FYS4150
COMPUTATIONAL PHYSICS
Project 5: Numerical solutions of the diffusion equation

Martin Krokan Hovden

December 19, 2019

Abstract

In this report we study the performance of numerical methods for solving the diffusion equations in one and two dimensions. We start with finding the analytical solutions of the general equation. The goal is then to test and compare the algorithms in terms of accuracy and computational time. When the code is working properly, we will solve a more practical problem where we study the temperature in the lithosphere.

For the one dimensional case we look at the solution for two different times, t = 0.02 and t = 0.25. For t=0.02 the solution is curved while for t=0.25 the solution is almost linear between the boundaries. For a grid with $\Delta t = 0.001$ and $\Delta x = 0.1$ the best MSE is obtained by Crank Nicolson. The MSE's is 0.00104 and 2.10e-6 for the given t-values. For a grid with $\Delta t = 0.00001$ and $\Delta x = 0.01$ the accuracy improves and we get MSE's of 1.045e-7 and 3.08e-9 for the time values. The main problem with the Crank Nicolson is the computational time. The best method when it comes to time is the Forward Euler method. The main problem with the Forward Euler method is its stability criteria.

In two dimension we implement the Forward Euler. The results are good, and we can improve accuracy by decreasing Δt and Δx. However, due to bad stability of the Forward Euler scheme we have to be careful when choosing the parameters. This is also the case in one dimension.

For the last part we study the temperature distribution in the lithosphere in a period from 1Gy ago until today. We assume that 1Gy ago the lithospehere were enriched with radioacitve elements. We find that the steady state before radioactive enrichement is a linear function. When constant heat is added from the radioactive elements, the temperature increases until it reaches a new steady state. We also look at a case where the radioactive elements decay. In this case the temperature begins to rise but after a while it starts to decrease.

Contents

1 INTRODUCTION

Partial differential equations is important in every parts of science and engineering. Their application ranges from modelling the temperature in the lithosphere to pricing options in finance using the famous Black and Scholes equation. Some differential equations have analytic solutions. However, in general we are not able to find a closed form solution. We then solve the problem using numerical methods instead. New methods for solving differential equations are always being researched. One recent approach is to use neural networks for solving them. However, the method of choice is still to use numerical methods.

In this report we study the diffusion equation. The diffusion equation in its most simple form is given by

$$\nabla^2 u(x,t) = \frac{\partial u(x,t)}{dt}. \tag{1}$$

It can for example be used for modelling the heat in a rod or the fluid flow between two moving plates. For the last part of the report we will also use the diffusion equation to simulate the temperature distribution in the lithosphere with radioactive enrichment. Code for three well known examples of numerical solvers for the diffusion equation is developed from scratch. The methods are the Forward Euler (FE) scheme, the Backward Euler (BE) scheme and the Crank Nicolson (CN) scheme. The theory behind the methods is studied in depth and various properties of the algorithms is discussed. For the diffusion equation in its most simple form we can easily derive analytical expression. This will be used to compare the performance of our algorithms. The comparison is done for different grids to see how the methods behave. Lastly we will set up the diffusion equation for simulating the temperature in the lithosphere. For this problem we will scale the equation to easier be able to use the methods derived for the simple case.

The report is split into four main parts. First we will present the theory behind the diffusion equation and the numerical methods. By deriving the methods from scratch we will get an understanding of how the methods finds the solution and pitfalls we have to avoid when setting up the algorithm. We will also study the problem of the temperature in the lithosphere and use scaling to solve the problem. After that the structure of the GitHub repository is explained. Some of the code is presented in the theory part. However, I have tried to comment the code well so I will refer to the comments and doc-string for more detailed explanations on how the algorithms are developed. After that the results for the three different algorithms are presented and their performance is discussed. The report is wrapped up with a short conclusion on the main findings in the report and ideas for future work.

2 THEORY

The theory and methods part is based on the lecture notes written by Morten Hjorth-Jensen for the course FYS4150 at the University of Oslo [1] and the book "Introduction to Partial Differential Equations" written by Tveito et. al [2]. For more detailed derivations and further information, these books are excellent references.

We will start by introducing the problem to be studied in one and two dimensions before we look at the three different numerical methods. After that we present the theory behind the problem of simulating the temperature in the lithosphere and how we can modify the previously discussed methods for solving the new problem.

2.1 Description of the problem in one dimension

In this project we study the diffusion equation. The diffusion equation is a partial differential equation. In its most general form it is given by

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

It can be used to model various problems in physics, for example the temperature gradient in a rod or the fluid flow between two plates where the upper plate moves[1].

The function we want to find is $u(x,t)$, where x is the spatial coordinate and t is time. The currently stated problem can be solved, but the solution is not unique. We want to find the solution for a specific set of conditions, so we introduce initial conditions and boundary conditions. The initial condition we will use is

$$u(x,0) = g(x) \quad 0 < x < L \tag{3}$$

where we set L = 1 and $t \geq 0$. In addition we have boundary conditions

$$u(0,t) = a(t) \quad t \geq 0 \tag{4}$$

and

$$u(1,t) = b(t) \quad t \geq 0. \tag{5}$$

One thing we will see is that the diffusion equation changes rapidly in the beginning and then slow down. Intuitively, we would expect the steady state temperature distribution to be linear between the two end-points if the temperatures at the ends is constant. When $t \to \infty$ the solution reaches a steady state, which means that the solution no longer changes with time. This can also be simulated numerically. Ideally we would have to simulate for $t \to \infty$. On a computer this is not possible. However, by simulating for a long enough time we will be able to reach a good approximation of the steady state. This will be seen later.

In general we can not find analytical solution to PDE's. However, the analytical solution can be found for the diffusion equation with the given conditions. This is beneficial since we are able to measure how good our numerical methods works and study the performance in detail.

2.2 Description of the problem in two dimensions

For the two dimensional case the solution is dependent on t, x and y. The number of spatial dimensions increase by one. By extending the diffusion equation to two dimensions we can write it in the general form

$$\nabla^2 u(x,y,t) = \frac{d^2u(x,y,t)}{dx^2} + \frac{d^2u(x,y,t)}{dy^2} = \frac{du(x,y,t)}{dt} \tag{6}$$

where

$$t > 0 \text{ and } x, y \in [0,1]. \tag{7}$$

The initial condtion we will use is

$$u(x,y,0) = \sin(\pi x)\sin(\pi y) \tag{8}$$

and all boundary conditions equal to zero. Intuitively, this would lead to the temperature being zero in the whole domain in the steady state.

2.3 Analytic solutions

In this section we derive analytical expressions for the solution of the diffusion equation in one and two spatial dimensions.

2.3.1 Analytic solution in one dimension

An often used method for solving the diffusion equation analytically is by using separation of variables. This means that we guess that the solution is. One way to solve the PDE is to use separation of variables. We assume that the solution of the equation can be written as a product of two function where one is only dependent on time and the other is only dependent on the spatial coordinates. We can then write

$$u(x,t) = ax + F(x)G(t). \tag{9}$$

From the boundary conditions we see that

$$u(0,t) = F(0)G(t) = 0 \tag{10}$$

and

$$u(1,t) = a + F(1)G(t) = 1 \tag{11}$$

By setting $a = \frac{1}{L}$ we see that

$$u(1,t) = 1 + F(1)G(t) = 1. \tag{12}$$

This means that we have to have

$$F(1)G(t) = 0 \tag{13}$$

By introducing v(x,t) = F(x)G(t), we can now solve v(x,t) with the standard boundary conditions

$$v(0,t) = v(1,t) = 0. \tag{14}$$

We can then write

$$u(x,t) = F(x)G(t) \tag{15}$$

where F is only dependent on the spatial coordinates and G on time. We insert this into the original equation and get that

$$\frac{F''(x)}{F(x)} = \frac{G''(t)}{G(t)} \tag{16}$$

Since this have to be true for all t and x we note that each side have to be equal to an constant $lambda^2$. We are now left with two ordinary differential equations to solve. We then get

$$\frac{F''(x)}{F(x)} = \frac{G''(t)}{G(t)} = -\lambda^2 \tag{17}$$

The minus is to make the next expression more compact.

$$F''(x) + \lambda^2 F(x) = 0 \tag{18}$$

and

$$G'(t) + \lambda^2 G(t) = 0. \tag{19}$$

Both equations can be solved using standard techniques for ordinary differential equations. See for example []. The solutions are

$$F(x) = A\sin(\lambda x) + B\cos(\lambda x) \tag{20}$$

and

$$G(t) = Ce^{-\lambda^2 t} \tag{21}$$

We can find the coefficients of F(x) by using the boundary conditions. Since $F(0) = 0$ we get that

$$B\cos(\lambda x) = 0 \tag{22}$$

so $B = 0$. From the other boundary condition we get that

$$F(1) = A\sin(\lambda) = 0 \tag{23}$$

so $\lambda = \pi k$ where k is an integer. By combining them we find that one solution to the partial problem is

$$v_n(x,t) = A_n \sin(\pi n x)e^{-(\pi n)^2 t} \tag{24}$$

and since the diffusion equation is linear, linear combinations of different solutions are also solutions

$$v(x,t) = \sum_{n=1}^{\infty} A_n \sin(\pi n x)e^{-(\pi n)^2 t} \tag{25}$$

By inserting this into equation 9 we see that

$$u(x,t) = x + \sum_{n=1}^{\infty} A_n \sin(\pi n x)e^{-(\pi n)^2 t} \tag{26}$$

To find the coefficient A_n we use the initial condition. This gives

$$u(x,0) = x + \sum_{n=1}^{\infty} A_n \sin(\pi n x) = 0. \tag{27}$$

From this it follows that

$$\sum_{n=1}^{\infty} A_n \sin(\pi n x) = -x. \tag{28}$$

By multiplying with $\sin(m\pi x)$ and integrating from zero to one with respect to x we get

$$\sum_{n=1}^{\infty} A_n \int_0^1 \sin(m\pi x)\sin(n\pi x)dx = -\int_0^1 x\sin(m\pi x)dx. \tag{29}$$

It can be shown that [2]

$$\int_0^1 \sin(m\pi x)\sin(n\pi x) = \delta_{mn} = \begin{cases} 1/2 & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}. \tag{30}$$

This means that the sine functions are orthogonal. Solving the integrals in equation 29 results in

$$A_m = -\frac{2}{m\pi}(-1)^{m+1}. \tag{31}$$

The complete expression for the analytic solution can then be written as

$$u(x,t) = x - \sum_{n=1}^{\infty} \frac{2}{n\pi}(-1)^{n+1}\sin(n\pi x)e^{-(n\pi)^2 t}. \tag{32}$$

This can easily be implemented in python with the following code

```python
v = 0
for n in range(1, 100):
        v += -2/(n*np.pi)*(-1)**(n+1)*np.exp(-(n*np.pi)**2*t)*np.sin(n*np.pi*x_values)
return x_values + v
```

where the upper limit of the range can be chosen according to the need for precision.

