



Project I report

Finite difference methods for the shallow water equations

Authors: Nicolò Giosuè Carlo Viscusi
Francesco Sala

Date: December 13, 2023

Course: Numerical Methods for Conservation Laws - MATH-459

Professor: Martin Licht

Assistant: Fernando Henriquez

Contents

1	Introduction	2
2	Part I	2
2.1	About the implementation	2
2.2	A first resolution	3
2.3	Error analysis	3
3	Part II	4
3.1	Numerical resolution	5
3.2	Error analysis	5
4	Part III	7
4.1	Solution with Lax-Friedrichs flux	7
4.2	Solution with Lax-Wendroff scheme	8
A	MATLAB code	11
A.1	Part1_1	11
A.2	Part1_2	14
A.3	Part1_3	19
A.4	Conservative scheme implementation	24
A.5	Physical flux	27
A.6	Numerical flux	28
A.6.1	Lax-Friedrichs	28
A.6.2	Lax-Wendroff	28

1 Introduction

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

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

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:

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

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 is implemented. To accomplish this result, the domain Ω is discretized by defining $N + 1$ nodes: let $\Delta x = \frac{2}{N}$ be the space between two contiguous nodes, then the nodes are $x_j = (j-1)\Delta x$, $j = 1, \dots, N+1$. Similarly, let the final time be $T = 0.5$; the temporal domain was discretized using $M + 1$ equally spaced nodes $t^n = (n-1)k$, $k = 1, \dots, M+1$ and with $k = \frac{T}{M}$. A conservative finite difference scheme can be written in the following form:

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

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:

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

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 are 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:

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

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

¹In 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:

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

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 $(N + 1) \times (K + 1)$ matrices `h` and `m` (containing the solution) are returned as outputs, as well as the discretized space and time vectors (`xvec` and `tvec`)².

2.2 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:

$$h(x, 0) = h_0(x) = 1 + 0.5 \sin(\pi x), \quad m(x, 0) = m_0(x) = u h_0(x) \quad (7)$$

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

$$\mathbf{S}(x, t) = \left[\begin{array}{c} \frac{\pi}{2}(u - 1) \cos(\pi(x - t)) \\ \frac{\pi}{2}(-u + u^2 + g h_0(x - t)) \cos(\pi(x - t)) \end{array} \right] \quad (8)$$

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

$$h(x, t) = h_0(x - t), \quad m(x, t) = u h(x, t) \quad (9)$$

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

$$k = \text{CFL} \frac{\Delta x}{\max_i(|u_i| + \sqrt{g h_i})} \quad (10)$$

with $\text{CFL} = 0.5$. 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 $\Delta x = 2^i$, $i = -4, \dots, -10$. 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`³. 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. For completeness, 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.

²The reader may refer to Appendix A.

³Once again, the reader may refer to Appendix A for more details about the implementation.

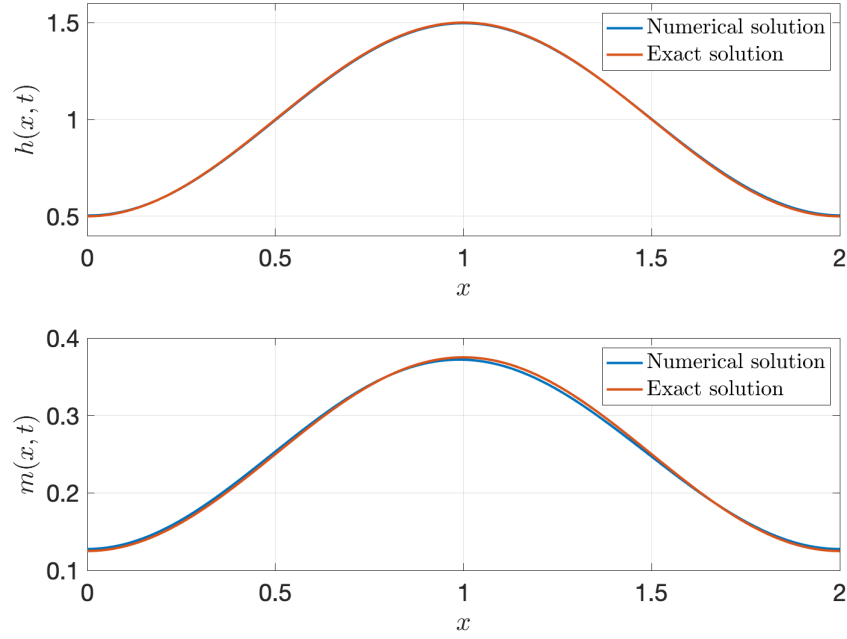


Figure 1: Comparison between exact and numerical solution for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom).

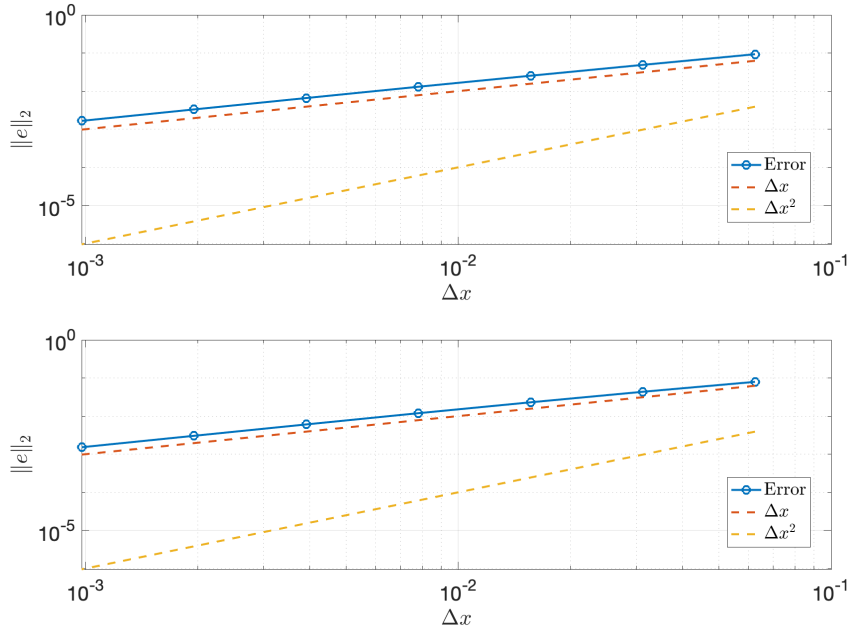


Figure 2: Log-log plot of the ℓ_2 norm of the error between exact and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). Order 1 and 2 convergence rates are also plotted, to allow for comparison.

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:

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

while the second set of initial conditions is:

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

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.

