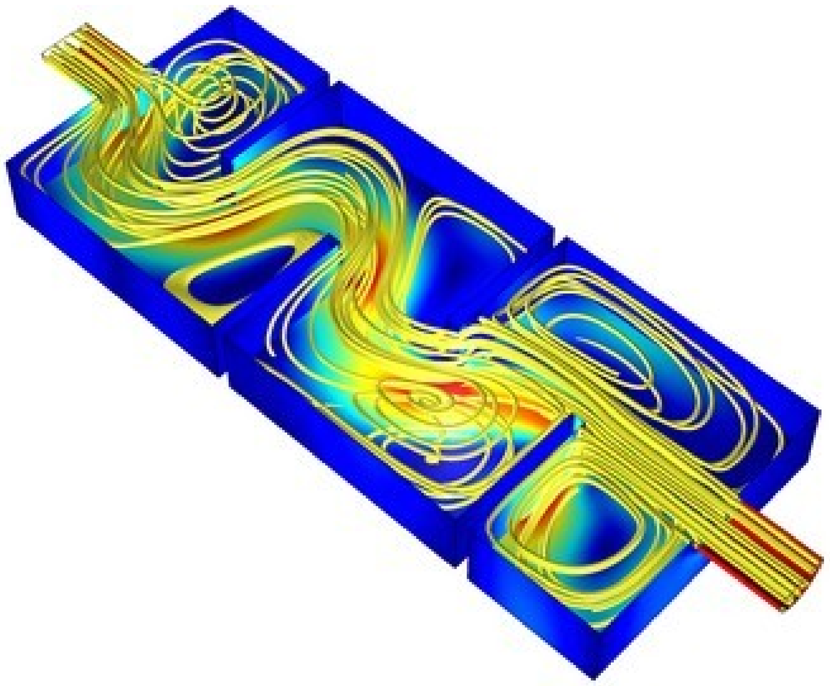
Numerical Methods for Engineers
201800131

Rob G.H. Lammertink

April 11, 2025



Contents

Part I

Reader

Chapter 1

Discretization of ODEs

Key to computational fluid dynamics are the techniques required to solve the governing equations. In the next chapters we will present the fluid dynamics equations, in their differential form. The governing equations include the conservation of mass, energy, and momentum. The process for obtaining solutions for these equations consists of two stages; first discretization and second the implementation.

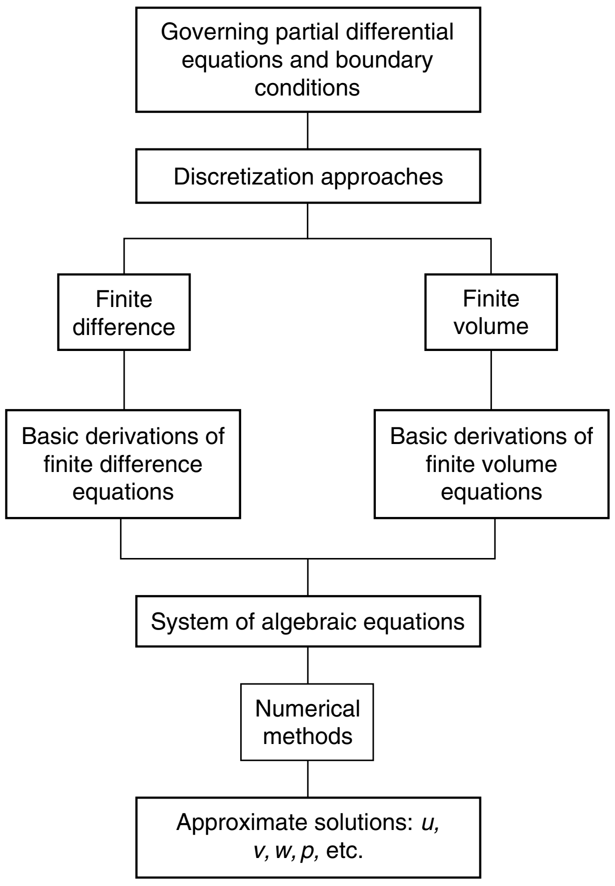

Governing partial differential equations and boundary conditions
Discretization approaches
Finite difference
Finite volume
Basic derivations of finite difference equations
Basic derivations of finite volume equations
System of algebraic equations
Numerical methods
Approximate solutions: *u, v, w, p,* etc.

Figure 1.1: Basics for CFD solution procedure

1.1 Finite Difference

The step for obtaining a numerical solution consist of converting the partial differential equations and boundary conditions into a system of discrete algebraic functions. As can be seen from figure 1.1, two main procedures, finite difference and finite volume, are indicated. It should be noted that it is not uncommon that the two methods produce the same set of equations after discretization. Here we will focus on the finite difference method. This method approximates all partial differential equations with finite difference equations:

$$\frac{du}{dx} = \frac{\Delta u}{\Delta x} \tag{1.1}$$

for the limit that $\Delta x \to 0$. As for Δx a finite difference will be taken, an approximation error will result, known as the truncation error. The magnitude of the error, and how it scales with the size of Δx depends on the discretization scheme used. For the finite difference scheme, the domain has to be discretized first. This implies that variables are located on a fixed grid (figure 1.2).

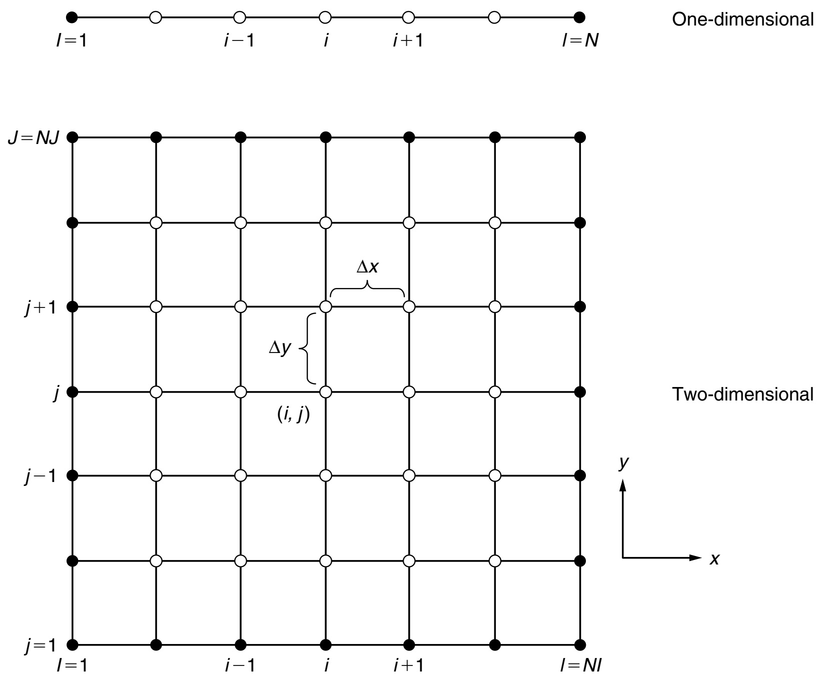


Figure 1.2: A one-dimensional and two-dimensional representation of a grid for a finite difference method.

In order to evaluate the values at different location on the grid, we use the Taylor series expansion. The general form of the Taylor series is given by:

$$\sum_{n=0}^{\infty} \frac{d^n f(a)}{dx^n} \frac{(x-a)^n}{n!} \tag{1.2}$$

This series can be used to evaluate f at a distance (or timestep) of $(x-a)$ away from a known point. For the one-dimensional example, this means that we can express a variable at location $i+1$ by a Taylor series expansion about point i.

$$u_{i+1} = u_i + \left(\frac{du}{dx}\right)_i \Delta x + \left(\frac{d^2u}{dx^2}\right)_i \frac{\Delta x^2}{2} + \left(\frac{d^3u}{dx^3}\right)_i \frac{\Delta x^3}{6} + ... \tag{1.3}$$

The number of terms is infinite. The approximation then uses only a finite number of terms from the Taylor series. Isolating the first-order derivative out of this equation and explicitly presenting the first two terms only gives:

$$\left(\frac{du}{dx}\right)_i = \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x) \tag{1.4}$$

where $O(\Delta x)$ represents the truncation error. The order of the truncation error here is said to be first order, as the error decreases linearly with Δx. The expression for the derivative in equation 1.4 is called a forward difference, since the point $i+1$ is located forward with respect to i and these are used to approximate the derivative at point i. Similarly, we can express the variable at location $i-1$ by:

$$u_{i-1} = u_i - \left(\frac{du}{dx}\right)_i \Delta x + \left(\frac{d^2u}{dx^2}\right)_i \frac{\Delta x^2}{2} - \left(\frac{d^3u}{dx^3}\right)_i \frac{\Delta x^3}{6} + ... \tag{1.5}$$

The backward difference is then given by:

$$\left(\frac{du}{dx}\right)_i = \frac{u_i - u_{i-1}}{\Delta x} + O(\Delta x) \tag{1.6}$$

The expression for central difference is obtained by subtracting equation 1.5 from equation 1.3:

$$\left(\frac{du}{dx}\right)_i = \frac{u_{i+1} - u_{i-1}}{2\Delta x} + O\left(\Delta x^2\right) \tag{1.7}$$

Notice that the truncation error for the central difference scheme is of order 2. This means that the error reduces quadratically as the step size Δx is reduced. This method is therefore more accurate compared to the forward and backward differencing, but it comes at the cost of requiring more points next to i. The higher order accuracy comes from the cancellation of terms in the Taylor series expansion. Higher order discretization schemes are possible by combining Taylor series from multiple location, e.g. u_{i-2}, u_{i-1}, u_{i+1}, u_{i+2}, etc.

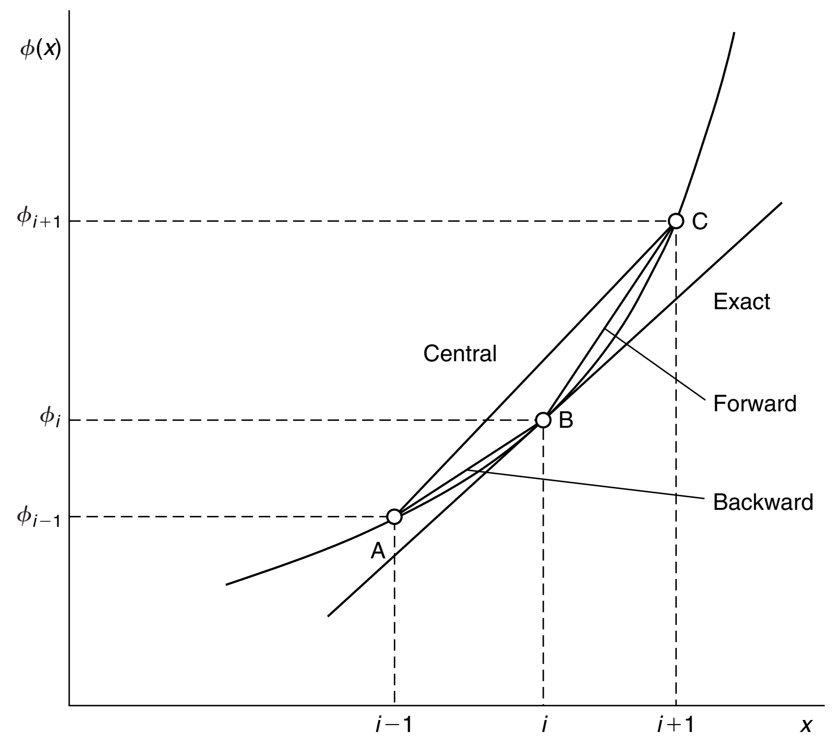


Figure 1.3: Representation of the finite difference schemes for the derivative $\left(\frac{du}{dx}\right)_i$ around $i-1$, i, and $i+1$

So far, we have obtained finite difference equations that approximate the first derivatives $\frac{du}{dx}$. It is also possible to obtain higher-order derivative approximations. Adding equation 1.5 to equation 1.3 gives:

$$u_{i-1} + u_{i+1} = 2u_i + \left(\frac{d^2u}{dx^2}\right)_i \Delta x^2 + ... \tag{1.8}$$

From which the approximation for the second order derivative can be isolated:

$$\left(\frac{d^2u}{dx^2}\right)_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O\left(\Delta x^2\right) \tag{1.9}$$

The truncation error is evidently a consequence of the chosen discretization formula. Generally, it is possible to have more accurate approximations, when including more terms in the approximations.

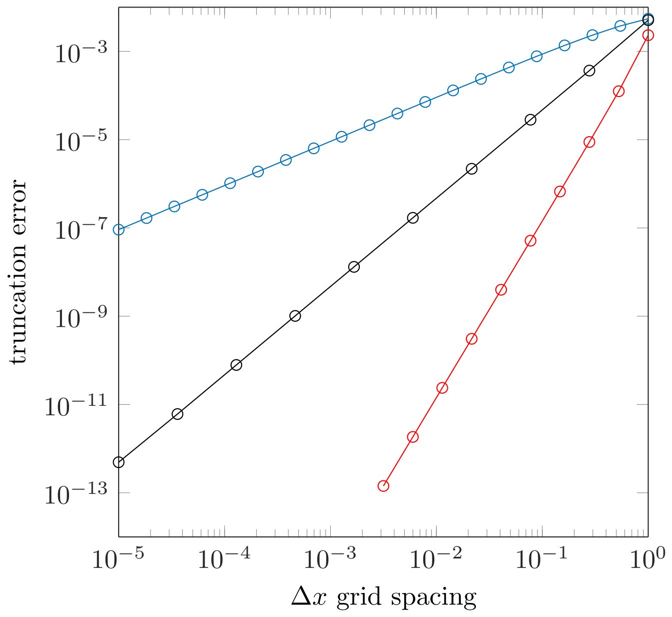


Figure 1.4: Truncation error as a function of step size Δx, for first order scheme, second order scheme, and a fourth order scheme

If one wants to restrict the number of points involved in a discretization formula, it is possible to construct this leading to the most accurate difference scheme. By taking a linear combination of the Taylor series for the terms we propose to include, we need to find each weight to zero as many of the low-order terms as possible. For instance, if we only envision to use i, $i+1$, and $i+2$ terms to approximate the first derivative $\frac{du}{dx}_i$, we have the following formula:

$$\frac{du}{dx}_i + au_i + bu_{i+1} + cu_{i+2} = O(?) \tag{1.10}$$

Using the Taylor expansions of the u_i, u_{i+1} and u_{i+2} terms, we can see that we need:

$$a + b + c = 0 \tag{1.11}$$

$$1 + b\Delta x + c2\Delta x = 0 \tag{1.12}$$

$$b\Delta x^2/2 + c2\Delta x^2 = 0 \tag{1.13}$$

in order to eliminate the required terms and arrive at the highest possible accuracy. In this example, the solution is the second order accurate scheme:

$$\left(\frac{du}{dx}\right)_i = \frac{-3u_i + 4u_{i+1} - u_{i+2}}{2\Delta x} + O(\Delta x^2) \tag{1.14}$$

Chapter 2

Numerical Solutions of ODEs, IVP

2.1 Initial value problems

Most fluid dynamics problems are described by partial differential equations (PDEs). Core to solving PDEs are ordinary differential equations (ODEs). Where PDEs describe properties as a function of more than one variable, e.g. space and time, ODEs describe such properties as a function of just one variable.

