are looking for a solution on the form that satisfies the homogenupus BCs for the transient term, resulting in

$$Y = D \cos(\sqrt{k^2 - \gamma^2} y) + E \sin(\sqrt{k^2 - \gamma^2} y) \quad (50)$$

For the above equation to satisfy the boundary conditions $Y(0) = 0$ and $Y(1) = 0$, we get that $D = 0$ and that $\sqrt{k^2 - \gamma^2} = m\pi, m = 1, 2, \dots, \infty$. The Y term is then given by

$$Y_m(y) = E_m \sin(m\pi y) \quad (51)$$

We can now calculate the constant k^2 , $k^2 = \gamma^2 + (m\pi)^2 = (n\pi)^2 + (m\pi)^2$. Putting this back into the t dependent transient term (43) we have that

$$T = A_{nm} e^{-\pi^2(m^2+n^2)t}. \quad (52)$$

Combining the x, y and t dependent terms ((47), (51) and (52) respectively), we end up with the fundamental solution

$$V(x, y, t) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} A_{mn} \sin(n\pi x) \sin(m\pi y) e^{-\pi^2(m^2+n^2)t}. \quad (53)$$

To find the coefficients A_{nm} , we use the initial condition

$$-U(x, y) = \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} A_{mn} \sin(n\pi x) \sin(m\pi y) \quad (54)$$

We can solve for the coefficient by again using the orthogonality from (22), multiplying the equation above by $\sin(p\pi x) \sin(q\pi y)$, and integrating over the x and y domain to get

$$\int_0^1 \int_0^1 -U(x, y) \sin(p\pi x) \sin(q\pi y) dx dy = \frac{A_{pq}}{4} \quad (55a)$$

$$\Rightarrow A_{pq} = -4 \int_0^1 \int_0^1 U(x, y) \sin(p\pi x) \sin(qn\pi y) dx dy. \quad (55b)$$

Finally we can combine the steady-state solution and the transient solution in order to get the analytical expression

$$\begin{aligned} u(x, y, t) &= U(x, y) + V(x, y, t) \\ &= U(x, y) + \sum_{n=1}^{\infty} \sum_{m=1}^{\infty} A_{mn} \sin(n\pi x) \sin(m\pi y) e^{-\pi^2(m^2+n^2)t}, \end{aligned} \quad (56)$$

where

$$U(x, y) = \frac{4}{\pi} \sum_{m=1}^{\infty} \frac{\sin((2m-1)\pi x) \sinh((2m-1)\pi y)}{\pi \sinh((2m-1)\pi)} \quad (57)$$

and

$$A_{mn} = -4 \int_0^1 \int_0^1 U(x, y) \sin(m\pi x) \sin(n\pi y) dx dy. \quad (58)$$

2.4 Numerical schemes

All the schemes will be derived from Taylor series expansions, and the truncation error will be related to the remainder in the Taylor-series expansions. This remainder is the error we get when truncating the series by leaving out the remainder.

2.4.1 Forward Euler

For the time derivative, we expand $u(x, t + \Delta t)$ around t

$$u(x, t + \Delta t) = u(x, t) + u_t(x, t)\Delta t + \mathcal{O}(\Delta t^2) \quad (59a)$$

$$\rightarrow u_t(x, t) = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + \mathcal{O}(\Delta t) \quad (59b)$$

The space derivative, which is a 2nd derivative, we derive by combining two Taylor series'

$$u(x + \Delta x, t) = u(x, t) + u_x(x, t)\Delta x + \frac{u_{xx}(x, t)\Delta x^2}{2} + \frac{u_{xxx}(x, t)\Delta x^3}{6} + \mathcal{O}(\Delta x^4) \quad (60a)$$

$$u(x - \Delta x, t) = u(x, t) - u_x(x, t)\Delta x + \frac{u_{xx}(x, t)\Delta x^2}{2} - \frac{u_{xxx}(x, t)\Delta x^3}{6} + \mathcal{O}(\Delta x^4) \quad (60b)$$

Now we add (60a) and (60b) and solve for $u_{xx}(x, t)$

$$\begin{aligned} (u(x + \Delta x, t) + u(x - \Delta x, t)) &= (u(x, t) + u(x, t)) \\ &+ (u_x(x, t)\Delta x + (-u_x(x, t)\Delta x)) \\ &+ \left(\frac{u_{xx}(x, t)\Delta x^2}{2} + \frac{u_{xx}(x, t)\Delta x^2}{2} \right) \end{aligned} \quad (61a)$$

$$\begin{aligned} &+ \left(\frac{u_{xxx}(x, t)\Delta x^3}{6} + (-\frac{u_{xxx}(x, t)\Delta x^3}{6}) \right) \\ &+ (\mathcal{O}(\Delta x^4) + \mathcal{O}(\Delta x^4)) \end{aligned}$$

$$= 2u(x, t) + u_{xx}(x, t)\Delta x^2 + \mathcal{O}(\Delta x^4) \quad (61b)$$

$$\rightarrow u_{xx}(x, t) = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad (61c)$$

Combining (59b) and (61c) we get the Forward Euler scheme

$$u_t(x, t) = u_{xx}(x, t) \quad (62a)$$

$$\frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + \mathcal{O}(\Delta t) = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad (62b)$$

From (62b) we see that the scheme has a truncation error that goes like $\mathcal{O}(\Delta t)$ in time and $\mathcal{O}(\Delta x^2)$ in space.

We will analyze the stability of the Forward Euler scheme (62b) by applying Neuman stability analysis. From the analytical solution of the problem, we know that the particular solutions are on the form $u = e^{-(k\pi)^2 t} e^{ik\pi x}$, where k is an integer greater than one. We observe that the solutions are stable in t , meaning that the solutions do not blow up as t increases. Based on the analytical particular solution, we make the numerical ansatz

$$u = a_k^n e^{ik\pi x_j} \quad (63)$$

For the numerical ansatz (63) to reproduce the characteristics of the analytical particular solution, with stability in t , we observe that $|a_k^n| < 1$ is necessary. We now plug in the ansatz (63) into the (62b) and derive an equation for $|a_k^n|$:

$$\frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \quad (64a)$$

$$\frac{a_k^{n+1} e^{ik\pi(j+1)\Delta x} - a_k^n e^{ik\pi j\Delta x}}{\Delta t} = \frac{a_k^n e^{ik\pi(j-1)\Delta x} - 2a_k^n e^{ik\pi j\Delta x} + a_k^n e^{ik\pi(j+1)\Delta x}}{\Delta x^2} \quad (64b)$$

$$a_k^n e^{ik\pi j\Delta x} \frac{a_k - 1}{\Delta t} = a_k^n e^{ik\pi j\Delta x} \frac{e^{-ik\pi\Delta x} - 2 + e^{ik\pi\Delta x}}{\Delta x^2} \quad (64c)$$

$$\frac{a_k - 1}{\Delta t} = \frac{e^{-ik\pi\Delta x} - 2 + e^{ik\pi\Delta x}}{\Delta x^2} \quad (64d)$$

$$a_k = 1 + \frac{\Delta t}{\Delta x^2} (e^{-ik\pi\Delta x} - 2 + e^{ik\pi\Delta x}) \quad (64e)$$

$$= 1 + \frac{\Delta t}{\Delta x^2} (2 \cos(k\pi\Delta x) - 2) \quad (64f)$$

$$= 1 + 2 \frac{\Delta t}{\Delta x^2} (\cos(k\pi\Delta x) - 1) \quad (64g)$$

$$= 1 + 2 \frac{\Delta t}{\Delta x^2} \left(-2 \sin^2\left(\frac{k\pi\Delta x}{2}\right) \right) \quad (64h)$$

$$= 1 - 4 \frac{\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi\Delta x}{2}\right) \quad (64i)$$

$$|a_k| = \left| 1 - 4 \frac{\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi\Delta x}{2}\right) \right| \quad (64j)$$

From (64j) we get

$$|a_k| < 1 \text{ if } \left| 1 - 4 \frac{\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi\Delta x}{2}\right) \right| < 1 \quad (65a)$$

$$\rightarrow \left| 1 - 4 \frac{\Delta t}{\Delta x^2} \right| < 1 \rightarrow |a_k| < 1 \text{ (Since } \sin^2(k\pi\Delta x/2)_{max} = 1) \quad (65b)$$

$$\rightarrow 1 - 4 \frac{\Delta t}{\Delta x^2} > -1 \quad (65c)$$

$$\rightarrow \frac{\Delta t}{\Delta x^2} < \frac{1}{2} \quad (65d)$$

(65d) gives that the Forward Euler scheme is conditionally stable, and the condition that ensures stability.

2.4.2 Backward Euler

Here we will do the same as we did for Forward Euler above: Derive the scheme, including truncation errors, and analyze stability.

The only change compared to Forward Euler, is the time discretization, which now becomes

$$u(x, t - \Delta t) = u(x, t) + u_t(x, t)\Delta t - \mathcal{O}(\Delta t^2) \quad (66a)$$

$$\rightarrow u_t(x, t) = \frac{u(x, t) - u(x, t - \Delta t)}{\Delta t} + \mathcal{O}(\Delta t) \quad (66b)$$

The space discretization is the same as for Forward Euler, (61c). Combining the space discretization (61c) and (66b) gives

$$\frac{u(x, t) - u(x, t - \Delta t)}{\Delta t} + \mathcal{O}(\Delta t) = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad (67a)$$

We note that the truncation errors have the same asymptotic behavior as for the Forward Euler scheme.