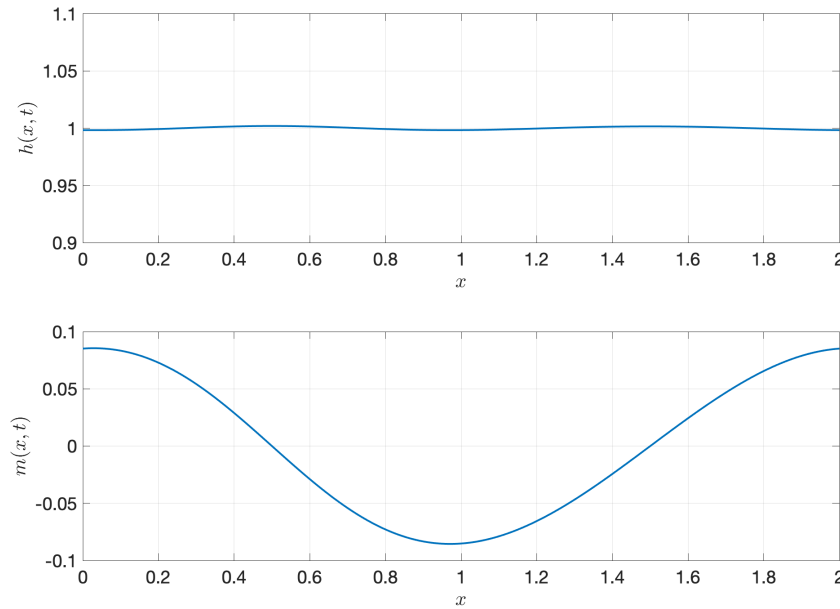


Figure 3: Numerical solution for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom) for the first set of initial conditions shown in Eq. 11.

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) 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 $\Delta x = 2^i$, $i = -6, \dots, -10$. 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 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.

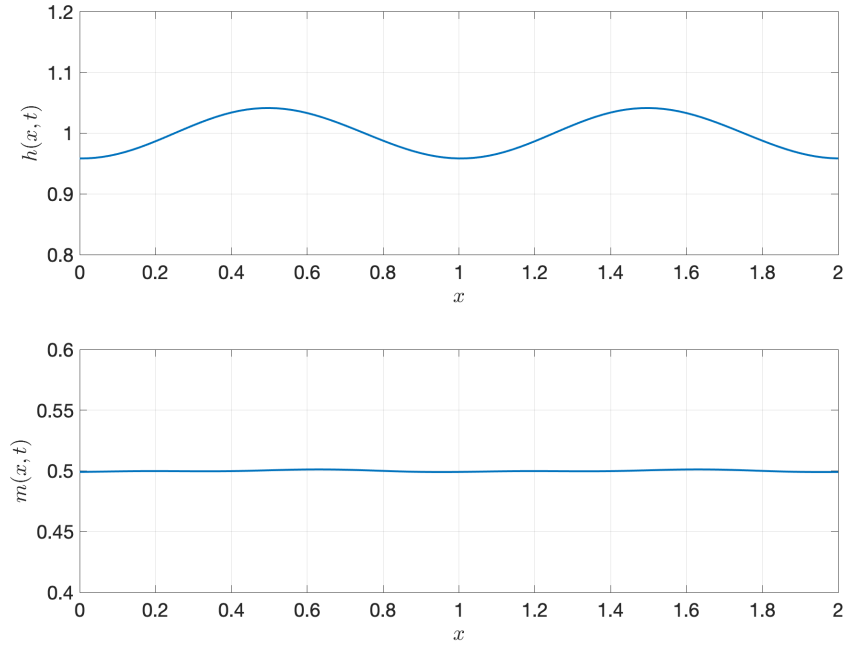


Figure 4: Numerical solution for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom) for the second set of initial conditions shown in Eq. 12.

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.

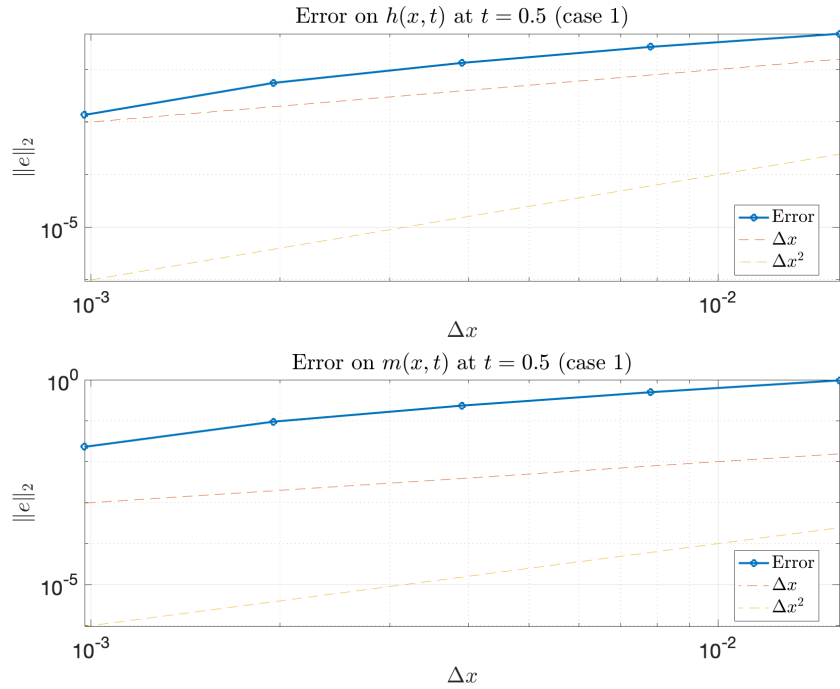


Figure 5: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom), for the first set of initial conditions, shown in Eq. 11. Order 1 and 2 convergence rates are also plotted, to ease the comparison.

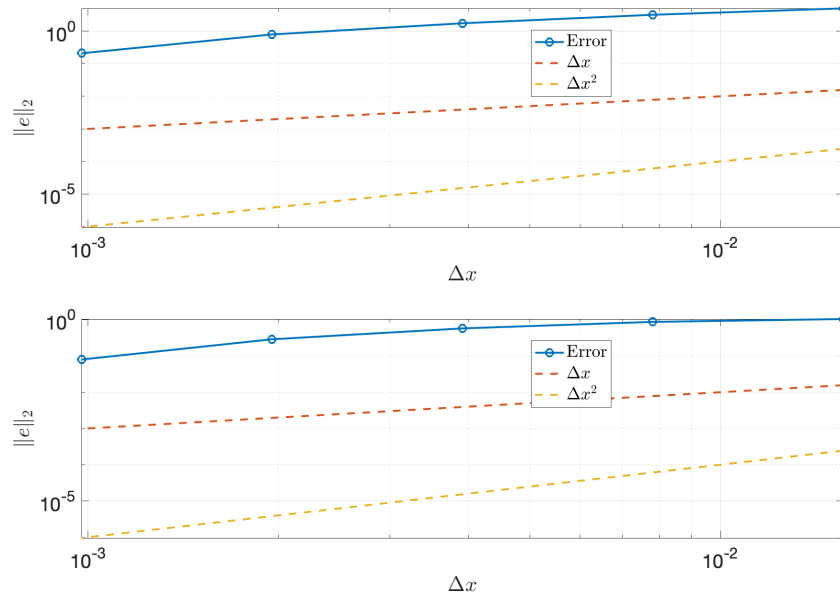


Figure 6: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 0.5$ for $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom), for the second set of initial conditions, shown in Eq. 12. Order 1 and 2 convergence rates are also plotted, to ease the comparison.

