Chapter 31-11: Parabolic PDEs Using the Explicit Method.

Marching in time is the easiest way to solve a parabolic equation. For this **explicit** method, the time steps must be small.

Defining Parabolic PDE's

•The general form for a second order linear PDE with two independent variables (x, y) and one dependent variable (u) is

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D = 0$$

•Recall the criteria for an equation of this type to be considered parabolic

$$B^2 - 4AC = 0$$

•For example, examine the heat-conduction equation given by

$$\alpha\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

where $A = \alpha$, $B = 0$, $C = 0$ and $D = -1$

then

$$B^2 - 4A\,C = 0 - 4(\alpha)(0) = 0$$

thus allowing us to classify the heat equation as parabolic.

A common example of the parabolic problem is the temperature in a heated rod.

The internal temperature of a metal rod exposed to two different temperatures at its two ends can be found using the heat conduction equation.

$$\alpha\frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t}$$

For a rod of length L divided into $n + 1$ nodes, $\Delta x = \frac{L}{n}$.

The time is similarly broken into time steps of Δt.

Hence T_i^j corresponds to the temperature at node i, that is,

$x = (i)(\Delta x)$ and time $t = (j)(\Delta t)$

If the definition $\Delta x = \frac{L}{n}$ is added, then the finite central divided difference approximation of the left hand side at a general interior node (i) can be written as

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{i,j} \cong \frac{T_{i+1}^j - 2T_i^j + T_{i-1}^j}{(\Delta x)^2}$$

where (j) is the node number along the time.

The time derivative on the right hand side is approximated by the forward divided difference method as

$$\left.\frac{\partial T}{\partial t}\right|_{i,j} \cong \frac{T_i^{j+1} - T_i^j}{\Delta t}$$

Substituting these approximations into the governing equation yields

$$\alpha \frac{T_{i+1}^{j} - 2T_{i}^{j} + T_{i-1}^{j}}{(\Delta x)^2} = \frac{T_{i}^{j+1} - T_{i}^{j}}{\Delta t}$$

Solving for the temp at the time node $j+1$ gives

$$T_{i}^{j+1} = T_{i}^{j} + \alpha \frac{\Delta t}{(\Delta x)^2} (T_{i+1}^{j} - 2T_{i}^{j} + T_{i-1}^{j})$$

choosing,

$$\lambda = \alpha \frac{\Delta t}{(\Delta x)^2}$$

we can write the equations as

$$T_{i}^{j+1} = T_{i}^{j} + \lambda (T_{i+1}^{j} - 2T_{i}^{j} + T_{i-1}^{j}).$$

•This equation can be solved explicitly because it can be written for each internal location node of the rod for time node $j+1$ in terms of the temperature at the time node.

•In other words, if we the temperature at node $j = 0$ is known, and the boundary temperatures, then the temperature at the next time step can be found.

•The process is continued by first finding the temperature at all nodes $j = 1$, and using these to find the temperature at the next time node, $j = 2$. This process continues until the time at which the temperature is of interest is reached.

The content of the last two green cells was gleaned from the following website: [https://nm.mathforcollege.com (https://nm.mathforcollege.com)](https://nm.mathforcollege.com)

1. Use the explicit scheme to solve the parabolic equation

$$u_t(x,t) = u_{xx}(x,t), xl < x < xr, 0 < t < tf$$

with the boundary conditions:

$$u(x,0) = f(x), xl < x < xr$$

$$u(0,t) = gl(t), u(1,t) = gr(t), 0 < t < tf$$

A special case is choosing f and g properly such that the analytic solution is:

$$u(x,t) = \sin(pi \cdot x)e^{(-pi^2 t)} + \sin(2pi)(x)e^{(-4pi^2 t)}$$

Solve this problem by the explicit scheme:

$$u(j,n+1) = u(j,n) + v(u(j+1,n) - 2u(j,n) + u(j-1,n))$$

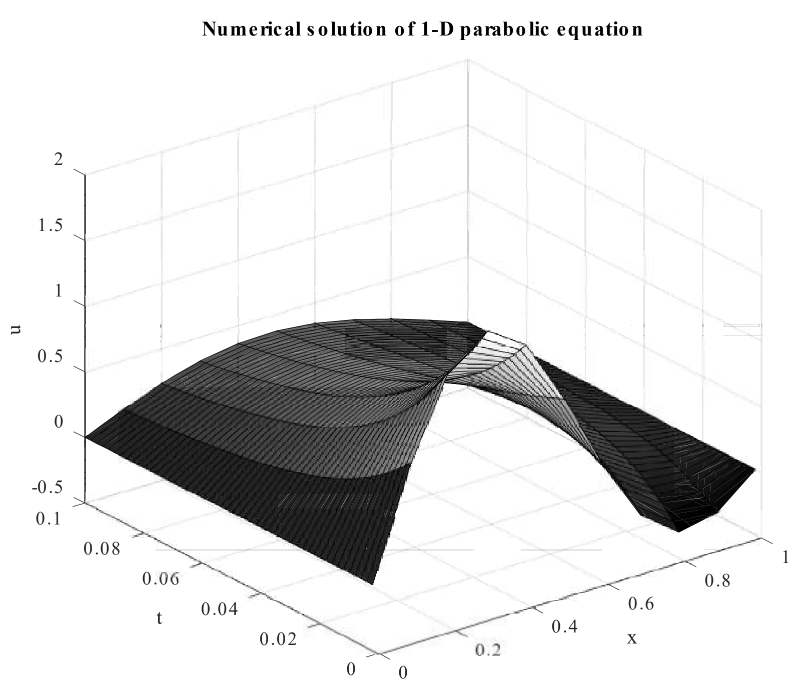
The following solution is as produced in Octaave

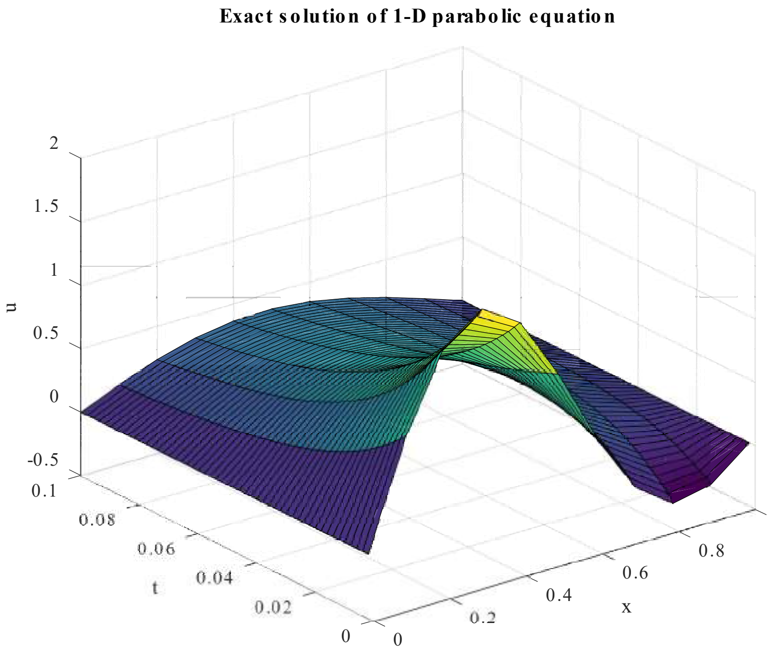
In [ ]:

```
clear all; % clear all variables in memory
xl=0; xr=1; % x domain [xl,xr]
J = 10; % J: number of division for x
dx = (xr-xl) / J; % dx: mesh size
tf = 0.1; % final simulation time
Nt = 50; % Nt: number of time steps
dt = tf/Nt;
mu = dt/(dx)^2;
if mu > 0.5 % make sure dt satisfy stability condition
  error('mu should < 0.5!')
end
% Evaluate the initial conditions
x = xl : dx : xr; % generate the grid point
% f(1:J+1) since array index starts from 1
f = sin(pi*x) + sin(2*pi*x);
% store the solution at all grid points for all time steps
u = zeros(J+1,Nt);
```

```
% Find the approximate solution at each time step
for n = 1:Nt
  t = n*dt; % current time
  % boundary condition at left side
  gl = sin(pi*xl)*exp(-pi*pi*t)+sin(2*pi*xl)*exp(-4*pi*pi*t);
  % boundary condition at right side
  gr = sin(pi*xr)*exp(-pi*pi*t)+sin(2*pi*xr)*exp(-4*pi*pi*t);
  if n==1 % first time step
    for j=2:J % interior nodes
    u(j,n) = f(j) + mu*(f(j+1)-2*f(j)+f(j-1));
    end
    u(1,n) = gl; % the left-end point
    u(J+1,n) = gr; % the right-end point
  else
    for j=2:J % interior nodes
    u(j,n)=u(j,n-1)+mu*(u(j+1,n-1)-2*u(j,n-1)+u(j-1,n-1));
    end
    u(1,n) = gl; % the left-end point
    u(J+1,n) = gr; % the right-end point
  end
  % calculate the analytic solution
  for j=1:J+1
    xj = xl + (j-1)*dx;
    u_ex(j,n)=sin(pi*xj)*exp(-pi*pi*t) ...
              +sin(2*pi*xj)*exp(-4*pi*pi*t);
  end
end

% Plot the results
tt = dt : dt : Nt*dt;
figure(1)
colormap(gray); % draw gray figure
surf(x,tt, u'); % 3-D surface plot
xlabel('x')
ylabel('t')
zlabel('u')
title('Numerical solution of 1-D parabolic equation')
figure(2)
surf(x,tt, u_ex'); % 3-D surface plot
xlabel('x')
ylabel('t')
zlabel('u')
title('Exact solution of 1-D parabolic equation')
```

