Mathematical Techniques for Physics part 1: Numerical Finite difference techniques for partial differential equations Version 2.0

Dr. Richard Stevens (r.j.a.m.stevens@utwente.nl)
Physics of Fluids Group, Max Planck Center Twente for Complex Fluid Dynamics, J. M. Burgers Center for Fluid Dynamics and MESA$^+$ Research Institute, University of Twente, P. O. Box 217, 7500 AE Enschede, The Netherlands

Contents

1 Introduction

This reader contains all the necessary information to do the exercises for the numerical homework assignments of part 1 of the Mathematical and Numerical Physics course 2020/2021. In chapter, 2, we discuss the derivation of the finite difference discretization. Chapter 3 discusses the different numerical methods that are considered in the lectures and homework assignments. Chapter 4 discusses the algorithms that you need to implement in the homework assignments. Chapter 5 discusses the methods that we use to analyze the convergence of the different methods. Aspects relevant for two-dimensional simulations are discussed in chapter 6.

2 Finite difference discretization

There are several ways to derive a discrete system of equations from the differential problem, each with their own merits for a given class of problems. Here we consider the use of the finite difference method to approximate the solution to a partial differential equation. For this, we discretize the space for which we want to solve the evolution of the dynamics subscribed by the partial differential equation. At each grid point, the different terms of the differential equation are replaced by an approximation containing the value in the grid point itself, and the values of neighboring grid points. The approximate relation is derived by performing Taylor expansions.

We start by considering a one dimensional problem. We consider a uniform grid of points x_i such that $x_i = i(\Delta x)$ for $i = 0, ..., n_x$ while $(\Delta x) = 1/n_x$ is fixed. Let ϕ_i denote the value of the unknown function ϕ at x_i, i.e. $\phi_i = \phi(x_i)$. The values of ϕ for neighboring points can be obtained by performing a Taylor series expansion. For example the value at ϕ_{i+1} can be estimated to be

$$\phi_{i+1} = \phi_i + (\Delta x) \left.\frac{\partial \phi}{\partial x}\right|_{x_i} + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} + \frac{(\Delta x)^3}{3!} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \frac{(\Delta x)^4}{4!} \left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} + ... \tag{1}$$

The above equation can be reordered to get an estimation for the first-order gradient at x_i

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_{i+1} - \phi_i}{(\Delta x)} - \frac{(\Delta x)}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} - \frac{(\Delta x)^2}{3!} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} - \frac{(\Delta x)^3}{4!} \left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} ... \tag{2}$$

which can be written in the following form

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_{i+1} - \phi_i}{(\Delta x)} + T^{\Delta x}(x_i) \tag{3}$$

Here the first term gives an approximation of the first derivative at $x = x_i$ and is known as the forward or upwind differencing method. The second term $T^{\Delta x}(x_i)$ gives the truncation error

$$T^{\Delta x}(x) = -\frac{(\Delta x)}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} - \frac{(\Delta x)^2}{3!} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} - \frac{(\Delta x)^3}{4!} \left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} ... \tag{4}$$

Note that the leading order term in the truncation is $\mathcal{O}(\Delta x)$. For functions ϕ that are smooth on the scale of the mesh size Δx the error will be dominated by the leading order term in Δx and therefore the above approximation is a first-order approximation of the first-order derivative.

Obviously, the above is not the only possible derivation for the first-order derivative. Instead of using upwind differencing we can also use backward or downwind differencing for which we obtain the following Taylor series expansion

$$\phi_{i-1} = \phi_i - (\Delta x) \left.\frac{\partial \phi}{\partial x}\right|_{x_i} + \frac{(\Delta x)^2}{2} \left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} - \frac{(\Delta x)^3}{3!} \left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \frac{(\Delta x)^4}{4!} \left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} - ... \tag{5}$$

which can be reordered as follows

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_i - \phi_{i-1}}{(\Delta x)} + T^{\Delta x}(x_i) \tag{6}$$

with

$$T^{\Delta x}(x) = \frac{(\Delta x)}{2}\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} - \frac{(\Delta x)^2}{3!}\left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \frac{(\Delta x)^3}{4!}\left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} \dots \tag{7}$$

Generally, higher order approximations can be constructed when a wider stencil is used. For example we can combine equation (1) and (5) to obtain

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_{i+1} - \phi_{i-1}}{2(\Delta x)} + T^{\Delta x}(x_i) \tag{8}$$

with

$$T^{\Delta x}(x) = -\frac{(\Delta x)^2}{3!}\left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \frac{(\Delta x)^4}{5!}\left.\frac{\partial^5 \phi}{\partial x^5}\right|_{x_i} \dots \tag{9}$$

One can see that this is a second-order approximation to $\partial\phi/\partial x$ at x_i using only two points. Note that this central approximation is the average of the one-sided first-order approximations in equations (3) and (6). As the leading order errors ($\mathcal{O}(\Delta x)$), see equations (4) and (7), are the same but with opposite sign they cancel when the average is taken.

Approximations for higher-order derivatives can be constructed in a similar way. For example, adding equations (1) and (5) gives

$$\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\Delta x)^2} + T^{\Delta x}(x_i) \tag{10}$$

with

$$T^{\Delta x}(x) = -\frac{(\Delta x)^2}{12}\left.\frac{\partial^4 \phi}{\partial x^4}\right|_{x_i} + \dots \tag{11}$$

which shows that equation (10) is a second-order approximation to $\partial^2\phi/\partial x^2$ at x_i.

The next section summarizes the results for the first- and second-order derivatives that are obtained using the Taylor expansion method described above. The approach for non-uniform grids is not fundamentally different, but the expressions become more complex and will not be considered here.

