MATH-459 – Numerical Methods for Conservation Laws Project 1. The Finite Volume Method

Benoît Müller

October 4, 2023

Remarks on the code

The code contains 9 files: `f.m, make_prob.m, G_LF.m, G_Roe.m, shallow_water.m, problem1.m, problem2.m, problem3.m,` and `problem4.m`.

The organization of the code is the following: the central function is `shallow_water.m`, it contain the loops that the methods make. The parameters are three structures that encapsulate the problem, the numerical method, and the display options (resp. `prob, meth, show`). We use `make_problem.m` to make the structure `prob` that `shallow_water.m` will use. The functions `G_LF.m` and `G_Roe.m` define the methods with to the flow `f.m`, and will be used by `shallow_water.m` by passing them in the parameter `meth`. Details are given in the rest of this document. The files `problem"x".m` are the runable scripts, they have the name of the problem and have cells that can be runned independently.

Problem 1

(a)

We set $\Omega = [a,b] = [0,2]$, $\Delta x = (b-a)/N$, $x_j = a + jh, \quad \forall j \in \{0,\dots,N\}$ and the midpoints $x_{j+1/2} = a + (j+1/2)\Delta x, \quad \forall j \in \{-1,\dots,N\}$. The equation can be writen

$$\mathbf{q}_t + f(\mathbf{q})_x = S$$

. The time dimension will be discretised by times $0 = t^0 < t^n < \cdots < t^{n_{\max}} = T$ and the values of the step are computed during the process. We discretize the derivatives with finite differences. The time derivative is aproximated by a variation of a forward finite difference:

$$\frac{Q_j^{n+1} - \frac{1}{2}(Q_{j-1}^n + Q_{j-1}^n)}{\Delta t}$$

, and the space derivative by a centered finite difference:

$$\frac{f(Q_{j+1}^n) - f(Q_{j-1}^n)}{2\Delta x}.$$

Putting this together and by approximating $S(x_j, t^n)$ by a S_j^n, it gives

$$\frac{Q_j^{n+1} - \frac{1}{2}(Q_{j-1}^n + Q_{j-1}^n)}{\Delta t} + \frac{f(Q_{j+1}^n) - f(Q_{j-1}^n)}{\Delta x} = S_j^n,$$

and explicitely

$$Q_j^{n+1} = \frac{1}{2}(Q_{j-1}^n + Q_{j-1}^n) - \frac{\Delta t}{2\Delta x}(f(Q_{j+1}^n) - f(Q_{j-1}^n)),$$

or in a conservative form

$$Q_j^{n+1} = Q_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n),$$

with $\lambda = \frac{\Delta x}{\Delta t}$ and $F_{j+1/2}^n = \frac{1}{2}(f(Q_j^n) - f(Q_{j+1}^n)) - \frac{1}{2\lambda}(Q_{j+1}^n + Q_j^n)$, where we recognise the Lax-Friedrichs flux. We will keep the first explicit description for the code because it is more efficient to calculate than the conservative form which add and substrate the term Q_j^n. The value of Δt must be small enough for the scheme to be monotone and it is given by the CFL condition: $k <$

$\Delta x/\max\|f'\|$. A computation of the jacobian of f and it's matrix norm by spectral decomposition show that $\|f'((q))\| = max(|u \pm \sqrt{gh}|) < |u| + \sqrt{gh}$, so we discretise it to find the value of the max and multiply by CFL:

$$\Delta t = k = \mathrm{CFL}\frac{\Delta x}{\max_j(|u_j^n| + \sqrt{gh_j^n})},$$

with $Q_j^n = (h_j^n, m_j^n)^\top$ and $u_j^n = m_j^n/h_j^n$.

(b)

The function f is implemented:

```
function F=f(h,m)
% the function flux of the equation
    g=1;
    F= [m; m.^2./h + 0.5*g*h.^2];
end
```

and the problem is defined in a structure prob by the function `make_prob`(we will need the problem multiple times):

```
function prob = make_prob(question)
% build the structure prob that describe a problem
% prob contain the initial condition q_0,
% (sometimes) the true solution q_true, the source S, the limit time T,
% and the type of boundary condition bound.
    periodic_bound = @(Q) [Q(:,end), Q, Q(:,1)];
    open_bound     = @(Q) [Q(:,1), Q, Q(:,end)];
    switch question
        case "1a"
            u=0.25;
            g=1;
            h0 = @(x) 1 + 0.5*sin(pi*x);
            m0 = @(x) u*h0(x);
            h_true = @(x,t) h0(x-t);
            m_true = @(x,t) u*h_true(x,t);

            prob.q0 = @(x) [h0(x); m0(x)];
            prob.q_true = @(x,t) [h_true(x,t); m_true(x,t)];
            prob.S= @(x,t) pi/2*cos(pi*(x-t)) .* ...
                [(u-1)*ones(1,length(x)) ; u^2-u+g*h0(x-t)];
            prob.T=2;
            prob.bound=periodic_bound;
```

We implement the scheme step in the function `G_LF`, where we ask the structure meth to contain `Q` the current $(Q_j^n)_j$, `t` the current time, `x` the discretized points of cell size $\Delta x =$`dx`

```
function meth = G_LF(prob, meth)
% lax-Friedrichs scheme
% return the structure meth with Q and t updated
    g=1;
    CFL=0.5;
    Qext = prob.bound(meth.Q);
    int=2:(meth.N+2);
    uu= meth.Q(2,:)./meth.Q(1,:);
    gradnorm = max( abs(uu) + sqrt(g*meth.Q(1,:)));
    k = CFL*meth.dx / gradnorm;
    if meth.t+k<=prob.T   % it is not exceeding T or we reached it already
        lambda = CFL / gradnorm; %=k/dx;
    else % it is not exceeding T
        k= prob.T-meth.t;
        lambda= k/meth.dx;
    end
    flux= f(Qext(1,:),Qext(2,:));
    SS= prob.S(meth.x,meth.t+k/2);
    meth.Q= 0.5*(Qext(:,int+1) + Qext(:,int-1))...
        -0.5*lambda*(flux(:,int+1) - flux(:,int-1)) + k*SS;
    meth.t= meth.t + k;
end
```

