Solution set 9:

TVD-RK and extensions

Exercise 9.1 Suppose that H_1 and H_2 are difference operators such that the schemes defined by

$$v_j^{n+1} = H_1(v^n;j) \ , \qquad w_j^{n+1} = H_2(w^n;j) \tag{1}$$

are TVD. That is, for sufficiently small time step,

$$TV(H_1(v^n;\cdot)) \le TV(v^n) \ , \qquad TV(H_2(w^n;\cdot)) \le TV(w^n) \qquad \forall n \ . \tag{2}$$

Let α_1 and α_2 be some positive numbers that satisfy $\alpha_1 + \alpha_2 = 1$, and define $H = \alpha_1 H_1 + \alpha_2 H_2$. Finally, suppose that u is a grid function defined by the scheme $u_j^{n+1} = H(u^n;j)$. Then, by the triangle inequality, we get

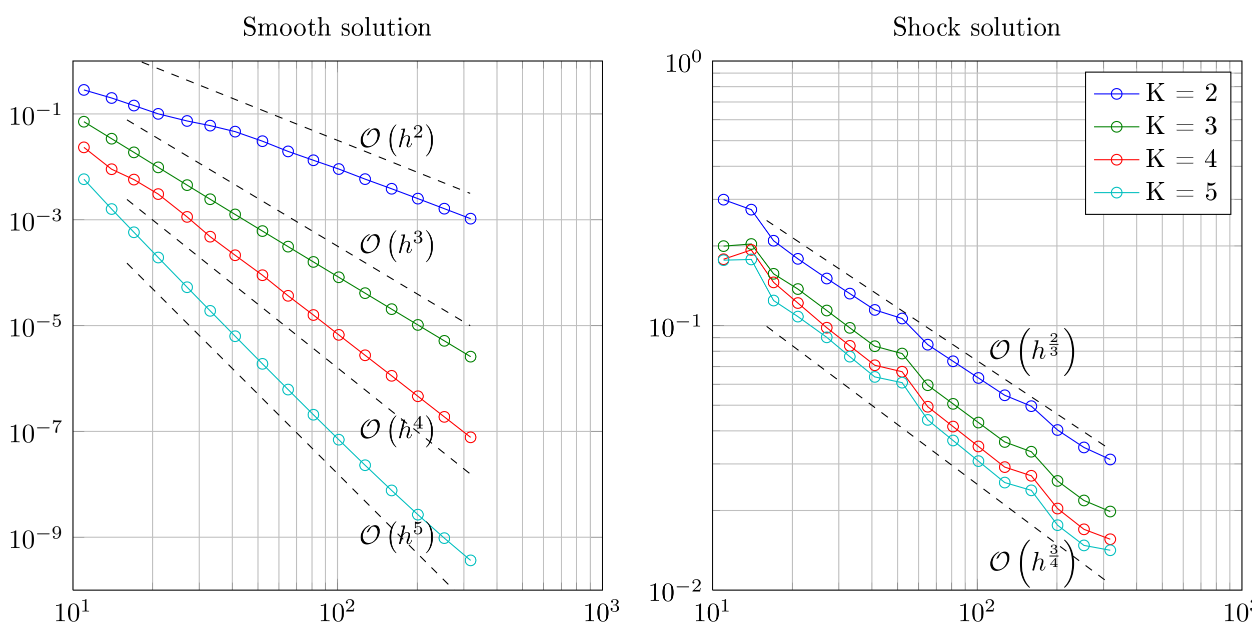
$$\begin{aligned} TV\left(u^{n+1}\right) &= TV(H(u^n;\cdot)) \\ &= TV(\alpha_1 H_1(u^n;\cdot) + \alpha_2 H_2(u^n;\cdot)) \\ &\le TV(\alpha_1 H_1(u^n;\cdot)) + TV(\alpha_2 H_2(u^n;\cdot)) \ . \end{aligned} \tag{3}$$

Since α_1 and α_2 are positive, the above implies

$$\begin{aligned} TV(H(u^n;\cdot)) &\le \alpha_1 TV(H_1(u^n;\cdot)) + \alpha_2 TV(H_2(u^n;\cdot)) \\ &\le \alpha_1 TV(u^n) + \alpha_2 TV(u^n) \\ &= TV(u^n) \ , \end{aligned} \tag{4}$$

where the last equality is due to $\alpha_1 + \alpha_2 = 1$. The argument used is valid for any convex combination of a finite number of TVD operators.