2.3.2 Analytic solution in two dimensions.

To test our algorithm we will derive the analytic solution for a simple example. The boundary conditions we will use for our test are given by

$$u(0,y,t) = u(x,0,t) = (1,y,t) = (x,1,t) = 0 \tag{33}$$

and the initial condition given by

$$u(x,y,0) = \sin(\pi x)\sin(\pi y). \tag{34}$$

We let $0 \leq x \leq 1$, $0 \leq y \leq 1$ and $t \geq 0$.

To solve the two dimensional case we will use the same technique as what we did for the one dimensional case. We start by separating the equations and then use the superposition principle to contruct the full solution. However, for the given initial conditions and boundary conditions we say that the solution is on the form

$$u(x,y,t) = X(x)Y(y)G(t). \tag{35}$$

where X is dependent on x, Y is dependent on y and G is dependent on t. By inserting this into equation 6 we see that

$$\frac{G'(t)}{G} = \frac{X''(x)}{X} + \frac{Y''(y)}{Y}. \tag{36}$$

Again, both sides have to be constants. We can then separate the problem into

$$X''(x)/X(x) = -\lambda_x^2 \tag{37}$$

$$Y''(y)/Y(y) = -\lambda_y^2 \tag{38}$$

and

$$G'(t)/G = -(\lambda_x^2 + \lambda_y^2). \tag{39}$$

The solutions is as for the one dimensional case,

$$X(x) = A\sin(\lambda_x x) \tag{40}$$

$$Y(y) = B\sin(\lambda_y y) \tag{41}$$

and

$$G(t) = e^{-(\lambda_x^2+\lambda_y^2)t}. \tag{42}$$

To fulfil the boundary conditions, we need $\lambda_x = m\pi$ and $\lambda_y = n\pi$ and we get that

$$X(x) = \sin(m\pi x) \text{ and } Y(y) = \sin(n\pi y) \tag{43}$$

and

$$G(t) = e^{-(m^2+n^2)\pi^2 t}. \tag{44}$$

Combining the separated solutions gives

$$u_{mn}(x,y,y) = A_{mn}\sin(m\pi x)\sin(n\pi y)e^{-(m^2+n^2)\pi^2 t} \tag{45}$$

and by the principle of superposition for linear pde's, we get that

$$u(x,y,t) = \sum_{m=1}^{\infty}\sum_{n=1}^{\infty} A_{mn}\sin(m\pi x)\sin(n\pi y)e^{-(m^2+n^2)\pi^2 t}. \tag{46}$$

The coefficients A_{mn} can be found using the same technique as before. We use the initial condition and by multiplying by $\sin(k\pi x)\sin(k\pi y)$ and integrating from 0 to 1 we get that

$$\sum_{m=1}^{\infty}\sum_{n=1}^{\infty}\int_0^1\int_0^1 A_{mn}\sin(m\pi x)\sin(n\pi y)\sin(k\pi x)\sin(j\pi y)dxdy = \int_0^1\int_0^1 \sin(\pi x)\sin(\pi x)\sin(k\pi x)\sin(j\pi y)dxdy. \tag{47}$$

By using the orthogonality of $\sin(m\pi x)$ and $\sin(n\pi x)$ as for the one dimensional case we get that

$$A_{mn} = \delta_{1,m}\delta_{1,n}. \tag{48}$$

The analytic solution then becomes

$$u(x,y,t) = \sin(\pi x)\sin(\pi y)e^{-2\pi^2 t} \tag{49}$$

since all other terms cancel out due to the coefficient. The analytic solution in two dimensions can be implemented in python with the following code snippet

```python
return np.sin(np.pi*x_values)*np.sin(np.pi*y_values)*np.exp(-2*(np.pi**2)*t)
```

2.4 Numerical methods for solving the PDE

We will now turn the focus on numerical methods for solving the diffusion equation. We will study three different schemes. Common for all of them is the use of numerical approximations of the derivatives and the second derivatives on a discretization of the domain. This is called the finite difference approach. Stability conditions and truncation error is presented for all algorithms, and we will see that the methods behaves quite differently.

For all the methods we make a grid in the spatial coordinates and in time. Let x_i denote the i'th coordinate point and t_j is the j'th time step. All of the numerical methods makes approximations of the functions derivatives at each grid point and tries to find a solution that satisfies the equation for each grid point. The idea is to replace the terms in the diffusion equation with numerical approximations of the derivatives. Using a finer grid typically leads to a better approximation of the solution. We will use the notation u_i^j where i is the spatial index and j is the time step. We can then write $u(x_i, t_j + \Delta t) = u_i^{j+1}$ and $u(x_i + \Delta x, t) = u_{i+1}^j$.

2.4.1 Forward Euler in one dimension

Forward Euler method is an explicit forward Euler scheme. This means that the values in the next time step is only dependent on values in the previous time step. For the forward Euler method we approximate the derivatives by

$$u_t \approx \frac{u_i^{j+1} - u_i^j}{\Delta t} \tag{50}$$

where u_t denotes the derivative of u with respect to t. We can also find

$$u_{xx} \approx \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}. \tag{51}$$

where u_{xx} is the second derivative of u with respect to x. The local approximation error of the two approximations are $O(\Delta t)$ and $O(\Delta x^2)$, respectively. Inserting the approximations into the diffusion equation gives that

$$\frac{u_i^{j+1} - u_i^j}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}, \tag{52}$$

We are interested in finding an expression for the solution in the next time step. By rearranging the equation we can find an recursive expression for finding the time development of u,

$$u_i^{j+1} = \frac{\Delta t}{\Delta x^2}\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j\right) + u_i^j \tag{53}$$

The forward step can be implemented in python with the following code snippet

```
u[t_index, x_index] = alpha*(u[t_index-1, x_index + 1] - 2*u[t_index-1, x_index] + \
              u[t_index-1, x_index - 1]) + u[t_index-1, x_index].
```

By iterating over the grid we can update the solution for the time-values we want. By introducing the vector

$$U^j = \begin{bmatrix} u_1^j \\ u_2^j \\ \cdots \\ u_n^j \end{bmatrix} \tag{54}$$

and the matrix

$$A = \begin{bmatrix} 1-\alpha & \alpha & 0 & \cdots & \cdots & 0 \\ \alpha & 1-\alpha & \alpha & 0 & \cdots & 0 \\ 0 & \alpha & 1-\alpha & \alpha & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 0 & \alpha & 1-\alpha \end{bmatrix}, \tag{55}$$

we can write the scheme as

$$U^{j+1} = AU^j. \tag{56}$$

We can then find the u-values for the j'th timestep by using the matrix iteratively

$$U^j = A^j U^0 \tag{57}$$

One of the main problems with the forward euler method is the bad stability conditions. The stability condition is $\alpha \leq \frac{1}{2}$. As mentioned, the solution of the diffusion equation is smooth and changes quite slowly. Short time-steps would then be unnecessary, because so little will happen between each step. It would then be beneficial to use methods where there is no restrictions on the choice of step lengths. We will look closer at two methods that solves this problem later. We will now derive the stability criterion for the Forward Euler scheme.

2.4.2 Stability of the forward Euler scheme

The forward euler method is easy to implement and provides an intuitive solution to the problem. However, it suffers from weak stability conditions. This means that if we are not careful when choosing the grid, we can end up with a solution that blows up. By rewriting equation 53 as

$$\boldsymbol{u}^{j+1} = A\boldsymbol{u}^{j} \tag{58}$$

the stability criterion is that

$$\rho(A) < 1 \tag{59}$$

where

$$\rho(A) = \max\{|\lambda| : \det(A - \lambda I) = 0\}. \tag{60}$$

This means that the largest eigenvalue have to be smaller than zero. We can now find the eigenvalues of A. The first step is to introduce the new matrix B and write A as

$$A = I - \alpha B, \tag{61}$$

where B is

$$\begin{bmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & -1 & 2 \end{bmatrix}. \tag{62}$$

We can then find the eigenvalues of A given the eigenvalues μ_i of B from

$$\lambda_i = 1 - \mu_i. \tag{63}$$

The elements of B can be written as

$$b_{i,j} = 2\delta_{i,j} - \delta_{i+1,j} - \delta_{i-1,j} \tag{64}$$

so the eigenvalue problem can be written as

$$(Bx)_i = 2x_i - x_{i+1} - x_{i-1} = \mu_i x_i \tag{65}$$

By saying that x can be rewritten as a linear combination of the basis $(\sin(\theta), \sin(2\theta)...)$ where $\theta = \frac{l\pi}{n} + 1$ we get by inserting into the equation that $\mu x_i = 2(1 - \cos(\theta))x_i$ which gives the eigenvalues $\mu_i = 2 - 2cos(\theta)$. Since we want $\rho(A) < 1$ we need that

$$-1 < 1 - \alpha(2 - 2cos(\theta)) < 1 \tag{66}$$

so we see that the condition is that $\alpha < 1 - \cos(\theta)^{-1}$. From this we get that the stability criterion is that

$$\alpha \leq 1/2. \tag{67}$$

This can be a problem when simulating a system over time. We often use $\Delta x = 0.01$ which means that we have to have $\Delta t < 0.5\Delta x^2 = 0.00005$. If we want to simulate over a long time period, this method becomes extremely slow. We will therefore study other methods that does not have the same restriction to the choice of grid.

