University of Amsterdam

Scientific Computing

Assignment 1

Analysis of Partial Differential Equations Using Finite Difference Methods

Author: Kevin de Vries
Student Number: 10579869
E-mail: kevin.devries@student.uva.nl

April 25, 2018

Contents

1 Introduction

Partial differential equations (PDE) are often used in mathematical models across a wide variety of disciplines including physics, chemistry and biology. They are often used to add a spatial component to a model in the form of a convection term or a diffusion term. Many models, however, contain some form of nonlinearity in space, which means that the PDE cannot be solved analytically. We will then have to resort to numerical approximations to obtain our result. There are multiple ways to numerically solve a PDE, including Finite Element methods and Finite Volume methods, but the simplest methods used are Finite Difference (FD) methods.

In this report we will analyze three important PDEs - namely the wave equation, the diffusion equation and the Laplace equation - using FD methods. For the analysis we will mostly focus on the accuracy of the numerical solutions and the convergence of the methods. For the analysis of the Laplace equation we will also compare different iterative methods and we will analyze the stable state after introducing a nonlinearity by inserting objects into the grid.

The report will be structured as follows. First we will analyze the 1D wave equation. Secondly we will analyze the 2D diffusion equation. Finally we will analyze the Laplace equation in 2D using iterative methods and draw our conclusions from the results.

2 Tools (optional exercise)

I will briefly interrupt the report in order to give my answer to the optional exercise.

I am currently using the Linux distribution Ubuntu 16.04. The programming tools which I mostly like to use are Vim for simple programming and code editing and Jupyter Notebook for creating analysis notebooks. I have not looked into many other programming tools outside of standard environments for tools such as MATLAB, due to the versatility of Vim. The ways in which I try to become a better programmer are trying to frequently use efficient coding styles such as Object Oriented Programming and trying to

vectorize or parallelize my code as frequently as possible (if it yields a non-trivial speed up at least). I also try to add useful comments while I am writing my code.

3 The 1D wave equation

We will first derive and analyze an explicit FD scheme to solve the 1D wave equation. The 1D wave equation is given by

$$\frac{\partial^2 \mathbf{\Psi}}{\partial t^2} = c^2 \frac{\partial^2 \mathbf{\Psi}}{\partial x^2} \tag{1}$$

where c is the propagation speed of the wave. We can approximate this equation using FD methods by using the Taylor expansion of $\mathbf{\Psi}$ to get

$$\mathbf{\Psi}(x + \Delta x, t) \approx \mathbf{\Psi}(x,t) + \frac{\partial \mathbf{\Psi}(x,t)}{\partial x}\Delta x + \frac{1}{2!}\frac{\partial^2 \mathbf{\Psi}(x,t)}{\partial x^2}\Delta x^2 + \mathcal{O}(\Delta x^3) \tag{2}$$

$$\mathbf{\Psi}(x - \Delta x, t) \approx \mathbf{\Psi}(x,t) - \frac{\partial \mathbf{\Psi}(x,t)}{\partial x}\Delta x + \frac{1}{2!}\frac{\partial^2 \mathbf{\Psi}(x,t)}{\partial x^2}\Delta x^2 - \mathcal{O}(\Delta x^3) \tag{3}$$

Adding equations (2) and (3) for both derivatives in time and space, truncating the Taylor expansion after the second order derivative and indexing the discretization according to $\mathbf{\Psi}(i\Delta x, k\Delta t) = \mathbf{\Psi}_i^k$ yields the central FD approximations

$$\frac{\partial^2 \mathbf{\Psi}_i^k}{\partial x^2} = \frac{\mathbf{\Psi}_{i+1}^k - 2\mathbf{\Psi}_i^k + \mathbf{\Psi}_{i-1}^k}{\Delta t^2} \tag{4}$$

$$\frac{\partial^2 \mathbf{\Psi}_i^k}{\partial t^2} = \frac{\mathbf{\Psi}_i^{k+1} - 2\mathbf{\Psi}_i^k + \mathbf{\Psi}_i^{k-1}}{\Delta x^2} \tag{5}$$

Inserting these approximations back into equation (1) and rewriting finally yields

$$\mathbf{\Psi}_i^{k+1} = 2\mathbf{\Psi}_i^k - \mathbf{\Psi}_i^{k-1} + \frac{c^2 \Delta t^2}{\Delta x^2}(\mathbf{\Psi}_{i+1}^k - 2\mathbf{\Psi}_i^k + \mathbf{\Psi}_{i-1}^k) \tag{6}$$

