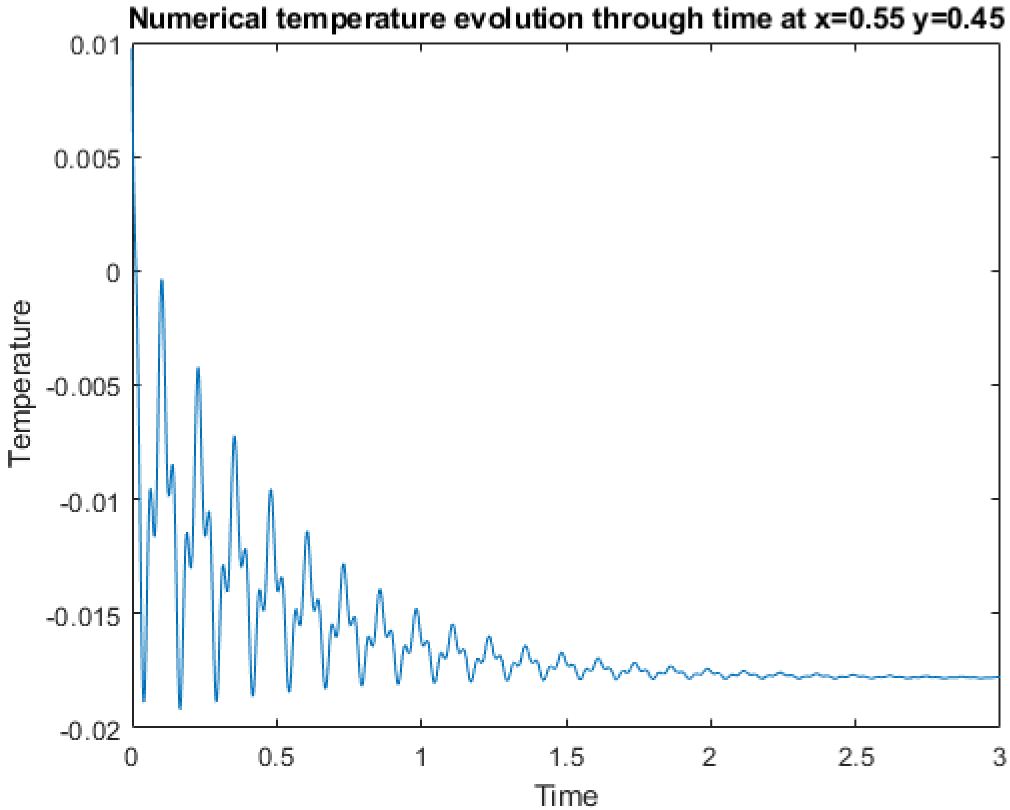
```
% A 2D unsteady diffusion(heat) equation with a heat source is to be solved in
 the square domain 0<=x<=1, 0<=y<=1
% The equation is: dTdt = alpha(d^2T/dx^2+d^2T/dy^2) + Q(x,y,t), where Q is
 the internal heat source term
% Q heat source term, initial condition, boundary conditions and all variables
 are given as follows:
% Q(x,y,t)=2.5*sin(4*pi*x)*sin(8*pi*y)*f(t), where f(t) is 1-(e^
#at*sin(omega*t)*cos(2*omega*t)).