2.4.3 Forward euler in 2D

The forward Euler method can easily be extended to solve problems in two spatial dimensions. The same approximations of the derivatives can again be used and by inserting into the equation we get that

$$\frac{u^{l+1}_{i,j} - u^{l}_{i,j}}{\Delta t} = \frac{u^{l}_{i+1,j} - 2u^{l}_{i,j} + u^{l}_{i-1,j}}{\Delta x^2} + \frac{u^{l}_{i,j+1} - 2u^{l}_{i,j} + u^{l}_{i,j-1}}{\Delta x^2}. \tag{68}$$

By rearranging the equation we get that

$$u^{l+1}_{i,j} = u^{l}_{i,j} + \alpha(u^{l}_{i+1,j} - 2u^{l}_{i,j} + u^{l}_{i-1,j} + u^{l}_{i,j+1} - 2u^{l}_{i,j} + u^{l}_{i,j-1}). \tag{69}$$

The truncation errors is $O(\Delta x^2)$, $O(\Delta y^2)$ and $O(\Delta t)$. This means that the error decreases fast if we decrease the step length in spatial dimensions. However, we would then have to decrease the step length in time even more to fulfil the stability criteria.

The forward step can be implemented in python with the following code snippet

```
u[1:-1, 1:-1] = u[1:-1, 1:-1] + dt*( (u[0:-2, 1:-1] -2*u[1:-1,1:-1] + u[2:,1:-1])/(dx**2) + \
                                        (u[1:-1, 0:-2] -2*u[1:-1,1:-1] +
                                           u[1:-1,2:])/(dy**2) )
```

2.4.4 Backward Euler

We have now looked at one explicit scheme for solving the diffusion equation. The method is easy to implement and gives good results. However, the stability conditions is a drawback. Increasing the number of grid-points in the spatial dimension would lead to an requirement of very low step lengths in time. We will now introduce the Backward Euler which fixes this problem.

The Backward Euler method is an implicit scheme. An implicit scheme is a method where we need to solve a set of linear equations for finding the values at the next time-step. Now we have to solve a system of equations for finding the solution. The main benefit of the backward euler is the stability conditions. It can be shown to converge for all combinations of step lengths. This will be shown later.

In the backward Euler we use change the approximations of the derivatives. We now use

$$u_t \approx \frac{u_i^j - u_i^{j-1}}{\Delta t} \tag{70}$$

and

$$u_{xx} \approx \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} \tag{71}$$

with the same local apporixmation errors as for the forward Euler scheme. Similarly as what we did for the forward scheme, we can find an recursive expression for the time development of u. We insert into the original diffusion equation and get that

$$\frac{u_i^j - u_i^{j-1}}{\Delta t} = \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2}. \tag{72}$$

By rearranging we get

$$u_i^{j-1} = u_i^j - \frac{\Delta t}{\Delta x^2}\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j\right) = -\alpha u_{i+1}^j + (1+2\alpha)u_i^j - \alpha u_{i-1}^j \tag{73}$$

where $\alpha = \frac{\Delta t}{\Delta x^2}$. This problem can be reformulated as a matrix equation. We will see that the matrix is tridiagonal, and we can use effective methods for solving such linear system. For a thorough analysis of tridiagonal solver, see [1]. By letting A be the tridiagonal matrix

$$A = \begin{bmatrix} 1+2\alpha & -\alpha & 0 & 0 & \dots & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & 0 & \dots & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & -\alpha & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & -\alpha & 1+2\alpha & -\alpha \\ 0 & 0 & \dots & \dots & 0 & -\alpha & 1+2\alpha \end{bmatrix} \tag{74}$$

we see that the equation can be rewritten as

$$\boldsymbol{u}^{j-1} = A\boldsymbol{u}^j \rightarrow \boldsymbol{u}^j = A^{-1}\boldsymbol{u}^{j-1} \tag{75}$$

Solving this linear system of equation will lead to the solution for time step j given the solution in time step j-1. Since we now can use a tridiagonal solver, we are able to speed up the computations.

The main part of the Backward Euler method is the step from one time-step to the next. Since A is a tridiagonal matrix we can use effective methods for solving it. In python we can use the `scipy.sparse` library to effectively step to the next time-step. This can be done by the following code snippet in python

```python
A = sc.sparse.csc_matrix(A)
for i in range(1, num_t_values):
    u[i,:] = spsolve(A , u[i-1, :] )
    u[i,-1] = b(1)
    u[i,0] = a(1)
return u
```

where `spsolve` is a funtion in the `scipy.sparse` library.

2.4.5 Stability and truncation error of the backward Euler scheme

Finding the stability criteria for the backward Euler method is done similarly to what we saw for the forward Euler method. We note that the eigenvalues of A is given by

$$\lambda_i = 1 + \alpha(2 - 2\cos(\theta)) \tag{76}$$

which always is larger than one. The eigenvalues of the inverse is therefore in the interval [0,1) which means that $\rho(A) > 1$. The implicit shceme is then stable for all combinations of Δx and Δt.

2.4.6 Crank-Nicolson

The last method we will study is called the Crank-Nicolson scheme. It is a mix between the Forward Euler and the Backward Euler. We can obtain this by making and weighted averege of the two by using a parameter θ which controls which we want to influence the results the most. Introducing

$$\frac{u_i^j - u_i^{j-1}}{\Delta t} = \theta \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{\Delta x^2} + (1-\theta)\frac{u_{i+1}^{j-1} - 2u_i^{j-1} + u_{i-1}^{j-1}}{\Delta x^2} \tag{77}$$

we see that by setting $\theta = 0$ or $\theta = 1$ we get the forward Euler and the backward Euler, respectively. By setting $\theta = 0.5$, we get the Crank Nicolson scheme. For a thorough derivation of the Crank Nicolson scheme, see [1].

As we did before, we can rearrange the expression and can write equation 77 as a matrix equation

$$(2I + \alpha B)V^j = (2I - \alpha B)V^{j-1} \tag{78}$$

where B is

$$\begin{bmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & ... & 0 \\ ... & ... & ... & ... & ... & ... \\ 0 & 0 & ... & 0 & -1 & 2 \end{bmatrix}. \tag{79}$$

When implementing the code for the Crank Nicolson scheme we can utilize that we are dealing with tridiagonal matrices. The equation can be solved for V^j given V^{j-1} by first calculating an intermediate vector. This means that we efficiently can solve the problem in two steps. The first step is to calculate the right hand side and get

$$(2I - \alpha B)V^{j-1} = \boldsymbol{b^{j-1}}. \tag{80}$$

This can efficiently be implemented in python using the same library as for Backward Euler. After that, we use a tridiagonal solver to solve

$$(2I + \alpha B)V^j = \boldsymbol{b}^{j-1} \tag{81}$$

which results in the solution for the next time step.

The steps can be implemented in python with the following code snippet

```
temp1 = (2*np.eye(num_x_values) + alpha*A)
temp1[0,0] = temp1[-1,-1] = 1
temp1[0,1] = temp1[-1,-2] = 0
temp1 = sc.sparse.csc_matrix(temp1)

temp2 = 2*np.eye(num_x_values) - alpha*A
temp2[0,0] = temp2[-1,-1] = 1
temp2[0,1] = temp2[-1,-2] = 0
temp2 = sc.sparse.csr_matrix(temp2)

for i in range(1, num_t_values):
    forward_step = temp2.dot(u[i-1,:])
    u[i,:] = spsolve(temp1,forward_step)
    u[i,-1] = b(1)
    u[i,0] = a(1)

return u
```

where we again have used th efficient `scipy.sparse` python library to speed up calculations with tridiagonal matrices.

2.4.7 Stability of the Crank Nicolson scheme

Again, we use the same technique for finding the stability limit of the Crank Nicolson scheme. By noting that we can write the scheme as

$$V^j = (2I + \alpha B)^{-1}(2I - \alpha B)V^{j-1} \tag{82}$$

we see that we need

$$\rho\big[(2I + \alpha B)^{-1}(2I - \alpha B)\big] < 1. \tag{83}$$

From the definition of the spectral function it follows that

$$|((2 + \alpha\mu_i)^{-1}(2 - \alpha\mu_i))| < 1. \tag{84}$$

Scheme	Truncation error	Stability criterion
CN	$O(\Delta t)$ and $O(\Delta x^2)$	No criteria
FE	$O(\Delta t)$ and $O(\Delta x)$	$\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}$
BE	$O(\Delta t)$ and $O(\Delta x)$	No criteria

Table 1: Stability critera and truncation errors for the three different finite difference schemes.

From the derivations for the forward Euler and the backward Euler schemes we know that $\mu_i = 2 - 2\cos(\theta)$. We see that $\mu_i \in (0, 4)$. Since $(2 - \alpha\mu_i) < (2 + \alpha\mu_i)$ it follows that that it is stable for all combinations of Δt and Δx.

In table 1 the different feautures of the schemes are summarized. From this we would expect the CN to get the most accurate results. We will study this later.

2.5 Temperature distribution in the lithosphere

We will now focus on a practical application of the diffusion equation. The equation can be scaled so that we can use approximately the same numerical methods as we have already developed.

The problem we will look at is simulating the temperature in the lithosphere from 1Gy ago until today. It has been shown that there was an active subduction zone on the western coast of Norway 1Gy ago. This caused the release of water from the oceanic lithosphere. Even though most of the chemical components that contain radioactive elements in the water raised to the surface, some of it was trapped in the mantle. This entrapment of radioactive elements led to production of heat, and we would expect the part of the lithosphere to be warmer than normal.

The equation we will use to describe the problem is an version of the diffusion equation,

$$\nabla(k\nabla T) + Q = \rho c_p \frac{dT}{dt} \tag{85}$$

where T is the temperature as a function of depth and time, $T = T(y, t)$. Q is the heat-production, c_p is the specific heat capacity, ρ is the density and k is the thermal conductivity. We assume that $\rho = 3.5 \cdot 10^3 Kg/m^3$, $k = 2.5W/m/\circ C$ and $c_p = 1000J/kg/°C^{-1}$ and constant over the whole lithosphere. Before we have only looked at the situation with no heat-production in the domain. With Q added, we can expect the solution to differ a bit from what we saw before.

