# Diffusion Equation

Vegard Falmår and Sigurd Sørlie Rustad

University of Oslo
Norway

December 18, 2020

## CONTENTS

note to self: kanskje ikke helt fornøyd med siste del av metode 1D og 2D? si noe om stability i 2D? Ellers burde alt være ganske nais. Unit testing mangler. diskutere at de første tidsstegene så er computed best? Derfor ser vi på midt i og stationary state.

## I. INTRODUCTION

The diffusion equation is used in many areas. Everything from how heat is transferred through a stick, to how an oil spill will spread. In this report we will try to solve the diffusion equation for one and two dimensions, and using different algorithms. We are going to study the diffusion equation with a constant collective diffusion coefficient, giving us a linear differential equation identical to the heat equation.

For our one dimensional problem, we use three different algorithms, explicit Forward Euler, implicit Backward Euler and implicit Crank-Nicolson.

For our studies we have used c++ for heavy computation, python for visualization and automation. All the code along with instructions on how to run it, can be cloned from our GitHub repository[1].

## II. THEORY

### The diffusion equation

The full diffusion equation reads

$$\frac{\partial u(\mathbf{r},t)}{\partial t} = \nabla \cdot \left[D(u,\mathbf{r})\nabla u(\mathbf{r},t)\right],$$

where $\mathbf{r}$ is a positional vector and $D(u,r)$ the collective diffusion coefficient. If $D(u,\mathbf{r}) = 1$ the equation simplifies to a linear differential equation

$$\frac{\partial u}{\partial t} = \nabla^2 u(\mathbf{r},t),$$

or

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) u(x,y,z,t) = \frac{\partial u(x,y,z,t)}{\partial t} \quad (1)$$

in cartesian coordinates. In this report we are mainly going to work with the diffusion equation in one and two dimensions, i.e.

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

### Mean absolute percentage error

The mean absolute percentage error (MAPE), is used to predict the accuracy of a prediction. We will use it to measure how good our numerical methods are, compared to theoretical values. Equation (2) shows how to calculate MAPE in percentage,

$$M = \left(\frac{1}{n}\sum_{n=1}^{n}\left|\frac{A_\mathrm{t} - A_\mathrm{c}}{A_\mathrm{t}}\right|\right) \cdot 100\%. \quad (2)$$

Here $A_\mathrm{t}$ and $A_\mathrm{c}$ are the theoretical and computed values, $n$ number of data points and $M$ MAPE in percent.

### Discretization

Equation (1) in one dimension reads

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t} \quad \text{or} \quad u_{xx} = u_t. \quad (3)$$

With $x \in [0, L]$ and boundary conditions

$$u(0,t) = a(t), \quad t \geq 0 \quad \wedge \quad u(L,t) = b(t), \quad t \geq 0,$$

we can approximate the solution by discretization. First introducing $\Delta x = L/(n+1)$ and $\Delta t$ as small steps in $x$-direction and time. Then we can define the value domain of $t$ and $x$,

$$t_j = j\Delta t, \quad j \in \mathbb{N}_0 \quad \wedge \quad x_i = i\Delta x, \quad \{i \in \mathbb{N}_0 | i \leq n+1\}.$$

### Explicit and implicit schemes

It is common to divide numerical algorithms into explicit and implicit schemes. When performing numerical integration, we iterate over a discrete set of grid points at which we evaluate the function in question. In explicit schemes, the value at the next grid point is determined entirely by known or previously calculated values. In implicit schemes, the value is determined by solving a coupled set of equations, often involving matrix or iterative techniques.

Using an implicit method instead of an explicit method usually requires more computation in every step, and they are often harder to implement. It can, in turn, save computation by allowing larger step sizes. Explicit methods are always conditionally stable, however the implicit methods we will use in this report are unconditionally stable. Of course, in order to achieve a desired *accuracy*, the step sizes can not be arbitrarily large for either the explicit or implicit schemes. It is generally the case, though, that implicit schemes allow for larger step sizes than explicit schemes.

[1] github.com/sigurdru/FYS3150/tree/master/Project5

### Explicit Forward Euler

The algorithm for explicit forward Euler in one dimension (from [1] chapter 10.2.1) reads

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

where

$$\alpha = \frac{\Delta t}{\Delta x^2},$$

and a local approximate error of $O(\Delta t)$ and $O(\Delta x^2)$. The discretization is explained in the appropriate section. Note that the expression on the right hand side, used to calculate the value at a time $t_j + \Delta t$, only contains the state of the system at time $t_j$. This can be written as a matrix equation (see [1] chapter 10.2.1)

$$\mathbf{u}_{j+1} = (\mathbb{1} - \alpha \mathbf{B})\mathbf{u}_j$$

where $\mathbb{1}$ is the identity matrix and

$$\begin{bmatrix} 2 & -1 & 0 & 0\dots \\ -1 & 2 & -1 & 0\dots \\ \dots & \dots & \dots & \dots \\ 0 & \dots & -1 & 2 \end{bmatrix} \quad \wedge \quad \mathbf{u}_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \dots \\ u_{n,j} \end{bmatrix}$$