% Tinitial=0.01*sin(pi*x)*sin(pi*y)
% T=0 on all four boundaries
% alpha=0.1, a=2 and omega=50

% To numercially solve this problem, we first discretize the spacial
 derivatives with 2nd order central difference(FDA),
% then advance through time using the Crank-Nicolson method, finally
% solving the result system of equations using the ADI method


% Setting up all the necessary parameters and the grid space
M=81; N=81; %spacial grid points for the x and y direction
h = 0.0125; %delta x and delta y
[X,Y] = meshgrid(0:h:1,0:h:1);
omega=50;
a=2;
alpha=0.1;
tstep= 0.002; %time step size
totaltime = 3; %we assume that within 3 seconds, the solution will reach
 steady state
tvec=0:tstep:totaltime;%time vector


%Setting up initial condition and source term
Temp=zeros((M-2),(N-2),length(tvec));%3D Temperature matrix without the BC
Tinitial=0.01 * sin(pi*X) .* sin(pi*Y); %initial value in every point
Tinitialnobc =Tinitial(2:end-1,2:end-1);% removing the values in 4 boundaries
Temp(:,:,1)=Tinitialnobc; %plugging the initial values into 3D temperature
 matrix
sourceq = 2.5*sin(4*pi*X).*sin(8*pi*Y);%heat source term
sourceq1 = sourceq(2:end-1,2:end-1);%removing boundary values