By introducing boundary conditions and initial conditions we can solve the problem numerically. At the surface the temperature is $8°C$ and 120km below the surface the temperature is set to $1300°C$. The reference system can be changed, and I will set y=0 to be 120 km below the surface while doing the calculations. The boundary conditions can then be formalized by

$$T(120, t) = 8°C \tag{86}$$

and

$$T(0, t) = 1300°C. \tag{87}$$

Even though we assume that c_p, k and ρ are constants, the heat production caused by the radioactive elements will be modelled as a piecewise constant function. We define

$$Q(y) = \begin{cases} 1.4\mu W/m^3 & 100 \leq y \leq 120 \\ 0.35\mu W/m^3 & 80 \leq y < 100 \\ 0.05\mu W/m^3 & 0 \leq y < 80 \end{cases} \tag{88}$$

This is interpreted as the heat production is higher closer to the surface.

We will see that we can scale the equations and then be able to use the same methods as before for solving them.

The analysis will be split into to parts. The first part looks at the temperature distribution before radioactive enrichment. We will find the steady state and use this as initial conditions for the analysis from 1Gy ago until today. We will then study two cases with radioactive enrichment. The first one is the case where Q is as in equation 88. With the steady state without radioactive elements as the initial state, we will simulate the temperature until today. In the second case we will assume that in addition to Q as in the previous case, we also have an additional heat source of $0.5\mu W/m^3$ in the mantle.

For the following analysis we assume that the distribution is independent of x, so we can look at the problem as a function of y and t, where y is depth. The equation we will be working with is then

$$\frac{\partial^2 T}{\partial y^2} + Q = \rho c_p \frac{\partial T}{\partial t}. \tag{89}$$

Scaling the equation will ease the analysis. It will also make it possible to use the Forward Euler scheme for solving the problem numerically. The first step is to introduce dimensionless variables. We can let $y^* = y/y_c$ where y_c is a characteristic length scale, $T^* = (T - T_0)/(T_1 - T_0)$ where $T_1 = 8$ and $T_0 = 1300$, and $t^* = t/t_c$ where t_c is a characteristic time. The characteristic sizes can be chosen as we want. However, for this particular problem it is appropriate to choose the characteristic time as 1Gy and the characteristic length as 120km. We convert til seconds and meters and get that

$$t_c = 10^9 * 365 * 24 * 60 * 60 \text{ s} \tag{90}$$

and

$$y_c = 120 * 10^3 \text{ m}. \tag{91}$$

Then t^*, T^* and y^* are dimensionless. Inserting this into equation 85 results in

$$k\frac{\partial^2[T^*(T_1 - T_0) + T_0]}{\partial(y^* y_c)^2} + Q = \rho c_p \frac{\partial[T^*(T_1 - T_0) + T_0]}{\partial t^* t_c}. \tag{92}$$

Rearranging gives the scaled equation

$$\frac{k t_c}{\rho c_p y_c^2}\frac{\partial^2 T^*}{(\partial y^*)^2} + \frac{t_c}{\rho c_p (T_1 - T_0)} Q = \frac{\partial T^*}{\partial t^*} \tag{93}$$

and by introducing constants C_1 and C_2 we get that

$$C_1 \frac{\partial^2 T^*}{(\partial y^*)^2} + C_2 Q = \frac{\partial T^*}{\partial t^*} \tag{94}$$

with the boundary conditions

$$T^*(0, t) = 1 \tag{95}$$

and

$$T^*(1, t) = 0 \tag{96}$$

for $t^* > 0$, $y^* \in [0, 1]$ and $T^* \in [0, 1]$. Again, we can transform the variables back to unscaled versions by using that $T = T^*(T_1 - T_0) + T_0$ and $y = y^* y_c$. This means that $t^* = 1$ corresponds to $t = 1Gy$. All the terms in equation 94 are now dimensionless.

The forward Euler method is used for solving the equation numerically. We can use the same techniques as we used for the simple case without a heat-term. We discretize the domain and uses approximations of the derivatives and inserts into the equation. Doing this leads to

$$C_1 \frac{T_{i-1}^j - 2T_i^j + T_{i+1}^j}{\Delta x^2} + C_2 Q_i^j = \frac{T_i^{i+1} - T_i^j}{\Delta t} \tag{97}$$

where I have dropped the marker to simplify the expression. Rearranging gives

$$T_i^{i+1} = \Delta t C_1 \frac{T_{i-1}^j - 2T_i^j + T_{i+1}^j}{\Delta x^2} + \Delta t C_2 Q_i^j + T_i^j, \tag{98}$$

where $Q_i^j = Q(y_i^j)$. This can be implemented in python in a similar way as for the simple case. For details, see the jupyter notebook `5g.ipynb`. We see that this can be solved by updating the temperature in the next time step given the temperature in the current time step.

When the scaled solution T^* is found we can transform back to the original scale with the formula

$$T = T^*(T_1 - T_0) + T_0. \tag{99}$$

The length scale and time scale is also converted back by

$$y = y^* y_c \text{ and } t = t^* t_c. \tag{100}$$

Before radioactive enrichment there are was heat generated, so Q = 0. The system was then in steady state which means that the time derivative vanishes. Inserting this into equation 85 gives

$$\frac{d^2 T}{dy^2} = 0. \tag{101}$$

The analytical solution for this can easily be obtained by integrating twice, and we get that

$$T(y,t) = Ay + B. \tag{102}$$

This means that the temperature distribution when the system is in a steady state with no radioactive matter is a linear function between the end-point temperatures. By using the boundary conditions we see that

$$T(0,t) = B = 1300 \tag{103}$$

and

$$T(120,t) = 120A + 1300 = 8 \rightarrow A = -1292/120 \tag{104}$$

so that

$$T(y,t) = -1292/120y + 1300. \tag{105}$$

where $y \in [0, 120]$ and $T \in [8, 1300]$. This can also be solved using the scaled equation. Solving this similarly to what was done for the unscaled version gives

$$T^*(y^*,t) = 1 - y^*. \tag{106}$$

for $y^* \in [0, 1]$. We can then go back to unscaled variables by using that

$$T = T^*(T_1 - T_0) + T_0. \tag{107}$$

and

$$y = y^* y_c. \tag{108}$$

We see that equation 101 is the laplace function in one dimension. This is the same as the diffusion equation when we let $t \rightarrow \infty$. On a computer we cant do simulations until infinity. To solve this numerically we can use the forward Euler method and do simulation until the system reach a steady state.

After radioactive enrichment, heat is produced in the lithosphere. We will first look at the steady state when Q is as described in equation 88. After that we will assume that the lithosphere above the slab gets an additional enrichment of radioactive elements that produces an heat of $0.5\mu W/m^3$ in the mantle in the range $y \in [0, 80]$. 40% of the additional heat are produced by U, 40% are produced by Th and the remaining 20% by K. They have halflives 4.47 Gy, 14.0 Gy and 1.25 Gy respectively. The additional heat can then be modelled as $Q_{\text{additional}} = 0.4 * 0.5e^{-t/4.47} + 0.4 * 0.5e^{-t/14.0} + 0.2 * 0.5e^{-t/1.25}$ for $y \in [0, 80]$ and zero everywhere else.

2.6 Comparing the methods

The comparison will focus on computational time and accuracy. For time we will simply measure the time it takes for the algorithm to compute the solution for all time-steps up to a given time t. To compare the accuracy of the schemes there are different methods that can be used. Since we know the exact solution of the problem we can easily obtain accurate error measures. One could for example extract the maximum difference between the numerical solution and the analytical solution. However, I do not think this is the best way to compare the methods. It is more interesting get an idea about how the method performs over the whole spatial interval. For the analysis I will therefore study the mean squared error for each time step. The MSE for a time t is given by

$$\frac{1}{N_x}\sum_{i=1}^{N_x}(u_{\text{numerical}} - u_{\text{analytical}})^2. \tag{109}$$

where N_x is the number of grid points in x-direction. For a qualitative analysis of how the methods compare we will also plot the error for each scheme. By plotting

$$\text{error} = u_{\text{analytic}} - u_{\text{numerical}} \tag{110}$$

we can study from which side the numerical solution approaches the exact solution.

3 IMPLEMENTATION

All code used in the project can be found at the GitHub repository: `https://github.com/MartinKHovden/CompPhys_project5`. Details about how the repository is structured are presented in this section.

The main part of the code is in the file `diffusion_solvers.py`. This is a library that contains functions for Forward Euler in one and two dimension, Backward Euler, and Crank Nicholson. In addition it contains tests that can be run to check if the different implementations are correct. Since the speed of python out of the box is quite bad, I have used numpy, numba and scipy to increase the performance. Especially in for loops we can increase the speed by simply putting

```
@jit
def function_name():
    ...
```

over the function we want to optimize. It does not work for all function, especially when a function calls functionality from another package. This is the case for Backward Euler and Crank Nicolson. For Forward Euler in one and two dimensions I have made a separate functions for the forward step and then used numba's jit to speed up the loop. For Backward Euler and Crank Nicolson we solve tridiagonal systems. It would be a waste of time to not utilize this to speed up the program. This is done by using the scipy.sparse library as described in the theory part. All the one dimensional solvers takes the same input. Therefore it can easily be experimented with different solvers on varying problems. The results from the two dimensional implementations is in the file `5e.py`. Parameters can be set by the user to test for different grids. The jupyter notebook `5c.py` contains scripts that uses the function to solve part c of the project.

The code for solving the geological part of the project is in the jupyter notebook `5g.ipynb`. The needed functions are also in the file `5g.py`. In the notebook an updated version of the Forward Euler scheme for solving the problem with the heat source is introduced. The notebook also produces results for the different cases we want to study. The implementation follows the description in the theory.

Plots for different problem are presented in the report, and are also located in the folder "Plots". All figures have titles corresponding to the problem they solve, and the filename tells what parameters are used and which problem it solves.

To run tests on the different solver, you simply run the command

```
C:.../pytest diffusion_solvers.py.
```

This will do some basic tests on the schemes to see if the boundaries are maintained throughout the simulation and if it reaches certain analytical known solutions.

4 RESULTS AND DISCUSSION

4.1 Diffusion equation in one dimension

We start by looking at the diffusion equation in one dimension using the three different finite difference schemes discussed in the theory part and compare the schemes performance. We also study how the solutions behaves for different t-values where the curvature of the function is different from each other.