Two plots are shown below. The first represents the numerical solution, and the second represents the exact solution. The impression is that they occupy the same space.





2. For the following problem involving the heat equation with a source term,

$$u_t = \beta u_{xx} + f(x,t), \qquad a < x < b, \quad t > 0$$

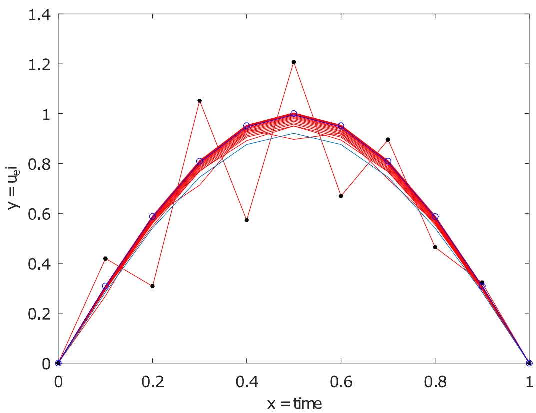
$$u(a,t) = g_1(t) \qquad u(b,t) = g_2(t) \qquad u(x,0) = u_0(x)$$

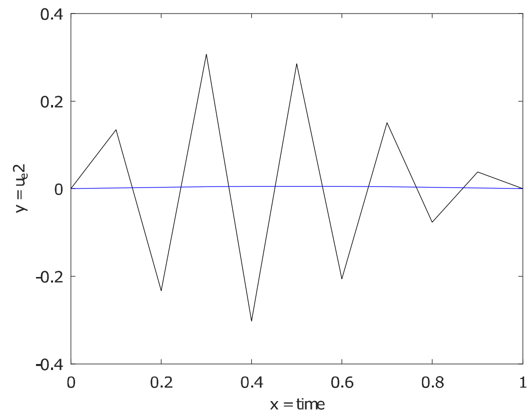
find a numerical solution for $u(x,t)$ at a particular time $T > 0$ or at certain times in the interval $0 < t < T$. Use the explicit branch of the finite differences method.