Heatsource(:,:,1)=sourceq1;

%Setting up the 3D source term matrix
for qq=2:(length(tvec))
    Heatsource(:,:,qq)=sourceq1.*(1 - exp(-
a.*tvec(qq)).*sin(omega.*tvec(qq)).*cos(2*omega*tvec(qq)));
end

%Setting up Heat source cell matrix for easy viewing and debugging
for yy=1:(length(tvec))
    Heatsourcecell{yy}=Heatsource(:,:,yy);
end
```

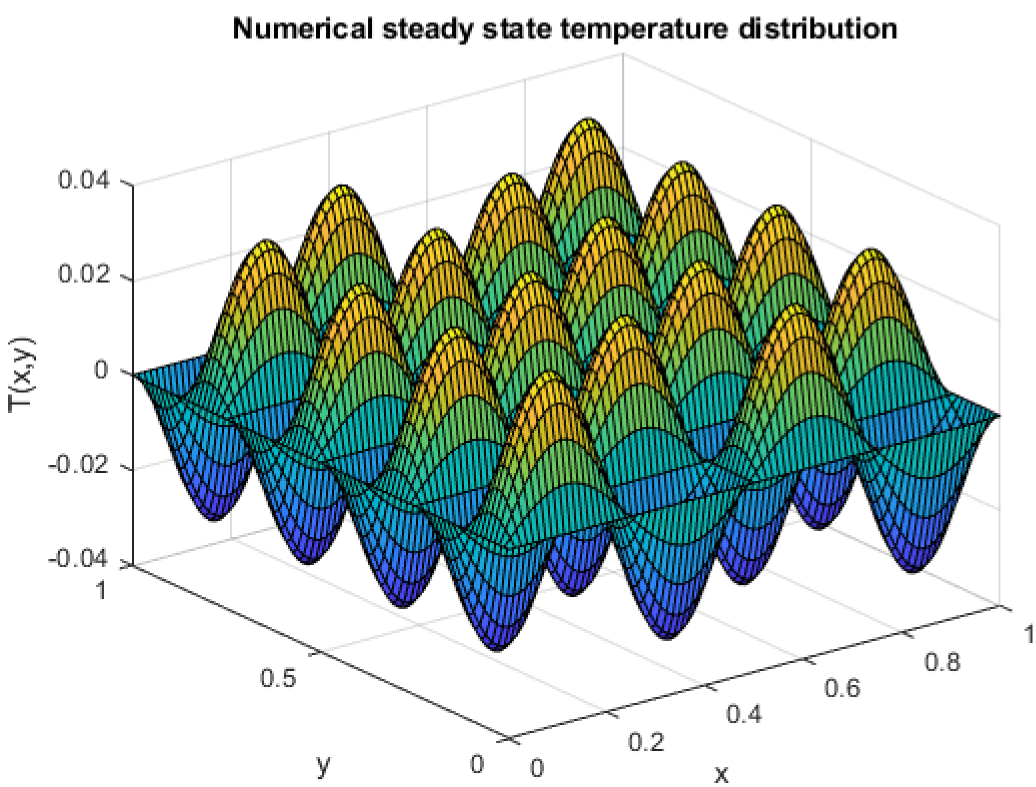
```
% The following tridiagonal matrices arise from the ADI method we are
 employing
I = eye(N-2);
Tridiag = zeros(N-2);
for i=1:N-3
    Tridiag(i,i) = -2;
    Tridiag(i,i+1) = 1;
    Tridiag(i+1,i) = 1;
end
Tridiag(N-2,N-2) = -2;
k=(alpha*tstep/(2*h^2));
IAxminus=(I-k*Tridiag);
IAyminus=(I-k*Tridiag);
IAxplus=(I+k*Tridiag);
IAyplus=(I+k*Tridiag);

%Setting up ADI method
zeta=zeros(M-2,N-2);
rhs1=zeros(M-2,N-2);
for tt=1:length(tvec)-1 %time count
    for zz=1:N-2 %calculating zeta matrix
     zeta(:,zz)=IAyplus*Temp(:,zz,tt);
    end

    for hh=1:N-2 %calculating rhs without the source term
     rhs1(hh,:)=IAxplus*zeta(hh,:)';
    end

  rhs=rhs1+tstep.*Heatsource(:,:,tt);%adding the heat source term to rhs

  for jj=1:N-2 %solving for IAxminus*z=rhs