We consider a vibrating string with length L where the boundaries are fixed at the origin. This results in the boundary conditions $\mathbf{\Psi}(x = 0, t) = \mathbf{\Psi}(x = L, t) = 0$. Additionally we take $\Delta x = \frac{L}{N}$ where N is the number of spacial grid points, which means that $\mathbf{\Psi}_0^k = \mathbf{\Psi}_N^k = 0$. With these boundary conditions

we can determine the FD approximation at the boundaries to be

$$\mathbf{\Psi}_1^{k+1} = 2\mathbf{\Psi}_1^k - \mathbf{\Psi}_1^{k-1} + \frac{c^2\Delta t^2}{\Delta x^2}(\mathbf{\Psi}_2^k - 2\mathbf{\Psi}_1^k) \tag{7}$$

$$\mathbf{\Psi}_{N-1}^{k+1} = 2\mathbf{\Psi}_{N-1}^k - \mathbf{\Psi}_{N-1}^{k-1} + \frac{c^2\Delta t^2}{\Delta x^2}(\mathbf{\Psi}_{N-2}^k - 2\mathbf{\Psi}_{N-1}^k) \tag{8}$$

Since we need the values of the two previous time points, we will to determine $\mathbf{\Psi}_i^1$ using only the initial conditions. We can do this by specifying both $\mathbf{\Psi}_i^0$ and $\mathbf{\Psi}_t(x, t = 0) = (\mathbf{\Psi}_t)_i^0$. We can then use the forward FD approximation to get

$$\mathbf{\Psi}_i^1 = \mathbf{\Psi}_i^0 + \Delta t(\mathbf{\Psi}_t)_i^0 \tag{9}$$

3.1 Results

In order to analyze our FD solver we use the default parameters $L = 1$, $c = 1$, $\Delta t = 0.001$ and $N = 1000$. We assume the string is at rest when $t = 0$, which means that $\mathbf{\Psi}_t(x, t = 0) = 0$. For the analysis we use the following initial conditions:

1. $\mathbf{\Psi}(x, t = 0) = \sin(2\pi x)$
2. $\mathbf{\Psi}(x, t = 0) = \sin(5\pi x)$
3. $\mathbf{\Psi}(x, t = 0) = \sin(5\pi x)$ if $\frac{1}{5} < x < \frac{2}{5}$, else $\mathbf{\Psi}(x, t = 0) = 0$

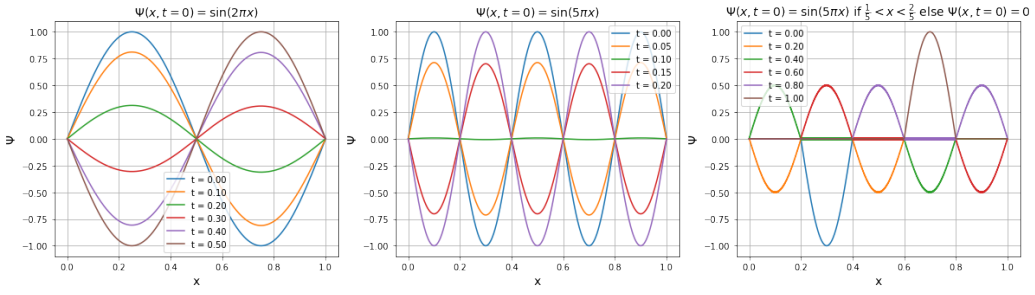

Figure 1: The time evolution of the 1D wave equation with initial conditions $\mathbf{\Psi}(x, t = 0) = \sin(2\pi x)$, $\mathbf{\Psi}(x, t = 0) = \sin(5\pi x)$ and $\mathbf{\Psi}(x, t = 0) = \sin(5\pi x)$ if $\frac{1}{5} < x < \frac{2}{5}$, else $\mathbf{\Psi}(x, t = 0) = 0$ respectively

The numerical solutions of the 1D wave equation with the given initial conditions can be found in Figure 1. We observe that with the first two initial conditions the solutions produce stationary waves. With the third initial condition however, we see that the wave splits into two smaller waves which propagate in opposite directions. These waves then recombine at $x = 0.7$ after which the wave splits into the two smaller waves again that propagate in opposite directions again. Finally the waves recombine again at $x = 0.3$, which was the initial state of the amplitude of the wave.