The stability requirement for this algorithm (also from [1] chapter 10.2.1) is

$$\rho(\mathbf{B}) \le 1 \implies \frac{\Delta t}{(\Delta x)^2} \le 1/2 \tag{5}$$

Where $\rho(\mathbf{B})$ is the spectral radius of $\mathbf{B}$:

$$\rho(\mathbf{B}) = \max\{|\lambda| : \det(\mathbf{B} - \lambda\mathbb{1}) = 0\}.$$

In two dimensions ($u(x,y)$) the method for solving is similar. Assuming the same number of integration points in x- and y-direction ($\Delta x = \Delta y = \Delta l$), the algorithm (from [1] chapter 10.2.5) reads

$$u^{l+1}_{i,j} = u^l_{i,j} + \alpha\left[u^l_{i+1,j} + u^l_{i-1,j} + u^l_{i,j+1} + u^l_{i,j-1} - 4u^l_{i,j}\right] \tag{6}$$

where $u^l_{i,j} = u(i\Delta l, j\Delta l, l\Delta t)$ and $\alpha = \Delta t/\Delta l^2$. With this algorithm we will get slightly different stability conditions. The Von Neumann stability analysis gives

$$\frac{\Delta t}{\Delta l^2} \le \frac{1}{4} \tag{7}$$

see [3] for derivation.

### Implicit Backward Euler

The Backward Euler algorithm uses the same centered difference in space as Forward Euler to approximate the second derivative

$$u_{xx} \approx \frac{u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j))}{\Delta x^2}, \tag{8}$$

but a backward formula for the time derivative:

$$u_t \approx \frac{u(x_i, t_j) - u(x_i, t_j - \Delta t)}{\Delta t}, \tag{9}$$

which also has a local truncation error of $O(\Delta t)$ and $O(\Delta x^2)$. Again, defining

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

we obtain by inserting (8) and (9) into our differential equation the equation describing Backward Euler:

$$u_{i,j-1} = -\alpha u_{i-1,j} + (1+2\alpha)u_{i,j} - \alpha u_{i+1,j} \tag{10}$$

Written out for all $i$, equation 10 becomes

$$\begin{aligned} -\alpha u_{0,j} + (1+2\alpha)u_{1,j} - \alpha u_{2,j} &= u_{1,j-1} \\ -\alpha u_{1,j} + (1+2\alpha)u_{2,j} - \alpha u_{3,j} &= u_{2,j-1} \\ &\dots \\ -\alpha u_{n-3,j} + (1+2\alpha)u_{n-2,j} - \alpha u_{n-1,j} &= u_{n-2,j-1} \\ -\alpha u_{n-2,j} + (1+2\alpha)u_{n-1,j} - \alpha u_{n,j} &= u_{n-1,j-1} \end{aligned}$$

In general, this can be rearranged slightly so that

$$\begin{aligned} (1+2\alpha)u_{1,j} - \alpha u_{2,j} &= u_{1,j-1} + \alpha u_{0,j} \\ -\alpha u_{1,j} + (1+2\alpha)u_{2,j} - \alpha u_{3,j} &= u_{2,j-1} \\ &\dots \\ -\alpha u_{n-3,j} + (1+2\alpha)u_{n-2,j} - \alpha u_{n-1,j} &= u_{n-2,j-1} \\ -\alpha u_{n-2,j} + (1+2\alpha)u_{n-1,j} &= u_{n-1,j-1} + \alpha u_{n,j} \end{aligned}$$

Let $\mathbf{v}_j$ be a vector containing the values of $u$ at $n-1$ points in space at a time $t_j$

$$\mathbf{v}_j = \begin{bmatrix} u_{1,j} \\ u_{2,j} \\ \vdots \\ u_{n-2,j} \\ u_{n-1,j} \end{bmatrix} \tag{11}$$

and the vector $\mathbf{b}_j$ be defined as follows:

$$\mathbf{b}_j = \begin{bmatrix} u_{1,j-1} + \alpha u_{0,j} \\ u_{2,j-1} \\ \vdots \\ u_{n-2,j-1} \\ u_{n-1,j-1} + \alpha u_{n,j} \end{bmatrix} \tag{12}$$

As the boundary conditions $u_{0,j}$ and $u_{n,j}$ are specified, we already know every component of $\mathbf{b}_j$. We can then rewrite equation (10) as the matrix equation

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j \tag{13}$$

where **A** is defined as

$$A = \begin{bmatrix} (1+2\alpha) & -\alpha & 0 & 0 & \dots & 0 \\ -\alpha & (1+2\alpha) & -\alpha & 0 & \dots & 0 \\ \vdots & & \ddots & & & \vdots \\ 0 & \dots & 0 & -\alpha & (1+2\alpha) & -\alpha \\ 0 & \dots & 0 & 0 & -\alpha & (1+2\alpha) \end{bmatrix} \tag{14}$$

This is a tridiagonal matrix with $(1+2\alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal.

