Project II report

WENO schemes for the shallow water equations

Authors: Nicolò Giosuè Carlo Viscusi
Francesco Sala

Date: January 19, 2024
Course: Numerical Methods for Conservation Laws - MATH-459
Professor: Martin Licht
Assistant: Fernando Henriquez

Contents

1 Introduction

In this project, Weighted Essentially Non-Oscillatory (WENO) schemes are 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 hyperbolic PDEs 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 over the temporal domain $\Omega_t = [0,1]$. $g = 1$ is assumed throughout the whole project. All the implemented code can be found here on GitHub.

2 Part I - Theoretical framework and motivation

A brief theoretical overview and motivation behind the ENO and WENO schemes are proposed[2]. First of all, the motivation behind ENO schemes is to achieve a higher-order approximation of the solution to a conservation law. Indeed, due to Godunov's barrier theorem ([1]), monotone schemes and linear schemes have the downfall of only first-order convergence. ENO schemes rely on a higher-order reconstruction of the cell values at the interface thanks to polynomial interpolation. Each value at the interface is obtained by selecting the smoothest among all possible stencils that can be used for the polynomial interpolation, where the smoothness is quantified using divided differences. The possibility of choosing adaptively the most suitable stencil makes ENO schemes highly non-linear. Indeed, if the stencil could not be varied, we would simply be using a linear scheme, known for only allowing first-order convergence. Such a method achieves $\mathcal{O}(\Delta x^K)$ accuracy, with $K-1$ the order of the polynomial reconstruction employed, while the dimension of the stencils used for reconstruction is K.

The motivation behind WENO schemes arises from the inefficient use of the available nodes from ENO schemes. Indeed, ENO schemes only allow for $\mathcal{O}(\Delta x^K)$ approximation in space, while evaluating $2K-1$ nodes, i.e. we are not making efficient use of the nodes. Besides, as reported in [1], "The nonlinear nature of the stencil selection suggests that even small changes in the solution can result in a different stencil choice. Hence, even for smooth problems, the error behavior can be noisy and uneven". These are the motivations for the extension of ENO schemes to obtain WENO. In this case, all the stencils available for the polynomial interpolation of the interface values are weighed and used for the reconstruction. The weights are chosen adaptively, taking into account the smoothness of the solution, so as to achieve the desired non-linear behavior. In particular, the weights are such that in smooth regions we have $\mathcal{O}(\Delta x^{2K-1})$ convergence ensured, while in non-smooth we have an ENO scheme-like behavior, i.e. the smoothest stencils are preferred for the reconstruction, thus passing to a convergence of $\mathcal{O}(\Delta x^K)$.

3 Part II - About the implementation

The WENO scheme is implemented by exploiting some functions made available in [1]. The main functions are `part2_3.m` and `part2_4.m`, which implement the problem cases described in the project

[1] In this context, a bold symbol is used for vectorial quantities.
[2] For a more rigorous discussion, the reader shall refer to [1].

description. The solution is evolved from time $t = 0$ to $t = T$ by the script `solver.m`. The initial condition is integrated over the cells using the `integral` command in Matlab. Then, at each time step, the solution is integrated in time using the Strong Stability Preserving (SSP) RK3 scheme, that is:

$$\begin{aligned}
\mathbf{q}_1 &= \mathbf{q}^n + k\,\mathrm{RHS}(\mathbf{q}^n, t^n) \\
\mathbf{q}_2 &= \frac{3}{4}\mathbf{q}^n + \frac{1}{4}(\mathbf{q}_1 + k\,\mathrm{RHS}(\mathbf{q}_1, t^n + k)) \\
\mathbf{q}_3 &= \frac{1}{3}\mathbf{q}^n + \frac{2}{3}(\mathbf{q}_2 + k\,\mathrm{RHS}(\mathbf{q}_2, t^n + \frac{k}{2})) \\
\mathbf{q}^{n+1} &= \mathbf{q}_3
\end{aligned}$$

Note that this scheme has an order of accuracy equal to 3. The timestep k is computed dynamically at each time according to a CFL condition:

$$k = \mathrm{CFL}\frac{\Delta x}{\max_i(|u_i| + \sqrt{gh_i}} \tag{3}$$

with $\Delta x = \frac{2}{N}$, where N is the number of cells and 2 is the width of the spatial domain Ω. If a source term is present, this is integrated exactly over the cells, to speed up the computations. The RHS that appears in the RK3 scheme is updated using the function `evalRHS`. This function in turn calls the script `WENO.m` to perform the actual reconstruction. In this project, only $K = 2$ is considered, but the implemented code can deal with any value of K. Once the reconstructed values are computed, the chosen numerical flux (Lax-Friedrichs in the present work) is exploited to compute the flux across the cells.

Concerning the 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 fictitious "external nodes" (or ghost cells) to update the solution in the same manner as for the interior nodes. The number of fictitious nodes depends on the dimension of the stencil used to perform the reconstruction. This computation is performed by `apply_bc.m`. A slightly different approach is followed 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 spatial domain is repeated to the left and right of the domain over fictitious nodes to be able to perform the reconstruction over the first and last cells. The problem is then solved, and the matrices `h` and `m` (containing the solution) are returned as outputs, as well as the discretized space and time vectors (`xc` and `tvec`)[3].

4 Part III - Source term

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:

$$h(x,0) = h_0(x) = 1 + 0.5\sin(\pi x), \quad m(x,0) = m_0(x) = uh_0(x) \tag{4}$$

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

$$\mathbf{S}(x,t) = \begin{bmatrix} \frac{\pi}{2}(u-1)\cos(\pi(x-t)) \\ \frac{\pi}{2}(-u+u^2+gh_0(x-t))\cos(\pi(x-t)) \end{bmatrix} \tag{5}$$

[3]The reader may refer to Appendix A for insights into the code.

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

$$h(x,t) = h_0(x-t), \quad m(x,t) = uh(x,t) \tag{6}$$

Regarding the numerical computations, periodic boundary conditions are selected (`bc = 'peri'`), and the time step k is chosen dynamically as described above with CFL $= 0.5$. The exact and the numerical solutions at the last time $T = 1$ are plotted in Fig. 1. Note that almost no difference between the exact and numerical solution for both $h(x,\,1)$ and $m(x,\,1)$ is noticeable. Indeed, despite using only $N = 500$ cells for discretizing the spatial domain, the two solutions overlap perfectly. Such an achievement is possible thanks to the high-order reconstruction of the WENO scheme, coupled with the order 3 accuracy of SSP-RK3.