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:

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

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 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 are

we guaranteed to converge to a weak solution, but also, since the Lax-Friedrichs scheme is monotone⁴, we converge towards an entropy solution.

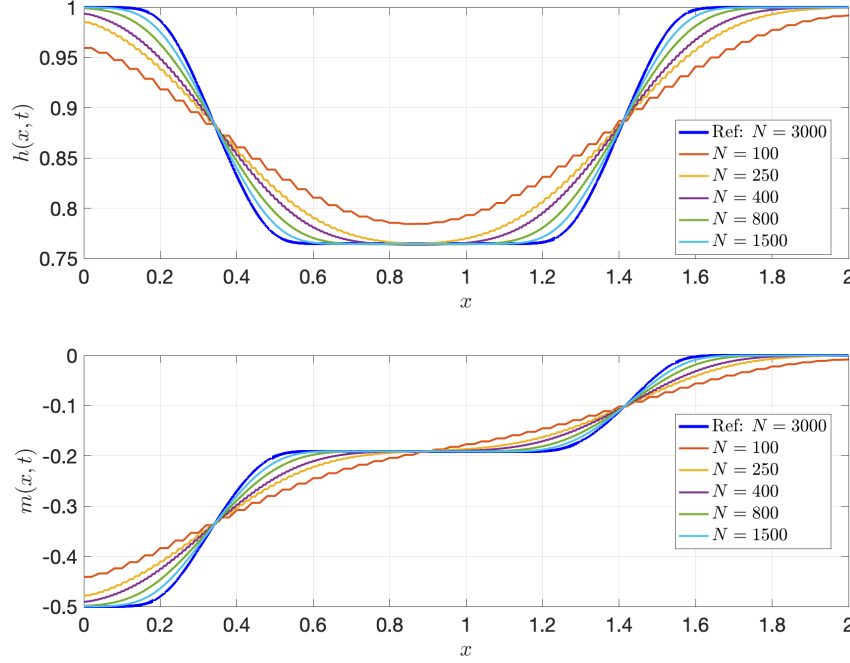


Figure 7: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with Lax-Friedrichs scheme and varying mesh sizes: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). The solution obtained with the finest mesh resolution is used as the reference, and it is plotted in blue.

4.2 Solution with Lax-Wendroff scheme

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

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

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: since the numerical scheme is converging, it 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 to converge towards the entropy solution. Finally, we compare the two solutions obtained with the two numerical schemes being analyzed with the smallest mesh sizes, as shown in Fig. 9.

⁴As mentioned in Sec. 3.1.

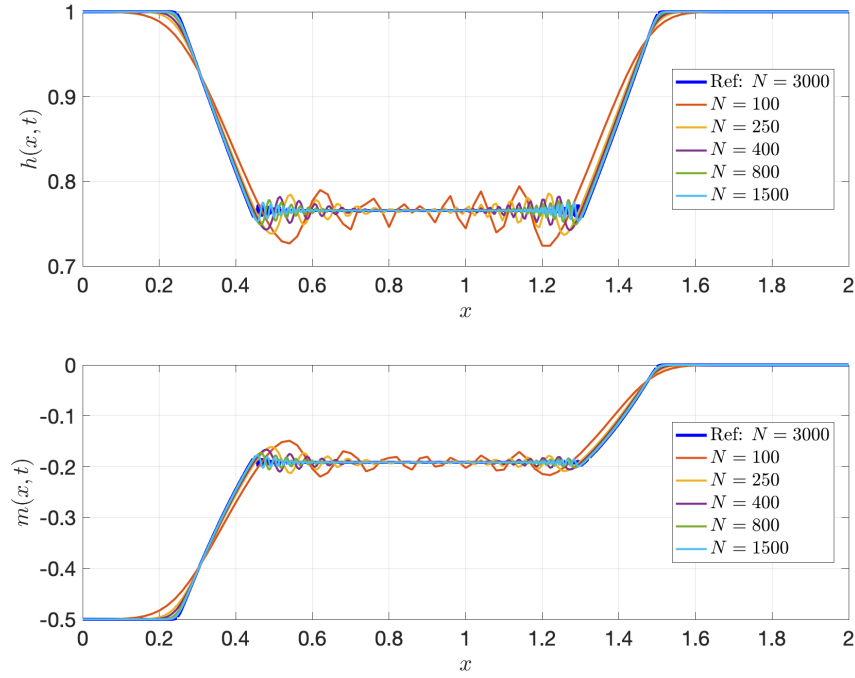


Figure 8: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with Lax-Wendroff scheme and varying mesh sizes: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom). The solution obtained with the finest mesh resolution is used as the reference, and it is plotted in blue.

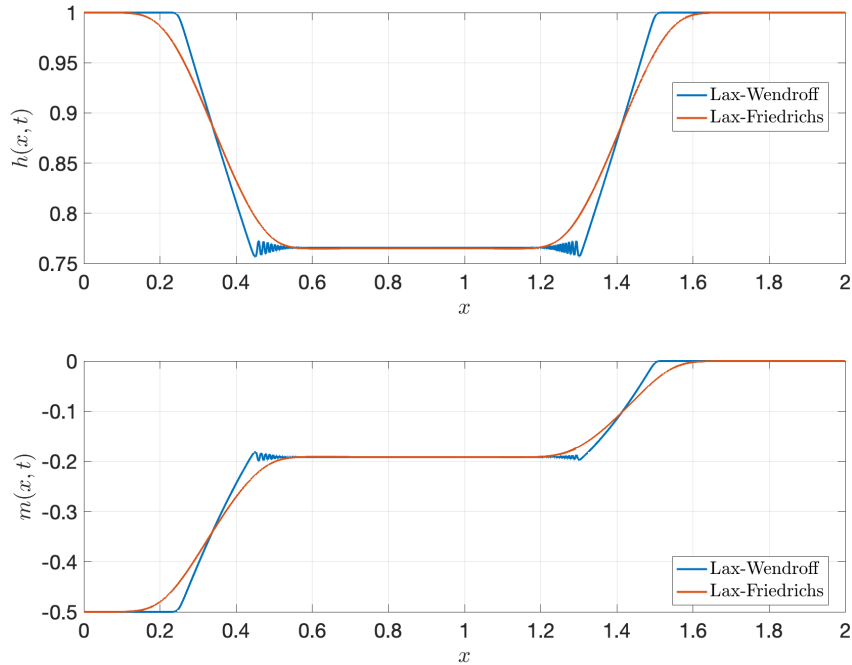


Figure 9: Numerical solutions for discontinuous initial conditions at final time $t = 0.5$, obtained with the two numerical schemes analyzed in this project, for the smallest mesh size considered, at final time $t = 0.5$: $h(x, 0.5)$ (top) and $m(x, 0.5)$ (bottom).