### Implicit Crank-Nicolson

The Crank-Nicolson algorithm uses a time-centered scheme centered around $t+\Delta t/2$, with a truncation error of $O(\Delta t^2)$ and $O(\Delta x^2)$. The time derivative is given by

$$u_t \approx \frac{u(x_i, t_j + \Delta t) - u(x_i, t_j)}{\Delta t} \tag{15}$$

$$u_{xx} \approx \frac{1}{2\Delta x^2}\Big( u(x_i + \Delta x, t_j) - 2u(x_i, t_j) + u(x_i - \Delta x, t_j) + u(x_i + \Delta x, t_j + \Delta t) - 2u(x_i, t_j + \Delta t) + u(x_i - \Delta x, t_j + \Delta t)\Big)$$

Inserting these two equations into our differential equation, we obtain (see Appendix, section VII A for derivation) that the equation describing the Crank-Nicolson algorithm can be written as a matrix equation

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j \tag{16}$$

The vector $\mathbf{v}_j$ is the same as in the case of Backward Euler, defined in (11). The matrix **A** is a tridiagonal matrix with $2(1+\alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal. The vector $\mathbf{b}_j$ is

$$\mathbf{b}_j = \begin{bmatrix} \alpha u_{0,j-1} + 2(1-\alpha)u_{1,j-1} + \alpha u_{2,j-1} + \alpha u_{0,j} \\ \alpha u_{1,j-1} + 2(1-\alpha)u_{2,j-1} + \alpha u_{3,j-1} \\ \vdots \\ \alpha u_{n-3,j-1} + 2(1-\alpha)u_{n-2,j-1} + \alpha u_{n-1,j-1} \\ \alpha u_{n-2,j-1} + 2(1-\alpha)u_{n-1,j-1} + \alpha u_{n,j-1} + \alpha u_{n,j} \end{bmatrix} \tag{17}$$

## III. METHODS

### A. One dimension

We will start by solving the one dimensional diffusion equation

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{\partial u(x,t)}{\partial t}, \quad x \in [0,1]$$

with initial conditions

$$u(x,0) = 0, \quad 0 < x < 1 \tag{18}$$

and boundary conditions

$$\begin{aligned} u(0,t) &= 0, \quad t \geq 0 \quad \text{and} \\ u(L,t) &= 1, \quad t \geq 0. \end{aligned} \tag{19}$$

As mentioned we use three methods, explicit forward Euler, implicit backward Euler and Crank-Nicolson. When using Explicit forward Euler, we only need to solve the difference-equation described by equation (4). When using Implicit backward Euler er Crank-Nicolson we need to solve the matrix-equations (13) and (16). There are several methods for solving such tridiagonal matrix equations, however we will use the method covered in a previous project. see project[2] page 2 (section A. Solve tridiagonal matrix equation). We will use $\Delta x = 1/10$, $\Delta x = 1/100$ and $\Delta t$ decided by equation (5). We then study compare our results with analytical ones (analytical results are derived below), at two points in time. First $t_1$ where our solution is smooth but curved, then $t_2$ where we have a linear (stable) solution. Hopefuly we will be able to decide on what method is best. We also check what happens when the difference in equation (5) is not satisfied. For all our results we also plot the difference between the results, as well as calculate the MAPE in percent (see equation (2)).

#### Analytic solution to the 1D diffusion equation

With the initial and boundary conditions described by equations (18) and (19) we can find an analytical solution. We expect the heat distribution to converge to a stable state $u_E$ as $t \to \infty$:

$$\lim_{t\to\infty} u(x,t) = u_E(x)$$

This final state should still satisfy our differential equation such that

$$\frac{\partial^2 u_E(x)}{\partial x^2} = \frac{\partial u_E(x)}{\partial t} = 0$$

The solution to this equation is

$$u_E(x) = Ax + B,$$

and the boundary conditions give

$$\begin{aligned} u_E(0) &= B = 0 \\ u_E(L) &= AL = 1 \quad \Rightarrow \quad u_E(x) = \frac{x}{L} \end{aligned}$$

[2] https://github.com/sigurdru/FYS3150/tree/master/Project1

Let us now define the function

$$v(x,t) = u(x,t) - u_E(x) \quad \text{which gives}$$
$$u(x,t) = v(x,t) + u_E(x)$$

We then have

$$\frac{\partial u}{\partial t} = \frac{\partial v}{\partial t} + \frac{\partial u_E}{\partial t} = \frac{\partial v}{\partial t}$$
$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u_E}{\partial x^2} = \frac{\partial^2 v}{\partial x^2}$$

Thus, $u(x,t)$ and $v(x,t)$ should both satisfy the same differential equation (3). The intial and boundary conditions for $v(x,t)$ are

$$v(x,0) = u(x,0) - u_E(x) = -u_E(x)$$
$$v(0,t) = u(0,t) - u_E(0) = 0$$
$$v(0,t) = u(L,t) - u_E(L) = 0$$

In other words, $v$ obeys the same equation as $u$, but with boundary conditions 0.

