Project B Numerical solution of the heat equation

Table of Contents

Exercise 4.1

See comments next to each line in the code of the following function for intepretation/explanation of what specific commands do.

```
type heat_eqn.m
```

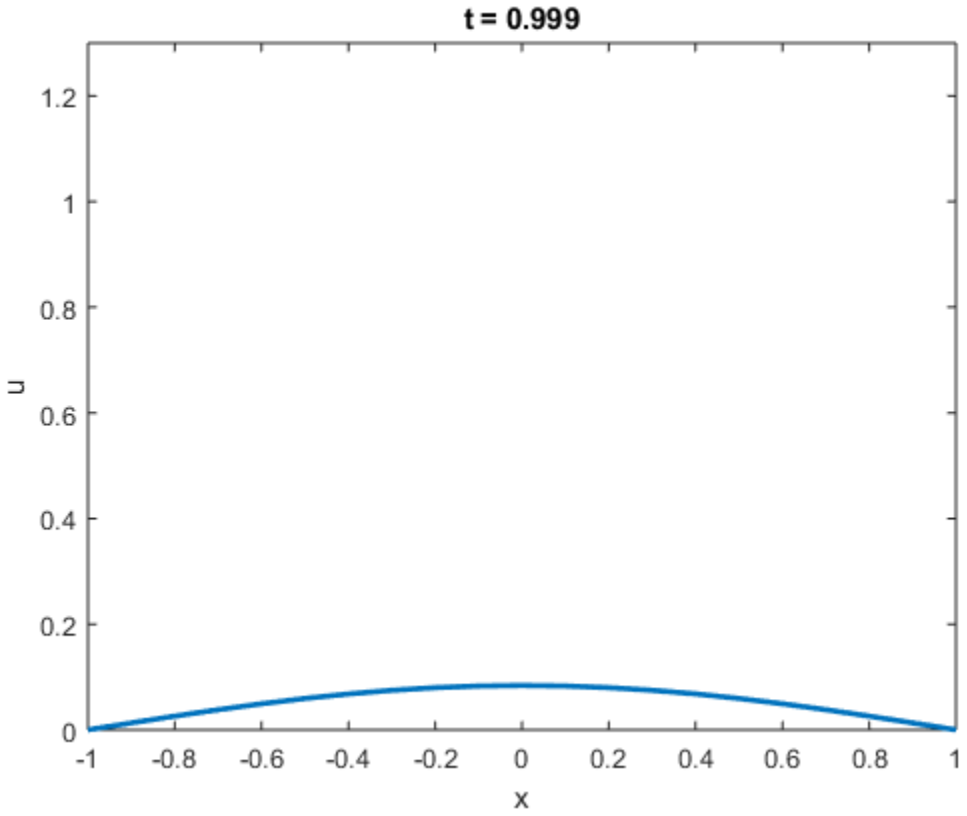
```
function u = heat_eqn(T,M,lambda)

%set up grid points
h = 2/M; % grid spacing for splitting the x-axis
x = (linspace(-1,1,M+1))'; % splitting the interval [-1,1]
k = lambda*h^2; % grid spacing for going ahead in time
N = ceil(T/k);

%initial conditions
u = cos(pi*x/2) + (sin(pi*x))/2; % u(x,0)
t = 0;

%plot u(x,0)
plot(x,u,'b','LineWidth',2); % plotting the function u(x,0)
axis([-1 1 0 1.3]); % setting the min/max values of the axes displayed
title('t = 0'); % adding a title that displays the initial time
xlabel('x'); % labeling x-axis
ylabel('u'); % labeling y-axis
```

```
%finite difference operator
e = ones(M-1,1); % generating an (M-1)x1 column vector of 1s only
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
% tridiagonal and sparse, with main diag of -2s, adjacent of -1s
I = speye(M-1); % sparse identity matrix
A = I + lambda*L; % A as defined in the problem

%main loop
while (t<T)
    u_inner = u(2:M,1); % gets the M-1 inner components of U^n
 u_inner = A*u_inner; % calculating solution forward in time
    u = [0; u_inner; 0]; % adding the boundary conditions
    %plot current solution
    hold off
    plot(x,u,'LineWidth',2); % plotting u(x,t)
    axis([-1 1 0 1.3]); % setting the min/max of the axes
 title(['t = ' num2str(t)]); % converts t to character to display
    xlabel('x'); % labeling x-axis
    ylabel('u'); % labeling y-axis
 pause(0.02); % pausing to display solution sufficiently long
    t = t + k; % going ahead in time by the specified amount
end
```

Part a)

```
heat_eqn(1,20,0.3)


ans =

         0
    0.0131
    0.0260
    0.0381
    0.0494
    0.0594
    0.0680
    0.0749
    0.0799
    0.0830
    0.0840
    0.0830
    0.0799
    0.0749
    0.0680
    0.0595
```

```
    0.0494
    0.0382
    0.0260
    0.0132
         0
```

