MATH 228b: HW4

April Novak

March 16, 2017

1

The continuity equation is used to model the density of cars:

$$\frac{\partial \rho}{\partial t} + \frac{\partial f(u,\rho)}{\partial x} = 0 \tag{1}$$

where ρ is the density of cars and $f(u)$ the flux of cars, which is a function of the car velocity u and density. A road is treated as a continuum, such that no individual cars are modeled. Integrating the above over $x_{i-1/2} \leq x \leq x_{i+1/2}$, where a node-centered finite volume method is to be used, gives:

$$\begin{aligned} \int_{x_{i-1/2}}^{x_{i+1/2}} \left(\frac{\partial \rho(x,t)}{\partial t} + \frac{\partial f(u,\rho)}{\partial x} \right) dx =& 0 \\ \frac{\partial}{\partial t} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t) dx + \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial f(u,\rho)}{\partial x} dx =& 0 \\ \frac{\partial}{\partial t} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t) dx + f(\rho(x_{i+1/2}),u) - f(\rho(x_{i-1/2}),u) =& 0 \end{aligned} \tag{2}$$

Then, integrating in time over $t_n \leq t \leq t_{n+1}$ gives:

$$\begin{aligned} \int_{t_n}^{t_{n+1}} \left(\frac{\partial}{\partial t} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t) dx \right) dt + \int_{t_n}^{t_{n+1}} \Big(f(\rho(x_{i+1/2}),u) - f(\rho(x_{i-1/2}),u) \Big) dt =& 0 \\ \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t_{n+1}) dx - \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t_n) dx + \int_{t_n}^{t_{n+1}} \Big(f(\rho(x_{i+1/2}),u) - f(\rho(x_{i-1/2}),u) \Big) dt =& 0 \end{aligned} \tag{3}$$

Multiplying through by $1/\Delta x$, or the inverse of the mesh spacing:

$$\frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t_{n+1}) dx - \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t_n) dx + \frac{1}{\Delta x} \int_{t_n}^{t_{n+1}} \Big(f(\rho(x_{i+1/2}),u) - f(\rho(x_{i-1/2}),u) \Big) dt = 0 \tag{4}$$

Then, we can define the solution cell i spatial average as P:

$$P_i^n \equiv \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} \rho(x,t_n) dx \tag{5}$$

Likewise, the numerical flux is defined as the flux cell i temporal average:

$$F_{i-1/2}^n \equiv \frac{1}{\Delta t} \int_{t_n}^{t_{n+1}} f(\rho(x_{i-1/2}),t) dt \tag{6}$$

Inserting these into Eq. (4) gives the fundamental structure of the finite volume method. At this point, the numerical flux can be defined by a large magnitude of different methods. Godunov's method, which is

essentially equivalent to an upwinding method, and Roe's method, which consists of an arithmetic average plus a stabilizing correction term, will be investigated in this question.

$$\begin{aligned} P_i^{n+1} - P_i^n + \frac{\Delta t}{\Delta x}\left(F_{i+1/2} - F_{i-1/2}\right) &= 0 \\ P_i^{n+1} &= P_i^n - \frac{\Delta t}{\Delta x}\left(F_{i+1/2} - F_{i-1/2}\right) \end{aligned} \tag{7}$$

Godunov's method specifies the numerical flux according to the value of the solution that would be present at the original Riemann discontinuity location after ϵ time has elapsed. For scalar conservation equations, this essentially reduces to upwinding.

$$\begin{aligned} F_{i+1/2} &= \begin{cases} \min_{\rho_i \le \rho \le \rho_{i+1}} f(\rho) & \rho_i < \rho_{i+1} \\ \max_{\rho_i \ge \rho \ge \rho_{i+1}} f(\rho) & \rho_i > \rho_{i+1} \end{cases} \\ F_{i-1/2} &= \begin{cases} \min_{\rho_{i-1} \le \rho \le \rho_i} f(\rho) & \rho_{i-1} < \rho_i \\ \max_{\rho_{i-1} \ge \rho \ge \rho_i} f(\rho) & \rho_{i-1} > \rho_i \end{cases} \end{aligned} \tag{8}$$

