```
function fe2dx_n
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Discussion
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 'fe2dx_n.m'   2D finite element Matlab code for Scheme 1 applied
% to the predator-prey system with Kinetics 1. The nodes and elements
% of the unstructured grid are loaded from external files 't_triang.dat'
% and 'p_coord.dat'.
%
% Boundary conditions:
%   Gamma: Neumann
%
% (C) 2009 Marcus R. Garvie. See 'mycopyright.txt' for details.
%
% Modified April 7, 2014
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                      Enter data for mesh geometry
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in 'p(2,n)', the 'n' coordinates of the nodes
load p_coord.dat -ascii
p = (p_coord)';
% Read in 't(3,no_elems)', the list of nodes for 'no_elems' elements
load t_triang.dat -ascii
t = (round(t_triang))';
% Construct the connectivity for the nodes on Gamma
edges = boundedges (p',t');
% Number of edges on Gamma
[e,junk] = size(edges);
% Degrees of freedom per variable (n)
[junk,n]=size(p);
% Number of elements (no_elems)
[junk,no_elems]=size(t);
% Extract vector of 'x' and 'y' values
x = p(1,:); y = p(2,:);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     Enter data for model
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User inputs of parameters
alpha = input('Enter parameter alpha    ');
beta = input('Enter parameter beta    ');
gamma = input('Enter parameter gamma    ');
delta = input('Enter parameter delta    ');
T = input('Enter maximum time T    ');
delt = input('Enter time-step Delta t    ');
% User inputs of initial data
u0_str = input('Enter initial data function u0(x,y)    ','s');
u0_anon = @(x,y)eval(u0_str);   % create anonymous function
u = 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
v = arrayfun(v0_anon,x,y)';
% Enter the boundary conditions
gu_str = input('Enter the Neumann b.c. gu(x,y,t) for u  ','s');
gu = @(x,y,t)eval(gu_str);   % create anonymous function
```

```
gv_str = input('Enter the Neumann b.c. gv(x,y,t) for v  ','s');
gv = @(x,y,t)eval(gv_str);   % create anonymous function
% Calculate and assign some constants
N=round(T/delt);
% Degrees of freedom per variable (n)
[junk,n]=size(p);
% Number of elements (no_elems)
[junk,no_elems]=size(t);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Assembly
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m_hat=zeros(n,1);
K=sparse(n,n);
for elem = 1:no_elems
    % Identify nodes ni, nj and nk in element 'elem'
    ni = t(1,elem);
    nj = t(2,elem);
    nk = t(3,elem);
    % Identify coordinates of nodes ni, nj and nk
    xi = p(1,ni);
    xj = p(1,nj);
    xk = p(1,nk);
    yi = p(2,ni);
    yj = p(2,nj);
    yk = p(2,nk);
    % Calculate the area of element 'elem'
    triangle_area = abs(xj*yk-xk*yj-xi*yk+xk*yi+xi*yj-xj*yi)/2;
    % Calculate some quantities needed to construct elements in K
    h1 = (xi-xj)*(yk-yj)-(xk-xj)*(yi-yj);
    h2 = (xj-xk)*(yi-yk)-(xi-xk)*(yj-yk);
    h3 = (xk-xi)*(yj-yi)-(xj-xi)*(yk-yi);
    s1 = (yj-yi)*(yk-yj)+(xi-xj)*(xj-xk);
    s2 = (yj-yi)*(yi-yk)+(xi-xj)*(xk-xi);
    s3 = (yk-yj)*(yi-yk)+(xj-xk)*(xk-xi);
    t1 = (yj-yi)^2+(xi-xj)^2;  % g* changed to t*
    t2 = (yk-yj)^2+(xj-xk)^2;
    t3 = (yi-yk)^2+(xk-xi)^2;
    % Calculate local contributions to m_hat
    m_hat_i = triangle_area/3;
    m_hat_j = m_hat_i;
    m_hat_k = m_hat_i;
    % calculate local contributions to K
    K_ki = triangle_area*s1/(h3*h1);
    K_ik = K_ki;
    K_kj = triangle_area*s2/(h3*h2);
    K_jk = K_kj;
    K_kk = triangle_area*t1/(h3^2);
    K_ij = triangle_area*s3/(h1*h2);
    K_ji = K_ij;
    K_ii = triangle_area*t2/(h1^2);
    K_jj = triangle_area*t3/(h2^2);
    % Add contributions to vector m_hat
    m_hat(nk)=m_hat(nk)+m_hat_k;
    m_hat(nj)=m_hat(nj)+m_hat_j;
    m_hat(ni)=m_hat(ni)+m_hat_i;
    % Add contributions to K
    K=K+sparse(nk,ni,K_ki,n,n);
```

```matlab
    K=K+sparse(ni,nk,K_ik,n,n);
    K=K+sparse(nk,nj,K_kj,n,n);
    K=K+sparse(nj,nk,K_jk,n,n);
    K=K+sparse(nk,nk,K_kk,n,n);
    K=K+sparse(ni,nj,K_ij,n,n);
    K=K+sparse(nj,ni,K_ji,n,n);
    K=K+sparse(ni,ni,K_ii,n,n);
    K=K+sparse(nj,nj,K_jj,n,n);
end
% Construct matrix L
ivec=1:n;
IM_hat=sparse(ivec,ivec,1./m_hat,n,n);
L=delt*IM_hat*K;
% Construct fixed parts of matrices A_{n-1} and C_{n-1}
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
    tn = nt*delt;
    % Initialize right-hand-side functions
    rhs_u = u;
    rhs_v = v;
    % 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);
    % 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));
    % Impose Neumann boundary condition on Gamma
    for i = 1:e
        node1 = edges(i,1);
        node2 = edges(i,2);
        x1 = p(1,node1);
        y1 = p(2,node1);
        x2 = p(1,node2);
        y2 = p(2,node2);
        im_hat1 = 1/m_hat(node1);
        im_hat2 = 1/m_hat(node2);
        gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
        rhs_u(node1) = rhs_u(node1) + delt*gu(x1,y1,tn)*im_hat1*gamma12/2;
        rhs_u(node2) = rhs_u(node2) + delt*gu(x2,y2,tn)*im_hat2*gamma12/2;
        rhs_v(node1) = rhs_v(node1) + delt*gv(x1,y1,tn)*im_hat1*gamma12/2;
        rhs_v(node2) = rhs_v(node2) + delt*gv(x2,y2,tn)*im_hat2*gamma12/2;
    end
    % Solve for v using GMRES
    [v,flagv,relresv,iterv]=gmres(C,rhs_v,[],1e-6,[],LC,UC,v);
    if flagv~=0 flagv,relresv,iterv,error('GMRES did not converge'),end
    r=rhs_u - B*v;
    % Solve for u using GMRES
    [u,flagu,relresu,iteru]=gmres(A,r,[],1e-6,[],LA,UA,u);
    if flagu~=0 flagu,relresu,iteru,error('GMRES did not converge'),end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Plot solutions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot solution for u
figure;
set(gcf,'Renderer','zbuffer');
trisurf(t',x,y,u,'FaceColor','interp','EdgeColor','interp');
colorbar;axis off;title('u');
view ( 2 );
axis equal on tight;
% Plot solution for v
figure;
set(gcf,'Renderer','zbuffer');
trisurf(t',x,y,v,'FaceColor','interp','EdgeColor','interp');
colorbar;axis off;title('v');
view ( 2 );
axis equal on tight;
```

Published with MATLAB® R2013b