In [ ]:

```
clear; close all
a = 0; b =1;
m = 10; n=20;

h = (b-a)/m;
k2 = h^2/2;
k = 2*h^2/1.0;

t = 0;
for i=1:m+1,
   x(i) = a + (i-1)*h; y1(i) = uexact(t,x(i)); y2(i) = 0; y4(i)=0;
end

tau = k/h^2;
tau2 = k2/h^2;
plot(x,y1,'k'); hold

y3=y1; y4=y1;

t = 0; t2=0;
for j=1:n,
  y1(1)=0; y1(m+1)=0;
  for i=2:m
    y2(i) = y1(i) + tau*(y1(i-1)-2*y1(i)+y1(i+1)) + k*f(t,x(i));
    y4(i) = y3(i) + tau2*(y3(i-1)-2*y3(i)+y3(i+1)) + k2*f(t2,x(i));
  end
  plot(x,y2,'r'); pause(0.25)
  t = t + k;  t2=t2+k2;
  y1 = y2; y3=y4;
end

 for i=1:m+1
   u_e(i) = uexact(t,x(i));
```

```
  u_e2(i) = uexact(t2,x(i));
end

max(abs(u_e-y2)), max(abs(u_e-y4))

figure(1); plot(x,y2,"markersize", 9,'k.',x,u_e,x,y4,"markersize", 4,'bo',x,u_e2,'b')
xlabel('x = time','fontsize',10)
ylabel('y = u_ei','fontsize',10)

figure(2); plot(x,y2-u_e,'k', x,y4-u_e2,'b')
xlabel('x = time','fontsize',10)
ylabel('y = u_e2','fontsize',10)
```





Regarding the forward Euler problem worked above. The plot on the left of the computed solutions uses different time step sizes and the exact solution at some time for the test problem. An error plot of the computed solution is beyond the range of stability, though smaller step sizes are acceptable. In the plot on the right the error is excessive, unstable and quickly leads to uncontrolled growth.

The time step constraint $\Delta t = \frac{h^2}{2\beta}$ for the explicit Euler method is generally considered to be a severe restriction, e.g., if $h = 0.01$, the final time is $T = 10$ and $\beta = 100$, then we need 2×107 steps to get the solution at the final time. The backward Euler method does not have the time step constraint, but it is only first-order accurate. If we want second-order accuracy $O(h^2)$ we need to take $\Delta t = O(h^2)$. One finite difference scheme that is second-order accurate both in space and time, without compromising stability and computational complexity, is the Crank–Nicolson scheme.

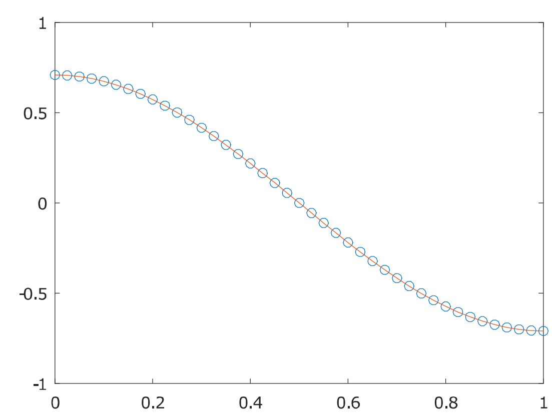
At each time step, we need to solve a tridiagonal system of equations to get U_i^{k+1}. The computational cost is only slightly more than that of the explicit Euler method in one space dimension, and we can take $\Delta t \simeq h$ and have second- order accuracy. Although the Crank–Nicolson scheme is an implicit method, it is much more efficient than the explicit Euler method since it is second- order accurate both in time and space with the same computational complexity.

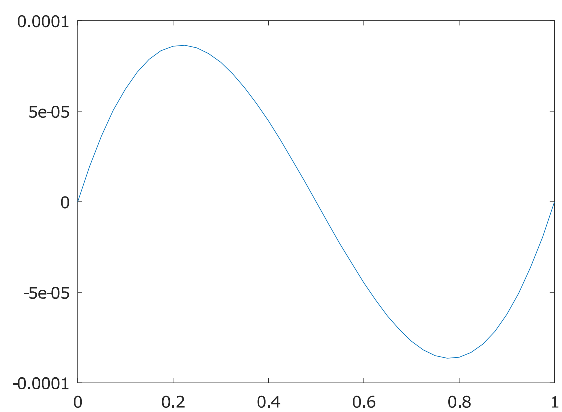
3. For the initial conditions shown as part of the blocks in the cell below, construct a Crank-Nicolson solution and plot.