2.1 Finite difference formulas for first-order derivate

Downwind first-order finite difference scheme, see also equation (6):

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_i - \phi_{i-1}}{\Delta x} + \frac{\Delta x}{2}\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} + \dots \tag{12}$$

Downwind second-order finite difference scheme:

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{3\phi_i - 4\phi_{i-1} + \phi_{i-2}}{2\Delta x} + \frac{(\Delta x)^2}{3}\left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \dots \tag{13}$$

Upwind first-order finite difference scheme, see also equation (3):

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_{i+1} - \phi_i}{\Delta x} - \frac{\Delta x}{2}\left.\frac{\partial^2 \phi}{\partial x^2}\right|_{x_i} + \dots \tag{14}$$

Upwind second-order finite difference scheme:

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{-3\phi_i + 4\phi_{i+1} - \phi_{i+2}}{2\Delta x} - \frac{(\Delta x)^2}{3}\left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \dots \tag{15}$$

Central second-order finite difference scheme, see also equation (8)

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} - \frac{(\Delta x)^2}{6}\left.\frac{\partial^3 \phi}{\partial x^3}\right|_{x_i} + \dots \tag{16}$$

Central fourth-order finite difference scheme:

$$\left.\frac{\partial \phi}{\partial x}\right|_{x_i} = \frac{-\phi_{i+2} + 8\phi_{i+1} - 8\phi_{i-1} + \phi_{i-2}}{12\Delta x} - \frac{(\Delta x)^4}{30}\left.\frac{\partial^5 \phi}{\partial x^5}\right|_{x_i} + \dots \tag{17}$$

2.2 Finite difference formulas for second-order derivates

Downwind first-order finite difference scheme:

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_i} = \frac{\phi_i - 2\phi_{i-1} + \phi_{i-2}}{(\Delta x)^2} + \Delta x \left.\frac{\partial^3\phi}{\partial x^3}\right|_{x_i} + ... \tag{18}$$

Downwind second-order finite difference scheme:

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_i} = \frac{2\phi_i - 5\phi_{i-1} + 4\phi_{i-2} - \phi_{i-3}}{(\Delta x)^2} - \frac{11(\Delta x)^2}{12}\left.\frac{\partial^4\phi}{\partial x^4}\right|_{x_i} + ... \tag{19}$$

Upwind first-order finite difference scheme:

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_i} = \frac{\phi_{i+2} - 2\phi_{i+1} + \phi_i}{(\Delta x)^2} - \Delta x \left.\frac{\partial^3\phi}{\partial x^3}\right|_{x_i} + ... \tag{20}$$

Upwind second-order finite difference scheme:

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_i} = \frac{2\phi_i - 5\phi_{i+1} + 4\phi_{i+2} - \phi_{i+3}}{(\Delta x)^2} + \frac{11(\Delta x)^2}{12}\left.\frac{\partial^4\phi}{\partial x^4}\right|_{x_i} + ... \tag{21}$$

Central second-order finite difference scheme, see also equation (10):

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{x_i} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{(\Delta x)^2} - \frac{(\Delta x)^2}{12}\left.\frac{\partial^4\phi}{\partial x^4}\right|_{x_i} + ... \tag{22}$$

Central fourth-order finite difference scheme:

$$\frac{\partial^2\phi}{\partial x^2} = \frac{-\phi_{i+2} + 16\phi_{i+1} - 30\phi_i + 16\phi_{i-1} - \phi_{i-2}}{12(\Delta x)^2} + \frac{(\Delta x)^4}{90}\left.\frac{\partial^6\phi}{\partial x^6}\right|_{x_i} + ... \tag{23}$$

3 Different numerical methods we will consider

To describe phase-separation problems discussed in the analytical part of the course we derived the following equation

$$\dot{\phi} + \mathbf{v}\cdot\nabla\phi = -\nabla\cdot(-M\nabla\mu + \mathbf{J}^n) \tag{24}$$

with

$$\mu(r) = a\phi + b\phi^3 - \kappa\nabla^2\phi \tag{25}$$

Assuming $a = -1$, $b = 1$, and $M(r)$ is independent of the spatial coordinate we can rewrite equation (24) and (25) as

$$\dot{\phi} = M\nabla^2(\phi^3 - \phi - \kappa\nabla^2\phi) \tag{26}$$

This equation is difficult to solve numerically due to the fourth-order spatial derivative, and the ϕ^3 term in the Laplacian. Therefore, for simplicity, we first neglect the "ϕ^3" and "$-\kappa\nabla^2\phi$" term in the Laplacian and consider [1]

$$\dot{\phi} = M\nabla^2\phi. \tag{27}$$

We first consider a one-dimensional domain $x \in [0, L]$ with the initial condition

$$\phi(x, 0) = f(x) \tag{28}$$

and Dirichlet boundary conditions

$$\phi(0, t) = \phi(L, t) = 0. \tag{29}$$

[1]note that a minus sign is added for convenience

3.1 Euler forward method

The first and simplest method we discuss is the Euler forward method. In section 3.1.1, we discuss the derivation of this method, and in section 3.1.2, we illustrate the Von Neumann analysis that is used to analyze the stability of the method.

3.1.1 The method

The Euler forward method is obtained by using an explicit first-order forward discretization for the temporal derivative at the left hand side of equation (27) and a second-order central finite difference method in space to represent the Laplacian on the right hand side, see equation (10). The grid points in space and time used in the Euler forward method are sketched in figure 1. This results in the following numerical scheme:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = M \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{(\Delta x)^2} \tag{30}$$

Bringing all the terms for the current time step n to the right hand and the terms for the next time step $n+1$ to the left hand side gives

$$\phi_i^{n+1} = (1 - 2\alpha)\phi_i^n + \alpha(\phi_{i+1}^n + \phi_{i-1}^n) \tag{31}$$

where

$$\alpha = M \frac{\Delta t}{(\Delta x)^2} \tag{32}$$

is a constant that is used to simplify the notation. This is known as an explicit method as only information from the current time step n is used to calculate the solution at the next time step $n+1$. As a result, equation (31) allows one to calculate the solution at the next time step directly from known information at the current time step. The Taylor series expansions discussed in chapter 2 show that the Euler forward method has a first-order local truncation error for the temporal derivative and a second-order local truncation error for the spatial derivative.

