Project I report

Finite difference methods for the shallow water equations

Authors:	Nicolò Giosuè Carlo Viscusi
	Francesco Sala
Date:	December 8, 2023
Course:	Numerical Methods for Conservation Laws - MATH-459
Professor:	Martin Licht
Assistant:	Fernando Henriquez

Contents

1 Introduction

In this project, a finite difference scheme was implemented to solve the shallow water equations:

$$\partial_t \begin{bmatrix} h \\ m \end{bmatrix} + \partial_x \begin{bmatrix} m \\ \frac{m^2}{h} + \frac{1}{2}gh^2 \end{bmatrix} = \mathbf{S}(x,t) \tag{1}$$

where $h(x,t)$ is the depth of the water, $m(x,t)$ is the discharge, g is the gravitational constant and $\mathbf{S}(x,t)$[1] is a source term (vector). This system of conservation laws is solved by considering appropriate boundary conditions (BCs) and initial conditions. Note that Eq. 1 can be rewritten in a more compact form as follows:

$$\partial_t \mathbf{q} + \partial_x \mathbf{f}(\mathbf{q}) = \mathbf{S}(x,t) \tag{2}$$

where $\mathbf{q}(x,t) = [h\ m]^T$. In what follows, this differential problem was solved in the spatial domain $\Omega = [0,2]$ and in a temporal domain that will be defined in each section. All the implemented code can be found here on GitHub.

2 Part I

In the first part, a quick overview of the implementation is proposed, as well as a toy example of an application for which an exact solution is available.

2.1 About the implementation

In the first part of the project, a finite difference scheme making use of Lax-Friedrichs flux as a numerical flux was implemented. To accomplish this result, the domain Ω was discretized by defining $N+1$ nodes: let $\Delta x = \frac{2}{N}$ be the space between two contiguous nodes, then the nodes are $x_j = j\Delta x$, $j = 1, \ldots, N+1$. Similarly, let the final time be $T = 0.5$; the temporal domain was discretized using $M+1$ equally spaced nodes $t^n = nk$, $k = 1, \ldots, M+1$ and with $k = \frac{T}{M}$. A conservative finite difference scheme can be written in the following form:

$$\mathbf{q}_j^{n+1} = \mathbf{q}_j^n - \frac{k}{\Delta x}(\mathbf{F}_{j+1/2}^n - \mathbf{F}_{j-1/2}^n) \tag{3}$$

where $\mathbf{F}$ is a numerical flux to be adequately chosen. All the numerical fluxes considered in this project make use of the evaluation of the flux in two contiguous cells, i.e. $\mathbf{F}_{j+1/2}^n = \mathbf{F}(\mathbf{q}_j^n, \mathbf{q}_{j+1}^n)$. In particular, the Lax-Friedrichs flux reads:

$$\mathbf{F}(\mathbf{u},\mathbf{v}) = \frac{\mathbf{f}(\mathbf{u}) + \mathbf{f}(\mathbf{v})}{2} - \frac{\Delta x}{k}\frac{(\mathbf{v}-\mathbf{u})}{2} \tag{4}$$

The scheme defined by Eq. 3 is an explicit conservative scheme: to update the solution at node j, only the solution at the previous time step and nodes j, $j-1$ and $j+1$ is needed.
Concerning boundary conditions, two different scenarios were implemented. The first one is the case of periodic boundary conditions, that is, at each time step, the numerical solution at the first ($j=1$) and last ($j=N+1$) node of the spatial domain is updated by introducing two fictitious "external nodes" x_0 and x_{N+2} to update the solution in the same manner as for the interior nodes, and by imposing:

$$\mathbf{q}_0^n = \mathbf{q}_N^n, \quad \mathbf{q}_{N+2}^n = \mathbf{q}_1^n \tag{5}$$

On the other hand, a slightly different approach is used when specifying open boundary conditions: this time, at each time step, the numerical solution at the first ($j=1$) and last ($j=N+1$) node of the

[1] In this context, a bold symbol is used for vectorial quantities.

spatial domain is updated by introducing again two fictitious "external nodes" x_0 and x_{N+2}, but this time imposing:

$$\mathbf{q}_0^n = \mathbf{q}_1^n, \quad \mathbf{q}_{N+2}^n = \mathbf{q}_{N+1}^n \tag{6}$$

The script `conservative_scheme.m` takes care of solving the problem. It takes as input the space and time domains, i.e. `xspan` as $= [a, b]$ and `tspan` as $[t_0, T]$, respectively. Then, the user must specify the number of space and time intervals, `N` and `K`, in which the specified domains will be discretized, and the initial conditions for each variable `h0` and `m0`. In the end, the numerical flux (`numerical_flux`), the physical flux (`flux_phys`), the source term `S` and the boundary condition option `bc` (either `'peri'` or `'open'`) must be specified too. The problem is then solved, and the `(N + 1) x (K + 1)` matrices `h` and `m` (containing the solution) are returned as outputs, as well as the discretized space and time vectors (`xvec` and `tvec`).

2.2 A first resolution

In the present section, the previously implemented method is tested against a problem for which an exact solution is available. The following smooth functions give the initial conditions for the problem:

$$h(x, 0) = h_0(x) = 1 + 0.5\sin(\pi x), \quad m(x, 0) = m_0(x) = uh_0(x) \tag{7}$$

where u is the horizontal velocity, set here constant and equal to 0.25. The source term reads:

$$\mathbf{S}(x, t) = \begin{bmatrix} \frac{\pi}{2}(u-1)\cos\left(\pi(x-t)\right) \\ \frac{\pi}{2}(-u + u^2 + gh_0(x-t))\cos\left(\pi(x-t)\right) \end{bmatrix} \tag{8}$$

Using the method of characteristics, the exact solution to this problem can be readily computed:

$$h(x, t) = h_0(x - t), \quad m(x, t) = uh(x, t) \tag{9}$$

Regarding the numerical computations, periodic boundary conditions are selected (`bc` = `'peri'`), and the time step is evaluated according to:

$$k = \text{CFL}\,\frac{\Delta x}{\max_i(|u_i| + \sqrt{gh_i})} \tag{10}$$

with CFL = 0.5. The exact and the numerical solutions at the last time step $T = 0.5$ are plotted in Fig. 1. Note that the difference between the exact and numerical solution for $h(x,\, 0.5)$ is barely noticeable, while for $m(x,\, 0.5)$ the accumulation of numerical error probably makes the distinction more clear.