It is evident that the Lax-Wendroff scheme better captures the discontinuity, with a substantially smaller

smearing out effect compared to the Lax-Friedrichs scheme⁵, 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-Friedrichs⁶.

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>.

⁵Bear in mind that the smearing effects are still present and cannot be completely avoided when solving the equation numerically.

⁶This is further discussed in [1].

A MATLAB code

A.1 Part1_1

```
1 clear
2 close all
3 clc
4
5 %%% Code by Francesco Sala and Nicolo' Viscusi %%%
6
7 % Set to true if you want to see the animation of the solutions over
time
8 animation = "True";
9
10 %% Resolution of the problem
11
12 % Definition of parameters
13 g = 1;
14 u = 0.25;
15
16 % Spatial domain
17 xspan = [0, 2];
18
19 % Temporal domain
20 tspan = [0, 0.5];
21
22 % Initial conditions
23 h0 = @(x) 1 + 0.5 * sin(pi * x);
24 m0 = @(x) u * h0(x);
25
26 % Number of grid points
27 N = 1000;
28
29 % Number of time steps
30 CFL = 0.5;
31
32 % Note that max(h0) = 1.5
33 k = CFL * (xspan(2) - xspan(1)) / N * 1 / (u + sqrt(g * 1.5));
34 K = round((tspan(end) - tspan(1)) / k);
35
36 % Source function
37 S = @(x, t) [pi/2 * (u - 1) * cos(pi * (x - t));
38             pi/2 * cos(pi * (x - t)) * (- u + u^2 + g * h0(x - t))];
39
40 % Here we use periodic boundary condition as the option ('peri')
41 bc = 'peri';
42
43 % Solve the problem
```

```

44 [h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan, tspan, N,
45     ...
46     K, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
47
48 % We visualize the solution
49 if animation == "True"
50     figure(1)
51     for i = 1 : 20 : length(tvec)
52
53         subplot(2, 1, 1)
54         plot(xvec, h(:, i), 'LineWidth', 2)
55         hold on
56         plot(xvec, h0(xvec - tvec(i)), 'Linewidth', 2)
57         title(['$h(x, t)$ at $t = $ ', num2str(tvec(i))], ...
58             'Interpreter', 'latex')
59         xlabel('$x$', 'Interpreter', 'latex')
60         ylabel('$h(x, t)$', 'Interpreter', 'latex')
61         grid on
62         xlim([0 2]);
63         ylim([0.4 1.6]);
64         hold off
65         legend('Numerical solution', 'Exact solution', ...
66             'Interpreter', 'latex')
67         set(gca, 'FontSize', 20)
68         drawnow
69
70         subplot(2, 1, 2)
71         plot(xvec, m(:, i), 'LineWidth', 2)
72         hold on
73         plot(xvec, u * h0(xvec - tvec(i)), 'Linewidth', 2)
74         title(['$m(x, t)$ at $t = $ ', num2str(tvec(i))], ...
75             'Interpreter', 'latex')
76         xlabel('$x$', 'Interpreter', 'latex')
77         ylabel('$m(x, t)$', 'Interpreter', 'latex')
78         grid on
79         xlim([0 2]);
80         ylim([0.1 0.4]);
81         hold off
82         legend('Numerical solution', 'Exact solution', ...
83             'Interpreter', 'latex')
84         set(gca, 'FontSize', 20)
85         drawnow
86
87     end
88 end
89
90 %% Error analysis
91

```

```

92 % We solve the same problem for different values of \Delta x
93 delta_x_vec = 2.^-(4:10);
94
95 % Note that we cannot solve for small values of delta_x, because we
    would
96 % need a too large matrix to store the solutions h and m
97 N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
98 err_h_vec = zeros(size(N_vec));
99 err_m_vec = zeros(size(N_vec));
100
101 for i=1:length(N_vec)
102     N = N_vec(i);
103     k = CFL * (xspan(2) - xspan(1)) / N * 1 / (u + sqrt(g * 1.5));
104     K = round((tspan(end) - tspan(1)) / k);
105     T_f = 0.5;
106     [h, m, ~, xvec, k, delta_x] = conservative_scheme(xspan, tspan, N,
        ...
107         K, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
108     err_h_vec(i) = 1/sqrt(N)*norm(h(:, end) - h0(xvec-T_f)');
109     err_m_vec(i) = 1/sqrt(N)*norm(m(:, end) - u*h0(xvec-T_f)');
110 end
111
112
113 % Plot the error
114 figure(2)
115
116 subplot(2,1,1)
117 loglog(delta_x_vec, err_h_vec, "o-", "Linewidth", 2)
118 hold on
119 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
120 xlabel('$\Delta x$', 'Interpreter', 'latex')
121 ylabel("$|e|_2$", "Interpreter", "latex")
122 title("Error on \ (h(x,t)\) at \ (t=0.5\)", "Interpreter", "latex")
123 legend("Error", "\ (\Delta x\)", "\ (\Delta x^2\)", "interpreter", ...
124     "latex", "location", "best")
125 set(gca, 'FontSize', 20)
126 grid on
127
128
129 subplot(2,1,2)
130 loglog(delta_x_vec, err_m_vec, "o-", "Linewidth", 2)
131 hold on
132 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
    "--")
133 xlabel('$\Delta x$', 'Interpreter', 'latex')
134 ylabel("$|e|_2$", "Interpreter", "latex")
135 title("Error on \ (m(x,t)\) at \ (t=0.5\)", "Interpreter", "latex")
136 legend("Error", "\ (\Delta x\)", "\ (\Delta x^2\)", "interpreter", ...