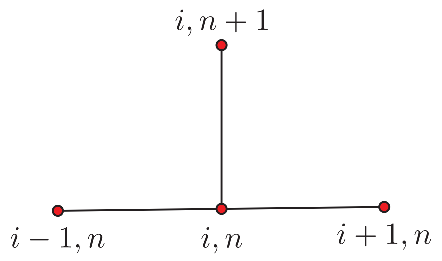

Figure 1: Sketch of the points used in the Euler forward method.

3.1.2 Von Neumann method to analyze the stability

To analyze the stability of the Euler forward method, we apply the von Neumann analysis, which is based on the decomposition of the errors into Fourier series. The analysis was briefly described in a 1947 article by British researchers Crank and Nicolson, see also section 3.4. Later, the method was given a more rigorous treatment in an article co-authored by John von Neumann, who is generally regarded as one of the greatest mathematicians of the twentieth century. Below we illustrate the application of this method

The basic idea is to substitute Fourier mode [2]

$$\phi_i^n = \lambda^n e^{[\mathrm{Img}\ k\ (i\Delta x)]} \tag{33}$$

[2]Here "Img" indicates the imaginary number and i the number of the grid

into the difference equation (31). We use

$$\phi_{i\pm1}^{n} = \lambda^{n} e^{[\text{Img } k \ ([i\pm1]\Delta x)]} = e^{[\pm\text{Img } k \ (\Delta x)]}\phi_i^n \tag{34}$$

to obtain

$$\lambda^{n+1}e^{[\text{Img } k \ (i\Delta x)]} = \lambda^{n} e^{[\text{Img } k \ (i\Delta x)]} + \alpha\lambda^{n} e^{[\text{Img } k \ (i\Delta x)]}(e^{[\text{Img } k \ \Delta x]} - 2 + e^{[-\text{Img } k \ \Delta x]}) \tag{35}$$

divide equation (35) by $\lambda^n e^{[\text{Img } k \ (i\Delta x)]}$ to obtain the requirements for λ

$$\begin{aligned} \lambda(k) &= 1 + \alpha(e^{[\text{Img } k \ \Delta x]} - 2 + e^{[-\text{Img } k \ \Delta x]}) & (36)\\ &= 1 - 2\alpha[1 - \cos(k(\Delta x))] & (37)\\ &= 1 - 4\alpha\sin^2\left(\frac{1}{2}k(\Delta x)\right) & (38) \end{aligned}$$

Here, $\lambda(k)$ indicates the amplification factor for each mode k. Obviously, the requirement for a stable discretization ensures that the growth mode for each mode is smaller than 1. The condition $|\lambda(k)| \leq 1$ implies that

$$\alpha \leq \frac{1}{2} \tag{39}$$

which means that the Von Neumann stability approach imposes a restriction on the time step for which the Euler forward method is stable. For the time step Δt we should namely ensure that

$$\Delta t \leq \frac{1}{2M}(\Delta x)^2 \tag{40}$$

3.2 DuFort-Frankel method

A way to improve the Euler method is to use central differencing method for time, while calculating the term ϕ_i^n in the central second-order spatial discretization, see right hand side of equation (30), by averaging it over the time levels ϕ_i^{n+1} and ϕ_i^{n-1}. A sketch of this method is given in figure 2 and the corresponding discretization is given by

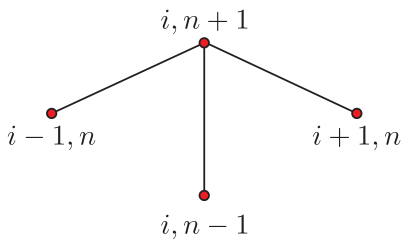

Figure 2: Sketch of the points used in the DuFort-Frankel method.

$$\frac{\phi_i^{n+1} - \phi_i^{n-1}}{2\Delta t} = M\frac{\phi_{i+1}^n - \frac{2(\phi_i^{n+1}+\phi_i^{n-1})}{2} + \phi_{i-1}^n}{(\Delta x)^2} \tag{41}$$

Collecting the terms for the next time step $n+1$ on the left hand side and the terms at the time step n and $n-1$ on the right hand side results in

$$\phi_i^{n+1} = \frac{1-2\alpha}{1+2\alpha}\phi_i^{n-1} + \frac{2\alpha}{1+2\alpha}\left(\phi_{i+1}^n + \phi_{i-1}^n\right) \tag{42}$$

Just as the Euler forward method this in an explicit method as there is a direct recipe to calculate ϕ_i^{n+1} from known values at the current and previous time step. It can be shown that the leading local truncation errors are $(\Delta t)^2$, $(\Delta x)^2$, and $(\Delta t)^2/(\Delta x)^2$. For consistency you have to ensure that $(\Delta t)/(\Delta x) \to 0$ when $\Delta t \to 0$ and $\Delta x \to 0$.

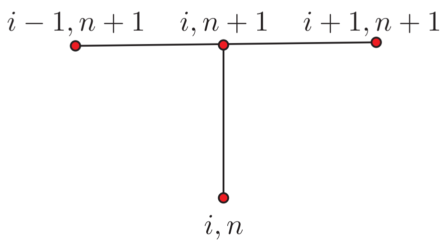

Figure 3: Sketch of the points used in the Euler backward method.

3.3 Euler backward method

A severe limitation of the Euler forward method is the restriction on time step; see equation (40). This means that many time steps are required to obtain a solution. Moreover, when we reduce Δx by a factor of 2, the time step has to be reduced by a factor of 4. While the DuFort-Frankel method is more advanced than the Euler forward method, we like to do even better.

A way to alleviate this time step restriction due to stability constraints is to use backward differencing in time. This method requires more sophisticated calculations that require more computational time; see chapter 4. The benefit of the Euler backward method is that the time step can be much bigger, especially for large grids, which makes this a beneficial approach in practice.