In this chapter, we will discuss some of the basics to solve ODEs via finite difference methods.

An ordinary differential equation describes the behaviour of a property (e.g. concentration, temperature) as a function of a single variable (e.g. time, location). Consider the following ODE:

$$\frac{du}{dt} = f(u,t) \qquad\qquad u(0) = u_0 \tag{2.1}$$

where $f(u,t)$ describes an arbitrary function of u and t and $u(0)$ provides the initial condition of u. This is an initial value problem. The initial value of u is known, and we are interested in obtaining $u(t)$. In order to estimate the value of u at later time, we will timestep forward with small steps. This stepping process is the core of finite difference methods.

$$\frac{du}{dt} = \frac{\Delta u}{\Delta t} \tag{2.2}$$

In order to evaluate the values at different location on the grid, we use the Taylor series expansion. The general form of the Taylor series is given by:

$$\sum_{n=0}^{\infty} \frac{d^n f(a)}{dt^n} \frac{(t-a)^n}{n!} \tag{2.3}$$

This series can be used to evaluate f some time $(t-a)$ later (or some distance away, in case of a spatial variable). For the one-dimensional example, this means that we can express a variable at time $i+1$ by a Taylor series expansion about point i.

$$u_{i+1} = u_i + \left(\frac{du}{dt}\right)_i \Delta t + \left(\frac{d^2u}{dt^2}\right)_i \frac{\Delta t^2}{2} + \left(\frac{d^3u}{dt^3}\right)_i \frac{\Delta t^3}{6} + ... \tag{2.4}$$

The number of terms is infinite. The approximation then uses only a finite number of terms from the Taylor series. Isolating the first-order derivative out of this equation and explicitly presenting the first two terms only gives:

$$\left(\frac{du}{dt}\right)_i = \frac{u_{i+1} - u_i}{\Delta t} + O\left(\Delta t\right) \tag{2.5}$$

where $O(\Delta t)$ represents the truncation error. The order of the truncation error here is said to be first order, as the error decreases linearly with Δt. The expression for the derivative in equation 2.5 is called a forward difference, or explicit Euler since the value at time $i+1$ is obtained from the values at the previous time i. We will now explore and analyze different discretization methods and their accuracy and stability.

We do this by introducing our model equation:

$$\frac{du}{dt} = \lambda u \tag{2.6}$$

The model equation will provide insight in the stability and accuracy of the discretization scheme. It also plays a crucial role, later on, when we are going to solve PDE's. An important feature of λ is that we allow it to be a complex number. The imaginary part of the coefficient results in oscillatory solutions (e.g. waves), and the real part indicates the growth or decay. Realize that a positive value for λ results in an exponentially growing *exact* solution. Therefore, to study whether discretization schemes are stable and accurate, we only consider the real part of λ to be zero or negative. The analytic solution to the model equation is:

$$u(t) = u_0 e^{\lambda t} \tag{2.7}$$

we can use $t = nh$, for n timesteps of $h = \Delta t$ to get

$$u(t) = u_0 \left(e^{\lambda h}\right)^n \tag{2.8}$$

such that it is clear that the amplification factor for the model equation is

$$e^{\lambda h} = 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} + O(h^5) \tag{2.9}$$

2.2 Explicit Euler

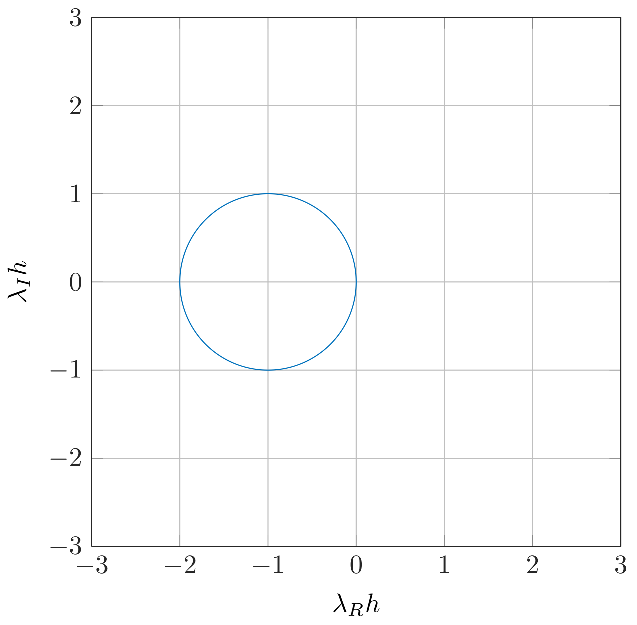


Figure 2.1: Explicit Euler stability plot, where stability is obtained only for λh values inside the indicated circle

We are rewriting the explicit Euler equation as:

$$u_{i+1} = u_i + hf(u_i, t_i) \tag{2.10}$$

where we assume a constant timestep $\Delta t = h$. Applying this explicit Euler scheme to our model equation gives:

$$u_{i+1} = u_i + h\lambda u_i = u_i(1 + \lambda h) \tag{2.11}$$

realize that the solution after n timesteps is then simply:

$$u_n = u_0(1 + \lambda h)^n \tag{2.12}$$

the term $(1 + \lambda h)$ is our amplification factor. It recovers the amplification factor of the analytic solution up to λh and is thus accurate to $O(h)$. For a stable numerical solution, this factor should remain bounded as n becomes large. The numerical solution is therefore only stable when:

$$|1 + \lambda h| \leq 1 \tag{2.13}$$

Recall that we allow λ to be a complex number. The region of stability for the explicit Euler scheme is presented in figure 2.1. Only for values of λh that are located inside the circle, the numerical scheme is stable. We call this conditionally stable.

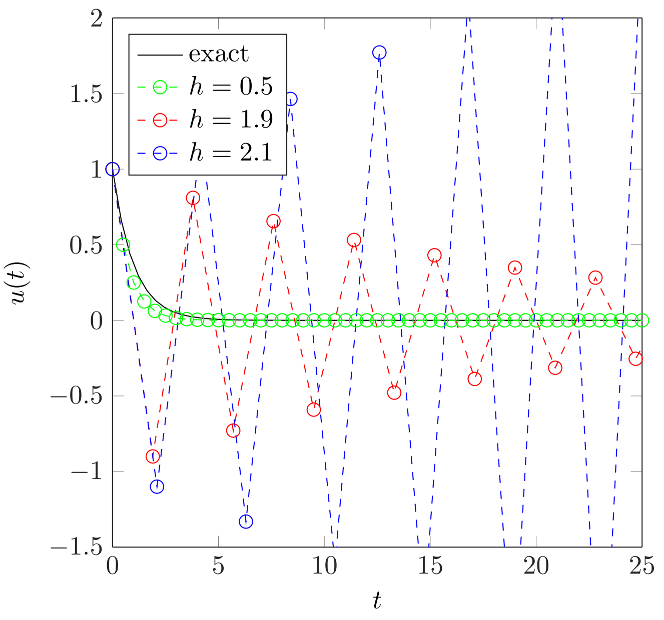


Figure 2.2: Plot indicating the numerical solution of the model equation, where $\lambda = -1$, using the explicit Euler scheme with different time step size h. For step sizes smaller than 2, the solution converges.

Depending on the value of λ, we must adjust the time step h such to meet this criteria. From the amplification factor we know that the numerical solution will be stable for real (and negative) λ, when $h \leq \frac{2}{|\lambda|}$. This is graphically demonstrated in figure 2.2. It is clear that a smaller step size follows the solution closer, but for step sizes close to the maximum the solution also converges. Step sizes larger than the maximum result in a diverging solution (blows up).

2.3 Implicit Euler

The implicit Euler scheme is given as follows:

$$u_{i+1} = u_i + hf(u_{i+1}, t_{i+1}) \tag{2.14}$$

This scheme is not explicit, as it requires values that are not known. This makes solving an implicit scheme completely different compared to an explicit one. When f is a linear

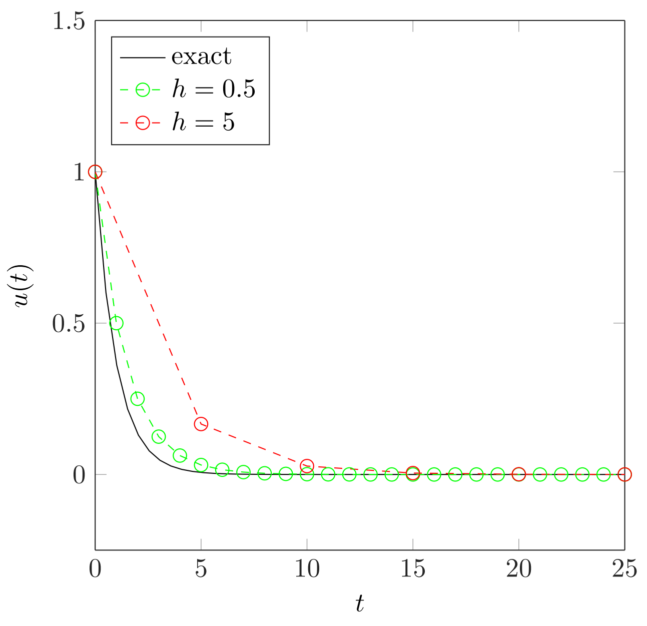


Figure 2.3: Plot indicating numerical solution of the model equation (with $\lambda = -1$) using the implicit Euler scheme. Even for very large step size, $h = 5$, the solution converges.

equation (as it is for the model equation) we can simply transpose the scheme:

$$u_{i+1} = u_i + \lambda h u_{i+1} \tag{2.15}$$

into

$$u_{i+1} = u_i \frac{1}{1 - \lambda h} \tag{2.16}$$

realize for any non-linear equation, e.g. containing an exponent or harmonic function, this will not be possible and we will have to solve a non-linear algebraic equation at each time step. This usually requires iterations and thus higher computational cost (or applying the linearization technique discussed later). Applying the implicit Euler scheme to the model equation we observe that the amplification factor provides the following stability criterium:

$$|\frac{1}{1 - \lambda h}| \leq 1 \tag{2.17}$$

The scheme is therefore unconditionally stable (as we only considered the real part of λ to be zero or negative). It is important to recognize that the implicit method does not put any restrictions on the step size. This is nicely demonstrated in figure 2.3. Where the explicit Euler scheme presented growing fluctuations for step sizes above a defined maximum, the implicit Euler scheme remains bounded, also for large step sizes.

2.4 Trapezoidal

The trapezoidal discretization scheme, often called Crank-Nicolson, is given by:

$$u_{i+1} = u_i + \frac{h}{2} \left(f(u_{i+1}, t_{i+1}) + f(u_i, t_i) \right) \tag{2.18}$$

it shows that it is also an implicit method. Effectively, it uses a time averaged evaluation (at $i + 1$ and i) to proceed to the next time step. We now apply the trapezoidal scheme to our model equation again:

$$u_{i+1} = u_i + \frac{h}{2} \left(\lambda u_{i+1} + \lambda u_i \right) \tag{2.19}$$

rearranging into

$$u_{i+1} = u_i \frac{1 + \lambda h/2}{1 - \lambda h/2} \tag{2.20}$$

the obtained amplification factor is again indicating the requirements for stability. It can be seen that the trapezoidal method is unconditionally stable when the real part of λ is zero or negative. Again, this is expected for implicit methods. Solving by implicit methods provides stability which evidently comes at the cost of higher computational demand. In the next section, a linearization technique is described that avoids such computationally demanding iterations.

2.5 Linearizing Implicit Methods

For a linear initial value problem, like our model equation, the implicit methods can be solved straightforward. However, most problems are described by a non-linear equation, where $f(u,t)$ contains non-linear terms, like polynomials, exponentials, or harmonics. Solving u_{i+1} at each timestep then requires solving a non-linear algebraic equation, which is computationally much more demanding compared to solving an explicit scheme where the new value u_{i+1} can be calculated from the old value u_i.

Linearization is based on the concept that one does not need to solve u_{i+1} to full accuracy. As long as the term $f(u_{i+1}, t_{i+1})$ can be approximated at comparable accuracy as the implicit scheme itself, we will still obtain a solution to our numeric equation. We can obtain an estimate for $f(u_{i+1}, t_{i+1})$ by its Taylor series expansion:

$$f(u_{i+1}, t_{i+1}) = f(u_i, t_{i+1}) + (u_{i+1} - u_i) \left. \frac{\partial f}{\partial u} \right|_{u_i, t_{i+1}} + \frac{1}{2}(u_{i+1} - u_i)^2 \left. \frac{\partial^2 f}{\partial u^2} \right|_{u_i, t_{i+1}} + ... \tag{2.21}$$

We know that $u_{i+1} - u_i = h$, so taking only the first two terms will give an expression that is of the same accuracy compared to the Trapezoidal scheme, but is linear in u_{i+1} and can then be solved explicitly. For example, performing the linearization with the trapezoidal scheme, we obtain:

$$u_{i+1} = u_i + \frac{h}{2} \left(f(u_i, t_{i+1}) + (u_{i+1} - u_i) \left. \frac{\partial f}{\partial u} \right|_{u_i, t_{i+1}} + f(u_i, t_i) \right) \tag{2.22}$$

which then can be reorganized to give:

$$u_{i+1} = u_i + \frac{h}{2} \frac{f(u_i, t_{i+1}) + f(u_i, t_i)}{1 - \frac{h}{2} \left. \frac{\partial f}{\partial u} \right|_{u_i, t_{i+1}}} \tag{2.23}$$

so, the time marching can proceed as an explicit scheme, very simple and fast. The scheme is also unconditionally stable, so large timesteps are allowed. The final form depends on the chosen implicit scheme and requires to obtain the derivative of f with respect to u.

