Exercise set 9

Numerical Methods for Hyperbolic Partial Differential Equations
IMATH, FS-2020
Lecturer: Dr. Philipp Öffner
Teaching Assistant: Davide Torlo

Problem 9.1 E-schemes (2pts)

Consider a consistent monotone 3-point numerical flux $f^{num}(u,v)$. The scheme given by this flux is said to be an *E-scheme* if it satisfies

$$(u_{j+1}-u_j)(f^{num}(u_j,u_{j+1})-f(u)) \leq 0 \quad \forall u \in [u_j,u_{j+1}] \text{ or } \forall u \in [u_{j+1},u_j], \tag{1}$$

according to the sign of $(u_{j+1}-u_j)$.

1. Prove that a 3-point monotone scheme is an E-scheme.

   Solution

   Let us rewrite the definition of E-scheme more explicitly. A numerical flux generates an E-scheme if

   $$\begin{cases} f^{num}(u_j,u_{j+1}) \leq f(u) & \forall u_j \leq u \leq u_{j+1}, \\ f^{num}(u_j,u_{j+1}) \geq f(u) & \forall u_{j+1} \leq u \leq u_j. \end{cases} \tag{2}$$

   Consider a monotone scheme $f^{num}(u,v)$, i.e., $u \mapsto f^{num}(u,v)$ is monotone non decreasing and $v \mapsto f^{num}(u,v)$ is monotone non increasing. We have

   $$\begin{cases} f^{num}(u_j,u_{j+1}) \leq f^{num}(u,u_{j+1}) \leq f^{num}(u,u) = f(u) & \text{if } u_j \leq u \leq u_{j+1} \\ f^{num}(u_j,u_{j+1}) \geq f^{num}(u,u_{j+1}) \geq f^{num}(u,u) = f(u) & \text{if } u_{j+1} \leq u \leq u_j. \end{cases} \tag{3}$$

   Hence, the scheme is an E-scheme.

2. Consider the Godunov flux

   $$f_G^{num}(u,v) := \begin{cases} \min_{u_j \leq u \leq u_{j+1}} f(u) & u_j \leq u_{j+1}, \\ \max_{u_{j+1} \leq u \leq u_j} f(u) & u_{j+1} \leq u_j. \end{cases} \tag{4}$$

   Prove that every monotone 3-point scheme with numerical flux f^{num} under CFL≤ 1 verifies

   $$\begin{cases} f^{num}(u_j,u_{j+1}) \leq f_G^{num}(u_j,u_{j+1}) & \text{if } u_j \leq u_{j+1}, \\ f^{num}(u_j,u_{j+1}) \geq f_G^{num}(u_j,u_{j+1}) & \text{if } u_{j+1} \leq u_j. \end{cases} \tag{5}$$

Solution

In case $u_j \le u_{j+1}$, we know for any $u \in [u_j, u_{j+1}]$ that $f^{num}(u_j, u_{j+1}) \le f(u)$ so, it is also true that

$$f^{num}(u_j, u_{j+1}) \le \min_{u_j \le u \le u_{j+1}} f(u) = f_G^{num}(u_j, u_{j+1}). \tag{6}$$

In case $u_j \ge u_{j+1}$, we know for any $u \in [u_{j+1}, u_j]$ that $f^{num}(u_j, u_{j+1}) \ge f(u)$ so, it is also true that

$$f^{num}(u_j, u_{j+1}) \ge \max_{u_{j+1} \le u \le u_j} f(u) = f_G^{num}(u_j, u_{j+1}). \tag{7}$$

Problem 9.2 Lax Wendroff (3 pts)

Code the two step Lax Wendroff scheme, where

$$\begin{cases} u_j^{n+1} = u_j^n - \lambda \left(f(u_{j+1/2}^{n+1/2}) - f(u_{j-1/2}^{n+1/2}) \right), \\ u_{j+1/2}^{n+1/2} = \frac{u_j + u_{j+1}}{2} - \frac{\lambda}{2} \left(f(u_{j+1}^n) - f(u_j^n) \right). \end{cases} \tag{8}$$

1. Write it in the conservation formulation.

   Solution

   We can write the scheme as

   $$u_j^{n+1} = u_j^n - \lambda \left(F_{j+1/2} - F_{j-1/2} \right) \tag{9}$$

   with

   $$F_{j+1/2}^n = f\left(\frac{u_j + u_{j+1}}{2} - \frac{\lambda}{2} \left(f(u_{j+1}^n) - f(u_j^n) \right) \right). \tag{10}$$

2. Code it in the usual code (check solution of Exercise Set 8).
3. Compare it with the classical Lax Wendroff scheme: check the accuracy order and the computational times:
   For $N \in \{2^k : k = 1, \dots, 10\}$ number of cells solve the advection equation $u_t + u_x = 0$ with $u_0(x) = \cos(\pi x)$ on $[-2, 2]$ with periodic boundary conditions. Compute the $\mathbb{L}^2$ error with respect the exact solution for both schemes and all N and the time needed to solve the scheme (see `tic, toc` in Matlab and package `time` in Python). Plot the error wrt N in an appropriate scale and the time and N in an appropriate scale. What can you say on the error? And on the time?