It is rather straightforward to see that (however see [4] chapter 3 for a more thorough derivation)

$$f_n(x,t) = e^{-C_n^2 t}\left(A\sin(C_n x) + B\cos(C_n x)\right) \tag{20}$$

is a particular solution to the one dimensional diffusion equation:

$$\frac{\partial f_n(x,t)}{\partial t} = \frac{\partial^2 f_n(x,t)}{\partial x^2} = -C_n^2 f_n(x,t)$$

The equation is linear and from the prinsiple of superposition we have that any sum of functions $f_n(x,t)$ with different values of $C_n$ is also a solution. The boundary conditions $v(0,t) = 0$ and $v(L,t) = 0$ give

$$f_n(0,t) = B_n e^{-C_n^2 t} = 0 \quad \Rightarrow \quad B_n = 0$$
$$f_n(L,t) = A_n e^{-C_n^2 t}\sin(C_n L) = 0 \quad \Rightarrow \quad C_n = \frac{n\pi}{L}, n \in \mathbb{Z}$$

The solution $v(x,t)$ can then be written as sum

$$v(x,t) = \sum_n A_n e^{-\left(\frac{n\pi}{L}\right)^2 t}\sin\left(\frac{n\pi}{L}x\right)$$

The coefficients $A_n$ are given by the initial condition $v(x,0) = -u_E(x)$ and can be determined from (see [1] chapter 10.2.4)

$$A_n = \frac{2}{L}\int_0^L -u_E(x)\ \sin\left(\frac{n\pi}{L}x\right)\,\mathrm{d}x$$
$$= -\frac{2}{L^2}\int_0^L x\ \sin\left(\frac{n\pi}{L}x\right)\,\mathrm{d}x$$

The solution to this is

$$A_n = \frac{2\left(\pi n\cos(\pi n) - \sin(\pi n)\right)}{\pi^2 n^2} \tag{21}$$

The complete solution to our differential equation is thus

$$u(x,t) = \frac{x}{L} + \sum_{n=1}^{\infty} A_n e^{-\left(\frac{n\pi}{L}\right)^2 t}\sin\left(\frac{n\pi}{L}x\right), \tag{22}$$

where the coefficients $A_n$ are given by equation (21).

## B. Two dimensions

Moving on to two dimensions we use explicit forward Euler. This means we need to solve the difference equation (6), for $x, y \in [0,1]$. We will use homogeneous Dirichlet boundary conditions

$$\begin{aligned} u(0,y,t) = u(1,y,t) = 0, &\quad 0 \le y \le 1, \quad t \ge 0 \text{ and} \\ u(x,0,t) = u(x,1,t) = 0, &\quad 0 \le x \le 1, \quad t \ge . \end{aligned} \tag{23}$$

and initial conditions given by

$$u(x,y,0) = \begin{cases} 1 & \text{if } 0 < y \le \frac{1}{2} \text{ and } 0 < x < 1 \\ 0 & \text{if } \frac{1}{2} < y < 1 \text{ and } 0 < x < 1 \end{cases} \tag{24}$$

As for the one we will test for different $\Delta t$'s, and compare it to the closed form solution covered bellow. We do one case where the difference in equation (7) is satisfied and one where it's not. We then plot the result and difference between them, also calculating MAPE in percent (equation (2)). For all cases we used $\Delta x = \Delta y = 1/50$.

### Parallelization

When solving the difference equation (6) we parallelize the program using OpenMp. While calculating $u_{i,j}^l$ we only need to worry about the surrounding points for the previous step, namely $u_{i,j}^{l-1}$ and $u_{i\pm1,j\pm1}^{l-1}$. In a grid pattern this would look something like:

$$\begin{array}{ccc} & u_{i,j+1}^{l-1} & \\ u_{i-1,j}^{l-1} & u_{i,j}^{l-1} & u_{i+1,j}^{l-1} \\ & u_{i,j-1}^{l-1} & \end{array}$$

This means that whenever we are done calculating $u_{i,j}^l$ and $u_{i\pm1,j\pm1}^l$ we can start calculating the next time step $u_{i,j}^{l+1}$. We found that if we run through $u_{i,j}^l$ diagonally, we can start calculating the next time step ($u_{i,j}^{l+1}$) one diagonal behind. A visual representation of this can be seen in figure 1. The red and black dots represent values of $u_{i,j}^l$ we have and haven't calculated respectively. We can then make sure one thread is calculating $u_{i,j}^l$ for one value of $l$ diagonally, another core the next time step one diagonal behind, and so forth.

