Michael Cowan

Computational Physics

Problem Set 8

1. Garcia Ch. 6 Problem 4

Part a

In this problem we model the diffusion of heat through a one dimensional system using the method of images. In order to reproduce the graphs shown in the book, a time of 0.03 s was used and only the actual bar and the first images (one on each side of the bar) are calculated and graphed, though the program can handle additional images.

Code:

1. clear all;
2. global n=input('How many images to calculate?');
3. global t=input('At what time (s) would you like to see the temperature distribution?');
4. global kappa=1;
5. global L=1;
6. function y=sigma(t)
7. global kappa
8. y=sqrt(2\*kappa\*t);
9. endfunction
10. function y=Tg(i,x,t)
11. y=(sigma(t)\*sqrt(2\*pi))^(-1)\*exp(-(x)^2/(2\*(sigma(t))^2));
12. endfunction
13. %Define the range over which to graph
14. x=(1:151)\*(3\*L/150)-3\*L/2-(3\*L/150);
15. for i=1:n
16. for j=1:151
17. T0(j)=Tg(0,x(j),t); %the distribution in the bar
18. T(j)=(-1)^i\*Tg(i,x(j)+i\*L,t)+(-1)^-i\*Tg(-i,x(j)-i\*L,t); %add the distributions in the images
19. Ttot=T0+T;
20. end
21. %plot the bar and the images separately
22. figure(1);
23. plot(x,T0,'-',x,T,'-.');
24. xlabel('x/L');
25. ylabel('T(x,t)');
26. end
27. %plot the bar and the images together
28. figure(2); clf;
29. plot(x,Ttot);
30. xlabel('x/L');
31. ylabel('T(x,t)');

Results:

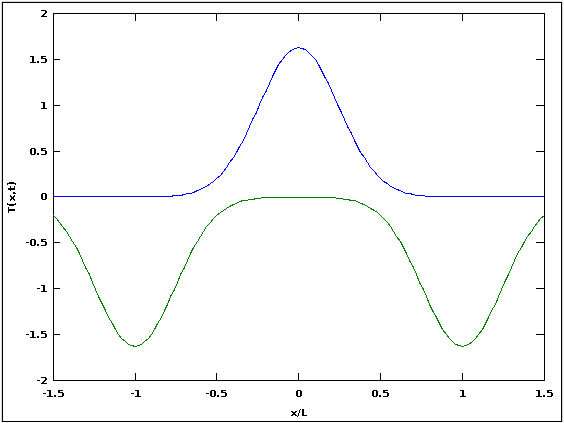
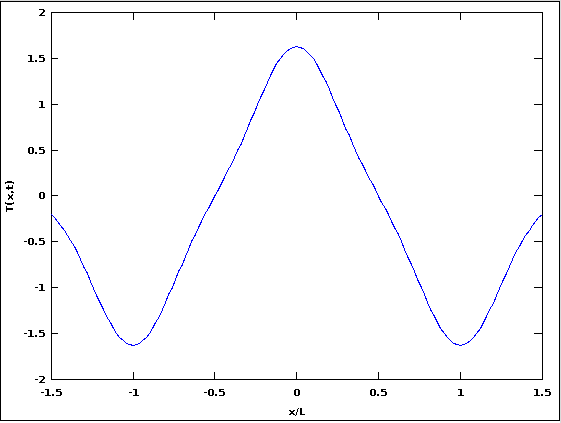


Figure 1 (above): Temperature distributions for the real bar and the first images

Figure 2 (below): Total temperature distribution for the real bar and the first images



Part b

In part b of the problem, we combine the given program **dfcts.m**, which uses the FTCS method to calculate the temperature profile of the bar, with our program from part a and calculate the difference in the results from each method. Some adjustments to my part a program had to made to display how the temperature in the bar over a period of time instead of at just one moment.

Code (my modifications to **dfcts.m** are in italics):