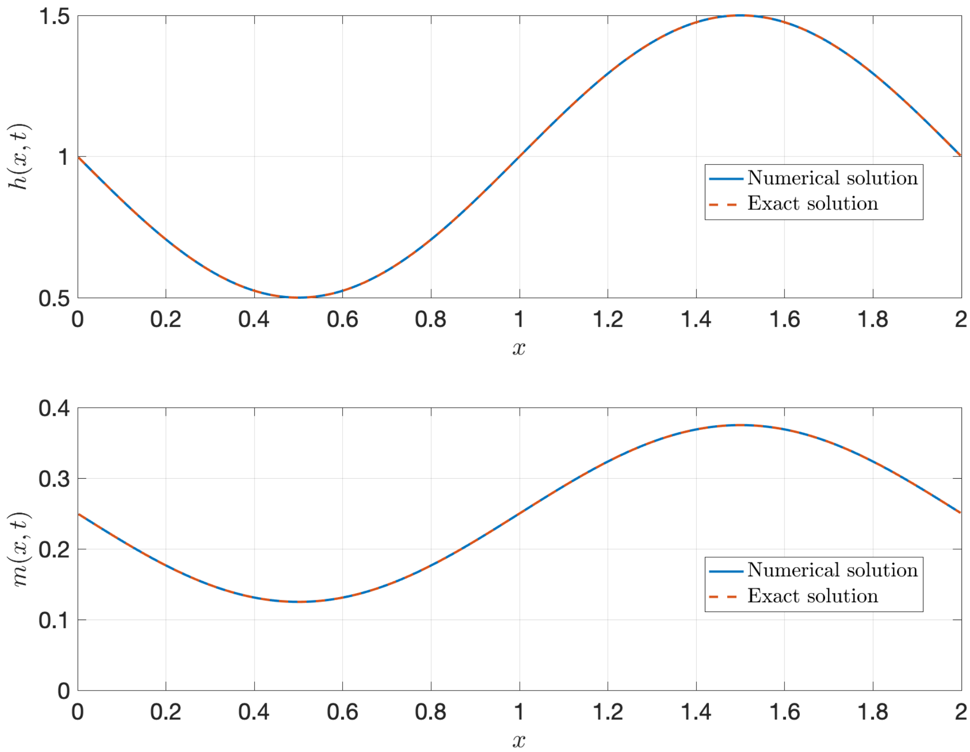

Figure 1: Comparison between exact and numerical solution for $h(x,\,1)$ (top) and $m(x,\,1)$ (bottom). Any difference between the two curves is barely discernible.

4.1 Error analysis

The order of convergence of the scheme is assessed by solving the problem multiple times varying the value of Δx, such that $\Delta x = 2^i$, $i = -6, \ldots, -9$. 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 the error is shown in Fig. 2. In both cases it is possible to note that the solution converges with order 3 in space: this comes as no surprise since the adopted WENO scheme makes use of order-2 polynomial reconstruction of the interface values, i.e. order of convergence $2K - 1 = 3$, and the time-stepping method SSP-RK3 is also of order 3.

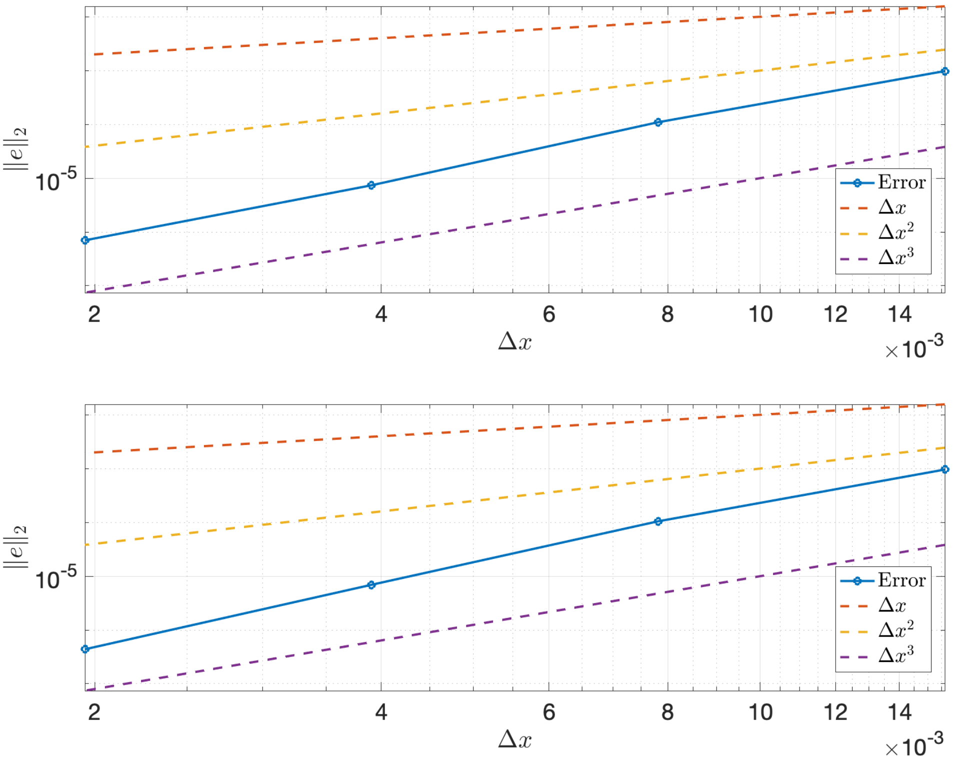

Figure 2: Log-log plot of the ℓ_2 norm of the error between exact and numerical solution at time $T = 1$ for $h(x,\,1)$ (top) and $m(x,\,1)$ (bottom). Order 1, 2, and 3 convergence rates are also plotted, to allow for comparison.

5 Part IV - Zero source term

The implemented numerical scheme is tested once again, this time with different parameters and no exact solution available. In particular, two sets of initial conditions are considered. The first set reads:

$$h(x,0) = h_0(x) = 1 - 0.1\sin(\pi x), \quad m(x,0) = m_0(x) = 0 \tag{7}$$

while the second set of initial conditions is:

$$h(x,0) = h_0(x) = 1 - 0.2\sin(2\pi x), \quad m(x,0) = m_0(x) = 0.5 \tag{8}$$

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

5.1 Numerical results

The results for the first and second set of initial conditions at the last time step $T = 1$ are plotted in Fig. 3 and Fig. 4, respectively.

