```
function fe2dx_p
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Discussion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'fe2dx_p.m'   finite element Matlab code for Scheme 1 applied
% to the predator-prey system with Kinetics 1 solved over the square.
% The geometry and grid are created here so no external files need to
% be imported.
%
% Boundary conditions:
%   Gamma: Periodic
%
% (C) 2014 Marcus R. Garvie. See 'mycopyright.txt' for details.
%
% Modified April 7, 2014
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Enter model parameters
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
alpha = input('Enter parameter alpha   ');
beta = input('Enter parameter beta   ');
gamma = input('Enter parameter gamma   ');
delta = input('Enter parameter delta   ');
a = input('Enter a in [a,b]^2   ');
b = input('Enter b in  [a,b]^2  ');
h = input('Enter space-step h   ');
T = input('Enter maximum time T   ');
delt = input('Enter time-step Delta t   ');
% Calculate and assign some constants
mu=delt/(h^2);
J=round((b-a)/h);
dimJ=J+1;
n = (dimJ)^2;   % no. of nodes (d.f.) for each dependent variable
N=round(T/delt);
% Create grid
indexI=1:dimJ;
x=(indexI-1)*h+a;
[X,Y]=meshgrid(x,x);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Enter initial data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
u0_str = input('Enter initial data function u0(x,y)   ','s');
u0_anon = @(x,y)eval(u0_str);   % create anonymous function
U0 = arrayfun(u0_anon,X,Y);
v0_str = input('Enter initial data function v0(x,y)   ','s');
v0_anon = @(x,y)eval(v0_str);   % create anonymous function
V0 = arrayfun(v0_anon,X,Y);
% Change orientation of initial data & convert to 1-D vector
U0=U0'; V0=V0'; u=U0(:); v=V0(:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 Assembly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
L=sparse(n,n);
L(1,1)=3; L(1,2)=-3/2; L(J+1,J+1)=6; L(J+1,J)=-3;
L=L+sparse(2:J,3:J+1,-1,n,n);
```

```
L=L+sparse(2:J,2:J,4,n,n);
L=L+sparse(2:J,1:J-1,-1,n,n);
L(1,J+2)=-3/2; L(J+1,2*J+2)=-3;
L=L+sparse(2:J,J+3:2*J+1,-2,n,n);
L(n-J,n-J)=6; L(n-J,n-J+1)=-3;
L(n,n)=3; L(n,n-1)=-3/2;
L=L+sparse(n-J+1:n-1,n-J+2:n,-1,n,n);
L=L+sparse(n-J+1:n-1,n-J+1:n-1,4,n,n);
L=L+sparse(n-J+1:n-1,n-J:n-2,-1,n,n);
L(n-J,n-(2*J+1))=-3; L(n,n-dimJ)=-3/2;
L=L+sparse(n-J+1:n-1,n-2*J:n-(J+2),-2,n,n);
L=L+sparse(J+2:n-dimJ,2*J+3:n,-1,n,n);
L=L+sparse(J+2:n-dimJ,1:n-2*dimJ,-1,n,n);
L=L+sparse(J+2:n-dimJ,J+2:n-dimJ,4,n,n);
L=L+sparse(J+2:n-(J+2),J+3:n-dimJ,-1,n,n);
L=L+sparse(J+2:dimJ:n-(2*J+1),J+3:dimJ:n-2*J,-1,n,n);
L=L+sparse(2*J+2:dimJ:n-2*dimJ,2*J+3:dimJ:n-(2*J+1),1,n,n);
L=L+sparse(J+3:n-dimJ,J+2:n-(J+2),-1,n,n);
L=L+sparse(2*J+2:dimJ:n-dimJ,2*J+1:dimJ:n-(J+2),-1,n,n);
L=L+sparse(2*J+3:dimJ:n-(2*J+1),2*J+2:dimJ:n-2*dimJ,1,n,n);
% Construct fixed parts of matrices A_{n-1} and C_{n-1}
L=mu*L;
A0=L+sparse(1:n,1:n,1-delt,n,n);
C0=delta*L+sparse(1:n,1:n,1+delt*gamma,n,n);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Time-stepping procedure
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for nt=1:N
    % Update coefficient matrices of linear system
    diag = abs(u);
    diag_entries = u./(alpha + abs(u));
    A = A0 + delt*sparse(1:n,1:n,diag,n,n);
    B = delt*sparse(1:n,1:n,diag_entries,n,n);
    C = C0 - delt*beta*sparse(1:n,1:n,diag_entries,n,n);
    % Impose periodic boundary conditions
    for s = 1:dimJ
        k1 = s*dimJ;
        k2 = (s-1)*dimJ+1;
        k3 = s;
        k4 = s+J*dimJ;
        C(k1,:)=0;
        C(k3,:)=0;
        C(k1,k1)=1;
        C(k3,k3)=1;
         v(k1) = v(k2);
         v(k3) = v(k4);
         A(k1,:)=0;
         A(k3,:)=0;
         A(k1,k1)=1;
         A(k3,k3)=1;
         B(k1,:)=0;
         B(k3,:)=0;
         u(k1) = u(k2);
         u(k3) = u(k4);
     end
    % Do the incomplete LU factorisation of C and A
    [LC,UC] = ilu(C,struct('type','ilutp','droptol',1e-5));
```

```
    [LA,UA] = ilu(A,struct('type','ilutp','droptol',1e-5));
    % Solve for v using GMRES
    [v,flagv,relresv,iterv]=gmres(C,v,4,1e-6,[],LC,UC,v);
    if flagv~=0 flagv,relresv,iterv,error('GMRES did not converge'),end
    r=u - B*v;
    % Solve for u using GMRES
    [u,flagu,relresu,iteru]=gmres(A,r,4,1e-6,[],LA,UA,u);
    if flagu~=0 flagu,relresu,iteru,error('GMRES did not converge'),end
end
% Re-order 1-D solution vectors into 2-D solution grids
V_grid=reshape(v,dimJ,dimJ); U_grid=reshape(u,dimJ,dimJ);
% Put solution grids into ij (matrix) orientation
V_grid=V_grid'; U_grid=U_grid';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                Plot solutions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure;pcolor(X,Y,U_grid);shading interp;colorbar;axis square xy;title('u')
figure;pcolor(X,Y,V_grid);shading interp;colorbar;axis square xy;title('v')
```

Published with MATLAB® R2013b