4 The 2D time dependent diffusion equation

Next we will implement and analyze the 2D time dependent diffusion equation. The time dependent diffusion equation is given by the expression

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{10}$$

where $c(x, y, t)$ is the concentration in 2D at time t and D is the diffusion coefficient. By using the central FD approximation for $\frac{\partial^2 c}{\partial x^2}$ and $\frac{\partial^2 c}{\partial y^2}$ and using the forward FD approximation for $\frac{\partial c}{\partial t}$ we can approximate the time dependent diffusion in 2D as

$$c_{i,j}^{k+1} = c_{i,j}^k + \frac{\Delta t D}{\Delta x^2}(c_{i+1,j}^k + c_{i-1,j}^k + c_{i,j+1}^k + c_{i,j-1}^k - 4c_{i,j}^k) \tag{11}$$

where we define $c(i\Delta x, j\Delta x, k\Delta t) = c_{i,j}^k$. The FD scheme is stable for $\Delta t \leq \frac{\Delta x^2}{4D}$. We assume periodic boundary conditions in the x-direction and Dirichlet boundary conditions in the y-direction with a constant source at the top and a sink at the bottom of the domain. Additionally we assume the domains of x and y to be in the interval $[0, 1]$. The boundary conditions are thus given by

$$c(x = 0, y, t) = c(x = 1, y, t) \tag{12}$$
$$c(x, y = 0, t) = 0 \tag{13}$$
$$c(x, y = 1, t) = 1 \tag{14}$$

For the initial condition we take

$$c(x, y, t = 0) = 0 \text{ for } 0 \leq x \leq 1, 0 \leq y < 1 \tag{15}$$

Due to the periodic boundary conditions we assume the grid points at 0 and N, where N is the number of grid points in both the x and y directions and thus $\Delta x = \frac{1}{N}$, to be the same. we can thus determine the FD approximation at the boundaries to be

$$c_{0,j}^{k+1} = c_{N,j}^{k+1} = c_{0,j}^{k} + \frac{\Delta t D}{\Delta x^2}(c_{1,j}^{k} + c_{N-1,j}^{k} + c_{0,j+1}^{k} + c_{0,j-1}^{k} - 4c_{0,j}^{k}) \tag{16}$$

$$c_{i,1}^{k+1} = c_{i,1}^{k} + \frac{\Delta t D}{\Delta x^2}(c_{i+1,1}^{k} + c_{i-1,1}^{k} + c_{i,2}^{k} - 4c_{i,1}^{k}) \tag{17}$$

$$c_{i,N-1}^{k+1} = c_{i,N-1}^{k} + \frac{\Delta t D}{\Delta x^2}(c_{i+1,N-1}^{k} + c_{i-1,N-1}^{k} + 1 + c_{i,N-2}^{k} - 4c_{i,N-1}^{k}) \tag{18}$$

For the set of initial and boundary conditions which we use an analytic solution can be derived. The analytic solution is given by

$$c(y,t) = \sum_{i=0}^{\infty} \operatorname{erfc}\left(\frac{1-y+2i}{2\sqrt{Dt}}\right) - \operatorname{erfc}\left(\frac{1+y+2i}{2\sqrt{Dt}}\right) \tag{19}$$

The solution is only dependent on y and t, due to the symmetry in x caused by the periodic boundary conditions in the x-direction. Note that when t goes to infinity the solution becomes $\lim_{t\to\infty} c(y,t) = y$. The system thus converges to a stable state where the concentration is linear in the y-direction.

4.1 Results

In order to test the correctness of our FD solver for the 2D diffusion equation we use the default parameters $D = 1$, $N = 50$ and $\Delta t = 0.0001$. Additionally we take the spacial coordinates in the domain $x, y \in [0, 1]$. The values for N and Δt have been chosen such that the stability condition of the explicit FD scheme is satisfied.