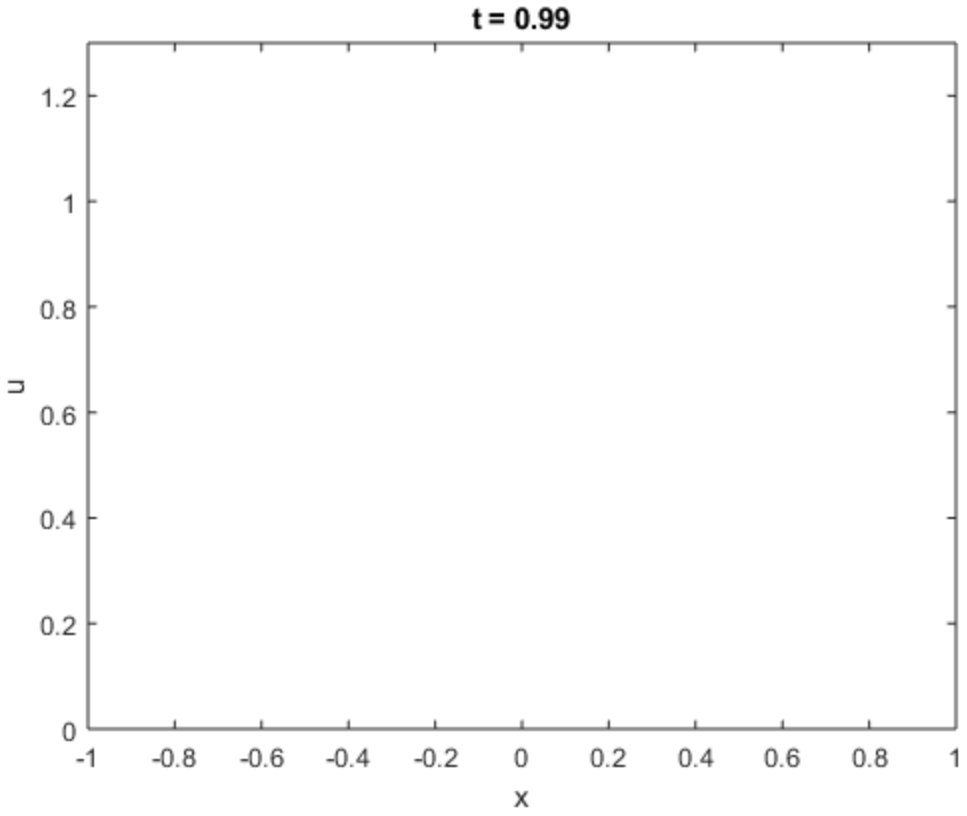


Part b)

```
heat_eqn(1,20,1)
```

```
ans =

   1.0e+29 *

         0
   -0.3047
    0.5459
   -0.6657
    0.6178
   -0.3719
   -0.0818
    0.7308
   -1.5391
    2.4490
```

```
-3.3853
 4.2609
-4.9854
 5.4743
-5.6582
 5.4915
-4.9587
 4.0780
-2.9006
 1.5072
      0
```



For $\lambda = 0.3$, the solution is bounded and stable: the heat is at first mostly concentrated just to the left of $x = 0.4$. Then, as time progresses, the heat is being distributed more and more evenly across the interval $-1 \leq x \leq 1$, and ultimately it is in the shape reminiscent of a low-arcing, symmetric parabola with the peak heat having shifted to where $x = 0$. The physical intepretation of this problem is that it is a Dirichlet problem with a zero steady-state situation, whereby the ends are not thermally insulated and the heat may escape the object, which is what happens as we go further in time. From the Fourier Series and PDEs course, we would expect the coefficients for the distribution of heat with time to decay exponentially, solidifying this claim further. Meanwhile, for $\lambda = 1$, our solution initially behaves almost identically to the previous one until just after $t = 0.35$ Indeed, upon reaching a shape similar to that previously obtained in a), it then becomes unstable, shooting off to $\pm\infty$ at some points and remaining 0 at others. It continues to exhibit this extreme behaviour by alternating the positions of these vertical lines all the way until $t = 1$.

Exercise 4.2

We use the stable_test function to create a 1×10 vector whose entries are a logical true (i.e. 1) if a stable solution occurs, or a logical false (i.e. 0) otherwise. The $(1, k)$-th entry corresponds to the type of solution for $\lambda = 0.k$, for the first 9 positive integers k and the last entry corresponds to $\lambda = 1$.

```
type stable_test.m
lambdaresults = [];
for lambda = 0.1 : 0.1 : 1
    lambdaresults = [lambdaresults stable_test(lambda)];
end
lambdaresults


function result = stable_test(lambda)
M = 20; % taking M = 20 as previously in the problem
e = ones(M-1,1);
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
I = speye(M-1);
A = I + lambda*L;
eigsvector = eigs(A); % returns the 6 largest abs of eigenvalues
        if max(abs(eigsvector)) <= 1
            result = true; % when every abs(eigenvalue)<=1
        else
            result = false; % if there exists an abs(eigenvalue)>1
        end
end


lambdaresults =

     1     1     1     1     1     0     0     0     0     0
```

Hence, if we run the stable_test.m code for $M = 20$, we see that our solutions will be stable as long as λ is strictly smaller than 0.6, i.e. it enables a stable solution for the first 5 values of λ and an unstable one from this point onwards.