2.3 Error analysis

The order of convergence of the scheme is assessed by solving the problem multiple times varying the value of Δx, such that $\Delta x = 2^i$, $i = -1, -2, \ldots, -7$. Then, for each Δx the ℓ_2 norm of the error is computed and stored in the vectors `err_h_vec` and `err_m_vec`. The log-log plot of such a result is shown in Fig. 2. In both cases it is possible to note that the solution converges with order 1 in space: this comes as no surprise since the adopted FD scheme makes use of order 1 stencils for the approximation of spatial derivatives. Actually, as thoroughly discussed in [1], the Lax-Friedrichs scheme happens to be a $\mathcal{O}(\Delta x + k)$ scheme, i.e. first-order accurate in both space and time.

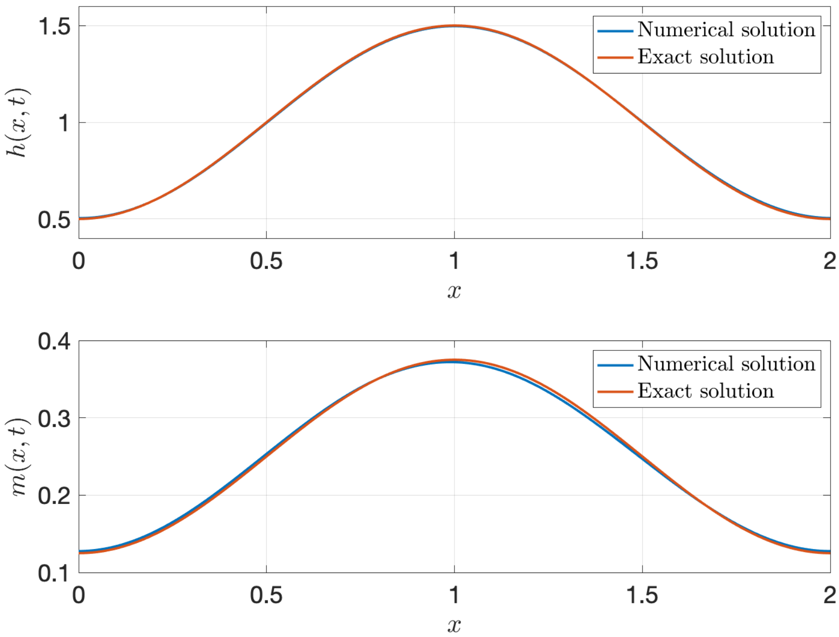

Figure 1: Comparison between exact and numerical solution for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom).

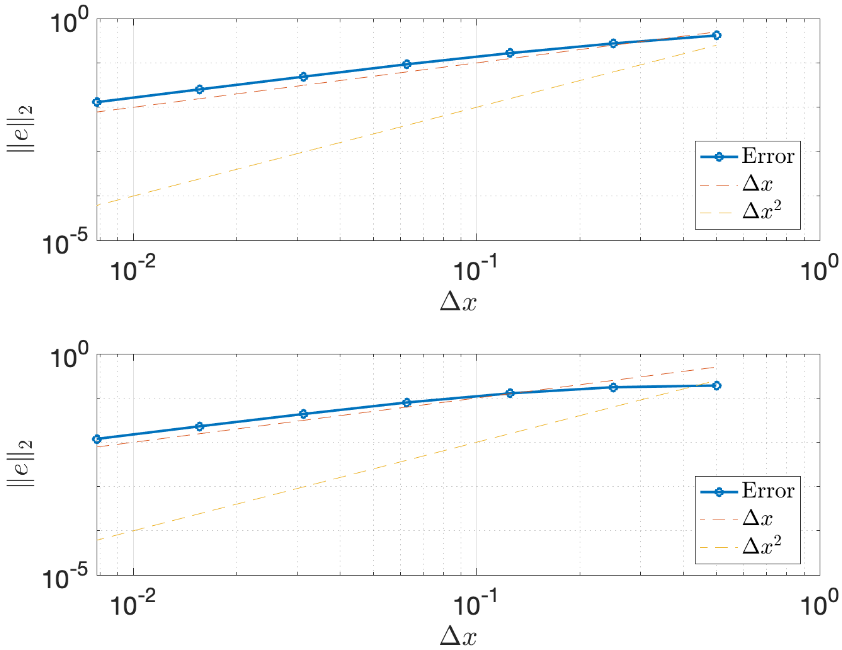

Figure 2: Log-log plot of the ℓ_2 norm of the error between exact and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). Order 1 and 2 convergence rates are also plotted, to allow for comparison.

3 Part II

In the present section, the implemented numerical scheme is tested once again, but with different parameters and no exact solution available. In particular, two sets of initial conditions are considered.

The first set reads:

$$h(x,0) = h_0(x) = 1 - 0.1\sin(\pi x), \quad m(x,0) = m_0(x) = 0 \tag{11}$$

while the second set of initial conditions writes:

$$h(x,0) = h_0(x) = 1 - 0.2\sin(2\pi x), \quad m(x,0) = m_0(x) = 0.5 \tag{12}$$

In both cases, periodic boundary conditions are considered and the source term is set to zero, i.e. $\mathbf{S} = \mathbf{0}$.

3.1 Numerical resolution

The results for the first and second set of initial conditions at the last time step $t = 0.5$ are plotted in Fig. 3 and Fig. 4 respectively.

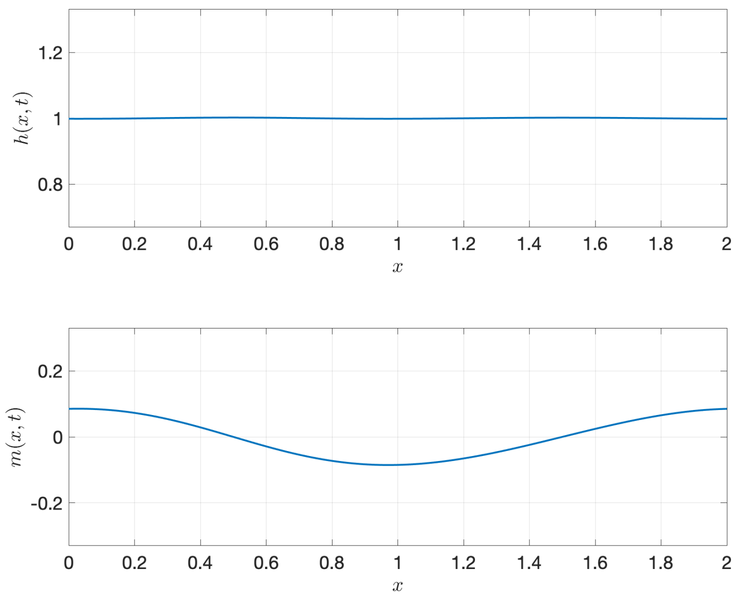

Figure 3: Numerical solution for $h(x,\, 0.5)$ (top) and $m(x,\, 0.5)$ (bottom) for the first set of initial conditions shown in Eq. 11.

