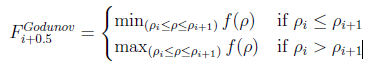
1. *Implement in Matlab the Godunov flux scheme.*

The Godunov flux function is given in Equation 1:

 (1)

The finite volume approach was used to discretize the domain, calculate elemental fluxes, use the flux function to calculate nodal fluxes across the domain, and then apply a time step to calculate the next step’s density values based on nodal fluxes and present densities. The code can be found in Appendix A. Results for the system at t=4 are presented in Figure 1, just as the “light” at x=0 turns green.



Figure : Instantaneous densities in domain based on Godunov flux function, t=4

2. *In the same Matlab code, implement a naive averaged flux scheme. How does this naive average scheme compare with the Godunov scheme? Suggest a reason for the poor performance of this scheme.*

The averaged flux function is given in Equation 2:

 (2)

Figure 2 presents two cases. In order to more easily visualize the calculation failures a smaller discrete time interval was chosen. In comparison to the other methods a time interval 10 and 5 times smaller was used. The system never has a chance to reach steady state. Errors continue to build up until eventually the density at a certain point exceeds the maximum float value and becomes ‘Inf’. This out-of-scale value is used in neighboring nodal flux calculations which then return ‘NaN’. The errors propagate upstream until they reach the domain boundary. For a short time densities upstream of the error continue to calculate until another out-of-scale value occurs; the process repeats until the entire domain is invalid.

The poor performance has to do with failing to properly capture the density at the shock interface. The density value fluctuates, creating erroneous propagations upstream and downstream of the shock interface. The density values attempt to “seek” the correct density value but overshoot. This creates a positive feedback loop where local fluctuations reinforce each other. As Figure 2 shows, choosing a smaller dt value delays catastrophic failure at any one point by more accurately capturing the instability in any one volume better, but it only delays the inevitable.





Figure : Average flux function with dt=0.08 dx (top) and dt=0.16 dx (bottom), usually dt=0.8 dx

3. *In the same Matlab code, implement the Lax-Friedrich's flux scheme as an alternative case. How does the L-F scheme compare with the Godunov scheme? Be sure to zoom into the plot of the car density along the x-axis.*

The Lax-Friedrich flux function is given by Equation 3:

 (3)

Figure 3 shows the instantaneous density values at t=4 for the Lax-Friedrich implementation. Qualitatively it seems that the discontinuities are not as sharp as the Godunov implementation. Figure 4 is a zoomed in comparison of the two methods. The discontinuity is visibly “smeared” across a larger space.

The flux function looks functionally similar to the naïve average function method but with and added term. The second portion of the flux function acts as an artificial viscosity that tends to decrease the nodal flux values. Larger density differences between adjacent volumes incur larger flux penalties. This artificial viscosity has the benefit of damping out the unstable fluctuations that are otherwise untenable in the average flux method. This of course comes at the expense of confounding true physical behavior with model behaviors. This is most noticeable at the shock discontinuities where the pressure gradient is largest. Additionally this method adds unwanted dissipative artifacts that are unrepresentative of the true system behavior.



Figure : Instantaneous density values across domain using Lax-Friedrich at t=4



Figure : Artificial viscosity apparent due to Lax-Friedrich flux function



Figure : Effect of artificial viscosity on resolution of shock discontinuity

The greater the damping by the artificial viscosity, the larger the dissipative effects will be. We want this to be large enough to damp oscillations enough so they can’t run away. Further, we’d like to damp out any unwanted fluctuations near discontinuities to more smothly represent the transition. However the dampin shouldn’t be unnecessarily large, otherwise no additional benefit will be realized and the model will just deviate further from physical reality.

Figure 5 presents density values in the area near a discontinuity with varying coefficients for the L-F method in comparison to the Godunov method. Values below ~0.2 damped insufficiently and the soultion diverged with time. While low values failed to fully damp deviations near the discontinuity, an alternate value of 0.35 more closely agreed with the Godunov solution than the default value of 0.5 while still fully damping fluctuations.

4. *In the same Matlab code, implement the Richtmyer flux scheme as a third alternative case. How does this Richtmyer scheme compare with the previous two?*

The Richtmyer flux function is given by Equation 4:

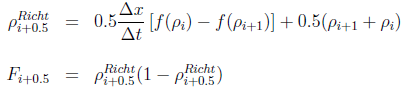
 (4)

Figure 6 shows the performance of the Richtmyer implementation. In comparison to either method it appears to resolve the discontinuity the best, with some time lag for a portion of the shock visible as a small dip, perhaps due to dispersive effects.



Figure : Shock resolution of Richtmyer flux function

5. *Of the four flux functions you examined, which appears to be (a) the most accurate for this problem, (b) the most dissipative and (c) the most efficient?*

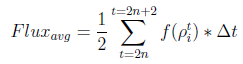
a) Godunov appears to be the most accurate resolving the discontinuity without noticeable dispersive or dissipative effects.

b) Lax-Friedrich appears to be the most dissipative, showing noticeable smearing of the discontinuity due to the artificial viscosity introduced.

c) While the average method might be the least computationally intensive, the results are unusable. The case checking in Godunov is prohibitive. Lax-Friedrich and Richtmyer have similar complexity, with Richtmyer resolving the discontinuity the best.