Exercise 4.3

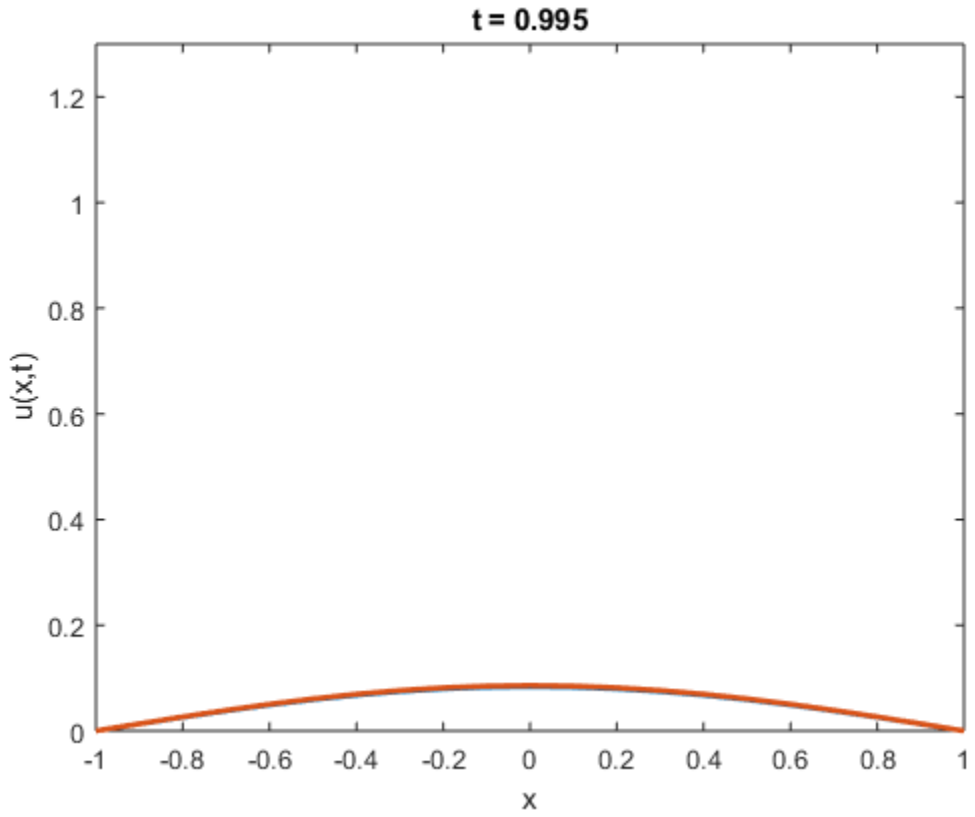
```
type heat_eqn1.m


function [u, u_exact] = heat_eqn1(T,M,lambda)

%set up grid points
h = 2/M; % grid spacing for splitting the x-axis
x = (linspace(-1,1,M+1))'; % splitting the interval [-1,1]
k = lambda*h^2; % grid spacing for going ahead in time
N = ceil(T/k);

%initial conditions
```

```
u = cos(pi*x/2) + (sin(pi*x))/2; % u(x,0)
t = 0;

%plot u(x,0)
plot(x,u,'b','LineWidth',2); % plotting the function u(x,0)
axis([-1 1 0 1.3]); % setting the min/max values of the axes displayed
title('t = 0'); % adding a title that displays the initial time
xlabel('x'); % labeling x-axis
ylabel('u'); % labeling y-axis

%finite difference operator
e = ones(M-1,1); % generating an (M-1)x1 column vector of 1s only
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
% tridiagonal and sparse, with main diag of -2s, adjacent of -1s
I = speye(M-1); % sparse identity matrix
A = I + lambda*L; % A as defined in the problem

%main loop
while (t<T)
    u_inner = u(2:M,1); % gets the M-1 inner components of U^n
 u_inner = A*u_inner; % calculating solution forward in time
    u = [0; u_inner; 0]; % adding the boundary conditions
    %plot current solution
    hold off
    plot(x,u,'LineWidth',2); % plotting u(x,t)
    hold on
    u_dash = exp(-((pi^2)*t)/4)*cos(pi*x/2)+(exp(-
(pi^2)*t)*(sin(pi*x)))/2;
    plot(x,u_dash,'LineWidth',2); % plotting the true solution
    axis([-1 1 0 1.3]); % setting the min/max of the axes
 title(['t = ' num2str(t)]); % converts t to character to display
    xlabel('x'); % labeling x-axis
    ylabel('u(x,t)'); % labeling y-axis
 pause(0.02); % pausing to display solution sufficiently long
    t = t + k; % going ahead in time by the specified amount
end
hold off

u_exact = exp(-((pi^2)*t)/4)*cos(pi*x/2)+(exp(-
(pi^2)*t)*(sin(pi*x)))/2;
% actual solution at endpoint as after loop, t is final time

end
```