   z(jj,:)=tridiag(IAxminus,rhs(jj,:));
  end
  for ii=1:M-2 %solving for IAyminus*temp=z
   Temp(:,ii,tt+1)=tridiag(IAyminus,z(:,ii));
  end
  tt; %loop count
end

for nn=1:length(tvec) %putting temp in cell matrix for easy viewing
   Tempcell{nn}=Temp(:,:,nn);
end

Tempwithbc=zeros(M,N,length(tvec));%creating the real Temp matrix with the
 Boundaries
for bb=1:length(tvec)%inserting Temp into real Temp matrix
Tempwithbc(2:end-1,2:end-1,bb)=Temp(:,:,bb);
end

for cc=1:length(tvec) %putting temp in cell matrix for easy viewing
   Tempwithbccell{cc}=Tempwithbc(:,:,cc);
end

%We plot the Temperature evolution at a specific point through time to
```

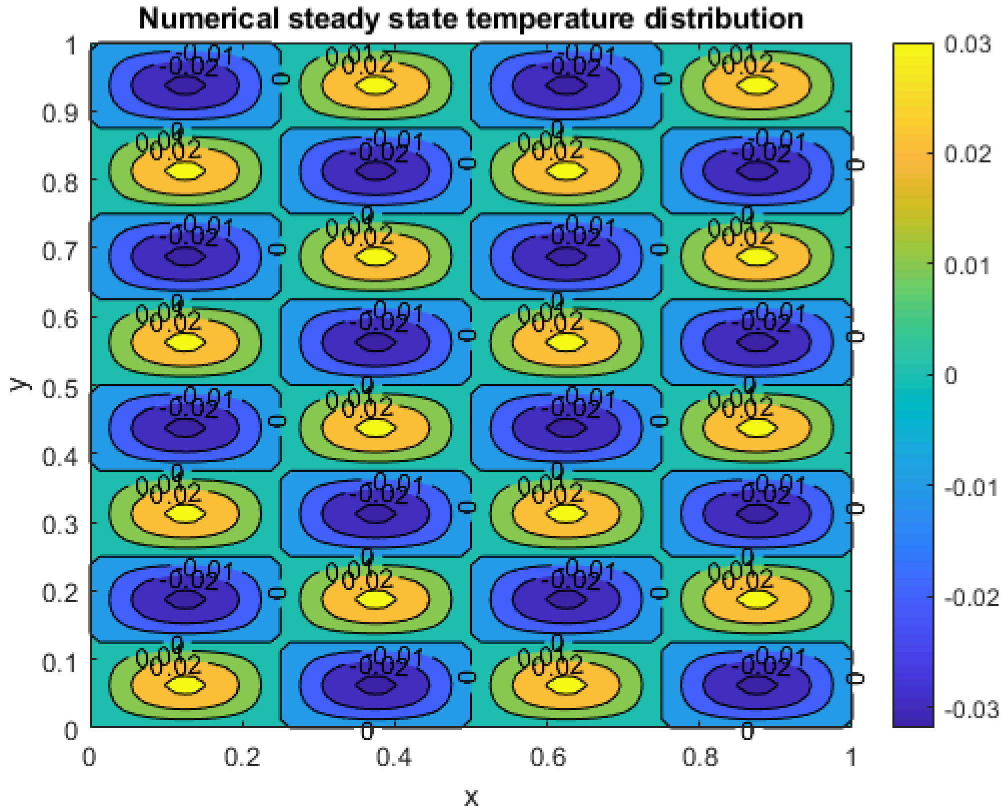
```
%verify if 3 seconds is enough to reach steady state
Tempwatpointvec=zeros(1,length(tvec));%Tempvector at x=55 y=45
for pp=1:(length(tvec))%putting all values at x=55 y=45 into a vector
Tempwatpointvec(1,pp)=Tempwithbc(45,37,pp);
end

%plotting
figure(1)
plot(tvec,Tempwatpointvec)
xlabel('Time')
ylabel('Temperature')
title('Numerical temperature evolution through time at x=0.55 y=0.45')
figure(2)
surf(X,Y,Tempwithbc(:,:,end))
xlabel('x')
ylabel('y')
zlabel('T(x,y)')