To analyze the stability of the Backward Euler scheme, we apply the same method as we did for Forward Euler and insert the ansatz (63) into (67a) to get

$$\frac{u(x, t) - u(x, t - \Delta t)}{\Delta t} = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \quad (68a)$$

$$\frac{a_k^n e^{ik\pi j \Delta x} - a_k^{n-1} e^{ik\pi j \Delta x}}{\Delta t} = \frac{a_k^n e^{ik\pi(j-1)\Delta x} - 2a_k^n e^{ik\pi j \Delta x} + a_k^n e^{ik\pi(1+j)\Delta x}}{\Delta x^2} \quad (68b)$$

$$a_k^n e^{ik\pi j \Delta x} \frac{1 - a_k^{-1}}{\Delta t} = a_k^n e^{ik\pi j \Delta x} \frac{e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}}{\Delta x^2} \quad (68c)$$

$$\frac{1 - a_k^{-1}}{\Delta t} = \frac{e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}}{\Delta x^2} \quad (68d)$$

$$a_k^{-1} = 1 - \frac{\Delta t}{\Delta x^2} (e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}) \quad (68e)$$

$$a_k = \frac{1}{1 - \frac{\Delta t}{\Delta x^2} (e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x})} \quad (68f)$$

$$\stackrel{(64)}{=} \frac{1}{1 + 4 \frac{\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right)} \quad (68g)$$

$$|a_k| = \left| \frac{1}{1 + 4 \frac{\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right)} \right| < 1. \quad (68h)$$

$$(68i)$$

From (68h) we see that, in contrast to the Forward Euler scheme, the Backward Euler scheme is unconditionally stable.

(67a) reveals another difference between the Backward Euler scheme and the Forward Euler scheme: (67a) is implicit in $u(x, t)$, meaning that we cannot solve (67a) directly for $u(x, t)$, as we did in the Forward Euler scheme. However, we can find $u(x, t)$ from (67a) by recognizing that (67a) can be rewritten as a linear system:

$$\frac{u(x, t) - u(x, t - \Delta t)}{\Delta t} = \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \quad (69a)$$

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2} \quad (69b)$$

$$\frac{\Delta x^2}{\Delta t} (u_i^n - u_i^{n-1}) = u_{i-1}^n - 2u_i^n + u_{i+1}^n \quad (69c)$$

$$-\left(u_{i-1}^n - \left(2 + \frac{\Delta x^2}{\Delta t}\right)u_i^n + u_{i+1}^n\right) = \frac{\Delta x^2}{\Delta t} u_i^{n-1} \quad (69d)$$

$$\left(-u_{i-1}^n + \left(2 + \frac{\Delta x^2}{\Delta t}\right)u_i^n - u_{i+1}^n\right) = \frac{\Delta x^2}{\Delta t} u_i^{n-1} \quad (69e)$$

$$\underbrace{\begin{bmatrix} 2 + \frac{\Delta x^2}{\Delta t} & -1 & \cdots & 0 \\ -1 & 2 + \frac{\Delta x^2}{\Delta t} & -1 & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & -1 & 2 + \frac{\Delta x^2}{\Delta t} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_N^n \end{bmatrix}}_{\mathbf{U}} = \frac{\Delta x^2}{\Delta t} \underbrace{\begin{bmatrix} u_0^{n-1} \\ u_1^{n-1} \\ \vdots \\ u_N^{n-1} \end{bmatrix}}_{\mathbf{b}} \quad (69f)$$

We see from (69f) that solving Backward Euler corresponds to solving a linear system $AU = \tilde{b}$, where A is a tridiagonal matrix.

2.4.3 Crank-Nicolson

Here we Taylor expand $u(x + \Delta x, t + \Delta t)$ and $u(x - \Delta x, t + \Delta t)$ around $t' = t + \Delta t/2$ to get

$$\begin{aligned} u(x + \Delta x, t + \Delta t) = & u(x, t') + \frac{\partial u(x, t')}{\partial x} \Delta x + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial x^2} \Delta x^2 + \frac{\partial^3 u(x, t')}{6\partial x^3} \Delta x^3 \\ & + \frac{\partial^2 u(x, t')}{2\partial t^2} \frac{\Delta t^2}{4} + \frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^4) \end{aligned} \quad (70a)$$

$$\begin{aligned} u(x - \Delta x, t + \Delta t) = & u(x, t') - \frac{\partial u(x, t')}{\partial x} \Delta x + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial x^2} \Delta x^2 - \frac{\partial^3 u(x, t')}{6\partial x^3} \Delta x^3 \\ & + \frac{\partial^2 u(x, t')}{2\partial t^2} \frac{\Delta t^2}{4} - \frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^4) \end{aligned} \quad (70b)$$

$$\begin{aligned} u(x + \Delta x, t) = & u(x, t') + \frac{\partial u(x, t')}{\partial x} \Delta x - \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial x^2} \Delta x^2 + \frac{\partial^3 u(x, t')}{6\partial x^3} \Delta x^3 \\ & + \frac{\partial^2 u(x, t')}{2\partial t^2} \frac{\Delta t^2}{4} - \frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^4) \end{aligned} \quad (70c)$$

$$\begin{aligned} u(x - \Delta x, t) = & u(x, t') - \frac{\partial u(x, t')}{\partial x} \Delta x - \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial x^2} \Delta x^2 - \frac{\partial^3 u(x, t')}{6\partial x^3} \Delta x^3 \\ & + \frac{\partial^2 u(x, t')}{2\partial t^2} \frac{\Delta t^2}{4} + \frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^4) \end{aligned} \quad (70d)$$

$$u(x, t + \Delta t) = u(x, t') + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial t^2} \Delta t^2 + \mathcal{O}(\Delta t^3) \quad (70e)$$

$$u(x, t) = u(x, t') - \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial^2 u(x, t')}{2\partial t^2} \Delta t^2 + \mathcal{O}(\Delta t^3) \quad (70f)$$

The above formulae are taken from Hjørth-Jensen's slides [2].

Combining (70e) and (70f) gives the time derivative

$$\begin{aligned} \left(u(x, t + \Delta t) - u(x, t) \right) = & \left(u(x, t') - u(x, t') \right) + \left(\frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} - \left(-\frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} \right) \right) \\ & + \left(\frac{\partial^2 u(x, t')}{2\partial t^2} \Delta t^2 - \frac{\partial^2 u(x, t')}{2\partial t^2} \Delta t^2 \right) + \left(\mathcal{O}(\Delta t^3) - \mathcal{O}(\Delta t^3) \right) \end{aligned} \quad (71a)$$

$$u(x, t + \Delta t) - u(x, t) = \frac{\partial u(x, t')}{\partial t} \Delta t + \mathcal{O}(\Delta t^3) \quad (71b)$$

$$\frac{\partial u(x, t')}{\partial t} = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + \mathcal{O}(\Delta t^2) \quad (71c)$$

Now for the spacial derivative. First we solve (70a) and (70b) for $u_{xx}(x, t')$

$$\begin{aligned}
u(x + \Delta x, t + \Delta t) + u(x - \Delta x, t + \Delta t) &= \left(u(x, t') + u(x, t') \right) + \left(\frac{\partial u(x, t')}{\partial x} \Delta x - \frac{\partial u(x, t')}{\partial x} \Delta x \right) \\
&+ \left(\frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} \right) + \left(\frac{\partial^2 u(x, t')}{2 \partial x^2} \Delta x^2 + \frac{\partial^2 u(x, t')}{2 \partial x^2} \Delta x^2 \right) \\
&+ \left(\frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{2} \right) + \left(\frac{\partial^3 u(x, t')}{6 \partial x^3} \Delta x^3 - \frac{\partial^3 u(x, t')}{6 \partial x^3} \Delta x^3 \right) \\
&+ \left(\frac{\partial^2 u(x, t')}{2 \partial t^2} \frac{\Delta t^2}{4} + \frac{\partial^2 u(x, t')}{2 \partial t^2} \frac{\Delta t^2}{4} \right) + \left(\frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x - \frac{\partial^2 u(x, t')}{\partial x \partial t} \frac{\Delta t}{2} \Delta x \right) \\
&+ \left(\mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta t^3) \right) + \mathcal{O}(\Delta x^4)
\end{aligned} \tag{72a}$$

$$= 2u(x, t') + \frac{\partial u(x, t')}{\partial t} \Delta t + \frac{\partial^2 u(x, t')}{\partial x^2} \Delta x^2 + \frac{\partial^2 u(x, t')}{2 \partial t^2} \frac{\Delta t^2}{2} + \mathcal{O}(\Delta t^3) + \mathcal{O}(\Delta x^4) \tag{72b}$$

$$= 2u(x, t') + \frac{\partial u(x, t')}{\partial t} \Delta t + \frac{\partial^2 u(x, t')}{\partial x^2} \Delta x^2 + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^4) \tag{72c}$$

$$\begin{aligned}
\frac{\partial^2 u(x, t')}{\partial x^2} \Delta x^2 &= u(x + \Delta x, t + \Delta t) + u(x - \Delta x, t + \Delta t) - 2u(x, t') \\
&+ \frac{\partial u(x, t')}{\partial t} \Delta t + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^4)
\end{aligned} \tag{72d}$$

$$\begin{aligned}
\frac{\partial^2 u(x, t')}{\partial x^2} &= \frac{u(x - \Delta x, t + \Delta t) - 2u(x, t') + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \\
&+ \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{\Delta x^2} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)
\end{aligned} \tag{72e}$$

Doing the same as in (72e) for (70c) and (70d) we obtain

$$\begin{aligned}
\frac{\partial^2 u(x, t')}{\partial x^2} &= \frac{u(x - \Delta x, t) - 2u(x, t') + u(x + \Delta x, t)}{\Delta x^2} \\
&- \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{\Delta x^2} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)
\end{aligned} \tag{73a}$$

Now we take the mean of (72e) and (73)

$$\begin{aligned}
u_{xx}(x, t') &= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t') + u(x + \Delta x, t + \Delta t)}{\Delta x^2} + \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{\Delta x^2} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \right) \\
&+ \frac{u(x - \Delta x, t) - 2u(x, t') + u(x + \Delta x, t)}{\Delta x^2} - \frac{\partial u(x, t')}{\partial t} \frac{\Delta t}{\Delta x^2} + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \\
&= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t') + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \right. \\
&+ \left. \frac{u(x - \Delta x, t) - 2u(x, t') + u(x + \Delta x, t)}{\Delta x^2} \right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \\
&= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t') + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \right. \\
&+ \left. \frac{u(x - \Delta x, t) - 2u(x, t') + u(x + \Delta x, t)}{\Delta x^2} \right) + \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2) \\
&= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t') + u(x + \Delta x, t + \Delta t)}{\Delta x^2} + \frac{u(x - \Delta x, t) - 2u(x, t') + u(x + \Delta x, t)}{\Delta x^2} \right) \\
&+ \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)
\end{aligned} \tag{74a}$$

Now combining (71c) and (74a) we get the Crank-Nicolson scheme

$$\begin{aligned} \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + \mathcal{O}(\Delta t^2) &= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \right. \\ &\quad \left. + \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \right) + \mathcal{O}(\Delta x^2), \end{aligned} \quad (75a)$$

where we have put both $\mathcal{O}(\Delta t^2)$ into a common term.