   Solution

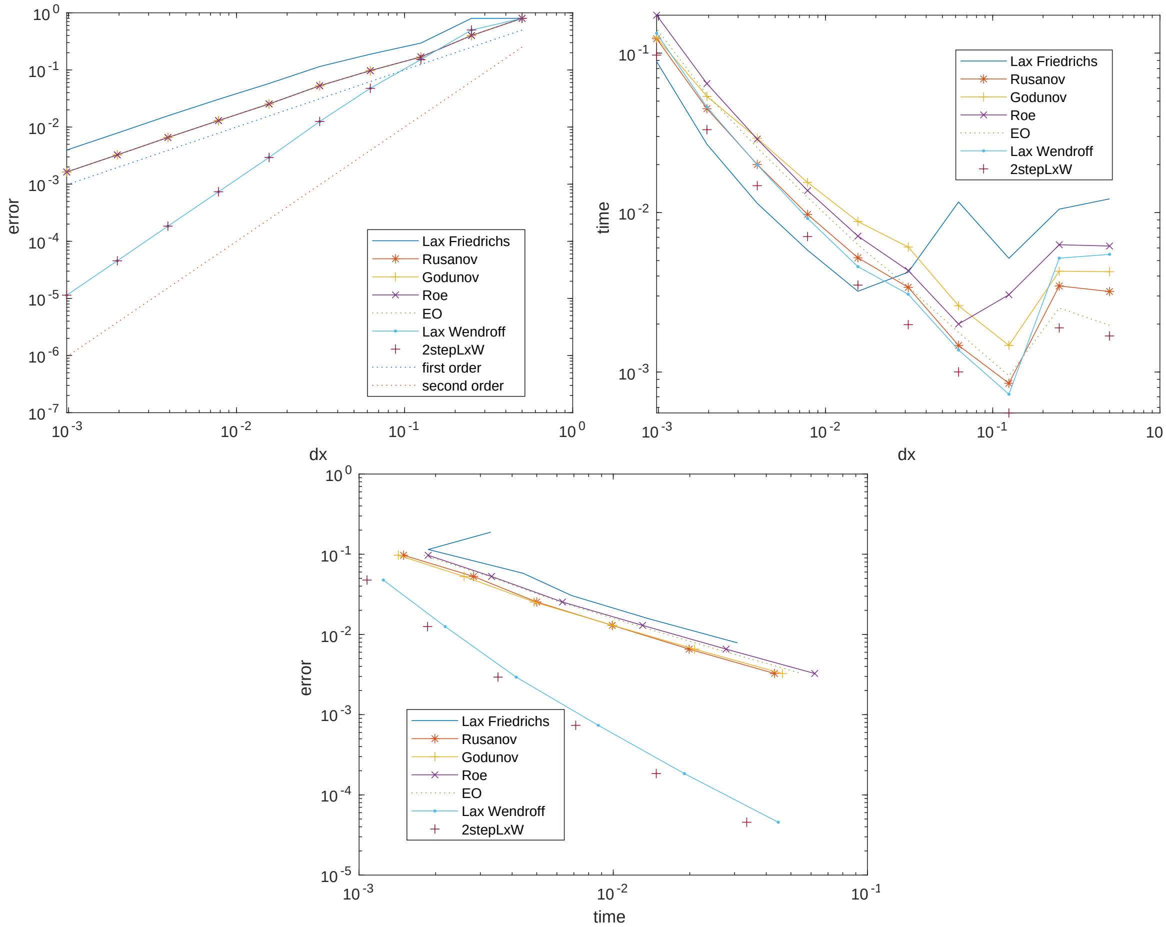
```
function [fNum] = numericalFlux(scheme,f,u,v,extra)
% encode in this function different numerical fluxes
switch scheme
    case "Lax Friedrichs"
        % extra should be lambda=dt/dx
        lam=extra{1};
        fNum=(f(u)+f(v))/2-(v-u)/lam/2;
    case "Lax Wendroff"
```

```
            % extra should be lambda=dt/dx
            lam=extra{1};
            df=extra{2};
            idxn= u==v;
            idx=logical(1-idxn);
            a=zeros(size(u));
            a(idx)=(f(u(idx))-f(v(idx)))./(u(idx)-v(idx));
            fNum=(f(u)+f(v))/2-lam/2*a.*(f(v)-f(u));
        case "2stepLxW"
            % extra should be lambda=dt/dx
            lam=extra{1};
            fNum=f(0.5*(u+v) -lam/2*(f(v)-f(u)));
        case "Rusanov"
            % extra should be f'
            df=extra{1};
            fNum=(f(u)+f(v))/2-max(abs(df(u)),abs(df(v))).*(v-u)/2;
        case "Godunov"
            % extra should be omega the unique local minimum of f
            omega=extra{1}*ones(size(u));
            fNum=max(f(max(u,omega)),f(min(v,omega)));
        case "Roe"
            % extra should be empty
            idxs=u==v;
            idx=logical(1-idxs);
            fNum=f(u);
            A=(f(u(idx))-f(v(idx)))./(u(idx)-v(idx));
            fNum(idx)=f(u(idx)).*(A>=0)+f(v(idx)).*(A<0);
        case "EO" %Engquist--Osher
            % extra should be omega the unique local minimum of f
            omega=extra{1}*ones(size(u));
            fNum=f(max(u,omega))+ f(min(v,omega));
end


end
```