We will start the analysis by looking at t=0.02. At this time-step the temperatures profile is still significantly curved. In figure 1 the numerical solutions for the three methods are plotted together with the analytical solution. As the temperature started out as zero in the hole domain, except for the boundary at x=1, we see that almost half of the domain still have temperature approximately equal to zero. From x=0.5 and up, the temperature increases rapidly. In the figure we have used $\Delta t = 0.00001$ and $\Delta x = 0.01$ for doing the numerical calculations. The step-lengths are chosen to be well within the stability limit of the forward Euler method. Later, we will study what happens if we are closer to the limit or crosses the limit. We see from the left part of the figure that all of the methods seems to approximate the solution well. Looking at the right part of the plot shows the methods behave quite similarly in the lower end of the domain. However, it seems like the backward Euler scheme have more trouble with approximating the solution close to the boundary at x=1. The forward Euler scheme and the Crank Nicolson scheme both have their maximum error around the middle of the domain. This is where the derivative of the function changes most, so this would be expected. From x=0 to x= 0.5 and from x=0.9 to x=1 the function is almost linear. This is where the errors of the FE and CN are smallest.

In figure 2 we see that by making Δt and Δx smaller, we get a smaller error. The shape of the error functions is approximately the same but the absolute values are smaller. The error are of order 10^{-5} compared to 10^{-3} for the larger step lengths. Again, we see that the BE is the worst, while FE and CN have similar error.

The main problem with FE is the stability limit. For $\Delta x = 0.01$ we would need $\Delta t \leq$ 5e-5. This becomes a problem if one would want to solve for high t-values. Scaling would then be necessary as we will see when looking at the temperature in the lithosphere. It is interesting to see how the results get by choosing the step lengths at the limit. A simulation ran with those parameters are shown in figure 3. We see that at the stability limit, the solution starts to behave very bad and zig-zags around the analytical solution. However, the absolute error isn't very high yet. By increasing Δt to 6e-5 the results gets much worse as illustrated in figure 4. The error blows up to an absolute value of over 1e45 and the FE method is useless for theese kind of grids. Both CN and BE are stable because they have no stability limit as discussed in the theory part.

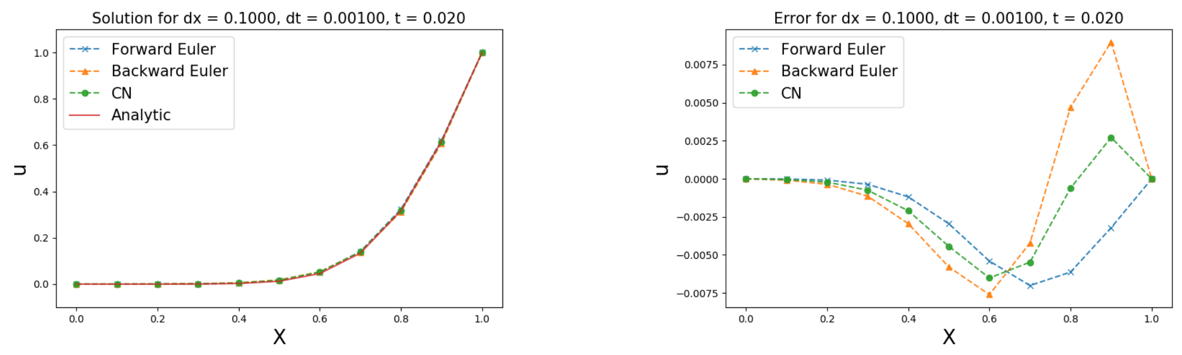

Figure 1: Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.02, $\Delta t = 0.001$ and $\Delta x = 0.1$.

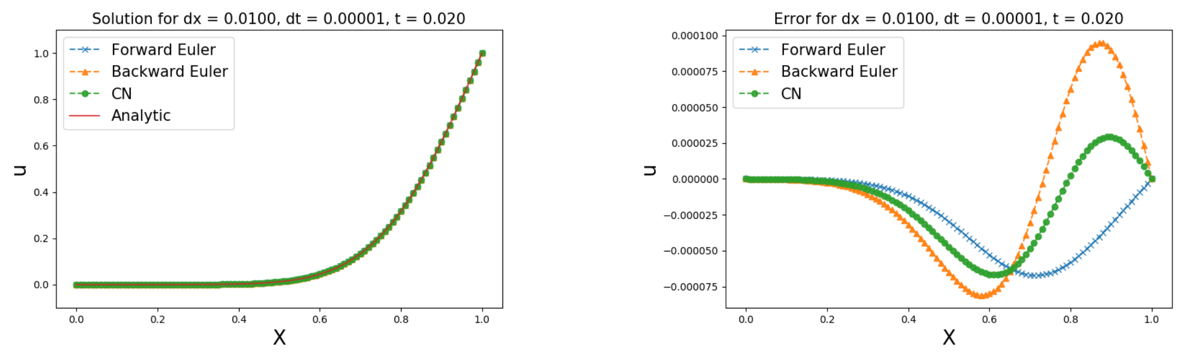

Figure 2: Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.02, $\Delta t = 0.00001$ and $\Delta x = 0.01$.

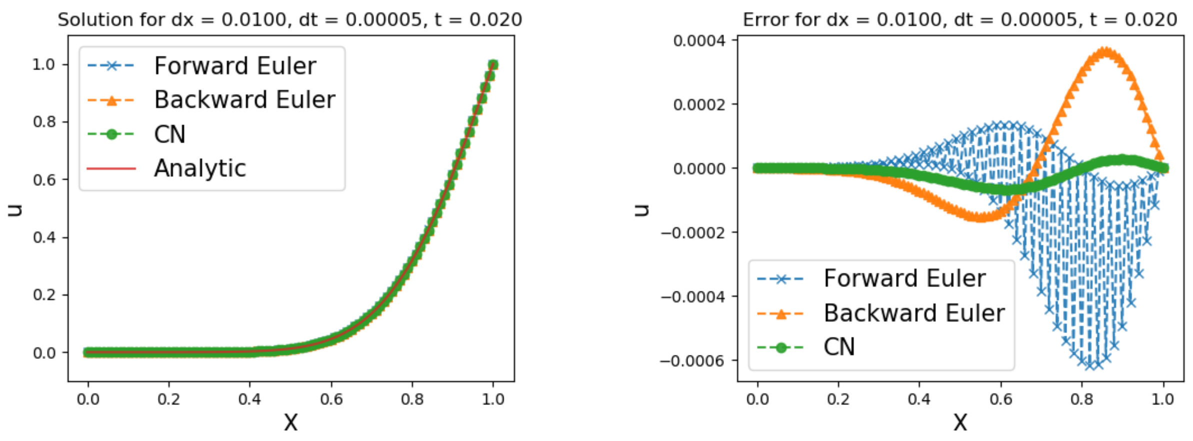

Figure 3: Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.02, $\Delta t = 0.00005$ and $\Delta x = 0.01$.

Looking at the results for t=0.25 in figure 5 and 6 we see that the solution is close to linear. It has almost reached the steady state of the system which is the linear function between the two boundary conditions. In the left plot we see that the approximations looks good. Again, by looking at the right part of the figures, the error varies a lot (relatively) from method to method. The error of FE is much smaller than for CN and BE. It looks like FE is better at approximating the solution when the solution is linear.

We note that the error of CN is in the middle of FE and BE for almost all x-values. We remember from the derivation of CN that it is a mix between the two, so this would be natural.

In table 2 and 3 the results are summarized in a compact way and the results can be studied in more detail. Again we note that FE is better the closer we get to a linear state, while the CN is slightly better than FE when the curve is more complex. The worst method with regards to accuracy is BE. When it comes to computational time, the difference varies with the step-lengths. We see that when $\Delta t = 0.001$ and $\Delta x = 0.1$, both BE and CN uses approximately 0.05 seconds, while FE uses approximately no time at all (below machine precision). When the grid is finer, the computational time of all the methods increase significantly. All of methods performs better for the linear state compared to the curved state. The superior performance of Crank Nicolson comes from the truncation error discussed in the theory.

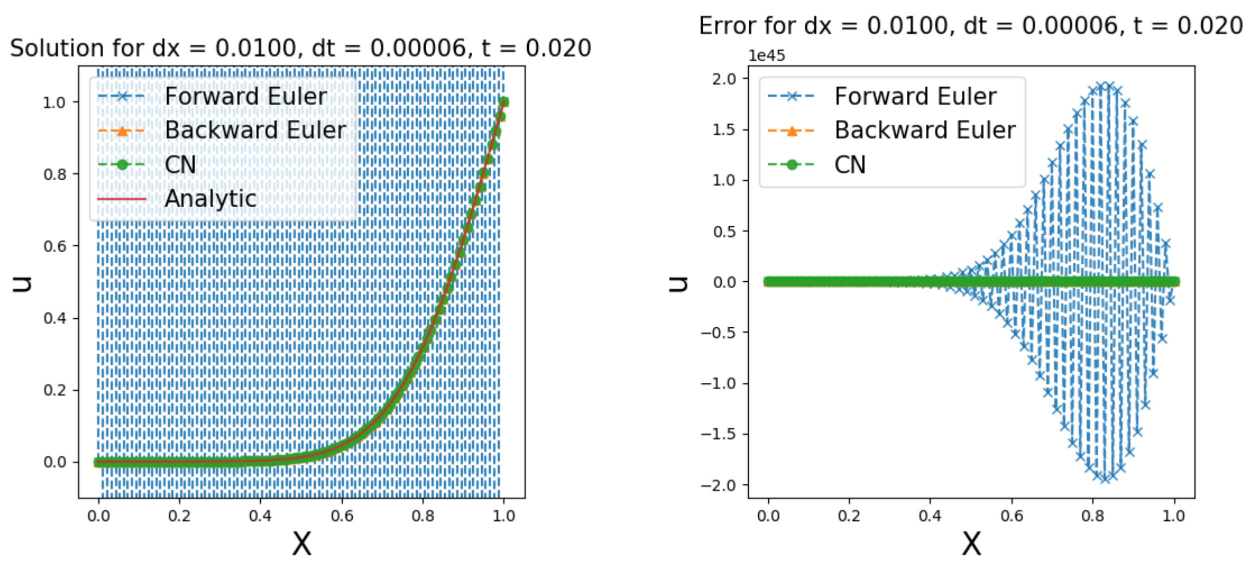

Figure 4: Illustration of a breach of the stability limit for forward Euler. Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.02, $\Delta t = 0.00006$ and $\Delta x = 0.01$.