### Analytical solution to the 2D diffusion equation

with boundary and initial conditions described by equations (23) and (24) we can find analytical solutions to the diffusion equation. Lets say $x \in [0,a]$ and

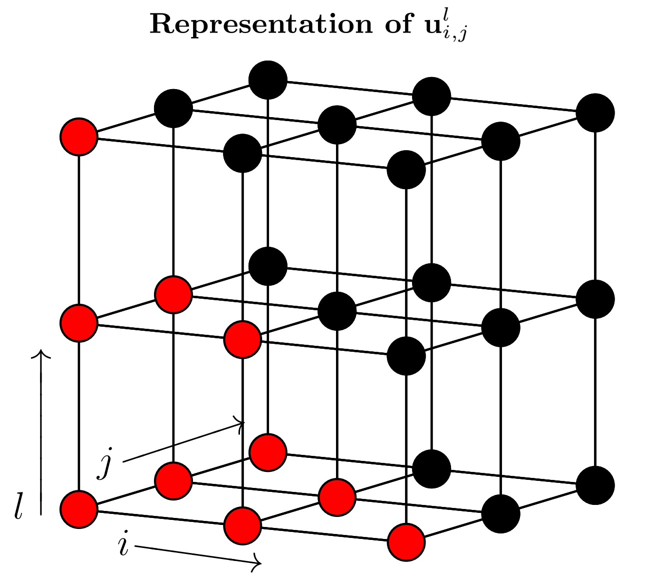


Figure 1. Visualization of how the parallelization works. $i$, $j$ and $l$ represent the indexes of $u_{i,j}^l$ and the red and black dots represent the values of $u_{i,j}^l$ we have and haven't calculated.

$y \in [0, b]$, with homogeneous Dirichlet boundary and initial conditions defined by some function $f(x, y)$. Then, from [2], the general solution is given by equation (25).

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} A_{mn}\sin(\mu_m x)\sin(\nu_n y)e^{-\lambda_{mn}^2 t}, \quad (25)$$

where

$$\mu_m = \frac{m\pi}{a} \quad \wedge \quad \nu_n = \frac{n\pi}{b} \quad \wedge \quad \lambda_{mn} = \sqrt{\mu_m^2 + \nu_n^2},$$

for $m, n, \in \mathbb{N}$ and

$$A_{mn} = \frac{4}{ab}\int_0^a\int_0^b dydx f(x,y)\sin\Big(\frac{m\pi}{a}x\Big)\sin\Big(\frac{n\pi}{b}\Big).$$

Inserting our initial conditions (24) and $a = b = 1$ into the equation above, we get the coefficients

$$\begin{aligned} A_{mn} &= 4\int_0^1\int_0^1 dydx u(x,y,0)\sin(m\pi x)\sin(n\pi y) \\ &= 4\int_0^1 \sin(m\pi x)dx \int_0^{1/2}\sin(n\pi y)dy \\ &= 4\left[-\frac{\cos(m\pi x)}{m\pi}\right]_0^1\left[-\frac{\cos(n\pi y)}{n\pi}\right]_0^{1/2} \\ &= 4\left(-\frac{(-1)^m - 1}{m\pi}\right)\left(-\frac{\cos\big(\frac{n\pi}{2}\big) - 1}{n\pi}\right) \\ &= \frac{4}{\pi^2}\left(\frac{((-1)^m - 1)\big(\cos\big(\frac{n\pi}{2}\big) - 1\big)}{mn}\right) \end{aligned}$$

Inserting this into equation (25)

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\frac{4}{\pi^2}\left(\frac{((-1)^m - 1)\big(\cos\big(\frac{n\pi}{2}\big) - 1\big)}{mn}\right)\cdot \sin(m\pi x)\sin(n\pi y)\exp\big(-((m\pi)^2 + (n\pi)^2)t\big)$$

## C. Unit testing

There are many moving parts in the code we use to produce the results. In order to make sure that our implementation is correct and to help track down potential errors, we have developed specific tests for the different parts of the code. To facilitate testing we have structured our code in distinct units with separate classes for the tridiagonal matrix solver, each of the three different one-dimensional solvers, and the two-dimensional solver. We use Catch2[3] to assert that the computed results match the expected and Valgrind[4] to verify that our implementation does not suffer from memory leakage.

### *Tridiagonal matrix solver*

This class solves a matrix equation $\mathbf{A}\mathbf{u} = \mathbf{b}$ for $\mathbf{u}$ when $\mathbf{A}$ is a tridiagonal matrix. The test case we have used is

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 6 \end{bmatrix}$$

which has the analytic solution

$$\mathbf{u} = [6,\ 11,\ 14,\ 14,\ 10]^{\mathrm{T}}$$

### *Explicit Forward Euler*

Equation (4) describes how the vector $\mathbf{u}$ changes from one time step to the next with Forward Euler. The test case we have used for this class is that with $\alpha = 2$

$$\mathbf{u}_j = [0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 3,\ 2,\ 1,\ 4,\ 6]^{\mathrm{T}}$$

should give

$$\mathbf{u}_{j+1} = [1,\ 1,\ 2,\ 3,\ 4,\ -1,\ 5,\ 2,\ 9,\ 2,\ 2]^{\mathrm{T}}$$

in the next time step.