We use them now in the function `shallow_water` implemented in the following code, where we ask the structure `show` to specify if we want to plot the evolution or not (`show.yes`) and with which frequency (`show.freq`). We return the data in a copy of the structure `meth` in which we added the results obtained.

```
function meth = shallow_water(prob, meth, show)
% compute the numerical solution to the water-shallow equation
% defined by prob with the method defined by meth and display with the
% conditions of cond.
% prob must contain: q_0, S, T, q_true(optional)
% meth must contain: N, G
% show must contain: yes, freq(if yes=1)
% return meth with Q at t=T
    a=0;
    b=2;
    meth.dx=(b-a)/meth.N;
    meth.x=a:meth.dx:b;
    meth.t=0;
    meth.Q=prob.q0(meth.x);
    iter = 0;
    if show.yes==1
        plot_every= meth.N / show.freq;
    end
    while meth.t<prob.T - 1e-10
            meth = meth.G(prob, meth);
        if show.yes==1 && (mod(iter,plot_every)==0||abs(meth.t-prob.T)<1e-10)
            figure(1)
            title("test")
            subplot(2,1,1)
            area(meth.x,meth.Q(1,:),'LineWidth',2);
            xlim([0 2]); ylim([0 1.5]); %[0 2]
            if  isfield(prob,"q_true")
                Q_true= prob.q_true(meth.x,meth.t);
                hold on
                plot(meth.x,Q_true(1,:),'LineWidth',2)
                legend('Numerical h','Exact h');
                hold off
            end
            grid on;
            title(['Time = ',num2str(meth.t)]);
            xlabel("x")
            ylabel("h(x)")

            subplot(2,1,2)
            plot(meth.x, meth.Q(2,:),'LineWidth',2);
            xlim([0 2]);  ylim([-1 0.5]); %[-1 1]
            if isfield(prob,"q_true")
                hold on
                plot(meth.x,Q_true(2,:),'LineWidth',2);
                legend('Numerical m','Exact m');
                hold off
            end
            grid on;
            title(['Time = ',num2str(meth.t)]);
            xlabel("x")
            ylabel("m(x)")
        end
        iter=iter+1;
    end
end
```

We have now all the tools that we need and write a script to test the code. We first show the evolution of the solution for a chosen Δx and then show the error in a log-plot when we increase it. The error is computed with the norm

$$\|Q^n\|_{\Delta x,1} = \Big\| \|(h^n, m^n)\|_2 \Big\|_{\Delta x,1} = \|\sqrt{(h^n)^2 + (m^n)^2}\|_{\Delta x,1} = \Delta x \sum_j \sqrt{(h_j^n)^2 + (m_j^n)^2},$$

and we compute it for the quantity $(Q_j^n - \mathbf{q}(x_j, t^n))_j$. Namely, we compose the L^1 norm on the sequences with the euclidean norm on the $\mathbb{R}^2$ space.

```
%% Problem 1.a): display the evolution
clear
prob1a = make_prob("1a");
meth.N=500;
meth.G= @(prob,meth) G_LF(prob,meth);
show.yes=1;
show.freq=40;

meth=shallow_water(prob1a, meth, show);

%% Problem 1.b): display the error
clear
prob1a = make_prob("1a");
meth.G= @(prob,meth) G_LF(prob,meth);
show.yes=0;
NN=floor(logspace(2,3.5,10)); %mets (2,3,10)
error=[];
for N=NN
    meth.N=N;
    meth= shallow_water(prob1a, meth, show);
    Q_true= prob1a.q_true(meth.x,prob1a.T);
    norm=sqrt(sum((Q_true - meth.Q).^2,1)); % ||.||_2 norm for each point
    norm= meth.dx*sum(norm); % ||.||_1,h norm of all points
    error=[error, norm];
end
loglog(2./NN,error,'.-')
slope=log(error(end)/error(1))/log(NN(1)/NN(end));
title("\bf Problem 1.b): Error of the scheme at $T=2$" + ...
    " as a function of $\Delta x$, the slope is "+num2str(slope), 'interpreter','latex')
xlabel("$\bf \Delta x$",'interpreter','latex')
ylabel("$\bf Error$",'interpreter','latex')
```

We obtain the following results:

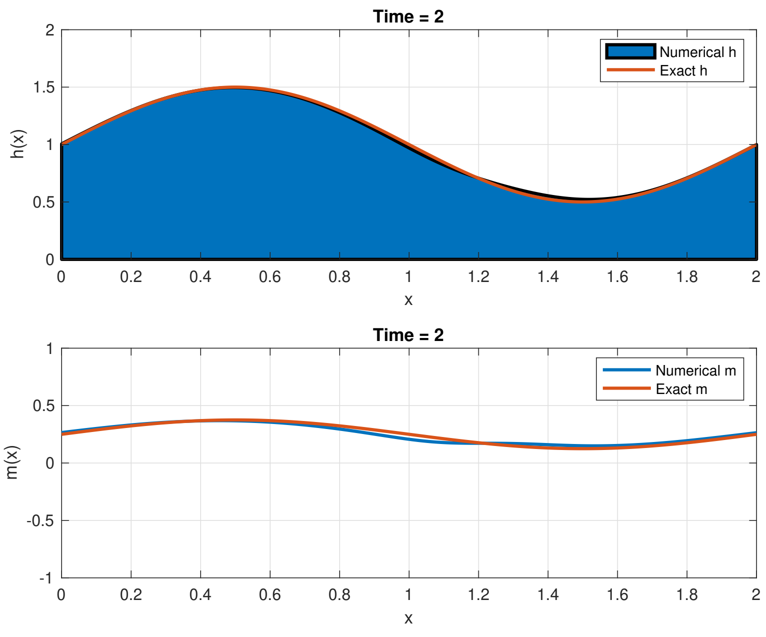


Figure 1: Plot of the solution at T=2 with the Lax-Friedrichs method.

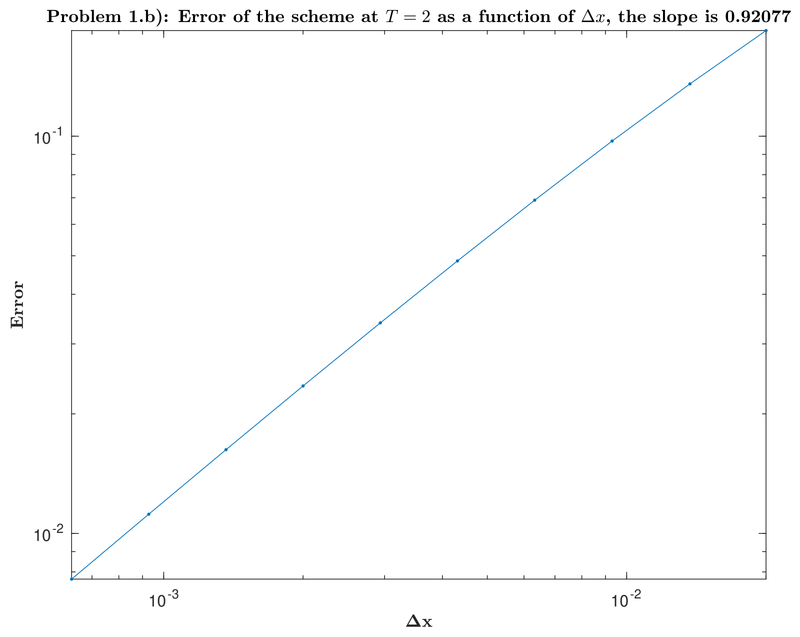


Figure 2: Plot of the solution at T=2 with the Lax-Friedrichs method.

and see that the error converge, with order one.