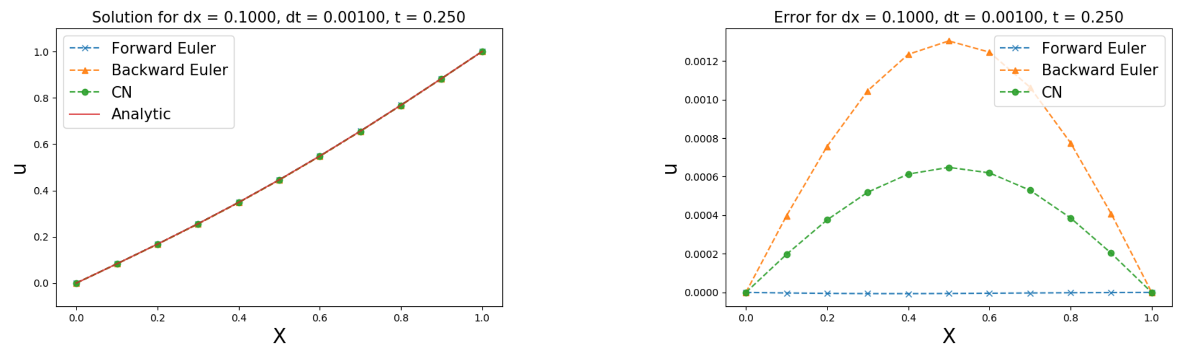

Figure 5: Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.25, $\Delta t = 0.001$ and $\Delta x = 0.1$.

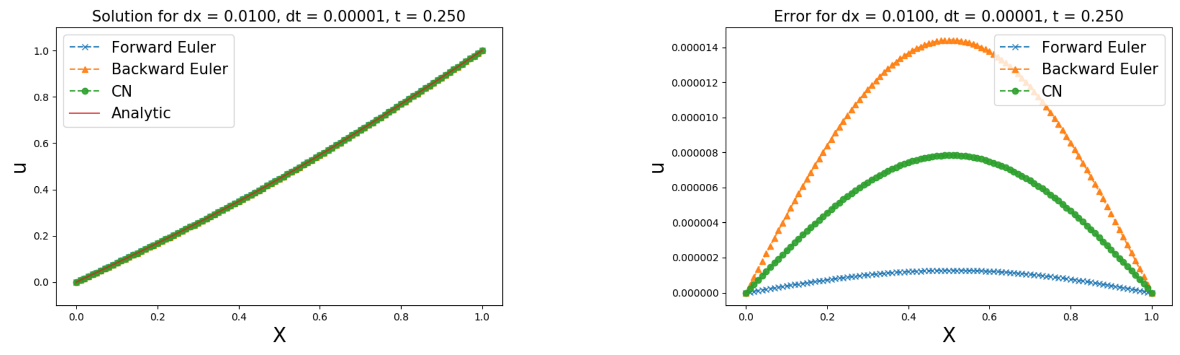

Figure 6: Left: Analytic solution plotted with the numerical solution for Forward Euler, Backward Euler and Crank Nicolson. Right : Error is plotted for the three methods. t = 0.25, $\Delta t = 1e-5$ and $\Delta x = 0.01$.

Method	Time [s]	MSE t = 0.02	MSE t = 0.25
Forward Euler	0.0073792	1.265e-07	8.119e-11
Backward Euler	9.38624	2.474e-07	1.034e-08
Crank Nicolson	8.859143	1.045e-07	3.080e-09

Table 2: Results for $\Delta t = 0.00001$ and $\Delta x = 0.01$. Time is for calculating solution up to t = 1.

Method	Time [s]	MSE t = 0.02	MSE t = 0.25
Forward Euler	0.000	0.00013	1.65e-10
Backward Euler	0.049	0.00033	8.49e-06
Crank Nicolson	0.058	0.000104	2.10e-06

Table 3: Results for $\Delta t = 0.001$ and $\Delta x = 0.1$. Time is for calculating solution up to t = 1.

4.2 Diffusion equation in two dimensions

We can now study the diffusion equation in two dimensions using the FE scheme. As described in the theory, we set the boundary conditions to zero and the intital condition to

$$u(x, y, 0) = \sin(\pi x)\sin(\pi y). \tag{111}$$

As we did for the one dimensional case the solution is studied for two different times, t=0.002 and t = 0.3. One where the solution is almost linear, and one where it is still curved. We start by studying the solution for t=0.002. In figure 7 and 8 the solution and error is plotted for two different grids. As we saw for the one dimensional case, the error is a lot smaller for the finer grid. This means that we can increase the accuracy by making the step-lengths smaller. Looking at the results for t=0.3 we see that the solution is

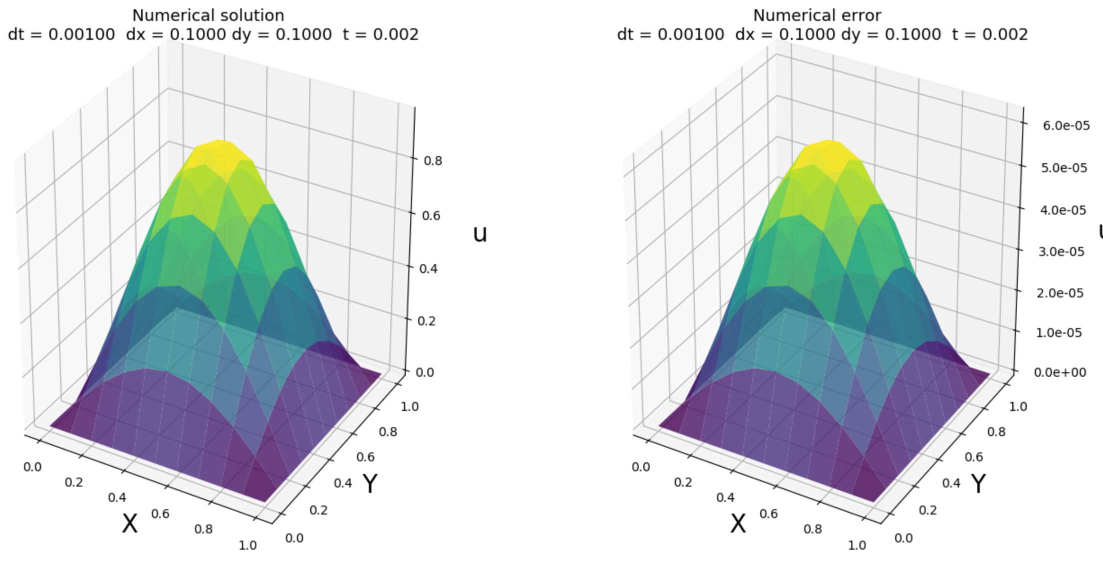

Figure 7: Left: Numerical solution using the forward Euler method in two dimensions for solving the diffusion equation. Right: Numerical error compared to the analytical results. $\Delta x = \Delta y = 0.1$, $\Delta t = 0.005$ and t = 0.002.

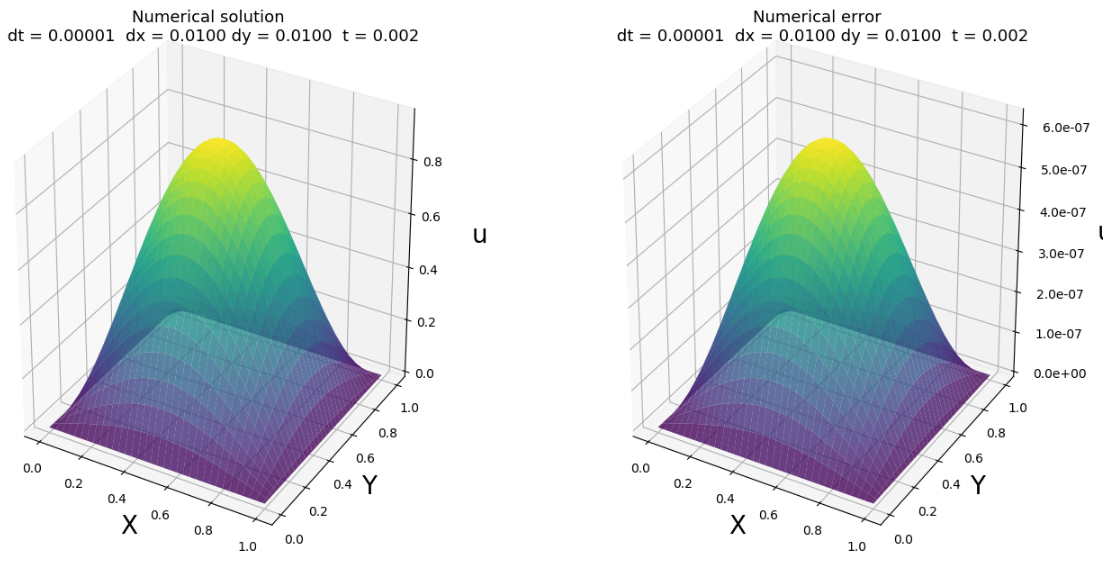

Figure 8: Left: Numerical solution using the forward Euler method in two dimensions for solving the diffusion equation. Right: Numerical error compared to the analytical results. $\Delta x = \Delta y = 0.01$, $\Delta t = 0.00001$ and t = 0.002.

almost flat. Note that the error is on different sides of the analytical solution when changing the grid. By looking at the plots we see that the absolute error is approximately half of the error for the previous case. When the results are stable, they are also accurate. However, we need to be careful when choosing the grid also in two dimensions. Again, we come over the same problem as for in one dimension. The stability of the forward Euler scheme. In figure 11 we see that the solution blows up for some combinations of step lengths. I am not sure about the exact stability conditions in two dimensions, but we see that by choosing Δt according to the stability criteria for the one dimensional case, we get a blown up solution.