Listing 1: NumericalFlux.m

```
%% Convergence
model.f=@(u) u;
model.df=@(u) ones(size(u));
model.T=1;
model.a=-2;
model.b=2;
model.u0=@(x) 1+ 0.2* cos(pi*x);% (x<1).*(x>0); %(x<1).*(x>0);%0.2* cos(pi*x);% uL*(x<0)+
    uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";
model.exact=@(x,t) model.u0(x-t);

model.entropy=@(x) abs(x);

solver.Nx = 100;
solver.CFL = 0.7;



schemes=["Lax Friedrichs";"Rusanov";"Godunov";"Roe";"EO";"Lax Wendroff";"2stepLxW"];
```

```
styles=["-","*-","+-","x-",":",".-","+"]
nn=6;
Ns=2.^[1:nn];

clear u x t ent errors times

for k=1:length(schemes)
    solver.scheme=schemes(k);
    for n=1:nn
        solver.Nx=Ns(n);
        tic
        [u,x,t,ent]= runScheme(model, solver);
        times(k,n)=toc;
        errors(k,n)=computeError(u,x,t,model);
    end
end

fig=figure()
for k=1:length(schemes)
    scheme=schemes(k);
    loglog(1./Ns,errors(k,:),styles(k),'DisplayName',scheme)
    hold on
end
loglog(1./Ns,1./Ns,':','DisplayName','first order')
loglog(1./Ns,1./Ns.^2,':','DisplayName','second order')
legend('Location','best')
xlabel('dx')
ylabel('error')
saveas(fig,'errorLxW.pdf')

fig=figure()
for k=1:length(schemes)
    scheme=schemes(k);
    loglog(1./Ns,times(k,:),styles(k),'DisplayName',scheme)
    hold on
end
legend('Location','best')
xlabel('dx')
ylabel('time')
saveas(fig,'computationalTimeLxW.pdf')

fig=figure()
for k=1:length(schemes)
    scheme=schemes(k);
    loglog(times(k,:),errors(k,:),styles(k),'DisplayName',scheme)
    hold on
end
legend('Location','best')
ylabel('error')
xlabel('time')
saveas(fig,'errorTimeLxW.pdf')



function err=computeError(u,x,t,model)
    err=norm(u(end,:)-model.exact(x,t(end)))*sqrt(x(2)-x(1));
end
```



Listing 2: testOrderSpeedLxW.m

We can see that the scheme, being always second order accurate, is also faster than the previous Lax Wendroff scheme.

Problem 9.3 MUSCL (5 pts + 1 extra)

In this problem we code the MUSCL scheme with different choices of *switching function*. The scheme can be summarized in the following steps.

1. Given the solution u_j^n, reconstruct the solution inside the cell $[x_{j-1/2}, x_{j+1/2}]$ as linear function

$$u_j^n(x) = u_j^n + \sigma_j^n (x - x_j), \quad x \in [x_{j-1/2}, x_{j+1/2}]. \tag{11}$$

Define $u_{j,\pm}^n := u_j^n(x_{j\pm 1/2})$.

