Michael Cowan

Computational Physics

Problem Set 9

Traffic Flow and Rubber-necking

Part a

In Part a, we are asked to modify the given **traffic.m** program to simulate traffic moving at a low density near the maximum possible speed. To do this, I simply modified the initial density profile to be 0.3 everywhere. As a result, the traffic maintained a constant low density at a high rate of speed for the duration of the run. I also made a minor adjustment to the plotting of the program so that the graph of the density and flow rate stayed open at the end of the program run.

Code (modifications to **traffic.m** in italics):

1. % traffic - Program to solve the generalized Burger
2. % equation for the traffic at a stop light problem
3. clear all; help traffic; % Clear memory and print header
4. %\* Select numerical parameters (time step, grid spacing, etc.).
5. method = menu('Choose a numerical method:', ...
6. 'FTCS','Lax','Lax-Wendroff');
7. N = input('Enter the number of grid points: ');
8. L = 400; % System size (meters)
9. h = L/N; % Grid spacing for periodic boundary conditions
10. v\_max = 25; % Maximum car speed (m/s)
11. fprintf('Suggested timestep is %g\n',h/v\_max);
12. tau = input('Enter time step (tau): ');
13. fprintf('Last car starts moving after %g steps\n', ...
14. (L/4)/(v\_max\*tau));
15. nstep = input('Enter number of steps: ');
16. coeff = tau/(2\*h); % Coefficient used by all schemes
17. coefflw = tau^2/(2\*h^2); % Coefficient used by Lax-Wendroff
18. %\* Set initial and boundary conditions
19. rho\_max = 1.0; % Maximum density
20. Flow\_max = 0.25\*rho\_max\*v\_max; % Maximum Flow
21. % Initial condition is low density everywhere
22. rho = zeros(1,N);
23. *for i=1:N*
24. *rho(i) = 0.3; % Initial density is constant*
25. *end*
26. % Use periodic boundary conditions
27. ip(1:N) = (1:N)+1; ip(N) = 1; % ip = i+1 with periodic b.c.
28. im(1:N) = (1:N)-1; im(1) = N; % im = i-1 with periodic b.c.
29. %\* Initialize plotting variables.
30. iplot = 1;
31. xplot = ((1:N)-1/2)\*h - L/2; % Record x scale for plot
32. rplot(:,1) = rho(:); % Record the initial state
33. tplot(1) = 0;
34. figure(1); clf; % Clear figure 1 window and bring forward
35. %\* Loop over desired number of steps.
36. for istep=1:nstep
37. %\* Compute the flow = (Density)\*(Velocity)
38. Flow = rho .\* (v\_max\*(1 - rho/rho\_max));
39. %\* Compute new values of density using FTCS,
40. % Lax or Lax-Wendroff method.
41. if( method == 1 ) %%% FTCS method %%%
42. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im));
43. elseif( method == 2 ) %%% Lax method %%%
44. rho(1:N) = .5\*(rho(ip)+rho(im)) ...
45. coeff\*(Flow(ip)-Flow(im));
46. else %%% Lax-Wendroff method %%%
47. cp = v\_max\*(1 - (rho(ip)+rho(1:N))/rho\_max);
48. cm = v\_max\*(1 - (rho(1:N)+rho(im))/rho\_max);
49. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im)) ...
50. + coefflw\*(cp.\*(Flow(ip)-Flow(1:N)) ...
51. cm.\*(Flow(1:N)-Flow(im)));
52. end
53. %\* Record density for plotting.
54. iplot = iplot+1;
55. rplot(:,iplot) = rho(:);
56. tplot(iplot) = tau\*istep;
57. %\* Display snap-shot of density versus position
58. plot(xplot,rho,'-',xplot,Flow/Flow\_max,'--');
59. xlabel('x'); ylabel('Density and Flow');
60. legend('\rho(x,t)','F(x,t)');
61. axis([-L/2, L/2, -0.1, 1.1]);
62. drawnow;
63. end
64. %\* Graph density versus position and time as wire-mesh plot
65. *figure(2); clf; % Clear figure 1 window and bring forward*
66. mesh(tplot,xplot,rplot)
67. xlabel('t'); ylabel('x'); zlabel('\rho');
68. title('Density versus position and time');
69. view([100 30]); % Rotate the plot for better view point
70. pause(1); % Pause 1 second between plots
71. %\* Graph contours of density versus position and time.
72. *figure(3); clf; % Clear figure 2 window and bring forward*
73. % Use rot90 function to graph t vs x since
74. % contour(rplot) graphs x vs t.
75. clevels = 0:(0.1):1; % Contour levels
76. cs = contour(xplot,tplot,flipud(rot90(rplot)),clevels);
77. clabel(cs); % Put labels on contour levels
78. xlabel('x'); ylabel('time'); title('Density contours');