Again, the numerical solution computed is extremely regular also with a reasonably coarse spatial discretization. The Lax-Friedrichs scheme, known to be monotone (and consequently also ℓ_1-contractive, total variation diminishing and monotonicity preserving, as specified in [1]) performs very well with the given smooth initial conditions.

3.2 Error analysis

As previously mentioned, since no exact solution is available for the case at hand, a slightly different approach for the error analysis is needed. In this case, the order of convergence of the scheme is assessed by solving the given problem multiple times varying the value of Δx, such that $\Delta x = 2^i$, $i = -1, -2, \ldots, -8$, as usual. Then, for each Δx the ℓ_2 norm of the error is computed, using two numerical solutions: the one obtained with the chosen Δx, and, as reference solution, the numerical

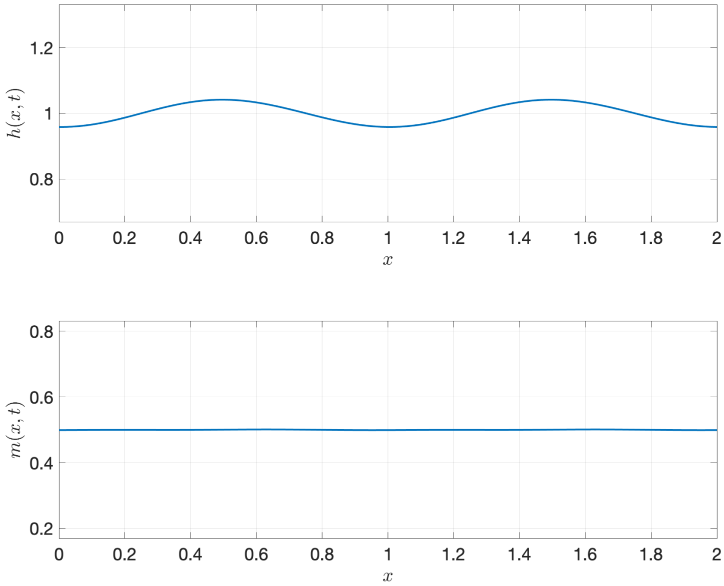

Figure 4: Numerical solution for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom) for the second set of initial conditions shown in Eq. 12.

solution computed on a distinctly finer mesh. Similarly to the previous case, once the error is computed, it is stored in the vectors `err_h_vec1`, `err_m_vec1` for the first set of initial conditions, and `err_h_vec2`, `err_m_vec2` for the second one. The log-log plot of such a result is shown in Fig. 5 and Fig. 6, for the first and second set of initial conditions, respectively.

In both cases, it can be noted that the order of convergence of the numerical solution is first order in space.

4 Part III

In this last section, a more complex problem is studied. Keeping the vanishing source term as in the previous case, the following discontinuous initial conditions are considered:

$$h(x,0) = h_0(x) = 1, \quad m(x,0) = m_0(x) = \begin{cases} -0.5 & x < 1 \\ 0 & x > 1 \end{cases} \tag{13}$$

with open boundary conditions.

4.1 Solution with Lax-Friedrichs flux

Initially, the Lax-Friedrichs flux is employed to solve the above problem. The plot in Fig. 7 shows the comparison between numerical solutions obtained with the Lax-Friedrichs scheme. The finest one is used as the reference solution (as in the previous section) and it can be noticed that decreasing the mesh size allows for a better approximation of the discontinuities in the solution. Indeed, when the mesh is not sufficiently fine, the discontinuities get smeared out. Even the finest solution obtained with the Lax-Friedrichs flux seems to still heavily smear out discontinuities. This is a limiting factor of such a numerical scheme [1]. In the true solution, shock waves arise and propagate in the domain

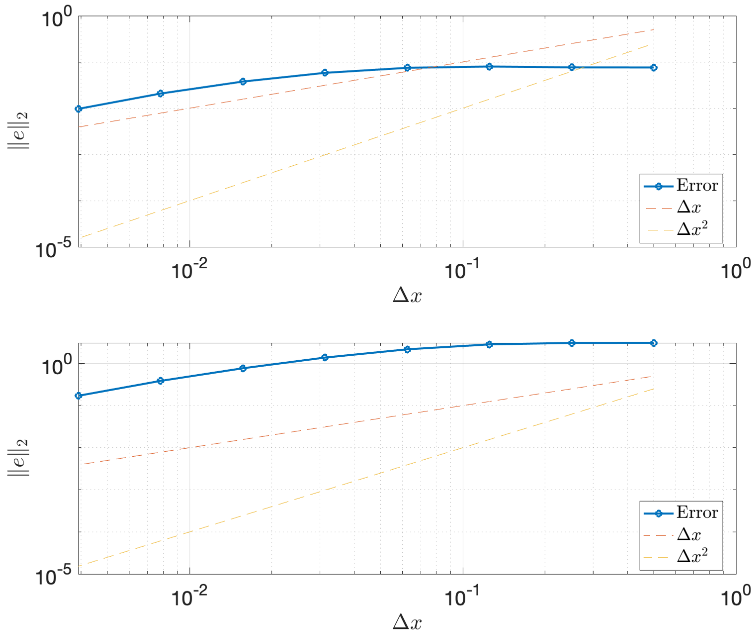

Figure 5: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom), for the first set of initial conditions, shown in Eq. 11. Order 1 and 2 convergence rates are also plotted, to ease the comparison.

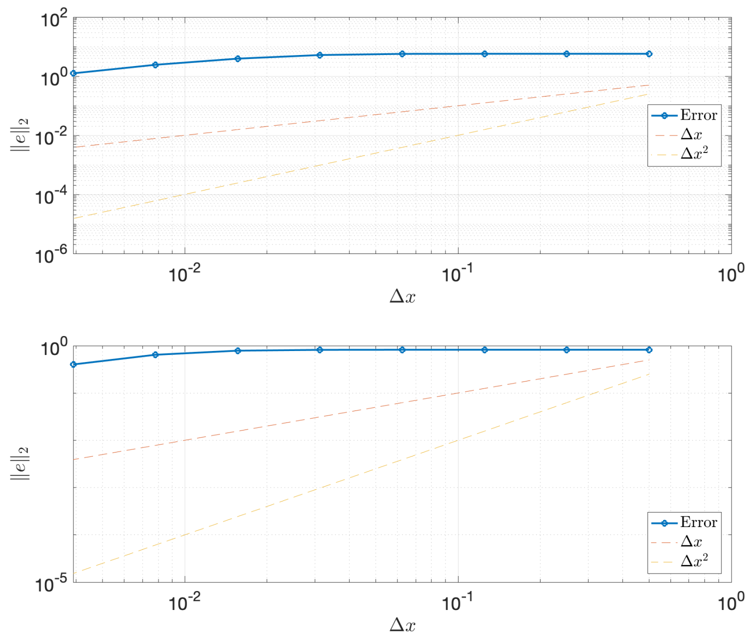

Figure 6: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom), for the second set of initial conditions, shown in Eq. 12. Order 1 and 2 convergence rates are also plotted, to ease the comparison.

due to the discontinuous initial condition. Since the Lax-Friedrichs flux is conservative and Lipschitz and the solution has uniformly bounded total variation, the hypotheses of the Lax-Wendroff theorem are satisfied: since convergence is achieved, the numerical solution is a weak solution, and the speed of the shocks is correctly predicted according to the Rankine-Hugoniot condition ([1]). Actually, not only (in case of convergence) are we guaranteed to obtain a weak solution, but also, since the Lax-Friedrichs scheme is monotone (as mentioned earlier), we converge toward an entropy solution.