The Euler backward method, which is also known as the implicit method, relies on using backward differencing in time and a second-order central difference scheme in space. The grid points used in this method are sketched in figure 3 and the discretization scheme reads

$$\frac{\phi_i^{n+1}-\phi_i^n}{\Delta t}=M\frac{\phi_{i+1}^{n+1}-2\phi_i^{n+1}+\phi_{i-1}^{n+1}}{(\Delta x)^2} \tag{43}$$

One can see that this system is not as easy to solve as the explicit methods discussed above. The reason is that we have to solve for three unknowns at the next time level $n+1$. Therefore, it is impossible to directly calculate the solution at ϕ_i^{n+1} as this requires information of the neighboring grid points ϕ_{i+1}^{n+1} and ϕ_{i-1}^{n+1}, which are also unknown. Collecting the terms at the current time step n on right hand side and the terms at the next time level $n+1$ at the left hand side gives

$$-\alpha\phi_{i+1}^{n+1}+(1+2\alpha)\phi_i^{n+1}-\alpha\phi_{i-1}^{n+1}=\phi_i^n \tag{44}$$

This means that the scheme can be written as a combination of n_x-1 linear equations for n_x-1 unknowns, which means it can be solved uniquely. However, instead of solving for ϕ_i^{n+1} using a trivial formula as in the explicit methods we need to solve the system of equations, subject under the boundary conditions ϕ_0^{n+1} and $\phi_{n_x}^{n+1}$. Therefore, it is convenient to write the scheme (44) in matrix form

$$A\phi^{n+1}=\phi^n \tag{45}$$

Which can be solved directly using

$$\phi^{n+1}=A^{-1}\phi^n \tag{46}$$

For relatively small problems, and on modern computers, this is nowadays possible. However, this is a very naive approach as the the matrix A has a very special structure. Note that only the diagonal, and sub- and superdiagonal have non-zero values. So A is a tridiagonal matrix. In chapter 4 we will discuss some methods that can be used to solve this system of equations more efficiently than using the naive approach. These more advanced methods can reduce the computational time and memory requirement of the eventual code by several orders of magnitude.

A significant advantage of the Euler backward method is that it is unconditionally stable. The proof is left as an exercise to the reader (homework problem 1c). However, unfortunately, this does not mean that time steps can be arbitrarily large. It is still necessary-* to select a time step that is small enough to ensure that the solution is converged. The Euler backward method has a local truncation error of $\mathcal{O}(\Delta t,(\Delta x)^2)$, i.e., the method is first-order in time, and second-order in space.

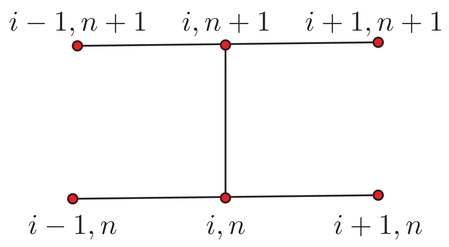

Figure 4: Sketch of the points used in the Crank-Nicolson method.

3.4 Crank-Nicolson Method

A very popular implicit method, introduced by Crank and Nicolson in 1947, is based on the central approximation around the point x_i, $t^{n+1/2}$ as follows

$$\frac{\phi_i^{n+1}-\phi_i^n}{2\frac{\Delta t}{2}} = M\frac{\phi_{i+1}^{n+1/2}-2\phi_i^{n+1/2}+\phi_{i-1}^{n+1/2}}{(\Delta x)^2} \tag{47}$$

The spatial derivatives are approximated by using an average in the points x_i, t^n and x_i, t^{n+1}, which results in

$$\frac{\phi_i^{n+1}-\phi_i^n}{\Delta t} = M\frac{(\phi_{i+1}^{n+1}-2\phi_i^{n+1}+\phi_{i-1}^{n+1})+(\phi_{i+1}^{n}-2\phi_i^{n}+\phi_{i-1}^{n})}{2(\Delta x)^2} \tag{48}$$

Collecting the terms at the current time step n on right hand side and the terms at the next time level $n+1$ on the left hand side, and introducing α, see equation (32), results in

$$-\alpha\phi_{i+1}^{n+1}+2(1+\alpha)\phi_i^{n+1}-\alpha\phi_{i-1}^{n+1} = \alpha\phi_{i+1}^{n}+2(1-\alpha)\phi_i^{n}+\alpha\phi_{i-1}^{n} \tag{49}$$

Just as for the implicit method the Crank-Nicolson method can be written in matrix form

$$A\phi^{n+1} = B\phi^n \tag{50}$$

where the right side at the current time level is fully known. Just as for the Euler backward scheme it can be shown that the Crank-Nicolson method is unconditionally stable. The benefit of the Crank-Nicolson method concerning the Euler backward method is that the local truncation error is second order in space and time ($\mathcal{O}((\Delta t)^2, (\Delta x)^2)$). As a result, the time step that can be used in the Crank-Nicolson Method is larger than in the Euler backward method.

4 Methods to solve system of equations

Above we have seen that both the implicit and the Crank-Nicolson method result in a system of linear equations of the form

$$A\phi^{n+1} = \phi^n, \tag{51}$$

which needs to be solved. While it is possible to calculate the solution directly using

$$\phi^{n+1} = A^{-1}\phi^n. \tag{52}$$

This is not the preferred method. Performing the matrix inversion and the corresponding matrix multiplication requires $\mathcal{O}(n)^3$ operations for a matrix of size n if you use the standard Gaussian elimination method. Better implementations are nowadays available in numerical libraries. In particular, algorithms based on the Strassen algorithm[3], which is included in linear algebra packages such as BLAS/LAPACK used in Matlab, have a computational complexity of $\mathcal{O}(n)^{2.807}$. However, due to the tridiagonal nature of the system of equations, we can do much better. In section 4.1 and 4.2, we discuss the use of LU decomposition and a tridiagonal matrix solver, which solve this problem using $\mathcal{O}(n)$ operations. For large systems, this results in a massive reduction of the computational requirements.

