Case study I: Models of transport, waves, and shallow waters

Lab 1. Introduction to the finite volume methods

This lab constitutes an introduction to some basic numerical schemes for conservation laws. In Lab 2 of this case study, we will apply these schemes and simulate the flow resulting from opening a sluice gate between two water reservoirs at different water levels.

Before starting to work on the exercises, please make sure that you have carefully read the lecture notes Conservation Laws, Transport Phenomena, and the Finite Volume Method!

We study here conservation laws in one space and one time variable,

\[ u_t + f(u)_x = 0 \]  \hspace{1cm} (differential form), \hspace{1cm} (1a)

\[ \frac{d}{dt} \int_a^b u \, dx + f(u(b, t)) - f(u(a, t)) = 0 \]  \hspace{1cm} (integral form), \hspace{1cm} (1b)

where \( a < b \) are arbitrary points in the spatial region of interest. The conservation laws will be considered in the context of an initial-value or an initial–boundary-value problem. We will solve these problems numerically by time stepping. As illustrated in figure 1, we successively compute the solution in horizontal layers; the bottom layer is given by the initial condition, and the solution is computed layer by layer by moving upward one time step at a time. Moreover, for practical reasons of storage, we need to restrict the computations to a bounded spatial region, so there is usually a need to impose boundary conditions at the left and/or right boundaries in figure 1.

![Figure 1. The finite volume scheme computes approximations \( u^n_i \) of cell averages in cells of width \( \Delta x \) at times \( t_n = n \Delta t, n = 0, 1, \ldots \). The cell averages are computed horizontally, layer by layer, starting with the lowest one.](image-url)

In the computer program, there are a few ways of handling the storage of the solution. One alternative is to store the full solution in a two-dimensional array, where the dimensions signify space and time, respectively. This storage scheme will work fine in this exercise, but it would be very memory consuming for two- and three-dimensional problems; the full solution is therefore rarely stored for problems with more than one space dimension. Since the schemes we use only requires the solution from one earlier time step when computing the solution at the new time step, a much more memory-effective alternative is to overwrite the same vector \( \mathbf{u} \), with dimension equal to the number of spatial points that are considered, with new data at each time steps. It is also possible to use two vectors \( \mathbf{u}_{\text{old}} \) and \( \mathbf{u}_{\text{new}} \) that hold data from the old and new time steps, respectively. You are free to use either method in your implementation.
1 Computer exercises

Hints and details on how to carry out the Matlab implementation are given in section 2.

1. Consider the transport equation

\[ u_t + cu_x = 0, \]  

where \( c \) is a positive scalar constant, with periodic boundary conditions \( u(0, t) = u(1, t) \) and initial condition \( u(x, 0) = \sin 2\pi x \). For \( c = 2 \), solve the above equation

(a) exactly,

(b) either with the upwind scheme or the Lax–Friedrichs method (these methods behave similarly),

(c) with the Richtmyer two-step Lax–Wendroff method.

Test various combinations of time and space discretizations in the above schemes and compare with the exact solution. For a fixed space discretization, change \( \Delta t \) and find the smallest so-called Courant number, \( c\Delta t/\Delta x \), for which the numerical solution starts behaving unstably. Change \( c \) and redo the above experiment. Does the smallest Courant number for which the system behaves unstable seem to depend on \( c, \Delta t, \) or \( \Delta x \)?

2. In problem 1, we investigated the behavior of the numerical schemes for smooth initial data. Here, we will study how the schemes behave under the discontinuous initial condition

\[ u(x, 0) = \begin{cases} 
1 & \text{if } x < 0.5, \\
0 & \text{else.} 
\end{cases} \]  

We will define the solution to equation (2) under the initial condition (3) as the transport of the initial condition along the characteristics. This solution satisfies the equation in the integral form (1b) and constitutes a weak solution to equation (2). Note that the solution is nondifferentiable, so it does not satisfy the differential form of the equation in a classical sense. Also note that there are two discontinuities; one initially at \( x = 0.5 \), due to definition (3), and one initially at \( x = 0 \) (or \( x = 1 \)), due to the periodic boundary conditions. Test also here various combinations of time and space discretizations and compare with the exact solution.

3. Consider Burgers’ equation

\[ u_t + f(u)_x = 0, \]  

where \( f(u) = u^2/2 \). We want to solve this equation on the spatial interval \((0, 1)\) with initial conditions \( u(0, x) = 1 + 0.7 \sin(10x) \) and boundary condition \( u(t, 0) = 1 \).

(a) Why is no boundary condition needed on the right boundary \((x = 1)\)?

(b) Roughly sketch the characteristics in \((x, t)\) space for the above boundary conditions. Try to anticipate how the solution will develop in different regions of the interval \((0, 1)\).

(c) Solve the problem numerically using the upwind scheme and the two-step Lax–Wendroff method.

4. In the above exercises, we considered two types of schemes, the Lax–Friedrich and the first-order upwind schemes on one hand, and the second-order Lax–Wendroff scheme on the other. The properties of these schemes were studied on smooth solutions as well as solutions involving jump discontinuities. From the results, compare the schemes and draw conclusions (pros and cons) with the different schemes applied to different types of problems.
2 Implementation instructions and hints

2.1 General instructions and hints for all exercises

• There are a number of different ways to carry out the Matlab implementation, and as usual, it is a good idea to think first before starting to code! Here are some implementation options:
  – Write a separate Matlab function for each problem.
  – Write a single function with options for the different cases.
  – A more advanced approach is to write separate functions for the numerical schemes (upwind, Lax–Friedrichs, and Lax–Wendroff). Having written these functions, the rest of the computer exercise reduces to writing flux functions for the two equations involved and experimenting by varying the initial condition, $\Delta x$, and $\Delta t$.

