Solution proposal to Project 2:

The WENO method

Question 2.1 (a) See SHU 1998. (b) In the ENO method for reconstruction of cell boundaries we spend a lot of computational effort for each cell to create and compare various stencils of length $K$ before finally choosing one of them. The idea with the WENO method is, instead of spending so much effort comparing stencils and choosing the most appropriate, lets just take all of them and instead spend computational effort on calculating some set of weights to be used at each cell, this has the added advantage of leading to $O(dx^{2k-1})$ accurate approximations in smooth regions.

Question 2.2 (a)(b) See DEMO script and functions attached. (c) The initial conditions are depicted in figure 1. The solution for each initial condition at $T=2$ is presented in figure 2 for $K = 2$ and in figure 3 for $K = 3$. We notice that for $K = 2$ everything looks nice and smooth but for $K = 3$ in the case of shock initial condition oscillations appear. When using a Lax-Friedrichs type flux, the large numerical diffusion dampens these oscillations, when using the Godunov’s type flux with Roe’s approximate Riemann solver, there is less artificial diffusion and the oscillations appear larger. (d) The first initial condition is smooth and remains smooth throughout the solution, the second initial condition is also smooth but two discontinuities arise during the solution procedure. The third initial condition contains two shocks and throughout the solution to this initial conditions discontinuities are presented. (e) See DEMO script and functions attached and figure 4 containing the accuracy results. (f) For the case of the smooth initial condition, initial condition 1, we measure third order accuracy. For the two initial conditions, 2 and 3, containing discontinuities, the accuracy degenerates to first order. (g) Yes. (f) First order. (i) Also first order, but the error constant is substantially better then when solving initial condition 3.

Question 2.3 (a) Done in previous project. (b) See DEMO script and functions attached. (c) See figure 4 containing the accuracy results (d) When using WENO reconstruction, the difference in accuracy between using a lax-Friedrich’s type flux and using a Godunov’s type with roes approximate Riemann matrix is qualitatively the same as when simply approximating cell interfaces with cell averages. There is an error constant difference between the Lax-Friedrichs flux and the Godunov’s flux, the Godunov’s flux is subject to less numerical diffusion than the Lax-Friedrichs type flux.
Figure 1: The initial conditions $q(x, 0)$. 

(a) Initial condition 1

(b) Initial condition 2

(c) Initial condition 3
Figure 2: Solution at $T = 2$ using $\Delta x = 0.05$ with $K = 2$. 
Figure 3: Solution at $T = 2$ using $\Delta x = 0.05$ with $K = 3$. 
Figure 4: Accuracy as measured in the 1-norm against a Roe’s method solution using 2000 grid cells with $K = 2$ on initial condition (a) 1, (b) 2, (c) 3.
clear all,clc

% In this DEMO script for project 2 we attempt to solve the one dimensional shallow
% water equation. The problem can be formulated as a conservation law in the form
% of a system of two coupled nonlinear partial differential equations. The solution
% can be visualized and the accuracy of the used method can be measured.

% Visualize and estimate accuracy
Visualize = 0;
Accuracy = 1;

% Numerical flux functions available
types = {'Lax-Friedrichs','Roes-Method'};

% Physical constants
g = 1;

% Choose stencil size to use K = 2(3) or K = 3(5)
K = 2;
C = GetC(K);

% Choose initial dataset 1 to 3
data = 1;

% Do magic
if Visualize
    view = [ 0 2 -0.5 2 ];
    h = 0.05;
    [k,X,nX,T,nT] = Setup(h);
    for i = 1:numel(types)
        Q = InitialData(data,X,nX,h/2);
        figure(i);
        plot(X,Q(1,:),'-b',X,Q(2,:)./Q(1,:),'-r');
        axis(view);
        grid on;
        xlabel('x');legend('h','u');
        %matlab2tikz([num2str(data),'_Initial.tikz'], 'height', 'igureheight', 'width', 'igurewidth');
        Qedge = zeros(2,2*nX+2);
        for t = 1:nT
            Q = SSPRK3(F,C,h,nX,X,Q,K,g,k,types{i});
            plot(X,Q(1,:),'-b',X,Q(2,:)./Q(1,:),'-r');
            axis(view)
            drawnow
        end
        [Qref,Xref] = GetRef(data,g,K);
        Qplot(1,:) = interp1(Xref,Qref(1,:),X);
        Qplot(2,:) = interp1(Xref,Qref(2,:),X);
        hold on;plot(X,Qplot(1,:),'-b',X,Qplot(2,:)./Qplot(1,:),'-r');
        xlabel('x');legend('h','u','ref');grid on;
        name = [types{i},'d',num2str(data),'k',num2str(K),'_Solution.tikz'];
        matlab2tikz(name,'height', 'igureheight', 'width', 'igurewidth');
    end
end

if Accuracy
    % Calculate ref solution
    [Qref,Xref] = GetRef(data,g,K);