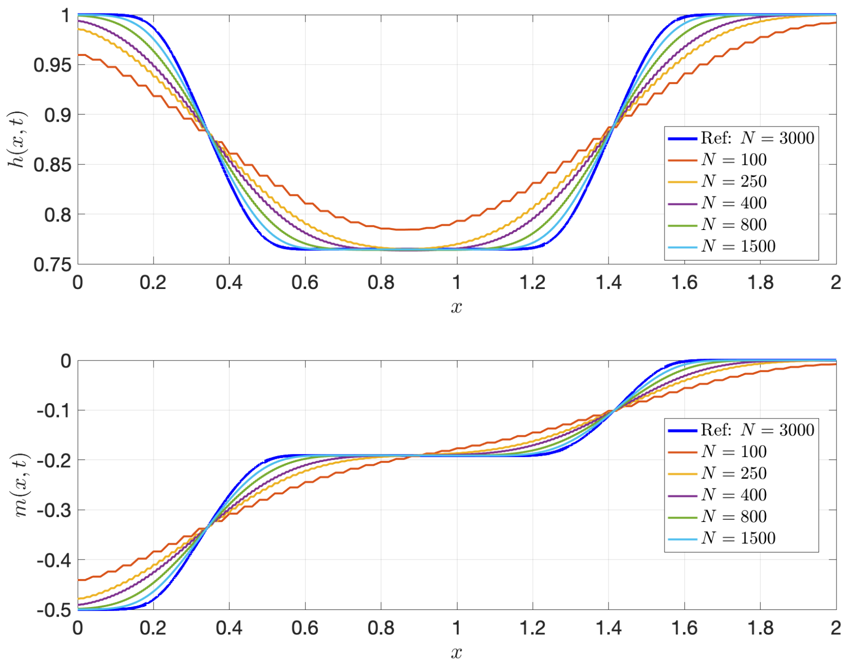

Figure 7: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with Lax-Friedrichs scheme and varying mesh sizes: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). The solution obtained with the finest mesh resolution is used as the reference, and it is plotted in blue.

4.2 Solution with Lax-Wendroff scheme

The same approach as in the previous case is followed, considering now the Lax-Wendroff numerical flux:

$$\mathbf{F}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{f}(\mathbf{u}) + \mathbf{f}(\mathbf{v})}{2} - \frac{k}{\Delta x}\mathbf{f}'\left(\frac{\mathbf{u}+\mathbf{v}}{2}\right)\frac{(\mathbf{v}-\mathbf{u})}{2} \tag{14}$$

This scheme requires the computation of the Jacobian matrix. By specifying this new numerical flux when using the function `conservative_scheme.m`, the Lax-Wendroff scheme is readily implemented. The plot in Fig. 8 shows the comparison between numerical solutions obtained with the Lax-Wendroff scheme just discussed. Again, the finest one is used as the reference solution, and it is possible to note that decreasing the mesh size allows for a better approximation of the discontinuities in the solution in this case as well.

By looking at the plots, it is clear that the Lax-Wendroff scheme suffers from numerical oscillations compared to the Lax-Friedrichs scheme: such oscillations decrease with decreasing mesh size but they never disappear. Instead, they become denser near the discontinuities. Nevertheless, the hypotheses for the Lax-Wendroff theorem are satisfied once again: the scheme converges to a weak solution, thus satisfying the Rankine-Hugoniot condition for the speed of the shocks. However, since the Lax-Wendroff flux is in general not monotone (e.g. the maximum principle is violated, see [1]), we are not guaranteed

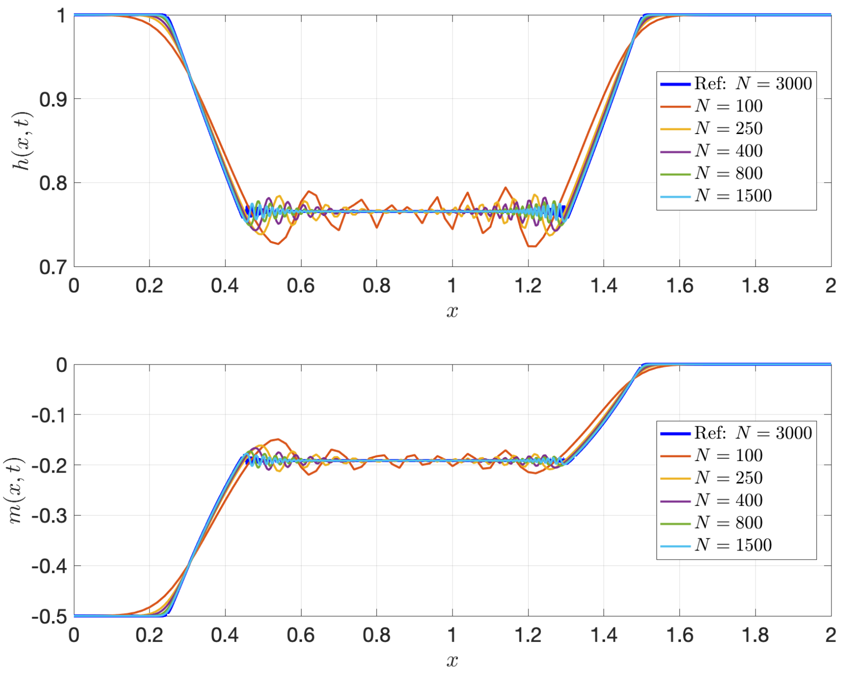

Figure 8: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with Lax-Wendroff scheme and varying mesh sizes: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). The solution obtained with the finest mesh resolution is used as the reference, and it is plotted in blue.

to converge towards the entropy solution. Finally, we now compare the two solutions obtained with the two numerical schemes being analyzed with the smallest mesh sizes, as shown in Fig. 9.