(75) shows that the Crank-Nicolson scheme is 2nd order in both time and space, implying better convergence properties for the Crank-Nicolson scheme compared to the Backward Euler scheme and the Forward Euler scheme.

We study the stability of the Crank-Nicolson scheme using the same method as for the previous schemes. Insertion of the ansatz (63) into the Crank-Nicolson scheme (75) and solving for $|a_k|$ gives:

$$\begin{aligned} \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} &= \frac{1}{2} \left(\frac{u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \right. \\ &\quad \left. + \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \right) \\ a_k^n e^{ik\pi j \Delta x} \frac{a_k - 1}{\Delta t} &= \frac{a_k^n e^{ik\pi j \Delta x}}{2} \left(\frac{a_k e^{-ik\pi \Delta x} - 2a_k + a_k e^{ik\pi \Delta x}}{\Delta x^2} + \frac{e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}}{\Delta x^2} \right) \\ \frac{a_k - 1}{\Delta t} &= \frac{1}{2} \left(\frac{a_k e^{-ik\pi \Delta x} - 2a_k + a_k e^{ik\pi \Delta x}}{\Delta x^2} + \frac{e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}}{\Delta x^2} \right) \\ &= \frac{1 + a_k}{2\Delta x^2} (e^{-ik\pi \Delta x} - 2 + e^{ik\pi \Delta x}) \\ &\stackrel{(64)}{=} -4 \frac{1 + a_k}{2\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right) \end{aligned} \quad (76a)$$

$$\begin{aligned} a_k - 1 &= (1 + a_k) \left(-\frac{2\Delta t}{\Delta x^2}\right) \sin^2\left(\frac{k\pi \Delta x}{2}\right) \\ \left(1 + \frac{2\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right)\right) a_k &= 1 - \frac{2\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right) \\ a_k &= \frac{1 - \frac{2\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right)}{1 + \frac{2\Delta t}{\Delta x^2} \sin^2\left(\frac{k\pi \Delta x}{2}\right)} < 1 \end{aligned} \quad (76b)$$

(76b) shows that the Crank-Nicolson scheme is unconditionally stable.

Based on the analyzis of the different schemes, we expect the Crank-Nicolson scheme to be the best scheme with respect to convergence and stability.

2.4.4 θ -rule

All the schemes derived above can be derived from a more general scheme, called the θ -rule scheme. To see this, first notice that the Crank-Nicolson scheme (75) is the average of the Forward Euler scheme (62b) and the Backward Euler scheme (67a). If we think of the θ -rule as a weighted average with weights θ and $1 - \theta$ so that $u_t = \theta u_{xx, BE} + (1 - \theta) u_{xx, FE}$, we see that the Crank-Nicolson scheme is just a special case of the θ -scheme with equal weights, $\theta = 1/2$. Based on this reasoning, we get the θ -rule

$$\begin{aligned} (1 - \theta) \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + \theta \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} &= \theta \frac{u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \\ &\quad + (1 - \theta) \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \end{aligned} \quad (77a)$$

$$\begin{aligned} \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} &= \theta \frac{u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \\ &\quad + (1 - \theta) \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \end{aligned} \quad (77b)$$

From the θ -scheme (77b) we see that $\theta = 0, 1/2, 1$ corresponds to the Forward Euler, the Crank-Nicolson and the Backward Euler scheme respectively.

When implementing the 1D-schemes, we will use the θ -scheme. Using the θ -scheme, we need only write one scheme instead of three.

2.4.5 Implementation of 1D problem

As mentioned in the previous paragraph, we implement the 1D schemes using the θ -scheme. We will now rewrite (77b) to a linear system, and then we will apply the Thomas algorithm to this system.

To ease the notation, we introduce

$$\alpha = \frac{\Delta t}{\Delta x^2}. \quad (78)$$

Insertion of α (78) into the θ -scheme (77b) and introducing the discretization $u(x + \Delta x, t - \Delta t) = u_{i+1}^{n-1}$ gives

$$\begin{aligned} \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} &= \theta \frac{u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t)}{\Delta x^2} \\ &+ (1 - \theta) \frac{u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)}{\Delta x^2} \end{aligned} \quad (79a)$$

$$\begin{aligned} u(x, t + \Delta t) - u(x, t) &= \alpha \theta \left(u(x - \Delta x, t + \Delta t) - 2u(x, t + \Delta t) + u(x + \Delta x, t + \Delta t) \right) \\ &+ \alpha(1 - \theta) \left(u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t) \right) \end{aligned} \quad (79b)$$

$$u_i^{n+1} - u_i^n = \alpha \theta \left(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1} \right) + \alpha(1 - \theta) \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n \right) \quad (79c)$$

$$u_i^n - u_i^{n-1} = \alpha \theta \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n \right) + \alpha(1 - \theta) \left(u_{i-1}^{n-1} - 2u_i^{n-1} + u_{i+1}^{n-1} \right) \quad (79d)$$

$$u_i^n - \alpha \theta \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n \right) = u_i^{n-1} + \alpha(1 - \theta) \left(u_{i-1}^{n-1} - 2u_i^{n-1} + u_{i+1}^{n-1} \right) \quad (79e)$$

$$- \alpha \theta u_{i-1}^n + (2\alpha\theta + 1)u_i^n - \alpha \theta u_{i+1}^n = \alpha(1 - \theta)u_{i-1}^{n-1} + \left(1 - 2\alpha(1 - \theta) \right) u_i^{n-1} + \alpha(1 - \theta)u_{i+1}^{n-1} \quad (79f)$$

$$- 2\alpha\theta u_{i-1}^n + 2(2\alpha\theta + 1)u_i^n - 2\alpha\theta u_{i+1}^n = 2\alpha(1 - \theta)u_{i-1}^{n-1} + 2 \left(1 - 2\alpha(1 - \theta) \right) u_i^{n-1} + 2\alpha(1 - \theta)u_{i+1}^{n-1} \quad (79g)$$

Now we rewrite (79g) as a matrix-vector equation $A_1 U^n = A_2 U^{n-1}$:

$$\begin{aligned} &\underbrace{\begin{bmatrix} 2(2\alpha\theta + 1) & -2\alpha\theta & \cdots & 0 \\ -2\alpha\theta & 2(2\alpha\theta + 1) & -2\alpha\theta & \vdots \\ \vdots & & \ddots & -2\alpha\theta \\ 0 & \cdots & -2\alpha\theta & 2(2\alpha\theta + 1) \end{bmatrix}}_{\mathbf{A}_1} \underbrace{\begin{bmatrix} u_1^n \\ u_2^n \\ \vdots \\ u_N^n \end{bmatrix}}_{\mathbf{U}^n} \\ &= \underbrace{\begin{bmatrix} 2(1 - 2\alpha(1 - \theta)) & 2\alpha(1 - \theta) & \cdots & 0 \\ 2\alpha(1 - \theta) & 2(1 - 2\alpha(1 - \theta)) & 2\alpha(1 - \theta) & \vdots \\ \vdots & & \ddots & 2\alpha(1 - \theta) \\ 0 & \cdots & 2\alpha(1 - \theta) & 2(1 - 2\alpha(1 - \theta)) \end{bmatrix}}_{\mathbf{A}_2} \underbrace{\begin{bmatrix} u_1^{n-1} \\ u_2^{n-1} \\ \vdots \\ u_N^{n-1} \end{bmatrix}}_{\mathbf{U}^{n-1}} \end{aligned} \quad (80a)$$

In (80a), the right hand side $\mathbf{A}_2 \mathbf{U}^{n-1}$ is known, since \mathbf{U}^{n-1} is the previous periods solution. Hence (80a) is a linear system of the known type $AU = b$, which needs to be solved at each time step.

We note that with $\theta = 0$ in (80a), the system reduces to the explicit Forward Euler system, which can be obtained from (62b). $\theta = 1$ gives the Backward Euler system (69f). $\theta = 1/2$ in (80a) gives the system that can be obtained

by rewrtng the Crank-Nicolson scheme (75) as a linear system.

In principle the above system can be solved by calculating the inverse of A_1 and solving $U^n = A_1^{-1}(A_2 U^{n-1})$. This is a costly operation and is avoided. Instead of calculating the inverse, one often solves these systems by a elimination method, e.g. Gaussian elimination or by LU. In this case, we note that we are dealing with a tridiagonal matrix. For tridiagonal systems, the Thomas algorithm gives an alterative way of solving the linear system which reduces the number of FLOPS considerably. In our case we are even more lucky. Our matrix is symmetric, and the elements are constant, so the standard Thomas algorithm can be further improved. We will now derive the algorithm that we implement.