Again, the numerical solution computed seems to be correctly converging also with a reasonably coarse spatial discretization if compared to the correct numerical solutions available from the previous project. The WENO scheme performs extremely well, even with not very regular solutions, as in the case shown in Fig. 4 for the second set of initial conditions. Note that the second set of initial conditions leads to a discontinuous solution at $T = 1$: this will affect the order of accuracy of the scheme.

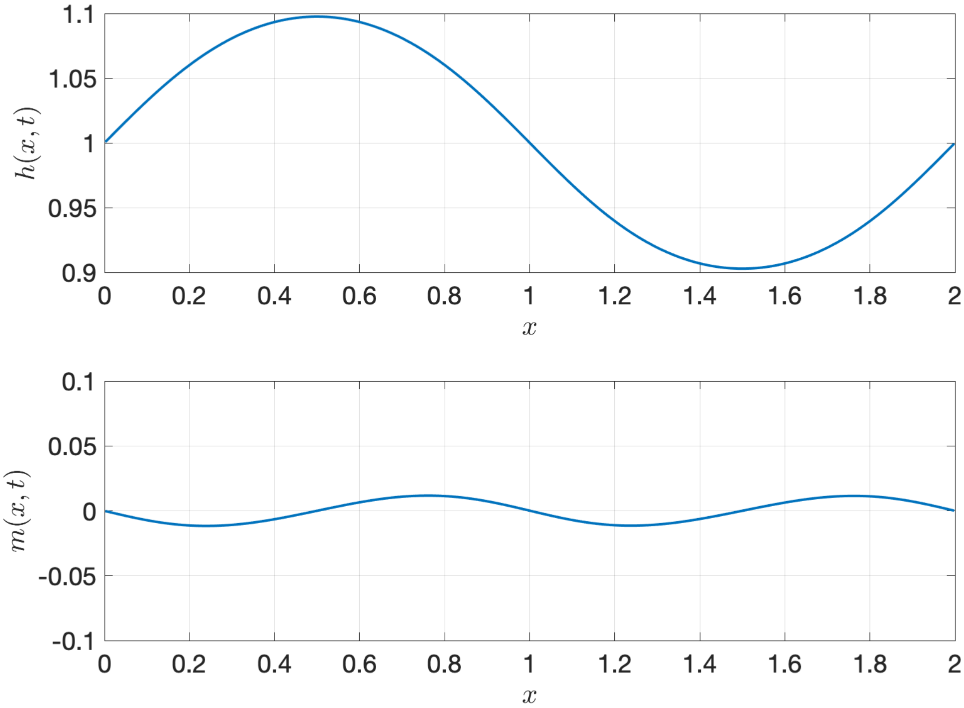

Figure 3: Numerical solution for $h(x,\,1)$ (top) and $m(x,\,1)$ (bottom) for the first set of initial conditions shown in Eq. 7.

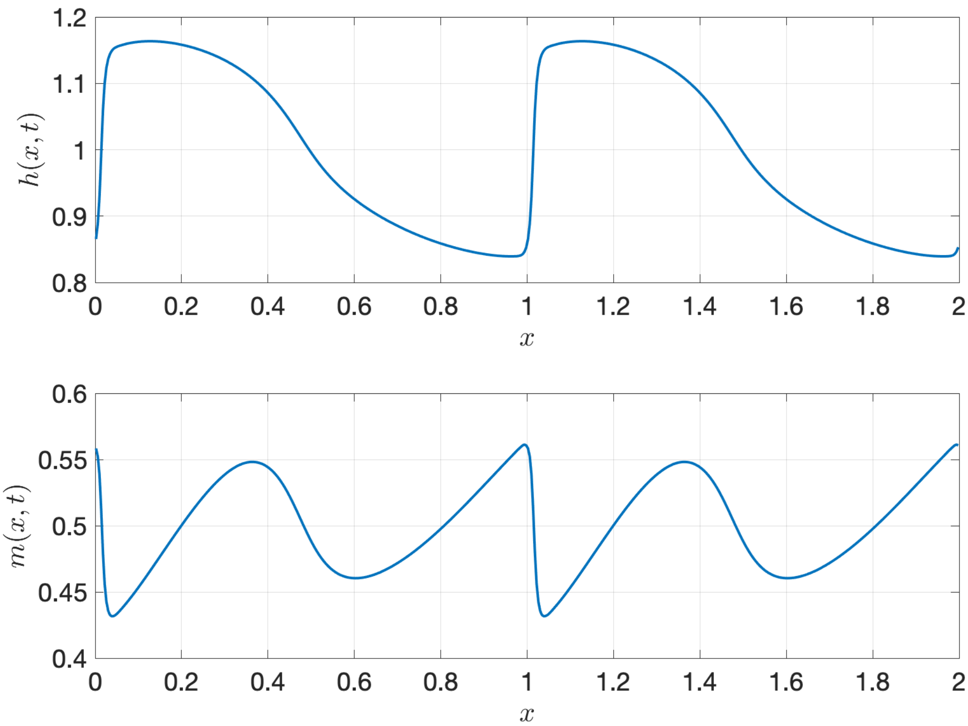

Figure 4: Numerical solution for $h(x,\,1)$ (top) and $m(x,\,1)$ (bottom) for the second set of initial conditions shown in Eq. 8.

5.2 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 $\Delta x = 2^i$, $i = -6, \ldots, -9$ again. 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 with $N = 1500$. 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.

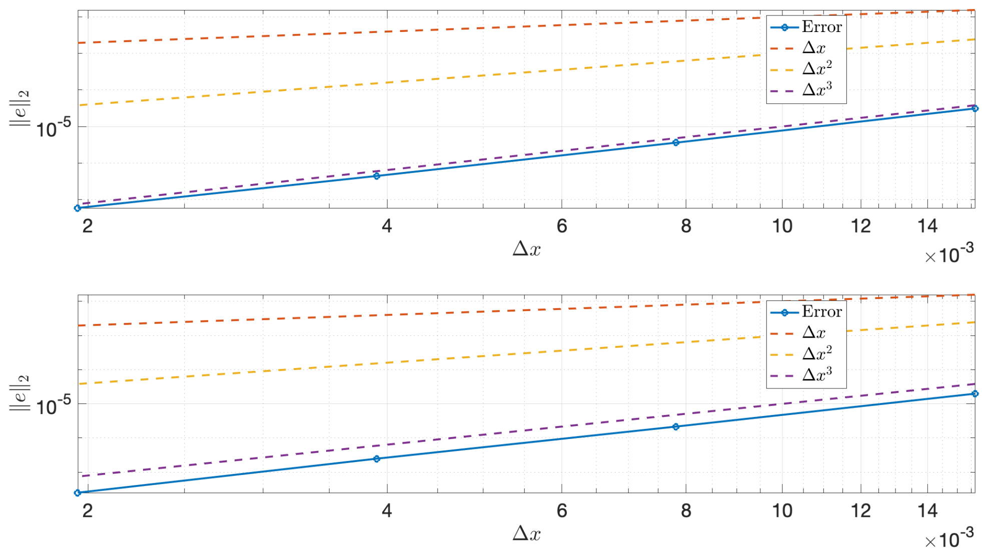

Figure 5: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 1$ for $h(x, 1)$ (top) and $m(x, 1)$ (bottom), for the first set of initial conditions, shown in Eq. 7. Order 1, 2, and 3 convergence rates are also plotted, to ease the comparison.