4.3 Temperature distribution in the lithosphere

The code for generating the plots for the different parts of the analysis is in the jupyter notebook `5g.ipynb`. The numerical solutions are found using $\Delta y^* = 0.02$ and $\Delta t^* = 0.00005$ and the forward Euler method on

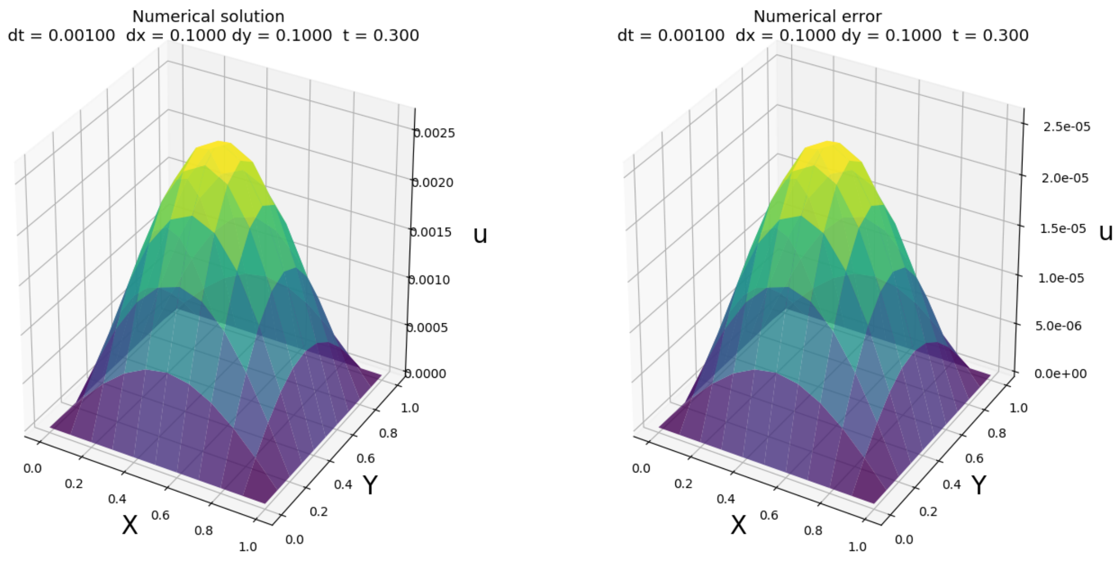

Figure 9: Left: Numerical solution using the forward Euler method in two dimensions for solving the diffusion equation. Right: Numerical error compared to the analytical results. $\Delta x = \Delta y = 0.1$, $\Delta t = 0.001$ and t = 0.3.

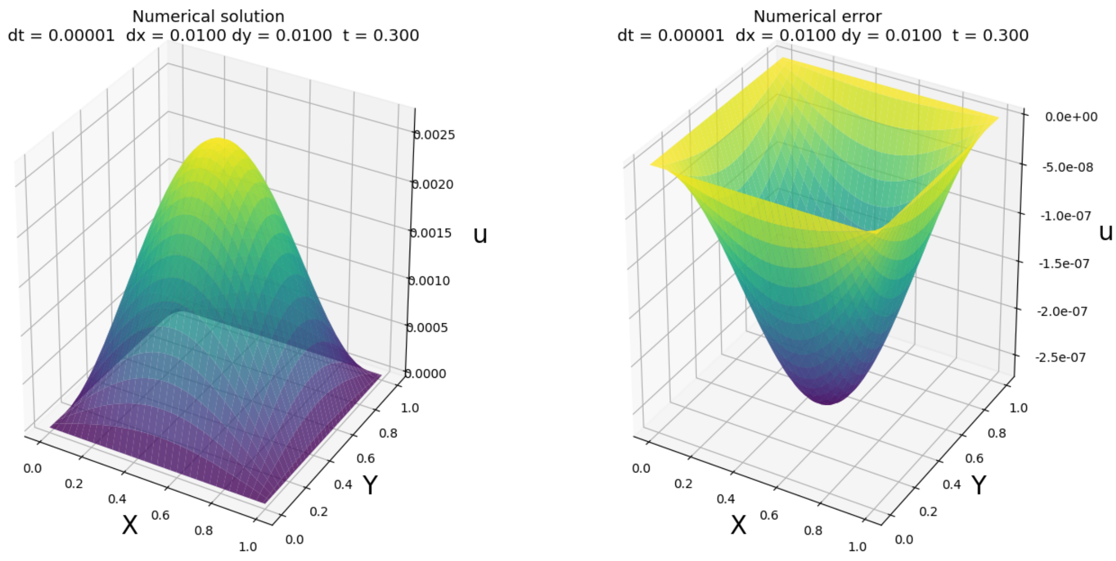

Figure 10: Left: Numerical solution using the forward Euler method in two dimensions for solving the diffusion equation. Right: Numerical error compared to the analytical results. $\Delta x = \Delta y = 0.01$, $\Delta t = 0.00001$ and t = 0.3.

the scaled form as in equation 94. The forward Euler method is modified to take care of the heat term. This is explained in the theory part and in the comments and doc-strings in the code.

We will now look at the temperature distribution in the lithosphere. We use the scaled equation as described in the theory part. When the solution is found for the scaled problem, we can scale it back to the original scale. The up-scaled solution is the solutions shown in the following plots. In the legends in the plot, Gy are counted from 1Gy ago. This means that the initial state is at 0Gy and today is 1Gy. In addition, the reference system for y is changed back so that the surface is a y=0km and the bottom is at y = -120km.

The first step of the analysis is to simulate the steady state with no radioactive sources. The temperature before it reaches the steady state is unknown. However, this does not matter. As long as the boundaries are held constant and no heat is added to the system, we will reach the correct steady state by simulating for long enough. This is done by using the diffusion equation without Q. Then use a random initial state and simulate until the temperature reaches a steady state. This can be seen in figure 12. We see that the steady state is a linear function between the boundary conditions which is intuitively correct. In the figure we see that the numerical solution overlaps with the analytical solution with no noticeable

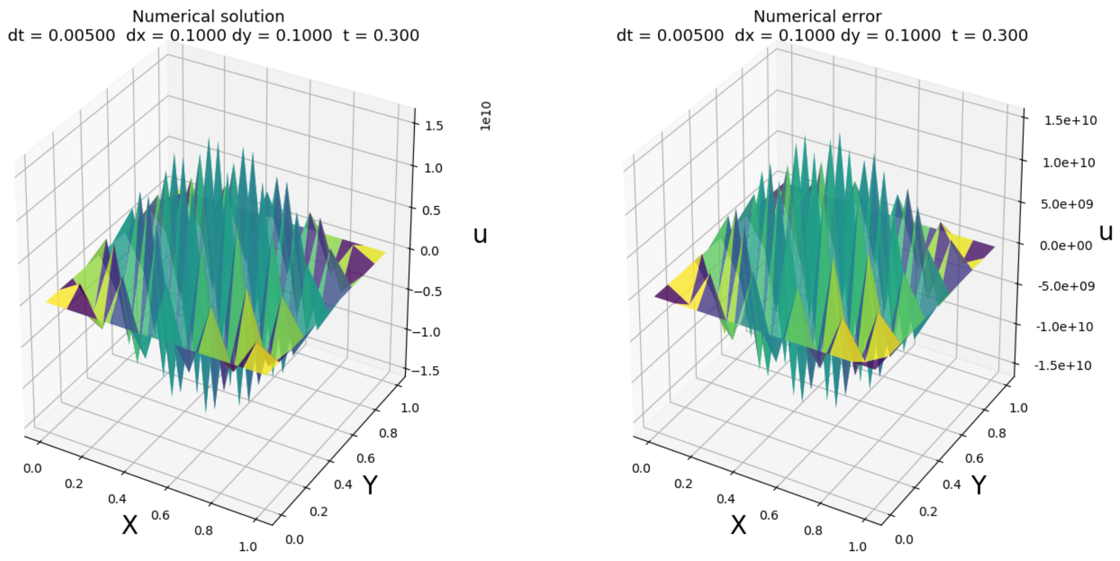

Figure 11: Left: Numerical solution using the forward Euler method in two dimensions for solving the diffusion equation. Right: Numerical error compared to the analytical results. $\Delta x = \Delta y = 0.1$, $\Delta t = 0.005$ and t = 0.3.

differences. The steady state is used as initial value for the rest of the analysis.

The next step in the analysis is to study the evolution of the temperature from 1Gy until today. We assume that 1Gy ago the temperature was in the steady state discussed previously. We also assume that 1Gy ago radioactive elements appears in the lithosphere. The radioactive elements works as a heat source. We would therefore assume that the temperature would be higher in the period after radioactive enrichment compared to before. Running simulations and plotting the solution gives the result in figure 13. We see that the results are what we expected and that we get a higher temperature values over the whole mantle. We note that the temperature increases quite fast in the beginning. The closer it gets to the steady state, the slower it converges. We see on the orange line, which is after 0.05Gy that we have reached around the halfway point towards the steady state. The change between 0.150Gy and 0.5Gy is significantly smaller. This is what we would expect from the discussion in the theory part. We also note that the absolute increase in temperature for each point is higher closer to the surface. This is because the heat production is highest in the region between the surface and 20km below. In the following layers the heat production decreases, and we see that the closer we get to the bottom, the smaller the absolute increase in temperature is.