Problem 2

(a)

We continue the function `make_prob` for these two new conditions:

```
        case "2a5"
            h0 = @(x) 1 - 0.1*sin(pi*x);
            m0 = @(x) zeros(size(x));
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=2;
            prob.bound=periodic_bound;
        case "2a6"
            h0 = @(x) 1 - 0.2*sin(2*pi*x);
            m0 = @(x) 0.5*ones(size(x));
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=2;
            prob.bound=periodic_bound;
```

and write the script file that will display the evolution of the solutions at the final state at $T = 2$.

```
%% Problem 2.a) display the evolution
clear
prob2a=make_prob("2a6"); %2a5 or 2a6
meth.G= @(prob,meth) G_LF(prob,meth);
meth.N=10000;
show.yes=1;
show.freq=10;
Q6=shallow_water(prob2a, meth, show);
```

We see that for conditions (5), the function that was smooth stay smooth, and all along the process (see the animated plot):

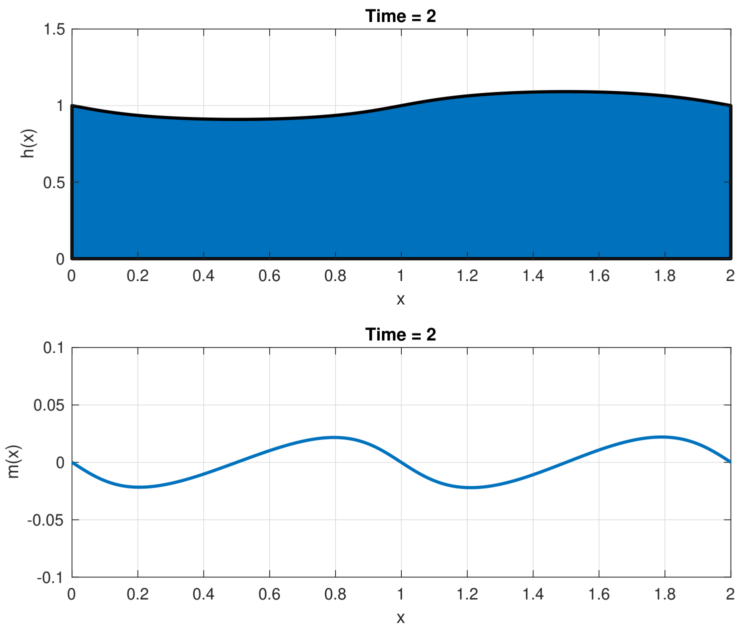


Figure 3: Plot of the solution at T=2 with the Lax-Friedrichs method and conditions (5.

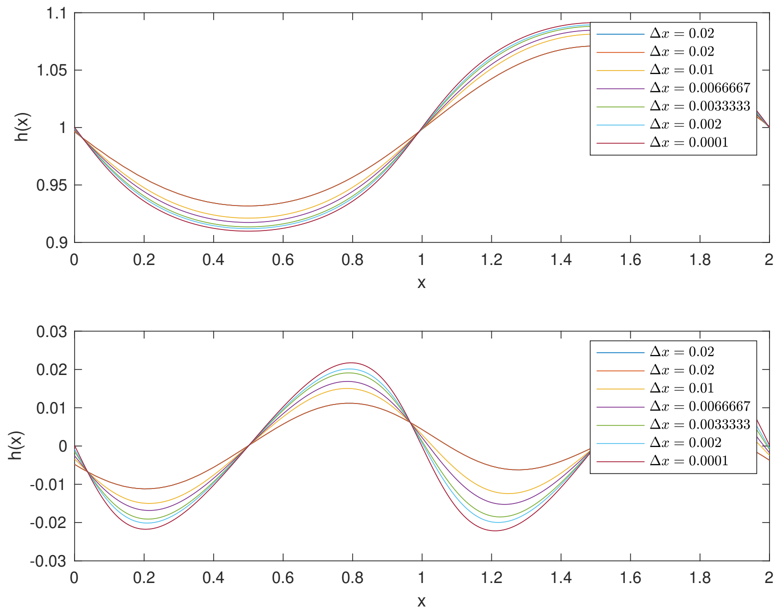


Figure 4: Plot of the solution at T=2 with the Lax-Friedrichs method for various Δx.

However, for the conditions (6), the solution become more and more discontinuous and we see chock waves appear:

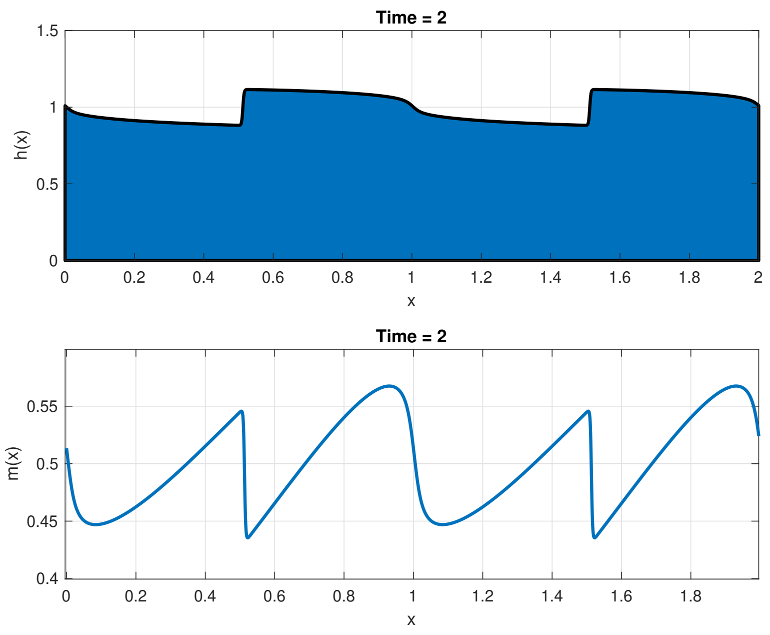


Figure 5: Plot of the solution at T=2 with the Lax-Friedrichs method.

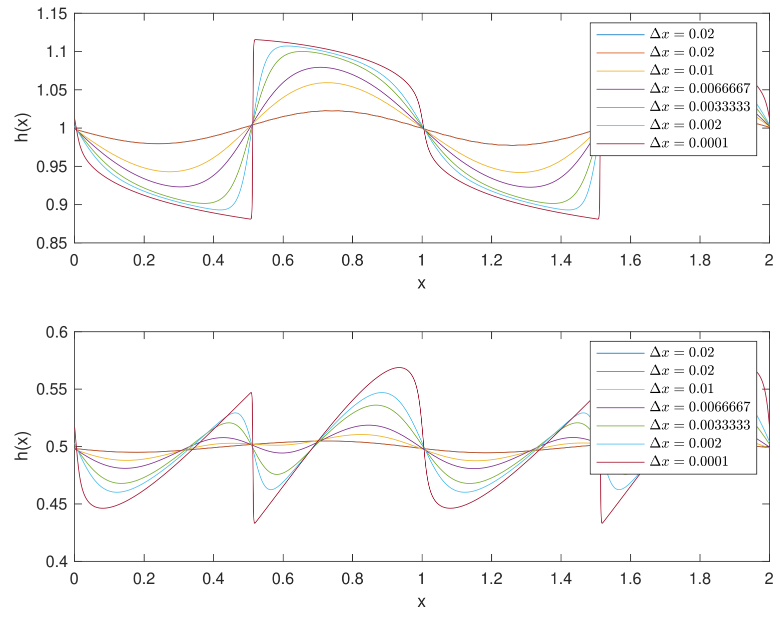


Figure 6: Plot of the solution at T=2 with the Lax-Friedrichs method for various Δx.