• All problems above should be solved on the interval $[0, 1]$ using $M$ cells of equal size $\Delta x$. Let the number of cells $M$ and the time–space step ratio $\lambda = \Delta t/\Delta x$ be input parameters.

• A warning: it is easy to get confused with the geometry and the indexing here! Take a close look at the geometry layout in figure 1. Recall that the unknown $u^n_i$ approximates an average over the computational cell $(x_{i-1/2}, x_{i+1/2})$, and that $u^n_i$ therefore is naturally associated with the cell center $x_i$. From figure 1, we see that the cell centers are located at $\Delta x/2, 3\Delta x/2, \ldots$, and the cell interfaces at $x_{i+1/2} = i \Delta x, i = 0, 1, \ldots, M$.

• When visualizing the solution, you can simply plot $u^n_1, \ldots, u^n_M$ against cell center coordinates $x_1, \ldots, x_M$. A vector $xcc$ with cell center coordinates can be created either with the command $xcc = \text{linspace}(dx/2, 1-dx/2, M)$ or $xcc = dx/2:dx:1-dx/2$, where $dx$ holds the value $\Delta x$. (Recall that a Matlab command like $0:0.3:1$ yields a vector containing floating point approximations of the numbers 0, 0.3, 0.6, and 0.9.) The command $\text{plot}(xcc, u, '*$') plots the cell center values contained in vector $u$ against the values $xcc$.

• When defining the numerical initial condition, you may simply evaluate the given initial condition formula in the cell centers. This strategy corresponds to approximating the cell averages of the initial condition using the mid-point quadrature rule in the cell-average integral.

2.2 Specific instructions and hints for exercise number $[n]$

[1,2] Use that the solution is constant along the characteristics to solve the transport problems exactly.

[1,2] When solving problems with periodic boundary conditions, think of the cells as being periodically wrapped, so that cell 1 has cell $M$ as its left neighbor and cell $M$ has cell 1 as its right neighbor. In Matlab, if the vector $u$ contains elements $u_1, u_2, \ldots, u_M$, the vector $u([2:end 1])$ will contain elements $u_2, u_3, \ldots, u_M, u_1$, that is, $u([2:end 1])$ contains the elements of $u$ periodically shifted one position to the left. Similarly $u([end 1:end-1])$ contains $u$ periodically shifted one position to the right. Using this trick, you can avoid loops for spatial indices and perform $u^n_{i+1}$ and $u^n_{i-1}$ with built-in periodic boundary conditions, for all $i$ simultaneously!

[2] The periodic square wave initial condition can in Matlab be written in a single line by defining a so-called anonymous function

$$f = \Phi(x) \text{ round}(1/2+\sin(2*\pi*x)/2);$$

The variable $f$ is called a function handle, $f(x)$ returns the value of the function at the point $x$, and $f$ can be passed as an input variable to a Matlab function defined in an m-file.

[3] Normally, when using the upwind scheme for nonlinear conservation laws, the sign of the local wave speed $f'$ needs to be estimated at each cell interface in order to know from which side the flux should be computed. However, with the current initial condition, the wave speed will be positive at all times, which means that a flux from the left may be used everywhere without checking $f'$. 

3
Note that the boundary condition \( u(0, t) = 1 \) should be used for this problem (and not the periodic boundary condition of exercises 1 and 2). One convenient way of implementing this condition is to imagine a ghost cell \( u_{-1}^n \) centered at \( x_{-1} = -\Delta x/2 \), and to assume that the average of the ghost cell value and the value in the first cell should be equal to the given boundary condition (\( = 1 \) in this case):

\[
\frac{u_{-1}^n + u_1^n}{2} = u_{BC}^n = 1.
\]

The ghost value \( u_{-1}^n = 2u_{BC}^n - u_1^n = 2 - u_1^n \) is then used in the evaluation of the numerical flux at the left cell interface in the first cell.

Although there is no boundary condition at the right boundary \( x = 1 \), the Lax–Wendroff method nevertheless requires information from both neighboring cells, that is, the method needs to access \( u_{M+1}^n \). This problem can also be handled by imagining a ghost cell, now located to the right of the last cell. The value of \( u \) in this \((M + 1)\)th cell may be set using linear interpolation based on the values in cells \( M \) and \( M - 1 \), that is \( u_{M+1}^n = 2u_M^n - u_{M-1}^n \).

Ghost cells can be implemented as follows in Matlab. Assume that we are in the case of the Lax–Wendroff method where we need both \( u_{-1}^n \) and \( u_{M+1}^n \). Define a Matlab vector \( u \) of length \( M + 2 \) that will hold the solution and the ghost values at current time level \( n \). The element \( u(1) \) will contain the left-side ghost value \( u_{-1}^n \), the element \( u(M + 2) \) the right-side ghost value \( u_{M+1}^n \), and \( u(2), \ldots, u(M + 1) \) will contain cell values \( u_1^n, \ldots, u_M^n \). At each time level, start by setting the values \( u(2), \ldots, u(M + 1) \) using the finite-volume scheme (or the initial condition when \( n = 0 \)). Then set the ghost cell values by the formulas \( u(M + 2) = 2u(M + 1) - u(M) \) and \( u(1) = 2 - u(2) \)