In the first case, it can be noted that the order of convergence of the numerical solution is third order in space. On the other hand, the order of convergence for the second set of initial conditions, while being initially parallel to Δx, becomes then parallel to Δx^2 for smaller values of Δx. This can be explained because, in the presence of a discontinuity, WENO schemes reduce to ENO schemes, that is the accuracy of the scheme changes abruptly to $\mathcal{O}(\Delta x^K)$. We hence expect to reach second-order accuracy. Further work should focus on further reducing the experimented Δx to see a slope parallel to Δx^2 and confirm the theoretical results. Fig. 7 shows the reference solution that was employed for the error computation.

6 Conclusion

In the present project, WENO schemes were implemented for the solution of a system of hyperbolic PDEs. It was possible to show that, given the order of the polynomial for the reconstruction $K - 1$, WENO schemes reach an accuracy of $2K - 1$. This is true as long as no discontinuities appear in the solution: in such a case, WENO schemes reduce to ENO schemes, yielding an accuracy of order K, as it was proved empirically.

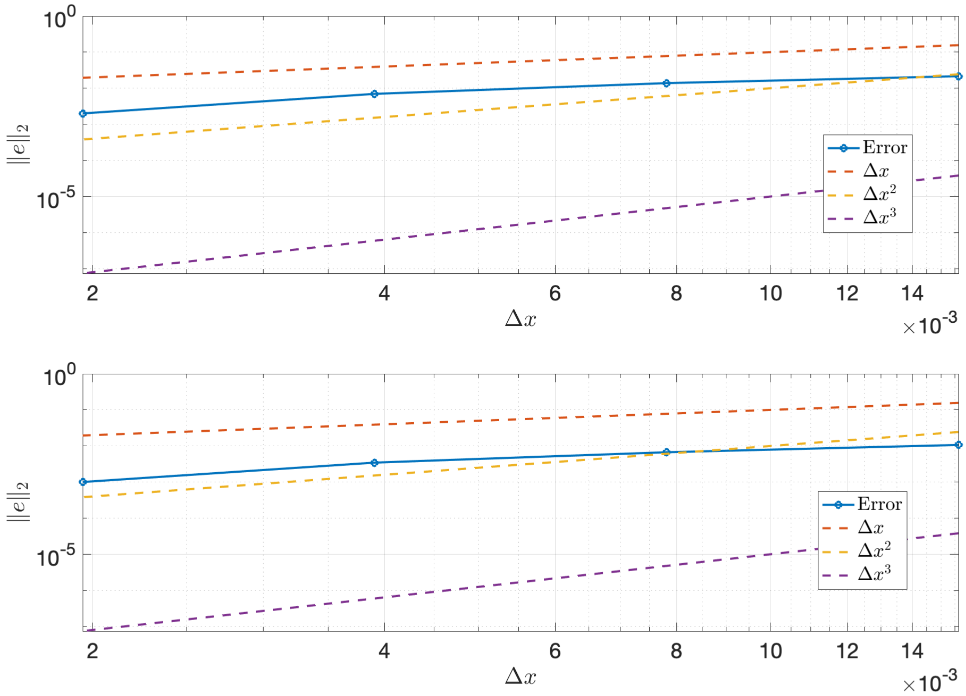

Figure 6: Log-log plot of the ℓ_2 norm of the error between reference and numerical solution at time $t = 1$ for $h(x, 1)$ (top) and $m(x, 1)$ (bottom), for the second set of initial conditions, shown in Eq. 8. Order 1, 2, and 3 convergence rates are also plotted, to ease the comparison.

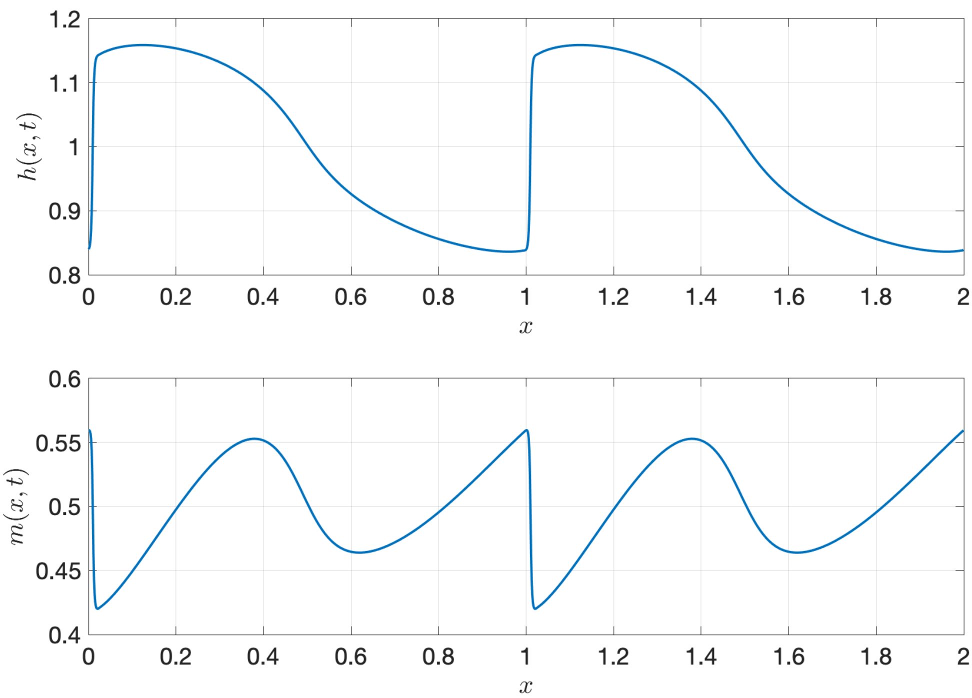

Figure 7: Numerical solution for $h(x, 1)$ (top) and $m(x, 1)$ (bottom) for the second set of initial conditions shown in Eq. 8, using a highly refined mesh featuring $N = 1500$ intervals.