Roe's method combines an arithmetic average of two neighboring fluxes based on cell-centered values with a stabilizing correction term, giving the following numerical fluxes:

$$\begin{aligned} F_{i+1/2} &= \frac{1}{2}\left(f(\rho_i) + f(\rho_{i+1})\right) - \frac{1}{2}u_{max}\left|\left(1 - \frac{\rho_i + \rho_{i+1}}{\rho_{max}}\right)\right|(\rho_{i+1} - \rho_i) \\ F_{i-1/2} &= \frac{1}{2}\left(f(\rho_{i-1}) + f(\rho_i)\right) - \frac{1}{2}u_{max}\left|\left(1 - \frac{\rho_{i-1} + \rho_i}{\rho_{max}}\right)\right|(\rho_i - \rho_{i-1}) \end{aligned} \tag{9}$$

These two models are both used to model a traffic light turning green at $t = 0$. The initial condition specifies $\rho_{-1/2}$, and in order to allow the simulation to evolve appropriately, a domain from $-5, 5$ is used (this should be increased if a longer time duration is studied). The density cannot simply be held at ρ_L at the traffic light, since this would not allow a rarefaction wave to develop, and so the actual boundary condition applied at each time step is to set the density at the very first spatial node (at $x = -5$, outside the range of interest) to ρ_L. The simulation results for several time steps are shown below, including $t = 2$.

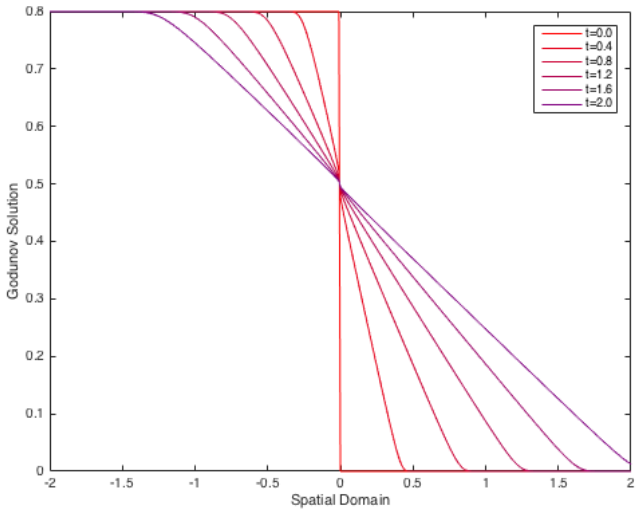


Figure 1. Solution at various time steps for the Godunov method.

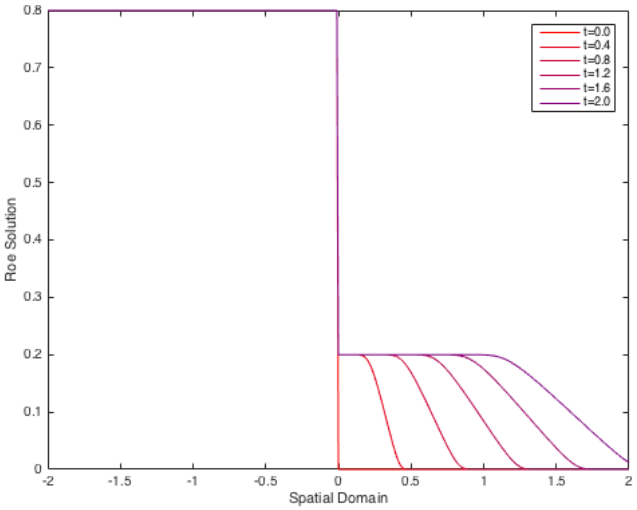


Figure 2. Solution at various time steps for the Roe method.

By superimposing the two above graphs, it can be seen that at the front of the rarefaction wave, the Godunov and Roe solutions agree well with each other - they only give different results in the vicinity of the initial shock. For instance, by comparing the difference to either of the previous two plots, it can be seen that the difference drops to close to zero at around 1.25 for $t = 2$, even though the front of the rarefaction wave reaches out to about 2. So, both schemes capture the behavior far from the shock, but differ in the vicinity of the shock. In addition, there is a slight "bump" in the solution at the location of the traffic light for the Godunov solution, but by refining the mesh in both space and time, this small discontinuity disappeared, and is likely a remnant of the fact that the Godunov method (as well as the Roe method) is simply an approximation to the flux for each finite volume.

The Godunov method gives the more realistic solution than Roe's method because Roe's method fails at resolving rarefaction waves, such as in this test problem. Roe's method is derived as a linearization about the averaged state $(\rho_i + \rho_{i+1})/2$ (for $F_{i+1/2}$). Due to this linearization, the solution will only consist of a single wave, as opposed to the *two* waves that should normally travel from rarefaction waves. So, Roe's method entirely misses the left-going wave that causes the density to decrease to the left of the traffic light. For cases where the solution does not contain a rarefaction wave, Roe's method will perform well, since a shock consists of a single wave, moving either to the left or right. The code for this section is included in the Appendix.