Exercise 4.4

```
[u u_exact] = heat_eqn1(1,20,0.5);
error = norm(u - u_exact, inf)
```

```
error =

   8.6179e-04
```



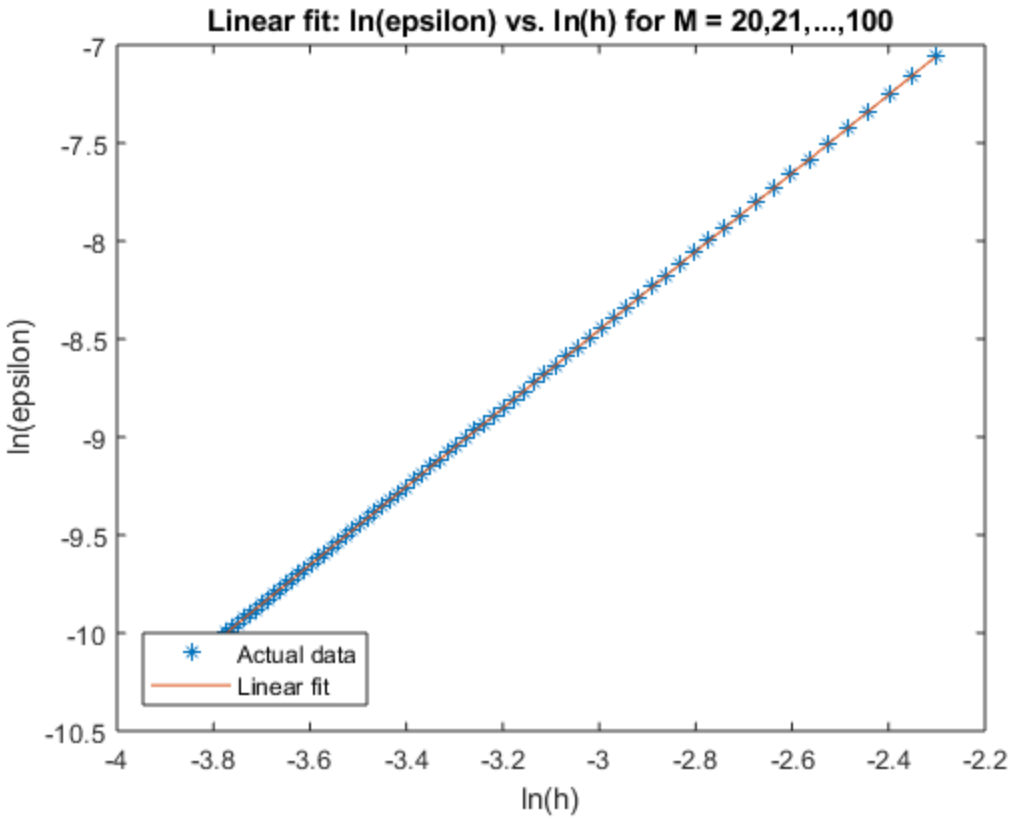
Exercise 4.5

In this case, the heat_eqn1 code has been altered notably to obtain heat_eqn1error: instead of outputting the exact and the numerical approximation of the solution, we now directly get the value of the infinite norm (our convention for the error). Furthermore, the code is vectorized, and operations including logarithming and applying the heat_eqn1error function are carried out in a component-wise manner as shown below.

```
type heat_eqn1error.m
```

```
mvector = []; % starting off with an empty vector
for M = 20:1:100 % looping over desired values of M
    mvector = [mvector 2/M]; % generating [2/20 2/21 2/22 ... 2/100]
end
```

```
x = log(mvector); % x = ln(epsilon) as defined in the question
y = log(arrayfun(@(M) heat_eqn1error(1, M, 0.5),
 linspace(20,100,81)));
% applying the function component-wise for M = {20, 21, ..., 100}
p = polyfit(x, y, 1) % linear regression for the data above
f = polyval(p,x); % returns a linear function based on polyfit
plot(x,y,'*',x,f,'-')
xlabel('ln(h)'); % labeling x-axis
ylabel('ln(epsilon)'); % labeling y-axis
legend('Actual data','Linear fit','Location','southwest')
title('Linear fit: ln(epsilon) vs. ln(h) for M = 20,21,...,100')


function [errorM] = heat_eqn1error(T,M,lambda)

%set up grid points
h = 2/M; % grid spacing for splitting the x-axis
x = (linspace(-1,1,M+1))'; % splitting the interval [-1,1]
k = lambda*h^2; % grid spacing for going ahead in time
N = ceil(T/k);

%initial conditions
u = cos(pi*x/2) + (sin(pi*x))/2; % ux,0)
t = 0; % starting at 0 time

%finite difference operator
e = ones(M-1,1); % generating an (M-1)x1 column vector of 1s only
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
% tridiagonal and sparse, with main diag of -2s, adjacent of -1s
I = speye(M-1); % sparse identity matrix
A = I + lambda*L; % A as defined in the problem

%main loop
while (t<T)
    u_inner = u(2:M,1); % gets the M-1 inner components of U^n
 u_inner = A*u_inner; % calculating solution forward in time
    u = [0; u_inner; 0]; % adding the boundary conditions
    t = t + k; % going ahead in time by the specified amount
end

u_exact = exp(-((pi^2)*t)/4)*cos(pi*x/2)+(exp(-
(pi^2)*t)*(sin(pi*x)))/2;
% actual solution at endpoint as after loop, t is the final time

errorM = norm(u - u_exact, inf); % the infinite norm = error

end

p =

    1.9986   -2.4580
```