(b)

we write a similar code as problem 1.b) using a reference solution obtained with a fine mesh. To compare solutions with different meshes of N cells, we choose the N so they are multiple of the smallest one. Like so, all points of the less fine mesh are on the other too. Indeed if $M = cN$, then the points of the N-mesh $x(N)_j = a + j(b-a)/N = a + cj(b-a)/M = x(M)_{cj}$ are points of the M-mesh. We take the same error as before and we understand the substraction $(Q(N) - Q(M))j$ as

$Q(N)_j - Q(M)_{cj}$, so that the error is

$$\Delta x(N) \sum_j \|Q(N)_j^n - Q(M)_{cj}^n\|_2 = \frac{b-a}{N} \sum_j \|Q(N)_j^n - Q(M)_{cj}^n\|_2$$

We obtain the following plot, and see that for condtitions (5), the error converge, it seems with order one.

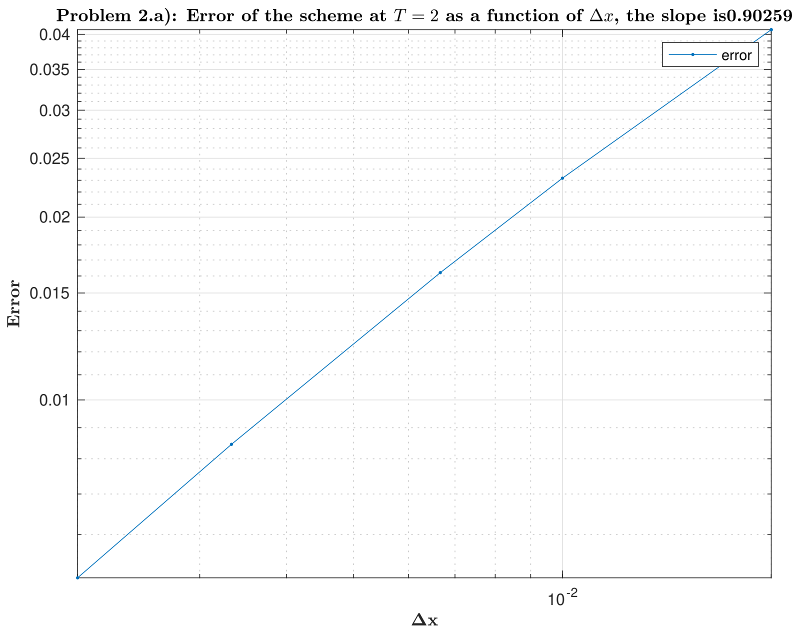


Figure 7: Plot of the error with the Lax-Friedrichs method for various Δx with conditions (5).

For the conditions (6), the convergence has an order smaller, however we see that we don't have a perfect straight line and that the slope get bigger and bigger as we decrease Δx. Maybe this is because we compare with a numerical solution and that we have shocks that the error isn't of order one as the theory tell us.

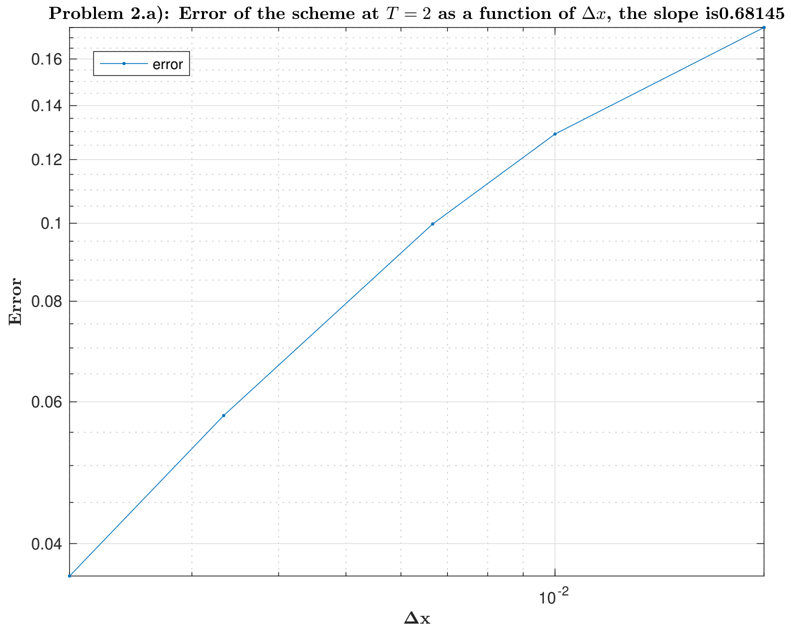


Figure 8: Plot of the error with the Lax-Friedrichs method for various Δx with conditions (6).

Problem 3

(a)

Godunov order barrier theorem tel us that schemes that are linear and monotonicity preserving are at most first order; monotone schemes are at most first order too. As a result, we must search for a more complex method to find a faster convergence. Godunov scheme is a candidate to do so. The basic scheme structure doesn't approximate any quantity and is exact. We have the freedom to compute or approximate the Riemann problem, as well as the integrals that we need. Depending on the quality of our approximations, one can hope having a better convergence than order one.

(b)

The Roe matrix $\hat{A}$ must satisfy $\hat{A}(\mathbf{q}_r-\mathbf{q}_l)=\mathbf{f}_r-\mathbf{f}_l$. We note generally $\Delta\phi=\phi_r-\phi_l$ and $\bar{\phi}=(\phi_r+\phi_l)/2$ for any ϕ. Suppose we introduce a parameter vector $\mathbf{z}$, such that the the jump in q and f can be expressed in terms of the jump in $\mathbf{z}$: $\Delta\mathbf{q}=B\Delta\mathbf{z}$, $\Delta\mathbf{f}=C\Delta\mathbf{z}$. Then clearly we can choose $\hat{A}=CB^{-1}$. We try with $\mathbf{z}=h^{-1/2}\mathbf{q}=(h/\sqrt{h},m/\sqrt{h})$. We first see that $(h,m)=(z_1^2,z_1z_2)$. Now, using $\Delta(ab)=\bar{a}\Delta b+\bar{b}\Delta a$,

$$\Delta\mathbf{q}=(\Delta h,\Delta m)=(2\bar{z}_1\Delta z_1,\bar{z}_1\Delta z_2+\bar{z}_2\Delta z_1)=\begin{pmatrix}2\bar{z}_1 & 0\\ \bar{z}_2 & \bar{z}_1\end{pmatrix}\Delta\mathbf{z}=:B\Delta\mathbf{z}$$