```

```

137     "latex", "location", "best")
138 grid on
139 set(gca, 'FontSize', 20)

```

A.2 Part1_2

```

1 clear
2 close all
3 clc
4
5 %%% Code by Francesco Sala and Nicolo' Viscusi %%%
6
7 % Set to true if you want to see the animation of the solutions over
time
8 animation = "True";
9
10 %% First set of initial conditions
11
12 % Spatial domain
13 xspan = [0 2];
14
15 % Temporal domain
16 tspan = [0 0.5];
17
18 % Initial conditions
19 h01 = @(x) 1 - 0.1 * sin(pi * x);
20 m01 = @(x) 0;
21
22 % Source function
23 S1 = @(x, t) [0;
24     0];
25
26 % We will use periodic boundary condition option ('peri')
27 bc = 'peri';
28
29
30 % We generate a reference solution
31 [h1_ex, m1_ex, tvec1_ex, xvec1_ex] = conservative_scheme(xspan, tspan,
32     ...
33     3000, 6000, h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);
34
35 % We now proceed with a less refined solution
36 N = 250;
37
38 % Number of time steps
39 CFL = 0.5;
40 K = N / CFL;
41

```

```

42 % Solve the problem
43 [h1, m1, tvec1, xvec1] = conservative_scheme(xspan, tspan, N, K, ...
44     h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);
45
46
47 % We visualize the solution
48 if animation == "True"
49     figure(1)
50     for i = 1 : 20 : length(tvec1)
51
52         subplot(2, 1, 1)
53         plot(xvec1, h1(:, i), 'LineWidth', 2)
54         title(['$h(x, t)$ at $t = $', num2str(tvec1(i))], ...
55             'Interpreter', 'latex')
56         xlabel('$x$', 'Interpreter', 'latex')
57         ylabel('$h(x, t)$', 'Interpreter', 'latex')
58         grid on
59         xlim([0 2]);
60         ylim([0.9 1.1]);
61         set(gca, 'FontSize', 20)
62         drawnow
63
64         subplot(2, 1, 2)
65         plot(xvec1, m1(:, i), 'LineWidth', 2)
66         title(['$m(x, t)$ at $t = $', num2str(tvec1(i))], ...
67             'Interpreter', 'latex')
68         xlabel('$x$', 'Interpreter', 'latex')
69         ylabel('$m(x, t)$', 'Interpreter', 'latex')
70         grid on
71         xlim([0 2]);
72         ylim([-0.1 0.1])
73         set(gca, 'FontSize', 20)
74         drawnow
75
76     end
77 end
78
79
80
81 %% Error analysis, initial condition 1
82
83 % We solve the same problem for different values of \Delta x
84 delta_x_vec = 2.^-(6:10);
85
86 % Note that we cannot solve for small values of delta_x, because we
87 % would
88 % need a too large matrix to store the solutions h and m
89 N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
90 err_h_vec1 = zeros(size(N_vec));

```



```

90 err_m_vec1 = zeros(size(N_vec));
91
92 for i=1:length(N_vec)
93     N = N_vec(i);
94
95     K = N / CFL;
96
97     [h1, m1, ~, xvec1_err] = conservative_scheme(xspan, tspan, N, K,
98         ...
99         h01, m01, @lax_friedrichs_flux, @flux_phys, S1, bc);
100
101     % We now want to compare h1(:, end) with h1_ex(:, end),
102     % but this second vector is defined on a different grid xvec1_ex
103     % We interpolate h1_ex(:, end) on the grid xvec1
104     h1_interp = interp1(xvec1_err, h1(:, end), xvec1_ex);
105     m1_interp = interp1(xvec1_err, m1(:, end), xvec1_ex);
106     err_h_vec1(i) = norm(h1_interp' - h1_ex(:, end));
107     err_m_vec1(i) = norm(m1_interp' - m1_ex(:, end));
108
109 end
110
111 % Plot error
112 figure(2)
113
114 subplot(2,1,1)
115 loglog(delta_x_vec, err_h_vec1, "o-", "Linewidth", 2)
116 hold on
117 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
118     "--")
119 xlabel('$\Delta x$', 'Interpreter', 'latex')
120 ylabel("$|e|_2$", "Interpreter", "latex")
121 title("Error on \h(x,t)\ at \t=0.5\ (case 1)", "Interpreter", "latex")
122
123 legend("Error", "\Delta x", "\Delta x^2", "interpreter", ...
124     "latex", "location", "best")
125 set(gca, 'FontSize', 20)
126 grid on
127
128 subplot(2,1,2)
129 loglog(delta_x_vec, err_m_vec1, "o-", "Linewidth", 2)
130 hold on
131 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
132     "--")
133 xlabel('$\Delta x$', 'Interpreter', 'latex')
134 ylabel("$|e|_2$", "Interpreter", "latex")
135 title("Error on \m(x,t)\ at \t=0.5\ (case 1)", "Interpreter", "latex")
136
137 legend("Error", "\Delta x", "\Delta x^2", "interpreter", ...

```

```

134     "latex", "location", "best")
135 grid on
136 set(gca, 'FontSize', 20)
137
138
139
140 %% Second set of initial conditions
141
142 % Initial conditions
143 h02 = @(x) 1 - 0.2 * sin(2 * pi * x);
144 m02 = @(x) 0.5;
145
146 % Source term
147 S2 = @(x, t) [0;
148     0];
149
150 % All the other parameters remain the same...
151
152 % First, a refined solution as reference "exact"
153 [h2_ex, m2_ex, tvec2_ex, xvec2_ex] = conservative_scheme(xspan, ...
154     tspan, 3000, 6000, h02, m02, @lax_friedrichs_flux, @flux_phys, S2,
155     bc);
156
157 % Solve the problem on a less refined mesh
158 N = 100;
159 K = 200;
160 [h2, m2, tvec2, xvec2] = conservative_scheme(xspan, tspan, N, K, ...
161     h02, m02, @lax_friedrichs_flux, @flux_phys, S2, bc);
162
163
164 % We visualize the solution
165 if animation == "True"
166     figure(3)
167     for i = 1 : 20 : length(tvec2)
168
169         subplot(2, 1, 1)
170         plot(xvec2, h2(:, i), 'LineWidth', 2)
171         title(['$h(x, t)$ at $t = $', num2str(tvec2(i))], ...
172             'Interpreter', 'latex')
173         xlabel('$x$', 'Interpreter', 'latex')
174         ylabel('$h(x, t)$', 'Interpreter', 'latex')
175         grid on
176         xlim([0 2]);
177         ylim([0.8 1.2]);
178         set(gca, 'FontSize', 20)
179         drawnow
180
181         subplot(2, 1, 2)

```

```

182         plot(xvec2, m2(:, i), 'LineWidth', 2)
183         title(['$m(x, t)$ at $t = $', num2str(tvec2(i))], ...
184             'Interpreter', 'latex')
185         xlabel('$x$', 'Interpreter', 'latex')
186         ylabel('$m(x, t)$', 'Interpreter', 'latex')
187         grid on
188         xlim([0 2]);
189         ylim([0.4 0.6])
190         set(gca, 'FontSize', 20)
191         drawnow
192
193     end
194 end
195
196
197
198 %% Error analysis, initial condition 2
199
200 % We solve the same problem for different values of \Delta x
201 delta_x_vec = 2.^-(6:10);
202
203 % Note that we cannot solve for small values of delta_x, because we
204 % would
205 % need a too large matrix to store the solutions h and m
206 N_vec = (xspan(2) - xspan(1)) ./ delta_x_vec ;
207 err_h_vec2 = zeros(size(N_vec));
208 err_m_vec2 = zeros(size(N_vec));
209
210 for i=1:length(N_vec)
211     N = N_vec(i);
212
213     K = N / CFL;
214
215     [h2, m2, ~, xvec2_err] = conservative_scheme(xspan, tspan, N, K,
216         ...
217         h02, m02, @lax_friedrichs_flux, @flux_phys, S2, bc);
218
219     % We now want to compare h1(:, end) with h1_ex(:, end), but this
220     % second vector is defined on a different grid xvec1_ex
221     % We interpolate h1_ex(:, end) on the grid xvec1
222     h2_interp = interp1(xvec2_err, h2(:, end), xvec2_ex);
223     m2_interp = interp1(xvec2_err, m2(:, end), xvec2_ex);
224
225     err_h_vec2(i) = norm(h2_interp' - h2_ex(:, end));
226     err_m_vec2(i) = norm(m2_interp' - m2_ex(:, end));
227
228 end