We can verify the stability using the model equation again:

$$\frac{du}{dt} = \lambda u = f \tag{2.24}$$

and thus

$$\frac{\partial f}{\partial u} = \lambda \tag{2.25}$$

which can be inserted into the linearized trapezoidal equation, and gives after reorganization:

$$u_{i+1} = u_i \left(1 + \frac{\lambda h}{1 - \frac{\lambda h}{2}} \right) \tag{2.26}$$

which is an identical amplification factor compared to the trapezoidal scheme, and thus gives the same accuracy.

2.6 Runge-Kutta

An general representation of a second order Runge-Kutta scheme is given by the following:

$$u_{i+1} = u_i + \left(1 - \frac{1}{2\alpha}\right) k_1 + \frac{1}{2\alpha} k_2 \tag{2.27}$$

$$k_1 = hf(u_i, t_i) \tag{2.28}$$

$$k_2 = hf(u_i + \alpha k_1, t_i + \alpha h) \tag{2.29}$$

Runge-Kutta methods correspond to methods that use intermediate points between the "old" i and "new" $i+1$ time. In fact, the method is explicitly moving forward in time using sub-time steps that are smaller. For example, a popular second order Runge-Kutta scheme, with $\alpha = 1/2$, is the following:

$$u_{i+1/2} = u_i + \frac{h}{2} f(u_i, t_i) \tag{2.30}$$

$$u_{i+1} = u_i + hf(u_{i+1/2}, t_{i+1/2}) \tag{2.31}$$

so, one first calculates the value for time $i+1/2$, followed by the evaluation at time $i+1$. We can apply this method to our model equation in order to obtain information on the stability. For the model equation we obtain:

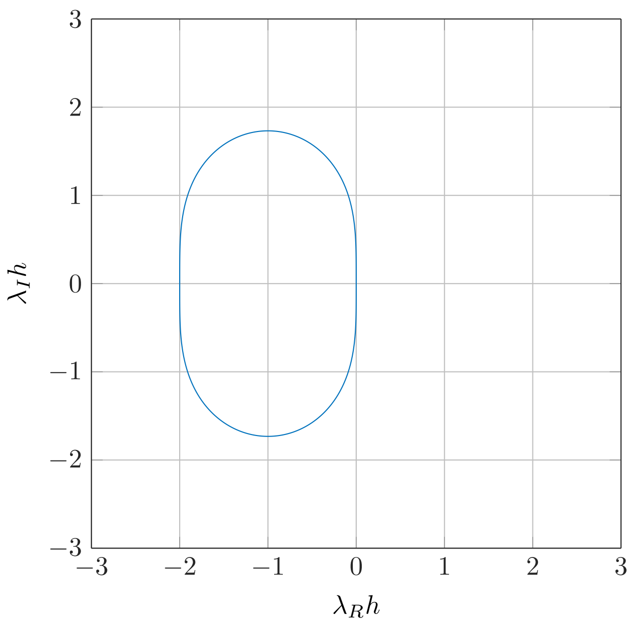


Figure 2.4: 2nd order Runga Kutta stability plot

$$u_{i+1} = u_i + h\lambda u_{i+1/2} = u_i + h\lambda \left(u_i + \frac{h}{2}\lambda u_i\right) = u_i \left(1 + \lambda h + \frac{\lambda^2 h^2}{2}\right) \tag{2.32}$$

for which the stability region is indicated in figure 2.4. We can see that the stability is increased for purely imaginary λ, which was not the case for the explicit Euler scheme. Also the amplification factor covers the analytic amplification factor up to the h^2 term so it is second order accurate.

Another popular Runge-Kutta scheme consist of even more intermediate steps. The so called 4th order RK is presented by:

$$u_{i+1} = u_i + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \tag{2.33}$$

$$k_1 = hf(u_i, t_i) \tag{2.34}$$

$$k_2 = hf(u_i + \frac{k_1}{2}, t_i + \frac{h}{2}) \tag{2.35}$$

$$k_3 = hf(u_i + \frac{k_2}{2}, t_i + \frac{h}{2}) \tag{2.36}$$

$$k_4 = hf(u_i + k_3, t_i + h) \tag{2.37}$$

Realize that this scheme also is an explicit scheme, but require more evaluations during the time stepping. As such i it somewhat more computationally expensive, but with the added benefit of increased stability. The stability plot is displayed in figure 2.5. The explicit scheme obtained by using the model equation results in:

$$u_{i+1} = u_i \left(1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24}\right) \tag{2.38}$$

which recovers up to the h^4 term from the analytic amplification factor and is thus accurate to the fourth order.

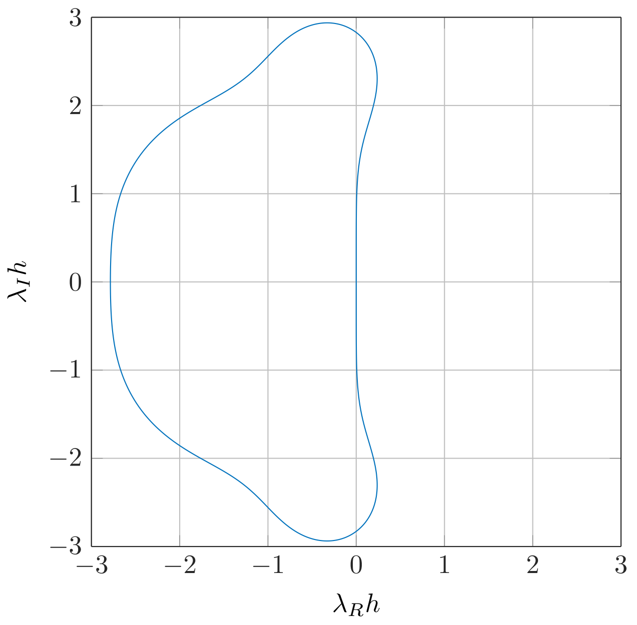


Figure 2.5: 4th order Runga Kutta stability plot

Chapter 3

Numerical Solutions of ODEs, BVP

3.1 Boundary Value Problems

So far, all ODEs we have dealt with considered a given starting value and marched forward from there, so called initial value problems. Here, we will deal with another type, namely boundary value problems. A boundary value problem can consist of the same differential equations, but must contain at least a second order derivative and conditions described at more than one location (or time). Such ODEs are therefore highly relevant to for instance steady diffusion problems, or solving Poisson, Laplace and Helmholtz equation. Notably, solving Poisson is crucial for solving pressure contributions, for instance, in the Navier Stokes equation.

3.2 Direct Method

One way to solve BVPs is by the direct method. This typically involves setting up a matrix that describes the system of equations. This is also one of the methods to solve elliptic equations, as we will discuss later. For now, we describe a 1D second order ODE:

$$\frac{d^2u}{dx^2} = f(u, u', x) \qquad\qquad u(0) = u_1 \qquad\qquad u(L) = u_L \tag{3.1}$$

We will focus only on the direct method to solve ODEs of this type. For that, we first discretize the ODE, for example using a second order scheme:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} = f(u_i, u_i', x_i) \tag{3.2}$$

When the system is linear, we can rewrite the discretized equation into the following form:

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = f_i \tag{3.3}$$

Since we have expressions for the both u_1 as well as u_N the coefficients a_i, b_i, and c_i (for i=2...$N-1$, the interior grid points only) can be represented by the following tridiagonal matrix:

$$\begin{bmatrix} b_2 & c_2 & 0 & \cdots & 0 \\ a_3 & b_3 & c_3 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & \cdots & 0 & a_{N-1} & b_{N-1} \end{bmatrix}$$

In Matlab, such matrices can be easily generated as follows:

```
A = diag(ones(N-2,b)) + diag(ones(N-3,c),1) + diag(ones(N-3,a),-1);
```

or even faster

```
A = gallery('tridiag',N-2,a,b,c);
```

The boundary conditions are still to be implementend via adjusting b_2, c_2 and f_2 for one side and a_{N-1}, b_{N-1} and f_{N-1} for the other side. If the boundary value is a constant, it will simply move to the right hand side. For example, the Dirichlet boundary conditions indicated above will result in:

$$\frac{u_3 - 2u_2}{\Delta x^2} = f_2 - \frac{u_1}{\Delta x^2} \tag{3.4}$$

$$\frac{-2u_{N-1} + u_{N-2}}{\Delta x^2} = f_{N-1} - \frac{u_L}{\Delta x^2} \tag{3.5}$$

We can thus compile the matrix for the interior points. This matrix, multiplied with a column vector $\vec{u}$ is then equal to $\vec{f}$. For example, for a 7 point grid, having thus 5 interior points, we obtain the following:

$$\begin{bmatrix} -2 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} \Delta x^2 f_2 - u_1 \\ \Delta x^2 f_3 \\ \Delta x^2 f_4 \\ \Delta x^2 f_5 \\ \Delta x^2 f_6 - u_7 \end{bmatrix}$$

Solving this BVP for $\vec{u}$ can be done using the matlab command $\vec{u} = A\backslash f$, where $\vec{A}\vec{u} = \vec{f}$. So, this method requires inversion of the matrix $\vec{A}$, which puts some requirements and desires regarding this matrix.

Implementing Neumann or Robin boundary conditions is also straightforward. If in the example above, the left boundary should be $u'(0) = \alpha$, this can be done using the Neumann boundary condition via the following finite difference approximation:

$$\frac{-3u_1 + 4u_2 - u_3}{2\Delta x} = \alpha \tag{3.6}$$

$$u_1 = \frac{2\Delta x\alpha - 4u_2 + u_3}{-3} \tag{3.7}$$

The obtained expression for u_1 can now be inserted into the discretized ODE 3.2:

$$\frac{u_1 - 2u_2 + u_3}{\Delta x^2} = f_2 \tag{3.8}$$

$$\frac{2\Delta x\alpha}{-3} + \frac{4u_2}{3} - \frac{u_3}{3} - 2u_2 + u_3 = \Delta x^2 f_2 \tag{3.9}$$

$$-u_2 + u_3 = \frac{3\Delta x^2 f_2}{2} + \Delta x\alpha \tag{3.10}$$

where it can be seen, that now also the matrix elements of the first row and the first element of $\vec{f}$ are changed:

$$\begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} 3\Delta x^2 f_2/2 + \Delta x\alpha \\ \Delta x^2 f_3 \\ \Delta x^2 f_4 \\ \Delta x^2 f_5 \\ \Delta x^2 f_6 - u_7 \end{bmatrix}$$

3.3 Iterative methods

In order to solve the differential equation iteratively, we use standard methods for initial value problems. We use a guess for the initial condition at one location and progress to see if it matches the condition at the other location. Remember that a boundary value problem must contain at least a second order derivative and conditions described at more than one location (or time).

The second-order differential 3.1 can be reduced to two first-order differentials by introducing $u' = v$:

$$\begin{cases} u' = v \\ v' = f(u, v, x) \end{cases} \tag{3.11}$$

with conditions $u(0) = u_1$ and $u(L) = u_L$. Thus, we still need a condition for $v(0)$ for which we use a guess. Then the system can be marched to $x = L$ to find $u(L)$ that can be compared to the condition u_L. We can iterate this process until we find a satisfactory agreement between $u(L)$ and u_L.

When the problem is linear, i.e. of form $u"(x) + A(x)u'(x) + B(x)u(x) = f(x)$, only two different initial guesses for $v(0) = u'(0)$ are required. Each initial guess will generate a solution of the equations: $u_1(x)$ and $u_2(x)$. Let's define the two obtained values for u_L as $u_1(L)$ and $u_2(L)$. It can be proven that:

$$u(x) = c_1 u_1(x) + c_2 u_2(x) \tag{3.12}$$

$$u_L = c_1 u_1(L) + c_2 u_2(L) \tag{3.13}$$

$$c_1 + c_2 = 1 \tag{3.14}$$

$$c_1 = \frac{u_L - u_2(L)}{u_1(L) - u_2(L)} \tag{3.15}$$

$$c_2 = \frac{u_1(L) - u_L}{u_1(L) - u_2(L)} \tag{3.16}$$

what in principle is done is known as the shooting method. By initializing the problem by a guessed first derivative, the angle of take-off has been set. By then analyzing the agreement between the other boundary condition with the obtained one, the angle of take-off can be adjusted.

When the problem is non-linear, iterative shooting can be used to propagate towards the solution. One example of such a root-finding method is the Secant method. As the problem is non-linear we can not simply obtain our solution by linear inter- or extrapolation of two random shots, as shown above. Instead, we are iteratively chosing a new shot, e.g. $u'(0)$ or $v(0)$, based on the outcome of two previous shots, $v(0)_1$ and $v(0)_2$:

$$v(0)_{new} = v(0)_1 + \frac{v(0)_2 - v(0)_1}{v(L)_2 - v(L)_1} \left(v^*(L) - v(L)_2\right) \tag{3.17}$$

in which $v^*(L)$ is the searched for boundary condition. After each iteration, v_2 becomes v_1 and v_{new} becomes v_2.

Chapter 4

Numerical solutions of PDEs

4.1 Semi-discretization

One can convert a partial differential equation, having more than 1 dimension variable like space and time, into a system of ordinary differential equations. For this, we use a finite difference scheme in all but one of the dimensions. Let's consider here the diffusion equation, with dimensions t and x:

$$\frac{\partial u}{\partial t} = \Gamma \frac{\partial^2 u}{\partial x^2} \tag{4.1}$$

with a given initial $u(x, 0)$ and boundary conditions $u(0, t)$ and $u(L, t)$. We can now discretize one of the two dimensions with a discretization scheme of choice. Evidently, the chosen discretization scheme will affect the resulting stability as we will see later. For now, we discretize the spatial dimension x using a central second-order difference scheme;

$$\frac{du_i}{dt} = \Gamma \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{4.2}$$

which can therefore be represented in matrix form:

$$\frac{d\vec{u}}{dt} = \vec{A}\vec{u} \tag{4.3}$$

with the tridiagonal matrix for the interior points:

$$\vec{A} = \frac{\Gamma}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix}$$

So now the original PDE has been semi-discretized to a system of ODEs. This system can then be solved with known methods, like explicit Euler, Runge-Kutta, etc. Evidently, the choice of discretization method for the remaining dimension, t in this case, should be made such that the numerical scheme is stable. In the next section, we will introduce the modified wavenumber analysis which provides information regarding the stability of the discretization scheme.