[3]While asymptotically faster algorithms are mathematically possible; only the Strassen algorithm is practically relevant for matrices of sizes that can be considered on actual machines.

4.1 LU decomposition

Given a system of linear equations in matrix form

$$Ax = b, \tag{53}$$

we want to solve the equation for x when A and b are known. Let A be a square matrix. An LU factorization refers to the factorization of A into two factors a lower triangular matrix L and an upper triangular matrix U:

$$A = LU. \tag{54}$$

In the lower triangular matrix all elements above the diagonal are zero, in the upper triangular matrix, all the elements below the diagonal are zero. For example, for a 3×3 matrix A, the LU decomposition looks like this:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

The results from the LU decomposition can be used to solve the system of equations more efficiently. The first step is to assume $y = Ux$, which implies that

$$Ly = b \tag{55}$$

This triangular system has to be solved for y. After that, you have to solve the triangular system

$$Ux = y. \tag{56}$$

for x. In both cases, we are dealing with triangular matrices (L and U). Hence equations (55) and (56) can be solved directly by forward and backward substitution without using Gaussian elimination. The above procedure can be repeatedly applied to solve the equation multiple times for different b. In this case, it is faster (and more convenient) to do an LU decomposition of the matrix A once and then solve the triangular matrices for the different b, rather than using Gaussian elimination each time. Conceptually, you can think of the matrices L and U as having the Gaussian elimination process encoded in them.

Generally, the cost of solving a system of linear equations is approximate $\frac{2}{3}n^3$ floating-point operations if the matrix A has to size n. However, for our tridiagonal system, libraries like BLAS/LAPACK can obtain a solution using $\mathcal{O}(n)$ operations. However, performing an expensive *LU* decomposition and solving equations (55) and (56) is still unnecessarily complicated. It is namely possible to solve the tridiagonal system directly this method we discuss in the next section.

4.2 Tridiagonal solver

The system of equations we wan to solve is a diagonally dominant tridiagonal system of the form

$$-a_i\phi_{i-1} + b_i\phi_i - c_i\phi_{i+1} = d_i \qquad \text{for } i = 1, 2, ..., k \tag{57}$$

with the boundary conditions

$$\phi_0 = 0 \text{ and } \phi_{n_x} = 0. \tag{58}$$

Note that we have omitted the time level n for the unknowns ϕ_i. We assume, which is often the case, that the coefficients a_i, b_i, and c_i satisfy the condition

$$\begin{gathered} a_i > 0, b_i > 0, \text{ and } c_i > 0 \\ b_i > a_i + c_i \end{gathered} \tag{59}$$

These, stronger than necessary, conditions ensure that the the matrix is **diagonally dominant**, with the diagonal element in each row being at least as large as the sum of the absolute values of the other elements.

The tridiagonal solver reduces the system of equations to upper triangular form, by first removing the term ϕ_{i-1} term in each of the equations.

$$\phi_i - e_i\phi_{i+1} = f_i \qquad \text{for } i = 1, 2, ..., k \tag{60}$$

The last of these equation therefore is

$$\phi_k - e_k\phi_{k+1} = f_k \tag{61}$$

and the next equation that is still in its original form is

$$-a_{k+1}\phi_k + b_{k+1}\phi_{k+1} - c_{k+1}\phi_{k+2} = d_{k+1}. \tag{62}$$

ϕ_k can be eliminated from these two equations, which gives

$$\phi_{k+1} - \frac{c_{k+1}}{b_{k+1} - a_{k+1}e_k}\phi_{k+2} = \frac{d_{k+1} + a_{k+1}f_k}{b_{k+1} - a_{k+1}e_k} \tag{63}$$

Comparing this with equation (60) shows that the coefficients e_i and f_i can be obtained from the following recurrence relations

$$e_i = \frac{c_i}{b_i - a_i e_{i-1}}, f_i = \frac{d_i + a_i f_{i-1}}{b_i - a_i e_{i-1}}, \qquad \text{for } i = 1, 2, ..., n_x - 1 \tag{64}$$

The boundary conditions give that

$$e_0 = f_0 = 0 \tag{65}$$

It is important to realize that this tridiagonal solver is not necessarily numerically stable. It requires that the system is diagonally dominant, see equation (59). However, the benefit of this algorithm compared to the LU decomposition is that it further eliminates computational overhead. This solver only requires 3 additions, 3 multiplications, and 2 division operations per mesh point. This should be compared to 2 additions and 2 multiplications per mesh point for the explicit algorithm. So using this tridiagonal matrix solver in the Euler backward method requires about twice as much computational time per grid point than using the Euler forward method. However, remember that the time step in the Euler backward method can be much bigger than in the Euler forward method.

It is essential to realize that adjustments in the above solver are required when you want to consider a periodic system. We do not consider this extension here.

4.3 Tridiagonal solver in matrix notation

This section describes the same procedure as outlined in the previous section, but now in matrix notation. So same information in a different form. Note that the direct tridiagonal matrix solver essentially uses Gaussian elimination and that we need to perform two sweeps, i.e.