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.

A MATLAB code

A.1 Part2_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";

%% Resolution of the problem

% First problem (for integration of source term in space)
PROBLEM = 1;

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

% Spatial domain
xspan = [0, 2];

% Temporal domain
tspan = [0, 1];

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

% Number of grid points
N = 500;

% CFL condition
CFL = 0.5;

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

% Choose order for WENO reconstruction
k = 2;

% Solve the problem
[h, m, xc, tvec] = solver(xspan, tspan, N, ...
    CFL, g, h0, m0, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

```

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

        subplot(2, 1, 1)
        plot(xc, h(:, i), 'LineWidth', 2)
        hold on
        plot(xc, h0(xc - 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([0.5 1.5]);
        hold off
        legend('Numerical solution', 'Exact solution', ...
            'Interpreter', 'latex', 'Location', 'best')
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xc, m(:, i), 'LineWidth', 2)
        hold on
        plot(xc, u * h0(xc - 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 0.4]);
        hold off
        legend('Numerical solution', 'Exact solution', ...
            'Interpreter', 'latex', 'Location', 'best')
        set(gca, 'Fontsize', 20)
        drawnow

    end
end



%%  Error analysis

% We solve the same problem for different values of \Delta x
delta_x_vec = 2.^-(6:9);

```

```
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);

    [h, m, xc, tvec] = solver(xspan, tspan, N, ...
        CFL, g, h0, m0, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

    err_h_vec(i) = 1/sqrt(N) * norm(h(:, end) - h0(xc-tspan(2))');
    err_m_vec(i) = 1/sqrt(N) * norm(m(:, end) - u*h0(xc-tspan(2))');
end


% Plot the error in loglog
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, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(h(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
    "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, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(m(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
    "interpreter", "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)
```

A.2 Part2_4

```
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 (first set of initial conditions)

% Second problem
PROBLEM = 2;

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

% Spatial domain
xspan = [0, 2];

% Temporal domain
tspan = [0, 1];

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

% Number of grid points
N = 500;

% Number of time steps
CFL = 0.5;

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

% Choose order for WENO reconstruction
k = 2;

% Solve the problem
[h1, m1, xc1, tvec1] = solver(xspan, tspan, N, ...
    CFL, g, h01, m01, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

% We visualize the solution
if animation == "True"

```

```
    figure(1)

    for i = 1 : 1 : length(tvec1)

        subplot(2, 1, 1)
        plot(xc1, 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
        xlim([0 2]);
        ylim([0.9 1.1]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xc1, 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
        xlim([0 2]);
        ylim([-0.1 0.1])
        set(gca, 'Fontsize', 20)
        drawnow

    end

end



%% Error analysis (first set of initial conditions)

% Generate a reference solution using sufficiently fine mesh
[h1_ex, m1_ex, xvec1_ex, ~] = solver(xspan, tspan, 1500, ...
    CFL, g, h01, m01, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

% We solve the same problem for different values of \Delta x
delta_x_vec =  2.^-(6:9);

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);

    [h1, m1, xc1, ~] = solver(xspan, tspan, N, ...
        CFL, g, h01, m01, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

    % 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 xc1
    h1ex_interp = interp1(xvec1_ex, h1_ex(:, end), xc1);
    m1ex_interp = interp1(xvec1_ex, m1_ex(:, end), xc1);

    % Compute norm 2 of the error
    err_h_vec1(i) =  1/sqrt(N) * norm(h1(:, end)' - h1ex_interp);
    err_m_vec1(i) =  1/sqrt(N) * norm(m1(:, end)' - m1ex_interp);

end


% Plot the error in loglog
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, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(h(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
    "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, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(m(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
    "interpreter", "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)
```

```matlab



%% Resolution of the problem (second set of initial conditions)

% Initial conditions
h02 = @(x) 1 - 0.2 * sin(2 * pi * x);
m02 = @(x) 0.5 + 0*x;

% Number of grid points
N = 500;

% Solve the problem
[h2, m2, xc2, tvec2] = solver(xspan, tspan, N, ...
    CFL, g, h02, m02, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

% We visualize the solution
if animation == "True"

    figure(3)

    for i = 1 : 1 : length(tvec2)

        subplot(2, 1, 1)
        plot(xc2, 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
        xlim([0 2]);
        ylim([0.8 1.2]);
        set(gca, 'Fontsize', 20)
        drawnow

        subplot(2, 1, 2)
        plot(xc2, 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
        xlim([0 2]);
        ylim([0.4 0.6])
        set(gca, 'Fontsize', 20)
        drawnow

    end

```

```
end



%%  Error analysis (second set of initial conditions)

% Generate a reference solution
[h2_ex, m2_ex, xvec2_ex, ~] = solver(xspan, tspan, 1500, ...
    CFL, g, h02, m02, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

% We solve the same problem for different values of \Delta x
delta_x_vec =  2.^-(6:9);

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);

    [h2, m2, xc2, ~] = solver(xspan, tspan, N, ...
        CFL, g, h02, m02, @LaxFriedrichs, @flux_phys, bc, k, PROBLEM);

    % 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 xc2
    h2ex_interp = interp1(xvec2_ex, h2_ex(:, end), xc2);
    m2ex_interp = interp1(xvec2_ex, m2_ex(:, end), xc2);

    err_h_vec2(i) =  1/sqrt(N) * norm(h2(:, end)' - h2ex_interp);
    err_m_vec2(i) =  1/sqrt(N) * norm(m2(:, end)' - m2ex_interp);

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, 10 * delta_x_vec, "--", "Linewidth", 2)
loglog(delta_x_vec, 100 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(h(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
```

```
    "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, 10 * delta_x_vec, "--", "Linewidth", 2)
loglog(delta_x_vec, 100 * delta_x_vec.^2, "--", "Linewidth", 2)
loglog(delta_x_vec, 10 * delta_x_vec.^3, "--", "Linewidth", 2)
xlabel('$\Delta x$', 'Interpreter', 'latex')
ylabel("$\|e\|_2$", "Interpreter","latex")
title("Error on \(m(x,t)\) at \(t=1\)", "Interpreter","latex")
legend("Error", "\(\Delta x\)", "\(\Delta x^2\)", "\(\Delta x^3\)", ...
    "interpreter", "latex", "location", "best")
grid on
set(gca, 'Fontsize', 20)
```