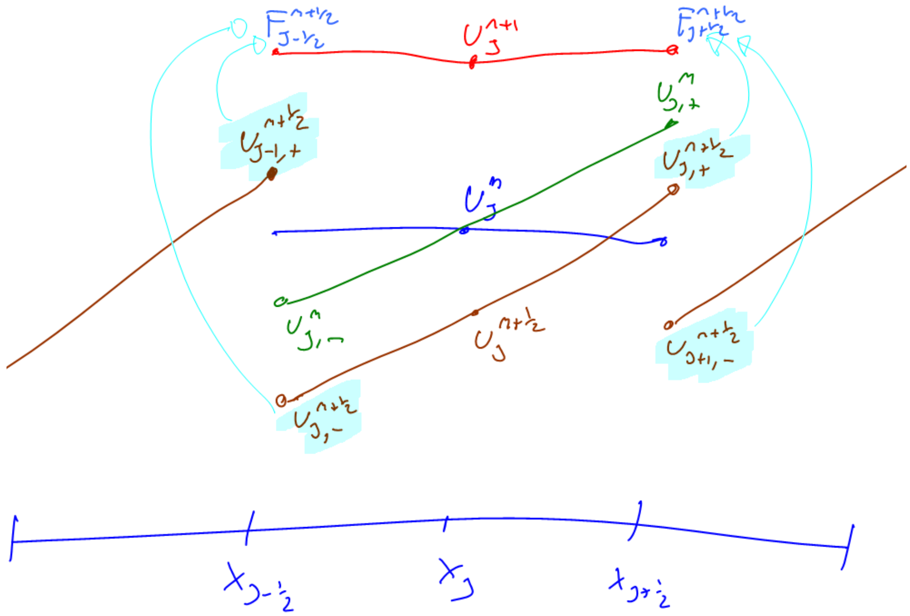


Figure 1: An amazing sketch of the MUSCL scheme. In order blue, green, brown, lightblue, red.

2. Using this reconstruction compute an intermediate step at time $t^{n+1/2}$ for the interface values

$$u_{j,\pm}^{n+1/2} = u_{j,\pm}^n - \frac{\lambda}{2}\left(f(u_{j,+}^n) - f(u_{j,-}^n)\right). \tag{12}$$

3. Apply the numerical flux on the intermediate states

$$u_j^{n+1} = u_j^n - \lambda\left(F^{n+1/2}_{j+1/2} - F^{n+1/2}_{j-1/2}\right) \tag{13}$$

   where the numerical fluxes are defined as

$$F^{n+1/2}_{j+1/2} = f^{num}(u^{n+1/2}_{j,+}, u^{n+1/2}_{j+1,-}). \tag{14}$$

Check figure 1 for an idea of the steps of the scheme.

The left choice to define this scheme are the way we define the slope σ_j^n, and it will depend on u_{j-1}, u_j, u_{j+1} and the numerical fluxes f^{num}.

The slope can be defined in a general way as

$$\sigma_j^n(u_{j-1}^n, u_j^n, u_{j+1}^n) = \frac{u_{j+1}^n - u_j^n}{\Delta x}\phi(r_j^n), \quad r_j^n := \frac{u_j^n - u_{j-1}^n}{u_{j+1}^n - u_j^n}. \tag{15}$$

Here, follows some possible choices for the *switching* function.

Name	*Switching* function $\phi(r)$
Upwind	0
Lax–Wendroff	1
Beam–Warming	r
Minmod	$\text{minmod}(1, r)$
Superbee	$\max(0, \min(1, 2r), \min(2, r))$
MC	$\max\left(0, \min\left(\frac{1+r}{2}, 2, 2r\right)\right)$
van Leer	$\frac{r+\lvert r\rvert}{1+\lvert r\rvert}$

Table 1: The most popular flux limiters.

Note that

$$\text{minmod}(a, b) = \begin{cases} \text{sign}(a)\min(|a|, |b|) & \text{if } ab \geq 0, \\ 0 & \text{else.} \end{cases} \tag{16}$$

1. Which condition should the switching function and the numerical flux f^{num} verify in order to have a TVD and second order scheme on linear advection problems?

   Solution

   On linear advection problems if we use a monotone numerical flux and the switching function is such that

   $$0 \leq \frac{\phi(r)}{r}, \phi(r) \leq 2,$$

   then the scheme is TVD, and it if also fulfills

   $$\phi(r) = \alpha + (1-\alpha)r, \quad \alpha \in [0, 1]$$

   then the scheme is second order.

2. Code a function `switchingFunctions`, which code all the functions contained in Table 1 (inputs: a string with the name of the function and r, output the value of the function evaluated in r). In another script, plot all the functions in one figure on the interval $[0, 5]$. Which of the functions does not verify the TVD conditions and which one does not verify the second order condition?