4.2 Modified wavenumber analysis

4.2.1 Stability

We will analyze the stability of numerical solutions by the modified wavenumber analysis. This analysis will nicely connect to the model problem analysis we have performed for

ODEs. This stability analysis will give guidance to the required time step size, as before. The modified wavenumber analysis assumes a solution of the following form:

$$u(x,t) = \psi(t)e^{ikx} \tag{4.4}$$

with wavenumber k. So the solution consists of the product of a spatial and a temporal function. Let's now substitute this, as a relevant example, into the 1D heat equation (or diffusion equation):

$$\frac{\partial u}{\partial t} = \Gamma \frac{\partial^2 u}{\partial x^2} \tag{4.5}$$

giving:

$$\frac{d\psi}{dt} = -\Gamma k^2 \psi(t) \tag{4.6}$$

note the similarity with the model problem, where $\lambda = -\Gamma k^2$. Equation (4.6) is the exact form, and k is the exact wavenumber. If we are interested in solving the heat equation, we could consider the semi-discretized scheme, where the spatial derivative has been approximated by the second order central difference:

$$\frac{du_i}{dt} = \Gamma \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} \tag{4.7}$$

We now insert $u_i = \psi(t)e^{ikx_i}$ realize that $u_{i+1} = \psi(t)e^{ikx_{i+1}} = \psi(t)e^{ikx_i}e^{ik\Delta x}$, and analogous for u_{i-1}, giving:

$$\frac{d\psi}{dt} = \frac{\Gamma}{\Delta x^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\psi(t) \tag{4.8}$$

or

$$\frac{d\psi}{dt} = -\Gamma k'^2 \psi(t) \tag{4.9}$$

with the modified wavenumber k'. We see that our spatial discretization scheme produces a unique modified wavenumber. We also recognize that equation (4.9) is identical to the model equation we used extensively to study the stability of ODEs, where $\lambda = -\Gamma k'^2$. Thus we have:

$$\lambda = \frac{\Gamma}{\Delta x^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right) \tag{4.10}$$

which is equivalent to

$$\lambda = \frac{2\Gamma}{\Delta x^2}\left(\cos(k\Delta x) - 1\right) \tag{4.11}$$

Now, based on the time discretization scheme that we chose, we will find the condition for the time step. Recall, for instance, that for the explicit Euler time stepping, we have the condition for stability:

$$\Delta t \leq \frac{2}{|\lambda|} \tag{4.12}$$

so, based on the spatial step size, Δx, we find a maximum time step size Δt. We further see that the diffusion equation results in a modified wavenumber that is real. Note that the time step is quadratic dependent on the spatial step. This is a severe restriction that requires strong time step refinement and thus computational cost.

Let's now consider the modified wavenumber analysis for the linear convection equation:

$$\frac{\partial u}{\partial t} = c\frac{\partial u}{\partial x} \tag{4.13}$$

with the exact solution of the wavenumber analysis:

$$\frac{d\psi}{dt} = ikc\psi(t) \tag{4.14}$$

If we semi-discretize this by using the central difference for the spatial derivative, we find:

$$\frac{du_i}{dt} = c\frac{u_{i+1} - u_{i-1}}{2\Delta x} \tag{4.15}$$

inserting the assumed solution $\psi(t)e^{ikx_i}$ for u_i gives:

$$\frac{d\psi}{dt} = \frac{c}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right)\psi \tag{4.16}$$

or rewritten as:

$$\frac{d\psi}{dt} = \frac{ic\sin(k\Delta x)}{\Delta x}\psi \tag{4.17}$$

which evidently corresponds to a purely imaginary λ. The fact that λ is purely imaginary, evidently means that explicit Euler time advancement will not work. Instead, an implicit time advancing scheme could be used.

4.2.2 Implicit time advancement

The indicated problems for the explicit time advancement above, resulting in time-step restrictions or instability, has resulted in use of implicit methods for parabolic equations like the generic transport equation. We will now consider first the implicit Euler, or backward Euler, method for time advancement. When we use this for the 1D heat equation (4.5), with a second order central difference in space, we get:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{\Gamma}{\Delta x^2}\left(u_{i+1}^{n+1} - 2u_i^{n+1} - u_{i-1}^{n+1}\right) \tag{4.18}$$

when collecting all the unknowns on the left, we get

$$u_i^{n+1} - \frac{\Gamma\Delta t}{\Delta x^2}\left(u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}\right) = u_i^n \tag{4.19}$$

In order to solve, this requires a set of linear algebraic equations to be solved at each time step. To investigate the stability, we now insert for $u_i^n = \psi^n e^{ikx_i}$ and $u_i^{n+1} = \psi^{n+1}e^{ikx_i}$ and corresponding terms for $i+1$ and $i-1$. We finally obtain after dividing by e^{ikx_i}

$$\psi^{n+1}\left(1 - \frac{2\Gamma\Delta t}{\Delta x^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\right) = \psi^n \tag{4.20}$$

which can also be written as

$$\psi^{n+1} = \frac{1}{1 + \frac{2\Gamma\Delta t}{\Delta x^2}(1 - \cos k\Delta x)}\psi^n \tag{4.21}$$

from which the amplification factor can be easily seen. This factor is always smaller than 1 and therefore the scheme is unconditionally stable. We previously derived the amplification factor for the implicit Euler scheme as $1/(1 - \lambda\Delta t)$, showing that the scheme is unconditionally stable. However, one should realize that for very large time steps, the amplification factor vanishes and the solution would be less accurate, despite being stable.

When performing a similar analysis for the linear convection problem, using an implicit Euler time advancement scheme and a backward difference scheme for space, we have:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + c\frac{u_i^{n+1} - u_{i-1}^{n+1}}{\Delta x} = 0 \tag{4.22}$$

After applying the same modified wavenumber analysis, we eventually get

$$\psi^{n+1} = \frac{1}{1 + \frac{c\Delta t}{\Delta x}(1 + i\sin k\Delta x - \cos k\Delta x)}\psi^n \tag{4.23}$$

The amplification factor indicates stability when c is positive only. Thus, compared to the explicit Euler time advancement scheme, the implicit scheme is stable also for large Δt as long as the velocity $c > 0$.

4.2.3 Accuracy

We can also obtain information regarding the accuracy using the modified wavenumber. Let us look again at the above example, the linear convection equation, where we obtained a modified wavenumber for the central difference scheme:

$$k' = \frac{\sin(k\Delta x)}{\Delta x} \tag{4.24}$$

We can use the Taylor expansion for $\sin k\Delta x$ to obtain the following series:

$$k' = \frac{1}{\Delta x}\left(k\Delta x - \frac{(k\Delta x)^3}{6} + O(\Delta x^5)\right) = k\left(1 - \frac{(k\Delta x)^2}{6} + O(\Delta x^4)\right) \tag{4.25}$$

which gives us the result that the central difference scheme is second order accurate.

Chapter 5

Generic Transport Equation

The generic transport equation is characterized by 4 terms, including accumulation, advection, diffusion, and a source term. The 1D version, with constant velocity, constant fluid properties, and no source term reads:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \Gamma \frac{\partial^2 u}{\partial x^2} \tag{5.1}$$

where α is the diffusion constant, and c the constant velocity. In this form, u could be the temperature or concentration when considering the energy or mass transport equations, respectively. This equation also presents some similarities to the Navier Stokes equation, but with some important differences as we will see later.

5.1 Modified Wavenumber Analysis

5.1.1 Central difference scheme

We can choose to discretize each of the terms in above equation by a method of choice. If we use the central differencing scheme for the spatial derivatives we obtain:

$$\frac{\partial u}{\partial t} = -c\frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + \Gamma \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{5.2}$$

Evidently, the choice of scheme will be important regarding the stability and accuracy of the numerical solution. We have treated both the diffusion term as well as the linear convection term separately before. Recalling, the modified wavenumber analysis assumes a solution of the following form:

$$u(x,t) = \psi(t)e^{ikx} \tag{5.3}$$

Substituting this assumed solution in the generic transport equation 5.1 given above, eventually results in:

$$\frac{d\psi}{dt} = -ikc\psi(t) - \Gamma k^2 \psi(t) \tag{5.4}$$

which is the exact form with the exact wavenumber k. If we now use the discretized equation, with central differences for spatial derivatives as indicated above, we obtain the following:

$$\frac{d\psi}{dt} = -\frac{c}{2\Delta x}\left(e^{ik\Delta x} - e^{-ik\Delta x}\right)\psi + \frac{\Gamma}{\Delta x^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\psi \tag{5.5}$$

which is equivalent to:

$$\frac{d\psi}{dt} = \left(-\frac{ic\sin(k\Delta x)}{\Delta x} + \frac{2\Gamma}{\Delta x^2}\left(\cos(k\Delta x) - 1\right)\right)\psi \tag{5.6}$$

As expected, λ, the part between brackets, now consists of both a real and imaginary part. We recall that for an explicit Euler scheme, which is the one proposed above for the time advancement, the numerical solution will be stable for:

$$\Delta t \leq \frac{2}{|\lambda|} \tag{5.7}$$

If there were no diffusion in the generic transport equation, i.e. $\Gamma = 0$, a purely imaginary λ would make the explicit Euler time advancement useless, as it is unstable. On the other hand, in the absence of any advection, i.e. $c = 0$, we find that the numerical solution is stable for

$$\Delta t \leq \frac{\Delta x^2}{2\Gamma} \tag{5.8}$$

which means that any refinement in spatial grid size will require a quadratic refinement in time. So, the choice for the central difference scheme for the spatial derivatives results in some issues regarding stability and grid sizes, when solving generic transport equations.

5.1.2 Backwards Difference Scheme

Some improvements over the mentioned stability and accuracy of the central difference scheme can be tackled by using a backward difference (or upwind) scheme. Note that this scheme is of lower accuracy compared to the central difference scheme. We now get the following expression in the modified wavenumber analysis:

$$\frac{d\psi}{dt} = -\frac{c}{\Delta x}\left(1 - e^{-ik\Delta x}\right)\psi + \frac{\Gamma}{\Delta x^2}\left(e^{ik\Delta x} - 2 + e^{-ik\Delta x}\right)\psi \tag{5.9}$$

being equivalent to:

$$\frac{d\psi}{dt} = \left(-\frac{c}{\Delta x}\left(i\sin(k\Delta x) - \cos(k\Delta x) + 1\right) + \frac{2\Gamma}{\Delta x^2}\left(\cos(k\Delta x) - 1\right)\right)\psi \tag{5.10}$$

which shows that even in the absence of diffusion, λ will not be purely imaginary. Hence, an explicit Euler time advancement scheme is possible, when satifying the following condition:

$$\Delta t \leq \frac{1}{\frac{2\Gamma}{\Delta x^2} + \frac{c}{\Delta x}} \tag{5.11}$$

Chapter 6

Elliptic equations

Elliptic equations are treated in detail separately, because of their importance in the incompressible Navier Stokes equation, among others. We have already considered boundary value problems in 1D before. The elliptic equation discussed here concern a boundary value problem in 2 dimensions.

6.1 2D Laplace and Poisson equation

$$\nabla^2 p = -q \tag{6.1}$$

where $q = 0$ corresponds to the Laplace equation. For $q \neq 0$ the equation is known as Poisson's equation. This equation has a broad range of applicability, including electrostatics, mechanical engineering and physics. Related to fluid dynamics, the Poisson equation is very relevant for solving the pressure gradient distribution, needed in the Navier Stokes equation. Discretizing the equation using a 2nd order central scheme, gives:

$$\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} = -q_{i,j} \tag{6.2}$$

We see that to evaluate p at position i, j the values on all sides are required. Elliptic equations are described for a closed domain, where the boundary conditions are specified.

6.2 Direct Method

Solving for $p_{i,j}$ requires the values from 4 surrounding grid points (east, west, north, south). For the interior points this is straightforward obtained. The boundary points require some special attention. As with the 1D BVP, a fixed boundary condition value (Dirichlet condition) will simply move to the right hand side of the equations, adding to $q_{i,j}$.

This is best described in a system of linear algebraic equations. For example, a 5x5 grid point system with $\Delta x = \Delta y = \Delta$, gives for the interior points the following:

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \begin{bmatrix} p_{2,2} \\ p_{3,2} \\ p_{4,2} \\ p_{2,3} \\ p_{3,3} \\ p_{4,3} \\ p_{2,4} \\ p_{3,4} \\ p_{4,4} \end{bmatrix} = \begin{bmatrix} -\Delta^2 q_{2,2} - p_{1,2} - p_{2,1} \\ -\Delta^2 q_{3,2} - p_{3,1} \\ -\Delta^2 q_{4,2} - p_{5,2} - p_{4,1} \\ -\Delta^2 q_{2,3} - p_{1,3} \\ -\Delta^2 q_{3,3} \\ -\Delta^2 q_{4,3} - p_{5,3} \\ -\Delta^2 q_{2,4} - p_{1,4} - p_{2,5} \\ -\Delta^2 q_{3,4} - p_{3,5} \\ -\Delta^2 q_{4,4} - p_{5,4} - p_{4,5} \end{bmatrix}$$

If the boundary is described by Neuman or Robin condition, the grid point at the boundary can be described in terms of the adjacent grid points. For example, the Neuman boundary condition given by:

$$\frac{\partial p}{\partial x} = f \tag{6.3}$$

with f being a constant, can be discretized by:

$$\frac{p_{2,j} - p_{1,j}}{\Delta x} = f \tag{6.4}$$

resulting in

$$p_{1,j} = p_{2,j} - f\Delta x \tag{6.5}$$

which can be inserted into the matrix above, where $p_{2,j}$ will then move then to the left hand side, as it is not a constant. This will effectively alter the coefficient in the matrix (-4 would become -3 in places where $p_{1,j}$ was present).

6.3 Iterative methods

The same equation as above can be solved using an iterative technique. For this, we guess an initial solution and use an algorithm to iterate and improve the solution. The most simple way of doing this, is by the Point Jacobi method. Starting from the "guess" solution, we update each point iteratively by increasing n:

$$p_{i,j}^{n+1} = \frac{1}{4}\left(p_{i-1,j}^{n} + p_{i+1,j}^{n} + p_{i,j-1}^{n} + p_{i,j+1}^{n}\right) + \frac{\Delta^2 q_{i,j}}{4} \tag{6.6}$$

The method is simple and straightforward, but it is also slowly converging. One important observation is that to compute the value at position i, j, only values from the previous iteration are used. When arriving at point i, j, evidently the point at $i-1, j$ and $i, j-1$ have already been updated in the current iteration. In fact, using the already updated values for these points does improve the convergence. This iteration method is known as Gauss-Seidel:

$$p_{i,j}^{n+1} = \frac{1}{4}\left(p_{i-1,j}^{n+1} + p_{i+1,j}^{n} + p_{i,j-1}^{n+1} + p_{i,j+1}^{n}\right) + \frac{\Delta^2 q_{i,j}}{4} \tag{6.7}$$

Another very successful iterative method is the Successive Over Relaxation (SOR). It is aimed at increasing the rate of convergence of the Gauss-Seidel method by introducing a so called relaxation parameter, ω. If we define the Gauss-Seidel algorithm above as $p_{i,j}^{GS}$, the SOR is given by:

$$p_{i,j}^{n+1} = p_{i,j}^{n} + \omega\left(p_{i,j}^{GS} - p_{i,j}^{n}\right) \tag{6.8}$$

Although beyond the scope, the optimal value for the relaxation parameter is usually between 1.7 and 1.9.