[3] https://github.com/catchorg/Catch2
[4] https://valgrind.org/

### *Implicit Backward Euler*

Equation (10) describes how the vector $\mathbf{u}$ changes from one time step to the next with Backward Euler. Here, the test verifies that with $\alpha = 2$

$$\mathbf{u}_j = [1,\ 14,\ -10,\ 18,\ 4,\ 2]^{\mathrm{T}}$$

gives

$$\mathbf{u}_{j+1} = [1,\ 4,\ 2,\ 6,\ 4,\ 2]^{\mathrm{T}}$$

in the next time step

### *Implicit Crank-Nicolson*

Using equation (26), which describes the Crank-Nicolson algorithm, we have developed the test case that with $\alpha = 2$, the initial state

$$\mathbf{u}_j = [1,\ 6,\ 14,\ 4,\ 2,\ 2]^{\mathrm{T}}$$

should give

$$\mathbf{u}_{j+1} = [1,\ 4,\ 2,\ 6,\ 4,\ 2]^{\mathrm{T}}$$

in the next time step.

### *Two dimensions*

In the two-dimensional case, knowing the state $\mathbf{u}^j$ in one time step, equation (6) directly gives the state $\mathbf{u}^{j+1}$ in the next time step. Starting from a state

$$\mathbf{u}^j = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

we have solved the equation for two time steps with $\alpha = 2$, giving us the following:

$$\mathbf{u}^{j+1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 4 & -5 & 0 \\ 0 & -2 & 5 & -8 & 0 \\ 0 & -25 & -14 & -35 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{u}^{j+2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -31 & -18 & 27 & 0 \\ 0 & -16 & -75 & -14 & 0 \\ 0 & 143 & -12 & 201 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

We run the 2D-solver using both one and two cores and verify that the computed results match the expected.

## IV. RESULTS

Figure 2 shows the results produced by the Forward Euler algorithm for a few select time steps. The dashed lines show the analytic solution using 200 addends in the Fourier sum in equation (22). The top two figures show the solution with $N_x = 10$, that is, eleven grid points with a distance $\Delta x = 1/10$ between. The bottom figure shows $N_x = 100$, giving $\Delta x = 1/100$. The first and third figure are produced with the stability condition satisfied ($\Delta t/\Delta x^2 = 0.4$), whereas the second has $\Delta t/\Delta x^2 = 0.6$.

Figure 3 shows the error for different number of integration points and values of $\Delta t$. Since the error for $\Delta x = 1/10$ and $\Delta t/\Delta x^2 = 0.6$ is obviously very large (see the second plot in figure 2), we have chosen to include here the errors for $\Delta t/\Delta x^2 = 0.4$ (the first and third plot) and $\Delta t/\Delta x^2 = 0.5$ (the second plot). The first two plots have $N_x = 10$, i.e. eleven integration points, and the third plot has $N_x = 100$. Different colors correspond to different times. The mean absolute percentage error (MAPE) can be read from the title for the different times.

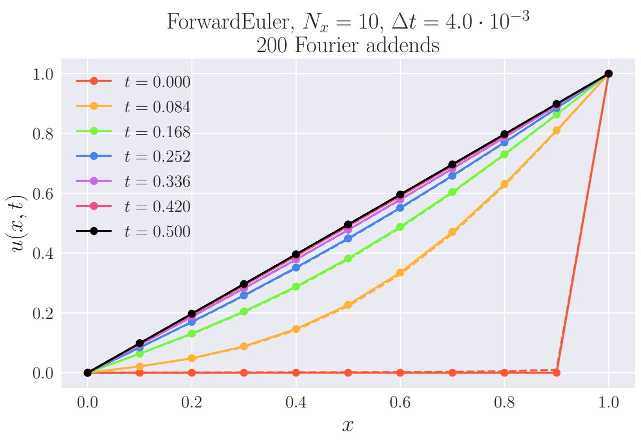
Forward 10 og 100 Error 100, dt = 4 og 5
To plots for 3D, en med stabilitetsbet. oppfylt, en uten