- Forward: transfer A into upper triangular form
  $(i = 1, 2, ...n_x - 1)$
- Backward: find solution via back-substitution
  $(i = n_x - 1, n_x - 2, ...1)$

4.3.1 Forward sweep

When we write equation (57) subject to the boundary conditions (equation (58)) in matrix form we obtain

$$\left(\begin{array}{ccccccc|c}
b_1 & -c_1 & & & & & & d_1 \\
-a_2 & b_2 & -c_2 & & & & & d_2 \\
 & -a_3 & b_3 & -c_3 & & & & d_3 \\
 & & \ddots & \ddots & \ddots & & & \\
 & & & \ddots & \ddots & \ddots & & \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & d_{n_x-1}
\end{array}\right)$$

Multiply first row by $1/b_1$ to obtain

$$\begin{pmatrix}
1 & -c_1/b_1 & & & & & & |d_1/b_1 \\
-a_2 & b_2 & -c_2 & & & & & |d_2 \\
 & -a_3 & b_3 & -c_3 & & & & |d_3 \\
 & & \ddots & \ddots & \ddots & & & | \\
 & & & \ddots & \ddots & \ddots & & | \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & |d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & |d_{n_x-1}
\end{pmatrix}$$

For convenience define

$$e_1 = \frac{c_1}{b_1} \qquad ; \qquad f_1 = \frac{d_1}{b_1} \tag{66}$$

This gives ..

$$\begin{pmatrix}
1 & -e_1 & & & & & & |f_1 \\
-a_2 & b_2 & -c_2 & & & & & |d_2 \\
 & -a_3 & b_3 & -c_3 & & & & |d_3 \\
 & & \ddots & \ddots & \ddots & & & | \\
 & & & \ddots & \ddots & \ddots & & | \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & |d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & |d_{n_x-1}
\end{pmatrix}$$

to eliminate $-a_2$ multiply first row by a_2 and add to the second row, which gives

$$\begin{pmatrix}
1 & -e_1 & & & & & & |f_1 \\
0 & (b_2 - a_2 e_1) & -c_2 & & & & & |d_2 + a_2 f_1 \\
 & -a_3 & b_3 & -c_3 & & & & |d_3 \\
 & & \ddots & \ddots & \ddots & & & | \\
 & & & \ddots & \ddots & \ddots & & | \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & |d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & |d_{n_x-1}
\end{pmatrix}$$

to obtain 1 on the diagonal of the second row, multiply by

$$\frac{1}{b_2 - a_2 e_1} \tag{67}$$

$$\begin{pmatrix}
1 & -e_1 & & & & & & |f_1 \\
0 & 1 & \frac{-c_2}{b_2 - a_2 e_1} & & & & & |\frac{d_2 + a_2 f_1}{b_2 - a_2 e_1} \\
 & -a_3 & b_3 & -c_3 & & & & |d_3 \\
 & & \ddots & \ddots & \ddots & & & | \\
 & & & \ddots & \ddots & \ddots & & | \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & |d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & |d_{n_x-1}
\end{pmatrix}$$

For convenience define

$$e_2 = \frac{c_2}{b_2 - a_2 e_1} \qquad ; \qquad f_2 = \frac{d_2 + a_2 f_1}{b_2 - a_2 e_1} \tag{68}$$

$$\left(\begin{array}{ccccccc|c}
1 & -e_1 & & & & & & f_1 \\
0 & 1 & -e_2 & & & & & f_2 \\
 & -a_3 & b_3 & -c_3 & & & & d_3 \\
 & & \ddots & \ddots & \ddots & & & \\
 & & & \ddots & \ddots & \ddots & & \\
 & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & d_{n_x-2} \\
 & & & & & -a_{n_x-1} & b_{n_x-1} & d_{n_x-1}
\end{array}\right)$$

Keep following the same procedure to obtain after k steps

$$\left(\begin{array}{cccccccccc|c}
1 & -e_1 & & & & & & & & & f_1 \\
0 & 1 & -e_2 & & & & & & & & f_2 \\
 & 0 & 1 & -e_3 & & & & & & & f_3 \\
 & & \ddots & \ddots & \ddots & & & & & & \\
 & & & 0 & 1 & -e_k & & & & & f_k \\
 & & & & -a_{k+1} & b_k+1 & -c_{k+1} & & & & d_{k+1} \\
 & & & & & \ddots & \ddots & \ddots & & & \\
 & & & & & & -a_{n_x-2} & b_{n_x-2} & -c_{n_x-2} & & d_{n_x-2} \\
 & & & & & & & -a_{n_x-1} & b_{n_x-1} & & d_{n_x-1}
\end{array}\right)$$

This means that for step $k+1$.

$$\phi_k - e_k\phi_{k+1} = f_k \qquad (69)$$

$$-a_{k+1}\phi_k + b_{k+1}\phi_{k+1} - c_{k+1}\phi_{k+2} = d_{k+1} \qquad (70)$$

which implies

$$\phi_{k+1} - \frac{c_{k+1}}{b_{k+1} - a_{k+1}e_k}\phi_{k+2} = \frac{d_{k+1} + a_{k+1}f_k}{b_{k+1} - a_{k+1}e_k} \qquad (71)$$

This means that e_i and f_i can be obtained as follows

$$e_i = \frac{c_i}{b_i - a_ie_{i-1}}, f_i = \frac{d_i + a_if_{i-1}}{b_i - a_ie_{i-1}}, \qquad \text{for } i = 1, 2, ..., n_x - 1 \qquad (72)$$

The boundary conditions give that

$$e_0 = f_0 = 0 \qquad (73)$$

4.3.2 Obtain solution using back substitution

After forward sweep you have an upper tridiagonal matrix and you obtain solution using back substitution. The upper tridiagonal from gives ϕ_k recursively using

$$\phi_k - e_k\phi_{k+1} = f_k \qquad (74)$$

Which implies that

$$\phi_k = f_k + e_k\phi_{k+1} \qquad \text{for } k = n_x - 1, n_x - 2, ..., 1 \qquad (75)$$

4.3.3 Final considerations

Please be aware of the following aspects, conditions, and limitations of these method. These have been mentioned above in passing and are summarized here for completeness and clarity.

- Remember that the tridiagonal matrix needs to be diagonally dominant. this is ensured when $a_j > 0, b_j > 0, c_j > 0$ and $b_j > a_j + c_j$, which gives $|e_i| < 1$.
- Recurrence may lead to errors if $|e_i| > 1$.

- The method needs to be adjusted when periodic boundary conditions are used (will not be considered in this course).
- Solution of linear system makes implicit scheme about twice as expensive per time step compared to explicit scheme.