From the linear regression model, we can see that the slope is approximately 1.9986, hence after getting rid of the logarithms, we obtain approx. that $\varepsilon \propto h^2$, with constant of proportionality equal to around $e^{-2.4580}$. Therefore ε and h satisfy approximately a positive quadratic relationship. This makes sense intuitively, as when we increase M (i.e. generate smaller steps for the x values) then h decreases. Hence, when using the finite difference approximations, we use points increasingly closer to the ones we were given before, thus (taking into account that our function is continuous and behaves well) the calculated estimates for the derivatives are better, so we should expect to see smaller errors.

Exercise 4.6

```
type heat_eqn2.m
```

```
function u = heat_eqn2(T,M,lambda)

%set up grid points
h = 2/M; % grid spacing for splitting the x-axis
x = (linspace(-1,1,M+1))'; % splitting the interval [-1,1]
k = lambda*h^2; % grid spacing for going ahead in time
N = ceil(T/k);

%initial conditions
u = cos(pi*x/2) + (sin(pi*x))/2; % u(x,0)
t = 0;
```

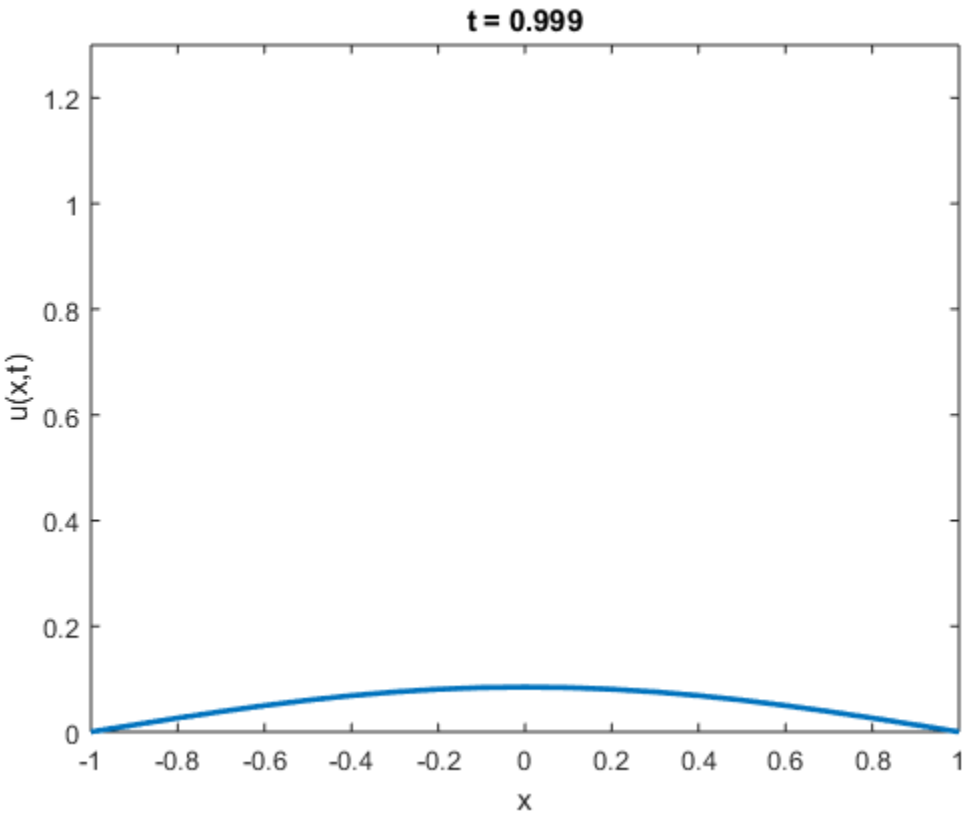
```
%plot u(x,0)
plot(x,u,'b','LineWidth',2); % plotting the function u(x,0)
axis([-1 1 0 1.3]); % setting the min/max values of the axes displayed
title('t = 0'); % adding a title that displays the initial time
xlabel('x'); % labeling x-axis
ylabel('u'); % labeling y-axis

%finite difference operator
e = ones(M-1,1); % generating an (M-1)x1 column vector of 1s only
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
% tridiagonal and sparse, with main diag of -2s, adjacent of -1s
I = speye(M-1); % sparse identity matrix
B = inv(I - (lambda/2)*L)*(I + (lambda/2)*L); % B as defined

%main loop
while (t<T)
    u_inner = u(2:M,1); % gets the M-1 inner components of U^n
 u_inner = B*u_inner; % calculating solution forward in time
    u = [0; u_inner; 0];
    %plot current solution
    hold off
    plot(x,u,'LineWidth',2); % plotting u(x,t)
    axis([-1 1 0 1.3]); % setting the min/max of the axes
 title(['t = ' num2str(t)]); % converts t to character to display
    xlabel('x'); % labeling x-axis
    ylabel('u(x,t)'); % labeling y-axis
 pause(0.02); % pausing to display solution sufficiently long
    t = t + k; % going ahead in time by the specified amount
end
```