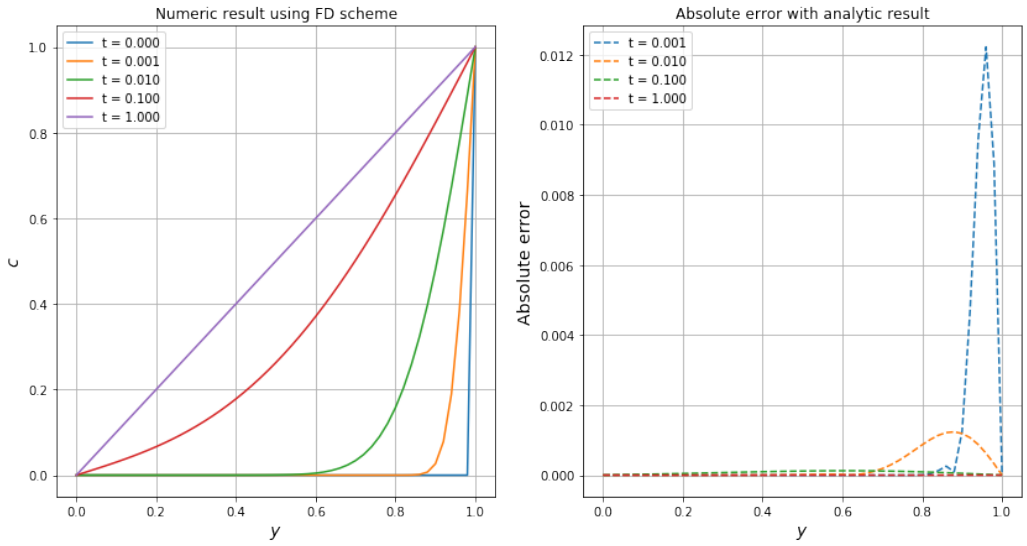

Figure 2: The concentration of the numeric solution of the diffusion equation dependent on the position in the y-direction at different times compared with the analytic solution

We test the correctness of the FD scheme by comparing the numerical solution with the analytic solution given in equation (19). The numerical solution and the comparison with the analytic solution are plotted in Figure 2. We find from Figure 2 that after enough time the solution indeed converges to a linear function dependent on y. We also find that the absolute error between the numerical and analytic solutions generally seems to be sufficiently small and even seems to decrease over time as the solution becomes increasingly linear.

We also observe the 2D domain of the solution at different times, which can be found in Figure 3. In this domain we see that the concentration indeed diffuses to the bottom of the domain and converges to a linear dependence on y after enough time, since the domain at time $t = 1$ seems to resemble this kind of linear dependence in the y-direction.

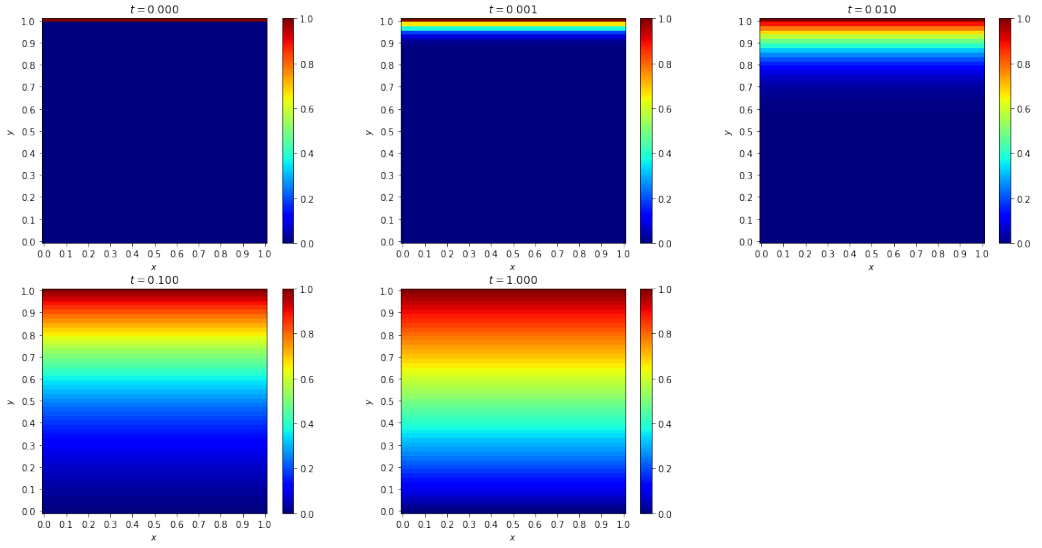

Figure 3: Visualization of the diffusion of the concentration over time

5 Iterative methods

Lastly we will implement and analyze the Laplace equation the using iterative methods: Jacobi iteration, Gauss-Seidel iteration and Successive Over Relaxation (SOR). The Laplace equation can be described as a special case of the diffusion equation where the system has already reached a stable state, which means that $\frac{\partial c}{\partial t} = 0$. This yield the expression for the Laplace equation

$$\nabla^2 c = 0 \tag{20}$$

The simplest iterative method which can be used to solve this equation is to use the maximum possible time step Δt which satisfies the stability condition for the explicit FD solver given in the previous section. From the stability condition we get $\Delta t = \frac{\Delta x^2}{4D}$, which we then insert into equation (11) to get the Jacobi iteration scheme

$$c_{i,j}^{k+1} = \frac{1}{4}(c_{i+1,j}^{k} + c_{i-1,j}^{k} + c_{i,j+1}^{k} + c_{i,j-1}^{k}) \tag{21}$$