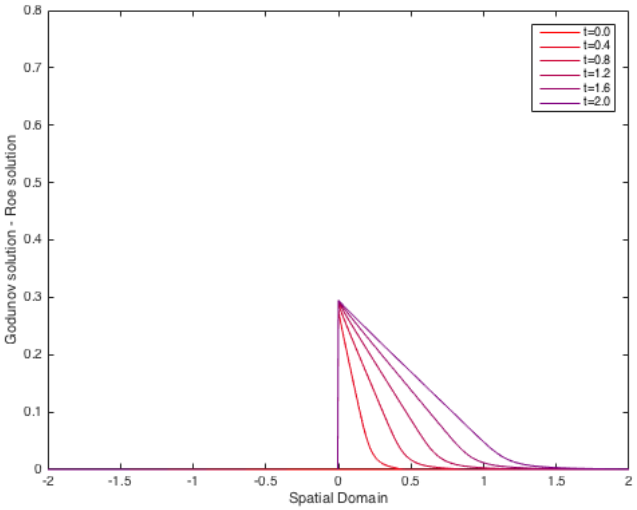


Figure 3. Difference between the Godunov solution and the Roe solution at various time steps.

2

Now, a traffic light is located at $x_{i-1/2}$ for $i = 1$, i.e. the very first node has a traffic light at its left boundary. This traffic light will be simulated by setting $F_{i-1/2} = 0$ for the cell to the right of the traffic light, and $F_{i+1/2} = 0$ for the cell to the left of the traffic light. The figure below shows the solution at several time steps (over one full period) once reaching steady-state. Only results with the Godunov method are shown, since from Question 1, it was shown that Roe's method did not give good results for rarefaction waves. As can be seen from the figure below, the effects of the traffic light are correctly simulated - because the results at $t = 15.0$ exactly overlapped those at $t = 17.0$, results are shown at $t = 15.1$, just after the traffic light has turned red. A plateau of maximum density is reached, indicating that cars are arriving at the light and stopping still because they cannot drive forward. At $t = 16.0$, the light is just about to turn green, and a rarefaction wave develops after the light turns green as cars from behind the light can approach it, and cars beyond it continue to drive away.

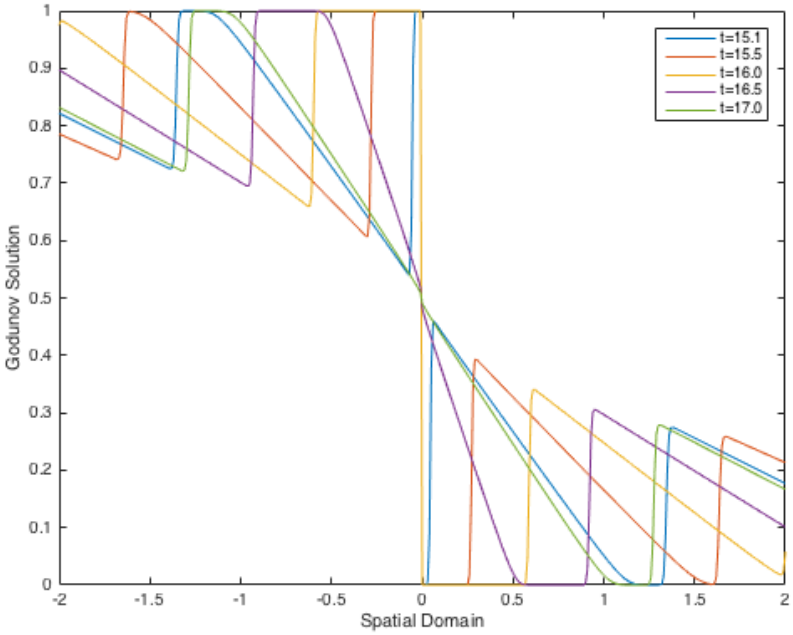


Figure 4. Godunov solution for a traffic light at $x = -\Delta x/2$ with a period of 2, where the green light and red light each occupy half a period.