Chapter 7

Non-uniform and staggered grids

7.1 non-dimensional grids

The discretization so far is performed by using a uniform grid with step sizes equal to Δt, Δx or Δy. One is typically free to chose the grid sizes in each direction. As seen above, the error associated with different discretization schemes, is dependent in these grid step sizes. First, the use of a scaled grid will be briefly introduced followed by the use of non-uniform grids. A scaled grid is particularly useful if the physical domain for the model is strongly anisotropic, e.g. a long narrow tube. If we for instance consider convection diffusion in 2D,

$$u\frac{\partial c}{\partial x} + v\frac{\partial c}{\partial y} = D\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right) \tag{7.1}$$

we can scale the domain by introducing

$$\tilde{x} = \frac{x}{L},\ \tilde{y} = \frac{y}{H} \tag{7.2}$$

where L and H are the length and height of the domain respectively. Introducing this into the species equation gives then the following scaled equation

$$\frac{u}{L}\frac{\partial c}{\partial \tilde{x}} + \frac{v}{H}\frac{\partial c}{\partial \tilde{y}} = \frac{D}{L^2}\frac{\partial^2 c}{\partial \tilde{x}^2} + \frac{D}{H^2}\frac{\partial^2 c}{\partial \tilde{y}^2} \tag{7.3}$$

One can see that it effectively introduces anisotropic velocity ($\frac{u}{L}$ and $\frac{v}{H}$) and diffusion coefficients ($\frac{D}{L^2}$ and $\frac{D}{H^2}$). The domain is now [0..1] for both $\tilde{x}$ and $\tilde{y}$. Similarly, one can scale the momentum balance which introduces anisotropic viscosity terms analogous.

7.2 coordinate transformation

In many cases, the outcome of the solution is not uniform, i.e. some regions display steep gradients while other regions are nearly constant. For instance, the velocity profile near a flat wall will have a strong gradient in velocity close to the wall and a constant velocity further away. A steep gradient means that the value changes strongly per step size. Hence, it will generally be better to reduce the step size in order to approach the exact solution more closely. On the other hand, reducing the step size will result in enhanced computational effort needed to solve the equations. The solution to this paradigm is the use of non-uniform grids.

A non-uniform grid has step sizes that depend on the actual location in the domain. The non-uniformity has to be known a priori. Grid refinement is then introduced in areas where large gradients are expected. In fluid dynamics, this is typically near solid walls and objects of flow obstruction and manipulation (e.g. bends, nozzles, corners).

The non-uniform grid is based on a transformation of the original grid of (x, y) to (α, β). There are many ways of defining the grid refinement. An example for one is shown in the figure below (7.1).

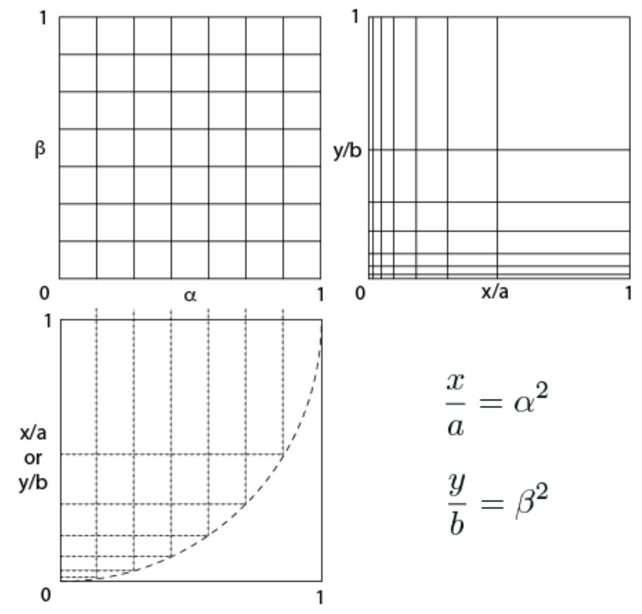


Figure 7.1: Illustration of grid refinement, using $\frac{x}{a} = \alpha^2$.

The non-uniform grid imposes that the step sizes are different at at different grid points.

$$x = f(\alpha),\ y = f(\beta) \tag{7.4}$$

The differential equations and the boundary conditions also need to be transformed. For a first order partial derivative of u this is:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \alpha}\frac{\partial \alpha}{\partial x} \tag{7.5}$$
$$= \frac{\partial u}{\partial \alpha}\frac{1}{\frac{\partial x}{\partial \alpha}} \tag{7.6}$$
$$= \frac{\partial u}{\partial \alpha}\frac{1}{x'(\alpha)} \tag{7.7}$$

And for a second order derivative this becomes:

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{x'(\alpha)}\frac{\partial}{\partial \alpha}\left(\frac{1}{x'(\alpha)}\frac{\partial u}{\partial \alpha}\right) \tag{7.8}$$

Considering the example where the grid refinement is given by $x = a\alpha^2$ this results in the following first and second order derivatives:

$$\frac{\partial u}{\partial x} = \frac{\partial u}{\partial \alpha}\frac{1}{2a\alpha} \tag{7.9}$$
$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{2a\alpha}\frac{\partial}{\partial \alpha}\left(\frac{1}{2a\alpha}\frac{\partial u}{\partial \alpha}\right) \tag{7.10}$$

A common used coordinate transformation involves Polar coordinates. Polar coordinates relate to Cartesian coordinates as $x = r\cos\theta$ and $y = r\sin\theta$ (note the location of

$\theta = 0$ coinciding with the x-axis, and $\theta = \pi/2$ with the positive y-axis). When considering a round object, for instance, the Polar coordinate system is very useful to capture the exact spherical geometry that would require extremely fine grids to approach in a common Cartesian coordinate system.

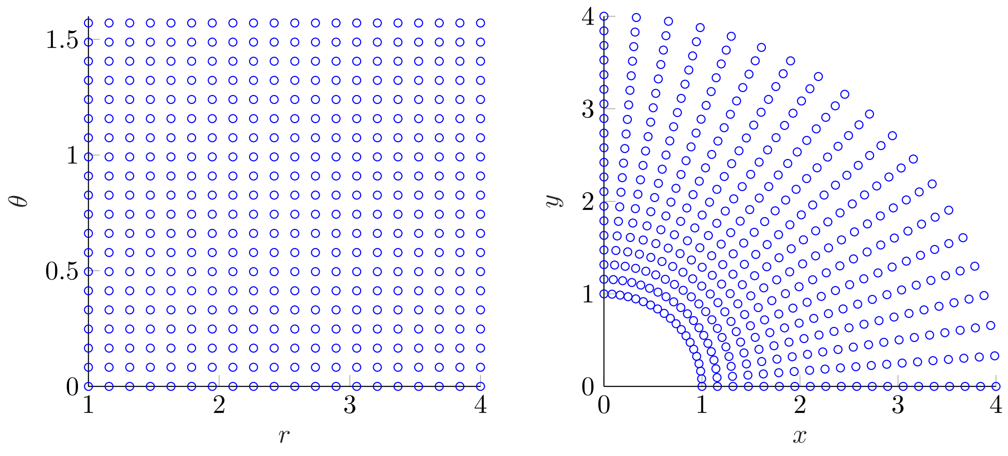


Figure 7.2: Polar grid transformation

7.3 Staggered grids

Staggered grids are extremely useful and provide advantages in terms of stability. Also, it represent the possibility for a better physical picture of the problem. For instance, when evaluating pressure and velocity, it is intuitive to define the pressure points at intermediate locations with respect to the velocity points. The velocity is directly dependent on the pressure gradient, and not the actual pressure in the incompressible Navier Stokes equation. The same holds for defining flux at intermediate locations with respect to concentration.

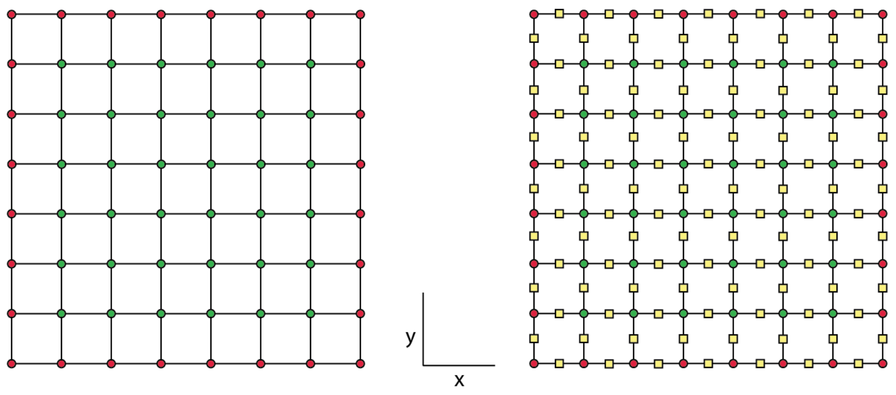


Figure 7.3: Illustration of a collocated grid (left), and a staggered grid (right). In the collocated grid, all variables are obtained at the same locations. In a staggered grid, some variables are obtained at the round points, some at the square points.

For an $n_x \times n_y$ elements geometry, the collocation grid leads to $(n_x+1)\times(n_y+1)$ grid points for all variables. For the staggered grid, this deserves a bit more explanation. The intermediate points, squares in figure 7.3, will only be influenced by their neighbours in one direction, while the internal grid points (green circles) will be influenced by their neighbours in all directions.

When solving the momentum equations, for example, the velocity terms are defined at the intermediate points. For the x-velocity u, we obtain an $n_x \times (n_y + 1)$ grid, and for the y-velocity v an $(n_x + 1) \times n_y$ grid. The pressure will be defined at $(n_x + 1) \times (n_y + 1)$ grid points. In this way, the velocity in a certain direction is governed by the pressure gradient in that very same direction by the neighboring pressure grid points.

Chapter 8

Navier Stokes

Although the Navier-Stokes equations display many some similarities to generic transport equations for mass and energy, the difference needs special attention in their treatment. First of all, the accumulation (unsteady), advection, and diffusion terms are quite similar to their counterparts in the generic equation, but the momentum equations are vector equations. Furthermore, the momentum equation contains a contribution from the pressure. The pressure is closely related to momentum transfer.

8.1 Compressible Flow

There are a rich set of interesting chemical engineering problems related to the transport of compressible fluids. For example, transport of natural gas in pipelines is an example of the flow of a compressible material. Another example is flow of air through a converging-diverging nozzle (jet engine), where the compressibility can give rise to very interesting behavior (supersonic flow).

Writing the momentum balance for a compressible fluid, one finds:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla p + \mu\nabla^2\boldsymbol{u} + \frac{\mu}{3}\nabla(\nabla\cdot\boldsymbol{u}) + \sum \boldsymbol{F}^{\text{body forces}} \tag{8.1}$$

with $\boldsymbol{u}$ is the velocity vector $[u, v]$. The continuity equation is:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot\rho\boldsymbol{u} = 0 \tag{8.2}$$

Notice that there are several terms related to the divergence of velocity (continuity of mass). Specifically, if $\nabla\cdot\boldsymbol{u}$ is zero, then the compressible Navier-Stokes equation reduces to the incompressible form (density is constant), which is discussed in the next section. An additional term in the momentum balance occurs, this is related to the viscous work of volume change. Treating this "bulk viscosity" coefficient as 0 is an approximation (see Batchelor, 1973 or other references for more details).

If the fluid is an ideal gas, this allows for direct substitution of the pressure in terms of density,

$$\rho = \frac{p}{RT} \tag{8.3}$$

where R is the specific gas constant (universal gas constant divided by the molar mass of the gas). If the system is isothermal, then this is a fully constitutive relationship to solve the momentum/continuity simultaneously. If there are temperature effects from friction or expansion/contraction (Joule-Thomson type effects), the energy balance must also be solved for which can be a considerable complication.

8.2 Incompressible Flow

For incompressible flow, the connection between pressure and density is not trivial, as there is no direct way of expressing the pressure in terms of velocity vectors. Instead, it is the pressure gradient that is relevant and pressure can be set at arbitrary levels (as liquid properties generally do not depend on pressure). Realize that this is completely different from compressible flow, where the pressure and density can be related in a direct manner. Recalling, the Navier Stokes for an incompressible fluid expressed in vector notation reads:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t}+\boldsymbol{u}\cdot\nabla\boldsymbol{u}\right)=-\nabla p+\mu\nabla^2\boldsymbol{u}+\sum\boldsymbol{F}^{\text{body forces}} \tag{8.4}$$

And the continuity equation:

$$\nabla\cdot\boldsymbol{u}=0 \tag{8.5}$$

There is, however, no obvious way to couple to velocity and the pressure. In case of a compressible flow, the equation of state provides a relation between the density and pressure (e.g. ideal gas law). In the explicit Euler time advancement method described below, we will see how to connect the pressure and velocity field.