title('Numerical steady state temperature distribution')
figure(3)
contourf(X,Y,Tempwithbc(:,:,end),'ShowText','on')
colorbar
xlabel('x')
ylabel('y')
zlabel('T(x,y)')
title('Numerical steady state temperature distribution')


%We can assume the exact steady state solution to be:T(x,y) =
%K*sin(4#x)*sin(8#y), where K is a constant to be determined
%After solving analytically, K is 5/(16*pi^2)

%Exact solution is below
Tempexact=(5/(16*pi^2)).*sin(4*pi*X).*sin(8*pi*Y);
%plotting
figure(4)
contourf(X,Y,Tempexact,'ShowText','on')
colorbar
title('Exact steady State solution of Temperature')
xlabel('X')
ylabel('Y')
figure(5)
surf(X,Y,Tempexact)
title('Exact steady State solution of Temperature')
xlabel('X')
ylabel('Y')
zlabel('T(x,y)')

%Comparing the exact steady state solution with the numerical solution
Temperr=Tempexact-Tempwithbc(:,:,end);
figure(6)
contourf(X,Y,Temperr,'ShowText','on')
colorbar
xlabel('X')
ylabel('Y')
```

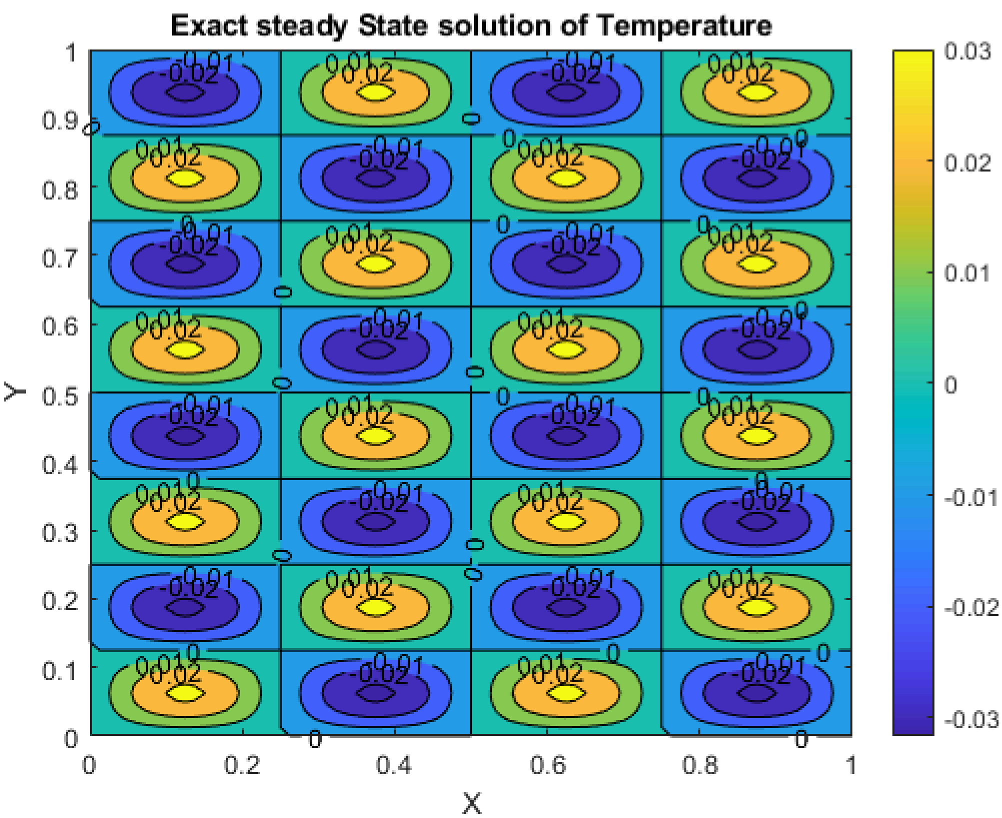
```
title('Error of steady state solutions (Exact vs Numerical)')


%Function for solving A*r=y, which arises from the ADI method that we are
%employing
function R = tridiag(A, Y)

n = size(A, 1);