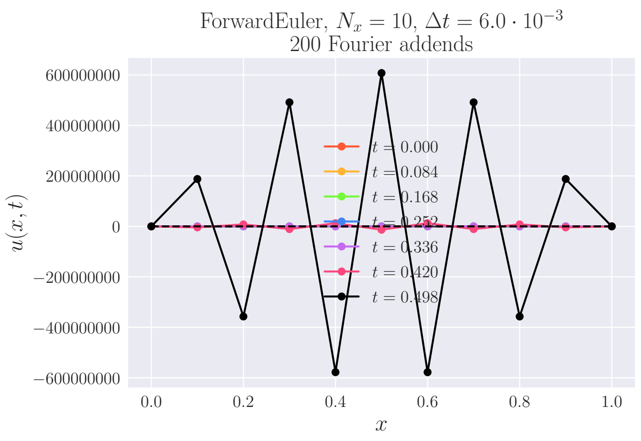
## V. DISCUSSION

LU-decomposition

## VI. CONCLUSION

Conclusion





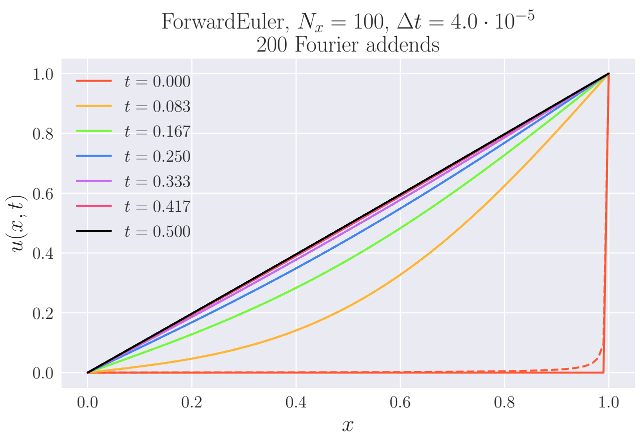


Figure 2. The results produced by the Forward Euler algorithm. The dashed lines show the analytical results, calculated with 200 addends in the Fourier sum in equation (22). The different colors denote the solution at different times. The top two figures show the solution with $N_x = 10$, that is, eleven grid points with a distance $\Delta x = 1/10$ between. The bottom figure shows $N_x = 100$, giving $\Delta x = 1/100$. The first and third figure are produced with the stability condition satisfied ($\Delta t/\Delta x^2 = 0.4$), whereas the second has $\Delta t/\Delta x^2 = 0.6$.

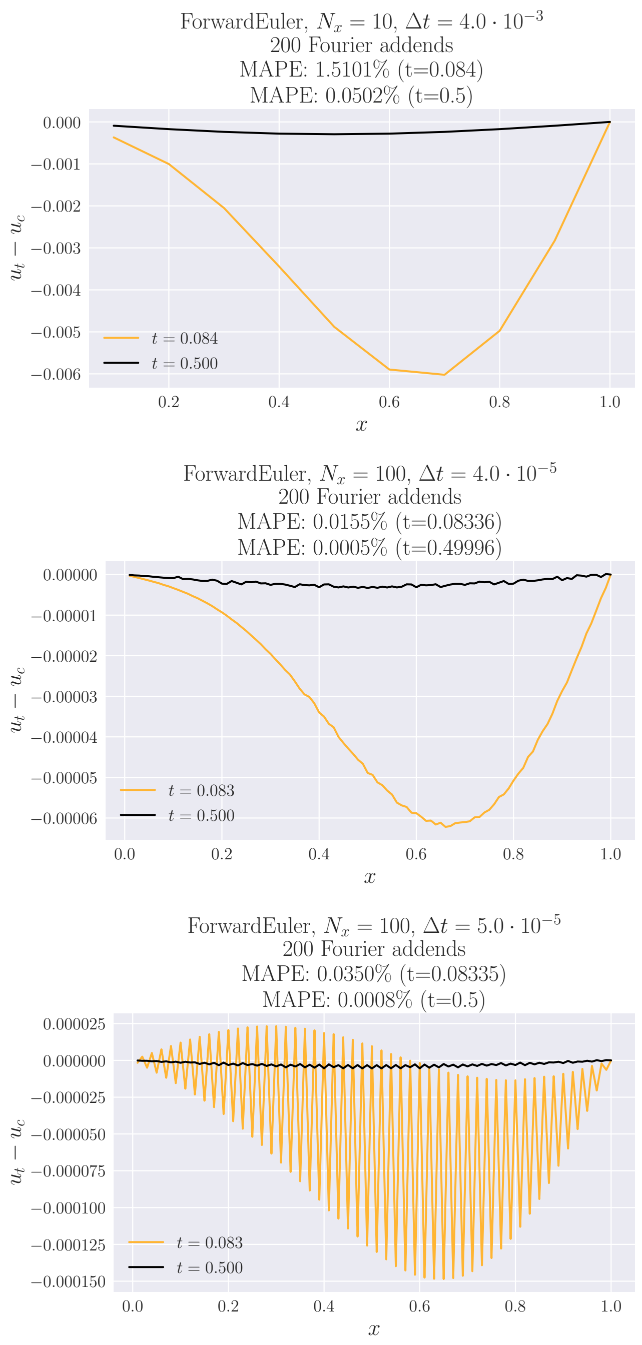


Figure 3. The error for different number of integration points and values of $\Delta t$ for the Forward Euler algorithm. Different colors correspond to different times. The mean absolute percentage error (MAPE) can be read from the title for the different times. The first and third plot have $\Delta t/\Delta x^2 = 0.4$ and the second has $\Delta t/\Delta x^2 = 0.5$. The first two plots have $N_x = 10$, i.e. eleven integration points, and the third plot has $N_x = 100$.