We start with a tridiagonal symmetric linear system $Au = f$, with A being 4×4 . The following show the forward substitution steps for this sytem

$$\begin{bmatrix} d_1 & e_1 & 0 & 0 & f_1 \\ e_2 & d_2 & e_2 & 0 & f_2 \\ 0 & e_3 & d_3 & e_3 & f_3 \\ 0 & 0 & e_4 & d_4 & f_4 \end{bmatrix} \rightarrow \begin{bmatrix} d_1 & e_1 & 0 & 0 & f_1 \\ 0 & \tilde{d}_2 & e_2 & 0 & \tilde{f}_2 \\ 0 & e_3 & d_3 & e_3 & f_3 \\ 0 & 0 & e_4 & d_4 & f_4 \end{bmatrix} \rightarrow \begin{bmatrix} d_1 & e_1 & 0 & 0 & f_1 \\ 0 & \tilde{d}_2 & e_2 & 0 & \tilde{f}_2 \\ 0 & 0 & \tilde{d}_3 & e_3 & \tilde{f}_3 \\ 0 & 0 & e_4 & d_4 & f_4 \end{bmatrix} \rightarrow \begin{bmatrix} d_1 & e_1 & 0 & 0 & f_1 \\ 0 & \tilde{d}_2 & e_2 & 0 & \tilde{f}_2 \\ 0 & 0 & \tilde{d}_3 & e_3 & \tilde{f}_3 \\ 0 & 0 & e_4 & \tilde{d}_4 & \tilde{f}_4 \end{bmatrix} \quad (81a)$$

From (80a) we have that in our case $e_1 = e_2 = \dots = e_N = -2\alpha$ and $d_1 = d_2 = \dots = d_N = 2(2\alpha\theta + 1)$. Calculating the first couple of \tilde{d} 's we quickly see that we get the following general expression for \tilde{d}_i

$$\tilde{d}_i = 2(2\alpha\theta + 1) + \frac{(2\alpha)^2}{\tilde{d}_{i-1}} \text{ for } i > 1. \quad (82)$$

Doing the same for \tilde{f} we get the general expression

$$\tilde{f}_i = f_i + \frac{2\alpha\theta}{\tilde{d}_{i-1}} \tilde{f}_{i-1} \text{ for } i > 1. \quad (83)$$

Having \tilde{d} and \tilde{f} we are ready to do the back substitution step. We have

$$\begin{bmatrix} d_1 & e_1 & 0 & 0 \\ 0 & \tilde{d}_2 & e_2 & 0 \\ 0 & 0 & \tilde{d}_3 & e_3 \\ 0 & 0 & e_4 & \tilde{d}_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ \tilde{f}_2 \\ \tilde{f}_3 \\ \tilde{f}_4 \end{bmatrix} \quad (84a)$$

From (84) we get

$$u_4 = \frac{\tilde{f}_4}{\tilde{d}_4} \quad (85)$$

and

$$u_i = \frac{\tilde{f}_i + 2\alpha\theta u_{i+1}}{\tilde{d}_i} \text{ for } i > 1. \quad (86)$$

The equations (82), (83), (85) and (86) is what we need for our algorithm.

```
for time in times:
    Calculate RHS f (= A_2 U^{n-1})

    // Forward substitution
    for row in rows (Start from 1st row):
        Calculate \tilde{d}
        Calculate \tilde{f}
    end row loop
```

```

// Backward substitution
For row in rows (Starting from last row)
    Calculate u
end row loop

// Update RHS for next time step
U^{n-1} = U^n

end time loop

```

2.4.6 2D explicit scheme

We will derive and apply a Forward Euler (explicit) scheme and a Backward Euler (implicit) scheme for the 2D case.

Compared to the 1D case, we get an extra term for u_{yy} in (62b) for the explicit scheme. u_{yy} is approximated in exactly the same way as u_{xx} (61c), so we get

$$u_{yy}(x, y, t) = \frac{u(x, y - \Delta y, t) - 2u(x, y, t) + u(x, y + \Delta y, t)}{\Delta y^2} + \mathcal{O}(\Delta y^2) \quad (87)$$

Adding (87) to the 1D FE scheme (62b) gives the explicit 2D-scheme

$$\begin{aligned} \frac{u(x, y, t + \Delta t) - u(x, y, t)}{\Delta t} + \mathcal{O}(\Delta t) &= \frac{u(x - \Delta x, y, t) - 2u(x, y, t) + u(x + \Delta x, y, t)}{\Delta x^2} \\ &+ \frac{u(x, y - \Delta y, t) - 2u(x, y, t) + u(x, y + \Delta y, t)}{\Delta y^2} \\ &+ \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2) \end{aligned} \quad (88a)$$

$$\rightarrow u_{ij}^{n+1} \stackrel{(78)}{\approx} u_{ij}^n + \alpha(u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n - 4u_{ij}^n), \quad (88b)$$

where we in the last transition assumed $\Delta x = \Delta y$ such that α is given as before.

First we observe that the truncation errors go like $\mathcal{O}(\Delta t)$, $\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta y^2)$.

We analyze the stability by inserting the ansatz

$$u = a_k^n e^{ik\pi\Delta x j} e^{ik\pi\Delta y l} \quad (89)$$

into (88b) and solve for $|a|$, as before

$$u_{ij}^{n+1} = u_{ij}^n + \alpha(u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n - 4u_{ij}^n) \quad (90a)$$

$$a_k^n e^{i\pi k\Delta x j} e^{i\pi k\Delta y l} a = a_k^n e^{i\pi k\Delta x j} e^{i\pi k\Delta y l} \left(1 + \alpha(e^{-ik\pi\Delta x} + e^{ik\pi\Delta x}) - 2 + e^{-ik\pi\Delta y} + e^{ik\pi\Delta y} - 2\right) \quad (90b)$$

$$a = \left(1 + \alpha(e^{-ik\pi\Delta x} + e^{ik\pi\Delta x}) - 2 + e^{-ik\pi\Delta y} + e^{ik\pi\Delta y} - 2\right) \quad (90c)$$

$$\stackrel{(64)}{=} 1 - 4\alpha \left(\sin^2(k\pi\Delta x/2) + \sin^2(k\pi\Delta y/2)\right) \quad (90d)$$

$$|a| = \left|1 - 4\alpha \left(\sin^2(k\pi\Delta x/2) + \sin^2(k\pi\Delta y/2)\right)\right| \quad (90e)$$

$$|a| < 1 \rightarrow \left|1 - 4\alpha \left(\sin^2(k\pi\Delta x/2) + \sin^2(k\pi\Delta y/2)\right)\right| < 1 \quad (90f)$$

$$\left|1 - 4\alpha \left(\sin^2(k\pi\Delta x/2) + \sin^2(k\pi\Delta y/2)\right)\right| < |1 - 8\alpha| \quad (90g)$$

Again assuming $\Delta x = \Delta y$, we have $\alpha = \Delta t / \Delta x^2 > 0$, so that (90f) and (90g) gives

$$1 - 8\alpha < -1 \quad (91a)$$

$$\alpha < \frac{1}{4}. \quad (91b)$$

(91b) gives that the 2D Forward Euler scheme is conditionally stable. We note that we could probably have applied a simpler ansatz than (89), since it in (88b) was already assumed that $\Delta x = \Delta y$.

The new thing in (88b), compared to the 1D case (62b), is that u_{ij}^{n+1} in the 2D case is a matrix and not a vector. The fact that we are dealing with a matrix instead of a vector, does not change anything with regards to the solution strategy. The equation (88b) is still explicit, the only difference in implementation is that instead of one single loop, a double loop over two space dimensions is needed now.

The algorithm for solving 2D Forward Euler follows.

```

Set IC
for time in times:
  Set BC

  for x in X:
    for y in Y:
      Calculate  $u_{ij}^{n+1} = f(u^n)$ 
    end y-loop
  end x-loop

  Set  $u^n = u^{n+1}$ 
end time-loop

```

2.4.7 2D implicit scheme

We start in the same way as we did for Forward Euler in 2D, and just add an extra term for u_{yy} to the 1D Backward Euler scheme (67a). In addition we will assume $h = \Delta x = \Delta y$.

$$\begin{aligned} \frac{u(x, y, t) - u(x, y, t - \Delta t)}{\Delta t} + \mathcal{O}(\Delta t) &= \frac{u(x - \Delta x, y, t) - 2u(x, y, t) + u(x + \Delta x, y, t)}{\Delta x^2} \\ &+ \frac{u(x, y - \Delta y, t) - 2u(x, y, t) + u(x, y + \Delta y, t)}{\Delta y^2} \end{aligned} \quad (92a)$$

$$\begin{aligned} &+ \mathcal{O}(\Delta x^2) + \mathcal{O}(\Delta y^2) \\ \frac{u_{i,j}^n - u_{i,j}^{n-1}}{\Delta t} &\approx \frac{u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n}{\Delta x^2} \\ &+ \frac{u_{i,j-1}^n - 2u_{i,j}^n + u_{i,j+1}^n}{\Delta y^2} \end{aligned} \quad (92b)$$

$$\begin{aligned} \frac{u_{i,j}^n - u_{i,j}^{n-1}}{\Delta t} &\stackrel{h=\Delta x=\Delta y}{\approx} \frac{u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n}{h^2} \\ &+ \frac{u_{i,j-1}^n - 2u_{i,j}^n + u_{i,j+1}^n}{h^2} \end{aligned} \quad (92c)$$

$$u_{i,j}^n - u_{i,j}^{n-1} \stackrel{(78)}{\approx} \alpha (u_{i-1,j}^n - 4u_{i,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n) \quad (92d)$$

$$u_{i,j}^n = u_{i,j}^{n-1} + \frac{1}{1 + 4\alpha} (u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n) \quad (92e)$$

First we observe that the truncation errors go like $\mathcal{O}(\Delta t)$, $\mathcal{O}(\Delta x^2)$ and $\mathcal{O}(\Delta y^2)$, so there is no difference in the truncation errors between Forward Euler and Backward Euler, just as for the 1D case.

We study stability of the 2D Backward Euler scheme by inserting the 2D ansatz (89) into (92d)

$$u_{i,j}^n - u_{i,j}^{n-1} = \alpha(u_{i-1,j}^n - 4u_{i,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n) \quad (93a)$$

$$a_k - 1 = \alpha a (e^{-ik\pi\Delta x} + e^{ik\pi\Delta x} - 2 + e^{-ik\pi\Delta y} + e^{ik\pi\Delta y} - 2) \quad (93b)$$

$$\stackrel{(64)}{=} -4\alpha a \left(\sin^2(k\pi\Delta x/2) + \sin^2(k\pi\Delta y/2) \right) \quad (93c)$$

$$\stackrel{\Delta x=\Delta y=h}{=} -8\alpha a \sin^2(kh/2) \quad (93d)$$

$$a = \frac{1}{1 + 8\alpha \sin^2(kh/2)} < 1 \quad (93e)$$

From (93e) we see that the Backward Euler scheme is unconditionally stable in two dimensions, as it was for one dimension. We note that we probably could have applied a simpler ansatz than (89), since it in (92d) was already assumed that $\Delta x = \Delta y$.