Results:

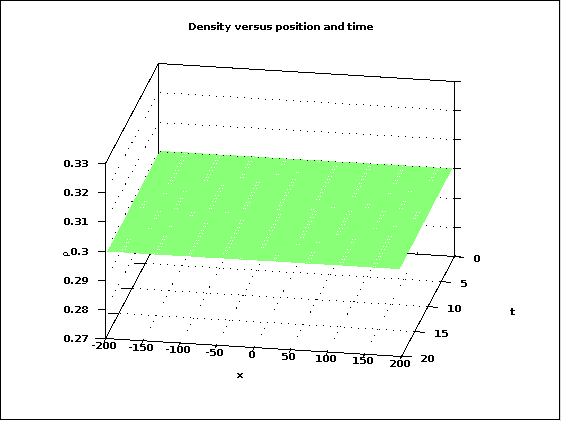
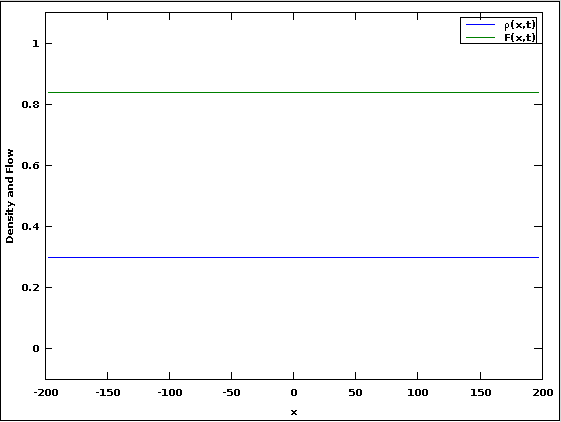


Figure 1: Density and Flow rate for constant low density, high velocity flow.

The graphs show that a constant density of 0.3 was maintained, as was a constant flow rate of ~0.85. The density contour plot was blank because the density was constant during the entire run.

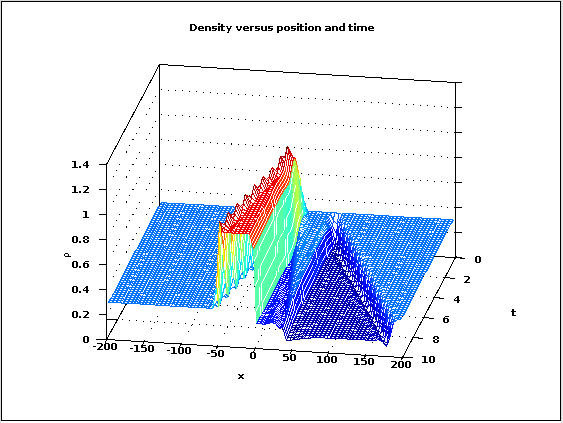
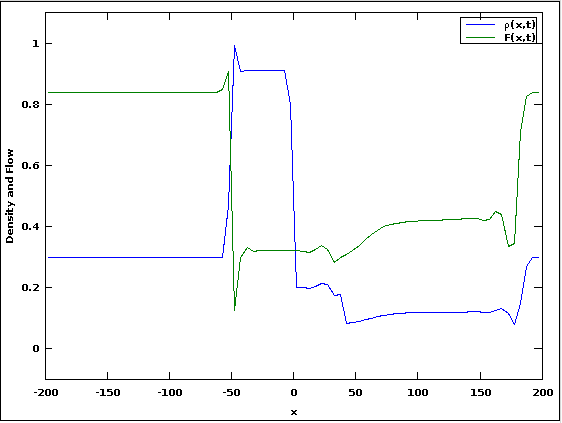
Part b