1. % dftcs - Program to solve the diffusion equation
2. % using the Forward Time Centered Space (FTCS) scheme.
3. clear; help dftcs; % Clear memory and print header
4. %\* Initialize parameters (time step, grid spacing, etc.).
5. *global tau = input('Enter time step: ');*
6. *global N = input('Enter the number of grid points: ');*
7. *global L = 1.; % The system extends from x=-L/2 to x=L/2*
8. *global h = L/(N-1); % Grid size*
9. *global kappa = 1.; % Diffusion coefficient*
10. *image=10; %number of images to calculate*
11. coeff = kappa\*tau/h^2;
12. if( coeff < 0.5 )
13. disp('Solution is expected to be stable');
14. else
15. disp('WARNING: Solution is expected to be unstable');
16. end
17. %\* Set initial and boundary conditions.
18. tt = zeros(N,1); % Initialize temperature to zero at all points
19. tt(round(N/2)) = 1/h; % Initial cond. is delta function in center
20. %% The boundary conditions are tt(1) = tt(N) = 0
21. %\* Set up loop and plot variables.
22. xplot = (0:N-1)\*h - L/2; % Record the x scale for plots
23. iplot = 1; % Counter used to count plots
24. nstep = 300; % Maximum number of iterations
25. nplots = 50; % Number of snapshots (plots) to take
26. plot\_step = nstep/nplots; % Number of time steps between plots
27. %\* Loop over the desired number of time steps.
28. for istep=1:nstep %% MAIN LOOP %%
29. %\* Compute new temperature using FTCS scheme.
30. tt(2:(N-1)) = tt(2:(N-1)) + ...
31. coeff\*(tt(3:N) + tt(1:(N-2)) - 2\*tt(2:(N-1)));
32. %\* Periodically record temperature for plotting.
33. if( rem(istep,plot\_step) < 1 ) % Every plot\_step steps
34. ttplot(:,iplot) = tt(:); % record tt(i) for plotting
35. tplot(iplot) = istep\*tau; % Record time for plots
36. iplot = iplot+1;
37. end
38. end
39. *function y=sigma(t)*
40. *global kappa*
41. *y=sqrt(2\*kappa\*t);*
42. *endfunction*
43. *function y=Tg(i,x,t)*
44. *y=(sigma(t)\*sqrt(2\*pi))^(-1)\*exp(-(x)^2/(2\*(sigma(t))^2));*
45. *endfunction*
46. *x=xplot; %create grid for Ta*
47. *for k=1:nplots*
48. *for i=1:image*
49. *for j=1:N*
50. *t=k\*tau;*
51. *T0(j,k)=Tg(0,x(j),t); %the distribution in the bar*
52. *T(j,k)=(-1)^i\*Tg(i,x(j)+i\*L,t)+(-1)^-i\*Tg(-i,x(j)-i\*L,t); %add the distributions in the images*
53. *Ttot=T0+T;*
54. *end*
55. *end*
56. *end*
57. *deltat=abs(Ttot-ttplot);*
58. %\* Plot temperature versus x and t as wire-mesh and contour plots.
59. figure(1); clf;
60. mesh(tplot,xplot,ttplot); % Wire-mesh surface plot
61. xlabel('Time'); ylabel('x'); zlabel('T(x,t)');
62. title('Diffusion of a delta spike');
63. pause(0.5);
64. figure(2); clf;
65. contourLevels = 0:0.5:10; contourLabels = 0:5;
66. cs = contour(tplot,xplot,ttplot,contourLevels); % Contour plot
67. clabel(cs,contourLabels); % Add labels to selected contour levels
68. xlabel('Time'); ylabel('x');
69. title('Temperature contour plot');
70. pause(0.5);
71. *figure(3);clf;*
72. *mesh(tplot,xplot,deltat);*
73. *xlabel('Time'); ylabel('x'); zlabel('|Ta(x,t)-Tc(x,t)|');*
74. *title('Comparison of the diffusion of a delta spike using FTCS and image methods');*
75. *pause(0.5);*
76. *figure(4);clf;*
77. *mesh(tplot,xplot,Ttot);*
78. *xlabel('Time'); ylabel('x'); zlabel('T(x,t)');*
79. *title('Diffusion of a delta spike using image method');*

Results:

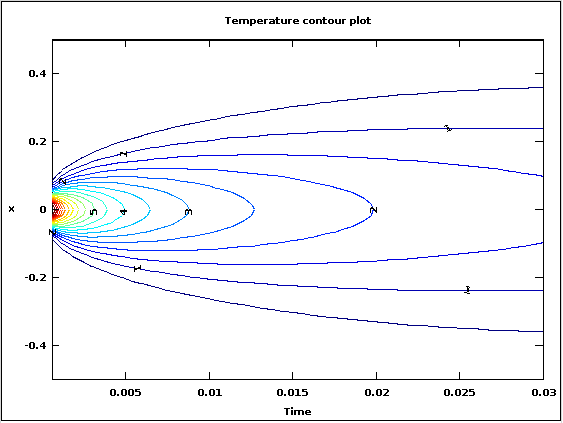
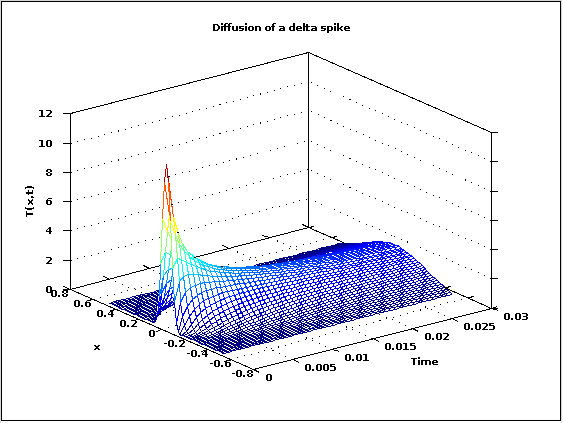


Figure 3: Plots produced by **dftcs.m** showing the diffusion of the delta spike over 0.03 s calculated using the FTCS method.