In [ ]:

```
clear; close all

 a = 0;  b=1; m = 40;
 h = (b-a)/m;  k = h; h1=h*h;
```

```
tfinal = 5.5; n=fix(tfinal/k);

 t = 0;
 for i=1:m+1,
   x(i) = a + (i-1)*h;
   u0(i) = uexact(t,x(i));
 end

%-------- Set-up the coefficient matrix for Dirichlet BC -----------

A = sparse(m+1,m+1);
for i=2:m,
  A(i,i) = 1/k+1/h1; A(i,i-1) = -0.5/h1; A(i,i+1)= -0.5/h1;
end
  A(1,1) = 1;
  A(m+1,m+1) = 1;

  b = zeros(m+1,1);

%--------------- Time Iteration --------------------------------

 for j=1:n,

   for i=2:m
     b(i) = u0(i)/k + 0.5*(u0(i-1)-2*u0(i)+u0(i+1))/h1 + f(t+0.5*k,x(i));
   end
     b(1) =  g1(t+k);                      % Dirichlet BC at x =a.
     b(m+1) = g2(t+k);                     % Dirichlet BC at x =b.

   u1 = A\b;
   t = t + k;
   u0 = u1;
 end

 u_e = zeros(m+1,1);
 for i=1:m+1
   u_e(i) = uexact(t,x(i));
 end

 e_inf = max(abs(u_e-u1));
 disp('m, final_time, error'); [m,t,e_inf]
 plot(x,u1,'o',x,u_e)
 figure(2); plot(x,u1-u_e)
```





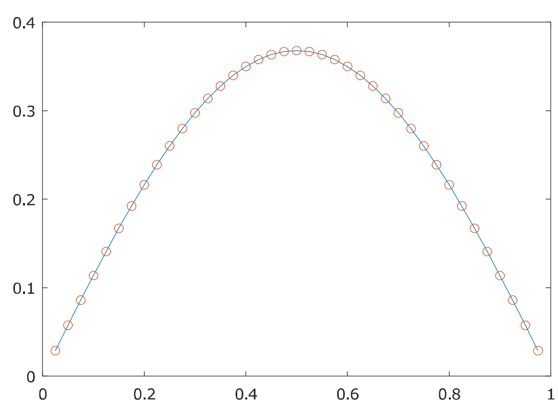
The method of lines, or MOL, is another explicit procedure within the finite differences method, and it is applicable to parabolic PDEs. With a good solver for ODEs or systems of ODEs, we can use the MOL to solve parabolic PDEs.

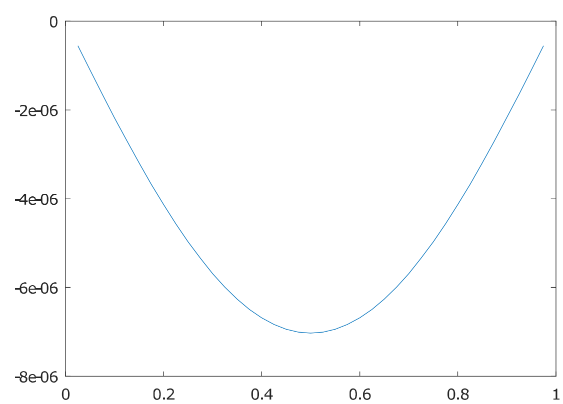
There are many efficient solvers for a system of ODEs. Most are based on high-order Runge–Kutta methods with adaptive time steps, e.g., ODE suite in Matlab, or dsode.f available through Netlib. However, it is important to recognize that the ODE system obtained from the MOL is typically stiff, i.e., the eigenvalues of A have very different scales. For example, for the heat equation the magnitude of the eigenvalues range from $O(1)$ to $O(\frac{1}{h^2})$.