We see that (92e) is an implicit scheme, since the only known is $u_{i,j}^n$. (92e) can be transformed into a linear system of the known type $Au = b$. The procedure for transforming (92e) into a linear system is the same as demonstrated for the 1D case, see Hjorth-Jensen [1] p. 317. However, there is one important difference compared to the 1D linear system: The matrix is not tridiagonal anymore! This means that we cannot apply the Thomas algorithm anymore, and standard methods as LU-decomposition becomes unpractical. However, in our case the matrix is positive definite, implying that iterative schemes converges to the true solution, Hjorth-Jensen [1] p. 322. Iterative methods are often much faster than direct methods like Gaussian elimination and LU-decomposition. Hence we will solve the implicit problem applying iterative methods.

2.4.8 Iterative methods

This section follows Olver and Shakiban's [3] chapter on iterative methods closesly. The goal is to present the main idea behind iterative methods, and to present the Jacobi method as part of a general framework.

Firstly, an iterative method is not a direct method, like e.g. Gaussian elimination, that solves the problem as it is stated, typically

$$\mathbf{A}\mathbf{u} = \mathbf{b}. \quad (94)$$

Instead, with iterative methods, one attempts solving (94) by solving another, affine iterative, system:

$$\mathbf{u}^{(k+1)} = \mathbf{T}\mathbf{u}^{(k)} + \mathbf{c}, \quad (95)$$

where k stands for iterations, T is a coefficient matrix of same size as A , and \mathbf{c} is a vector. We observe that the right hand side is an affine function of $\mathbf{u}^{(k)}$. For comparison, a linear iterative system is on the form $\mathbf{u}^{(k+1)} = \mathbf{T}\mathbf{u}^{(k)}$, so it is the addition of the vector \mathbf{c} that makes our iterative an affine one, and not a linear one. The affine representation is more genereal compared to the linear one, and allows for more reflexibility in developing iterative schemes.

So how is using (95) in place of solving the original system better? It might not be better. But potentially it can be a lot faster. Look at the FLOP count. In (95) the matrix-vector multiplication is $\mathcal{O}(N)$ flops, and the addition is of the same order, giving $\mathcal{O}(N)$ FLOPS per iteration. The total FLOP-count for the iterative method is $\mathcal{O}kN$ FLOPS. In comparison the Gaussian elimination method is always $\mathcal{O}(N^3)$ FLOPS. This shows that if we can construct a T and \mathbf{c} in our affine iterative scheme (95) that converges to the true solution after few iterations, the iterative scheme will be much less demanding with respect to FLOPS compared to direct methods like Gaussian elimination.

We have shown that the potential of the iterative methods with regards to FLOPS is bug. Now we need to analyse condition for when the method actually works. For the iterative method (95) to work, it firstly has to converge:

$$\lim_{k \rightarrow \infty} \mathbf{u}^k = \mathbf{u}^*. \quad (96)$$

Taking the limit on both sides of (95) and using (96) we get

$$\lim_{k \rightarrow \infty} \mathbf{u}^{k+1} = \lim_{k \rightarrow \infty} \mathbf{T}\mathbf{u}^k + \mathbf{c} \quad (97a)$$

$$\mathbf{u}^* = \mathbf{T}\mathbf{u}^* + \mathbf{c}. \quad (97b)$$

(97b) is a fixed point equation, and we have that the solution to the iterative scheme has to converge to a fixed point, \mathbf{u}^* .

Secondly, the convergent solution \mathbf{u}^* that is approached as $k \rightarrow \infty$ must become equal to the solution \mathbf{u} of the original problem, $\mathbf{A}\mathbf{u} = \mathbf{b}$. If the affine iterative system (95) converges to a fixed point that differs from the true solution, $\mathbf{u}^* \neq \mathbf{u}$, the fixed point solution is of little use for us.

Lets now look at conditions for the 1st condition, convergence to the fixed point \mathbf{u}^* , to hold. We study the error with respect to the fixed point

$$\mathbf{e}^{k+1} = \mathbf{u}^{k+1} - \mathbf{u}^* \quad (98a)$$

$$\stackrel{(95)}{=} \mathbf{T}\mathbf{u}^k + \mathbf{c} - \mathbf{u}^* \quad (98b)$$

$$= \mathbf{T}\mathbf{u}^k + \mathbf{c} - (\mathbf{T}\mathbf{u}^* + \mathbf{c}) \quad (98c)$$

$$= \mathbf{T}(\mathbf{u}^k - \mathbf{u}^*) \quad (98d)$$

$$= \mathbf{T}\mathbf{e}^k \quad (98e)$$

$$= \mathbf{T}^k \mathbf{e}^{(0)}. \quad (98f)$$

(98f) shows that the development of the error in convergence towards the fixed point depends on the same matrix as the original iterative problem, \mathbf{T} . We have convergence towards the fixed point \mathbf{u}^* if \mathbf{T} is a convergent matrix. \mathbf{T} is a convergent matrix if the spectral radius of \mathbf{T} is less than one, $\rho(\mathbf{T}) < 1$. The speed of convergence is inversely related to the spectral radius, implying that we want to construct T with as low spectral radius as possible.

We now have a condition for the first requirement for our iterative scheme: convergence to the fixed point. Now for the 2nd condition: convergence to the true solution. It turns out that convergence to the true solution depends on the chosen scheme, represented by the matrix T in our case, and the coefficient matrix of the original problem, A . We have that both the Jacobi scheme, to be derived, and the Gauss-Seidel scheme are convergent to the true solution if A is strictly diagonally dominant.

We sum up our results in this section:

The iteration scheme (95) converges to the solution of the original linear system (94) if the following two conditions are satisfied

- The spectral radius of T in (95) is less than one.
- A in (94) is strictly diagonally dominant (In the case of Jacobi and Gauss-Seidel)

Next we will derive the Jacobi-algorithm and relate it to our general iterative system (95).

2.4.9 Jacobi's iteration algorithm

We begin by a rewrite of the linear system (94), where we split A into one strictly lower triangular matrix, L , one strictly upper triangular matrix, U , and one diagonal matrix, D :

$$\mathbf{A}\mathbf{u} = \mathbf{b} \quad (99a)$$

$$(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{u} = \mathbf{b} \quad (99b)$$

$$\mathbf{u} = \mathbf{D}^{-1} \left(-(\mathbf{L} + \mathbf{D})\mathbf{u} + \mathbf{b} \right) \quad (99c)$$

The above is nothing new. It is still the original system. With the Jacobi algorithm, one sets $\mathbf{u}^{k+1} = \mathbf{u}$ on the left hand side of (99c) and $\mathbf{u} = \mathbf{u}^k$ on the right hand side to get

$$\mathbf{u}^{k+1} = D^{-1} \left(-(L + D)\mathbf{u}^k + \mathbf{b} \right) \quad (100a)$$

We see that the Jacobi-scheme (100a) fits our general iterative fixed point scheme (95), with

$$T = -D^{-1}(L + D) \quad (101a)$$

$$\mathbf{c} = D^{-1}\mathbf{b}. \quad (101b)$$

It is perhaps easier to see the workings of the Jacbi algorithm (100a) by considering a single row in the matrix-vector system (100a)

$$u_i^{(k+1)} = -\frac{1}{a_{ii}} \sum_{j=1, i \neq j} a_{ij} u_j^{(k)} + \frac{b_i}{a_{ii}}. \quad (102a)$$

From (102) we see that only non-diagonal terms are used in the iteration on the right hand side, and that all terms are divided by the diagonal terms from the original matrix A . (102) is very close to the form of the implemented algorithm.

Using (102) we will in the next section see that it is easy to formulate a Jacobi-solver for the 2D implicit scheme (92e).

Now we will apply Jacobi's algorithm (102) to the 2D implicit scheme (92e). We see that (92e) is already almost set up like a Jacobi-scheme. We get

$$u_{i,j}^{n,(k+1)} = u_{i,j}^{n-1} + \frac{1}{1 + 4\alpha} (u_{i-1,j}^{n,(k)} + u_{i+1,j}^{n,(k)} + u_{i,j-1}^{n,(k)} + u_{i,j+1}^{n,(k)}) \quad (103a)$$

(103) is our Jacobi-Solver. By setting

$$a_{ii} = -(1 + 4\alpha) \quad (104a)$$

$$b_i = a_{ii} u_{i,j}^{n-1} \quad (104b)$$

in (103) we see that (103) corresponds to the Jacobi-algorithm (102). We also note that only 4 neighboring pints are needed.

Following Hjorth-Jensen [1] p. 319, we get the following main algorithm for the Jacobi-solver

```

for time in times:
  while ((iterations < maxIterations) && (error > allowedError))
    u_{Previous iteration} = u_{new}
    for x in xPoints:
      for y in yPoints:
        Calculate u_{new} = f(u_Old) (Jacobi expression)
        Cacalucate error: error += u_{New}(x,y) - u_{previous iteration}(x,y)
      end y-loop
    end x-loop
    Cacalculate average error: error /= (Nx*Ny);
  end while-loop
  Update u with iteration solution: u_{Previous time step} = u_{New}
end time-loop

```

2.4.10 Gauss-Seidel's iteration scheme

Looking at the row version of the Jacobi scheme, (102), we see that all the u -terms on the right hand side applies values from the previous iteration. However, except for $i = 1$, we will have calculated new u 's for all $j < i$. Assuming the method is converging, we could improve our iteration by including the new values on the right hand side of (102) for $j < i$. This is the Gauss-Seidel scheme, which we can write in row version as

$$u_i^{(k+1)} = -\frac{1}{a_{ii}} \left(\sum_{j=1, j<i} a_{ij} u_j^{(k+1)} + \sum_{j=1, j>i} a_{ij} u_j^{(k)} \right) + \frac{b_i}{a_{ii}}. \quad (105a)$$

(105) is straight forward to implement, and gives an algorithm very similar to the algorithm for Jacobi's iteration scheme

```
for time in times:
    while ((iterations < maxIterations) && (error > allowedError))
        u_{Previous iteration} = u_{new}
        for x in xPoints:
            for y in yPoints:
                Calculate u_{new} = f(u_{new}) (Gauss-Seidel expression)
                Cacaluate error: error += u_{new}(x,y) - u_{previous iteration}(x,y)
            end y-loop
        end x-loop
        Cacalculate average error: error /= (Nx*Ny);
    end while-loop
    Update u with iteration solution: u_{Previous time step} = u_{New}
end time-loop
```

Note that we in the above algorithm have wirtten " u_{new} " in the Gauss-Seidel expression, while in (105) there is both u_{new} iteration and u_{old} iteration. The reason we write only u_{new} , is that in the numerical implementation, we work directly on the u_{new} -array, so it will contain both old and new iteration values as it is updated as we go thorough the loop. Hence in Gauss-Seidel $u_{previous}$ only used for calculating the error and is only updated once per iteration in the while loop.