Similarily, we see $\mathbf{f}=(m,m^2/h+gh^2/2)=(z_1z_2,z_2^2+gz_1^2/2)$, and again

$$\Delta\mathbf{f}=(\bar{z}_1\Delta z_2+\bar{z}_2\Delta z_1,2\bar{z}_2\Delta z_2+g\bar{z}_1\Delta z_1)=\begin{pmatrix}\bar{z}_2 & \bar{z}_1\\ g\bar{z}_1 & 2\bar{z}_2\end{pmatrix}\Delta\mathbf{z}=:C\Delta\mathbf{z}.$$

We can then compute

$$CB^{-1}=\begin{pmatrix}\bar{z}_2 & \bar{z}_1\\ g\bar{z}_1 & 2\bar{z}_2\end{pmatrix}\begin{pmatrix}2\bar{z}_1 & 0\\ \bar{z}_2 & \bar{z}_1\end{pmatrix}^{-1}=\begin{pmatrix}\bar{z}_2 & \bar{z}_1\\ g\bar{z}_1 & 2\bar{z}_2\end{pmatrix}\begin{pmatrix}\bar{z}_1 & 0\\ -\bar{z}_2 & 2\bar{z}_1\end{pmatrix}(2\bar{z}_1{}^2)^{-1}=\begin{pmatrix}0 & 1\\ g\bar{z}_1^2-(\bar{z}_2/\bar{z}_1)^2 & 2\bar{z}_2/\bar{z}_1\end{pmatrix}=:\hat{A}.$$

The Jacobian flux is

$$f'(\mathbf{q})=\begin{pmatrix}0 & 1\\ -m^2/h^2+gh & 2m/h\end{pmatrix}$$

and we see that since $z_1^2=h$ and $z_2/z_1=m/h$, if $\mathbf{q}_l,\mathbf{q}_r\to\mathbf{q}$, so does $\hat{A}\to f'(\mathbf{q})$, so $\hat{A}$ is consistent.

(c)

Godunov's method can be outlined in two steps. Suppose we have an approximation Q^n, (i) Define $\tilde{\mathbf{q}}(x,t)$ for all x and $t^n<t<t^{n+1}=t^n+k$ as the exact solution to the conservation law $\tilde{\mathbf{q}}_t+f(\mathbf{q})_x=\mathbf{S}$, satisfying the initial condition $\tilde{\mathbf{q}}(x,t^n)=Q_j^n$, $\forall x\in]x_{j-1/2},x_{j+1/2}[$. Average the resulting function $\tilde{Q}(x,t^{n+1})$ over each cell $(x_{j-1/2},x_{j+1/2})$ to obtain the approximation

$$Q^{n+1}=\frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}}\tilde{\mathbf{q}}(x,t^{n+1})dx$$

at t^{n+1}. Now this procedure can be repeated to advance to the next time-step. In Step (i), we need to solve an exact Riemann problem at each cell-interface, over a small time interval $]t^n,t^{n+1}[$. Since $\tilde{\mathbf{q}}^n$ is a solution to a balance equation, it is similar to a conservation law, but inhomogeneous. However, we can do the same procedure to find the integral form of the conservative law. We just integrate along space and time, and use the hypothesis on the solution. In this case, the linearity of the integral allow us to see that there is almost the same law, with the difference that we have the integral in S to add. We rewrite indeed Q^{n+1} as

$$\frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}}\tilde{\mathbf{q}}(x,t^n)dx-\frac{1}{\Delta x}\int_{t_n}^{t_{n+1}}f(\tilde{\mathbf{q}}(x_{j+1/2},t)-f(\tilde{\mathbf{q}}(x_{j-1/2},t)dt+\frac{1}{\Delta x}\int_{x_{j-1/2}}^{x_{j+1/2}}\int_{t_n}^{t_{n+1}}S$$

$$=:Q^n-\frac{\Delta t}{\Delta x}(F_{j+1/2}^n-F_{j-1/2}^n)+\Delta tS_j^n$$

with $F_{j+1/2}^n$ the mean of $\tilde{\mathbf{q}}(x_{j+1/2},.)$ along $]t^n,t^{n+1}[$ and S_j^n the mean of S on $]x_{j-1/2},x_{j+1/2}[\times]t^n,t^{n+1}[$. Here S_j^n depend only on the computation of the integral, since we know S. $F_{j+1/2}$ is determined by $\tilde{\mathbf{q}}$ which is determined by the initial conditions Q_j^n only, and it is locally a Riemman problem around $x_{j+1/2}$, so it only depends on Q_j^n and Q_{j+1}^n. By the mean theorem we can write $F_{j+1/2} = f(u^*(Q_j^n, Q_{j+1}^n))$, and if Δt is small enough, $\mathbf{q}(x_{j+1/2}, t)$ is the result of locals Riemann problems that does not interact, so we can solve them separately by Riemann solvers or approximations.

In this Project, we evaluate all the integral by midpoint formulas and resolve the Riemann problem by the Roe method whose matrix was developed in b).

We now implement the Roe method with the Roe matrix as described. For the Roe matrix, we explicitly compute the value of $|A| = V|S|V^{-1}$ by simple algebra (spectral decomposition and some substitutions that are detailed in the code in purpose to have a simple formula). The code is the following.

```matlab
function meth = G_Roe(prob, meth)
% Roe scheme
% return the structure meth with Q and t updated
    CFL= 0.5;
    g=1;
    Qext = prob.bound(meth.Q);
    uu= meth.Q(2,:) ./ meth.Q(1,:);
    gradnorm = max( abs(uu) + sqrt(g*meth.Q(1,:)));
    k = CFL*meth.dx / gradnorm;
    if meth.t+k <= prob.T   % it is not exceeding T or we reached it already
        lambda = CFL / gradnorm; %=k/dx;
    else % it is not exceeding T
        k= prob.T - meth.t;
        lambda= k / meth.dx;
    end
    flux = zeros(size(meth.Q));
    for j=1:meth.N+2
        flux(:,j)=F(Qext(:,j),Qext(:,j+1));
    end
    SS= prob.S(meth.x,meth.t+k/2);
    meth.Q = meth.Q - lambda*(flux(:,2:end) - flux(:,1:end-1)) + k*SS;
    meth.t = meth.t+k;
end

function absA = abs_roe_matrix(Ql,Qr)
% compute the spectral absolute value of the Roe matrix
    g=1;
    Q = [Ql,Qr];
    Z = Q./sqrt(Q(1,:));
    t = sqrt(g*mean(Z(1,:).^2));
    r = mean(Z(2,:)) / mean(Z(1,:));
    s=t+r;
    d=t-r;
    absA = [s*abs(d)+d*abs(s)   , abs(s)-abs(d);
         s*d*(abs(s)-abs(d)), s*abs(s)+d*abs(d)] / (2*t);
end

 function flux = F(u,v)
 % The Roe flux
    g=1;
    absA= abs_roe_matrix(u,v);
    flux = 0.5*(f(u(1),u(2)) + f(v(1),v(2))) - 0.5*absA*(v-u);
 end

%  function A=abs_roe_matrix_old(Ql,Qr)
%     g=1;
%     Q = [Ql,Qr];
%     Z = Q./sqrt(Q(1,:));
%     ratio = mean(Z(2,:)) / mean(Z(1,:));
%     A = [0 , 1 ; g*mean(Z(1,:).^2) - ratio^2 , 2*ratio ];
%     [V,D] = eig(A);
%     A = V*abs(D)/V;
% end
```

where we used the fact that for the roe method the resulting flux is

$$F(u, v) = (f(u) + f(v))/2 - |A|(v - u)/2$$

(d)