   Solution

```
function sigma=switchingFunction(r,solver)
switch solver.limiter
    case ”vanLeer”
        sigma= (r+abs(r))./(1+abs(r));
    case ”upwind”
        sigma=zeros(size(r));
    case ”Lax Wendroff”
        sigma=ones(size(r));
    case ”Beam Warming”
        sigma=r;
    case ”minmod”
```

```
        sigma=minmod(ones(size(r)),r);
    case "Superbee"
        sigma=max(zeros(size(r)),min(ones(size(r)),2*r));
        sigma=max(sigma,min(2*ones(size(r)),r));
    case "MC"
        sigma=min((1+r)/2,2*ones(size(r)));
        sigma=min(sigma,2*r);
        sigma=max(sigma,zeros(size(r)));
end
end

function c=minmod(x,y)
    c= (sign(x)+sign(y))/2.*min(abs(x),abs(y));
end
```

Listing 3: switchingFunction.m

```
limiters=[ "minmod", "vanLeer","upwind","Lax Wendroff", "Beam Warming","Superbee","MC"];
xx=linspace(0,5,101);
fig=figure()
styles=["*-","+-","x-","-",":",".-","+"];
for k=1:length(limiters)
    solver.limiter=limiters(k);
    yy=switchingFunction(xx,solver);
    plot(xx,yy,styles(k),'DisplayName',limiters(k))
    hold on
end

title('Switching functions')
legend('Location','best')
saveas(fig,'switchingFunctions.pdf')
system('pdfcrop switchingFunctions.pdf')
```

Listing 4: testSwitchingFunctions.m

We clearly see in figure 2 that the upwind function does not verify the second order condition and the Beam Warming and Lax Wendroff do not verify the TVD condition, while the other functions verify all the properties.

3. Code the MUSCL scheme as presented above, making use of the old function for numerical fluxes and of the `switchingFunctions`.
   Hint: build a function `computeSlope`, which compute the slopes of the intervals, check that the reconstruction that you obtain is TVD (a simple linear advection problem with few Ns is enough to check it).

   Solution

```
function [u,x,t,ent] = runMUSCL(model, solver,varargin)
if nargin<3
    withplot=0;
else
    withplot=varargin{1};
end
N=solver.Nx;
```