3 Results

The results following results are displayed in this section: 1D, 2D explicit, 2D implicit, Parallelization, Iterations.

3.1 1D

The two figures below show the 1D solutions for the three schemes for the coarse mesh case, $\Delta x = 0.1$, for two different discretization, one discretization satisfying the stability requirement for Forward Euler and one discretization not satisfying the stability requirement.

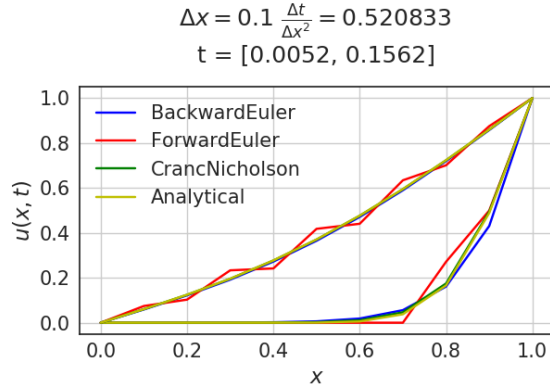


Figure 2: 1D. Coarse mesh. Stability requirement Forward Euler not satisfied.

Forward Euler is unstable, while the other schemes are stable. Backward Euler and Crank-Nicolson produce results close to the analytical solution for all time steps. Steady state is approached for Backward Euler and Crank-Nicolson.

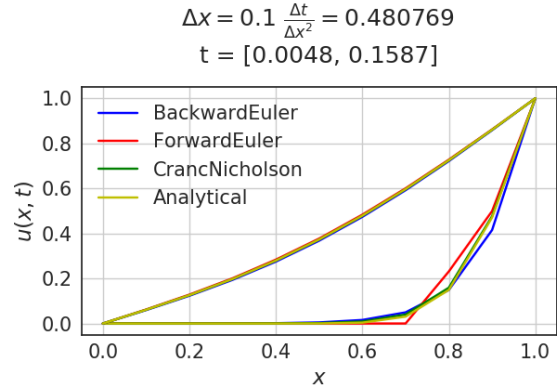


Figure 3: 1D. Coarse mesh. Stability requirement Forward satisfied.

Forward Euler is now stable. All schemes produce results looking qualitatively like the analytical solution. Forward Euler has the largest error..

From the figures above, we see that the Forward Euler scheme is very sensitive to the stability requirement, implying that a very fine time step is needed for acceptable results for Forward Euler. We also see that the Backward Euler scheme and the Crank-Nicolson scheme are stable regardless of the stability requirement for Forward Euler being satisfied or not. From this point of view, the Backward and Crank-Nicolson schemes are preferred, since they need fewer time steps. All these observations fit well with our numerical findings, that except for Forward Euler, the other schemes are unconditionally stable.

The next two figures show the same two types of discretizations with respect to the stability requirement as the figures above, but now the mesh is finer, $\Delta x = 0.01$.

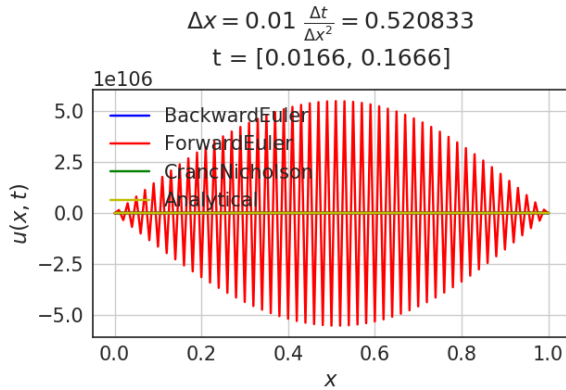


Figure 4: 1D. Fine mesh. Stability requirement Forward Euler not satisfied.

As for the coarse mesh, Forward Euler is unstable.

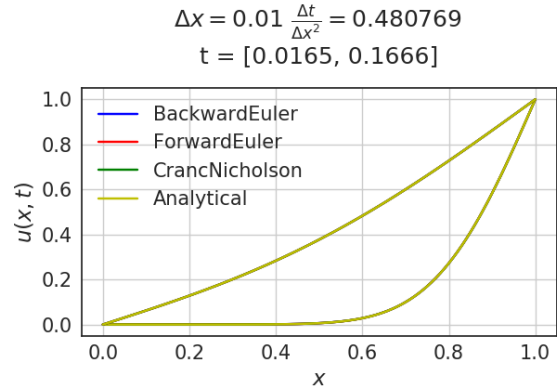


Figure 5: 1D. Fine mesh. Stability requirement Forward satisfied.

Forward Euler is stable. All methods give solutions close to the analytical solution.

Again we see the sensitivity of Forward Euler to the stability requirement. A small breach of the stability requirement gives large oscillations, making the solution non-physical. So even if the mesh is much finer compared to the previous figures, this is not enough to avoid unphysical oscillations in the Forward Euler scheme when the stability requirement for Forward Euler is violated. On the other hand, the figure to the right shows that Forward Euler,

and the other methods, gives very good fit to the analytical solution when the stability requirement is met.

The next figure shows the same as the figure to the right above, with the difference that the results for Forward Euler is removed. Forward Euler is removed so that we can see the solutions of the other two schemes more clearly.

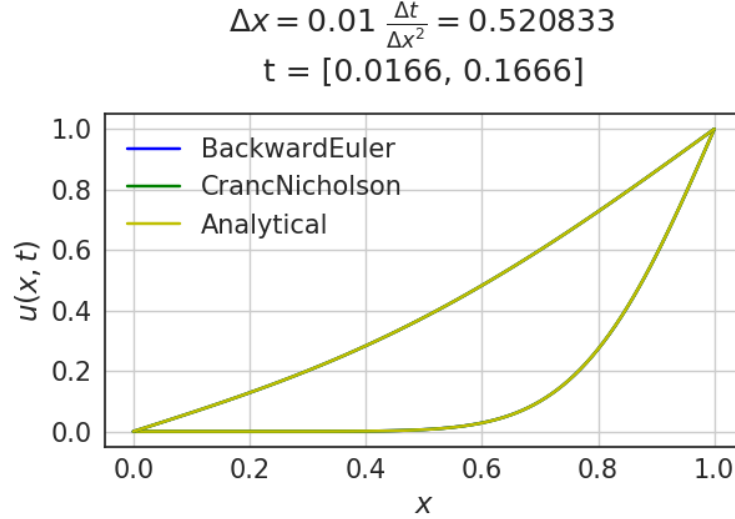


Figure 6: Same as Figure 4 without Forward Euler.

Backward Euler and Crank-Nicolson produces very good results below also when the stability limit for Forward Euler is not satisfied.

From the figure above we see that both Backward Euler and Crank-Nicolson produces very good results in the case where the stability requirement of Forward Euler is violated.

In order to compare Backward Euler to Crank-Nicolson, we study the coarse case in more detail making a zoom of Figure 3:

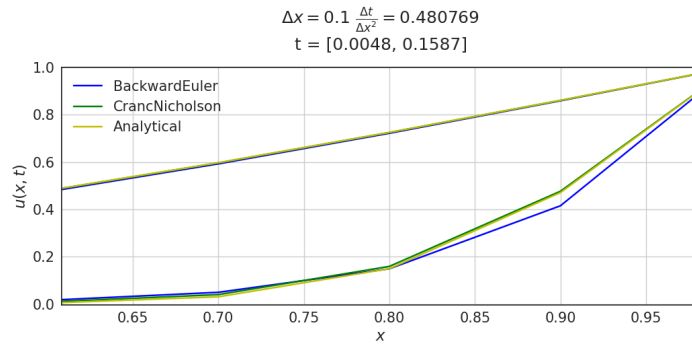


Figure 7: Same as Figure 3 without Forward Euler. Zoomed.

Crank-Nicolson is almost identical with analytical solution and outperforms Backward Euler.

From the above figure we see that, compared to the Backward Euler scheme, the Crank-Nicolson scheme is closer to the analytic solution. This is not unexpected, given our derivations in the theory section, where we showed that the Crank-Nicolson scheme is one order higher in time than the other schemes when it comes to truncation error.

3.2 2D

3.2.1 Explicit scheme

The next four figures show the 2D results for the explicit scheme, at two different points in time, when the stability requirement is met.

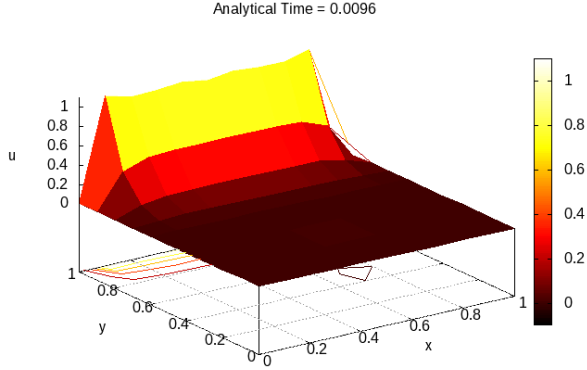


Figure 8: 2D. $\Delta x = 0.1$. $\Delta t = 0.96 \cdot$ stability limit. Early time. Analytical Solution. *Analytical solution seems correct.*

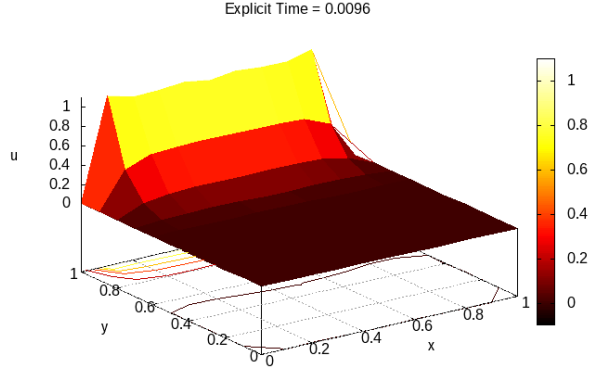


Figure 9: 2D. $\Delta x = 0.1$. $\Delta t = 0.96 \cdot$ stability limit. Early time. Explicit scheme. *Explicit scheme produces results very similar to analytical solution when the stability requirement is satisfied.*