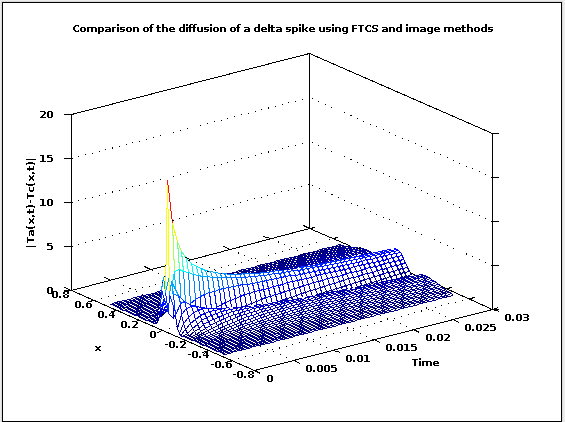
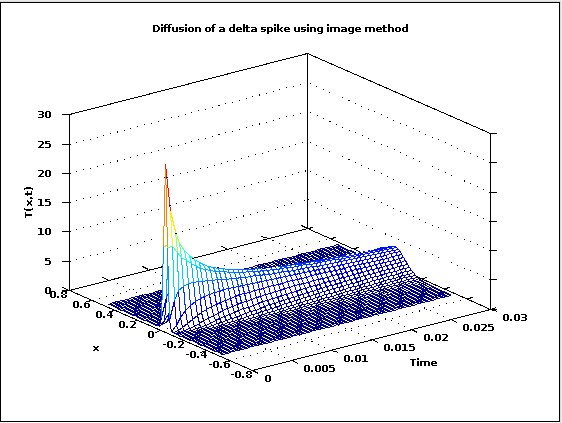


Figure 4: The diffusion of the delta spike over 0.03 s using the image method and the comparison of the image method to the FTCS method.

It is apparent that there are significant differences in the results of the two calculation methods (if the two methods produced identical results the last plot would be flat). In particular, the Gaussian spreads out much quicker for the FTCS method than for the image method. At first I supposed that the difference was due to my using only one image in the calculation, as I did in part a, but increasing the number of images to ten did not significantly change the result.

2. Garcia Chapter 6 Problem 11

In this problem we modify the given program **neutrn.m** to add a curve, , where . This allows us to compare the growth or decay rate of the neutron density to simple exponential growth/decay. In part a, this is done with a regular plot; in part b, the plot is modified to use a log scale on the vertical axis.

Code (modifications to **neutrn.m** are in italics; the only change for part b is to change the plot command in line 62 to semilogy):

1. % neutrn - Program to solve the neutron diffusion equation
2. % using the Forward Time Centered Space (FTCS) scheme.
3. clear; help neutrn; % Clear memory and print header
4. %\* Initialize parameters (time step, grid points, etc.).
5. tau = input('Enter time step: ');
6. N = input('Enter the number of grid points: ');
7. L = input('Enter system length: ');
8. % The system extends from x=-L/2 to x=L/2
9. h = L/(N-1); % Grid size
10. D = 1.; % Diffusion coefficient
11. C = 1.; % Generation rate
12. coeff = D\*tau/h^2;
13. coeff2 = C\*tau;
14. if( coeff < 0.5 )
15. disp('Solution is expected to be stable');
16. else
17. disp('WARNING: Solution is expected to be unstable');
18. end
19. %\* Set initial and boundary conditions.
20. nn = zeros(N,1); % Initialize density to zero at all points
21. nn\_new = zeros(N,1); % Initialize temporary array used by FTCS
22. nn(round(N/2)) = 1/h; % Initial cond. is delta function in center
23. %% The boundary conditions are nn(1) = nn(N) = 0
24. %\* Set up loop and plot variables.
25. xplot = (0:N-1)\*h - L/2; % Record the x scale for plots
26. iplot = 1; % Counter used to count plots
27. nstep = input('Enter number of time steps: ');
28. nplots = 50; % Number of snapshots (plots) to take
29. plot\_step = nstep/nplots; % Number of time steps between plots
30. %\* Loop over the desired number of time steps.
31. for istep=1:nstep %% MAIN LOOP %%
32. %\* Compute the new density using FTCS scheme.
33. nn\_new(2:(N-1)) = nn(2:(N-1)) + ...
34. coeff\*(nn(3:N) + nn(1:(N-2)) - 2\*nn(2:(N-1))) + ...
35. coeff2\*nn(2:(N-1));
36. nn = nn\_new; % Reset temperature to new values
37. %\* Periodically record the density for plotting.
38. if( rem(istep,plot\_step) < 1 ) % Every plot\_step steps
39. nnplot(:,iplot) = nn(:); % record nn(i) for plotting
40. tplot(iplot) = istep\*tau; % Record time for plots
41. nAve(iplot) = mean(nn); % Record average density
42. iplot = iplot+1;
43. end
44. end
45. %\* Plot density versus x and t as a 3D-surface plot
46. figure(1); clf;
47. mesh(tplot,xplot,nnplot);
48. xlabel('Time'); ylabel('x'); zlabel('n(x,t)');
49. title('Neutron diffusion');
50. %\* Plot average neutron density versus time
51. figure(2); clf;
52. *alpha=C-D\*pi^2/L^2;*
53. *y=nAve(1)\*exp(alpha\*(tplot));*
54. *plot(tplot,nAve,'\*',tplot,y,'--');*
55. xlabel('Time'); ylabel('Average density');
56. title(['L = ',num2str(L),' (L\_c = \pi)']);

Results:

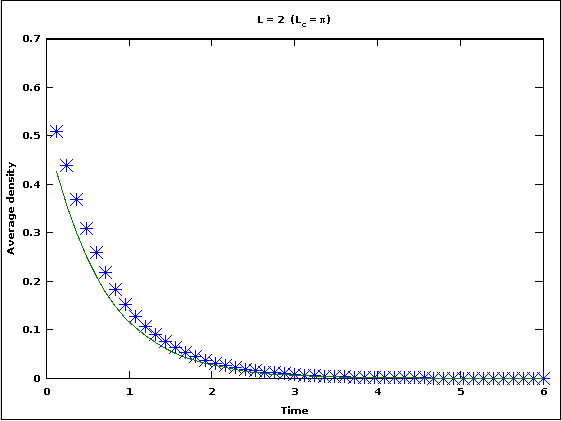
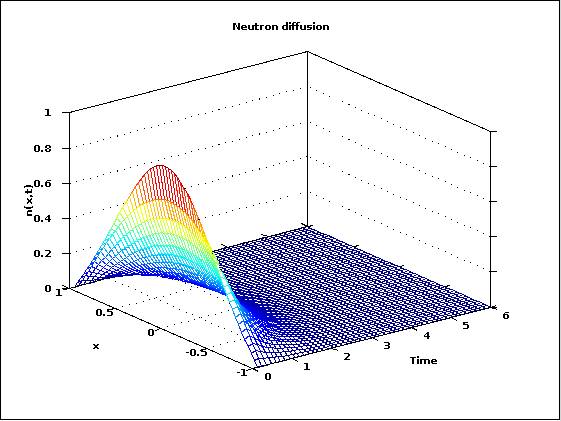


Figure 5: Mesh plot of and plot of with system length L=2 (subcritical)

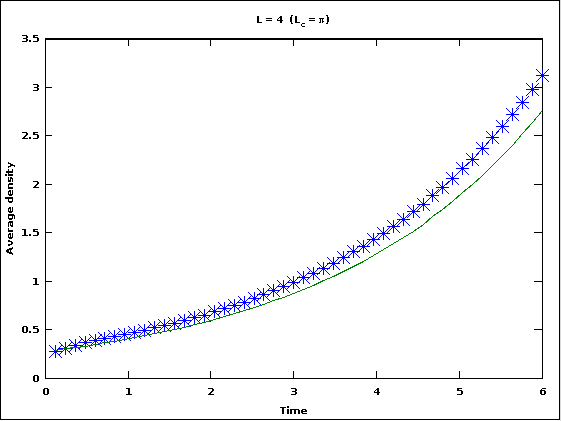
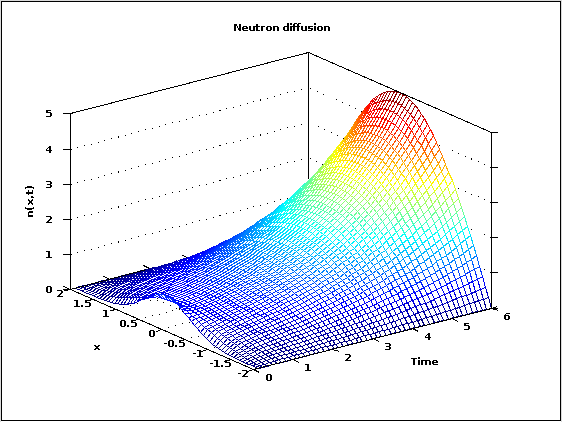


Figure 6: Mesh plot of and plot of with system length L=4 (supercritical)

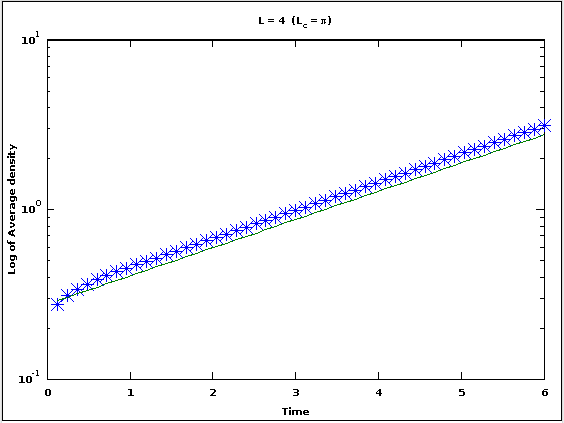
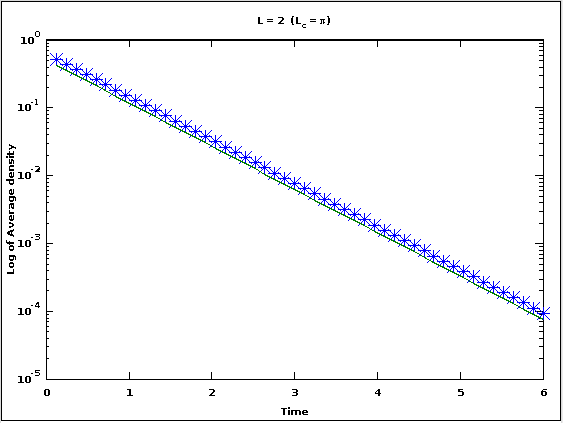


Figure 7: Log plot of with system length L=2 and L=4

Both systems follow the exponential curve reasonably closely.