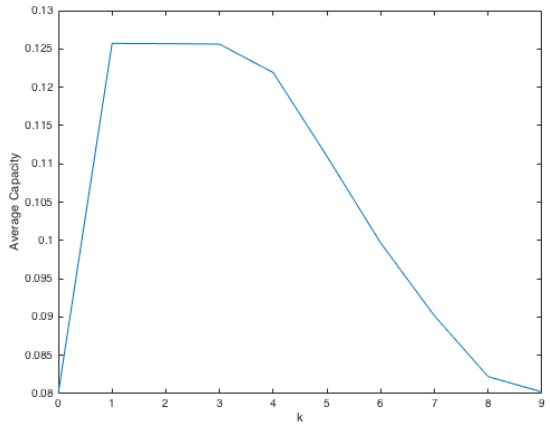
By conservation of the mass of cars per unit length of the road (i.e. continuity), the average flow $\dot{q}$ can be computed at any point in the domain (and this is verified by computing the estimated flow at 50 different points in the domain).

$$\dot{q} = \frac{1}{N_T} \sum_{n=1}^{N_T} f_n \tag{10}$$

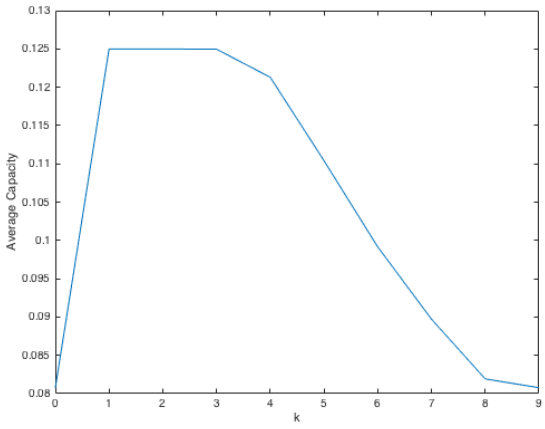
where N_T is the number of time steps per period, which for the time step given in Question 1 is 125. The average flow is computed to be roughly 0.125 cars/length^{2}time^2, where the units are consistent with those given in the problem. The code for this section is included in the Appendix.

3

This question models a domain with two traffic lights. For both lights, the incoming and outgoing numerical fluxes exactly at the light are set to zero when the light is red, and no condition is imposed otherwise. It was discovered that the calculated capacity is a function of the precise switching algorithm used for the lights. For my algorithm, the results are the same regardless of which light begins as red or green, as shown in the figure below. This occurs because the results are measured once reaching steady state, and initial transients related to the switching have died out. The switching is performed by the second light, i.e. the first light switches every half period regardless of the color of the second light. The second light will switch after the delay time steps following a switch of the first light. As can be seen, it appears that $k = 1, 2, 3$ give the highest road capacity, since some cars will be able to approach the second light while it is red and build up such that the density beyond the second light is very low once the second light turns green - this allows roughly the same capacity as was observed in Question 2 with a single stop light. When the lights are fairly out-of-sync, such as for $5 \leq k \leq 9$, cars will spend a relatively long time waiting at the second light even though they have the ability to pass the first light. The code for this section is given in the Appendix.



(a) Left light is green, right light is red at $t = 0$.



(b) Left light is red, right light is green at $t = 0$.

Figure 5. Road average capacity as a function of k, for $\tau = kT/10$.

4 Appendix

4.1 Question 1

4.1.1 `flux.m`

This function calculates the flux given a density.

```
function [f] = flux(p, p_max, u_max)
f = p .* u_max .* (1 - p./p_max);
end
```

4.1.2 `Godunov2.m`

This function calculates the left and right numerical fluxes using the Godunov method.

```
function [F_left, F_right] = Godunov2(left_point, mid_point, right_point, p_max, u_max
    ↪ )
flux_left = flux(left_point, p_max, u_max);
flux_mid = flux(mid_point, p_max, u_max);
flux_right = flux(right_point, p_max, u_max);

if (mid_point < 0.5 && right_point < 0.5) || (mid_point > 0.5 && right_point > 0.5)
    if mid_point < right_point
        F_right = min(flux_mid, flux_right);
    else
        F_right = max(flux_mid, flux_right);
    end
else
    if mid_point < right_point
        F_right = min(flux_mid, flux_right);
    else
        F_right = flux(p_max / 2, p_max, u_max);
    end
end

if (left_point < 0.5 && mid_point < 0.5) || (left_point > 0.5 && mid_point > 0.5)
```

```
    if left_point < mid_point
        F_left = min(flux_left, flux_mid);
    else
        F_left = max(flux_left, flux_mid);
    end
else
    if left_point < mid_point
        F_left = min(flux_left, flux_mid);
    else
        F_left = flux(p_max / 2, p_max, u_max);
    end
end

end
```