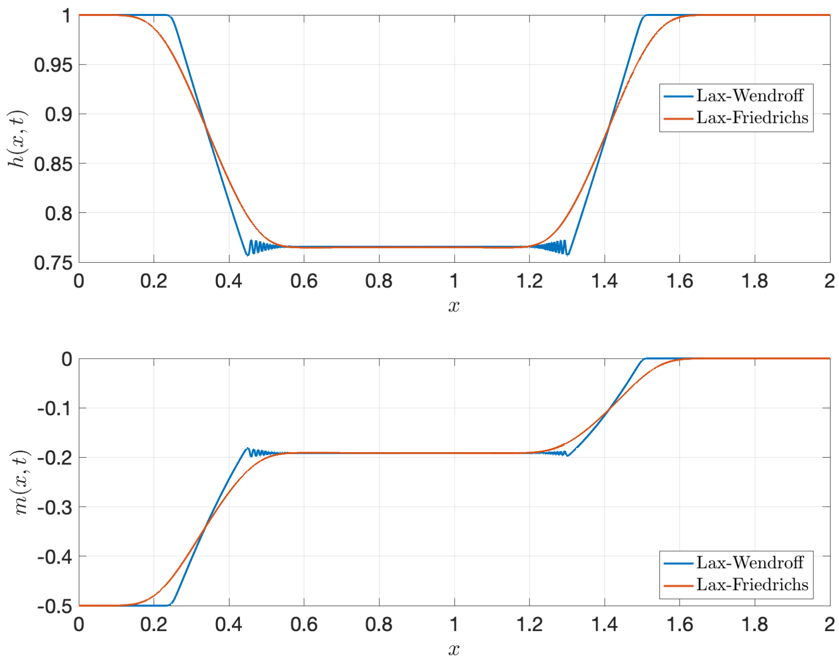

Figure 9: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with the two numerical schemes analyzed in this project for the smallest mesh size considered at final time $t = 0.5$: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom).

It is evident that the Lax-Wendroff scheme better captures the discontinuity, with a substantially smaller smearing out effect compared to the Lax-Friedrichs scheme[2], despite suffering from the aforementioned numerical oscillation near the discontinuities. This behavior is well known in the literature: it can be shown that the Lax-Wendroff scheme is less dissipative than Lax-Friedrichs, as also discussed in [1].

References

[1] Jan S. Hesthaven. *Numerical Methods for Conservation Laws.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2018. doi: 10.1137/1.9781611975109. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611975109.

[2] Bear in mind that the smearing effects are still present and cannot be completely avoided when solving the equation numerically.

A MATLAB code

A.1 Part1_1

```
clear
close all
clc

%%% Code by Francesco Sala and Nicolo' Viscusi %%%

% Set to true if you want to see the animation of the solutions over
    time
animation = "True";

%% Resolution of the problem

% Definition of parameters
g = 1;
u = 0.25;

% Spatial domain
xspan = [0, 2];

% Temporal domain
tspan = [0, 0.5];

% Initial conditions
h0 = @(x) 1 + 0.5 * sin(pi * x);
m0 = @(x) u * h0(x);

% Number of grid points
N = 1000;

% Number of time steps
CFL = 0.5;

% Note that max(h0) = 1.5
k = CFL * (xspan(2) - xspan(1)) / N * 1 / (u + sqrt(g * 1.5));
K = round((tspan(end) - tspan(1)) / k);

% Source function
S = @(x, t) [pi/2 * (u - 1) * cos(pi * (x - t));
    pi/2 * cos(pi * (x - t)) * (- u + u^2 + g * h0(x - t))];

% Here we use periodic boundary condition as the option ('peri')
bc = 'peri';

% Solve the problem
```

```matlab
[h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan, tspan, N, ...
    K, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);


% We visualize the solution
if animation == "True"
    figure(1)
    for i = 1 : 20 : length(tvec)

        subplot(2, 1, 1)
        plot(xvec, h(:, i), 'LineWidth', 2)
        hold on
        plot(xvec, h0(xvec - tvec(i)), 'Linewidth', 2)
        title(['$h(x, t)$ at $t = $ ', num2str(tvec(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$h(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        ylim([0.4 1.6]);
        hold off
        legend('Numerical solution', 'Exact solution', ...
            'Interpreter', 'latex')
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xvec, m(:, i), 'LineWidth', 2)
        hold on
        plot(xvec, u * h0(xvec - tvec(i)), 'Linewidth', 2)
        title(['$m(x, t)$ at $t = $ ', num2str(tvec(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$m(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        ylim([0.1 0.4]);
        hold off
        legend('Numerical solution', 'Exact solution', ...
            'Interpreter', 'latex')
        set(gca, 'Fontsize', 20)
        drawnow

    end
end

%%  Error analysis

```

```
% We solve the same problem for different values of \Delta x
delta_x_vec =  2.^-(1:7);

% Note that we cannot solve for smalle values of delta_x, because we would
% need a too large matrix to store the solutions h and m
N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
err_h_vec = zeros(size(N_vec));
err_m_vec = zeros(size(N_vec));

for i=1:length(N_vec)
    N = N_vec(i);
    k = CFL * (xspan(2) - xspan(1)) / N * 1 / (u + sqrt(g * 1.5));
    K = round((tspan(end) - tspan(1)) / k);
    T_f = 0.5;
    [h, m, ~, xvec, k, delta_x] = conservative_scheme(xspan, tspan, N, ...
        K, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
    err_h_vec(i) = 1/sqrt(N)*norm(h(:, end) -h0(xvec-T_f)');
    err_m_vec(i) = 1/sqrt(N)*norm(m(:, end) - u*h0(xvec-T_f)');
end


% Plot the error
figure(2)

subplot(2,1,1)
loglog(delta_x_vec, err_h_vec , "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2, "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
% title("Error on \(h(x,t)\) at \(t=2\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
    "latex",  "location", "best")
set(gca, 'Fontsize', 20)
grid on


subplot(2,1,2)
loglog(delta_x_vec, err_m_vec, "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2, "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
% title("Error on \(m(x,t)\) at \(t=2\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
```

```
    "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)
```

A.2 Part1_2

```
clear
close all
clc

%%% Code by Francesco Sala and Nicolo' Viscusi %%%

% Set to true if you want to see the animation of the solutions over
    time
animation = "True";

%% First set of initial conditions

% Spatial domain
xspan = [0 2];

% Temporal domain
tspan = [0 0.5];

% Initial conditions
h01 = @(x) 1 - 0.1 * sin(pi * x);
m01 = @(x) 0;

% Source function
S1 = @(x, t) [0;
    0];

% We will use periodic boundary condition option ('peri')
bc = 'peri';


% We generate a reference solution
[h1_ex, m1_ex, tvec1_ex, xvec1_ex] = conservative_scheme(xspan, tspan,
    ...
    2000, 4000, h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);


% We now proceed with a less refined solution
N = 250;

% Number of time steps
CFL = 0.5;
K = N / CFL;

