MATH-459 Numerical Methods for Conservation Laws by Prof. Jan S. Hesthaven

Solution set 8:

ENO and WENO method

Exercise 8.1 **(a)** See the ENO function attached below. **(b)** See the GetC function attached below. **(c)** See the ENO function attached below. Run the script ENOWENO_DEMO to see the stencil choosing process visualized and the ENO generated cell boundary approximations compared with cell boundary approximations using fixed, centered and upwind stencils.

Exercise 8.2 **(a)** We start tge calculation by writing P_r

$$P_r(x) = \sum_{j=0}^{k} V(x_{i-r+j-0.5})\, l_{i-r+j-0.5}(x) \tag{1}$$

Here $l_{i-r+j-0.5}(x)$ are the Lagrange polynomials. For $k=2$, we have

$$\begin{aligned} P_r(x) =& \frac{V(x_{i-r-0.5})}{2h^2}(x - x_{i-r+0.5})(x - x_{i-r+1.5}) - \frac{V(x_{i-r+0.5})}{h^2}(x - x_{i-r-0.5})(x - x_{i-r+1.5}) \\ &+ \frac{V(x_{i-r+3.5})}{2h^2}(x - x_{i-r-0.5})(x - x_{i-r+0.5}) \end{aligned}$$

which implies

$$\frac{dp_r}{dx}(x) = \frac{d^2 P_r}{dx^2}(x) = \frac{1}{h^2}\left[V(x_{i-r+1.5}) - 2V(x_{i-r+0.5}) + V(x_{i-r-0.5})\right] = \frac{1}{h}\left(\overline{U}_{i-r+1} - \overline{U}_{i-r}\right) \tag{2}$$

Therefore,

$$\beta_r = h \int_{x_{i-0.5}}^{x_{i-0.5}} \frac{1}{h^2}\left(\overline{U}_{i-r+1} - \overline{U}_{i-r}\right)^2 dx = \left(\overline{U}_{i-r+1} - \overline{U}_{i-r}\right)^2 \tag{3}$$

which completes the proof. **(b)** See function WENO attached below. Run the script ENOWENO_DEMO to see the WENO generated cell boundary approximations compared with cell boundary approximations using fixed, centered and upwind stencils.

```
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
        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
```

```
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
```

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

```
function [ Wedge ] = FindWeights(nX,Ug,K)

% Preallocate memory and get coefficients for for Wedge
Wedge = zeros(2*nX+2,K);
e = 10^(-6);
d = GetD(K);
A = zeros(1,K);
B = zeros(1,K);

% Calculate the right cell boundary at cell k (ghostcell)
B = GetB(B,Ug(1:(2*K-1)),K);
Wedge(1,:) = GetW(K,A,B,d,e);

% Calculate the left and right cell boundary for each cell in the grid
for i = (K+1):(nX+K)
    B = GetB(B,Ug((i-(K-1)):(i+(K-1))),K);
    Wedge((i-K)*2,:) = GetW(K,A,B,fliplr(d),e);
    Wedge((i-K)*2+1,:) = GetW(K,A,B,d,e);
end

% Calculate the left cell boundary at cell nX+k+1 (ghostcell)
B = GetB(B,Ug((nX+2):(nX+2*K)),K);
Wedge(2*nX+2,:) = GetW(K,A,B,fliplr(d),e);
```

```
function [ ] = PlotEdges(tstr,nX,X,u,Xedge,Uedge,i)
figure(i);
barheight   = 0.1/2;
offset      = 0.004;
Color       = {'b','m','r','c','g'};
for i = 1:nX
     plot(X(i),u(i),'ok');hold on
end
for i = 1:nX
     plot([Xedge(2*i);Xedge(2*i+1)],[u(i),u(i)],'-k');
end
for i = 1:2:(2*nX+1)
     plot([Xedge(i);Xedge(i)]-offset,[Uedge(i)-barheight,Uedge(i)+barheight],Color{rem(i,5)+1});
end
for i = 2:2:(2*nX+2)
     plot([Xedge(i);Xedge(i)]+offset,[Uedge(i)-barheight,Uedge(i)+barheight],Color{rem(i+1,5)+1});
end
title(tstr);
axis([-0.5 2.5 -1.5 2.5])
hold off
end
```