8.3 Explicit Euler Time Advancement

The discretization of the Navier Stokes equation first deserves some explanation. Isolating all the temporal terms on the left side and spatial terms on the right (without any body forces) side we get:

$$\frac{\partial \boldsymbol{u}}{\partial t}=-\frac{1}{\rho}\nabla p-\boldsymbol{u}\cdot\nabla\boldsymbol{u}+\nu\nabla^2\boldsymbol{u} \tag{8.6}$$

The convective and diffusive terms of the equation can be spatially discretized by known methods. Here, we replace these terms by RHS. Using an explicit Euler time advancement scheme, we have:

$$u_i^{n+1}-u_i^n=\Delta t\left(\text{RHS}^n-\frac{1}{\rho}\frac{p_{i+1,j}^{n+1}-p_{i-1,j}^{n+1}}{2\Delta x}\right) \tag{8.7}$$

Here we see that RHS is based on the known solution from the previous time n, but that the pressure p must be calculated for time $n+1$. This is to guarantee that the new computed velocity at $n+1$ is divergence free. We can solve this by a so-called predictor-corrector method. In the first step, the predictor step, we will compute the velocity at an intermediate time $*$, based on only the RHS terms and neglecting the pressure term:

$$\frac{\boldsymbol{u}^*-\boldsymbol{u}^n}{\Delta t}=\text{RHS}^n=-\boldsymbol{u}^n\cdot\nabla\boldsymbol{u}^n+\nu\nabla^2\boldsymbol{u}^n \tag{8.8}$$

With this first step we can obtain the velocity $\boldsymbol{u}^*$ via the known methods using for instance central discretization for the convective term and second order central discretization for the diffusive term. In the second step, the corrector step, we obtain the velocity at time $n+1$ by including the pressure term:

$$\frac{\boldsymbol{u}^{n+1}-\boldsymbol{u}^*}{\Delta t}=-\frac{1}{\rho}\nabla p^{n+1} \tag{8.9}$$

It is easy to see that the addition of 8.8 and 8.9 leads to 8.7. We can obtain the pressure distribution by taking the diverge of the equation 8.9 above, and imposing $\nabla\cdot\boldsymbol{u}^{n+1}=0$, leading to

$$\nabla^2 p^{n+1}=\frac{\rho}{\Delta t}\nabla\cdot\boldsymbol{u}^* \tag{8.10}$$

which is called the pressure Poisson equation. This will assure that the velocity field at $n+1$ is divergence free.

Part II

Case Descriptions

Case 1

Initial Value Problems

In this assignment we will solve a system of first order ODEs, with a given initial condition. You will solve these ODEs while probing the accuracy of the solution, the stability and the computation cost.

1.1 The SIR model, an IVP

The SIR model is a basic model used to describe the outbreak and spread of a disease. It contains the time-dependent evolution of susceptibles S, infectives I, and recovered R. It is based on simple assumptions that the total population $(S + I + R)$ remains constant.

$$\frac{dS}{dt} = -rSI \tag{1.1}$$

$$\frac{dI}{dt} = rSI - aI \tag{1.2}$$

$$\frac{dR}{dt} = aI \tag{1.3}$$

where r and a are rate constants for infection and recovery, respectively. You may recognize that this model is equivalent to reaction kinetics, for the synthesis of R from S, with intermediate I. Realize that the equations remain the same when S, I, and R represent fractions of the total population. The set of ODEs can be solved, using an initial condition, defining S_0, I_0, and R_0.

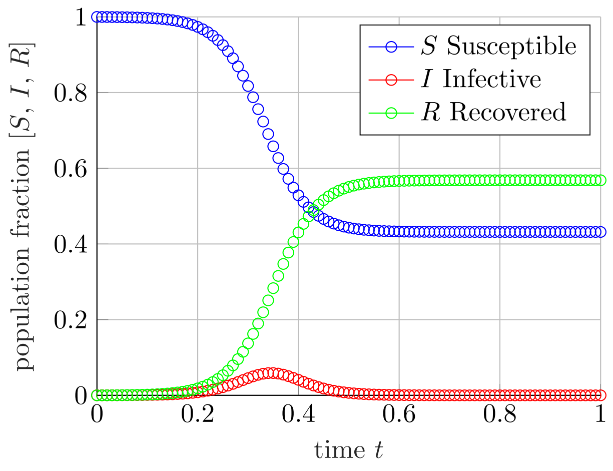


Figure 1.1: Time evolution according to the SIR model.

1. Discretize the set of equations, using the explicit Euler scheme as well as the implicit Euler scheme.

2. Start with the explicit Euler scheme and code this into the starting Matlab file. Use $r = 1$ and $a = 0.5$ as infection and recovery rate constants.

3. Investigate the stability of the scheme by altering the time step size.

4. Investigate the accuracy of the scheme by comparing the maximum of infected I_{max} between your scheme and the analytic maximum.

5. Do this for different time step sizes and analyze how the error and step size are scaling (try on log-log plot).

6. Try another explicit scheme (e.g. Runge Kutta) and investigate its stability and accuracy.

7. Code the implicit Euler scheme. Realize, that at every time step you need to solve for $S(i + 1)$ using a solver like Matlabs 'fsolve'.

8. Analyze the stability, accuracy, and computation time for this method and compare with the other methods.

9. Use Matlabs own ODE45 solver as well, for comparison to your own schemes.

10. To understand the spread of a disease some better, see what changing I_0 and some other parameters does.

11. Once the fraction of infectives has reached 4 %, the government decides stricter quarantine rules, and stimulates social distancing. This results in a reduction for r to 0.6. Investigate how this will change the disease spread (e.g. total number of people infected/recovered). Also investigate the effect of more or less stricter rules by adjusting r during the quarantine period.

Case 2

Boundary Value Problems

2.1 1D Poisson equation, a BVP

The potential distribution through an electrolyte can be described by the well-known Poisson equation:

$$\frac{d^2\phi}{dx^2} = -k\left(c_+(x) - c_-(x)\right) \tag{2.1}$$

with c_+ and c_- being the cation and anion concentrations, respectively, and k some proportionality constant. When we assume that ions are distributed according to the Boltzmann distribution, we take the concentration of each ion to be proportional to $e^{-z\phi}$, with ion valency z. This results in a non-linear ODE,

$$\frac{d^2\phi}{dx^2} = k'\left(e^{\phi} - e^{-\phi}\right) \tag{2.2}$$

However, for some conditions, notably low potentials ϕ, we can approximate the above equation by realizing $\left(e^{\phi} - e^{-\phi}\right) \sim 2\phi$, resulting then in a linear ODE. We will investigate ways to solve the Poisson equation using the direct method, the shooting method, and the build-in solver. For this, we search for a solution of ϕ on the domain of [0,1], with set boundary potential values for $\phi(0)$ and $\phi(1)$.

1. Discretize, using the second-order difference scheme. Do this for both for the non-linear form as well as the linear form.
2. Show that the second-order difference scheme can also be obtained by a backward difference of a forward difference, as $\frac{d^2c}{dx^2} = \frac{d}{dx}\left(\frac{dc}{dx}\right)$.
3. Break the second-order problem into two first-order equations. These are required when using the built-in solver or the shooting method.
4. Solve the boundary value problem with the matlab built-in solver *bvp4c*. For this solver, you need to write the second-order ODE as a system of first-order ODEs. Investigate the influence of the boundary potential values and the value of the proportionality constant k'.
5. The discretized equation for the interior grid points can be presented by $\vec{A} \cdot \vec{c} = \vec{b}$. Generate the matrix and solve the Poisson equation by the direct method. Analyze the differences between the direct method and the build-in method.
6. Use the shooting method to solve the Poisson equation. First, try this with the linear ODE form, then with the non-linear form.

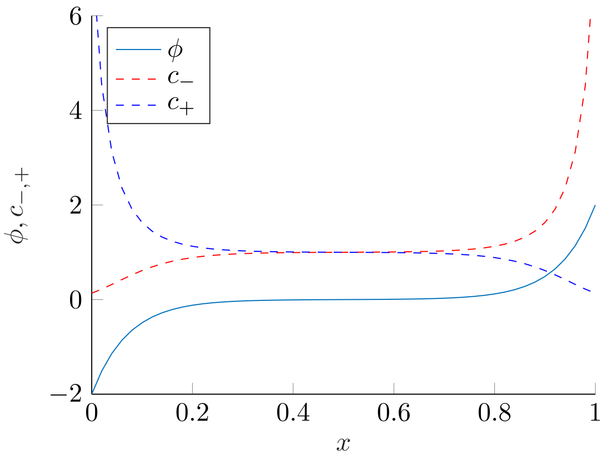


Figure 2.1: Potential distribution between 2 electrodes, solved using the Poisson equation. The resulting ion concentration profiles are also shown.

Case 3

Numerical Solutions of PDEs

In this case, we are going to solve some differential equations in matlab, via the discretization method. We will use explicit Euler time advancement in all these cases, and will look at the influence of different spatial discretization schemes on the stability of the numerical solution.

3.1 1D Linear Convection Equation

As a start, a simple 1D linear convection equation is solved by a provided matlab code. The equation is given as:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = 0 \tag{3.1}$$

where c is the variable value of interest (e.g. concentration), t is time, u is the linear velocity, and x is space. This equation needs to be discretized in space and time. For this discretization we use a grid of i by n, where i is the index of grid in x, and n is the index of grid in t. For the numerical scheme, we use explicit Euler for time t advancement, and a backward difference scheme in space x:

$$\frac{c_i^{n+1} - c_i^n}{\Delta t} + u\frac{c_i^n - c_{i-1}^n}{\Delta x} \tag{3.2}$$

This equation can be transposed to isolate the variable value at t_{n+1} from variable values at t_n:

$$c_i^{n+1} = c_i^n - u\frac{\Delta t}{\Delta x}\left(c_i^n - c_{i-1}^n\right) \tag{3.3}$$

Use the following initial conditions:

$$c(x,0) = e^{-10(5-x)^2} \tag{3.4}$$

We will use Matlab to numerically solve the discretized and transposed equation. The code provided solves the first 1D linear convection problem and can be used as starting structure for all other problems.

1. Understand and run the Matlab code and vary the step sizes Δx and Δt. Is the numerical solution stable?
2. What does changing the value of the linear velocity u do? Also check for negative velocity.
3. Discretize according to a central difference (CD) for x and adjust the code. Is the numerical solution stable?
4. Analyse your results using the modified wavenumber.
5. Solve the same 1D linear convection problem using an implicit Euler scheme and explain its performance.

3.2 1D Diffusion

1D diffusion is also recognized as Fick's second law, or the heat equation. It describes the transient concentration (or temperature) profiles in the absence of convection.

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2} \tag{3.5}$$

where D is the diffusion coefficient. For the second order derivative we use the second order central difference scheme. The procedure is the same as the previous:

1. Discretize and isolate c_i^{n+1} using an explicit Euler time advancement scheme.
2. Adjust the matlab code and run it.
3. Try different values for D, the diffusion coefficient, and vary the step sizes Δx and Δt.
4. Solve the same system using the implicit Euler time advancement and explain its performance.

3.3 1D Linear Convection and Diffusion

convection and diffusion:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} = D\frac{\partial^2 c}{\partial x^2} \tag{3.6}$$

1. Discretize, using a central difference scheme for the spatial derivative.
2. Adjust the matlab code and run it.
3. Investigate the stability by changing the time and spatial parameters, as well as the linear velocity and diffusion coefficient.
4. Explain the stability observations by analyzing the modified wavenumber.

3.4 2D Convection and Diffusion

Here we will analyze the distribution of matter in 2 dimensions via convection and diffusion. We will use u and v to describe the velocity in x and y direction, respectively, and c as the concentration. We will use a given velocity profile given by the following:

$$u = \cos(2\pi x)\sin(2\pi y) \qquad v = -\sin(2\pi x)\cos(2\pi y) \tag{3.7}$$

This velocity profile consists of a set of vortices that is periodic on the [0..1] domain for x and y, see figure 3.1. The velocity profile will be used to solve the mass transport that proceeds via convection and diffusion in 2D:

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial x} + v\frac{\partial c}{\partial y} = D\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right) \tag{3.8}$$

with diffusion coefficient D. Although the velocity is a steady one, not time dependent, the mass distribution will clearly be. We will therefore solve for the concentration profile $c(x, y, t)$.

1. The velocity field is divergence free, meaning that it is incompressible. Show that this is the case.
2. Discretize the 2D convection diffusion equation with a central difference scheme for the spatial derivative.

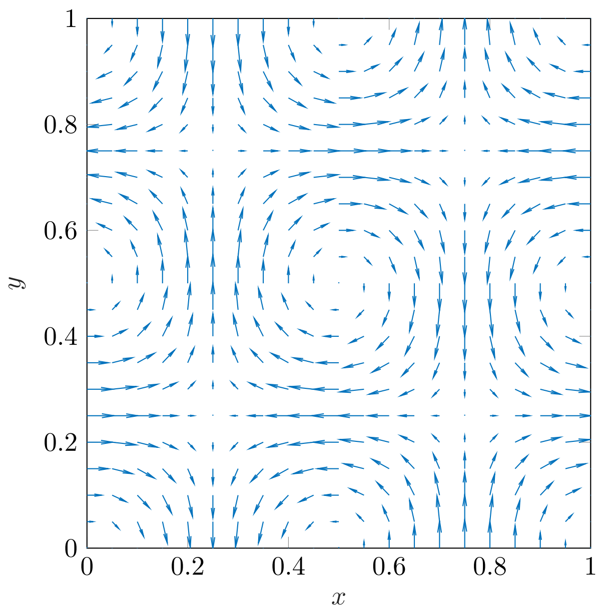


Figure 3.1: Velocity profile used in this case to study the convection diffusion equation

3. Transpose
4. Adjust and expand the matlab code.
5. Solve the development of the concentration profile for a suitable time period, using an appropriate value for D, the diffusion coefficient. For this you still need to define the boundary conditions for the concentration, which you may define yourself.

Case 4

Poisson Equation

In this case, we are considering solving the Poisson equation, an elliptic equation. This equation, and its efficiency in solving it, is crucial for solving the incompressible Navier Stokes equations. As we will see later, the Poisson equation is required to obtain the pressure distribution, which needs to be calculated at each time step for the unsteady Navier Stokes. It is also used to obtain the potential distribution for a charge distribution. The Poisson equation reads:

$$\nabla^2 w = -p \tag{4.1}$$

We will compare two different ways in numerically computing the Poisson equation. One where we iterate towards the steady solution, and one where we directly solve. We will analyze the accuracy and efficiency so that we can select either method later when we need it to solve the Navier-Stokes equations.

We will discretize the Poisson equation using a 2nd-order central scheme, giving:

$$\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{\Delta x^2} + \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{\Delta y^2} = -p_{i,j} \tag{4.2}$$

The physical system we are considering here corresponds to the velocity profile that is obtained inside a square-shaped tube. Considering the steady-state Navier-Stokes equation, we can obtain the following Poisson equation:

$$\eta \left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} \right) = -\frac{\Delta P}{\Delta z} \tag{4.3}$$

where w is the z-direction velocity component. The solution presents a cross-sectional velocity profile for a square-shaped channel.

4.1 Iterative Solution

When we consider the iterative solution, we are in fact, updating the value of $w_{i,j}$ iteratively from an initial guess. We proceed with this iteration until the solution has converged by user-defined criteria.

```
while ( err > tol )
    iter = iter + 1;
    for j = 2:N-1
        for i = 2:N-1
            Wnew(i,j) = (Wj(i+1,j)+Wj(i-1,j)+Wj(i,j+1)+Wj(i,j-1))/4 + ...
                dx^2/4*P(i,j);
        end
    end
    err = max(max(abs(Wnew-Wj)))/max(max(abs(Wj)));
    Wj = Wnew;
    Ej(iter) = err;
end
```

The method above corresponds to the point Jacobi method, where each iteration updates the values based on the previous ones. Other iteration schemes are also possible and will be investigated in this case.