Now that we have all the tools needed we make a script that display the wanted plots. The code is the same as before but replacing the function `G_LF` by `G_Roe` each time. We obtain the same plots than before for the solutions, we don't show them then again (the code can display them if needed). The only difference we can see is for the error, for the problem 1.b), we see the error is a before, of order one:

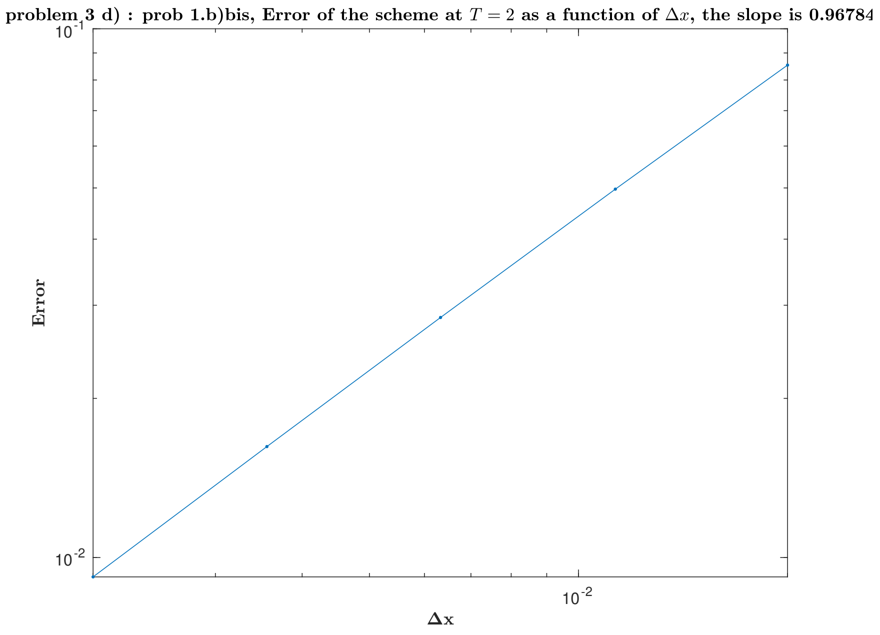


Figure 9: Plot of the error with the Roe method for various Δx.

For the problem 2.b) however, we see that the plot isn't a perfect line. This is because the error is compared to a numerical solution, and the scheme converge artificially faster than if it was compared to the real solution. The mean slope stay one so this confirm the theory that say it is of order one, because we've done order one approximation to the integrals by taking a midpoint formula and not a higher scheme like the use of polynomial interpolation of order more than zero. here are these plots:

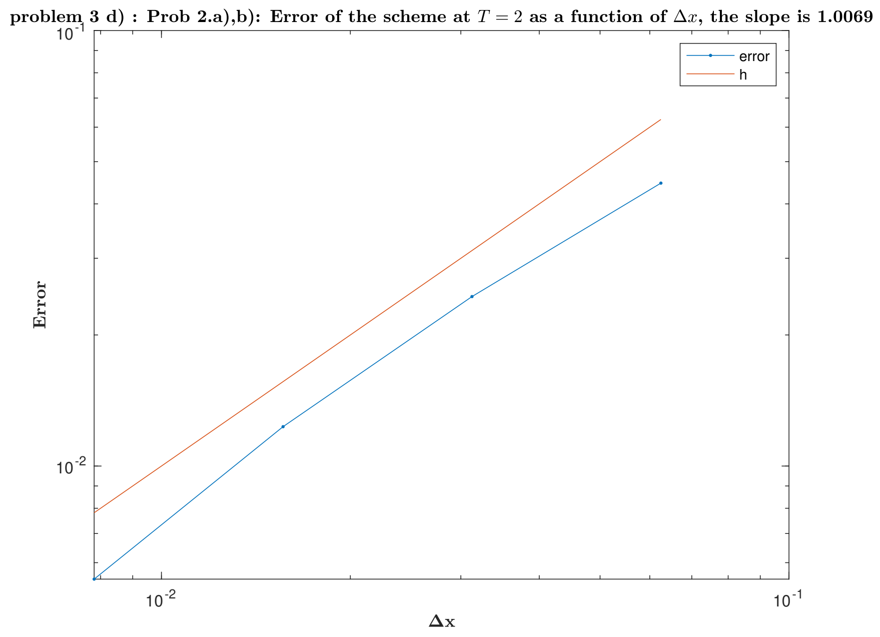


Figure 10: Plot of the error with the Roe method for various Δx with conditions (5).

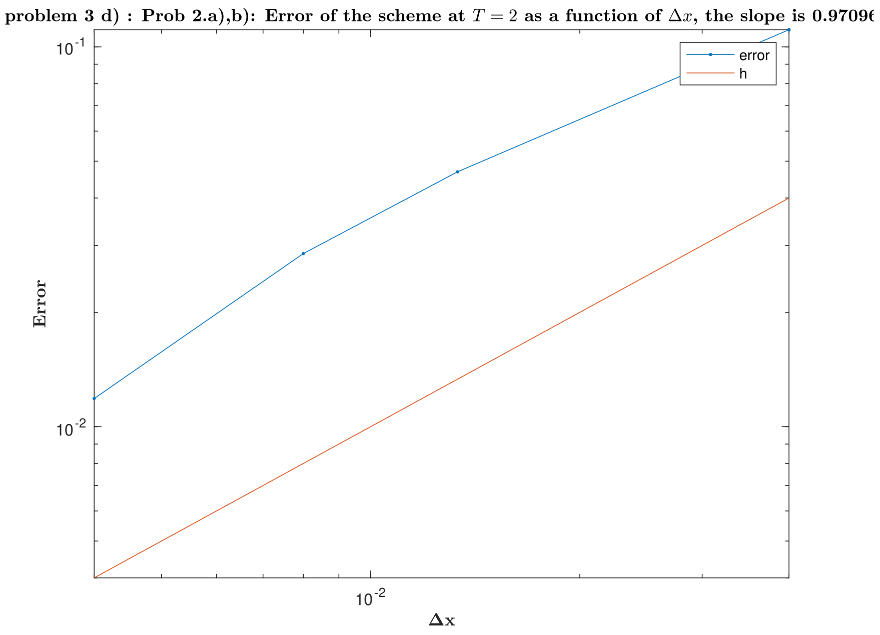


Figure 11: Plot of the error with the Roe method for various Δx with conditions (6).