Exercise 9.2 **(a)** See code attached. **(b)** See figures 1 and 2 generated by the DEMO script, in the figures the accuracy is measured for various K in the ENO reconstruction of cell interfaces, a third order TVD-RK method with very small time-step is used for integration. **(c)** It seems to work as predicted. In smooth regions we now recover higher order accuracy, figure 1. However, the shocks are still resolved with $\le \mathcal{O}\left(\Delta x^1\right)$ accuracy, figure 2.



(a) (b)

Figure 1: Accuracy measured in the 1-norm at $T = 1$ when using the (a) smooth (b) discontinious initial condition.

```matlab
clear all,clc
% Run DEMO script to demonstrate the accuracy properties of the ENO
% cell interface reconstruction method in solving a simple scalar
% hyperbolic equation with periodic boundaries and smooth and non-smooth
% initial conditions. Integration using third order accurate SSP RK method.

%% Setup
data     = [1,2];
K        = [2,3,4,5];
H        = 1./ceil(1./(0.1.^(1:0.1:2.5)));
Cells    = zeros(numel(K),numel(H),numel(data));
Error    = zeros(numel(K),numel(H),numel(data));

%% Gather data
for d = 1:numel(data)
    for h = 1:numel(H)
        for o = 1:numel(K)
            % Setup problem
            [k,X,nX,T,nT] = Setup(H(h));
            % Create initial data
            if d == 1
                U            = InitialCondition(@Smooth, X, H(h) )';
            elseif d == 2
                U            = InitialCondition(@Shock, X, H(h) )';
            end
            u = U;
            C = GetC(K(o));
            % Solve
            for n = 1:nT
                U = SSPRK3ENO(C,H(h),nX,X,U,K(o),k);
            end
            Error(o,h,d) = norm((U-u),1)/nX;
            Cells(o,h,d) = nX;
        end
    end
end


%% Plot data
fp = 3;
ts = {'Smooth solution','Shock solution'};
for i = 1:numel(data)
    figure(i);
    for o = 1:numel(K)
        loglog(Cells(o,:,i),Error(o,:,i),'-o');
        hold all;
        S(o,:,i) = polyfit(log(Cells(o,end-fp:end,i)),log(Error(o,end-fp-2:end-2,i)),1);
    end
    legend('K = 2','K = 3','K = 4','K = 5');
    title(ts{i});
    grid on
    if i == 1
        X = ceil(1./(0.1.^(1.2:0.05:2.5)));
        loglog(X,10^(2.2)*(1./X).^5,'--k');
        loglog(X,10^(2.2)*(1./X).^4,'--k');
        loglog(X,10^(2.5)*(1./X).^3,'--k');
        loglog(X,10^(2.5)*(1./X).^2,'--k');
        text(1.2*10^(2.0),10^(-1.5),'O(dx2)')
        text(1.2*10^(2.0),10^(-3.5),'O(dx3)')
        text(1.2*10^(2.0),10^(-7.0),'O(dx4)')
        text(1.2*10^(2.0),10^(-9.0),'O(dx5)')
        axis([10^1 10^3 10^(-10) 10^(0)])
    elseif i == 2
        X = ceil(1./(0.1.^(1.2:0.05:2.5)));
        loglog(X,10^(-0.1)*(1./X).^(3/4),'--b');
        loglog(X,10^(0.20)*(1./X).^(2/3),'--c');
        text(1.1*10^(2.0),10^(-1.1),'O(dx2/3)')
        text(1.1*10^(2.0),10^(-1.9),'O(dx3/4)')
        axis([10^1 10^3 10^(-2) 10^(0)])
    end
    hold off
    matlab2tikz(['Figure',num2str(i),'.tikz'], 'height', '\figureheight', 'width', '\figurewidth');
end
```

```
function [k,X,nX,T,nT] = Setup(h)
k = 0.05*h;
X = 0:h:1;
nX = numel(X);
T = 0:k:1;
nT = numel(T)-1;
end
```

```
function [ U ] = InitialCondition(fun, X, h )
U = zeros(size(X));
dx = 0.5*h;
for i = 1:numel(X)
    U(i) = integral(fun,X(i)-dx,X(i)+dx,'AbsTol',1e-4)/h;
end
end
```

```
function [ u ] = Shock(x)
u = zeros(size(x));
for i = 1:numel(x)