[1] Morten Hjorth-Jensen, Computational Physics, Lecture Notes Fall 2015, August 2015, https://github.com/CompPhysics/ComputationalPhysics/blob/master/doc/Lectures/lectures2015.pdf.
[2] Ryan C. Daileda, Trinity University, Partial Differential Equations, March 6, 2012, http://ramanujan.math.trinity.edu/rdaileda/teach/s12/m3357/lectures/lecture_3_6_short.pdf.
[3] massachusetts institute of technology, Numerical Methods for Partial Differential Equations, March 31. 2009, https://ocw.mit.edu/courses/mathematics/18-336-numerical-methods-for-partial-differential-equations-spring-2009/lecture-notes/MIT18_336S09_lec14.pdf.
[4] Tveito, Aslak and Winther, Ragnar, Introduction to Partial Differential Equations A Computational Approach, 2005.

## VII. APPENDIX

### A. Derivation of equation for the Crank-Nicolson scheme

Again we define

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

Inserting the approximations of $u_t$ and $u_{xx}$ stated in the Theory section into our differential equation, we get

$$\begin{aligned}
u_t &= u_{xx} \\
\frac{u_{i,j+1} - u_{i,j}}{\Delta t} &= \frac{1}{2\Delta x^2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) \\
u_{i,j+1} - u_{i,j} &= \frac{\alpha}{2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) \\
u_{i,j+1} - \frac{\alpha}{2}\Big(u_{i+1,j+1} - 2u_{i,j+1} + u_{i-1,j+1}\Big) &= u_{i,j} + \frac{\alpha}{2}\Big(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\Big) \\
2u_{i,j+1} - \alpha u_{i+1,j+1} + 2\alpha u_{i,j+1} - \alpha u_{i-1,j+1} &= 2u_{i,j} + \alpha u_{i+1,j} - 2\alpha u_{i,j} + \alpha u_{i-1,j} \\
-\alpha u_{i-1,j+1} + 2(1+\alpha)u_{i,j+1} - \alpha u_{i+1,j+1} &= \alpha u_{i-1,j} + 2(1-\alpha)u_{i,j} + \alpha u_{i+1,j}
\end{aligned} \tag{26}$$

Written out explicitly for all $i$, the equation reads

$$\begin{aligned}
-\alpha u_{0,j+1} + 2(1+\alpha)u_{1,j+1} - \alpha u_{2,j+1} &= \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} \\
-\alpha u_{1,j+1} + 2(1+\alpha)u_{2,j+1} - \alpha u_{3,j+1} &= \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\
&\vdots \\
-\alpha u_{n-3,j+1} + 2(1+\alpha)u_{n-2,j+1} - \alpha u_{n-1,j+1} &= \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\
-\alpha u_{n-2,j+1} + 2(1+\alpha)u_{n-1,j+1} - \alpha u_{n,j+1} &= \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j}
\end{aligned}$$

Similar to the case of Backward Euler, this set of equations can be rearranged as follows

$$\begin{aligned}
2(1+\alpha)u_{1,j+1} - \alpha u_{2,j+1} &= \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} + \alpha u_{0,j+1} \\
-\alpha u_{1,j+1} + 2(1+\alpha)u_{2,j+1} - \alpha u_{3,j+1} &= \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\
&\vdots \\
-\alpha u_{n-3,j+1} + 2(1+\alpha)u_{n-2,j+1} - \alpha u_{n-1,j+1} &= \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\
-\alpha u_{n-2,j+1} + 2(1+\alpha)u_{n-1,j+1} &= \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j} + \alpha u_{n,j+1}
\end{aligned}$$

We can define the vector $\mathbf{b}_j$ to hold the values on the right side of each of these equations:

$$\mathbf{b}_j = \begin{bmatrix} \alpha u_{0,j} + 2(1-\alpha)u_{1,j} + \alpha u_{2,j} + \alpha u_{0,j+1} \\ \alpha u_{1,j} + 2(1-\alpha)u_{2,j} + \alpha u_{3,j} \\ \vdots \\ \alpha u_{n-3,j} + 2(1-\alpha)u_{n-2,j} + \alpha u_{n-1,j} \\ \alpha u_{n-2,j} + 2(1-\alpha)u_{n-1,j} + \alpha u_{n,j} + \alpha u_{n,j+1} \end{bmatrix} \tag{27}$$

Using this and the definition of the vector $\mathbf{v}_j$ from (11), we can write the set of equations as

$$\mathbf{A}\mathbf{v}_j = \mathbf{b}_j$$

where the matrix $\mathbf{A}$ is a tridiagonal matrix with $2(1 + \alpha)$ on the diagonal and $-\alpha$ directly above and below the diagonal.