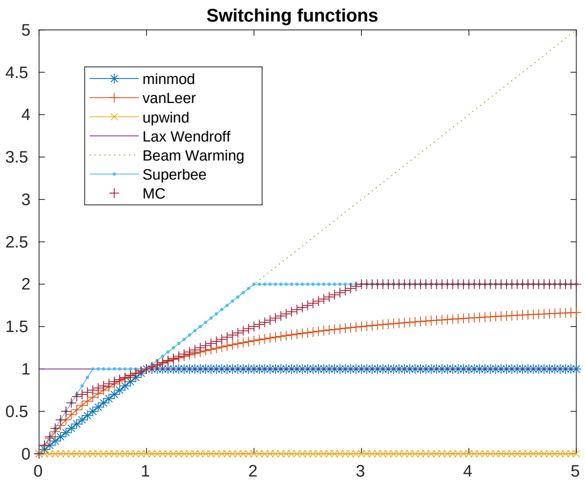


Figure 2: Switching Functions

```
x=linspace(model.a,model.b,solver.Nx+1);
dx=x(2)-x(1);
maxdfu=max(abs(model.df(model.u0(x))));
dt=dx*solver.CFL/maxdfu;
Nt=ceil(model.T/dt);
dt=model.T/Nt;
t=linspace(0,model.T,Nt+1);
%u(j,n+1)= H(u(j-1,n),u(j,n),u(j+1,n))
if model.BC=="periodic"
    j=1:N;
    jR=[2:N,1];
    jL=[N, 1:N-1];
    u=zeros(Nt+1,N);
    ent=zeros(Nt+1,N);
    x=x(j);
elseif model.BC=="dirichlet"
    j=[2:N];
    jL=[1:N-1];
    jR=[3:N+1];
    x=x(1:N+1);
    u=zeros(Nt+1,N+1);
    ent=zeros(Nt+1,N+1);
    u(:,1)=model.u0(x(1));
    u(:,N+1)=model.u0(x(N+1));
elseif model.BC=="neumann"
    j=[1:N+1];
    jL=[1, 1:N];
    jR=[2:N+1, N+1];
    x=x(1:N+1);
    u=zeros(Nt+1,N+1);
    ent=zeros(Nt+1,N+1);
end

u(1,:)=model.u0(x);
```

```
ent(1,:)=model.entropy(u(1,:));
    extra=defineExtraScheme(solver.scheme,dt/dx,model.df);
for kt=2:Nt+1
    un=u(kt-1,:);
    slopes=slope(un(jL),un(j),un(jR), solver);
    um=un; up=un; u1m=un; u1p=un;
    um(j)=un(j)-slopes/2;
    up(j)=un(j)+slopes/2;
    plotReconstruction(j,dx,x,un,um,up,withplot);
    u1p(j)=up(j) - dt/dx/2 *( model.f(up(j)) -model.f(um(j)) );
    u1m(j)=um(j) - dt/dx/2 *( model.f(up(j)) -model.f(um(j)) );
    plotReconstruction(j,dx,x,un,u1m,u1p,withplot);
    %    u2star=u(kt-1,:);
    %ustar(j)=u(kt-1,j)-evolutionOperator(u(kt-1,:), j, jL, jR, solver, dt, dx,x, model,
    extra,withplot);
%    u2star(j)=ustar-evolutionOperator(ustar, j, jL, jR, solver, dt, dx,x, model,extra,
    withplot);
%    u(kt,j)=0.5*(u2star(j)+u(kt-1,j));%u(kt-1,j)-evolutionOperator(ustar, j, jL, jR,
    solver, dt, dx,x, model,extra,withplot);
    u(kt,j)=un(j)- dt/dx*(...
        numericalFlux(solver.scheme,model.f,u1p(j),u1m(jR),extra)...
        -numericalFlux(solver.scheme,model.f,u1p(jL),u1m(j),extra));
    if withplot
        for k=1:length(j)
            plot(x(j(k))+[-dx/2,dx/2],[u(kt,j(k)),u(kt,j(k))],'r')
        end
        drawnow
        pause(1)
    end
    ent(kt,:)=model.entropy(u(kt,:));
end

end

function plotReconstruction(j,dx,x,u,um,up,withplot)
    if withplot
        figure(4)
        clf()
        plot(x(j),u(j),'x')
        hold on
        for k=1:length(j)
            plot(x(j(k))+[-dx/2,dx/2],[um(j(k)),up(j(k))],'-')
        end
        drawnow
        pause(0.5)
    end
end


function extra=defineExtraScheme(scheme,lam,df)
switch scheme
    case {"Lax Friedrichs","2stepLxW"}
        extra={lam};
    case "Lax Wendroff"
        extra={lam, df};
    case "Rusanov"
        extra={df};
```

```
    case {"Godunov","EO"}
        extra={0}; %only Burgers
    case "Roe"
        extra={};
end

end

function sigma=slope(uL,u,uR,solver)
idx=logical(1-(uR==u));
r=zeros(size(u));
r(idx)=(u(idx)-uL(idx))./(uR(idx)-u(idx));
sigma=(uR-u).*switchingFunction(r,solver);

end
```

Listing 5: runMUSCL.m

4. Test the accuracy of the method on $\partial_t u + \partial_x u = 0$ with $u_0(x) = 1 + 0.2\cos(\pi x)$ on $[-2, 2]$ for all switching functions and Rusanov as numerical flux. Which limiter performs better? Which one does not reach second order of accuracy?

5. Test the TVD property of the scheme. For $\partial_t u + \partial_x u = 0$ with

$$u_0(x) = \begin{cases} 1 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases}$$

on $[-2, 2]$ with all the limiters and Rusanov numerical flux compute the TV of the solution as a function of time on a simulation with $N = 100$ cells, $T = 1$ and CFL=0.9. Which limiter does not fulfill the TVD conditions?