4.2 Direct Solution

For the direct method, we need to construct a matrix analogous to the BVP case (1D diffusion and reaction). However, we are now considering a 2D problem for the pressure distribution. The matrix required to solve the system of algebraic equations can be simply fabricated in matlab using the following code:

```
TN = gallery('tridiag',N−2,1,−2,1);
TNxN = kron(speye(N−2),TN) + kron(TN,eye(N−2));
```

Carefully check the elements in this matrix and generate the code to solve this system (see also the chapter on elliptic equations).

For this case, we are comparing different methods to solve the Poisson equation. We will analyze them on efficiency (speed) and accuracy.

1. Solve the system by the iterative method and analyse the number of iterations required for a relative tolerance of 10^{-4} between iterations. Analyze how the number of iterations scales with the set relative tolerance.

2. Solve the same system by the Gauss-Seidel iterative method and compare the number of iterations as well as the time for computation. Do this for different mesh sizes.

3. Solve using the Successive Over Relaxation, and optimize the relaxation parameter.

4. Solve the same system using the direct method. Analyse the computation time required as a function of mesh size.

5. We want to study the flow profiles for elongated channel cross sections, so-called shallow channels. If κ is the aspect ratio (height/width) of the channel, the non-dimensional Poisson equation becomes:

$$\kappa^2 \frac{\partial^2 \tilde{w}}{\partial \tilde{x}^2} + \frac{\partial^2 \tilde{w}}{\partial \tilde{y}^2} = -1 \tag{4.4}$$

   where we have normalized the velocity w by $\Delta P h^2/\Delta z \eta$. Investigate how you can incorporate the channel aspect ratio in the code and experiment with different values of it.

6. Try to introduce another boundary condition. For instance, one that describes a slip boundary, i.e. zero stress given by $\partial w/\partial n = 0$ (zero normal velocity gradient)

Case 5

Poisson and Nernst-Planck

The transport of ions can be described by the Nernst-Planck equation. This equation includes, besides the advective and diffusive parts, also an electric potential gradient driven part (electromigration). The electric potential is then a function of the charge density, i.e. on the distribution of the cations and anions, via the Poisson equation. The equations are therefore coupled and need to be solved in a synchronous manner.

Here, we will be using a staggered grid. A staggered grid is a method for discretization, where all variables are not defined at the same positions as in a collocated grid. The potential, ϕ, and ion concentrations, c_+ and c_-, will be defined at the cell centers of which there will be N. As the potential ϕ is defined at the cell centers, we can define the electric field, or $\nabla\phi$ at the cell faces of which there are N+1. The same goes for the ion fluxes J. Realize that, to obtain the electromigration flux at the cell faces, you need the concentration values also at the cell faces. For that, you can just average the neighboring cell centers values. The staggering also allows to define the fluxes at the cell faces in a physical correct manner, being governed by the concentration and potential gradient:

$$\nabla^2\phi = -\frac{\rho}{\varepsilon} \tag{5.1}$$

$$\rho = e\left(z_+c_+ + z_-c_-\right) \tag{5.2}$$

$$J = -D\nabla c - \frac{Dzc}{V_T}\nabla\phi \tag{5.3}$$

$$\frac{\partial c}{\partial t} = -\nabla \cdot J \tag{5.4}$$

Where ε is the permittivity, D the diffusion coefficient, z the valency, and $V_T = RT/F$ the thermal voltage. First, we need to solve the potential distribution by the Poisson equation based on the charge distribution ρ. The potential ϕ and the ion concentration c_+ and c_- are defined at the intermediate grid points, i.e. the cell centers. The electric field, i.e. the potential gradients $E = -\partial\phi/\partial x$, and the ion fluxes are then obtained at the cell faces. With these fluxes, the concentration profiles can then be updated, and the iteration progresses to the next time step.

5.1 Ion transport in 1D

This section concerns the unsteady transport of ions, where the cations and anions are initially located at distinct positions, e.g. by the attraction of a negatively and positively charged electrode. The unsteady transport follows, when the electrodes are switched off or removed. We will analyze the resulting transport based on diffusion and electromigration. The code below provides the time propagation where first the charge distribution is calculated, then the potential distribution is obtained (via Poisson equation), followed by the ion fluxes and finally updating the ion distribution.

```
%% Solve combined PDE
for i=0:dt:t_final

    %obtain phi values
    b(1:N) = z*e*(C_pos(1:N)-C_neg(1:N)); % charge density
    b = (-dx*dx/epsilon) * b;
    phi = A\b;

    %Find E
    E(2:N) = -(phi(2:N)-phi(1:N-1))/dx;

    %negative ions
    f_neg(2:N) = -D*(C_neg(2:N)-C_neg(1:N-1))/dx -...
        (D/2/V_T)*(C_neg(2:N)+C_neg(1:N-1)).*E(2:N);       %calculate ...
            flux values
    C_neg = C_neg + (dt/dx)*(f_neg(1:N)-f_neg(2:N+1)); %calculate C at ...
        the next time step

    %positive ions
    f_pos(2:N)= -D*(C_pos(2:N)-C_pos(1:N-1))/dx + ...
        (D/2/V_T)*(C_pos(2:N)+C_pos(1:N-1)).*E(2:N);       %calculate ...
            flux values
    C_pos = C_pos + (dt/dx)*(f_pos(1:N)-f_pos(2:N+1)); %calculate C at ...
        the next time step

end
```

1. Complete the code by defining the required constants (take $D = 10^{-3}$, and take 1 for z, e and ε), the matrix A for the Poisson equation, and the initial ion concentration distribution $c_\pm = e^{-10(x\pm3)^2} + 1$ for each ion on a grid $-5 \le x \le 5$. Use for the left BC $\phi = 0$ and for the right BC $\nabla\phi = 0$.

2. Investigate the time evolution of the concentration and potential distribution.

3. Analyze the temporal change of concentration due to ion migration and diffusion, by comparing the corresponding fluxes.

4. Analyze the fluxes for different amount of background concentration.

5. Adjust the code such that you simulate 2 electrodes positioned at each end (at $x = -5$ and $x = 5$), with a fixed different potential applied to each. Analyze the steady state ion concentration and potential distribution of this situation for different applied potential differences.

6. Using a no-flux boundary in the previous example corresponds to a purely capacitive process (ions are stored in the double layer and no electrode reaction takes place). Adjust the code to account for an ion flux in case of electrode reactions.

Case 6

Incompressible Navier Stokes

This case will be the first to deal with the Navier Stokes equation. We will work on this case using matlab. Recalling, the Navier Stokes for an incompressible fluid expressed in vector notation reads:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla p + \mu\nabla^2\boldsymbol{u} + \sum F^{bodyforces} \tag{6.1}$$

where $\boldsymbol{u}$ represents the velocity vector $[u, v]$. The continuity equation for incompressible flow reads:

$$\nabla\cdot\boldsymbol{u} = 0 \tag{6.2}$$

In this assignment, we will numerically solve the Navier Stokes equation, while continuity is imposed. Further information regarding solving the Navier Stokes and continuity equations are given in chapter 8.

6.1 mesh

We will use a staggered mesh, where the pressure is computed at the cell centers, while the velocities are computed at the cell faces (see figure 6.1). This choice is preferred, as velocity is connected to the pressure gradient.

6.2 Discretization

The system is solved via the predictor-corrector method. See chapter 8. During the predictor step, the viscous and convective terms are spatially discretized.

$$\boldsymbol{u}^* = \boldsymbol{u}^n + \Delta t\left(-\boldsymbol{u}^n\cdot\nabla\boldsymbol{u}^n + \nu\nabla^2\boldsymbol{u}^n\right) \tag{6.3}$$

For just the u velocity (x-direction), we obtain:

$$u^* = u^n + \Delta t\left(-\left(u^n\frac{\partial u^n}{dx} + v^n\frac{\partial u^n}{dy}\right) + \nu\left(\frac{\partial^2 u^n}{\partial x^2} + \frac{\partial^2 u^n}{\partial y^2}\right)\right) \tag{6.4}$$

with the following discretization:

$$u\frac{\partial u}{dx} = u_{i,j}\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} \tag{6.5}$$

$$v\frac{\partial u}{dy} = \frac{1}{4}\left(v_{i-1,j} + v_{i,j} + v_{i-1,j+1} + v_{i,j+1}\right)\frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} \tag{6.6}$$

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta x^2} \tag{6.7}$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta y^2} \tag{6.8}$$

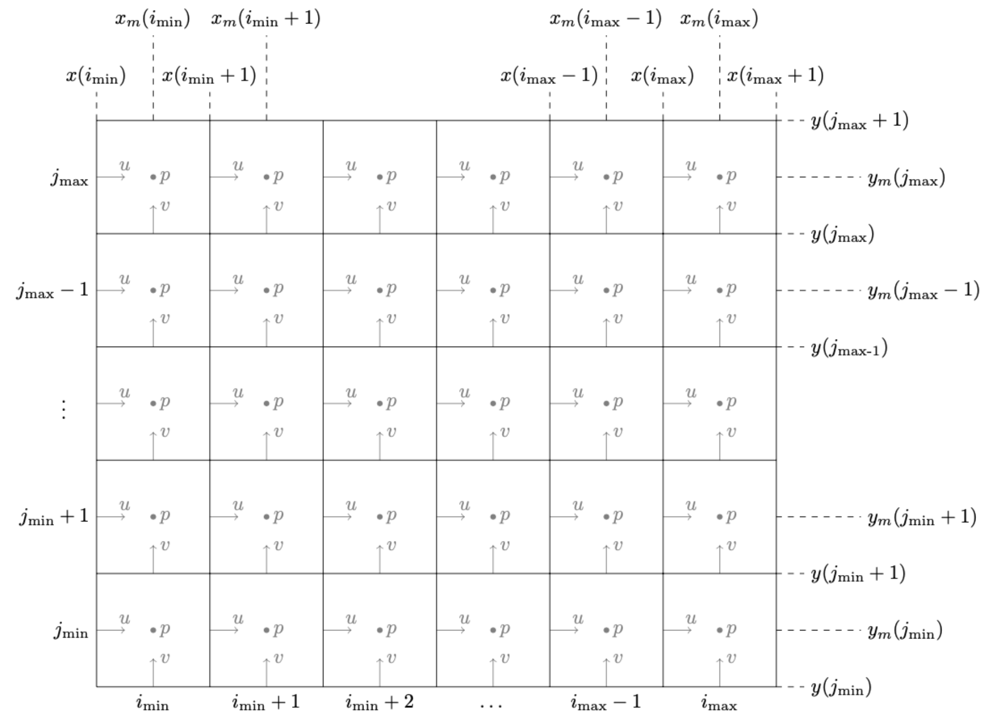


Figure 6.1: Computational mesh indicating the locations where pressure p and velocity terms u and v are computed.

It is evident that v in 6.6 needs to be estimated at the location of u, which is done by averaging the neighboring values (or interpolating).

For the pressure Poisson equation, we have:

$$\nabla^2 p^{n+1} = -\frac{\rho}{\Delta t}\nabla \cdot \boldsymbol{u}^* \tag{6.9}$$

which reads in discretized form

$$\frac{p^{n+1}_{i-1,j} - 2p^{n+1}_{i,j} + p^{n+1}_{i+1,j}}{\Delta x^2} + \frac{p^{n+1}_{i,j-1} - 2p^{n+1}_{i,j} + p^{n+1}_{i,j+1}}{\Delta y^2} = -\frac{\rho}{\Delta t}\left(\frac{u^*_{i+1,j} - u^*_{i,j}}{\Delta x} + \frac{v^*_{i,j+1} - v^*_{i,j}}{\Delta y}\right) \tag{6.10}$$

The divergence is easily calculated by a forward difference, as this exists exactly at the location of the pressure node. This equation can be solved by composing a Laplacian matrix operator $\boldsymbol{L}$, such that the equation reads as $\boldsymbol{L}\boldsymbol{p}^{n+1} = \boldsymbol{R}$. In this, both $\boldsymbol{p}^{n+1}$ and $\boldsymbol{R}$ are both organized into large vectors. With the pressure obtained at time $n+1$, the velocity field at time can now be calculated by the corrector step from $\boldsymbol{u}^*$.

Now, the boundary conditions still need to be implemented. This is a bit tricky, as u is not defined at the top and bottom boundary, while v is not defined at the left and right boundary (see figure 6.1). For this we use the ghost points that are located outside of the actual mesh. The actual boundary lies exactly between the ghost point and the min/max locations. For the top boundary (and similar for others), we can write

$$u_{i,jmax+1} = -u_{i,jmax} + 2u_{top} \tag{6.11}$$

1. Complete the matlab code that is provided. Also include plotting of the resulting velocity and pressure.

2. Run the code for the lid-driven cavity problem and vary parameters like viscosity and velocity.

3. Think of a way to check for a steady velocity field.

4. Also, think of a way to analyze the time till steady state.
5. Compare the outcome, in terms of velocity field, with literature (e.g. Erturk, E., Corke, T. C., Gokcol, C. (2004, November 16). Numerical Solutions of 2-D Steady Incompressible Driven Cavity Flow at High Reynolds Numbers. http://doi.org/10.1002/fld.953). Find ways to ensure that you reach a steady state.