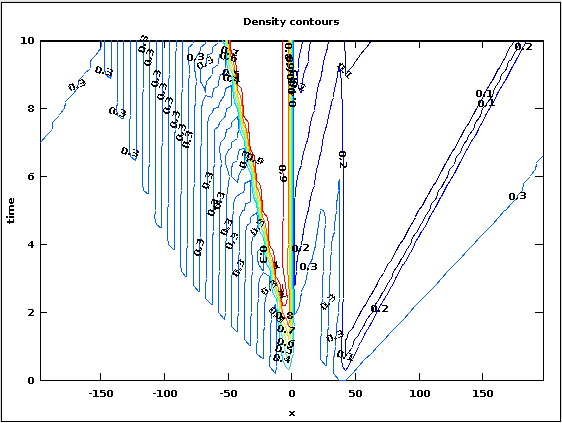
In this part, I created a traffic backup by setting vmax to 0.5 in the space 0<x<L/10. The result is pretty much what I expected: a decrease in flow rate that propagates backwards from the backup, with the flow rate increasing again once the traffic gets past the backup.

Code (modifications to **traffic.m** in italics):

1. % traffic - Program to solve the generalized Burger
2. % equation for the traffic at a stop light problem
3. clear all; help traffic; % Clear memory and print header
4. %\* Select numerical parameters (time step, grid spacing, etc.).
5. method = menu('Choose a numerical method:', ...
6. 'FTCS','Lax','Lax-Wendroff');
7. N = input('Enter the number of grid points: ');
8. L = 400; % System size (meters)
9. h = L/N; % Grid spacing for periodic boundary conditions
10. v\_max = 25; % Maximum car speed (m/s)
11. fprintf('Suggested timestep is %g\n',h/v\_max);
12. tau = input('Enter time step (tau): ');
13. nstep = input('Enter number of steps: ');
14. coeff = tau/(2\*h); % Coefficient used by all schemes
15. coefflw = tau^2/(2\*h^2); % Coefficient used by Lax-Wendroff
16. %\* Set initial and boundary conditions
17. rho\_max = 1.0; % Maximum density
18. Flow\_max = 0.25\*rho\_max\*v\_max; % Maximum Flow
19. % Initial condition is low density everywhere
20. rho = zeros(1,N);
21. *for i=1:N*
22. *rho(i) = 0.3; % Initial density is constant*
23. *end*
24. % Use periodic boundary conditions
25. ip(1:N) = (1:N)+1; ip(N) = 1; % ip = i+1 with periodic b.c.
26. im(1:N) = (1:N)-1; im(1) = N; % im = i-1 with periodic b.c.
27. %\* Initialize plotting variables.
28. iplot = 1;
29. xplot = ((1:N)-1/2)\*h - L/2; % Record x scale for plot
30. rplot(:,1) = rho(:); % Record the initial state
31. tplot(1) = 0;
32. figure(1); clf; % Clear figure 1 window and bring forward
33. *v\_maxM=v\_max\*ones(1,N);*
34. %\* Loop over desired number of steps.
35. for istep=1:nstep
36. *for j=round(N/2):(round(N/2)+round(N/10))*
37. *v\_maxM(j) = .5\*v\_max; % create the backup*
38. *end*
39. %\* Compute the flow = (Density)\*(Velocity)
40. *Flow = rho .\* (v\_maxM.\*(1 - rho/rho\_max));*
41. %\* Compute new values of density using FTCS,
42. % Lax or Lax-Wendroff method.
43. if( method == 1 ) %%% FTCS method %%%
44. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im));
45. elseif( method == 2 ) %%% Lax method %%%
46. rho(1:N) = .5\*(rho(ip)+rho(im)) ...
47. coeff\*(Flow(ip)-Flow(im));
48. else %%% Lax-Wendroff method %%%
49. *cp = v\_maxM.\*(1 - (rho(ip)+rho(1:N))/rho\_max);*
50. *cm = v\_maxM.\*(1 - (rho(1:N)+rho(im))/rho\_max);*
51. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im)) ...
52. + coefflw\*(cp.\*(Flow(ip)-Flow(1:N)) ...
53. cm.\*(Flow(1:N)-Flow(im)));
54. end
55. %\* Record density for plotting.
56. iplot = iplot+1;
57. rplot(:,iplot) = rho(:);
58. tplot(iplot) = tau\*istep;
59. %\* Display snap-shot of density versus position
60. plot(xplot,rho,'-',xplot,Flow/Flow\_max,'--');
61. xlabel('x'); ylabel('Density and Flow');
62. legend('\rho(x,t)','F(x,t)');
63. axis([-L/2, L/2, -0.1, 1.1]);
64. drawnow;
65. end
66. %\* Graph density versus position and time as wire-mesh plot
67. *figure(2); clf; % Clear figure 1 window and bring forward*
68. mesh(tplot,xplot,rplot)
69. xlabel('t'); ylabel('x'); zlabel('\rho');
70. title('Density versus position and time');
71. view([100 30]); % Rotate the plot for better view point
72. pause(1); % Pause 1 second between plots
73. %\* Graph contours of density versus position and time.
74. *figure(3); clf; % Clear figure 2 window and bring forward*
75. % Use rot90 function to graph t vs x since
76. % contour(rplot) graphs x vs t.
77. clevels = 0:(0.1):1; % Contour levels
78. cs = contour(xplot,tplot,flipud(rot90(rplot)),clevels);
79. clabel(cs); % Put labels on contour levels
80. xlabel('x'); ylabel('time'); title('Density contours');