5 Errors and convergence

In this section, we discuss how we analyze the convergence of the different methods discussed above. Before discussing the way we calculate the error in section 5.2, it is useful to give some definitions first:

5.1 Definitions of terminology

Truncation error:
The truncation error is caused by using the finite difference approximations. In chapter 2, we have seen that the local truncation error can be found by using a Taylor series expansion. The truncation error can be reduced by reducing Δx or by switching to a higher-order approximation. Analysis of the local truncation error gives essential information about the theoretical characteristics of a numerical scheme.

Roundoff error:
Roundoff errors are introduced in the solution because computers represent real numbers using a finite number of significant digits. Roundoff errors can become problematic when you multiply or divide by a huge or tiny number. In many cases, this can be prevented by using an appropriate normalization of the equations. A significant roundoff error can also be introduced when you have to subtract two nearly identical numbers. This can, for example, happen if Δx is too small. This means that the effort to decrease truncation errors can have unintended consequences by introducing significant roundoff errors.

Consistent:
A numerical method is considered consistent if the finite difference representation converges to the real solution of the partial differential equation when the grid spacing and the time step are reduced to zero.

Convergent:
A method is considered to be convergent when the difference between the numerical and real solution converges monotonically to zero as the spatial and temporal discretization are reduced to zero.

Stability:
A numerical method is stable when the difference between the numerical and the real solution remains bounded as the number of time steps increases to infinity. Note that this does not necessarily mean that the solution is accurate.

5.2 Calculation of errors

To analyze the error of the simulations we first define an error vector as follows

$$e(x,t) = \phi_{\text{numerical}}(x,t) - \phi_{\text{analytical}}(x,t) \tag{76}$$

The error array e can be used to determine the L_1, L_2, and L_∞ norms.

5.2.1 L_1 norm

The L_1-norm is defined as the sum of the absolute values of the error vector we defined above, i.e.

$$\|e\|_1 = \sum_{i=1}^{n_x} |e_i| \tag{77}$$

5.2.2 L_2 norm; L_p norm

The L_2 is defined as the square root of the sum of the squares of the absolute values of its components:

$$\|e\|_2 = \sqrt{\sum_{i=1}^{n_x} |e_i|^2} \tag{78}$$

Note that the absolute sign is there as the general L_p norm is defined as

$$\|e\|_p = \left(\sum_{i=1}^{n_x} |e_i|^p\right)^{1/p} \tag{79}$$

5.2.3 L_∞ norm

The infinity norm is defined as the maximum of the absolute values of its components:

$$\|e\|_\infty = \max_{i=1..n_x} |e_i| \tag{80}$$

Note that these norms determine the *global error.* The global error is an accumulation of the local truncation errors committed in each step up the point in time. It is more difficult to estimate the global error than the local truncation error. Here it is sufficient to know that, in general, the convergence of the global error is one order less than the local truncation error. This means, for example, that the convergence of the global error for the Euler forward method is $\mathcal{O}(\Delta x)$, while the leading-order local truncation is of order $\mathcal{O}(\Delta x)^2$. Therefore, the Euler forward method is considered to be a first-order method, i.e., when Δx reduces by a factor 2 the error reduces by a factor of roughly 2.

5.2.4 What norm to pick?

Note that there is some important qualitative difference between the different forms:

- Small entries in the error vector (76) contribute more to the L_1 norm than to the L_2 norm.
- Large entries in the error vector (76) contribute more to the L_2 norm than to the L_1 norm.
- The infinity norm looks at the maximum error at any location in the domain.

Thus the L_1 norm is a good choice when we prefer a few large errors, i.e., minimizing the L_1 norm may lead to many small entries in the error vector and a few large entries. Minimizing the L_2 error ensures that large outliers are prevented. Minimizing the L_∞ norm is the strictest way to verify the accuracy of your numerical scheme.

6 Two-dimensional simulations

Considering the success of the Crank-Nicholson method for the one-dimensional problem, it is natural to extend it to the two-dimensional system. In two dimensions, the Crank-Nicholson method reads

$$\begin{aligned}
\phi_{i,j}^{n+1} &= \phi_{i,j}^{n} + \\
&\quad +\alpha_x \left(\phi_{i+1,j}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i-1,j}^{n+1}\right) \\
&\quad +\alpha_x \left(\phi_{i+1,j}^{n} - 2\phi_{i,j}^{n} + \phi_{i-1,j}^{n}\right) \\
&\quad +\alpha_y \left(\phi_{i,j+1}^{n+1} - 2\phi_{i,j}^{n+1} + \phi_{i,j-1}^{n+1}\right) \\
&\quad +\alpha_y \left(\phi_{i,j+1}^{n} - 2\phi_{i,j}^{n} + \phi_{i,j-1}^{n}\right)
\end{aligned} \tag{81}$$

where $\alpha_x = M\frac{\Delta t}{(\Delta x)^2}$ and $\alpha_y = M\frac{\Delta t}{(\Delta y)^2}$. Instead of writing out the discretization for the Laplacian, it is easier to denote the corresponding terms using $\alpha_x\delta_x^2$ and $\alpha_y\delta_y^2$, respectively. Using this simplification in notation, we can write the above numerical scheme as follows:

$$(1 - \alpha_x\delta_x^2 - \alpha_y\delta_y^2)\phi^{n+1} = (1 + \alpha_x\delta_x^2 + \alpha_y\delta_y^2)\phi^n \tag{82}$$

In one dimension, the computational time that is required to solve the set of linear equations for the Crank-Nicholson method is relatively limited, but this was easily compensated by the much larger time step that is allowed by this method. However, for the two-dimensional scheme above, one has to solve a linear system of $(N_x - 1)(N_y - 1)$ equations. An additional complication is that the matrix structure is not tridiagonal anymore. This means that it is very computationally intensive to solve the above system of equation. While on modern computers it is by no means out of the question, it is possible to solve this problem more efficiently.