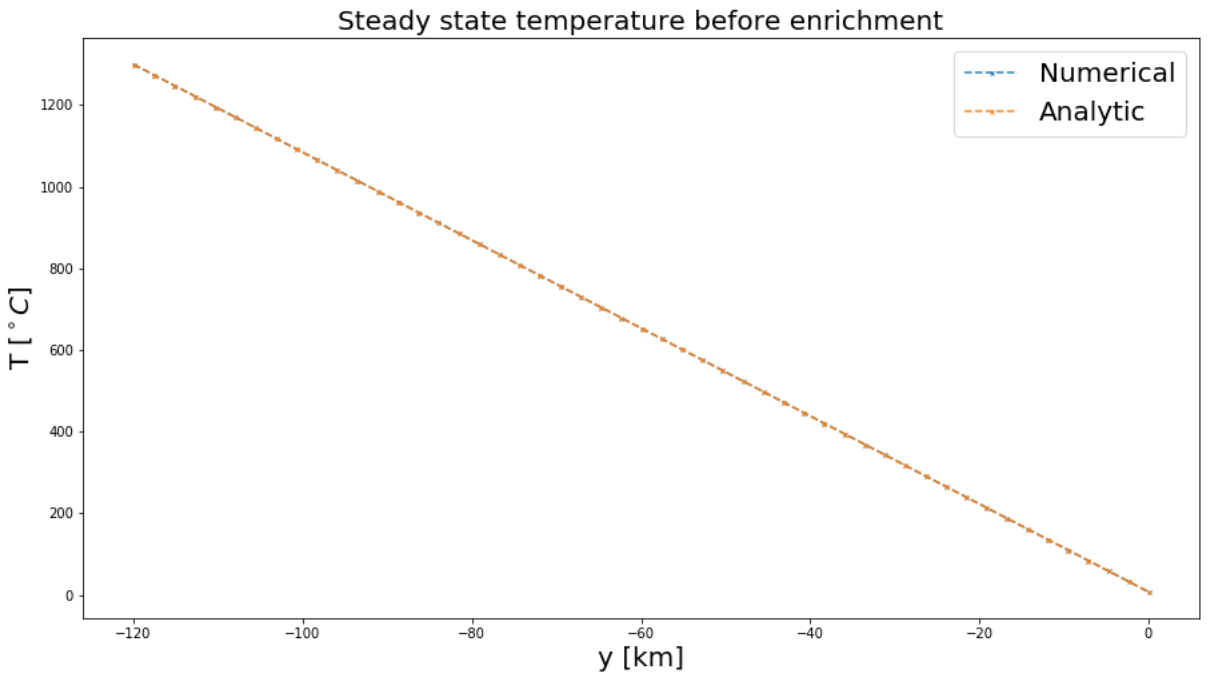

Figure 12: Steady state temperature before radioactive enrichment. Q = 0 and $\frac{dT}{dt} = 0$. Numerical solution calculated using forward Euler.

A more realistic case is obtained by adding another feature to the simulation. We assume that an additional heat source of $0.5\mu W/m^3$ is added to the mantle. The results are shown in figure 14. The behaviour is quite different from the previous case. Since the heat source starts out higher than the previous case in the mantle, we would assume that the temperature reaches a higher value at some point in time. However, since we also model the decay of the radioactive elements, the heat will decrease, and one could think that after a long time, the temperature will reach the same steady state as for the previous case. We will now look at the development of the temperature from 1Gy ago until today. As we can see in the figure the temperature increases fast in the beginning and then the speed decreases. As we discussed, the simulation shows that it reaches a higher temperature than the previous case. However, now the temperature decreases after reaching the maximum value. This is as expected.

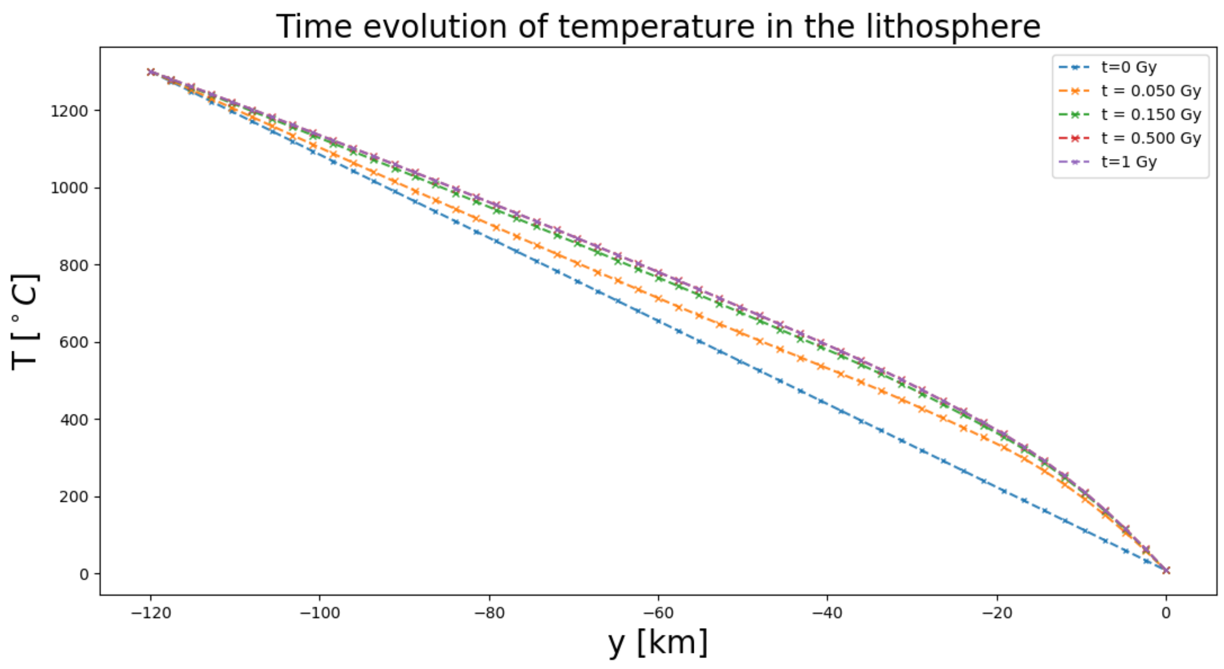

Figure 13: Temperature evolution from 1Gy ago until today, with Q as in equation ??

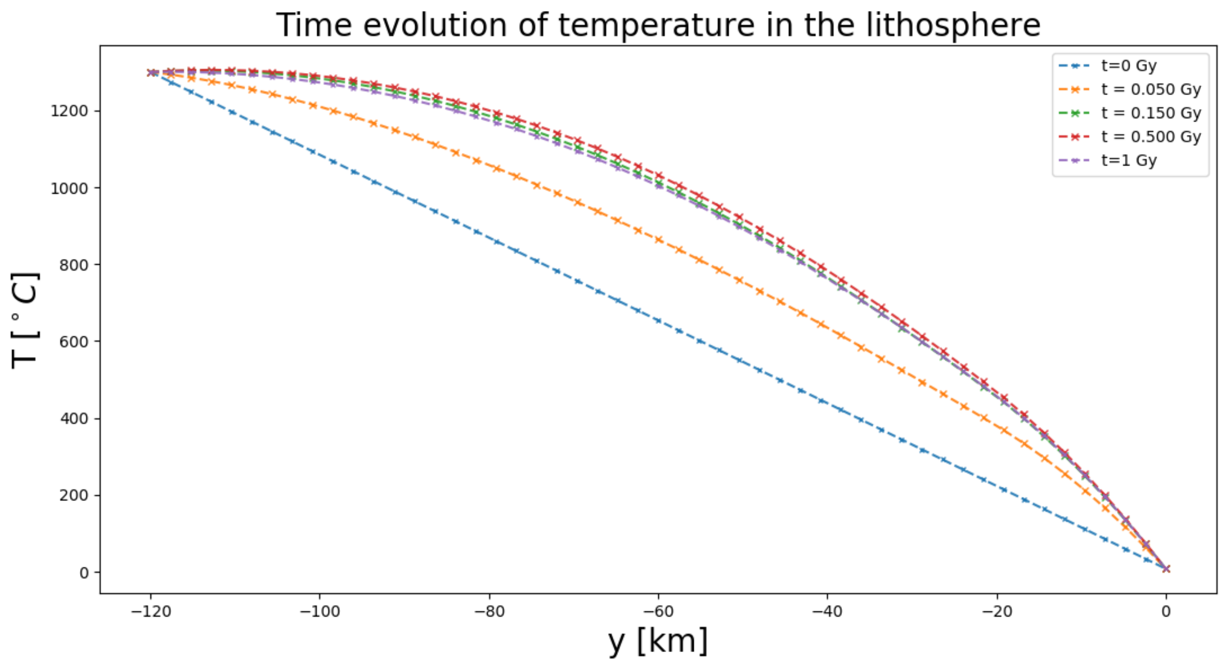

Figure 14: Temperature distribution in the lithosphere from 1Gy ago until today. Q as in equation 88 plus $0.5\mu W/m^3$ in the mantle for $y \in [-120, -40]$.

5 CONCLUSION

In this report we looked at various numerical methods for solving the diffusion equation in both one and two dimensions. The performance of the algorithms was compared for the different cases. For the one dimensional case we saw that all methods performed quite well. The main problem was the stability limit of the Forward Euler schemes. We saw that only slight breaches of the limit lead to a blown up solution. The other methods performed well for all α's. When we moved away from the limit, Forward Euler, Backward Euler and Crank Nicolson all gave acceptable results. There was however some differences in the MSE's.

Crank Nicolson gave the best MSE for all tests. For $\Delta t = 0.001$ and $\Delta x = 0.1$ the MSE of Crank Nicolson was 0.000104 for t = 0.02 and 2.10e-6 for t = 0.25. For the grid with $\Delta t = 0.00001$ and $\Delta x = 0.01$ the MSE's was 1.045e-7 for t = 0.02 and 3.08e-9 for t = 0.25. The main problem with Crank Nicolson was the slow completion time compared to Forward Euler. The results from Forward Euler was close to the ones from Crank Nicolson. I would argue that the time differences can make the Forward Euler method the best choice for some problems when we an afford to give up a small bit of accuracy. However, if precision is the most important thing, Crank Nicolson is the best scheme for the diffusion equation in one dimension.

For the two dimensional case, we see the same results as for the one dimensional case for Forward Euler. We get better approximations by increasing the number of grid points. However, when we cross the stability limit, the solution blows up. I have not found the stability limit for the 2d case, but we see that by choosing step length as the limit for the one dimensional case, the solution blows up.

For the last part of the project we looked at the temperature distribution in the lithosphere. The forward Euler scheme was specialized to solve the new problem and the original problem was scaled to make the Forward Euler method more suitable. The results was as we expected. The steady state with no radioactive enrichment was linear between the boundary values. With only the constant Q added, the temperature increased and reached a new steady state. However, when we added an additional heat source that decayed over time the temperature increased to a maximum value before it started to decrease. This is what was expected. The Forward Euler method worked well for solving this problem.

For further work it would be interesting to test for even finer grids to see how high accuracy we can get. This is computationally expensive so I did not have time to test this for now. In addition the Forward Euler method in two dimension can be parallellized. This would speed up calculations. One could also implement more advanced methods like the finite element method to see if the results can improve even further. For the two dimensional case it is also possible to implement implicit schemes. It would be interesting to see if they give good results where the Forward Euler method blows up.

References

[1] Jensen M. H., *Computational Physics, lecture notes 2015*, Oslo, 2015.

[2] Tveito A., Winther R., *Introduction to Partial Differential Equations: A Computational Approach*, Springer, 1998.