    % Measure accuracy of methods
    figure(numel(types)+1);
    n = 10.^(1:8:1:2:0);
    h = 2./cell(n);
    error = zeros(numel(h),1);
    cells = zeros(numel(h),1);
    slope = zeros(numel(types),2);
    legen = {'-ob','--or','-k'};
    for i = 1:numel(types)
        for j = 1:numel(h)
            [k,X,nX,T,nT] = Setup(h(j));
            % Do magic
        end
    end
end
Q = InitialData(data,X,nX,h(j)/2);  
F = zeros(2,nX+1);  
for t = 1:nT  
Q = SSPRK3(F,C,h(j),nX,X,Q,K,g,k,types{i});  
end  

% Measure error  
error(j) = norm(interp1(Xref,Qref(1,:),X) - Q(1,:),1)/nX;  
cells(j) = nX;  
end  

% Plot  
slope{i,:} = polyfit(log(cells{end-1:end}),log(error(end-1:end)),1);  
loglog(cells,error,legen{i});hold on;  
end  

switch data  
  case 1  
    ebar = 10^(2.3)*1./n.^3;  
    loglog(n, ebar, legen{3})  
    axis([10^1 2*10^2 10^(-5) 10^(-1)])  
    lgdstring = 'O(dx3)';  
  case 2  
    ebar = 10^(-0.4)*1./n.^1;  
    loglog(n, ebar, legen{3})  
    axis([10^1 2*10^2 10^(-3) 10^(-1)])  
    lgdstring = 'O(dx1)';  
  case 3  
    ebar = 10^(0.5)*1./n.^1;  
    loglog(n, ebar, legen{3})  
    axis([10^1 2*10^2 10^(-2) 10^(-0)])  
    lgdstring = 'O(dx1)';  
end  
xlabel('Cells');ylabel('Error');grid on;  
legend(types{1},types{2},lgdstring);  
slope(:,1)  
name = ['data_',num2str(data),'k',num2str(K),'_Acc.tikz'];  
matlab2tikz(name, 'height', 'igureheight', 'width', 'igurewidth');  
end

function [ k,X,nX,T,nT ] = Setup(h)  
k = 0.5*h;  
X = 0:h:2;  
nX = numel(X);  
T = 0:k:2;  
nT = numel(T)-1;  
end

function Q = InitialData(data,X,nX,dx)  

% Initial pressure distribution  
switch data  
  case 1  
    H = @(x) 1 - 0.1*sin(pi*x);  
    HU = @(x) zeros(size(x));  
  case 2  
    H = @(x) 1 - 0.2*sin(2*pi*x);  
    HU = @(x) 0.5*ones(size(x));  
  case 3  
    H = @(x) (x<1) .* 1.5 + (x>=1) .* 0.5;  
    HU = @(x) zeros(size(x));  
end  

% Integrate data  
Q = zeros(2,nX);  
for i = 1:numel(X)  
  Q(1,i) = integral(H,X(i)-dx,X(i)+dx,'AbsTol',1e-10)/(dx*2);  
  Q(2,i) = integral(HU,X(i)-dx,X(i)+dx,'AbsTol',1e-10)/(dx*2);  
end  
end
function [ Qref,Xref ] = GetRef(data,g,K)

Name = ['Qref_','num2str(data),'_k',num2str(K),'.mat'];
if exist(Name, 'file') == 2;
    load(Name);
else
    K = 2;
    C = GetC(K);
    h = 0.001;
    wstring = 'Computing reference solution';
    wh = waitbar(0,wstring);
    [k,Xref,nX,T,nT] = Setup(h);
    Qref = initialData(data,Xref,nX,h/2);
    F = zeros(2,nX+1);
    for t = 1:nT
        Qref = SSPRK3(F,C,h,nX,Xref,Qref,K,g,k,'Roes-Method');
        waitbar(t/nT);
    end
    close(wh);
    save(Name,'Qref','Xref');
end
end

function [ Q ] = SSPRK3(F,C,h,nX,X,Q,K,g,k,type)

% Do first step in the Runge-Kutta
Qedge(1,:) = WENO(C,h,nX,X,Q(1,:),K);
Qedge(2,:) = WENO(C,h,nX,X,Q(2,:),K);
F = Flux(F,Qedge,nX,g,h,k,type);
Y1 = Q(:,1)−k/h*(F(:,2:nX+1)−F(:,1:nX));