In order to test if the solution has converged to a stable state we define a convergence measure

$$\delta \equiv \max |c_{i,j}^{k+1} - c_{i,j}^{k}| < \varepsilon \tag{22}$$

where ε is the tolerance below which we consider the solution to have converged to the stable state. We can improve on the amount of iterations needed for convergence by using a Successive Over Relaxation iteration scheme, which is given by

$$c_{i,j}^{k+1} = \frac{\omega}{4}(c_{i+1,j}^{k+1} + c_{i-1,j}^{k} + c_{i,j+1}^{k+1} + c_{i,j-1}^{k}) + (1-\omega)c_{i,j}^{k} \tag{23}$$

where ω is a relaxation parameter. The case where $\omega = 1$ is known as Gauss-Seidel iteration. For the SOR scheme it is known that the optimal value for ω is in the interval $1.7 < \omega < 2.0$.

5.1 Red-Black ordering

In the case of Gauss-Seidel iteration and SOR the iteration can be done in place. The SOR iteration scheme much less obvious to parallelize or vectorize than Jacobi iteration. One way to allow the SOR iterations to be done in parallel is to use Red-Black ordering. When using Red-Black ordering we divide the grid points into sets of red and black grid points, which are ordered in a checkerboard pattern. This means that red grid points alternate with black grid points in space. Red grid points thus only interact with black grid points and vice versa. We can then perform the SOR iterations by alternating an iteration of the red grid points with an iteration of the black grid points and checking for convergence after each subsequent iteration of both red and black grid points. This particular iteration ordering lends itself easier to parallelization and vectorization. In this report we make use of a vectorized implementation for SOR which makes use of Red-Black ordering.

5.2 Results

In order to test the correctness of the implementations of the iterative methods we start with $N = 50$ and we set the tolerance to $\varepsilon = 10^{-5}$. We test the correctness of the Jacobi iteration, Gauss-Seidel iteration and SOR by examining the stable states after convergence.

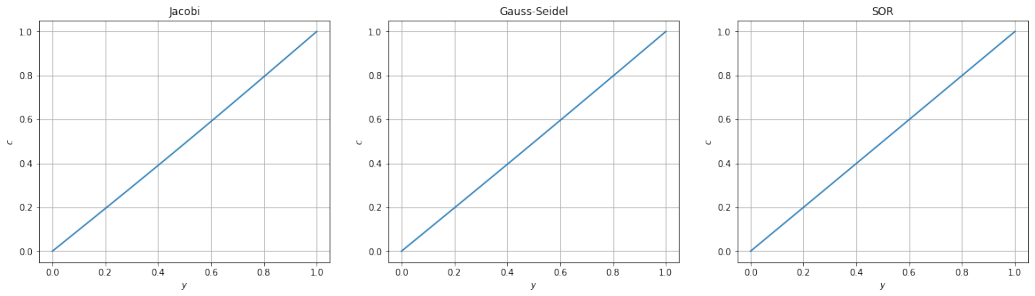

Figure 4: The concentration as a function of y of the steady states resulting from Jacobi iteration, Gauss-Seidel iteration and SOR respectively

The concentration as a function of the y coordinate can be found in Figure 4. We find that for each iterative method the result has converged to the stable state, which we can see from the linear dependence on y.