```

```
% Solve the problem
[h1, m1, tvec1, xvec1] = conservative_scheme(xspan, tspan, N, K, ...
    h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);


% We visualize the solution
if animation == "True"
    figure(1)
    for i = 1 : 20 : length(tvec1)

        subplot(2, 1, 1)
        plot(xvec1, h1(:, i), 'LineWidth', 2)
        title(['$h(x, t)$ at $t = $', num2str(tvec1(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$h(x, t)$', 'Interpreter', 'latex')
        grid on
        axis equal
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xvec1, m1(:, i), 'LineWidth', 2)
        title(['$m(x, t)$ at $t = $', num2str(tvec1(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$m(x, t)$', 'Interpreter', 'latex')
        grid on
        axis equal
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

    end
end



%% Error analysis, initial condition 1

% We solve the same problem for different values of \Delta x
delta_x_vec =  2.^-(1:8);

% Note that we cannot solve for small values of delta_x, because we
    would
% need a too large matrix to store the solutions h and m
N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
err_h_vec1 = zeros(size(N_vec));
```

```
err_m_vec1 = zeros(size(N_vec));

for i=1:length(N_vec)
    N = N_vec(i);

    K = N / CFL;

    [h1, m1, ~, xvec1_err] = conservative_scheme(xspan, tspan, N, K,
        ...
        h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);

    % We now want to compare h1(:, end) with h1_ex(:, end),
    % but this second vector is defined on a different grid xvec1_ex
    % We interpolate h1_ex(:, end) on the grid xvec1
    h1_interp = interp1(xvec1_err, h1(:, end), xvec1_ex);
    m1_interp = interp1(xvec1_err, m1(:, end), xvec1_ex);
    err_h_vec1(i) = norm(h1_interp' -h1_ex(:, end));
    err_m_vec1(i) = norm(m1_interp' - m1_ex(:, end));

end


% Plot error
figure(2)

subplot(2,1,1)
loglog(delta_x_vec, err_h_vec1, "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(h(x,t)\) at \(t=0.5\) (case 1)", "Interpreter","latex
    ")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
    "latex", "location", "best")
set(gca, 'Fontsize', 20)
grid on

subplot(2,1,2)
loglog(delta_x_vec, err_m_vec1, "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(m(x,t)\) at \(t=0.5\) (case 1)", "Interpreter","latex
    ")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
```

```matlab
    "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)



%% Second set of initial conditions

% Initial conditions
h02 = @(x) 1 - 0.2 * sin(2 * pi * x);
m02 = @(x) 0.5;

% Source term
S2 = @(x, t) [0;
    0];

% All the other parameters remain the same...

% First, a refined solution as reference "exact"
[h2_ex, m2_ex, tvec2_ex, xvec2_ex] = conservative_scheme(xspan, ...
    tspan, 2000, 4000, h02, m02, @lax_friedrichs_flux, @flux_phys, S2, bc);

% Solve the problem on a less refined mesh
N = 100;
K = 200;
[h2, m2, tvec2, xvec2] = conservative_scheme(xspan, tspan, N, K, ...
    h02, m02, @lax_friedrichs_flux, @flux_phys, S2, bc);



% We visualize the solution
if animation == "True"
    figure(3)
    for i = 1 : 20 : length(tvec2)

        subplot(2, 1, 1)
        plot(xvec2, h2(:, i), 'LineWidth', 2)
        title(['$h(x, t)$ at $t = $', num2str(tvec2(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$h(x, t)$', 'Interpreter', 'latex')
        grid on
        axis equal
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
```

```
        plot(xvec2, m2(:, i), 'LineWidth', 2)
        title(['$m(x, t)$ at $t = $', num2str(tvec2(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$m(x, t)$', 'Interpreter', 'latex')
        grid on
        axis equal
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

    end
end



%% Error analysis, initial condition 2

% We solve the same problem for different values of \Delta x
delta_x_vec =  2.^-(1:8);

% Note that we cannot solve for small values of delta_x, because we
    would
% need a too large matrix to store the solutions h and m
N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
err_h_vec2 = zeros(size(N_vec));
err_m_vec2 = zeros(size(N_vec));

for i=1:length(N_vec)
    N = N_vec(i);

    K = N / CFL;

    [h2, m2, ~, xvec2_err] = conservative_scheme(xspan, tspan, N, K,
        ...
        h02, m02, @lax_friedrichs_flux, @flux_phys, S2, bc);

    % We now want to compare h1(:, end) with h1_ex(:, end), but this
    % second vector is defined on a different grid xvec1_ex
    % We interpolate h1_ex(:, end) on the grid xvec1
    h2_interp = interp1(xvec2_err, h2(:, end), xvec2_ex);
    m2_interp = interp1(xvec2_err, m2(:, end), xvec2_ex);

    err_h_vec2(i) = norm(h2_interp' - h2_ex(:, end));
    err_m_vec2(i) = norm(m2_interp' - m2_ex(:, end));

end

```

```
% Plot the error
figure(4)

subplot(2,1,1)
loglog(delta_x_vec, err_h_vec2 , "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(h(x,t)\) at \(t=0.5\) (case 2)", "Interpreter","latex
    ")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
     "latex",  "location", "best")
set(gca, 'Fontsize', 20)
grid on

subplot(2,1,2)
loglog(delta_x_vec, err_m_vec2, "o-", "Linewidth", 2)
hold on
loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(m(x,t)\) at \(t=0.5\) (case 2)", "Interpreter","latex
    ")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
     "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)
```