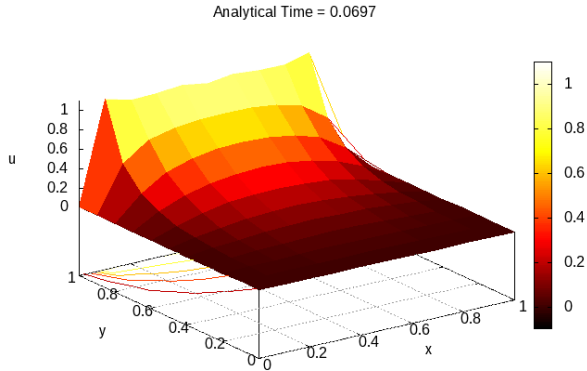


Figure 10: 2D. $\Delta x = 0.1$. $\Delta t = 0.96 \cdot$ stability limit. Later time. Analytical Solution. *Analytical solution seems correct.*

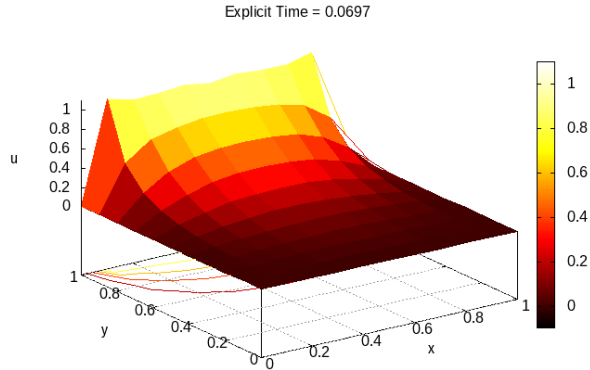


Figure 11: 2D. $\Delta x = 0.1$. $\Delta t = 0.96 \cdot$ stability limit. Later time. Explicit scheme. *Explicit scheme produces results very similar to analytical solution when stability requirement is satisfied.*

From the figures above, we see that when the stability requirement of the explicit scheme is met, the analytical solution is well approximated by the explicit scheme both in the early time of simulations and at a time closer to the steady state.

The next two figures show the analytical and numerical solution from the explicit scheme when the stability requirement of the explicit scheme is violated.

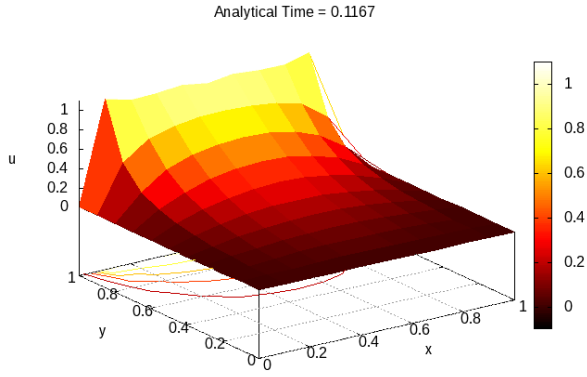


Figure 12: 2D. $\Delta x = 0.1$. $\Delta t = 1.1 \cdot$ stability limit. Analytical Solution.

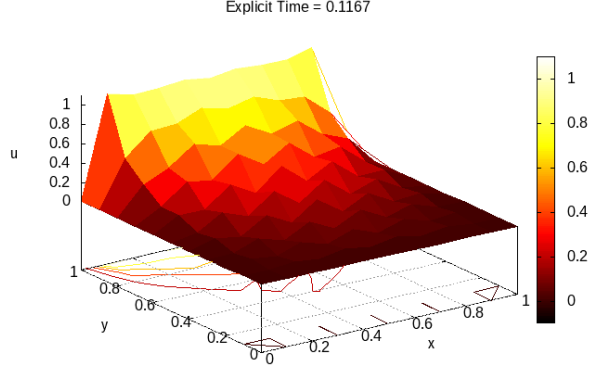


Figure 13: 2D. $\Delta x = 0.1$. $\Delta t = 1.1 \cdot$ stability limit. Explicit scheme.
The explicit scheme is unstable when the stability requirement is not met.

From the above two figures we see once again the sensitivity of the explicit scheme to the stability requirement we derived. We see that a small violation of the stability requirement gives unphysical oscillations in the explicit solution, just as in the 1D case.

3.2.2 Implicit scheme

The four figures below show the 2D results for the implicit scheme at two points in time, one time being close to the starting time and one time closer to the steady state. The discretization violates the stability requirement for the explicit scheme.

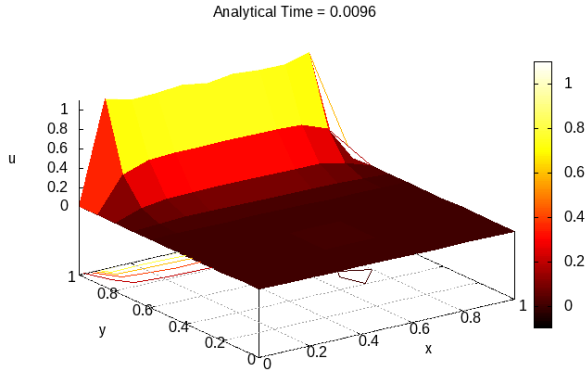


Figure 14: 2D. $\Delta x = 0.1$. $\Delta t = 1.1 \cdot$ stability limit. Early time. Analytical Solution.

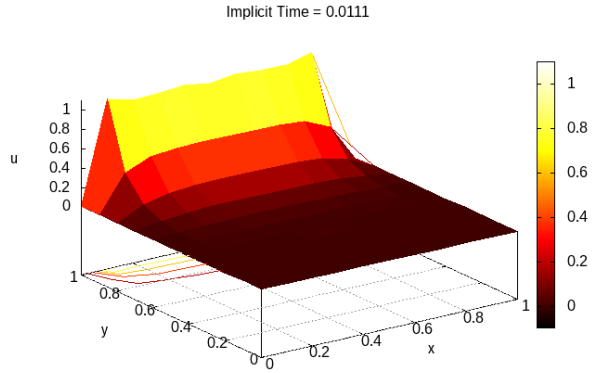


Figure 15: 2D. $\Delta x = 0.1$. $\Delta t = 0.96 \cdot$ stability limit. Early time. Implicit scheme.
Implicit scheme produces results very similar to analytical solution, when the stability requirement for Forward Euler is not satisfied.

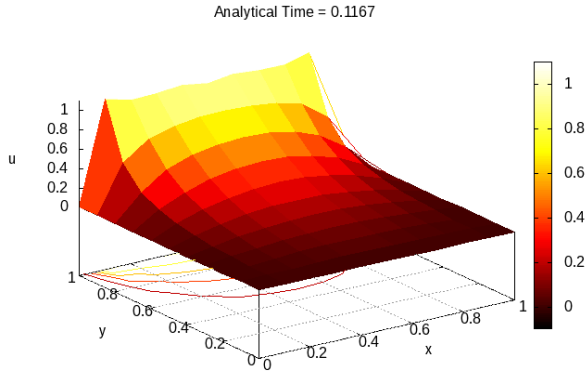


Figure 16: 2D. $\Delta x = 0.1$. $\Delta t = 1.1 \cdot$ stability limit. Analytical Solution.

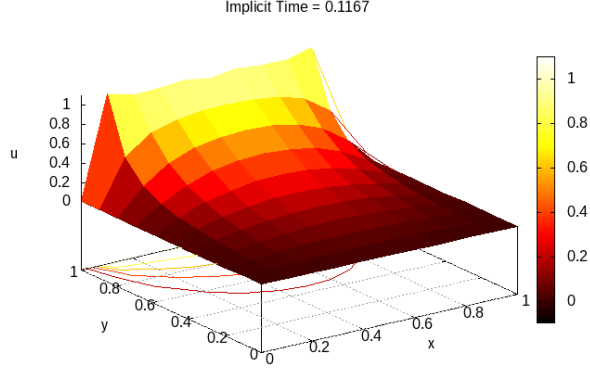


Figure 17: 2D. $\Delta x = 0.1$. $\Delta t = 1.1 \cdot$ stability limit. Explicit scheme.

The implicit scheme is stable when the stability requirement for the explicit scheme is not met.

From the figures above, we see that the implicit scheme gives good approximations of the analytical solution, both for the early time and for the later time. We also note that the results are produced for the same discretization that gave oscillations from the explicit scheme, implying that the implicit scheme is less costly computationally compared to the explicit scheme. With the implicit scheme, for a given space discretization, a coarser time discretization can be chosen without getting bad unphysical results as in the explicit scheme. Once again we observe the good fit with our theoretical derivations that gave that the implicit scheme is unconditionally stable and the explicit scheme being conditionally stable.

3.3 Parallelization

The below two figures show the relation between solution times for the implicit method and the analytical solution. Also the effect of parallelization of the analytical solution is shown.

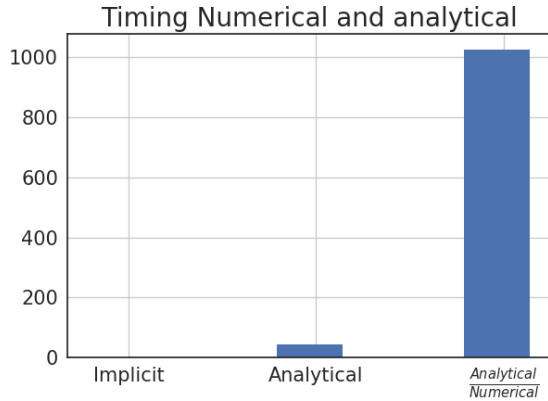


Figure 18: 2D. Timing. Numerical scheme: Implicit Jacobi iteration. Analytical solution using 30 points in Gaussian quadrature.

The evaluation of the analytical solution takes 1000 times longer than the numerical solution of the implicit 2D scheme.