a = diag(A, -1);  % subdiagonal
b = diag(A);      % diagonal
c = diag(A, 1);   % superdiagonal

% Forward elimination
c_star = zeros(n, 1);
d_star = zeros(n, 1);
c_star(1) = c(1) / b(1);
d_star(1) = Y(1) / b(1);
for i = 2:n-1
    temp = b(i) - a(i-1)*c_star(i-1);
    c_star(i) = c(i) / temp;
    d_star(i) = (Y(i) - a(i-1)*d_star(i-1)) / temp;
end
d_star(n) = (Y(n) - a(n-1)*d_star(n-1)) / (b(n) - a(n-1)*c_star(n-1));

% Back substitution
R = zeros(n, 1);
R(n) = d_star(n);
for i = n-1:-1:1
    R(i) = d_star(i) - c_star(i)*R(i+1);
end
end
```

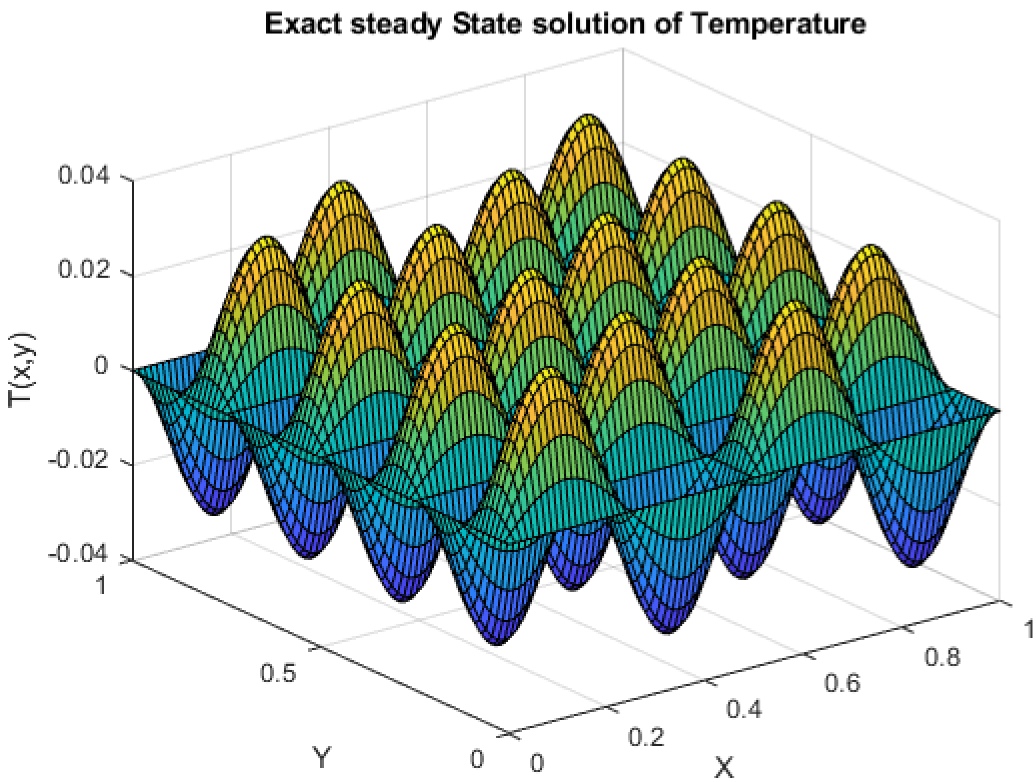


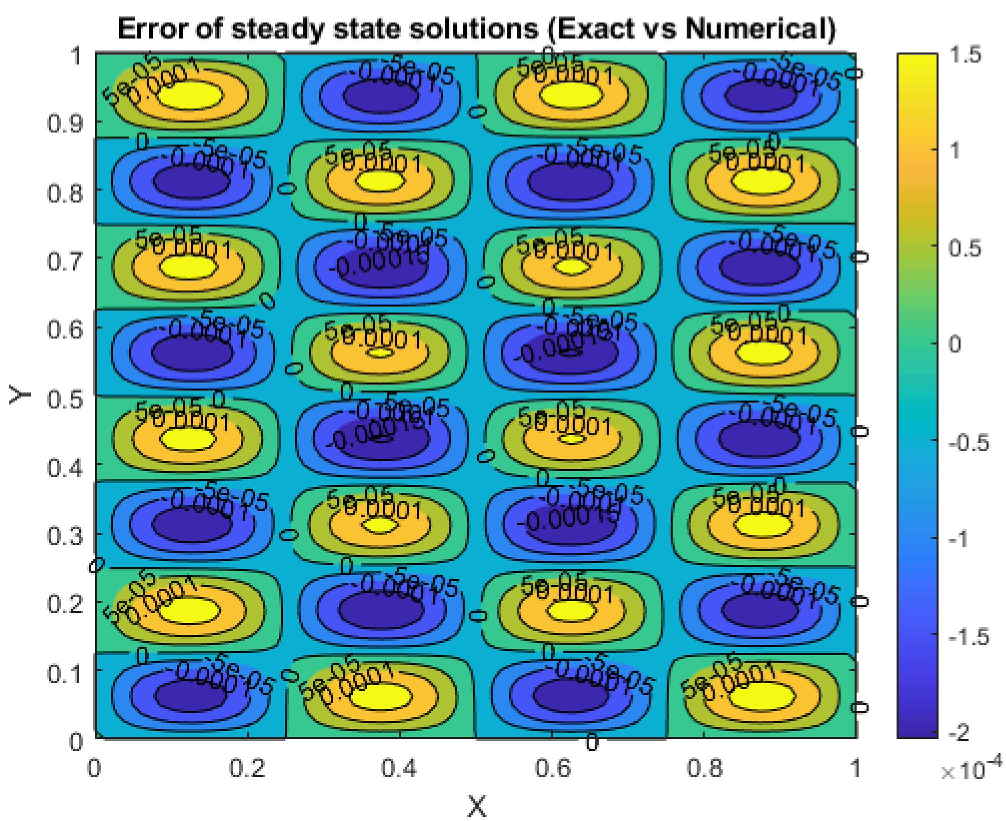
From the plot we can see that the temperature does reach steady state when time is 3 seconds, therefore justifying our choice for totaltime=3




Numerical steady state temperature distribution
y
x
0.03
0.02
0.01
0
-0.01
-0.02
-0.03


Exact steady State solution of Temperature
Y
X
0.03
0.02
0.01
0
-0.01
-0.02
-0.03





After obtaining both the exact and the numerical solution, we subtract the values at every point of these two solutions and plot the error values in a contour plot. The error is of the order of 10^-4.

Published with MATLAB® R2022a