Solution

```
%% Test MUSCL SCHEME on LINEAR ADVECTION equation
model.f=@(u) u;%;.^2;
model.df=@(u) ones(size(u));
model.T=1;
model.a=-2;
model.b=2;
model.u0=@(x) (x<1).*(x>0);%1+ 0.2* cos(pi*x);% 0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos
    (pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";
model.exact=@(x,t) model.u0(x-t);

model.entropy = @(u)  abs(u);%u.^2;%abs(u); %u.^2/2; %@(u)
model.eFlux = @(u) sign(u).*f(u); % @(u) u.^3/3;

%% One example
solver.Nx = 40;
solver.CFL=0.8;%0.7;
solver.limiter="minmod";%"Beam Warming";
solver.scheme="Rusanov";

runMUSCL(model, solver);
```

```
%% Convergence
model.u0=@(x) 1+ 0.2* cos(pi*x);% 0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.exact=@(x,t) model.u0(x-t);


solver.Nx = 100;
solver.CFL = 0.9;
solver.limiter="MC";

schemes=["Lax Friedrichs";"Rusanov";"Godunov";"Roe";"EO";"Lax Wendroff";"2stepLxW"];
styles=["-","*-","+-","x-",":",".-","+"];
nn=10;
Ns=2.^[1:nn];

clear u x t ent errors times

for k=1:length(schemes)
    solver.scheme=schemes(k);
    for n=1:length(Ns)
        solver.Nx=Ns(n);
        tic
        [u,x,t,ent]= runMUSCL(model, solver);
        times(k,n)=toc;
        errors(k,n)=computeError(u,x,t,model);
    end
end

figure()
for k=1:length(schemes)
    scheme=schemes(k);
    loglog(1./Ns,errors(k,:),styles(k),'DisplayName',scheme)
    hold on
end
loglog(1./Ns,1./Ns,':','DisplayName','first order')
loglog(1./Ns,100*1./Ns.^2,':','DisplayName','second order')
legend('Location','best')
xlabel('dx')
ylabel('error')

%% Different limiters
model.u0=@(x) 1+ 0.2* cos(pi*x);% 0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.exact=@(x,t) model.u0(x-t);


solver.Nx = 100;
solver.CFL = 0.9;

solver.scheme="Rusanov";%"Godunov";%"Rusanov";
limiters=[ "minmod", "vanLeer","upwind","Lax Wendroff", "Beam Warming","Superbee","MC"];
styles=["-","*-","+-","x-",":",".-","+"];
nn=10;
Ns=2.^[2:nn];

clear u x t ent errors times
```

```
for k=1:length(limiters)
    solver.limiter=limiters(k);
    for n=1:nn-1
        solver.Nx=Ns(n);
        tic
        [u,x,t,ent]= runMUSCL(model, solver);
        times(k,n)=toc;
        errors(k,n)=computeError(u,x,t,model);
    end
end

fig=figure()
for k=1:length(limiters)
    scheme=limiters(k);
    loglog(1./Ns,errors(k,:),styles(k),'DisplayName',scheme)
    hold on
end
title(sprintf('Convergence with numerical flux %s',solver.scheme))
loglog(1./Ns,1./Ns,':','DisplayName','first order')
loglog(1./Ns,10*1./Ns.^2,':','DisplayName','second order')
legend('Location','best')
xlabel('dx')
ylabel('error')
saveas(fig, sprintf('convergence_%s.pdf',solver.scheme))


%% plot TV instability

model.f=@(u) u;%.^2;
model.df=@(u) ones(size(u));
model.T=1;
model.a=-2;
model.b=2;
model.u0=@(x) (x<1).*(x>0);% 1+ 0.2* cos(pi*x);%  %(x<1).*(x>0);%0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";
model.exact=@(x,t) model.u0(x-t);

model.entropy = @(u)  abs(u);%u.^2;%abs(u); %u.^2/2; %@(u)
model.eFlux = @(u) sign(u).*f(u); % @(u) u.^3/3;

solver.scheme="Rusanov";
solver.Nx=100;
solver.CFL=0.8;


limiters=[ "minmod", "vanLeer","upwind","Lax Wendroff", "Beam Warming","Superbee","MC"];
styles=["-","*-","+-","x-",":",".-","+"];
clear TV
fig=figure()
k=0;
for limiter=limiters
    k=k+1;
    solver.limiter=limiter;
    [u,x,t,ent]= runMUSCL(model, solver);
    for n=1:numel(t)
```

```
        TV(k,n) = sum(abs( diff(u(n,:))))+abs(u(n,end)-u(n,1));
    end
    plot(t,TV(k,:), styles(k),'DisplayName', limiter)
    hold on
end
legend('Location', 'best')
title(sprintf('Total variation %s',solver.scheme))
saveas(fig,sprintf('TV_MUSCL%s.pdf', solver.scheme))


%% plot TV instability for Burger

model.f=@(u) u.^2;
model.df=@(u) u;%ones(size(u));
model.T=1;
model.a=-2;
model.b=2;
model.u0=@(x) (x<1).*(x>0);%1+ 0.2* cos(pi*x);%  %0.2* cos(pi*x);% uL*(x<0)+uR*(x>=0); %cos(pi*x);%cos(pi*x);
model.BC="periodic";%"dirichlet";%"periodic";
model.exact=@(x,t) model.u0(x-t);

model.entropy = @(u)  abs(u);%u.^2;%abs(u); %u.^2/2; %@(u)
model.eFlux = @(u) sign(u).*f(u); % @(u) u.^3/3;

solver.scheme="Lax Friedrichs";
solver.Nx=100;
solver.CFL=0.5;


%runMUSCL(model, solver,1);


limiters=["minmod"];%, "vanLeer","upwind","Lax Wendroff", "Beam Warming","Superbee","MC"];
styles=["-","*-","+-","x-",":",".-","+"];
clear TV
fig=figure()
k=0;
for limiter=limiters
    k=k+1;
    solver.limiter=limiter;
    [u,x,t,ent]= runMUSCL(model, solver);
    for n=1:numel(t)
        TV(k,n) = sum(abs( diff(u(n,:))))+abs(u(n,end)-u(n,1));
    end
    semilogy(t,TV(k,:), styles(k),'DisplayName', limiter)
    hold on
end
legend('Location', 'best')
title(sprintf('Total variation %s CFL %g',solver.scheme,solver.CFL))
saveas(fig, 'TV_Burgers.pdf')
system('pdfcrop TV_Burgers.pdf TV_Burgers.pdf')



function err=computeError(u,x,t,model)
```