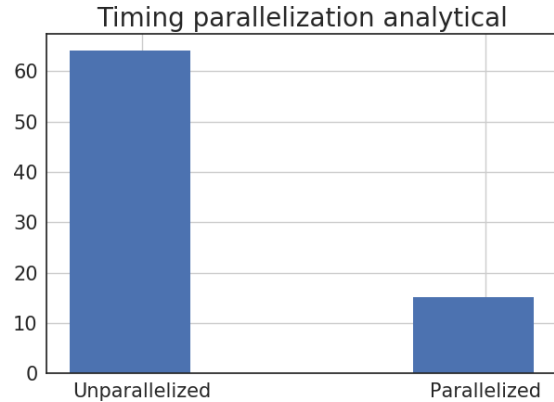


Figure 19: 2D. Analytical. Times parallelized and unparallelized.

Parallelization of the analytical solution gives a speed by a factor of 4.

The figure to the left above shows that the evaluation of the analytical solution takes up considerably more time than the solution of the numerical scheme. The reason for this has to do with the analytical function involves a

sum of a double integral of a double sum (yes, that is correct), resulting in many loops.

We have tried implementing the analytical solution in an efficient way, by using Gaussian Quadrature with Legendre polynomials. In the lectures we have seen that use of Gaussian quadrature is far less demanding compared to the standard Newton-Cotes integral methods like the trapezoidal and Simpson's rule when it comes to the number of necessary integration points.

The huge computational cost of calculating the analytical solution made us parallelize this part of the code (in addition to the implicit Jacobi iteration scheme) using open MP. The figure to the right above shows that the effect of parallingizing the analytical solution is a speed up by a factor of 4, which is considerable.

4 Iterations

The figure below shows the number of iterations needed for convergence, when applying Jacobi's and Gauss-Seidel's iteration methods to the implicit 2D scheme, for different matrix sizes for the mcoefficient matrix of the linear system.

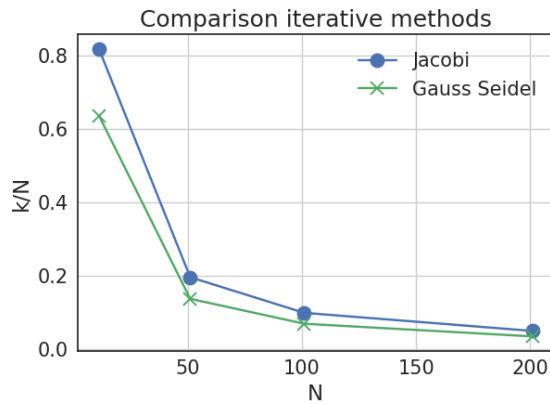


Figure 20: Number of iterations at the last time step divided by N ($A = N \times N$). Tolerance = 0.001.

For both schemes the number of iterations considerable lower then N , making the iterative methods much faster than the classical direct solvers. Gauss-Seidel needs fewer iterations than Jacobi.

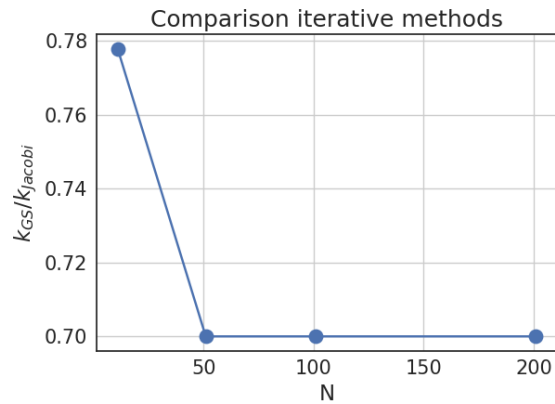


Figure 21: Number of iterations in Gauss-Seidel divided by number of iterations Jacobi.

The number of iterations in Gauss-Seidel seems to stabilize to 2/3 of the number of Jacobi-iterations as the system grows (N increaes), giving a speed up of around 1/3 going from Jacobi to Gauss-Seidel.

The figure to the left above shows that for both iteration methods the number of iterations reduces to less than 10 per cent of the number of elements in one direction in the coefficient matrix, N . From our discussion of FLOP count in the theory section, we know that the fixed point iteration method, which Jacobi and Gauss-Seidel are examples of, is $\mathcal{O}(kN)$ with regards to FLOPS. With the figure above to the left showing that $k < 0.1N$, we get that the iteration methods are considerable more effiecient compared to LU, which goes like $\mathcal{O}(N^2)$ FLOPS, and even faster compared to Gaussian elimination, which goes like $\mathcal{O}(N^3)$ FLOPS, see Hjorth-Jensen [1] p. 173 for FLOP counts on the direct methods.

From the figure to the right we see that Gauss-Seidel needs about 2/3 of the number of iterations that Jacobi needs. This share looks stable as the system increases, that is as N grows.

The fact that Gauss-Seidel needs 1/3 less iterations than Jacobi suggests that Gauss-Seidel should be our preffered method for solving the 2D problem with the implicit scheme. However, the Jacobi algorithm is very easy to parallelize, since each point in the loop only depends on previous iterations. From the previous section we have seen that parallelization by Open MP can give a speed up by a factor of 4, which is considerable more than the 1/3 FLOPS one saves by choosing Gauss-Seidel over Jacobi. Hence by parallingizing Jacobi, we get a faster solver compared to running unparallized Gauss-Seidel.

The results in the figures above are based on the iteration number for the last time period of simulations. We calculated the iteration number for all time periods, and the variation in iteration number between time periods was very low, implying that the above figure is representable for all time periods.

5 Conclusions

Applying Taylor expansions, we have derived numerical finite difference schemes for both the 1D and the 2D diffusion equation. For the 1D diffusion equation we have derived the Forward Euler scheme, the Backward Euler scheme and the Crank-Nicolson scheme. For the 2D diffusion equation we derived an explicit scheme based on the Forward Euler time derivative approximation, and an implicit scheme, based on the Backward Euler approximation of the time derivative.

Truncation errors are derived for all schemes. We find that all schemes have truncation errors that are second order in the space variables, so the truncation error goes like $\mathcal{O}(\Delta h^2)$, where h is a spatial step size. Except for the Crank-Nicolson scheme, which is second order also in time, all the other schemes are first order in time.

We have performed stability analysis of all schemes, using Neumann stability analysis. Except for the Forward Euler scheme, all schemes are unconditionally stable. The Forward Euler scheme is found to be stable for $\Delta t/\Delta x^2 < 1/2$, in the 1D case, and stable for $\Delta t/h^2 < 1/4$ in the quadratic ($h = \Delta x = \Delta y$) 2D case.

Analytical solutions for the 1D and the 2D diffusion equation are derived using the separation of variables technique.

The 1D cases are implemented in c++ by using one single scheme, the θ -scheme. We show how all the schemes in the 1D case are special cases of the θ -scheme.

Linear system formulations are derived for all schemes. In the 1D case, using the θ -scheme, we find that all the schemes corresponds to versions of a tridiagonal linear systems. For tridiagonal systems, the Thomas algorithm, which is much more efficient with regard to FLOPS compared to Gaussian elimination and LU-decomposition, can, and is, used. We utilize the fact that the coefficients in the coefficient matrix are constants by creating a version of the Thomas algorithm that is more efficient than the original Thomas algorithm.

In the 2D case, the linear system is still sparse, but it is no longer tridiagonal. The explicit scheme is still straight forward to implement, while the implicit scheme no longer can be solved by the Thomas algorithm. Other methods are necessary for obtaining efficient solutions of the implicit 2D scheme.

Iterative method are potentially very efficient methods solving the kind of system we have in the 2D implicit scheme. We present a general iterative scheme, the fixed point iteration scheme. Conditions for convergence of the iterative systems to the true solutions are presented. We derive a FLOP count of the iterative scheme, and find that it goes like $\mathcal{O}(kN)$ FLOPS, where k is the number of iterations and N is the dimension of the matrix A of the linear system.

We present two versions of the general fixed point iteration scheme: the Jacobi method and the Gauss-Seidel method. It is shown how methods relates to the iterative fixed point scheme. We derive and implement algorithms for the 2D implicit scheme for both methods. The number of iterations needed for convergence of the iterative schemes reveals huge savings regards to FLOPS compared to direct methods like Gaussian elimination and LU-decomposition. Gauss-Seidel is shown to be more efficient than Jacobi, Gauss-Seidel needing only 2/3 of the iterations of Jacobi.

In order to speed up the calculations further, we use parallel computing by applying Open MP. Open MP is applied to the Jacobi solver for the 2D implicit scheme and for the numerical evaluation of the analytical solution. We find that Open MP gives a speed up of these parts of the code by a factor of four.

We find that the numerical evaluation of the analytical solution in 2D takes up considerable more time than evaluation of the numerical schemes! The numerical implementation involves mixes of double sums and double integrals, resulting in many FLOPS. In order to increase the efficiency of the numerical evaluation of the analytical solution, we apply Gaussian quadrature with Legendre polynomials, which is known to be much more efficient compared to

standard integral methods like e.g. Simpson's rule, for evaluation of the integral.

Our numerical results confirms the theoretical derivations. Forward Euler is sensitive around the stability limit, both in the 1D case and in the 2D case. A slight increase of the time step parameter above the stability limit gives unstable solutions, while a slight decrease of the time step parameter below the stability limit gives stable solutions.

For 1D the Backward Euler and Crank-Nicolson seems to perform better compared to the Forward Euler scheme, also when the stability condition for the Forward Euler scheme is satisfied. The Crank-Nicolson scheme looks more precise, giving results closer to the analytical solution, compared to the Backward Euler scheme. This last point was not unexpected, knowing from the theoretical derivations that the truncation error of the Crank-Nicolson scheme is one order higher in time compared to the Backward Euler scheme.

Interesting extension to this analysis would be calculations of the numerical truncation errors, comparing these to the analytical derivations. A more detailed analysis of the Gaussian quadrature, comparing it with other integration methods with regards to number of needed points, would be interesting. Introduction of relaxation parameters to the iterative scheme and studying the effect of this on the efficiency is another potential direction of future work.

6 Bibliography

- [1] Hjorth-Jensen, M.(2015) Computational physics. Lectures fall 2015. <https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/Lectures>
- [2] <https://github.com/CompPhysics/ComputationalPhysics/tree/master/doc/pub/pde>
- [3] Olver, P.J and Shakiban, C.(2006) Applied linear algebra. Pearson Prentice Hall.