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

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

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

% 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);
```

```
function [Uedge,Xedge] = WENO(C,h,nX,X,u,K)

% Add ghost points, modify this line according to initial condition type
Ug = [u(nX-K:nX-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    = j-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,j)   = sum( C(r+1,:)' .* Ug((i-r):(i+K-1-r)) );
        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);
```

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

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

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

% Loop over every cell, at each cell find and save the stencil leading to the smoothest approximation
switch Type
    case 'left'
            s = 0;
            r = (K-1);
    case 'center'
            s = floor((K-1)/2);
            r = ceil((K-1)/2);
    case 'right'
            s = (K-1);
            r = 0;
end
for i = (K):(K+nX+1)
    % Save stencil
    stencilIndex(:,i-K+1) = (i-r):1:(i+s);
    stencilR(i-K+1)       = r;
end

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

end
```

```
clear all,clc
%% ENOWENO_DEMO
% Run this demo script to visualize the ENO reconstruction procedure
% and the ENO WENO cell boundary approximations

% Discretization
h   = 0.05;
k   = 0.5*h;
X   = 0:h:2;
nX  = numel(X);

% Generate something to be approximated
u = zeros(nX,1);
switch 1
    case 1
        u(X<1) = sin(pi*X(X<1));
        u(find(1.5>X & X>=1)) = 1;
        u(X>=1.5) = sin(2*pi*X(X>=1.5));
    case 2
        u = sin(2*pi*X)';
    case 3
        u(X<1) = 1.5;
        u(X>=1) = 0.5;
end

%% Fixed stencils
K = 5;C = GetC(K);

% Plot the edges generated at each cell using an upwind stencil
[Uedge,Xedge] = FIXED(C,h,nX,X,u,K,'left');
name = ['Cell boundary approximation, upwind stencil, K = ',num2str(K)];
PlotEdges(name,nX,X,u,Xedge,Uedge,1)

% Plot the edges generated at each cell using a downwind stencil
[Uedge,Xedge] = FIXED(C,h,nX,X,u,K,'right');
name = ['Cell boundary approximation, downwind stencil, K = ',num2str(K)];
PlotEdges(name,nX,X,u,Xedge,Uedge,2)

% Plot the edges generated at each cell using a centered stencil
[Uedge,Xedge] = FIXED(C,h,nX,X,u,K,'center');
name = ['Cell boundary approximation, centered stencil, K = ',num2str(K)];
PlotEdges(name,nX,X,u,Xedge,Uedge,3)

%% ENO Method

% Plot the edges generated at each cell using the ENO method
[Uedge,Xedge,stencilIndex,stencilR] = ENO(C,h,nX,X,u,K);
name = ['Cell boundary approximation, with ENO method, K = ',num2str(K)];
PlotEdges(name,nX,X,u,Xedge,Uedge,4)

% Visualize ENO procedure
figure(5);
Ug = [u(nX-K:nX-1);u;u(2:K+1)];
Xg = [(X(1)-h*K):h:(X(1)-h),X,(h+X(end)):h:(X(end)+h*K)];
for i = 1:nX+2
    plot(Xg(1:K+1),Ug(1:K+1),'-ok','MarkerSize',10);hold on
    plot(Xg(nX+K:end),Ug(nX+K:end),'-ok','MarkerSize',10);
    plot(Xg(K+1:nX+K),Ug(K+1:nX+K),'-og','MarkerSize',10);
    plot(Xg(stencilIndex(:,i)),Ug(stencilIndex(:,i),1),'or','MarkerSize',10);
    plot(Xg(i+K-1),Ug(i+K-1),'xb','MarkerSize',16);
    grid on;title('ENO choosing stencils');drawnow;pause(0.25)
    hold off
end

%% WENO Method (K = 2 & 3) only
K = 2;C = GetC(K);

% Plot the edges generated at each cell using the WENO method
[Uedge,Xedge] = WENO(C,h,nX,X,u,K);
name = ['Cell boundary approximation, with WENO method, K = ',num2str(K)];
PlotEdges(name,nX,X,u,Xedge,Uedge,6)
```