```

```

229 % Plot the error
230 figure(4)
231
232 subplot(2,1,1)
233 loglog(delta_x_vec, err_h_vec2, "o-", "Linewidth", 2)
234 hold on
235 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
236        "--")
237 xlabel('$\Delta x$', 'Interpreter', 'latex')
238 ylabel("$|e|_2$", "Interpreter","latex")
239 title("Error on \h(x,t)\ at \t=0.5\ (case 2)", "Interpreter","latex")
240 legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
241        "latex", "location", "best")
242 set(gca, 'FontSize', 20)
243 grid on
244
245 subplot(2,1,2)
246 loglog(delta_x_vec, err_m_vec2, "o-", "Linewidth", 2)
247 hold on
248 loglog(delta_x_vec, delta_x_vec, "--", delta_x_vec, delta_x_vec.^2,
249        "--")
250 xlabel('$\Delta x$', 'Interpreter', 'latex')
251 ylabel("$|e|_2$", "Interpreter","latex")
252 title("Error on \m(x,t)\ at \t=0.5\ (case 2)", "Interpreter","latex")
253 legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "interpreter", ...
254        "latex", "location", "best")
255 grid on
256 set(gca, 'FontSize', 20)

```

A.3 Part1_3

```

1 clear
2 close all
3 clc
4
5 %%% Code by Francesco Sala and Nicolo' Viscusi %%%
6
7 % Set to true if you want to see the animation of the solutions over
8 % time
9 animation = "True";
10
11 %%% Definition of parameters
12
13 % Boundary conditions option
14 bc = 'open';
15
16 % Initial conditions

```

```

16 h0 = @(x) 1;
17 m0 = @(x) -0.5 * (x < 1);
18
19 % Source term
20 S = @(x, t) [0;
21             0];
22
23 % Spatial domain
24 xspan = [0, 2];
25
26 % Temporal domain
27 tspan = [0 0.5];
28
29 % Number of points in space and time
30 N = 100;
31 K = 200;
32
33
34
35 %% Solve the problem with Lax-Friedrichs flux
36 [h, m, tvec, xvec] = conservative_scheme(xspan, tspan, N, K, h0,...
37     m0, @lax_friedrichs_flux, @flux_phys, S, bc);
38
39 % Reference solution with very fine mesh
40 [h_exf, m_exf, tvec_exf, xvec_exf] = conservative_scheme(xspan, ...
41     tspan, 3000, 6000, h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc)
42     ;
43
44 % Animation of the solution
45 % Visualize the solution
46 if animation == "True"
47     figure(1);
48     for i = 1 : 20 : length(tvec)
49
50         subplot(2, 1, 1)
51         plot(xvec, h(:, i), 'LineWidth', 2)
52         title(['$h(x, t)$ at $t = $', num2str(tvec(i))], ...
53             'Interpreter', 'latex')
54         xlabel('$x$', 'Interpreter', 'latex')
55         ylabel('$h(x, t)$', 'Interpreter', 'latex')
56         grid on
57         xlim([0 2]);
58         set(gca, 'FontSize', 20)
59         drawnow
60
61         subplot(2, 1, 2)
62         plot(xvec, m(:, i), 'LineWidth', 2)
63         title(['$m(x, t)$ at $t = $', num2str(tvec(i))], ...
64             'Interpreter', 'latex')

```

```

64         xlabel('$x$', 'Interpreter', 'latex')
65         ylabel('$m(x, t)$', 'Interpreter', 'latex')
66         grid on
67         xlim([0 2]);
68         set(gca, 'FontSize', 20)
69         drawnow
70
71     end
72 end
73
74
75 % Compute a set of numerical solutions obtained with gradually
76 % decreasing mesh sizes
77 [h1, m1, ~, xvec1] = conservative_scheme(xspan, tspan, 100, 200,...
78     h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
79 [h2, m2, ~, xvec2] = conservative_scheme(xspan, tspan, 250, 500,...
80     h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
81 [h3, m3, ~, xvec3] = conservative_scheme(xspan, tspan, 400, 800, ...
82     h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
83 [h4, m4, ~, xvec4] = conservative_scheme(xspan, tspan, 800, 1600,...
84     h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
85 [h5, m5, ~, xvec5] = conservative_scheme(xspan, tspan, 1500, 3000, ...
86     h0, m0, @lax_friedrichs_flux, @flux_phys, S, bc);
87
88 figure(2)
89 subplot(2, 1, 1)
90 plot(xvec_exf, h_exf(:, end), '-b', 'LineWidth', 3)
91 hold on
92 plot(xvec1, h1(:, end), 'LineWidth', 2)
93 plot(xvec2, h2(:, end), 'LineWidth', 2)
94 plot(xvec3, h3(:, end), 'LineWidth', 2)
95 plot(xvec4, h4(:, end), 'LineWidth', 2)
96 plot(xvec5, h5(:, end), 'LineWidth', 2)
97 legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$',...
98     '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
99     best')
100 xlabel('$x$', 'Interpreter', 'latex')
101 ylabel('$h(x, t)$', 'Interpreter', 'latex')
102 grid on
103 xlim([0 2]);
104 set(gca, 'FontSize', 20)
105
106 subplot(2, 1, 2)
107 plot(xvec_exf, m_exf(:, end), '-b', 'LineWidth', 3)
108 hold on
109 plot(xvec1, m1(:, end), 'LineWidth', 2)
110 plot(xvec2, m2(:, end), 'LineWidth', 2)
111 plot(xvec3, m3(:, end), 'LineWidth', 2)