Exercise 4.7

Part a)

```
heat_eqn2(1,20,0.3)


ans =

         0
    0.0133
    0.0262
    0.0385
    0.0498
    0.0599
    0.0686
    0.0755
    0.0806
    0.0838
    0.0848
    0.0838
    0.0807
    0.0756
```

```
    0.0686
    0.0600
    0.0499
    0.0385
    0.0262
    0.0133
         0
```

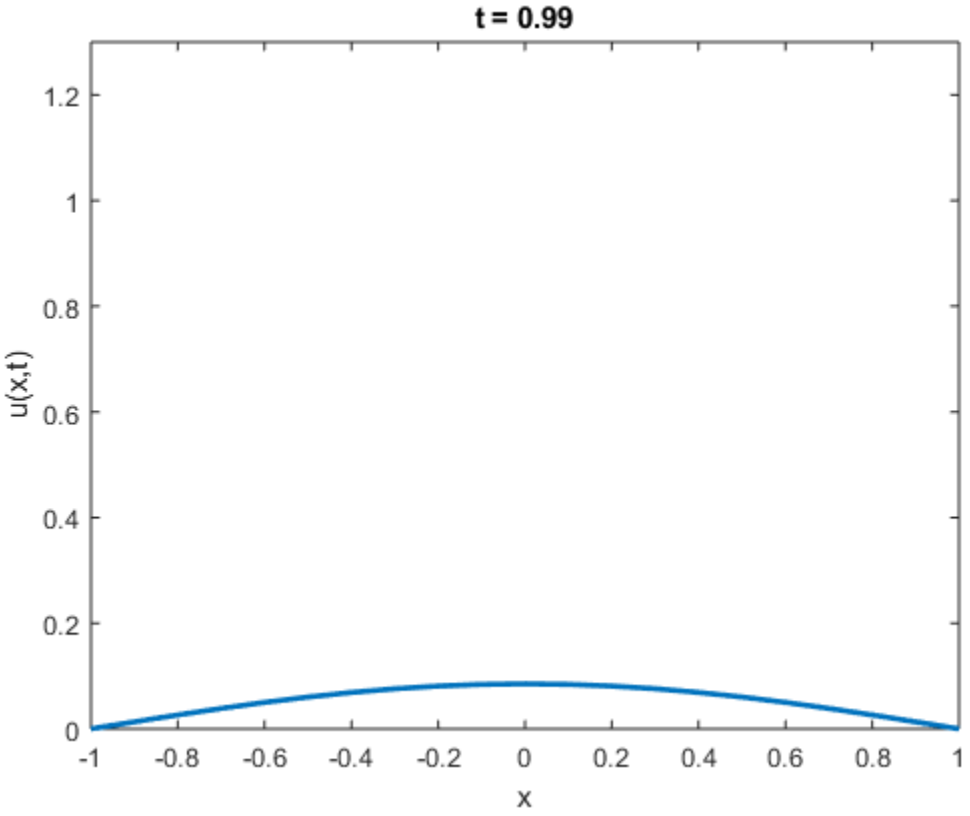


Part b)

```
heat_eqn2(1,20,1)
```

```
ans =

         0
    0.0133
    0.0263
    0.0387
    0.0501
    0.0602
    0.0689
    0.0759
    0.0810
    0.0842
```

```
0.0852
0.0842
0.0811
0.0760
0.0690
0.0603
0.0501
0.0387
0.0264
0.0133
     0
```



For $\lambda = 0.3$, the solution is again stable: the heat is being distributed more and more evenly across the interval, and ultimately it is in the shape reminiscent of a low-arcing, symmetric parabola with the peak heat having shifted to where $x = 0$. In other words, it behaves almost identically to the numerical approximation initially for the same value of λ. Furthermore, for $\lambda = 1$, the difference from initially is notable: the odd vertical line behaviour no longer exists, and instead we have an analogous behaviour to that for $\lambda = 0.3$, i.e. our solution still is bounded and stable. This is consistent with the observation in the project booklet that solutions will stay bounded when using the more elaborate method regardless of the value of λ.