    if x(i) <= 0.5
        u(i) = 1;
    else
        u(i) = 0;
    end
end
end
```

```
function [ u ] = Smooth(x)
u = sin(2*pi*x);
end
```

```
function [ U ] = SSPRK3ENO(C,h,nX,X,U,K,k)

% Do first step in the Runge-Kutta
Y1 = zeros(size(U));Uedge = ENO(C,h,nX,X,U,K);
for j = 1:nX
    Y1(j) = U(j)-k/h*(LaxFlux(h,k,Uedge(2*j+1),Uedge(2*j+2))-...
        LaxFlux(h,k,Uedge(2*j-1),Uedge(2*j-0)));
end

% Do second step in the Runge-Kutta
Y2 = zeros(size(U));Y1edge = ENO(C,h,nX,X,Y1,K);
for j = 1:nX
    Y2(j) = (3/4)*U(j)+0.25*Y1(j)-0.25*k/h*(LaxFlux(h,k,Y1edge(2*j+1),Y1edge(2*j+2))-...
        LaxFlux(h,k,Y1edge(2*j-1),Y1edge(2*j-0)));
end

% Do the third step in the Runge-Kutta
Y2edge = ENO(C,h,nX,X,Y2,K);
for j = 1:nX
    U(j) = (1/3)*U(j)+(2/3)*Y2(j)-(2/3)*k/h*(LaxFlux(h,k,Y2edge(2*j+1),Y2edge(2*j+2))-...
        LaxFlux(h,k,Y2edge(2*j-1),Y2edge(2*j-0)));
end

end
```

```
function [ flux ] = LaxFlux(h,k,a,b)
flux = 0.5*(1/1)*(a-b)+0.5*(f(a)+f(b));
end
```

```matlab
function [Uedge,Xedge,stencilIndex,stencilR] = ENO(C,h,nX,X,u,K)

% Preallocate memory
Uedge           = zeros(2*nX+2,1);
stencilIndex    = zeros(K,nX+2);
stencilR        = zeros(1,nX+2);

% Add ghost points, modify this line according to initial condition type
Ug = [u(nX-K:nX-1);u;u(2:K+1)];

% Loop over every cell, at each cell find and save the stencil leading to
% the smoothest approximation
for i = (K):(K+nX+1)
    s = 0;
    r = 0;
    for k = 1:K-1
        % Add stencil point r
        Vr = DivDiff(Ug(i-r-1:i+s));
        % Add stencil point s
        Vs = DivDiff(Ug(i-r:i+s+1));
        if abs(Vr) > abs(Vs)
            s = s + 1;
        else
            r = r + 1;
        end
    end
    % Save stencil
    stencilIndex(:,i-K+1) = (i-r):1:(i+s);
    stencilR(i-K+1)       = r;
end

% Use the stencils and coefficients to approximate values at cell boundaries
Uedge(1)            = sum(C(stencilR(1)+2,:)'.*Ug(stencilIndex(:,1)));
Uedge(2:2:(2*nX))   = sum(C(stencilR(2:nX+1)+1,:)'.*Ug(stencilIndex(:,2:nX+1)));
Uedge(3:2:(2*nX+1)) = sum(C(stencilR(2:nX+1)+2,:)'.*Ug(stencilIndex(:,2:nX+1)));
Uedge(end)          = sum(C(stencilR(end)+1,:)'.*Ug(stencilIndex(:,end)));

% Pass this, somebody might want it..
Xedge           = reshape([X-h/2,X(end)+h/2;X-h/2,X(end)+h/2],2*nX+2,1);
```

```matlab
function [ v ] = DivDiff( V )
if numel(V) == 1
    v = V;
else
    v = DivDiff(V(2:end)) - DivDiff(V(1:end-1));
end
end
```

```matlab
function [ flux ] = f( u )
flux = u;
end
```

```
function [ C ] = GetC(K)
% Calculate matrix with interpolation coefficients
C = zeros(K+1,K);
for r = -1:1:K-1
    for j = 0:K-1
        temp = 0;
        for m = j+1:K
            % Sum denominator
            C1 = 0;
            for l = 0:K
                if l ~= m
                    temp1 = 1;
                    for q = 0:K
                        if (q ~= m) && (q ~= l)
                            temp1 = temp1*(r-q+1);
                        end
                    end
                    C1 = C1 + temp1;
                end
            end
            % Sum numerator
            temp2 = 1;
            for l = 0:K
                if l ~= m
                    temp2 = temp2*(m-l);
                end
            end
            C2 = temp2;
            % Devide and add
            temp = temp + C1/C2;
        end
        C(r+2,j+1) = temp;
    end
end
end
```