```

```

112 plot(xvec4, m4(:, end), 'LineWidth', 2)
113 plot(xvec5, m5(:, end), 'LineWidth', 2)
114 legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$', ...
115         '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
        best')
116 xlabel('$x$', 'Interpreter', 'latex')
117 ylabel('$m(x, t)$', 'Interpreter', 'latex')
118 grid on
119 xlim([0 2]);
120 set(gca, 'FontSize', 20)
121
122
123
124 %% Solve the problem with Lax-Wendroff flux
125 [h_lw, m_lw, tvec_lw, xvec_lw] = conservative_scheme(xspan, tspan, N,
    ...
126             K, h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
127
128 % Reference solution with very fine mesh
129 [h_exw, m_exw, tvec_exw, xvec_exw] = conservative_scheme(xspan, tspan,
    ...
130             3000, 6000, h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
131
132 % Animation of the solution
133 % Visualize the solution
134 if animation == "True"
135     figure(3);
136     for i = 1 : 20 : length(tvec_lw)
137
138         subplot(2, 1, 1)
139         plot(xvec_lw, h_lw(:, i), 'LineWidth', 2)
140         title(['Lax-Wendroff: $h(x, t)$ at $t = $', ...
141             num2str(tvec_lw(i))], 'Interpreter', 'latex')
142         xlabel('$x$', 'Interpreter', 'latex')
143         ylabel('$h(x, t)$', 'Interpreter', 'latex')
144         grid on
145         xlim([0 2]);
146         set(gca, 'FontSize', 20)
147         drawnow
148
149         subplot(2, 1, 2)
150         plot(xvec_lw, m_lw(:, i), 'LineWidth', 2)
151         title(['Lax-Wendroff: $m(x, t)$ at $t = $', ...
152             num2str(tvec_lw(i))], 'Interpreter', 'latex')
153         xlabel('$x$', 'Interpreter', 'latex')
154         ylabel('$m(x, t)$', 'Interpreter', 'latex')
155         grid on
156         xlim([0 2]);
157         set(gca, 'FontSize', 20)

```

```

158         drawnow
159
160     end
161 end
162
163
164 % We now compute a set of numerical solutions obtained with gradually
165 % decreasing mesh sizes
166 [h1, m1, ~, xvec1] = conservative_scheme(xspan, tspan, 100, 200, ...
167     h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
168 [h2, m2, ~, xvec2] = conservative_scheme(xspan, tspan, 250, 500, ...
169     h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
170 [h3, m3, ~, xvec3] = conservative_scheme(xspan, tspan, 400, 800, ...
171     h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
172 [h4, m4, ~, xvec4] = conservative_scheme(xspan, tspan, 800, 1600, ...
173     h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
174 [h5, m5, ~, xvec5] = conservative_scheme(xspan, tspan, 1500, 3000, ...
175     h0, m0, @lax_wendroff_flux, @flux_phys, S, bc);
176
177 figure(4)
178 subplot(2, 1, 1)
179 plot(xvec_exw, h_exw(:, end), '-b', 'LineWidth', 3)
180 hold on
181 plot(xvec1, h1(:, end), 'LineWidth', 2)
182 plot(xvec2, h2(:, end), 'LineWidth', 2)
183 plot(xvec3, h3(:, end), 'LineWidth', 2)
184 plot(xvec4, h4(:, end), 'LineWidth', 2)
185 plot(xvec5, h5(:, end), 'LineWidth', 2)
186 legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$', ...
187     '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
188     best')
189 xlabel('$x$', 'Interpreter', 'latex')
190 ylabel('$h(x, t)$', 'Interpreter', 'latex')
191 grid on
192 xlim([0 2]);
193 set(gca, 'FontSize', 20)
194
195 subplot(2, 1, 2)
196 plot(xvec_exw, m_exw(:, end), '-b', 'LineWidth', 3)
197 hold on
198 plot(xvec1, m1(:, end), 'LineWidth', 2)
199 plot(xvec2, m2(:, end), 'LineWidth', 2)
200 plot(xvec3, m3(:, end), 'LineWidth', 2)
201 plot(xvec4, m4(:, end), 'LineWidth', 2)
202 plot(xvec5, m5(:, end), 'LineWidth', 2)
203 legend('Ref: $N = 3000$', '$N = 100$', '$N = 250$', '$N = 400$', ...
204     '$N = 800$', '$N = 1500$', 'Interpreter', 'latex', 'Location', '
205     best')

```



```

205 xlabel('$x$', 'Interpreter', 'latex')
206 ylabel('$m(x, t)$', 'Interpreter', 'latex')
207 grid on
208 xlim([0 2]);
209 set(gca, 'FontSize', 20)
210
211
212 %% Comparison between finest solutions of Lax-Friedrichs and Lax-
    Wendroff
213 figure(5)
214 subplot(2, 1, 1)
215 plot(xvec_exw, h_exw(:, end), 'LineWidth', 2)
216 hold on
217 plot(xvec_exf, h_exf(:, end), 'LineWidth', 2)
218 legend('Lax-Wendroff', 'Lax-Friedrichs', 'Interpreter', 'latex', ...
219     'Location', 'best')
220 xlabel('$x$', 'Interpreter', 'latex')
221 ylabel('$h(x, t)$', 'Interpreter', 'latex')
222 grid on
223 xlim([0 2]);
224 set(gca, 'FontSize', 20)
225
226
227 subplot(2, 1, 2)
228 plot(xvec_exw, m_exw(:, end), 'LineWidth', 2)
229 hold on
230 plot(xvec_exf, m_exf(:, end), 'LineWidth', 2)
231 legend('Lax-Wendroff', 'Lax-Friedrichs', 'Interpreter', 'latex', ...
232     'Location', 'best')
233 xlabel('$x$', 'Interpreter', 'latex')
234 ylabel('$m(x, t)$', 'Interpreter', 'latex')
235 grid on
236 xlim([0 2]);
237 set(gca, 'FontSize', 20)

```

A.4 Conservative scheme implementation

```

1 function [h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan,
    ...
2     tspan, N, K, h0, m0, numerical_flux, flux_phys, S, bc)
3
4
5 % CONSERVATIVE_SCHEME - Implements the conservative scheme
6 %                         for solving hyperbolic PDEs.
7 %
8 % [h, m, tvec, xvec, k, delta_x] = conservative_scheme(xspan, ...
9 %     tspan, N, K, h0, m0, numerical_flux, flux_phys, S, bc)
10 %
11 % INPUTS:

```

```

12 %   xspan           - Spatial domain [x_start, x_end].
13 %   tspan           - Temporal domain [t_start, t_end].
14 %   N               - Number of spatial grid points.
15 %   K               - Number of temporal grid points.
16 %   h0              - Function handle for initial water depth.
17 %   m0              - Function handle for initial discharge.
18 %   numerical_flux  - Function handle for numerical flux computation.
19 %   flux_phys        - Function handle for the physical flux function.
20 %   S                - Source term function.
21 %   bc              - Boundary condition: 'peri' (periodic), 'open' (
    open).
22 %
23 % OUTPUTS:
24 %   h               - Water depth (matrix) over space and time.
25 %   m               - Discharge (matrix) over space and time.
26 %   tvec            - Temporal grid vector.
27 %   xvec            - Spatial grid vector.
28 %   k               - Time step size.
29 %   delta_x         - Spatial grid spacing.
30 %
31 % DESCRIPTION:
32 %   This function implements the conservative scheme for solving
    hyperbolic
33 %   partial differential equations (PDEs) that model shallow water flow
    .
34 %   It evolves the water depth (h) and discharge (m) over a specified
35 %   spatial and temporal domain using the conservative scheme formula:
36 %
37 %    $u_j^{n+1} = u_j^n - k/h * (F_{j+1/2}^n - F_{j-1/2}^n) + k * S_j^n,$ 
38 %
39 %   where u represents [h; m], F is the flux, S is the source term, k
    is
40 %   the time step, and delta_x is the spatial grid spacing. The
    boundary
41 %   conditions (bc) can be set to 'peri' (periodic), 'open' (open).
42 %
43 % Authors: [Francesco Sala, Nicolò Viscusi]
44 % December 2023
45
46
47 tvec      = linspace(tspan(1), tspan(end), K + 1);
48 xvec      = linspace(xspan(1), xspan(end), N + 1);
49 h         = zeros(N + 1, K + 1);
50 m         = zeros(N + 1, K + 1);
51
52 delta_x   = (xspan(2) - xspan(1)) / N;
53 k         = (tspan(2) - tspan(1)) / K;
54
55 % Set the initial conditions

```

```

56 h(:, 1) = h0(xvec);
57 m(:, 1) = m0(xvec);
58
59
60 for i = 2 : length(tvec)
61
62     j = 1;
63     source = S(xvec(j), tvec(i));
64
65     if bc == 'peri'
66
67         rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
68             [h(j + 1, i - 1); m(j + 1, i - 1)], delta_x, k) - ...
69             numerical_flux(flux_phys, [h(end - 1, i - 1); ...
70                 m(end - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k)
71             ;
72         h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
73         m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);
74
75     elseif bc == 'open'
76
77         rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
78             [h(j + 1, i - 1); m(j + 1, i - 1)], delta_x, k) - ...
79             numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
80                 [h(j, i - 1); m(j, i - 1)], delta_x, k);
81         h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
82         m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);
83
84     end
85
86     for j = 2 : (length(xvec) - 1)
87
88         source = S(xvec(j), tvec(i));
89         rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
90             [h(j + 1, i - 1); m(j + 1, i - 1)], delta_x, k) - ...
91             numerical_flux(flux_phys, [h(j - 1, i - 1); ...
92                 m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
93         h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
94         m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);
95
96     end
97
98     j = length(xvec);
99     source = S(xvec(j), tvec(i));
100
101     if bc == 'peri'
102
103         rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
104             [h(2, i - 1); m(2, i - 1)], delta_x, k) - ...

```

```

104         numerical_flux(flux_phys, [h(j - 1, i - 1); ...
105             m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
106     h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
107     m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);
108
109     elseif bc == 'open'
110
111         rhs = numerical_flux(flux_phys, [h(j, i - 1); m(j, i - 1)], ...
112             [h(j, i - 1); m(j, i - 1)], delta_x, k) - ...
113             numerical_flux(flux_phys, [h(j - 1, i - 1); ...
114                 m(j - 1, i - 1)], [h(j, i - 1); m(j, i - 1)], delta_x, k);
115         h(j, i) = h(j, i - 1) - k/delta_x * rhs(1) + k * source(1);
116         m(j, i) = m(j, i - 1) - k/delta_x * rhs(2) + k * source(2);
117
118     end
119
120 end
121
122 end

```

A.5 Physical flux

```

1  function f = flux_phys(q)
2
3  % FLUX_PHYS - Computes the physical flux function for the shallow
4  %              water equations.
5  %
6  %   f = flux_phys(q)
7  %
8  % INPUTS:
9  %   q - Vector of state variables [h, m], where
10 %       h: Water depth
11 %       m: Discharge
12 %
13 % OUTPUT:
14 %   f - Vector representing the physical flux corresponding to the
15 %       input
16 %       state q.
17 %
18 % DESCRIPTION:
19 %   This function calculates the physical flux for the shallow water
20 %   equations.
21 %
22 % Authors: [Francesco Sala, Nicolò Viscusi]
23 % December 2023
24
25 g = 1;
26 h = q(1);
27 m = q(2);

```

```

27
28 f = [m;
29      m.^2./h + 1/2*g*h.^2];
30
31 return

```

A.6 Numerical flux

A.6.1 Lax-Friedrichs

```

1 function F = lax_friedrichs_flux(flux_phys, u, v, delta_x, k)
2
3 % LAX_FRIEDRICHS_FLUX - Computes the Lax-Friedrichs flux for a given
4 %                        physical flux function.
5 %
6 % F = lax_friedrichs_flux(flux_phys, u, v, delta_x, k)
7 %
8 % INPUTS:
9 % flux_phys - Function handle for the physical flux function.
10 % u         - State vector at position j (qj).
11 % v         - State vector at position j+1 (qj+1).
12 % delta_x   - Spatial grid spacing.
13 % k         - Time step size.
14 %
15 % OUTPUT:
16 % F         - Lax-Friedrichs flux between u and v.
17 %
18 % DESCRIPTION:
19 % This function calculates the Lax-Friedrichs flux between two
20 % neighboring states, u and v, based on the given physical flux
21 % function
22 % (flux_phys).
23 %
24 % Authors: [Francesco Sala, Nicolo' Viscusi]
25 % December 2023
26
27 F = 0.5 * (flux_phys(u) + flux_phys(v) - delta_x / k * (v - u));
28
29 return

```

A.6.2 Lax-Wendroff

```

1 function F = lax_wendroff_flux(flux_phys, u, v, delta_x, k)
2 % LAX_WENDROFF_FLUX - Computes the Lax-Wendroff flux for a given
3 %                        physical flux function.
4 %
5 % F = lax_wendroff_flux(flux_phys, u, v, delta_x, k)
6 %
7 % INPUTS:

```

```

8 % flux_phys - Function handle for the physical flux function.
9 % u - State vector at position j (qj).
10 % v - State vector at position j+1 (qj+1).
11 % delta_x - Spatial grid spacing.
12 % k - Time step size.
13 %
14 % OUTPUT:
15 % F - Lax-Wendroff flux between u and v.
16 %
17 % DESCRIPTION:
18 % This function calculates the Lax-Wendroff flux between two
19 % neighboring states, u and v, based on the given physical flux
20 % function (flux_phys).
21 %
22 %
23 % Authors: [Francesco Sala, Nicolo' Viscusi]
24 % December 2023
25
26 F = 0.5 * (flux_phys(u) + flux_phys(v) - k / delta_x * ...
27     flux_phys_prime((u + v) / 2) * (flux_phys(v) - flux_phys(u)));
28
29 function f_prime = flux_phys_prime(q)
30     % Compute the Jacobian of the flux
31     g = 1;
32     h = q(1);
33     m = q(2);
34     f_prime = [0, 1;
35         -m.^2./(h.^2) + g*h, 2 * m./h ];
36 end
37 end

```