4.1.3 Roe.m

This function calculates the left and right numerical fluxes using the Roe method.

```
function [F_left, F_right] = Roe(left_point, mid_point, right_point, p_max, u_max)
flux_left = flux(left_point, p_max, u_max);
flux_mid = flux(mid_point, p_max, u_max);
flux_right = flux(right_point, p_max, u_max);

F_left = 0.5 * (flux_left + flux_mid) - ...
    0.5 * u_max * abs(1 - (left_point + mid_point)/p_max) * (mid_point - left_point);
F_right = 0.5 * (flux_mid + flux_right) - ...
    0.5 * u_max * abs(1 - (mid_point + right_point)/p_max) * (right_point - mid_point)
        ↪ ;
end
```

4.1.4 Q1.m

This function calculates the solution for a single traffic light using both the Godunov and Roe methods.

```
% Q1, HW 4
clear all
p_max = 1.0; u_max = 1.0;
p_left = 0.8; p_right = 0.0;
dx = 4/400;
dt = 0.8 * dx / u_max;
XL = -5; X = 5; T = 2;

mesh = XL:dx:X;
time = 0:dt:T;

backsize = length(mesh(mesh < 0.0));
frontsize = length(mesh) - backsize;

pG = cell(length(time), 1); pR = cell(length(time), 1);
for i = 1:length(time)
    pG{i} = [p_left .* ones(1, backsize), p_right .* ones(1, frontsize)];
    pR{i} = [p_left .* ones(1, backsize), p_right .* ones(1, frontsize)];
end

for t = 1:length(time)
    F_leftG = zeros(size(mesh)); F_rightG = zeros(size(mesh));
    F_leftR = zeros(size(mesh)); F_rightR = zeros(size(mesh));
```

```
    for i = 1:(length(mesh) - 1)
        if i == 1
            left_pointG = p_left;
            left_pointR = p_left;
        else
            left_pointG = pG{t}(i - 1);
            left_pointR = pR{t}(i - 1);
        end

        [F_leftG(i), F_rightG(i)] = Godunov2(left_pointG, pG{t}(i), pG{t}(i+1), p_max,
            ↪ u_max);
        [F_leftR(i), F_rightR(i)] = Roe(left_pointR, pR{t}(i), pR{t}(i+1), p_max,
            ↪ u_max);

    pG{t + 1}(i) = pG{t}(i) - (dt / dx) * (F_rightG(i) - F_leftG(i));
    pR{t + 1}(i) = pR{t}(i) - (dt / dx) * (F_rightR(i) - F_leftR(i));
    end
end

dc = 0.0;
Godunov = false; Roe = false; difference = true;
plottime = 1:50:length(time);
for t = plottime
    Gcolor = [1.0 - dc, 0.0, dc];
    if (Godunov)
        plot(mesh, pG{t}, 'Color', Gcolor)
        ylabel('Godunov Solution')
    elseif (Roe)
        plot(mesh, pR{t}, 'Color', Gcolor)
        ylabel('Roe Solution')
    elseif (difference)
        plot(mesh, pG{t} - pR{t}, 'Color', Gcolor)
        ylabel('Godunov solution - Roe solution')
    else
        plot(mesh, pG{t}, 'g', mesh, pR{t}, 'r')
        drawnow
    end

    hold on
    ylim([0, 0.8])
    xlim([-2, 2])
    xlabel('Spatial Domain')
    dc = dc + 0.1;
end
legend('t=0.0', 't=0.4', 't=0.8', 't=1.2', 't=1.6', 't=2.0')
```