A.3 Part1_3

```
clear
close all
clc

%%% Code by Francesco Sala and Nicolo' Viscusi %%%

% Set to true if you want to see the animation of the solutions over
    time
animation = "True";

%% Definition of parameters

% Boundary conditions option
bc = 'open';

% Initial conditions
```

```matlab
h0 = @(x) 1;
m0 = @(x) -0.5 * (x < 1);

% Source term
S = @(x, t) [0;
             0];

% Spatial domain
xspan = [0, 2];

% Temporal domain
tspan = [0 0.5];

% Number of points in space and time
N = 100;
K = 200;



%% Solve the problem with Lax-Friedrichs flux
[h, m, tvec, xvec] = conservative_scheme(xspan, tspan, N, K, h0,...
    m0, @lax_friedrichs_flux, @flux_phys, S, bc);

% Reference solution with very fine mesh
[h_exf, m_exf, tvec_exf, xvec_exf] = conservative_scheme(xspan, ...
    tspan, 3000, 6000, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc)
    ;

% Animation of the solution
% Visualize the solution
if animation == "True"
    figure(1);
    for i = 1 : 20 : length(tvec)

        subplot(2, 1, 1)
        plot(xvec, h(:, i), 'LineWidth', 2)
        title(['$h(x, t)$ at $t = $', num2str(tvec(i))], ...
            'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$h(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xvec, m(:, i), 'LineWidth', 2)
        title(['$m(x, t)$ at $t = $', num2str(tvec(i))],  ...
            'Interpreter', 'latex')
```

```
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$m(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

    end
end


% Compute a set of numerical solutions obtained with gradually
% decreasing mesh sizes
[h1, m1, ~, xvec1] = conservative_scheme(xspan, tspan, 100, 200,...
    h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
[h2, m2, ~, xvec2] = conservative_scheme(xspan, tspan, 250, 500,...
    h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
[h3, m3, ~, xvec3] = conservative_scheme(xspan, tspan, 400, 800, ...
    h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
[h4, m4, ~, xvec4] = conservative_scheme(xspan, tspan, 800, 1600,...
    h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
[h5, m5, ~, xvec5] = conservative_scheme(xspan, tspan, 1500, 3000, ...
    h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);

figure(2)
subplot(2, 1, 1)
plot(xvec_exf, h_exf(:, end), '-b', 'LineWidth', 3)
hold on
plot(xvec1, h1(:, end), 'LineWidth', 2)
plot(xvec2, h2(:, end), 'LineWidth', 2)
plot(xvec3, h3(:, end), 'LineWidth', 2)
plot(xvec4, h4(:, end), 'LineWidth', 2)
plot(xvec5, h5(:, end), 'LineWidth', 2)
legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$',...
    '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
        best')
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$h(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)


subplot(2, 1, 2)
plot(xvec_exf, m_exf(:, end), '-b', 'LineWidth', 3)
hold on
plot(xvec1, m1(:, end), 'LineWidth', 2)
plot(xvec2, m2(:, end), 'LineWidth', 2)
plot(xvec3, m3(:, end), 'LineWidth', 2)
```

```
plot(xvec4, m4(:, end), 'LineWidth', 2)
plot(xvec5, m5(:, end), 'LineWidth', 2)
legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$',...
    '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
       best')
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$m(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)



%% Solve the problem with Lax-Wendroff flux
[h_lw, m_lw, tvec_lw, xvec_lw] = conservative_scheme(xspan, tspan, N,
   ...
    K, h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);

% Reference solution with very fine mesh
[h_exw, m_exw, tvec_exw, xvec_exw] = conservative_scheme(xspan, tspan,
   ...
    3000, 6000, h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);

% Animation of the solution
% Visualize the solution
if animation == "True"
    figure(3);
    for i = 1 : 20 : length(tvec_lw)

        subplot(2, 1, 1)
        plot(xvec_lw, h_lw(:, i), 'LineWidth', 2)
        title(['Lax-Wendroff: $h(x, t)$ at $t = $', ...
            num2str(tvec_lw(i))], 'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$h(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xvec_lw, m_lw(:, i), 'LineWidth', 2)
        title(['Lax-Wendroff: $m(x, t)$ at $t = $', ...
            num2str(tvec_lw(i))], 'Interpreter', 'latex')
        xlabel('$x$', 'Interpreter', 'latex')
        ylabel('$m(x, t)$', 'Interpreter', 'latex')
        grid on
        xlim([0 2]);
        set(gca, 'Fontsize', 20)
```

```
        drawnow

    end
end


% We now compute a set of numerical solutions obtained with gradually
% decreasing mesh sizes
[h1, m1, ~, xvec1] = conservative_scheme(xspan, tspan, 100, 200, ...
    h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
[h2, m2, ~, xvec2] = conservative_scheme(xspan, tspan, 250, 500, ...
    h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
[h3, m3, ~, xvec3] = conservative_scheme(xspan, tspan, 400, 800, ...
    h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
[h4, m4, ~, xvec4] = conservative_scheme(xspan, tspan, 800, 1600, ...
    h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
[h5, m5, ~, xvec5] = conservative_scheme(xspan, tspan, 1500, 3000, ...
    h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);

figure(4)
subplot(2, 1, 1)
plot(xvec_exw, h_exw(:, end), '-b', 'LineWidth', 3)
hold on
plot(xvec1, h1(:, end), 'LineWidth', 2)
plot(xvec2, h2(:, end), 'LineWidth', 2)
plot(xvec3, h3(:, end), 'LineWidth', 2)
plot(xvec4, h4(:, end), 'LineWidth', 2)
plot(xvec5, h5(:, end), 'LineWidth', 2)
legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$', ...
    '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', 'best')
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$h(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)


subplot(2, 1, 2)
plot(xvec_exw, m_exw(:, end), '-b', 'LineWidth', 3)
hold on
plot(xvec1, m1(:, end), 'LineWidth', 2)
plot(xvec2, m2(:, end), 'LineWidth', 2)
plot(xvec3, m3(:, end), 'LineWidth', 2)
plot(xvec4, m4(:, end), 'LineWidth', 2)
plot(xvec5, m5(:, end), 'LineWidth', 2)
legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$',...
    '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', 'best')
```

```
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$m(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)


%% Comparison between finest solutions of Lax-Friedrichs and Lax-
    Wendroff
figure(5)
subplot(2, 1, 1)
plot(xvec_exw, h_exw(:, end), 'LineWidth', 2)
hold on
plot(xvec_exf, h_exf(:, end), 'LineWidth', 2)
legend('Lax-Wendroff', 'Lax-Friedrichs', 'Interpreter', 'latex', ...
    'Location', 'best')
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$h(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)


subplot(2, 1, 2)
plot(xvec_exw, m_exw(:, end), 'LineWidth', 2)
hold on
plot(xvec_exf, m_exf(:, end), 'LineWidth', 2)
legend('Lax-Wendroff', 'Lax-Friedrichs', 'Interpreter', 'latex', ...
    'Location', 'best')
xlabel('$x$', 'Interpreter', 'latex')
ylabel('$m(x, t)$', 'Interpreter', 'latex')
grid on
xlim([0 2]);
set(gca, 'Fontsize', 20)
```