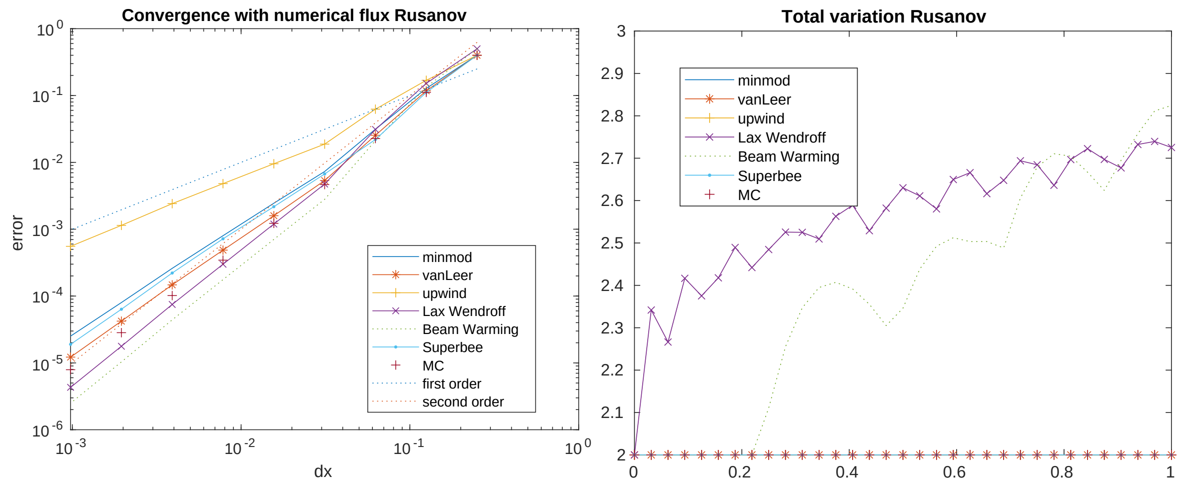


Figure 3: Convergence error and TV of MUSCL schemes

```
    err=norm(u(end,:)-model.exact(x,t(end)))*sqrt(x(2)-x(1));
end
```

Listing 6: testConvergenceMUSCL.m

We see that all the methods respect the conditions we theoretically found: upwind is 1st order, Lax-Wendroff and Beam Warming are not TVD, the rest is second order and TVD.

6. Test the scheme with Burgers' equation. What is happening? Why? Can you find a switching function and a numerical flux that, under some conditions, give a TVD scheme?

Solution

The conditions are not valid also for the Burgers' equation. What we can observe is that if we lower the CFL conditions and we choose a very diffusive numerical flux as the Lax Friedrichs, we can obtain TVD for the minmod limiter.

Organiser: Davide Torlo, Office: home (davide.torlo@math.uzh.ch)

Published: May 7, 2020

Due date: May 14, 2020, h10.00 (use the upload tool of my.math.uzh.ch, see wiki.math.uzh.ch/public/student_upload_homework or if you have troubles send me an email).

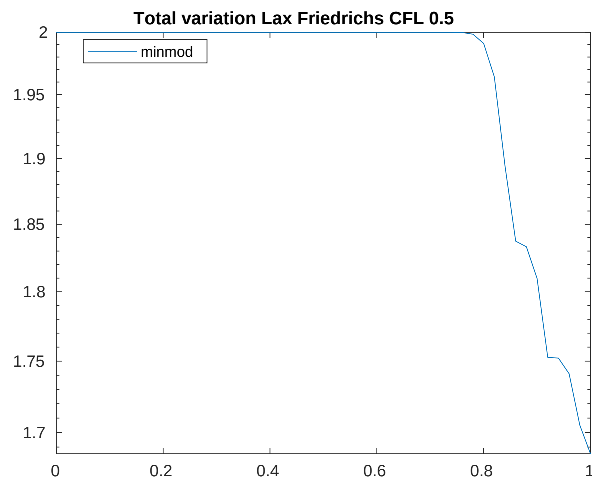


Figure 4: Convergence error and TV of MUSCL schemes on Burgers equation