Results:



Figure 2: Traffic backup with vmax=0.5 at 0<x<L/10.

Looking at the Density vs. position and time graph, you can see that the right edge of the density increase stays fixed at x=0, while the left edge propagates backwards. This is the traffic building up behind the scene of the accident. There is then a particularly low density area to the right of the backup, before the traffic returns to its normal density.

Also of interest is the negative spike in the flow rate that can be seen in the Density and Flow graph at x≈-50. It appears that the traffic slows down to a speed less than that necessary to navigate the high density area of the traffic backup. At the same time (or rather, position), the density spikes to 1. This corresponds to my own observations of traffic flow, where people brake harder than necessary when they see a slowdown ahead of them, which has the effect of increasing the severity of the backup behind them.

Part c

We change the program so that the traffic backup we created in Part b now lasts for a finite period of time, T. The backup lasts for just 16 time steps, while a total of 200 time steps are computed.

Code (modifications to **traffic.m** in italics):

1. % traffic - Program to solve the generalized Burger
2. % equation for the traffic at a stop light problem
3. clear all; help traffic; % Clear memory and print header
4. %\* Select numerical parameters (time step, grid spacing, etc.).
5. method = menu('Choose a numerical method:', ...
6. 'FTCS','Lax','Lax-Wendroff');
7. N = input('Enter the number of grid points: ');
8. L = 400; % System size (meters)
9. h = L/N; % Grid spacing for periodic boundary conditions
10. v\_max = 25; % Maximum car speed (m/s)
11. fprintf('Suggested timestep is %g\n',h/v\_max);
12. tau = input('Enter time step (tau): ');
13. nstep = input('Enter number of steps: ');
14. coeff = tau/(2\*h); % Coefficient used by all schemes
15. coefflw = tau^2/(2\*h^2); % Coefficient used by Lax-Wendroff
16. %\* Set initial and boundary conditions
17. rho\_max = 1.0; % Maximum density
18. Flow\_max = 0.25\*rho\_max\*v\_max; % Maximum Flow
19. % Initial condition is low density everywhere
20. rho = zeros(1,N);
21. *for i=1:N*
22. *rho(i) = 0.3; % Initial density is constant*
23. *end*
24. % Use periodic boundary conditions
25. ip(1:N) = (1:N)+1; ip(N) = 1; % ip = i+1 with periodic b.c.
26. im(1:N) = (1:N)-1; im(1) = N; % im = i-1 with periodic b.c.