A.3 Integrate the problem in time

```
function [h, m, xc, tvec] = solver(xspan, tspan, N, CFL, g, h0, m0, ...
    flux, flux_phys, bc, k, PROBLEM)
% SOLVER - Solve a hyperbolic conservation law using the SSP-RK3 scheme.
%
%   [h, m, xc, tvec] = SOLVER(xspan, tspan, N, CFL, g, h0, m0, ...
%                             flux, flux_phys, bc, k, PROBLEM)
%   solves a hyperbolic conservation law using the SSP-RK3 scheme
%   and returns the solutions for water height 'h', discharge 'm',
%   spatial grid locations 'xc' and the time vector 'tvec'.
%
%   Input:
%       xspan      - Spatial domain [x_start, x_end].
%       tspan      - Time domain [t_start, t_end].
%       N          - Number of spatial grid points.
%       CFL        - Courant-Friedrichs-Lewy (CFL) number for adaptive
%                    time-stepping.
%       g          - Gravitational constant (g=1).
%       h0         - Function handle for the initial water height
%                    condition.
%       m0         - Function handle for the initial discharge condition.
%       flux       - Flux function for the conservative variables.
%       flux_phys  - Flux function in physical variables.
%       bc         - String specifying the type of boundary condition.
%                    Supported values: 'peri' (periodic), 'open'
%                    (open boundary).
%       k          - Order of accuracy for WENO reconstruction.
```

```
%          PROBLEM    - Integer specifying the problem type.
%                       Supported values: 1 or 2.
%
%    Output:
%          h          - Matrix of water height solutions over time.
%          m          - Matrix of discharge solutions over time.
%          xc         - Spatial grid locations.
%          tvec       - Time vector.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% January 2024


% Define preliminary variables
dx       = (xspan(2) - xspan(1)) / N;
xc       = (xspan(1) + 0.5 * dx) : dx : (xspan(2) - 0.5 * dx);
xf       = linspace(xspan(1), xspan(2), N + 1);
h        = zeros(N,1);
m        = zeros(N,1);
tvec     = 0;

% We compute cell-averages of the initial condition
for j = 1 : N
    h(j) = integral(h0, xf(j), xf(j+1), 'AbsTol', 1e-14) / dx;
    m(j) = integral(m0, xf(j), xf(j+1), 'AbsTol', 1e-14) / dx;
end

% Initialize polynomial coefficients (of degree k-1)
Crec = zeros(k + 1, k);
for r=-1:k-1
    Crec(r+2,:) = ReconstructWeights(k,r);
end

% Initialize linear weights
dw = LinearWeights(k,0);

% Compute smoothness indicator matrices
beta = zeros(k,k,k);
for r=0:k-1
    xl = -1/2 + [-r:1:k-r];
    beta(:,:,r+1) = betarcalc(xl,k);
end

% Store the initial condtion in q, to have size(q) = [2, Nspacenodes]
q = [h';m'];

% We can now start solving the problem
t = 0;

```

```matlab
while (t < tspan(2))

    vel = q(2, :) ./ q(1,:);

    % Adaptive time-step
    dt = dx * CFL / max(abs(vel) + sqrt(g * q(1,:)));

    % DEBUGGING
    if dt < 1e-5
        disp(h)
        disp(m)
        error("The solution is exploding")
    end

    if(t + dt >= tspan(2))
       dt = tspan(2) - t;
    end

    qold = q;

    % Define variables that will store the source terms
    Source1 = zeros(2, length(xf)-1);
    Source2 = zeros(2, length(xf)-1);
    Source3 = zeros(2, length(xf)-1);

    % Integrate once for all the source term and evaluate it at
       different
    % time steps for later use from SSP-RK3
    if PROBLEM == 1

        Source1 = integrate_source(xf(1:end-1), xf(2:end), ...
            t, PROBLEM);
        Source2 = integrate_source(xf(1:end-1), xf(2:end), ...
            t + dt, PROBLEM);
        Source3 = integrate_source(xf(1:end-1), xf(2:end), ...
            t + 0.5 * dt, PROBLEM);

    end


    % We use Runge-Kutta Strong Stability Preserving scheme to
       integrate
    % in time (SSP-RK3) - see exercise sheet for algorithm

    % SSP-RK3 Stage1
    RHS = evalRHS(q, N, dt, dx, flux, flux_phys, bc, Crec, ...
        k, xf, dw, beta);
    q1 = qold + dt/dx * (RHS + Source1);

```

```
    % SSP-RK3 Stage2
    RHS = evalRHS(q1, N, dt, dx, flux, flux_phys, bc, Crec, ...
        k, xf, dw, beta);
    q2 = 3*qold/4.0 + (q1 + dt/dx * (RHS + Source2))/4.0;

    % SSP-RK3 Stage3
    RHS = evalRHS(q2, N, dt, dx, flux, flux_phys, bc, Crec, ...
        k, xf, dw, beta);
    q3 = qold/3.0 + 2.0*(q2 + dt/dx * (RHS + Source3))/3.0;

    q = q3;

    t = t + dt;

    % Store the new solution
    h = [h, q(1, :)'];
    m = [m, q(2, :)'];
    tvec = [tvec, t];

end
```