(e)

We have already spoken about the convergence of the error since both method converge, and to the same solution, it become more convincing that it approximate the solution. We have seen that they both converge with the same rate.

Problem 4

(a)

We continue the `make_prob` function for the new problems:

```
        case "4a"
            h0 = @(x) ones(size(x));
            m0 = @(x) -1.5*(x<1);
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=0.5;
            prob.bound=open_bound;
    end
end
```

and compute with `shallow_water` a reference solution.

```
%% Problem 4.a)
clear
prob4a = make_prob("4a");
meth.N=10000;
meth.G= @(prob,meth) G_LF(prob,meth);
show.yes=1;
show.freq=100;

methref=shallow_water(prob4a, meth, show);
Qref=methref.Q;
```

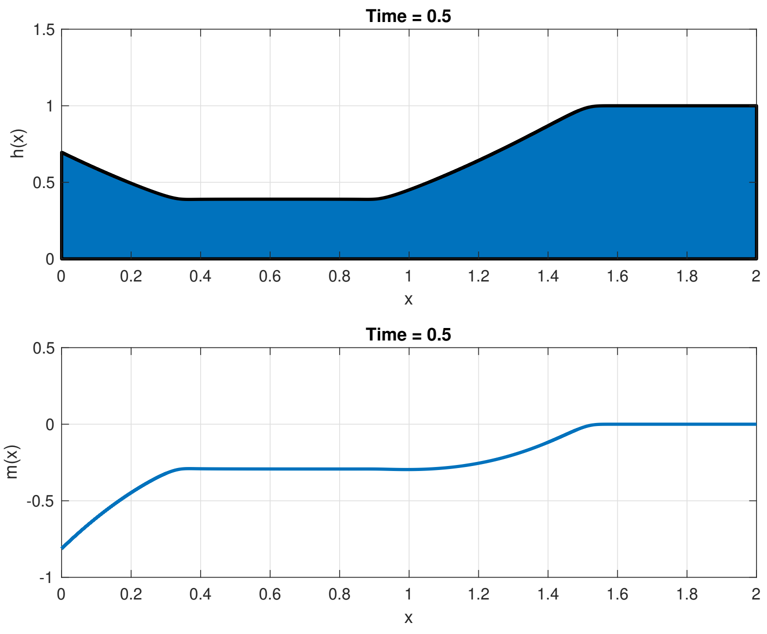


Figure 12: Numerical solution for problem 4.a) with the L.-F. scheme and $\Delta x = 2/10000$

(b)

We code a similar script like in 1.b) and 2b) that show the plots:

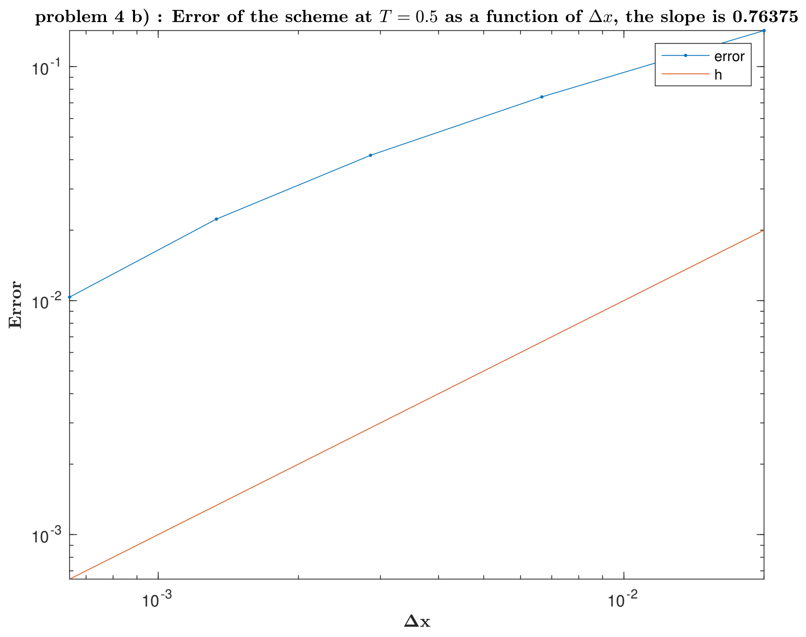


Figure 13: Numerical solutions for conditions (7) with the L.-F. scheme and various Δx

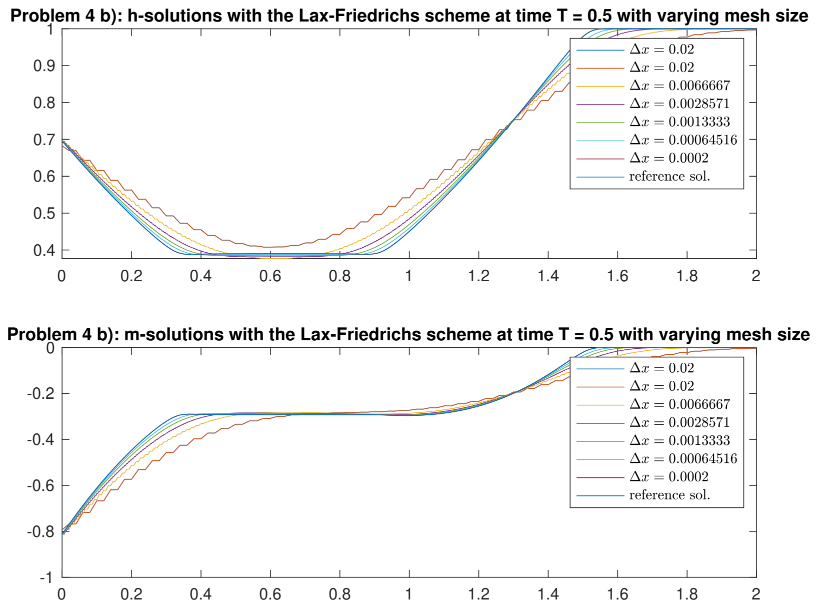


Figure 14: Error of the LF scheme with respect to Δx

(c)

Same experiment, different scheme. We just replace the line 5 and 4 of the two last codes by `meth.G= @(prob,meth)`, instead of `G_Roe(prob,meth)`. We don't show again here the code since it has nothing new. Here are the results:

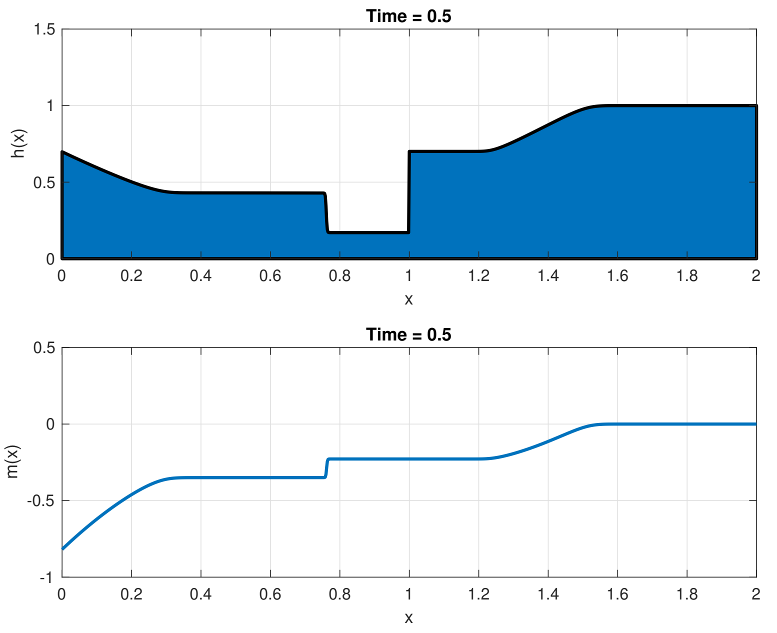


Figure 15: Numerical solution for problem 4.a) with the Roe scheme and $\Delta x = 2/1000$

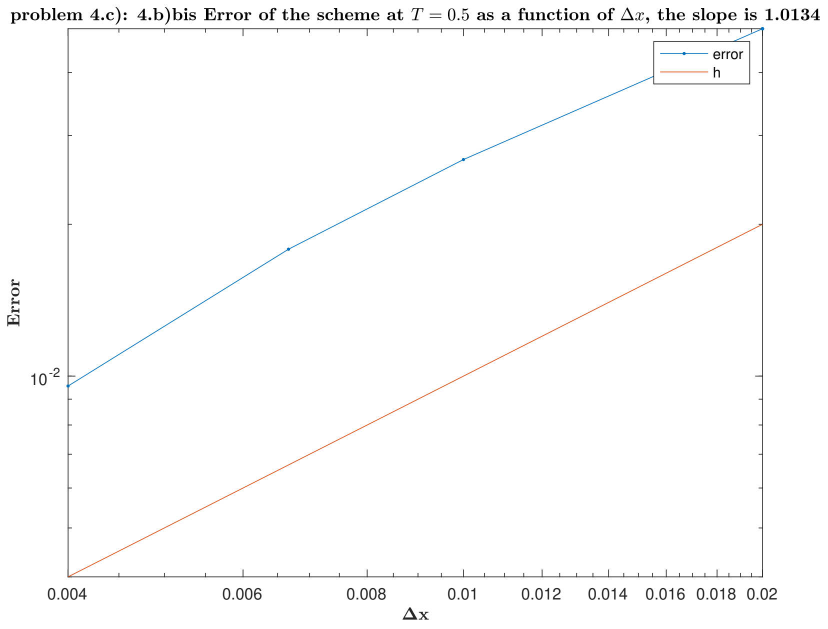
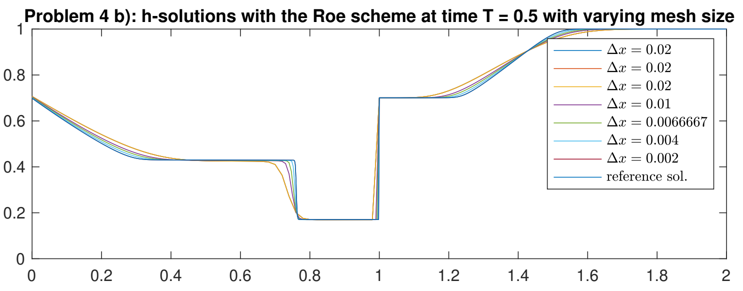


Figure 16: Numerical solutions for conditions (7) with the Roe scheme and various Δx



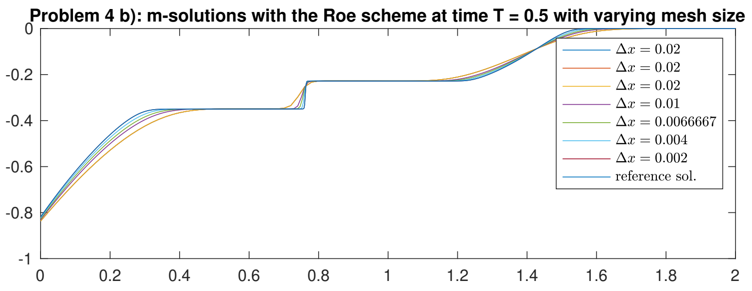


Figure 17: Error of the Roe scheme with respect to Δx

We immediately see that the solution the Roe scheme gives us is really not the same as for the LF scheme. It has indeed a discontinuity in the middle that has been form since the first times of the evolution. At first there were a short and deep sink which in the case of LF, transformed into a diffusion wave. In the case of Roe a stair at the middle appear and stay discontinuous along the evolution without diffusing. Obviously this comes from the fact that the LF scheme doesn't resolve the Riemann problem, with the cost of a strong dissipating effect.

Annex

the whole code of `code/make_prob.m`:

```
function prob = make_prob(question)
% build the structure prob that describe a problem
% prob contain the initial condition q_0,
% (sometimes) the true solution q_true, the source S, the limit time T,
% and the type of boundary condition bound.
    periodic_bound = @(Q) [Q(:,end), Q, Q(:,1)];
    open_bound     = @(Q) [Q(:,1), Q, Q(:,end)];
    switch question
        case "1a"
            u=0.25;
            g=1;
            h0 = @(x) 1 + 0.5*sin(pi*x);
            m0 = @(x) u*h0(x);
            h_true = @(x,t) h0(x-t);
            m_true = @(x,t) u*h_true(x,t);

            prob.q0 = @(x) [h0(x); m0(x)];
            prob.q_true = @(x,t) [h_true(x,t); m_true(x,t)];
            prob.S= @(x,t) pi/2*cos(pi*(x-t)) .* ...
                [(u-1)*ones(1,length(x)) ; u^2-u+g*h0(x-t)];
            prob.T=2;
            prob.bound=periodic_bound;
        case "2a5"
            h0 = @(x) 1 - 0.1*sin(pi*x);
            m0 = @(x) zeros(size(x));
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=2;
            prob.bound=periodic_bound;
```

```matlab
        case "2a6"
            h0 = @(x) 1 - 0.2*sin(2*pi*x);
            m0 = @(x) 0.5*ones(size(x));
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=2;
            prob.bound=periodic_bound;
        case "4a"
            h0 = @(x) ones(size(x));
            m0 = @(x) -1.5*(x<1);
            prob.q0 = @(x) [h0(x); m0(x)];
            prob.S= @(x,t) zeros(size(x));
            prob.T=0.5;
            prob.bound=open_bound;
    end
end
```

All the code is in the annexed file.