27. %\* Initialize plotting variables.
28. iplot = 1;
29. xplot = ((1:N)-1/2)\*h - L/2; % Record x scale for plot
30. rplot(:,1) = rho(:); % Record the initial state
31. tplot(1) = 0;
32. figure(1); clf; % Clear figure 1 window and bring forward
33. *v\_maxM=v\_max\*ones(1,N);*
34. *%\* Loop over desired number of steps.*
35. *for istep=1:nstep*
36. *for j=round(N/2):(round(N/2)+round(N/10))*
37. *if istep<=20 && istep>=5*
38. *v\_maxM(j) = .5\*v\_max; % create the backup*
39. *else*
40. *v\_maxM(j) = 1\*v\_max; % remove the backup*
41. *endif*
42. *end*
43. %\* Compute the flow = (Density)\*(Velocity)
44. *Flow = rho .\* (v\_maxM.\*(1 - rho/rho\_max));*
45. %\* Compute new values of density using FTCS,
46. % Lax or Lax-Wendroff method.
47. if( method == 1 ) %%% FTCS method %%%
48. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im));
49. elseif( method == 2 ) %%% Lax method %%%
50. rho(1:N) = .5\*(rho(ip)+rho(im)) ...
51. coeff\*(Flow(ip)-Flow(im));
52. else %%% Lax-Wendroff method %%%
53. cp = v\_maxM.\*(1 - (rho(ip)+rho(1:N))/rho\_max);
54. cm = v\_maxM.\*(1 - (rho(1:N)+rho(im))/rho\_max);
55. rho(1:N) = rho(1:N) - coeff\*(Flow(ip)-Flow(im)) ...
56. + coefflw\*(cp.\*(Flow(ip)-Flow(1:N)) ...
57. cm.\*(Flow(1:N)-Flow(im)));
58. end
59. %\* Record density for plotting.
60. iplot = iplot+1;
61. rplot(:,iplot) = rho(:);
62. tplot(iplot) = tau\*istep;
63. %\* Display snap-shot of density versus position
64. plot(xplot,rho,'-',xplot,Flow/Flow\_max,'--');
65. xlabel('x'); ylabel('Density and Flow');
66. legend('\rho(x,t)','F(x,t)');
67. axis([-L/2, L/2, -0.1, 1.1]);
68. drawnow;
69. end
70. %\* Graph density versus position and time as wire-mesh plot
71. figure(2); clf; % Clear figure 2 window and bring forward
72. mesh(tplot,xplot,rplot);
73. xlabel('t'); ylabel('x'); zlabel('\rho');
74. title('Density versus position and time');
75. view([100 30]); % Rotate the plot for better view point
76. pause(1); % Pause 1 second between plots
77. %\* Graph contours of density versus position and time.
78. figure(3); clf; % Clear figure 3 window and bring forward
79. % Use rot90 function to graph t vs x since
80. % contour(rplot) graphs x vs t.
81. clevels = 0:(0.1):1; % Contour levels
82. cs = contour(xplot,tplot,flipud(rot90(rplot)),clevels);
83. clabel(cs); % Put labels on contour levels
84. xlabel('x'); ylabel('time'); title('Density contours');

Results:

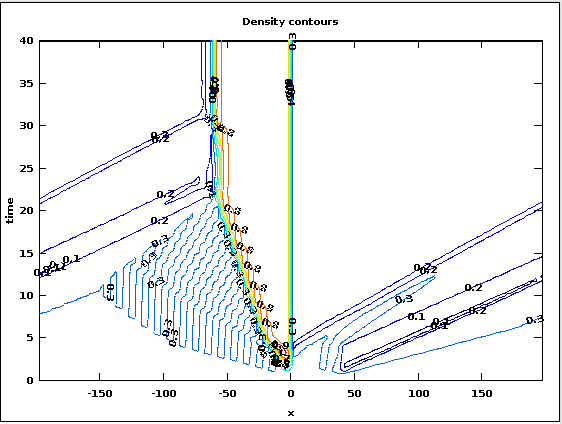
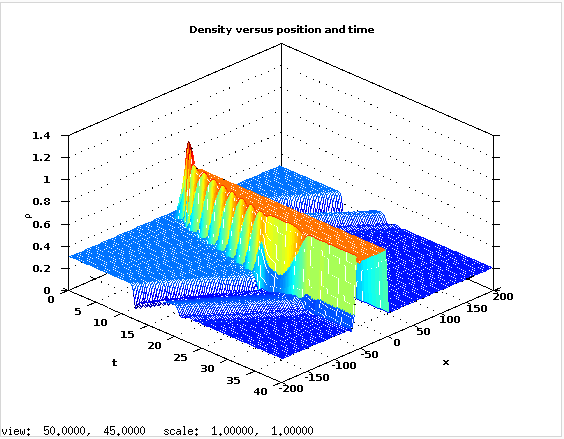


Figure 3: Traffic backup with vmax=0.5 at 0<x<L/10 for 5 ≤ t ≤ 20; Lax-Wendroff scheme.

Using the Lax-Wendroff scheme, a density plateau persists even though T<<t. The traffic returns to its normal speed everywhere in space except at that plateau.

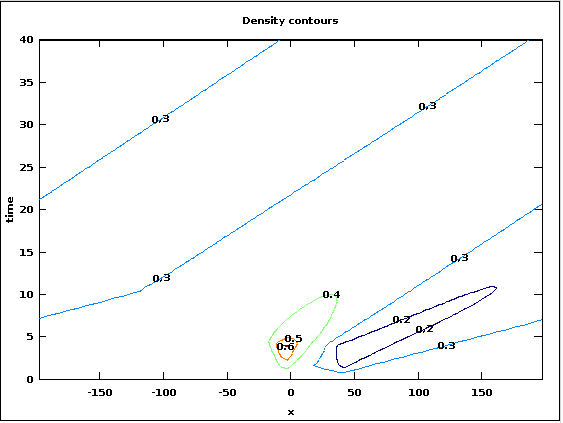
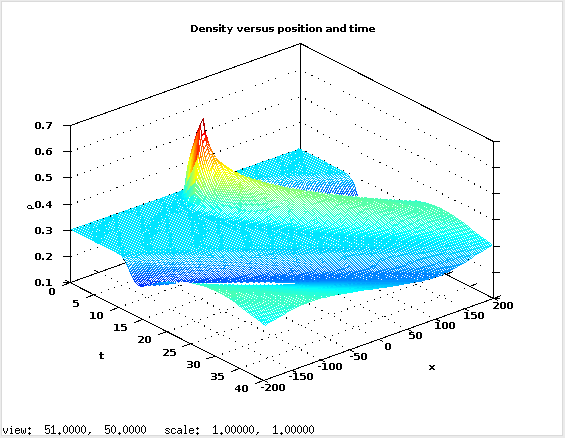


Figure 4: Traffic backup with vmax=0.5 at 0<x<L/10 for 5 ≤ t ≤ 20; Lax scheme.

However, if the Lax scheme is used instead to compute the evolution of the traffic flow, the plateau that persisted in the Lax-Wendroff scheme eventually dissipates and the flow approaches the steady state found in Part a.

I did also try a few other values for vmax for the durations of the backup. The results were all similar, except that the smaller the value of vmax, the longer it took for the backup to dissipate, as I would expect.