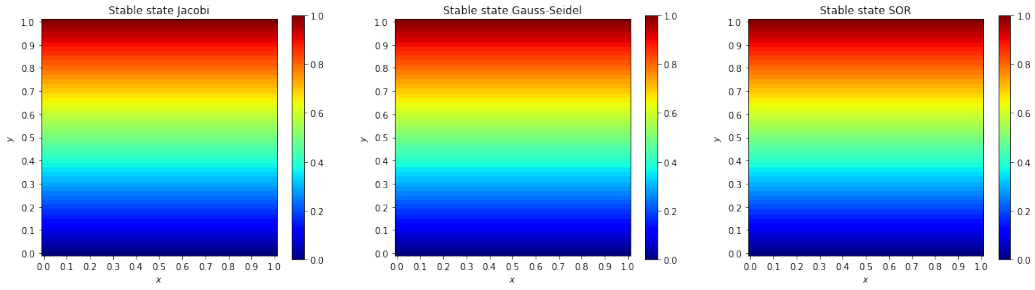

Figure 5: Visualization of the 2D domains of the stable states resulting from Jacobi iteration, Gauss-Seidel iteration and SOR respectively

We also visualize the 2D domains of the stable states for each iterative method, which can be seen in Figure 5. We find the same domain as the domain in Figure 3 after enough time has passed.

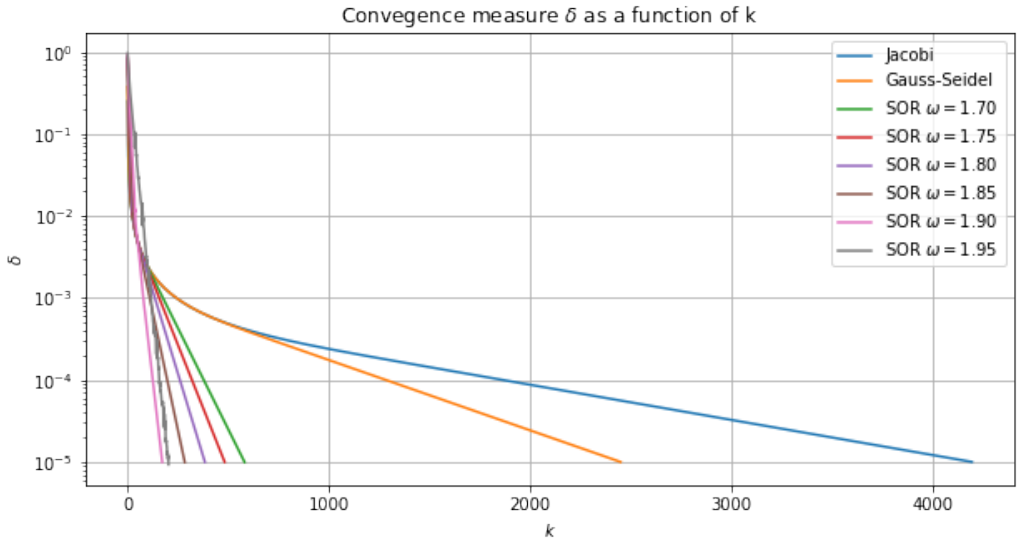

Figure 6: Convergence measure δ as a function of the number of iterations k needed for convergence for Jacobi iteration, Gauss-Seidel iteration and SOR using varying values for ω

Next we investigate the convergence of the methods. We first compare the convergence measure δ as a function of the number of iterations k between Jacobi iteration, Gauss-Seidel iteration and SOR using different ω in the domain $1.7 < \omega < 2$. A plot of the comparison is shown in Figure 6. We find that Jacobi iteration seems to converge the slowest by far whereas Gauss-Seidel iteration converges approximately twice as fast. SOR however seems to converge the fastest with the number iterations decreasing as ω increases. We find that the convergence seems to be optimal for $1.9 \leq \omega < 1.95$.

We also try to find the optimal ω for SOR and its dependence on N. In order to find the optimal ω for each grid size we plot we number of iterations for varying ω with multiple different N. We then find the optimal ω for each N and plot those ω as a function of N. The results are plotted in Figure 7.