% Do second step in the Runge-Kutta
Y2edge(1,:) = WENO(C,h,nX,X,Y1(1,:),K);
Y2edge(2,:) = WENO(C,h,nX,X,Y1(2,:),K);
F = Flux(F,Y2edge,nX,g,h,k,type);
Y2 = (3/4)*Q(:,1)+0.25*Y1−0.25*k/h*(F(:,2:nX+1)−F(:,1:nX));

% Do the third step in the Runge-Kutta
Y3edge(1,:) = WENO(C,h,nX,X,Y2(1,:),K);
Y3edge(2,:) = WENO(C,h,nX,X,Y2(2,:),K);
F = Flux(F,Y3edge,nX,g,h,k,type);
Q = \frac{1}{3}*Q(:,1)+(2/3)*Y2−(2/3)*k/h*(F(:,2:nX+1)−F(:,1:nX));
end

function F = Flux(F,Qedge,nX,g,h,k,type)

switch type
    case 'Lax-Friedrichs'
        for i = 1:nX+1
            a = Qedge(:,2*i-1);
            b = Qedge(:,2*i);
            F(:,i) = 0.5*(h/k)*(a−b)+0.5*(f(a,g)+f(b,g));
        end
    case 'Roes-Method'
        for i = 1:nX+1
            Ql = Qedge(:,2*i-1);
            Qr = Qedge(:,2*i);
            uavg = (Ql(2)/sqrt(Ql(1)+Qr(2))+Qr(2)/sqrt(Ql(1)+Qr(1)))/(sqrt(Ql(1)+Qr(1)));
            cavg = sqrt(g*(Ql(1)+Qr(1))/2);
            L = [uavg−cavg;0;0,uavg+cavg];
            A = [1,1;L(1,1),L(2,2)] + abs(L) = 0.5/cavg*[L(2,2),−1;−L(1,1),1];
            F(:,i) = 0.5*(f(Ql,g)+f(Qr,g)) = 0.5*A*(Qr−Ql);
        end
end
function f = f(Q,g)
f = [Q(2,:);Q(2,:).^2./Q(1,:)+0.5*g*Q(1,:).^2];
end

function [Uedge,Xedge] = WENO(C,h,nX,X,u,K)
% Add ghost points, modify this line according to initial condition type
Ug = [u(1:nX−K+1),u,u(2*K+1)];

% Calculate the values at the edges of each cell for each of the K stencils
U = zeros(2*nX+2,K);
for j = 1:K
    r = −1−1;
    U(1,j) = sum( C(r+2,:) .* Ug((K−r):(2*K−1−r)) );
    for i = (K+1):(nX+K)
        U((i−K)*2+1,j) = sum( C(r+2,:) .* Ug((i−r):(i+K−1−r)) );
    end
    U(2*nX+2,j) = sum( C(r+1,:) .* Ug((nX+K+1−r):(nX+K+1+K−1−r)) );
end

% Find weights Wedge for Uedge
W = FindWeights(nX,Ug,K);

% Apply weights and return Uedge
Uedge = zeros(2*nX+2,1);
for i = 1:(2*nX+2)
    Uedge(i) = sum(U(i,:).*W(i,:));
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);
end

function [ Wedge ] = GetW(K,A,B,d,e)
Wedge = zeros(1,K);
for j = 1:K
    A(j) = d(j)/((e+B(j))^2);
end
for j = 1:K
    Wedge(1,j) = A(j)/sum(A);
end
end

function [ B ] = GetB(B,U,K)
% Return Beta coefficients
switch K
    case 2
        B(1) = (U(3)−U(2))^2;
        B(2) = (U(2)−U(1))^2;
    case 3
        B(1) = (13/12)*(U(3)−2*U(4)+U(5))^2 + 0.25*(3*U(3)−4*U(4)+U(5))^2;
        B(2) = (13/12)*(U(2)−2*U(3)+U(4))^2 + 0.25*(U(2)−U(4))^2;
        B(3) = (13/12)*(U(1)−2*U(2)+U(3))^2 + 0.25*(U(1)−4*U(2)+3*U(3))^2;
    otherwise
        error('Only WENO coefficients for K = {2,3} available');
end
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

```matlab
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;
    % Divide and add
    temp = temp + C1/C2;
end
C(r+2,j+1) = temp;
end
end
end

function [ d ] = GetD(K)
% Return d coefficients
d = zeros(1,K);
switch K
    case 2
        d(1) = 2/3;
        d(2) = 1/3;
        d(1) = 3/4;
        d(2) = 1/4;
    case 3
        d(1) = 3/10;
        d(2) = 3/5;
        d(3) = 1/10;
    otherwise
        error('Only WENO coefficients for K = [2,3] available');
end
end
```