Exercise 4.8

Generalizing the component-wise expressions to obtain a matrix form for the equations, we obtain the following:

$$(I - \frac{\lambda}{2}L)U^{n+1} = ((\beta k + 1)I + \frac{\lambda}{2}L)U^n - \beta k(U^n)^3$$

where the last vector is cubed component-wise. Hence, we can set:

$$C = (I - \frac{\lambda}{2}L)^{-1}$$

$$D = (\beta k + 1)I + \frac{\lambda}{2}L.$$

Our problem then becomes:

$$U^{n+1} = C(DU^n - \beta k(U^n)^3)$$

where again we have component-wise cubing. This is reflected in the code of the allen.cahn function below, where this step is repeated in the iterative while loop.

```
type allen_cahn.m
```

```
function u = allen_cahn(T,M,lambda,beta)

%set up grid points
h = 2/M; % grid spacing for splitting the x-axis
x = (linspace(-1,1,M+1))'; % splitting the interval [-1,1]
k = lambda*h^2; % grid spacing for going ahead in time
N = ceil(T/k);

%initial conditions
u = cos(pi*x/2) + (sin(pi*x))/2; % u(x,0)
t = 0;

%plot u(x,0)
plot(x,u,'b','LineWidth',2); % plotting the function u(x,0)
axis([-1 1 0 1.3]); % setting the min/max values of the axes displayed
title('t = 0'); % adding a title that displays the initial time
xlabel('x'); % labeling x-axis
ylabel('u'); % labeling y-axis

%finite difference operator
e = ones(M-1,1); % generating an (M-1)x1 column vector of 1s only
L = spdiags([e  -2*e  e], [-1 0 1], M-1, M-1);
% tridiagonal and sparse, with main diag of -2s, adjacent of -1s
I = speye(M-1); % sparse identity matrix
C = inv(I - (lambda/2)*L);
D = ((lambda/2)*L + (beta*k + 1)*I);

%main loop
while (t<T)
    u_inner = u(2:M,1); % gets the M-1 inner components of U^n
 u_inner = C*(D*u_inner -  beta*k*u_inner.^3); % described above
    u = [0; u_inner; 0]; % adding boundary conditions
    %plot current solution
    hold off
```

```
    plot(x,u,'LineWidth',2); % plotting u(x,t)
    axis([-1 1 0 1.3]); % setting the min/max of the axes
 title(['t = ' num2str(t)]); % converts t to character to display
    xlabel('x'); % labeling x-axis
    ylabel('u(x,t)'); % labeling y-axis
 pause(0.02); % pausing to display solution sufficiently long
    t = t + k; % going ahead in time by the specified amount
end
```