3. Garcia Chapter 6 Problem 14

In this problem we modify the **neutrn.m** program to include a tamper outside of the neutron source. That is, there is a region where neutron diffusion occurs without neutron creation. In other words, we extend the system by a length factor a, but with C=0 for |x|>L/2.

Code (my modifications to **neutrn.m** are in italics):

1. % neutrn - Program to solve the neutron diffusion equation
2. % using the Forward Time Centered Space (FTCS) scheme.
3. clear; help neutrn; % Clear memory and print header
4. %\* Initialize parameters (time step, grid points, etc.).
5. tau = input('Enter time step: ');
6. N = input('Enter the number of grid points: ');
7. L = input('Enter system length containing the neutron source: ');
8. % The system extends from x=-L/2 to x=L/2
9. *a=input('Enter the multiple of L for the tamper material: ');*
10. *h = a\*L/(N-1); % Grid size*
11. D = 1.; % Diffusion coefficient
12. C = 1.; % Generation rate
13. coeff = D\*tau/h^2;
14. coeff2 = C\*tau;
15. if( coeff < 0.5 )
16. disp('Solution is expected to be stable');
17. else
18. disp('WARNING: Solution is expected to be unstable');
19. end
20. %\* Set initial and boundary conditions.
21. nn = zeros(N,1); % Initialize density to zero at all points
22. nn\_new = zeros(N,1); % Initialize temporary array used by FTCS
23. nn(round(N/2)) = 1/h; % Initial cond. is delta function in center
24. %% The boundary conditions are nn(1) = nn(N) = 0
25. %\* Set up loop and plot variables.
26. xplot = (0:N-1)\*h - a\*L/2; % Record the x scale for plots
27. iplot = 1; % Counter used to count plots
28. nstep = input('Enter number of time steps: ');
29. nplots = 50; % Number of snapshots (plots) to take
30. plot\_step = nstep/nplots; % Number of time steps between plots
31. %\* Loop over the desired number of time steps.
32. for istep=1:nstep %% MAIN LOOP %%
33. *if (abs(xplot)>=L/2)*
34. *coeff2=0;*
35. *endif*
36. %\* Compute the new density using FTCS scheme.
37. nn\_new(2:(N-1)) = nn(2:(N-1)) + ...
38. coeff\*(nn(3:N) + nn(1:(N-2)) - 2\*nn(2:(N-1))) + ...
39. coeff2\*nn(2:(N-1));
40. nn = nn\_new; % Reset temperature to new values
41. %\* Periodically record the density for plotting.
42. if( rem(istep,plot\_step) < 1 ) % Every plot\_step steps
43. nnplot(:,iplot) = nn(:); % record nn(i) for plotting
44. tplot(iplot) = istep\*tau; % Record time for plots
45. nAve(iplot) = mean(nn); % Record average density
46. iplot = iplot+1;
47. end
48. end
49. %\* Plot density versus x and t as a 3D-surface plot
50. figure(1); clf;
51. mesh(tplot,xplot,nnplot);
52. xlabel('Time'); ylabel('x'); zlabel('n(x,t)');
53. title('Neutron diffusion');
54. %\* Plot average neutron density versus time
55. figure(2); clf;
56. plot(tplot,nAve,'\*');
57. xlabel('Time'); ylabel('Average density');
58. *title(['L = ',num2str(L),' a = ',num2str(a)]);*

Results:

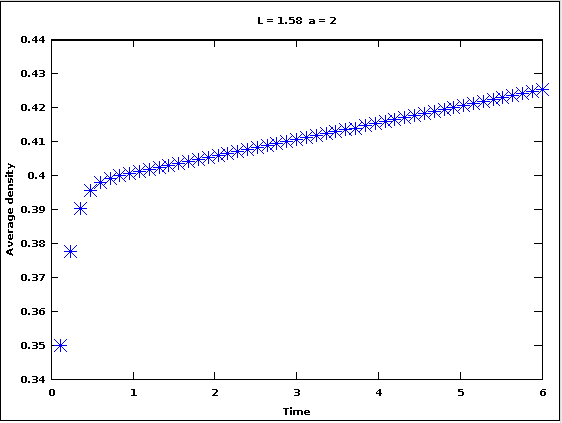
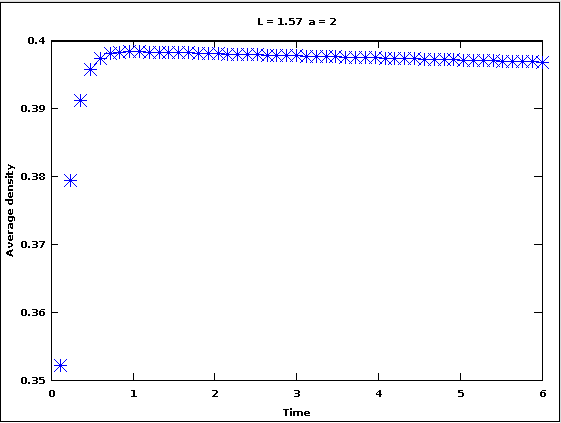