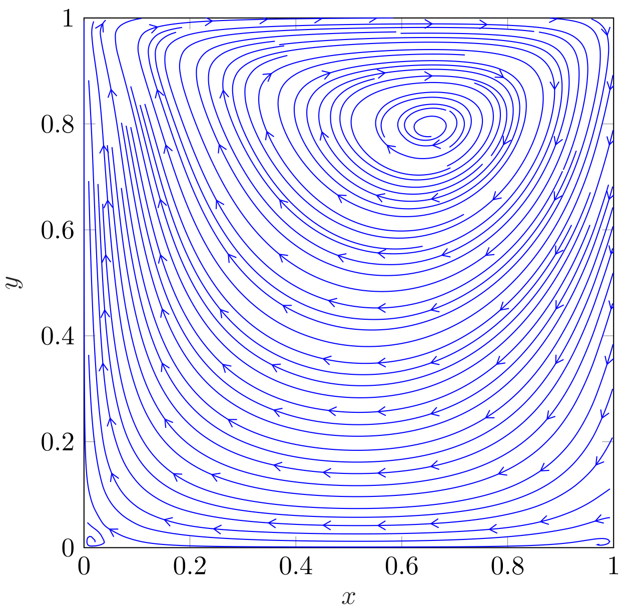


Figure 6.2: Streamline obtained for cavity flow problem

6.3 Case Report

Prepare a short and concise report presenting your results and findings. Include the following:

1. The resulting velocity vector plots as well as the pressure distribution.
2. Analyze what your Matlab model provides when the mass conservation is imposed based on a divergence-free (u^*, v^*). Use a corrector step based on $\frac{u^* - u^n}{\Delta t} = -\frac{1}{\rho}\nabla p$.
3. Analyze the outcome if you assume the so-called Stokes flow (creeping flow). This can be imposed by assuming the convective part negligible compared to the diffusive part of the Navier-Stokes equation. Why is this valid for very low Reynolds numbers?
4. Discuss the influence of viscosity on the outcome and time evolution of the solution.
5. Vary the velocity of the moving lid and discuss its influence on the solution. Change iteration parameters, if required.
6. Analyze your code regarding stability for varying Δx, Δy and Δt. Discuss your findings.

Case 7

Final Case

For the final case, we will use a paper from the literature as a starting point. This paper will describe some kind of fluid dynamic problem, mainly with experimental results. You will have to design a CFD model that one can use to validate the experimental observations. This will be a challenge but likely a very stimulating exercise with possibly very surprising outcome.

To structure this case, there are a few phases to be identified:

1. Make an initial plan of the problem and how you propose to tackle it. It is possible that you only want to make a model about a part of the experiment. Also, your model can be (and most often is) a simplification of the actual physical experiment. Include in your approach what physics is required.

2. Discuss this plan with the teachers. This is mainly meant to protect yourself and evaluate the feasibility of the approach. Also discuss what exact outcome you foresee to generate (what will be your variables).

3. Perform the modeling. Analyze your data carefully and compare/verify it to the experiments from the paper.

4. Prepare a short paper out of this, as if you were publishing these results related to the paper you started with.

Part III

Appendices

Appendix 1

Discretization schemes

This chapter describes and analyses different discretisation schemes and their stability. The analysis is done for a linear convection problem, given by:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{1.1}$$

1.1 FD in time, BD in space

The discretization scheme consists of FD in time, BD in space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{\Delta x}\left(u_i^n - u_{i-1}^n\right) = 0 \tag{1.2}$$

The difference between the discretised equation and the exact solution provides the truncation error. Using the corresponding Taylor series and the exact equation gives:

$$\begin{aligned}\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{\Delta x}\left(u_i^n - u_{i-1}^n\right) - \left(\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x}\right)_i^n &= \\ \frac{\Delta t}{2}\left(\frac{\partial^2 u}{\partial t^2}\right)_i^n - \frac{c\Delta x}{2}\left(\frac{\partial^2 u}{\partial x^2}\right)_i^n &+ O\left(\Delta t^2, \Delta x^2\right)\end{aligned} \tag{1.3}$$

Hence, the modified differential equation can be obtained, by introducing $\bar{u}$ for which the PDE is exactly equal to the discretised equation:

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2}\left(\frac{\partial^2 \bar{u}}{\partial t^2}\right)_i^n - \frac{c\Delta x}{2}\left(\frac{\partial^2 \bar{u}}{\partial x^2}\right)_i^n + O\left(\Delta t^2, \Delta x^2\right) \tag{1.4}$$

The first time derivative can be isolated and time differentiated:

$$\frac{\partial \bar{u}}{\partial t} = -c\frac{\partial \bar{u}}{\partial x} + O\left(\Delta t, \Delta x\right) \tag{1.5}$$

$$\frac{\partial^2 \bar{u}}{\partial t^2} = -c\frac{\partial}{\partial t}\frac{\partial \bar{u}}{\partial x} + O\left(\Delta t, \Delta x\right) \tag{1.6}$$

$$= c^2\frac{\partial^2 \bar{u}}{\partial x^2} + O\left(\Delta t, \Delta x\right) \tag{1.7}$$

So, now the final MDE can be obtained:

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{c^2\Delta t}{2}\frac{\partial^2 \bar{u}}{\partial x^2} - \frac{c\Delta x}{2}\frac{\partial^2 \bar{u}}{\partial x^2} + O\left(\Delta t^2, \Delta x^2\right) \tag{1.8}$$

or

$$\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x} = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2} + O\left(\Delta t^2, \Delta x^2\right) \tag{1.9}$$

From this MDE it is apparent that we'll have no numerical diffusion when $\frac{c\Delta t}{\Delta x} = 1$.

1.2 FD in time, FD in space

The discretization scheme consists of FD both in time and space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{\Delta x}\left(u_{i+1}^n - u_i^n\right) = 0 \tag{1.10}$$

Similar as before, we can obtain the modified differential equation by introducing $\bar{u}$ to get:

$$-\left(\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x}\right)_i^n = \frac{\Delta t}{2}\left(\frac{\partial^2 \bar{u}}{\partial t^2}\right)_i^n + \frac{c\Delta x}{2}\left(\frac{\partial^2 \bar{u}}{\partial x^2}\right)_i^n + O\left(\Delta t^2, \Delta x^2\right) \tag{1.11}$$

And with similar expressions for $\frac{\partial^2 \bar{u}}{\partial t^2}$ we obtain:

$$\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x} = \frac{c\Delta x}{2}\left(-1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 \bar{u}}{\partial x^2} + O\left(\Delta t^2, \Delta x^2\right) \tag{1.12}$$

Clearly, this MDE contains a "diffusion coefficient" that for all values for Δx and Δt is negative, and hence unstable.

1.3 FD in time, CD in space

The discretization scheme consists of FD both in time and CD in space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + \frac{c}{2\Delta x}\left(u_{i+1}^n - u_{i-1}^n\right) = 0 \tag{1.13}$$

Following the same procedure as in the previous sections, we obtain a final MDE:

$$\frac{\partial \bar{u}}{\partial t} + c\frac{\partial \bar{u}}{\partial x} = -\frac{\Delta t c^2}{2}\frac{\partial^2 \bar{u}}{\partial x^2} + O\left(\Delta t^2, \Delta x^2\right) \tag{1.14}$$

1.4 Lax Friedrichs method

As shown in the previous section, for a linear convection problem, the FD in time and CD in space discretisation results in un unstable scheme. The Lax Friedrichs method of solving the above partial differential equation is given by:

$$\frac{u_i^{n+1} - \frac{1}{2}\left(u_{i+1}^n + u_{i-1}^n\right)}{\Delta t} + \frac{c}{2\Delta x}\left(u_{i+1}^n - u_{i-1}^n\right) = 0 \tag{1.15}$$

1.5 Leapfrog scheme

The previous schemes all approximate the new time step based on the solution of the previous time (the solution at $n+1$ is obtained from the one at n). In the Leapfrog scheme, the solutions of two time steps are used to approximate the new time step:

$$\frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t} + \frac{c}{2\Delta x}\left(u_{i+1}^n - u_{i-1}^n\right) = 0 \tag{1.16}$$

Appendix 2

MATLAB: A (Very) Brief Introduction

2.1 Introduction

MATLAB (an abbreviation of MATrix LABoratory) is an extremely powerful scripting language for solving mathematical problems (numerical or symbolic). It has the flexibility to write your own potentially very complicated programs, as well as an extremely large library of useful programs organized in packages called toolboxes. For example, the optimization toolbox is extremely useful for solving nonlinear equations or performing least-squares regression fits.

The primary interest for iCFD is to use MATLAB to understand and run numerical methods for solving ordinary differential equations (ODEs), partial differential equations (PDEs) and differential-algebraic equations (DAEs). Numerical methods will be coded by you, so having a good understanding of the syntax of MATLAB programming will greatly benefit your time in the course. MATLAB also has built-in functionalities for some of the problems we are interested in and we will compare the codes you will write with these predefined functions as well. There is a huge library of scientific MATLAB codes freely available, many from the Mathworks website itself (file-exchange).

2.2 Formatting Commands

Display Commands

```
;         % ; suppresses display output
clc       % clc, UNIX command to clear the screen
clear x   % clears variable x from the workspace
clear all % clears all variables from the workspace and breakpoints
close x   % closes figure x
close all % closes all figure windows
```

Numerical Formatting

```
format short      % short—form real number
format long       % long—form real number
format short e    % short—form scientific notation
format long e     % long—form scientific notation
format rational   % rational numbers
format hex        % hexadecimal display
```

2.3 Help Files, Documentation and Examining Functions

The help files in MATLAB are extremely comprehensive, they can be accessed generally by typing 'help' in the command window. 'help' for specific functions as exists, simply type 'help name' for the name of a specific function. For more thorough documentation and examples, you can use the 'doc' command.
Example

```
help fsolve
doc fsolve
```

You can immediately see the difference between typing 'help' vs. 'doc'. You can also see the specific code by typing 'type name' in the command window, MATLAB will output the function in the command window for you to examine.

2.4 Scalar, Vector and Matrix Operations

After that slight preamble, now we come to beginning to use MATLAB for computation. MATLAB generally treats scalars, vectors and matrices as arrays of varying dimension. This means the notation for implementing them is the same, it also has some conventions about what to do if you attempt various operations with combinations of two types, such as multiply vectors.
Scalars and Vectors

```
A = [1 2 3 4 5]

B = A+A

C = A.*A

D = A*A
```

Notice what happens when you try to multiply without A*A vs. A.*A, the error message says the dimensions are not compatible. This is because .* represents element by element multiplication (dot-product) while * represents vector/matrix multiplication. In order to multiple matrices, the dimensions must be compatible. Trying typing:

```
D=A*A'

E = A'*A
```

The use of ' is a shortcut for transpose, which is quite handy as you will be transposing many matrices during your time in this course.
Matrices
There is no real difference as to how MATLAB treats scalars [1x1], vectors [Nx1 or 1xN] or matrices [MxN]. Higher dimensional arrays are also possible but the predefined operations (inversion, transpose, etc.) are often not defined in these cases.

2.5 Boolean and Logical Operators

As with other programming languages, you can perform logical/boolean type operations in MATLAB. > (greater), >= (greater than or equal), < (less than), <= (less than or equal), == (equal), and ! = (not equal) can all be used.

```
A = [0 1 2 3 4 5 6]
```

```
A>3
A≥3

A = 1;
B = 3;
C = 5;

if A>B
 disp('This is not going to display anything...')
 elseif (B>A)&(C>B)
 disp('This should be the one to display...')
 else
 disp('Also no...')
end
```

2.6 Loops

for loops run over a specified number of operations. while loops repeat as long as a condition is true. This essentially means a while loop may not run (condition is checked first).

```
A = 0
for i=1:5
  A =1+A
end

B = 0
while B<3
B = 1+B
end
```

Be careful to avoid defining an infinite loop (condition is always satisfied) for while loops. It is also preferable to avoid the use of for loops when they can be replaced by matrix operations, as MATLAB precompiles matrix operation commands and does not do this for for loops (MATLAB is a high-level scripting language, you do not compile an executable every time you run a program). This effectively means that for loops are much slower computationally (although sometimes unavoidable).

2.7 Special MATLAB Operators & Matrices

2.7.1 \

Backslash (\) is an extremely convenient predefined MATLAB command to take the inverse of a matrix. For example,

```
A = [[1 2];[5 -1]]

B = [1 2]'

C = A\B

Btest = A*C
```

You can see that this does the matrix inversion for you directly, it is the generally preferred way to solve linear equations of this nature.

2.7.2 :

: operators are also extremely useful as tools to retrieve values within vectors or matrices for given ranges. For example,

```
A = [[1 2 3];[4 5 6];[7 8 9]]

B = A(:,1)

C = A(1,:)
```

2.7.3 Special Matrices and Operations

```
A = zeros(5,1)
B = zeros(5,5)
C = ones(5)
D = eye(3)

E = diag(D)

F = magic(3)
x = linspace(0,10,11)
y = 0:1:10
rank(D)

det(C)
```

The first 4 matrices/vectors should be fairly self-explanatory, zeros generates a MxN matrix filled with zeros, ones does the same but filled with ... ones. What do you think the last command means with respect to C? Try typing inv(C) to see. linspace is a useful command to create a linearly spaced vector/array between a start and end point with a number of points (the default is 100).

2.8 Scripts vs. Functions

Scripts are the default creation of the editor, they execute the commands line by line and any computed quantities are stored in the workspace. Functions need a function definition line at the first line, you can place functions within functions and for many of the predefined numerical methods a function is required (the equation to be solved, quantity computed to be fit to data, etc). Scripts cannot have functions in the same file but can call functions that are saved as a separate file.

Start the MATLAB editor, type the following commands, save and run.

```
A = 3
B = magic(A)
```

Now try creating a separate file with these lines. You will need to save it to run, the default name will be the name you gave the function (not the value it is equal to, that is the value or quantity that will be returned by the function).

```
function B = myMagic(A)
B = magic(A);
```

2.9 Plotting

```
x=linspace(0,1);
y = x.^2+sin(x*pi);
plot(x,y,'o')
```

Try adding the following lines:

```
xlabel('x')
ylabel('y')
title('Our Plotted Function')
```

There are many more options available, try doc plot to see what other options you have for legends, symbols, etc. 2d plots can be made using surf or pcolor, requiring a matrix of x-values and y-values (this can be generated from the individual vectors using the meshgrid command, we will make use of this in your cases).

```
x=linspace(0,1,20);
y=linspace(0,1,10);
[X,Y] = meshgrid(x,y);
Z = X.^2+sin(Y*pi);
pcolor(Y,X,Z)
```

Try typing the following lines:

```
colormap(jet)
shading interp
```

If you want to flip the horizontal and vertical, you implement the plot as pcolor(X,Y,Z').

2.10 Debugging and Other Tips

This is maybe the most useful section for your work in iCFD, that is how to debug effectively in MATLAB. As with a conventional programming language, MATLAB gives you the ability to watch variables as you run a program. Under the editor window, observe the command "Breakpoints". You can use this to add a breakpoint to a line where if you run the script or function, it will run up to this point. After this, you can then continue running the program or you can step through the program line by line. If you hold your cursor over a variable name, you will see the value. This is extremely useful for seeing if for example your program is doing what you think it should be. You can also see the values in matrices, etc. in the workspace.

Most important in this course is that you are looking for physical insight, not numbers. For example, what should the radial dependence of velocity in a long straight tube under laminar flow?