4.2 Question 2

4.2.1 Q2.m

This function calculates the solution for a cycling traffic light using Godunov's method.

```
% Q2, HW 4
clear all
p_max = 1.0; u_max = 1.0;
p_left = p_max/2; p_right = 0.0;
dx = 4/400;
```

```
dt = 0.8 * dx / u_max;
XL = -3; X = 5; T = 30;

mesh = XL:dx:X;
time = 0:dt:T;

% number of time steps while red, equal to number of time steps while green
period = ceil(1/dt);
red = false;

backsize = length(mesh(mesh < 0.0));
frontsize = length(mesh) - backsize;

% bounding nodes for light 1
light1 = [backsize, backsize + 1];

pG = cell(length(time), 1);
for i = 1:length(time)
    pG{i} = [0.0 .* ones(1, length(mesh))];
end

for t = 1:length(time)
    F_leftG = zeros(size(mesh)); F_rightG = zeros(size(mesh));

    if mod(t, period) == 0
        red = ~red;
    end

    for i = 1:(length(mesh) - 1)
        if i == 1
            left_pointG = p_left;
        else
            left_pointG = pG{t}(i - 1);
        end

        [F_leftG(i), F_rightG(i)] = Godunov2(left_pointG, pG{t}(i), pG{t}(i+1), p_max,
            ↪ u_max);

        if (red && (i == light1(1)))
            F_rightG(i) = 0.0;
        end

        if (red && (i == light1(2)))
            F_leftG(i) = 0.0;
        end

    pG{t + 1}(i) = pG{t}(i) - (dt / dx) * (F_rightG(i) - F_leftG(i));
    end
end

plottime = floor([15.1, 15.5, 16, 16.5, 17] ./ dt);
for t = plottime
    plot(mesh, pG{t})
    ylabel('Godunov␣Solution')
    hold on
    ylim([0, 1.0])
    xlim([-2, 2])
    xlabel('Spatial␣Domain')
end
```

```
legend('t=15.1', 't=15.5', 't=16.0', 't=16.5', 't=17.0')

% compute average flow
for location = (backsize + 100):10:(backsize + 500)
    sum = 0;
    for t = (15*period):1:(17*period)
        sum = sum + flux(pG{t}(location), p_max, u_max)./period;
    end
    sprintf('The estimated flow at location %d is %d', location*dx, sum)
end
```

4.3 Question 3

4.3.1 Q3.m

This function calculates the solution for two cycling traffic lights using Godunov's method.

```
% Q3, HW 4
clear all
p_max = 1.0; u_max = 1.0;
p_left = p_max/2; p_right = 0.0;
dx = 4/400;
dt = 0.8 * dx / u_max;
XL = -2; X = 4; T = 28;

mesh = XL:dx:X;
time = 0:dt:T;

% number of time steps while red, equal to number of time steps while green
period = ceil(1/dt);
red1 = false; t_switch1 = 0;
red2 = true;

backsize = length(mesh(mesh < 0.0));
frontsize = length(mesh) - backsize;
back2 = length(mesh(mesh < 0.15));
front2 = length(mesh) - back2;

% bounding nodes for lights
light1 = [backsize, backsize + 1];
light2 = [back2, back2 + 1];

K = 0:9;
Loc = floor(back2 + 100);
averages = [];

for k = K
tau = floor(k * period / 10); % number of time steps to delay the second light

    pG = cell(length(time), 1);
    for i = 1:length(time)
        pG{i} = [0.0 .* ones(1, length(mesh))];
    end

    switch_flag = 0;
    for t = 1:length(time)
        F_leftG = zeros(size(mesh)); F_rightG = zeros(size(mesh));
```

```
        switch_flag = switch_flag + 1;
        if mod(t, period) == 0
            switch_flag = 0;
            red1 = ~red1;
        end

        if switch_flag == tau
            red2 = ~red2;
        end

        for i = 1:(length(mesh) - 1)
            if i == 1
                left_pointG = p_left;
            else
                left_pointG = pG{t}(i - 1);
            end

            [F_leftG(i), F_rightG(i)] = Godunov2(left_pointG, pG{t}(i), pG{t}(i+1),
                ↪ p_max, u_max);

            if (red1 && (i == light1(1)))
                F_rightG(i) = 0.0;
            end

            if (red1 && (i == light1(2)))
                F_leftG(i) = 0.0;
            end

            if (red2 && (i == light2(1)))
                F_rightG(i) = 0.0;
            end

            if (red2 && (i == light2(2)))
                F_leftG(i) = 0.0;
            end

        pG{t + 1}(i) = pG{t}(i) - (dt / dx) * (F_rightG(i) - F_leftG(i));
        end
    end

    pmatrix = [];
    for tt = 1:length(time)
        pmatrix(tt, :) = pG{tt};
    end

    % compute average flow
    averages = [averages, sum(flux(pmatrix(25*period:27*period, Loc), p_max, u_max))./
        ↪  (2*period)];
end

plot(K, averages)
xlabel('k')
ylabel('Average Capacity')
ylim([0.08, 0.13])
```