A.4 Conservative scheme implementation

```
function [h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan,
    ...
    tspan, N, K, h0, m0, numerical_flux, flux_phys, S, bc)


% CONSERVATIVE_SCHEME - Implements the conservative scheme
%                       for solving hyperbolic PDEs.
%
%   [h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan, ...
%           tspan, N, K, h0, m0, numerical_flux, flux_phys, S, bc)
%
% INPUTS:
```

```
%   xspan            - Spatial domain [x_start, x_end].
%   tspan            - Temporal domain [t_start, t_end].
%   N                - Number of spatial grid points.
%   K                - Number of temporal grid points.
%   h0               - Function handle for initial water depth.
%   m0               - Function handle for initial discharge.
%   numerical_flux - Function handle for numerical flux computation.
%   flux_phys        - Function handle for the physical flux function.
%   S                - Source term function.
%   bc               - Boundary condition: 'peri' (periodic), 'open' (
    open).
%
% OUTPUTS:
%   h                - Water depth (matrix) over space and time.
%   m                - Discharge (matrix) over space and time.
%   tvec             - Temporal grid vector.
%   xvec             - Spatial grid vector.
%   k                - Time step size.
%   delta_x          - Spatial grid spacing.
%
% DESCRIPTION:
%   This function implements the conservative scheme for solving
    hyperbolic
%   partial differential equations (PDEs) that model shallow water flow
    .
%   It evolves the water depth (h) and discharge (m) over a specified
%   spatial and temporal domain using the conservative scheme formula:
%
%   u_j^n+1 = u_j^n - k/h * (F_j+1/2^n - F_j-1/2^n) + k * S_j^n,
%
%   where u represents [h; m], F is the flux, S is the source term, k
    is
%   the time step, and delta_x is the spatial grid spacing. The
    boundary
%   conditions (bc) can be set to 'peri' (periodic), 'open' (open).
%
% Authors: Francesco Sala and Nicolo' Viscusi
% December 2023


tvec    = linspace(tspan(1), tspan(end), K + 1);
xvec    = linspace(xspan(1), xspan(end), N + 1);
h       = zeros(N + 1, K + 1);
m       = zeros(N + 1, K + 1);

delta_x = (xspan(2) - xspan(1)) / N;
k       = (tspan(2) - tspan(1)) / K;

% Set the initial conditions
```

```matlab
h(:, 1) = h0(xvec);
m(:, 1) = m0(xvec);


for i = 2 : length(tvec)

    j = 1;
    source = S(xvec(j), tvec(i));

    if bc == 'peri'

        rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
            [h(j + 1, i - 1); m(j + 1, i - 1)], delta_x, k) - ...
            numerical_flux(flux_phys, [h(end - 1, i - 1); ...
            m(end - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
        h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
        m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);

    elseif bc == 'open'

        rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
            [h(j + 1, i - 1); m(j + 1,i - 1)], delta_x, k) - ...
            numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
            [h(j, i - 1); m(j, i - 1)], delta_x, k);
        h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
        m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);

    end

    for j = 2 : (length(xvec) - 1)

        source = S(xvec(j), tvec(i));
        rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
            [h(j + 1, i - 1); m(j + 1, i - 1)], delta_x, k) - ...
            numerical_flux(flux_phys, [h(j - 1, i - 1);  ...
            m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
        h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
        m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);

    end

    j = length(xvec);
    source = S(xvec(j), tvec(i));

    if bc == 'peri'

        rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
            [h(2, i - 1); m(2, i - 1)], delta_x, k) - ...
```

```
                numerical_flux(flux_phys, [h(j - 1, i - 1); ...
                m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
            h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
            m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);

        elseif bc == 'open'

            rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
                [h(j, i - 1); m(j, i - 1)], delta_x, k) - ...
                numerical_flux(flux_phys, [h(j - 1, i - 1);  ...
                m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
            h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
            m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);

        end

end

end
```

A.5 Physical flux

```
function f = flux_phys(q)

% FLUX_PHYS - Computes the physical flux function for the shallow
%             water equations.
%
%   f = flux_phys(q)
%
% INPUTS:
%   q    - Vector of state variables [h, m], where
%            h: Water depth
%            m: Discharge
%
% OUTPUT:
%   f    - Vector representing the physical flux corresponding to the
  input
%          state q.
%
% DESCRIPTION:
%   This function calculates the physical flux for the shallow water
%   equations.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% December 2023

g = 1;
h = q(1);
m = q(2);
```

```matlab

f = [m;
     m.^2./h + 1/2*g*h.^2];

return
```

A.6 Numerical flux

A.6.1 Lax-Friedrichs

```matlab
function F = lax_friedrichs_flux(flux_phys, u, v, delta_x, k)
% LAX_FRIEDRICHS_FLUX - Computes the Lax-Friedrichs flux for a given
%                        physical flux function.
%
%   F = lax_friedrichs_flux(flux_phys, u, v, delta_x, k)
%
% INPUTS:
%   flux_phys   - Function handle for the physical flux function.
%   u           - State vector at position j (qj).
%   v           - State vector at position j+1 (qj+1).
%   delta_x     - Spatial grid spacing.
%   k           - Time step size.
%
% OUTPUT:
%   F           - Lax-Friedrichs flux between u and v.
%
% DESCRIPTION:
%   This function calculates the Lax-Friedrichs flux between two neighboring
%   states, u and v, based on the given physical flux function (flux_phys).
%   The Lax-Friedrichs scheme is a numerical method often used for solving
%   hyperbolic partial differential equations.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% December 2023

% Lax-Friedrichs flux: u and v have size (2,1), and correspond to qj and qj+1
F = 0.5 * (flux_phys(u) + flux_phys(v) - delta_x / k * (v - u));

return
```

A.6.2 Lax-Wendroff

```matlab
function F = lax_wendroff_flux(flux_phys, u, v, delta_x, k)
% LAX_WENDROFF_FLUX - Computes the Lax-Wendroff flux for a given
%                      physical flux function.
```

```
%
%   F = lax_wendroff_flux(flux_phys, u, v, delta_x, k)
%
% INPUTS:
%   flux_phys    - Function handle for the physical flux function.
%   u            - State vector at position j (qj).
%   v            - State vector at position j+1 (qj+1).
%   delta_x      - Spatial grid spacing.
%   k            - Time step size.
%
% OUTPUT:
%   F            - Lax-Wendroff flux between u and v.
%
% DESCRIPTION:
%   This function calculates the Lax-Wendroff flux between two
    neighboring
%   states, u and v, based on the given physical flux function (
    flux_phys).
%   The Lax-Wendroff scheme is a numerical method commonly used for
    solving
%   hyperbolic partial differential equations.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% December 2023

F = 0.5 * (flux_phys(u) + flux_phys(v) - k / delta_x * ...
    flux_phys_prime((u + v) / 2) * (flux_phys(v) - flux_phys(u)));

    function f_prime = flux_phys_prime(q)
        % Compute the Jacobian of the flux
        g = 1;
        h = q(1);
        m = q(2);
        f_prime = [0, 1;
            -m.^2./(h.^2) + g*h, 2 * m./h  ];
    end


end
```