4. For the initial conditions shown as part of the displayed banner in the cell below, construct a Method of Lines solution and plot.

In [ ]:

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                         %
%    Example of Method of Line for the heat equation                      %
%                                                                         %
%          u_t = u_{xx} + f(x,t)                                          %
%                                                                         %
%    Using Matlab built in function ode23s or ode15s.                     %
%                                                                         %
%    Test problem:                                                        %
%                 Exact solution: u(t,x) = exp(-t) sin(pi*x)              %
%                 Source term:    f(t,x) = exp(-t) sin(pi*x)(pi^2-1)      %
%                                                                         %
%    Files needed for the test:                                           %
%                                                                         %
%     mol.m:      This file, the main calling code.                       %
%     yfun_mol.m: The file defines the ODE system                         %
%     ux_mol.m:   The exact solution.                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear; close;

global n h x

a = 0; b=1; n=40; tfinal = 1;

h=(b-a)/n; tspan=[0,tfinal];

x= zeros(n-1,1);
for i=1:n-1;
  x(i) = a + i*h;
  y0(i) = ux_mol(0,x(i));
end

[t y] = ode23s('yfun_mol',tspan,y0);

[mr,nc] = size(y);  y_mol=y(mr,:);
for i=1:n-1,
  ye(i) = ux_mol(tfinal,x(i));
end

e = max(abs(ye-y_mol))
plot(x,ye); hold
plot(x,y_mol,'o')
figure(2); plot(x,y_mol-ye)
```





The last method to be discussed as part of the parabolic explicit tour is ADI (Alternating Direction Implicit). The ADI is a time splitting or fractional step method. The idea is to use an implicit discretization in one direction and an explicit discretization in another direction. For the heat equation $u_t = u_{xx} + u_{yy} + f(x, y, t)$, the ADI method is

$$\frac{U_{ij}^{k+\frac{1}{2}} - U_{i,j}^{k}}{(\Delta t)/2} = \frac{U_{i-1,j}^{k+\frac{1}{2}} - 2U_{ij}^{k+\frac{1}{2}} + U_{i+1,j}^{k+\frac{1}{2}}}{h_x^2} + \frac{U_{i,j-1}^{k} - 2U_{ij}^{k} + U_{i,j+1}^{k}}{h_y^2} + f_{ij}^{k+\frac{1}{2}}$$

$\Longrightarrow$

$$\frac{U_{ij}^{k+1} - U_{i,j}^{k+\frac{1}{2}}}{(\Delta t)/2} = \frac{U_{i-1,j}^{k+\frac{1}{2}} - 2U_{ij}^{k+\frac{1}{2}} + U_{i+1,j}^{k+\frac{1}{2}}}{h_x^2} + \frac{U_{i,j-1}^{k+1} - 2U_{ij}^{k+1} + U_{i,j+1}^{k+1}}{h_y^2} + f_{ij}^{k+\frac{1}{2}}$$

which is second order in time and in space if $u(x, y, t) \in C^4(\Omega)$, where Ω is the bounded domain where the PDE is defined. It is unconditionally stable for linear problems. We can use symbolic expressions to discuss the method.

The key idea of the ADI method is to use the implicit discretization dimension by dimension by taking advantage of fast tridiagonal solvers.

5. For the initial conditions shown as part of the displayed banner in the cell below, construct an ADI solution and plot.

In [ ]:

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                              %
%    Example of ADI Method for 2D heat equation                                %
%                                                                              %
%          u_t = u_{xx} + u_{yy} + f(x,t)                                      %
%                                                                              %
%    Test problme:                                                             %
%      Exact solution: u(t,x,y) = exp(-t) sin(pi*x) sin(pi*y)                  %
%      Source term:  f(t,x,y) = exp(-t) sin(pi*x) sin(pi*y) (2pi^2-1)          %
%                                                                              %
%    Files needed for the test:                                                %
%                                                                              %
%     adi.m:      This file, the main calling code.                            %
%     f.m:        The file defines the f(t,x,y)                                %
%     uexact.m:    The exact solution.                                         %
%                                                                              %
%     Results:         n             e            ratio                        %
%                     10           0.0041                                      %
%     t_final=0.5     20           0.0010           4.1                        %
%                     40           2.5192e-04       3.97                       %
%                     80           6.3069e-05       3.9944                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   clear; close all;

   a = 0; b=1;  c=0; d=1; n = 40;  tfinal = 0.5;

   m = n;
   h = (b-a)/n;        dt=h;
   h1 = h*h;
   x=a:h:b; y=c:h:d;

%-- Initial condition:

   t = 0;
   for i=1:m+1,
      for j=1:m+1,
         u1(i,j) = uexact(t,x(i),y(j));
      end
   end

%---------- Big loop for time t --------------------------------------

k_t = fix(tfinal/dt);

for k=1:k_t