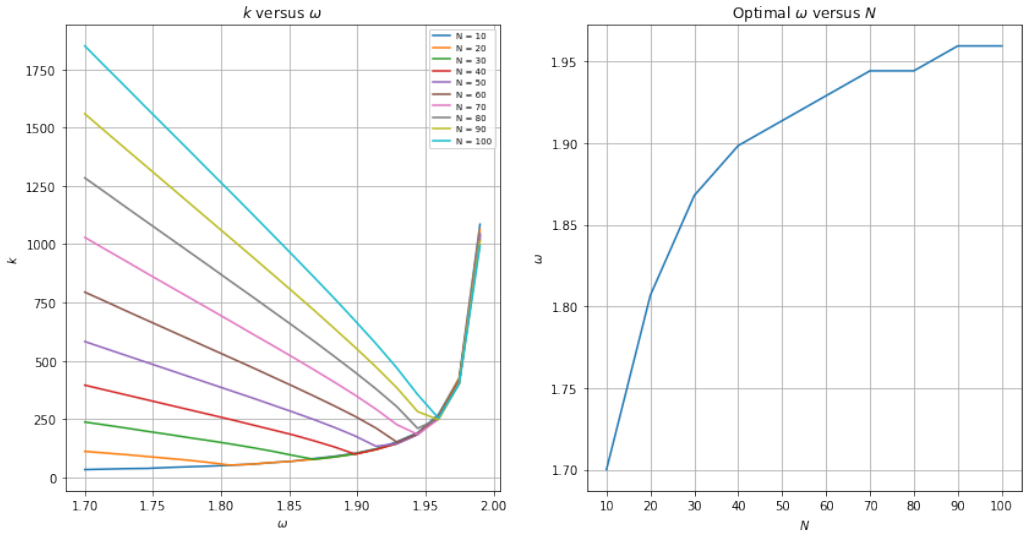

Figure 7: Plots of the the number of iterations k needed for convergence as a function of ω and the optimal ω as a function of the grid size N

In Figure 7 we find that the optimal ω increases as N increases. Furthermore we find that the number of iterations needed to converge seems to become the same for all grid sizes when ω becomes larger than the optimal ω. We also find that the optimal ω seems to increase faster for lower grid sizes and starts to increase slowly to $\omega = 2$ for larger grid sizes.

5.2.1 Objects in the grid

Now that we have tested the correctness of the iterative methods and investigated the optimal ω we can test our implementation on a non-uniform grid by adding objects into the grid. We define the objects as grid points i, j which act as sinks where $c^k_{i,j} = 0$ for all k. In our implementation we add objects by setting the concentrations at the grid points occupied by an object to zero after each iteration.

We investigate the effect of adding objects to the grid on the convergence and optimal ω by examining two object additions. The first addition is a circle which we load into the grid from an image file. The second addition we use are two squares close to the upper corners of the grid along with the circle. The object additions along with the 2D domains of the stable states

can be found in Figure 8.

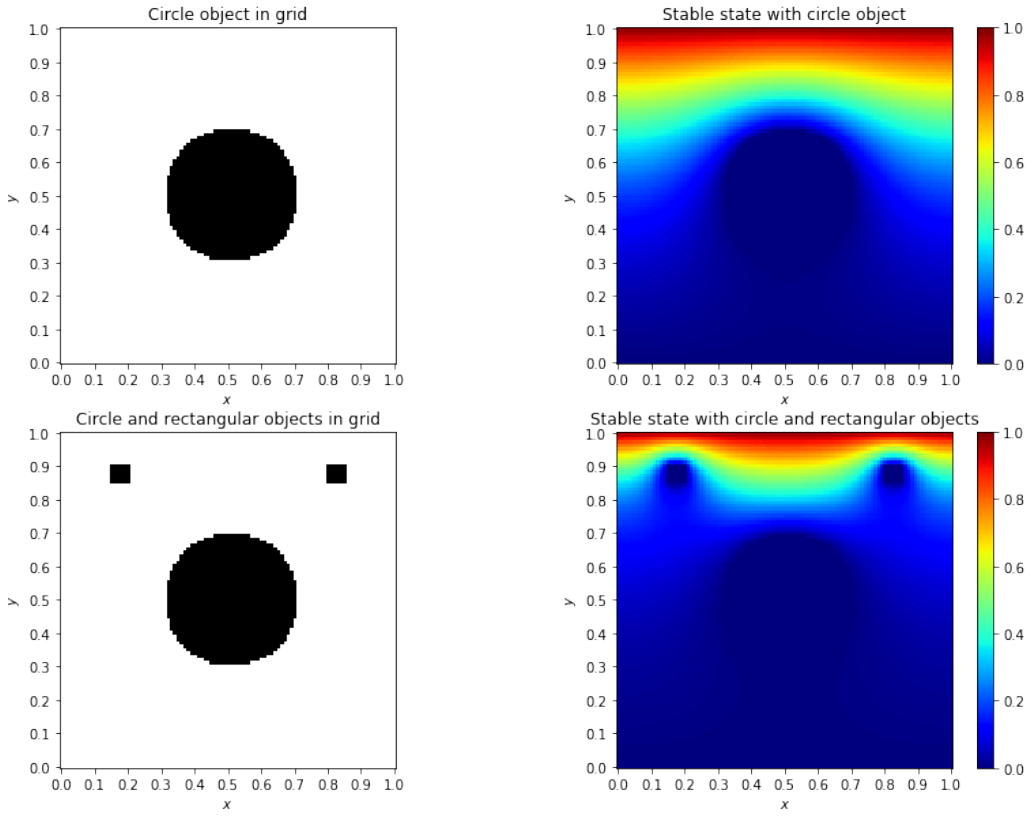

Figure 8: Visualization of the stable states and the objects in the grid for a grid with a circle object and a grid with a circle object and two rectangular objects