A.4 Evaluate RHS

```
function RHS = evalRHS(U, N, dt, dx, flux, flux_phys, bc, Crec, ...
    k, xf, dw, beta)

% EVALRHS - Evaluate the right-hand side (RHS) of a numerical scheme
   for a
%          hyperbolic conservation law.
%
%  RHS = EVALRHS(U, N, dt, dx, flux, flux_phys, bc, Crec, k, xf, dw,
   beta)
%  computes the RHS of a hyperbolic conservation law based on the
   given
%  solution vector 'U', numerical parameters, and flux functions.
%
%  Input:
%      U           - Solution matrix with two components.
%      N           - Number of grid points in the spatial domain.
%      dt          - Time step size.
%      dx          - Spatial grid spacing.
%      flux        - Flux function for the conservative variables.
%      flux_phys   - Physical flux.
%      bc          - String specifying the type of boundary condition.
%                    Supported values: 'peri' (periodic), 'open'
%                    (open boundary).
%      Crec        - Reconstruction method coefficient
%                    (for WENO reconstruction).
%      k           - Order of accuracy for WENO reconstruction.
```

```
%         xf           - Spatial grid locations.
%         dw           - WENO weights.
%         beta         - Parameter for WENO reconstruction.
%
%     Output:
%         RHS          - Computed right-hand side vector for
%                        the conservation law.
%
%     Note:
%         This function applies appropriate boundary conditions,
%         performs WENO reconstruction, and computes the flux differences
%         to obtain the RHS.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% January 2024

% Apply appropriate boundary conditions
U = apply_bc(U, bc, k);

% Obtain reconstructed states
hl = zeros(1, N+2);
hr = zeros(1, N+2);

ml = zeros(1, N+2);
mr = zeros(1, N+2);

for i = 1:N+2
    [hl(i), hr(i)] = WENO(xf, U(1, i:(i+2*(k-1)))', k, Crec, dw, beta);
    [ml(i), mr(i)] = WENO(xf, U(2, i:(i+2*(k-1)))', k, Crec, dw, beta);
end

qr = [hr(2:N+1); mr(2:N+1)]; ql = [hl(2:N+1); ml(2:N+1)];
qm = [hr(1:N); mr(1:N)]; qp = [hl(3:N+2); ml(3:N+2)];

fluxval1 = flux(flux_phys, qr, qp, dx, dt);
fluxval2 = flux(flux_phys, qm, ql, dx, dt);

RHS = - (fluxval1-fluxval2);

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]
% January 2024

g = 1;
h = q(1, :);
m = q(2, :);

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

return
```

A.6 Numerical flux

```
function fval = LaxFriedrichs(flux_phys, Ul, Ur, dx, dt)
% LAXFRIEDRICHS - Compute the Lax-Friedrichs flux at a cell interface.
%
%   fval = LAXFRIEDRICHS(flux_phys, Ul, Ur, dx, dt) computes the
%          Lax-Friedrichs flux at a cell interface based on the
%          physical flux function, left-state 'Ul', right-state 'Ur',
%          spatial grid spacing 'dx', and time step size 'dt'.
%
%   Input:
%       flux_phys - Physical flux function.
%       Ul        - Left-state values.
%       Ur        - Right-state values.
%       dx        - Spatial grid spacing.
%       dt        - Time step size.
%
%   Output:
%       fval      - Computed Lax-Friedrichs flux at the cell interface.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% January 2024
```

```


fval = 0.5 * (flux_phys(Ul) + flux_phys(Ur)) - 0.5 * dx/ dt * (Ur - Ul)
    ;

return;
```

A.7 Apply boundary conditions

```
function U = apply_bc(Ui, bc, m)
% APPLY_BC - Apply boundary conditions to a given solution matrix.
%
%   U = APPLY_BC(Ui, bc, m) applies boundary conditions to the input
    matrix
%   UI based on the specified boundary condition type 'BC' and the
    number of
%   ghost points 'M'. The function returns the modified matrix 'U' with
%   added ghost points.
%
%   Input:
%       Ui    - Input matrix without boundary conditions.
%       bc    - String specifying the type of boundary condition.
%               Supported values: 'peri' (periodic), 'open' (open
    boundary).
%       m     - Number of ghost points to be added on each side.
%
%   Output:
%       U     - Matrix with applied boundary conditions.
%
% Function available on Moodle for MATH-459 course
% January 2024


switch bc

    case 'peri'

        U = [Ui(:, end-m+1 : end), Ui, Ui(:,1:m)];

    case 'open'

        U = [repmat(Ui(:,1), 1, m), Ui, repmat(Ui(:,end),1,m)];

end

end
```

A.8 Integrate source term in space exactly

```
function intS = integrate_source(xa, xb, t, PROBLEM)

% INTEGRATE_SOURCE - Compute the spatial integral of the source term
    for
%                    a given hyperbolic problem at a time t, using the
%                    exact solution of the integral.
%
%   intS = INTEGRATE_SOURCE(xa, xb, t, PROBLEM) computes the spatial
%   integral of the source term for a specified problem at a given time
%   't'.
%
%   Input:
%       xa      - Left spatial boundary.
%       xb      - Right spatial boundary.
%       t       - Time at which the source term is evaluated.
%       PROBLEM - Integer specifying the problem type.
%                 Supported values: 1 or 2.
%
%   Output:
%       intS    - Computed spatial integral of the source term.
%
%   Parameters:
%       u       - Constant parameter (problem-dependent).
%       g       - Constant parameter (problem-dependent).
%
%   Problem Descriptions:
%       PROBLEM = 1: Computes the integral for the source term.
%       PROBLEM = 2: Returns zero for the source term.
%
% Authors: [Francesco Sala, Nicolo' Viscusi]
% January 2024

u = 0.25;
g = 1;

if PROBLEM == 1

    int_x_S = @(xa,xb) [0.5*(u-1).*(sin(pi*(t-xa))-sin(pi*(t-xb)));
        1/8*sin(pi*(t-xb)).*(g*sin(pi*(t-xb)) -4*(g+(u-1)*u))- ...
        1/8*sin(pi*(t-xa)).*(g*sin(pi*(t-xa)) -4*(g+(u-1)*u))];
    intS = int_x_S(xa,xb);

elseif PROBLEM == 2

    int_x_S = @(xa,xb) [0*xa; 0*xa];
    intS = int_x_S(xa, xb);

end
```

```
return
```

A.9 Functions by Jan H. Hesthaven

A.9.1 WENO reconstruction

```
function [um,up] = WENO(xloc,uloc,m,Crec,dw,beta);
% Purpose: Compute the left and right cell interface values using an
   WENO
%           approach based on 2m-1 long vectors uloc with cell
% Set WENO parameters
% Function implemented by Jan S. Hesthaven

p = 1; q = m-1; vareps = 1e-6;

% Treat special case of m=1 - no stencil to select
if (m==1)
    um = uloc(1); up = uloc(1);
else
    alpham = zeros(m,1); alphap = zeros(m,1);
    upl = zeros(m,1); uml = zeros(m,1); betar = zeros(m,1);

    % Compute um and up based on different stencils and
    % smoothness indicators and alpha
    for r=0:m-1;
        umh = uloc(m-r+[0:m-1]);
        upl(r+1) = Crec(r+2,:)*umh; uml(r+1) = Crec(r+1,:)*umh;
        betar(r+1) = umh'*beta(:,:,r+1)*umh;
    end;

    % Compute alpha weights - classic WENO
    alphap = dw./(vareps+betar).^(2*p);
    alpham = flipud(dw)./(vareps+betar).^(2*p);

%    % Compute alpha weights - WENO-Z
%      tau = abs(betar(1) - betar(m));
%      if mod(m,2)==0
%          tau = abs(betar(1)-betar(2) - betar(m-1) + betar(m));
%      end
%      alphap = dw.*(1 + (tau./(vareps+betar)).^q);
%      alpham = flipud(dw).*(1 + (tau./(vareps+betar)).^q);

    % Compute nonlinear weights and cell interface values
    um=alpham'*uml/sum(alpham); up=alphap'*upl/sum(alphap);
end
return
```