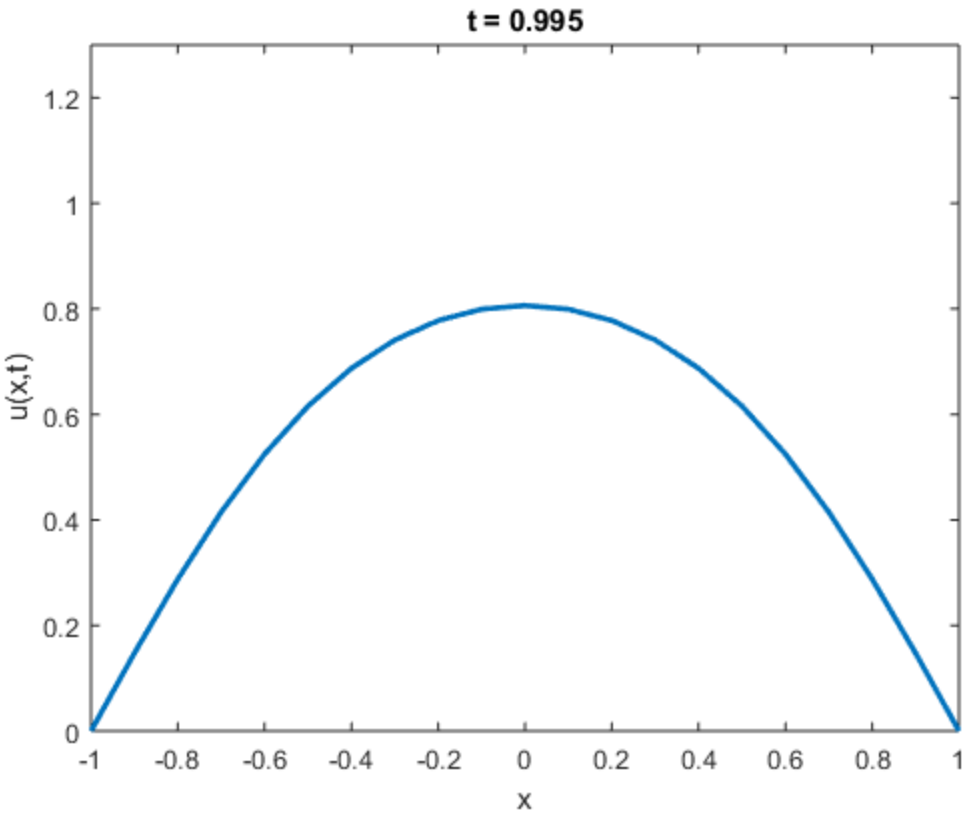
Exercise 4.9

For aesthetic and spatial purposes, we again place the vector solutions in a matrix (as we did previously for λ). The k-th column of the matrix represents the solution for $\beta = k$, where $1 \leq k \leq 5$. Generate also a solution matrix for $t = 2000$ using the allen_cahn_infinite code, which only suppresses the animation. The first matrix is called acmatrix, whereas the latter is infmatrix.

```
acmatrix = [];
for beta  = 1:5
   acmatrix = [acmatrix allen_cahn(1,20,0.5,beta)];
   % placing the solutions in a 21x5 matrix
end
acmatrix

infmatrix = [];
for beta  = 1:5
   infmatrix = [infmatrix allen_cahn_infinite(2000,20,0.5,beta)];
   % placing the solutions in a 21x5 matrix
end
infmatrix
```

```
acmatrix =

         0         0         0         0         0
    0.0294    0.0572    0.0920    0.1233    0.1475
    0.0581    0.1129    0.1810    0.2416    0.2878
    0.0854    0.1656    0.2645    0.3508    0.4148
    0.1105    0.2140    0.3401    0.4476    0.5247
    0.1329    0.2569    0.4061    0.5299    0.6156
    0.1520    0.2933    0.4611    0.5969    0.6873
    0.1673    0.3224    0.5044    0.6483    0.7409
    0.1786    0.3436    0.5356    0.6845    0.7778
    0.1855    0.3564    0.5543    0.7059    0.7993
    0.1878    0.3608    0.5606    0.7130    0.8064
    0.1855    0.3565    0.5544    0.7060    0.7993
    0.1786    0.3436    0.5356    0.6845    0.7778
    0.1674    0.3225    0.5045    0.6483    0.7409
    0.1521    0.2934    0.4612    0.5969    0.6874
    0.1330    0.2570    0.4062    0.5300    0.6156
    0.1106    0.2141    0.3402    0.4477    0.5247
    0.0854    0.1657    0.2646    0.3509    0.4149
    0.0582    0.1130    0.1811    0.2417    0.2878
    0.0295    0.0573    0.0920    0.1233    0.1475
```

```
         0         0         0         0         0


infmatrix =

         0         0         0         0         0
    0.0000    0.0000    0.0789    0.1220    0.1474
    0.0000    0.0000    0.1554    0.2391    0.2876
    0.0000    0.0000    0.2274    0.3473    0.4147
    0.0000    0.0000    0.2929    0.4432    0.5245
    0.0000    0.0000    0.3503    0.5249    0.6154
    0.0000    0.0000    0.3986    0.5914    0.6871
    0.0000    0.0000    0.4368    0.6425    0.7407
    0.0000    0.0000    0.4644    0.6785    0.7776
    0.0000    0.0000    0.4810    0.6998    0.7991
    0.0000    0.0000    0.4866    0.7069    0.8061
    0.0000    0.0000    0.4810    0.6998    0.7991
    0.0000    0.0000    0.4644    0.6785    0.7776
    0.0000    0.0000    0.4368    0.6425    0.7407
    0.0000    0.0000    0.3986    0.5914    0.6871
    0.0000    0.0000    0.3503    0.5249    0.6154
    0.0000    0.0000    0.2929    0.4432    0.5245
    0.0000    0.0000    0.2274    0.3473    0.4147
    0.0000    0.0000    0.1554    0.2391    0.2876
    0.0000    0.0000    0.0789    0.1220    0.1474
         0         0         0         0         0
```



The Allen-Cahn equation provides us with a trend that is quite a bit different from the ones previously observed, whereby the heat tended to zero symmetrically. Consider as previously the case when $\lambda = 0.5$. In the animations obtained now, the heat initially behaves analogously, as it decreases, and tends to a parabolic-like shape symmetric about the line $x = 0$. For the initial values of $\beta = 1, 2$, the final solutions seems to be tending very closely to 0, although in the latter case this convergence is somewhat slower. From $\beta = 3$ and onwards, the solutions tend to a parabolic-likerelationship that is now quite taller than in the previous cases, and are clearly distinct from the zero solutions. The peaks of these "parabolae" are greater as β increases. Upon a conversation with my tutor, I was advised to conceptually think of the Allen-Cahn equation as being governed by a (chemical) reaction in addition to the flow of the heat. After heat flows away from the peak at the right of $x = 0$, the value of $u(x, t)$ drops below 1 everywhere. As soon as $u < 1$, $\beta(u - u^3) > 0$, as the cubic term has less of an effect than the linear one. Note that further in time, as the solution "stabilizes" around a certain value of $u(x, t)$, then u_t approximately vanishes. Hence, the total contribution of the right-hand side must be approximately zero, hence u_{xx} is of about the same magnitude as $\beta(u - u^3)$ but negative. This is consistent with the observation that as β increases so does the net effect of the reaction, and there is an increase in the magnitude of u_{xx} so that each "parabola" is steeper (and taller) than the one for the previous smaller value of β. For completeness, I have included an allen_cahn_infinite.m code which takes the same inputs as the allen_cahn function, yet skips the animation and only outputs the solution vector. I repeatedly used this in the command window to consolidate the behaviour as t becomes very large, say 10000 or 20000, so I concluded that the negligible change at these great times should support my conjecture.

Published with MATLAB® R2017b