We investigate the convergence and optimal ω of these additions for different grid sizes by using the same objects where the sizes are proportional to the grid size used. We calculate number of iterations needed for convergence as a function of ω for both object additions. The results for both object additions can be found in Figure 9.

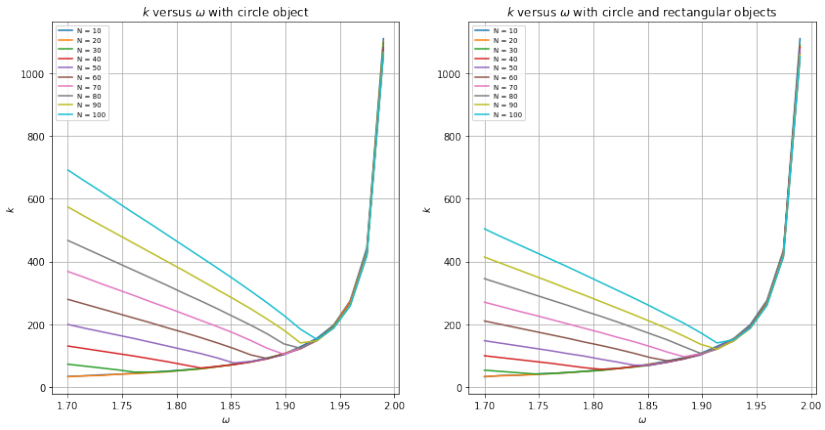

Figure 9: Number of iterations needed for convergence as a function of ω for a grid with a circle object and a grid with a circle object and two rectangular objects

From Figure 9 we can discern from a comparison with Figure 7 that adding objects to the grid seems to decrease the optimal ω. We find that adding two smaller objects close to the source decreases the optimal ω by little in comparison with adding the circle to the grid. Using the plot of the optimal ω as a function of N given in Figure 10 we do however find that the optimal ω does decrease when adding the smaller objects.

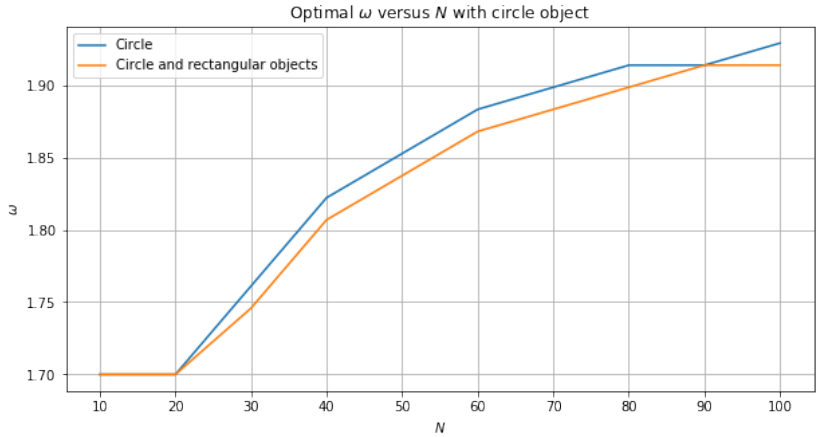

Figure 10: Optimal ω as a function of N for a grid with a circle object and a grid with a circle object and two rectangular objects

6 Conclusions

From the analysis of the numerical solutions of the 1D wave equation and the time dependent 2D diffusion equation we conclude that the explicit FD methods used produce sufficiently accurate estimates as given by the comparison with the analytic solution and the visualizations of the time integration of the PDEs. From the analysis of the Laplace equation using iterative methods we conclude that all iterative methods converge the stable state given by the analytic solution. We also conclude that the optimal relaxation parameter used for the SOR method is heavily dependent on the number of grid points in the system and increases with the number of grid points. Finally we can conclude that adding objects into the grid decreases the optimal value of the relaxation parameter for the SOR method.