```

```
t1 = t + dt; t2 = t + dt/2;

%--- sweep in x-direction --------------------------------------

for i=1:m+1,                                % Boundary condition.
  u2(i,1) = uexact(t2,x(i),y(1));
  u2(i,n+1) = uexact(t2,x(i),y(n+1));
  u2(1,i) = uexact(t2,x(1),y(i));
  u2(m+1,i) = uexact(t2,x(m+1),y(i));
end

for j = 2:n,                               % Look for fixed y(j)

   A = sparse(m-1,m-1); b=zeros(m-1,1);
   for i=2:m,
      b(i-1) = (u1(i,j-1) -2*u1(i,j) + u1(i,j+1))/h1 + ...
        f(t2,x(i),y(j)) + 2*u1(i,j)/dt;
      if i == 2
        b(i-1) = b(i-1) + uexact(t2,x(i-1),y(j))/h1;
        A(i-1,i) = -1/h1;
      else
    if i==m
          b(i-1) = b(i-1) + uexact(t2,x(i+1),y(j))/h1;
          A(i-1,i-2) =  -1/h1;
    else
       A(i-1,i) = -1/h1;
       A(i-1,i-2) = -1/h1;
        end
      end

      A(i-1,i-1) = 2/dt + 2/h1;
    end

     ut = A\b;                            % Solve the diagonal matrix.
     for i=1:m-1,
    u2(i+1,j) = ut(i);
     end

 end                                      % Finish x-sweep.

%------------- loop in y -direction --------------------------------

for i=1:m+1,                                  % Boundary condition
  u1(i,1) = uexact(t1,x(i),y(1));
  u1(i,n+1) = uexact(t1,x(i),y(m+1));
  u1(1,i) = uexact(t1,x(1),y(i));
  u1(m+1,i) = uexact(t1,x(m+1),y(i));
end

for i = 2:m,

   A = sparse(m-1,m-1); b=zeros(m-1,1);
   for j=2:n,
      b(j-1) = (u2(i-1,j) -2*u2(i,j) + u2(i+1,j))/h1 + ...
                f(t2,x(i),y(j)) + 2*u2(i,j)/dt;
      if j == 2
        b(j-1) = b(j-1) + uexact(t1,x(i),y(j-1))/h1;
        A(j-1,j) = -1/h1;
      else
        if j==n
          b(j-1) = b(j-1) + uexact(t1,x(i),y(j+1))/h1;
          A(j-1,j-2) =  -1/h1;
        else
           A(j-1,j) = -1/h1;
           A(j-1,j-2) = -1/h1;
        end
      end

      A(j-1,j-1) = 2/dt + 2/h1;              % Solve the system
    end

     ut = A\b;
     for j=1:n-1,
        u1(i,j+1) = ut(j);
     end

 end                               % Finish y-sweep.

 t = t + dt;

```

```
%--- finish ADI method at this time level, go to the next time level.

end       %-- Finished with the loop in time

%----------- Data analysis ----------------------------------

  for i=1:m+1,
    for j=1:n+1,
       ue(i,j) = uexact(tfinal,x(i),y(j));
    end
  end

  e = max(max(abs(u1-ue)))        % The infinity error.

  mesh(u1);                       % Plot the computed solution.
```