6. *Using the Godunov scheme, maintain the traffic light at x = 0 and add a second traffic light at x = 0.25. Assuming that the second light alternates between red for 1 unit of time and then green for 1 unit of time, determine the optimal time lag between the two lights to maximize the average flow of cars along the road.*

The average flux at steady state is given by Equation 5.

 (5)

A range of values were trialed for the offset of the second light from the triggering of the first one. Figure 7 presents the range of offsets tested. Past 2 time units periodicity occurs and so offsets past this don’t need to be tested. The maximum average flux occurred when the offset was between ~-0.05-0.05 time units, with the worst performance occuring at ~1.6 time units offset. Transition in behaviors occur at critical times such as when the expansion wave just misses the second light, or where the offset falls within the window of travel time between the two lights.

Interestingly the timing doesn’t have much effect ultimately on the range of realizable average flux. This is because the steady state flow mostly depends on the duty cycle of the worst obstruction. The average flux for just the one light at x=0 was the same as the optimal two light case. For largely disparate duty cycles for two lights the average flux is strongly coupled to the dominant light, with offset changes for the other having little effect. In situations where the two light have similar but non-equal duty cycles or where the frequency of light switching differs significant gains can be made by proper tuning.



Figure : Determination of optimal light triggering

**Appendix A – Program Code**

%Nodes are numbered as follows:

% 1 2 3 4

% O==O==O==O==O

%The element is associated with the node to its right, they share the same

%'i'

close all

clear all

clc

rhoMax= 1.0;

rhoL= 0.8;

rhoR= 0.0;

uMax= 1.0;

dx= 1/100;

dt= 0.8\*dx/uMax;

TimeIncrements= 9/dt; %Mimimum time run to ensure steady state at all points for moving average of flux

xLeft= -1;

xRight= 1;

%Allow user to select flux function

[FluxFunction,ok] = listdlg('PromptString','Select flux function:',...

'SelectionMode','single',...

'ListString',{'Godunov';'Average';'Lax-Friedrich';'Richtmyer'},...

'ListSize',[160 100]);

if FluxFunction==2 %Set finer timestep for average to see errors propogate

warndlg('Setting smaller dt to show how errors propogate','Attention')

key=2;

pause(1)

dt=dt/10;

TimeIncrements= 2/dt;

new=0;

end

% Spatial Discretization

x=xLeft:dx:xRight;

% Initial Conditions

mid=(length(x)-1)/2;

rho(1:mid+1)=rhoL;

rho(mid+2:2\*mid+1)=rhoR;

%Pre-define matrices for speed

saved= zeros(1,2/dt);

FluxNodal= zeros(1,2\*mid);

%Flux triggers

lights(1,:)= [2/dt, 1/dt, mid+1, 0]; %Sets flux=0 with [freq, duty cycle, x\_loc, offset]

freq=1;

trig=2;

loc=3;

offset=4;

counter=0;

%To optimize light2 offset uncomment this

%for iter=0:.005:2

total=0;

counter=counter+1;

%To optimize light2 offset uncomment this

%lights(2,:)= [2/dt, 1/dt, mid+1+(.25/dx), iter/dt];

for t=0:TimeIncrements;

% Calculate flux inside elements

FluxElemental = uMax.\*rho.\*(1-rho./rhoMax);

% Spatial loop

for i= 1:length(x)-1;

switch FluxFunction

case 1 %Godunov

if rho(i)<rho(i+1) %Choked downstream

FluxNodal(i) = min(FluxElemental(i), FluxElemental(i+1));

elseif rho(i)>=rho(i+1) %Unchoked

if rho(i)>rhoMax/2 && rho(i+1)<rhoMax/2 %Check to see if quadratic peak bounded between

FluxNodal(i) = rhoMax\*uMax/4;

else

FluxNodal(i) = max(FluxElemental(i), FluxElemental(i+1));

end

end

case 2 %Average

FluxNodal(i)=mean([FluxElemental(i), FluxElemental(i+1)]);

case 3 %Lax-Friedrich

FluxNodal(i)=0.5\*(FluxElemental(i)+FluxElemental(i+1))-0.5\*((dx/dt)\*(rho(i+1)-rho(i)));

case 4 %Richtmyer

rhoRicht= ((dx/dt)\*(FluxElemental(i)-FluxElemental(i+1))+rho(i+1)+rho(i))/2;

FluxNodal(i)= rhoRicht\*(1-rhoRicht);

end

end

%Trigger for each of the "lights" with a particular frequency, at a

%specified time and place

for l=1:size(lights,1)

if mod(t-lights(l,offset),lights(l,freq)) >= lights(l,trig)

FluxNodal(lights(l,loc)) = 0;

end

end

% Calculate moving average of flux through desired element

%Continual moving average method

saved(floor(mod(t,2/dt)+1))= FluxElemental(50);

FluxAvg= sum(saved)\*(dt/2);

%Average at "end" of time period

% if t>=375

% total= total+FluxElemental(50);

% end

% Spatial loop for new Rho vals

for i = 2:length(x)-1

rho(i) = rho(i) - dt/dx\*(FluxNodal(i) - FluxNodal(i-1));

end

plot(x, rho,'o-')

hold on

plot([-1,1],[1.2,-.2],'w.')

hold off

text(1,1,num2str(t\*dt));

text(1,.8,num2str(FluxAvg));

pause(0.001)

end

%To optimize light2 offset uncomment this

% FluxAvg2=total\*(dt/2);

% savedAvg(counter)= FluxAvg;

% end

% plot(0:.005:2,savedAvg./max(savedAvg))