Figure 8: For a tamper length a=2, the critical length L is between 1.57 and 1.58

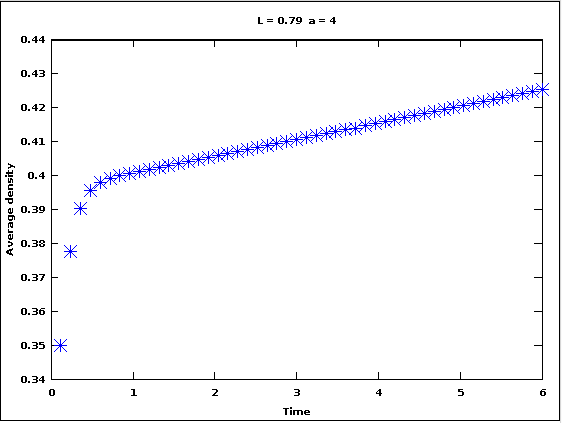
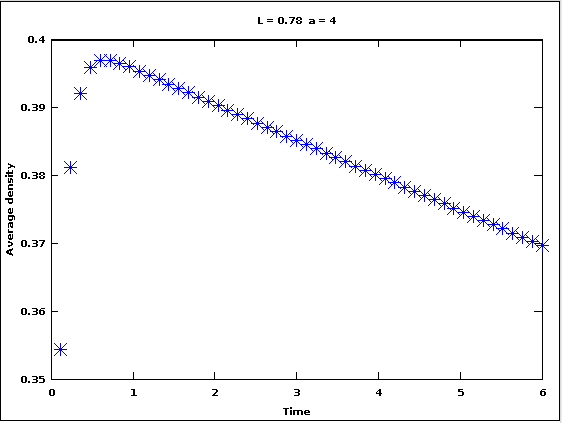


Figure 9: For a tamper length a=4, the critical length L is between 0.78 and 0.79

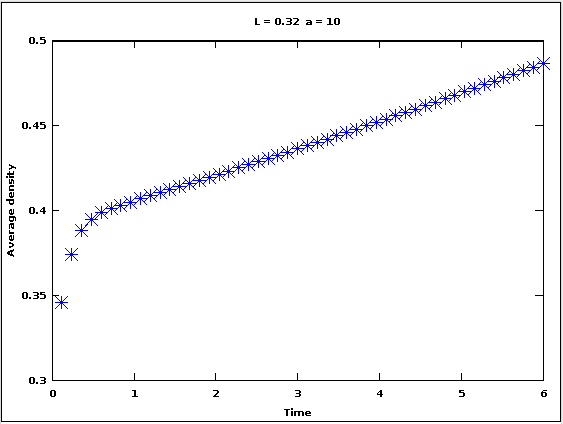
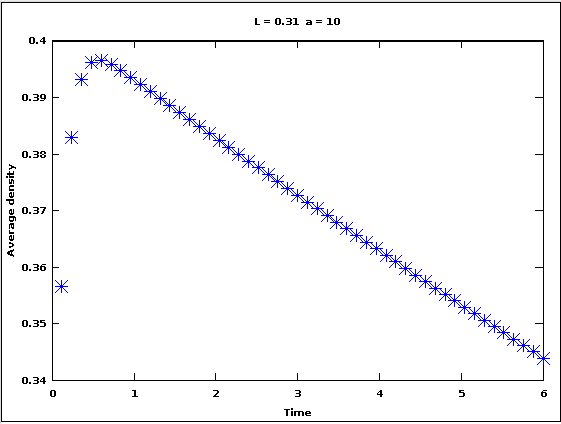


Figure 10: For a tamper length a=10, the critical length L is between 0.31 and 0.32

4. Spectral Method for diffusion problem

Part a

In part a, we are asked to modify the given **diffusion.m** program to handle initial temperature distributions that are not limited to even functions. This was done by basically doubling up all of the functions in the program, but replacing cos terms with sin terms. The additional functions were then added onto the total to create a program that could handle any type of initial distribution. It should be noted, however, that the program still requires the initial distribution to be coded directly into the appropriate function, rather than entered as an input.

Code (my modifications to **diffusion.m** are in italics):