We have seen above that it is possible to solve an implicit method in one direction efficiently. Thus, we will try to implement an implicit scheme in one direction instead of two. For example, using an implicit method in the x-direction reads:

$$(1 - \alpha_x\delta_x^2)\phi^{n+1} = (1 + \alpha_x\delta_x^2 + 2\alpha_y\delta_y^2)\phi^n \tag{83}$$

i.e. $N_y - 1$ sets of linear tridiagonal systems as we had before. Alternatively, we can use an implicit scheme in the y-direction as follows:

$$(1 - \alpha_y\delta_y^2)\phi^{n+1} = (1 + 2\alpha_x\delta_x^2 + \alpha_y\delta_y^2)\phi^n \tag{84}$$

i.e. $N_x - 1$ sets of linear tridiagonal systems. While this improves the stability of the method, it is not ideal. An analysis of the stability namely reveals that, while there are no stability restrictions in the implicit direction, there is still a severe time step restriction in the explicit direction.

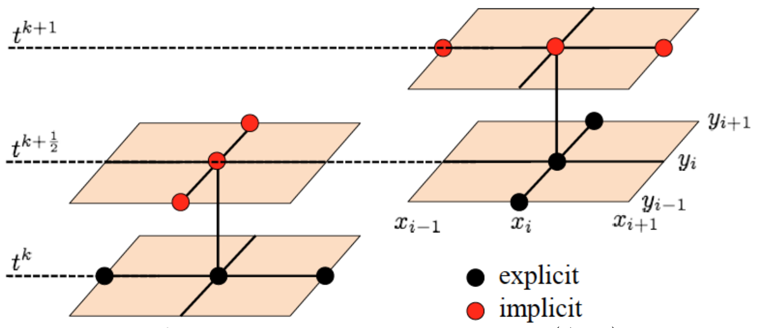

Figure 5: Visual impression of "alternating direction implicit" (ADI) method in which each time step is split up in two parts. In one part of the time step computations are explicit in x and implicit in y while in the second part of the time step it is the other way around.

It turns out that it is possible to combine the "implicit in x" and "implicit in y" schemes conveniently. This idea is known as the "alternating direction implicit" (ADI) method, see figure 5 for a visual impression. In this method, the problem is solved by alternating the direction in which the implicit scheme is used. Such a scheme was first proposed in 1955 by Peaceman and Rachford. The first step in its derivation is to factor the full Crank-Nicholson scheme, see equation (82), as follows

$$(1 - \alpha_x\delta_x^2)(1 - \alpha_y\delta_y^2)\phi^{n+1} = (1 + \alpha_x\delta_x^2)(1 + \alpha_y\delta_y^2)\phi^n \tag{85}$$

Notice that the expansion of the products is not exactly the same as in the original scheme as writing out the factorizations gives

$$(1 + \alpha_x\delta_x^2)(1 + \alpha_y\delta_y^2) = 1 + \alpha_x\delta_x^2 + \alpha_y\delta_y^2 + \alpha_y\alpha_x\delta_x^2\delta_y^2, \tag{86}$$

$$(1 - \alpha_x\delta_x^2)(1 - \alpha_y\delta_y^2) = 1 - \alpha_x\delta_x^2 - \alpha_y\delta_y^2 + \alpha_y\alpha_x\delta_x^2\delta_y^2. \tag{87}$$

This means that the factorization introduces a difference of $\alpha_y \alpha_x \delta_x^2 \delta_y^2$. However, it can be shown that this term is of similar magnitude as some of the leading order truncation errors and therefore this error does not severely affect the performance. When we introduce an intermediate time level $n + 1/2$ we can rewrite equation (85) as follows, again for a visual impression see figure 5.

$$\begin{aligned}(1 - \alpha_x \delta_x^2)\phi^{n+1/2} &= (1 + \alpha_y \delta_y^2)\phi^n \qquad (88)\\ (1 - \alpha_y \delta_y^2)\phi^{n+1} &= (1 + \alpha_x \delta_x^2)\phi^{n+1/2} \qquad (89)\end{aligned}$$

Solving the above system of equations requires one to solve $N_y - 1$ tridiagonal systems (of length $N_x - 1$) and $N_x - 1$ tridiagonal systems (of length $N_y - 1$), separately. This is significantly less work than solving the full linear system of $(N_x - 1)(N_y - 1)$ equations. By multiplying the equations in the above system with $(1 + \alpha_x \delta_x^2)$ and $(1 - \alpha_x \delta_x^2)$ as is indicated below one can show that the system is in fact equivalent to equation (85)

$$\begin{aligned}(1 - \alpha_x \delta_x^2)\phi^{n+1/2} &= (1 + \alpha_y \delta_y^2)\phi^n \qquad &&| \quad (1 + \alpha_x \delta_x^2)\\ (1 - \alpha_y \delta_y^2)\phi^{n+1} &= (1 + \alpha_x \delta_x^2)\phi^{n+1/2} \qquad &&| \quad (1 - \alpha_x \delta_x^2)\end{aligned}$$

which gives

$$\begin{aligned}(1 + \alpha_x \delta_x^2)(1 - \alpha_x \delta_x^2)\phi^{n+1/2} &= (1 + \alpha_x \delta_x^2)(1 + \alpha_y \delta_y^2)\phi^n \qquad (90)\\ (1 - \alpha_x \delta_x^2)(1 - \alpha_y \delta_y^2)\phi^{n+1} &= (1 - \alpha_x \delta_x^2)(1 + \alpha_x \delta_x^2)\phi^{n+1/2} \qquad (91)\end{aligned}$$

Here you can recognize that the $\phi^{n+1/2}$ is the same in both cases, which means

$$(1 - \alpha_x \delta_x^2)(1 - \alpha_y \delta_y^2)\phi^{n+1} = (1 + \alpha_x \delta_x^2)(1 + \alpha_y \delta_y^2)\phi^n \qquad (92)$$

which is identical to equation (85).