A.9.2 Reconstruction weights

```
function [c] = ReconstructWeights(m,r)
% Purpose: Compute weights c_ir for reconstruction
%              v_{j+1/2} = \sum_{j=0}^{m-1} c_{ir} v_{i-r+j}
%       with m=order and r=shift (-1<=r<=m-1).
% Function implemented by Jan S. Hesthaven

c = zeros(1,m); fh = @(s) (-1)^(s+m)*prod(1:s)*prod(1:(m-s));
for i=0:m-1
    q = linspace(i+1,m,m-i);
    for q=(i+1):m
        if (q~=r+1)
           c(i+1) = c(i+1) + fh(r+1)/fh(q)/(r+1-q);
        else
           c(i+1) = c(i+1) - (harmonic(m-r-1)-harmonic(r+1));
        end
    end
end
end
```

A.9.3 Qcalc

```
function [Qelem] = Qcalc(D,m,l);
% Purpose: Evaluate entries in the smoothness indicator for WENO
% Function implemented by Jan S. Hesthaven

[x,w] = LegendreGQ(m); xq = x./2; Qelem = 0;
for i=1:m+1
    xvec = zeros(m-l+1,1);
    for k=0:m-l  xvec(k+1) = xq(i)^k./prod(1:k); end;
    Qelem = Qelem + (xvec'*D*xvec)*w(i)/2;
end
return
```

A.9.4 betarcalc

```
function [errmat] = betarcalc(x,m)
% Purpose: Compute matrix to allow evaluation of smoothness indicator
   in
%          WENO based on stencil [x] of length m+1 .
%          Returns sum of operators for l=1.m-1
% Function by Jan H. Hesthaven

% Evaluate Lagrange polynomials
[cw] = lagrangeweights(x);

% Compute error matrix for l=1..m-1
errmat = zeros(m,m);
for l=2:m
```

```
    % Evaluate coefficients for derivative of Lagrange polynomial
    dw = zeros(m,m-l+1);
    for k=0:(m-l)
      for q=0:m-1
        dw(q+1,k+1) = sum(cw((q+2):m+1,k+l+1));
      end
    end

    % Evaluate entries in matrix for order 'l'
    Qmat = zeros(m,m);
    for p=0:m-1
      for q=0:m-1
         D = dw(q+1,:)'*dw(p+1,:);
         Qmat(p+1,q+1) = Qcalc(D,m,l);
      end
    end
    errmat = errmat + Qmat;
end
return
```

A.9.5 betarcalc

```
function [errmat] = betarcalc(x,m)
% Purpose: Compute matrix to allow evaluation of smoothness indicator
    in
%           WENO based on stencil [x] of length m+1 .
%           Returns sum of operators for l=1.m-1
% Function by Jan H. Hesthaven

% Evaluate Lagrange polynomials
[cw] = lagrangeweights(x);

% Compute error matrix for l=1..m-1
errmat = zeros(m,m);
for l=2:m
    % Evaluate coefficients for derivative of Lagrange polynomial
    dw = zeros(m,m-l+1);
    for k=0:(m-l)
      for q=0:m-1
        dw(q+1,k+1) = sum(cw((q+2):m+1,k+l+1));
      end
    end

    % Evaluate entries in matrix for order 'l'
    Qmat = zeros(m,m);
    for p=0:m-1
      for q=0:m-1
         D = dw(q+1,:)'*dw(p+1,:);
         Qmat(p+1,q+1) = Qcalc(D,m,l);
```

```
      end
    end
    errmat = errmat + Qmat;
end
return
```

A.9.6 LegendreGQ

```
function [x,w] = LegendreGQ(m);
% function [x,w] = LegendreGQ(m)
% Purpose: Compute the m'th order Legendre Gauss quadrature points, x,
%          and weights, w
% Function implemented by Jan S. Hesthaven

if (m==0) x(1)=0; w(1) = 2; return; end;

% Form symmetric matrix from recurrence.
J = zeros(m+1); h1 = 2*(0:m);
J = diag(2./(h1(1:m)+2).*...
     sqrt((1:m).*((1:m)).*((1:m)).*((1:m))./(h1(1:m)+1)./(h1(1:m)+3))
        ,1);
J(1,1)=0; J = J + J';

%Compute quadrature by eigenvalue solve
[V,D] = eig(J); x = diag(D); w = 2*(V(1,:)').^2;
return
```

A.9.7 LinearWeights

```
function [d]=LinearWeights(m,r0)
% Purpose: Compute linear weights for maximum accuracy 2m-1,
% using stencil shifted $r_0=-1,0$ points upwind.
% Function implemented by Jan S. Hesthaven

A = zeros(m,m); b = zeros(m,1);

% Setup linear system for coefficients
for i=1:m
  col = ReconstructWeights(m,i-1+r0);
  A(1:(m+1-i),i) = col(i:m)';
end

% Setup righthand side for maximum accuracy and solve
crhs = ReconstructWeights(2*m-1,m-1+r0);
b = crhs(m:(2*m-1))'; d = A\b;
return
```

A.9.8 Lagrange weights

```
function [cw] = lagrangeweights(x)
% Purpose: Compute weights for Taylor expansion of Lagrange polynomial
%          based on x and evaluated at 0.
%          Method due to Fornberg (SIAM Review, 1998, 685-691)
% Function implemented by Jan S. Hesthaven

np = length(x); cw=zeros(np,np);
cw(1,1)=1.0; c1 = 1.0; c4 = x(1);
for i=2:np
    mn = min(i,np-1)+1;
    c2 = 1.0; c5 = c4; c4 = x(i);
    for j=1:i-1
      c3 = x(i)-x(j); c2 = c2*c3;
      if (j==i-1)
          for k=mn:-1:2
              cw(i,k) = c1*((k-1)*cw(i-1,k-1)-c5*cw(i-1,k))/c2;
          end
          cw(i,1) = -c1*c5*cw(i-1,1)/c2;
      end
      for k=mn:-1:2
          cw(j,k) = (c4*cw(j,k)-(k-1)*cw(j,k-1))/c3;
      end
      cw(j,1) = c4*cw(j,1)/c3;
    end
    c1=c2;
end
return
```