1. % diffusion equation solution using
2. % spectral method.
3. %
4. % E.H. Majzoub for CompPhys 4350
5. % UM-St. Louis
6. % 16 Nov 2010
7. *% Modified by M. Cowan 27 Apr 2015*
8. clear;
9. warning("off","all");
10. t0=0; t1=0.001; t2=0.003; t3=0.01; t4=0.03; t5=0.1;
11. global maxorder=10;
12. maxplot=60;
13. % let the length L=1
14. global L=1;
15. global k=1;
16. global A;
17. *global B;*
18. % B.C.s: T=0 at x= +/- L/2
19. # Frist, we define the basis set
20. #
21. *function y = omega1(n)*
22. global L;
23. n=n-1;
24. y = (2\*n+1)\*pi/(L);
25. endfunction
26. *function y = omega2(n)*
27. *global L;*
28. *n=n-1;*
29. *y = (2\*n)\*pi/(L);*
30. *endfunction*
31. *function y = u1(n,x)*
32. global L;
33. y = (2/L) \* cos( omega1(n) \* x );
34. endfunction
35. *function y = u2(n,x)*
36. *global L;*
37. *y = (2/L) \* sin( omega2(n) \* x );*
38. *endfunction*
39. *# Define zero time distribution*
40. *function y = fzero(x)*
41. *global L;*
42. *if (x>=0 && x<=L/4)*
43. *y=4/L;*
44. *else*
45. *y=0;*
46. *end*
47. *endfunction*
48. # Define some functions for integration
49. *function y = ufmult1(n1,x)*
50. *y = u1(n1,x)\*fzero(x);*
51. endfunction
52. *function y = ufmult2(n1,x)*
53. *y = u2(n1,x)\*fzero(x);*
54. *endfunction*
55. # calculate the coeffs
56. *function y = coef1(n)*
57. global L;
58. *[v,ierror,nval] = quad( @(x) ufmult1(n,x),-L/2,L/2);*
59. y = v;
60. endfunction
61. *function y = coef2(n)*
62. *global L;*
63. *[v,ierror,nval] = quad( @(x) ufmult2(n,x),-L/2,L/2);*
64. *y = v;*
65. *endfunction*
66. # calculate the temperature distribution
67. function y = T(x,t)
68. *global k L A B maxorder;*
69. y=0;
70. for n=1:maxorder
71. *y = y + A(n)\*exp(-k\*omega1(n)\*\*2\*t)\*cos(omega1(n)\*x) + B(n)\*exp(-k\*omega2(n)\*\*2\*t)\*sin(omega2(n)\*x);*
72. endfor
73. endfunction
74. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
75. % calculate the coefs first
76. for i=1:maxorder
77. A(i) = coef1(i);
78. *B(i) = coef2(i);*
79. endfor
80. x = -L/2-(L/maxplot) + (1:maxplot+1)\*(L/maxplot);
81. for i=1:maxplot+1
82. yfzero(i) = fzero(x(i));
83. yT0(i) = T(x(i),t0);
84. yT1(i) = T(x(i),t1);
85. yT2(i) = T(x(i),t2);
86. yT3(i) = T(x(i),t3);
87. yT4(i) = T(x(i),t4);
88. yT5(i) = T(x(i),t5);
89. endfor
90. figure(1); clf;
91. plot(x,yfzero,'-',x,yT1,'-',x,yT2,'-',x,yT3,'-',x,yT4,'-',x,yT5,'-');
92. legend("t0","t1","t2","t3","t4","t5");
93. xlabel("Position [L]");
94. ylabel("Temperature (norm)");
95. refresh(1);
96. print("diffout.png");

Results:

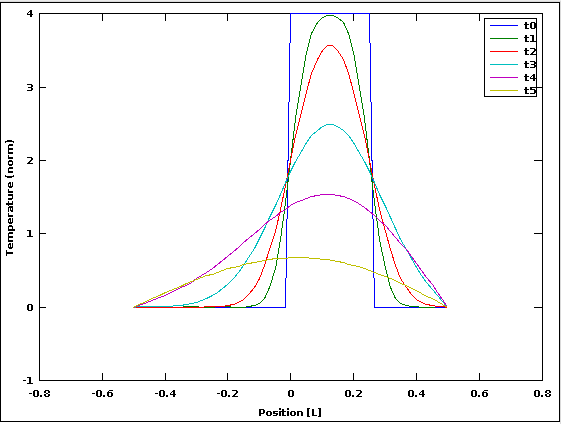


Figure 11: Evolution of the temperature distribution

As you can see in the graph above, the distribution starts as an odd square wave function and gradually spreads out, approaching a Gaussian distribution centered at x=0 as expected.

Part b

In part b, we modify the program from part a to include a heat source at the center of the bar. Although the instructions said to refer to section 6.3 and the neutron diffusion problem described therein, the book section uses the FTCS scheme to discretize the problem while the diffusion.m program uses direct integration, so a different approach is required. So instead I used a delta function located at x=0 to simulate the source. The delta function source was not time dependent, so the heat generated by the source gradually spreads out and comes to dominate the system as the initial temperature distribution diffuses, eventually ending up in the expected Gaussian distribution. For comparison I included a plot of the temperature distribution without the heat source; as expected, this system tends to a temperature of zero everywhere due to the sinks located at the ends of the bar.

Code:

1. % diffusion equation solution using
2. % spectral method.
3. %
4. % E.H. Majzoub for CompPhys 4350
5. % UM-St. Louis
6. % 16 Nov 2010
7. % Modified by M. Cowan 27 Apr 2015
8. clear;
9. warning("off","all");
10. t0=0; t1=0.001; t2=0.003; t3=0.01; t4=0.03; t5=0.1;
11. global maxorder=10;
12. maxplot=60;
13. % let the length L=1
14. global L=1;
15. global k=1;
16. global A;
17. global B;
18. global S;
19. global h=L/maxplot;
20. % B.C.s: T=0 at x= +/- L/2
21. # Frist, we define the basis set
22. #
23. function y = omega1(n)
24. global L;
25. n=n-1;
26. y = (2\*n+1)\*pi/(L);
27. endfunction
28. function y = omega2(n)
29. global L;
30. n=n-1;
31. y = (2\*n)\*pi/(L);
32. endfunction
33. function y = u1(n,x)
34. global L;
35. y = (2/L) \* cos( omega1(n) \* x );
36. endfunction
37. function y = u2(n,x)
38. global L;
39. y = (2/L) \* sin( omega2(n) \* x );
40. endfunction
41. # Define zero time distribution
42. function y = fzero(x)
43. global L;
44. if (x>=0 && x<=L/4)
45. y=4/L;
46. else
47. y=0;
48. end
49. endfunction
50. function y = source(x)
51. global h;
52. y=0;
53. if (abs(x)<=h/4)
54. y=1/h;
55. end
56. endfunction
57. # Define some functions for integration
58. function y = ufmult1(n1,x)
59. y = u1(n1,x)\*fzero(x);
60. endfunction
61. function y = ufmult2(n1,x)
62. y = u2(n1,x)\*fzero(x);
63. endfunction
64. function y = ufmult3(n1,x)
65. global L;
66. y = (2/L)\*u1(n1,x)\*source(x);
67. endfunction
68. # calculate the coeffs
69. function y = coef1(n)
70. global L;
71. [v,ierror,nval] = quad( @(x) ufmult1(n,x),-L/2,L/2);
72. y = v;
73. endfunction
74. function y = coef2(n)
75. global L;
76. [v,ierror,nval] = quad( @(x) ufmult2(n,x),-L/2,L/2);
77. y = v;
78. endfunction
79. function y = coef3(n)
80. global L;
81. [v,ierror,nval] = quad( @(x) ufmult3(n,x),-L/2,L/2);
82. y = v;
83. endfunction
84. # calculate the temperature distribution
85. function y = T(x,t)
86. global k L A B maxorder;
87. y=0;
88. for n=1:maxorder
89. y = y + A(n)\*exp(-k\*omega1(n)\*\*2\*t)\*cos(omega1(n)\*x) + B(n)\*exp(-k\*omega2(n)\*\*2\*t)\*sin(omega2(n)\*x);
90. endfor
91. endfunction
92. # calculate the source
93. function y = Ts(x)
94. global k S maxorder;
95. y=0;
96. for(n=1:maxorder)
97. y=y+S(n)\*exp(-k\*omega1(n)^2\*.05)\*u1(n,x);
98. endfor
99. endfunction
100. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
101. % calculate the coefs first
102. for i=1:maxorder
103. A(i) = coef1(i);
104. B(i) = coef2(i);
105. S(i) = coef3(i);
106. endfor
107. x = -L/2-(L/maxplot) + (1:maxplot+1)\*(L/maxplot);
108. for(j=1:50)
109. time(j)=j/50;
110. for(i=1:maxplot+1)
111. yT(i,j)=T(x(i),time(j));
112. yTs(i,j)=Ts(x(i));
113. endfor
114. endfor
115. yTot=yT.+yTs;
116. figure(1); clf;
117. mesh(time,x,yT);
118. xlabel('Time'); ylabel('x'); zlabel('T(x,t)');
119. title('Temperature distribution without heat source');
120. figure(2); clf;
121. mesh(time,x,yTot);
122. xlabel('Time'); ylabel('x'); zlabel('T(x,t)');
123. title('Temperature distribution with heat source');

Results:

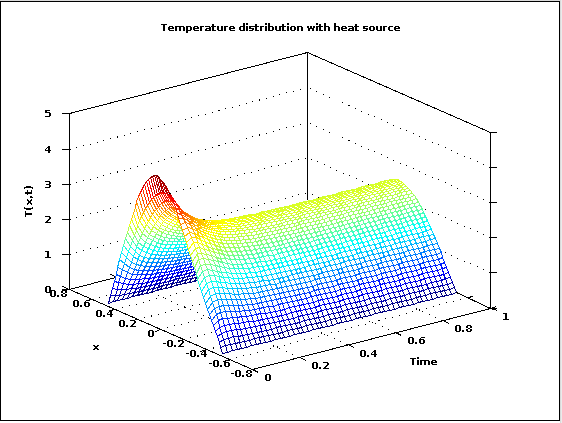
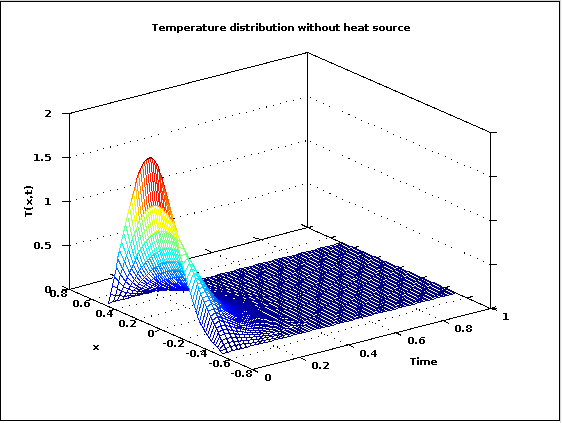


Figure 12: Comparison of the diffusion of heat with and without a heat source at x=0.

As expected, the system without a heat source dissipates all of the initial heat out through the sinks located at the ends of the bar. With a heat source, however, the temperature settles into a Gaussian distribution centered around the heat source at x=0.