```
function fe2dx_p_fast ( alpha, beta, gamma, delta, a, b, h, T, delt, u0f, v0f )
%*****************************************************************************80
%
%% FE2DX_P_FAST applies Scheme 1 with Kinetics 1 to predator prey in the square.
%
%  Discussion:
%
%    FE2DX_P_FAST is a "fast" version of FE2DX_P.
%
%    FE2DX_P is a 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 within this function, so no external
%    files need to be imported.
%
%    Periodic boundary conditions are applied.
%
%    This function has 11 input parameters.  All, some, or none of them may
%    be supplied as command line arguments or as functional parameters.
%    Parameters not supplied through the argument list will be prompted for.
%
%    The parameters ALPHA, BETA, GAMMA and DELTA appear in the predator-prey
%    equations as follows:
%
%      dUdT =         nabla U +      U*V/(U+ALPHA) + U*(1-U)
%      dVdT = delta * nabla V + BETA*U*V/(U+ALPHA) - GAMMA * V
%
%  Licensing:
%
%    Copyright (C) 2014 Marcus R. Garvie.
%    See 'mycopyright.txt' for details.
%
%  Modified:
%
%    29 April 2014
%
%  Author:
%
%    Marcus R. Garvie and John Burkardt.
%
%  Reference:
%
%    Marcus R Garvie, John Burkardt, Jeff Morgan,
%    Simple Finite Element Methods for Approximating Predator-Prey Dynamics
%    in Two Dimensions using MATLAB,
%    Submitted to Bulletin of Mathematical Biology, 2014.
%
%  Parameters:
%
%    Input, real ALPHA, a parameter in the predator prey equations.
%    0 < ALPHA.
%
%    Input, real BETA, a parameter in the predator prey equations.
%    0 < BETA.
%
%    Input, real GAMMA, a parameter in the predator prey equations.
```

```
%    0 < GAMMA.
%
%    Input, real DELTA, a parameter in the predator prey equations.
%    0 < DELTA.
%
%    Input, real A, B, the endpoints of the spatial interval.
%    The spatial region is a square [A,B]x[A,B].  A < B.
%
%    Input, real H, the spatial step size used to discretize [A,B].
%    0 < H.
%
%    Input, real T, the maximum time.
%    0 < T.
%
%    Input, real DELT, the time step to use in integrating from 0 to T.
%    0 < DELT.
%
%    Input, string U0F or function pointer @U0F, a function for the initial
%    condition of U(X,Y).
%
%    Input, string V0F or function pointer @V0F, a function for the initial
%    condition of V(X,Y).
%
%*****************************************************************************80
%  Enter model parameters.
%*****************************************************************************80
  if ( nargin < 1 )
    alpha = input ( 'Enter parameter alpha:  ' );
  elseif ( ischar ( alpha ) )
    alpha = str2num ( alpha );
  end
  if ( nargin < 2 )
    beta = input ( 'Enter parameter beta:  ' );
  elseif ( ischar ( beta ) )
    beta = str2num ( beta );
  end
  if ( nargin < 3 )
    gamma = input ( 'Enter parameter gamma:  ' );
  elseif ( ischar ( gamma ) )
    gamma = str2num ( gamma );
  end
  if ( nargin < 4 )
    delta = input ( 'Enter parameter delta:  ' );
  elseif ( ischar ( delta ) )
    delta = str2num ( delta );
  end
  if ( nargin < 5 )
    a = input ( 'Enter a in [a,b]^2:  ' );
  elseif ( ischar ( a ) )
    a = str2num ( a );
  end
  if ( nargin < 6 )
    b = input ( 'Enter b in [a,b]^2:  ' );
  elseif ( ischar ( b ) )
    b = str2num ( b );
  end
  if ( nargin < 7 )
```

```
    h = input ( 'Enter space-step h:  ' );
  elseif ( ischar ( h ) )
    h = str2num ( h );
  end
  if ( nargin < 8 )
    T = input ( 'Enter maximum time T:  ' );
  elseif ( ischar ( T ) )
    T = str2num ( T );
  end
  if ( nargin < 9 )
    delt = input ( 'Enter time-step delt:  ' );
  elseif ( ischar ( delt ) )
    delt = str2num ( delt );
  end
  fprintf ( 1, '  Using ALPHA = %g\n', alpha );
  fprintf ( 1, '  Using BETA = %g\n', beta );
  fprintf ( 1, '  Using GAMMA = %g\n', gamma );
  fprintf ( 1, '  Using DELTA = %g\n', delta );
  fprintf ( 1, '  Using A = %g\n', a );
  fprintf ( 1, '  Using B = %g\n', b );
  fprintf ( 1, '  Using H = %g\n', h );
  fprintf ( 1, '  Using T = %g\n', T );
  fprintf ( 1, '  Using DELT = %g\n', delt );
%
%  Calculate and assign some constants.
%
  mu = delt / ( h ^ 2 );
  J = round ( ( b - a ) / h );
  dimJ = J + 1;
%
%  Compute number of nodes for each dependent variable.
%
  n = dimJ ^ 2;
%
%  N = number of time steps.
%
  N = round ( T / delt );
  fprintf ( 1, '\n' );
  fprintf ( 1, '  1D grid size is %d\n', dimJ );
  fprintf ( 1, '  2D grid size is %d\n', n );
  fprintf ( 1, '  Using N = %d time steps\n', N );
%
%  Create the spatial grid.
%
  indexI = 1 : dimJ;
  x = a + ( indexI - 1 ) * h;
  [ X, Y ] = meshgrid ( x, x );
%
%  Initial condition.
%
  if ( nargin < 10 )
    u0_str = input ( 'Enter initial data function u0(x,y):  ', 's' );
    u0f = @(x,y) eval ( u0_str );
  elseif ( ischar ( u0f ) )
    u0_str = u0f;
    u0f = @(x,y) eval ( u0_str );
  end
```

```
  U0 = ( arrayfun ( u0f, X, Y ) )';
  if ( nargin < 11 )
    v0_str = input ( 'Enter initial data function v0(x,y):  ', 's' );
    v0f = @(x,y) eval ( v0_str );
  elseif ( ischar ( v0f ) )
    v0_str = v0f;
    v0f = @(x,y) eval ( v0_str );
  end
  V0 = ( arrayfun ( v0f, X, Y ) )';
%
%  Convert to 1-D vector.
%
%    11 21 becomes 11
%    12 22         12
%                  21
%                  22
%
  u = U0(:);
  v = V0(:);
%*****************************************************************************80
%  Assembly.
%*****************************************************************************80
  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);
%
%  Set the coefficients of the linear equations that impose boundary conditions.
%
  for s = 1 : dimJ
    k1 = s*dimJ;
    k3 = s;
    A0(k1,:)=0;
    A0(k1,k1)=1;
    A0(k3,:)=0;
    A0(k3,k3)=1;
    C0(k1,:)=0;
    C0(k1,k1)=1;
    C0(k3,:)=0;
    C0(k3,k3)=1;
  end
  fprintf ( 1, '\n' );
  fprintf ( 1, '  Matrix size N = %d\n', n );
  fprintf ( 1, '  A0 nonzeros = %d\n', nnz ( A0 ) );
  fprintf ( 1, '  C0 nonzeros = %d\n', nnz ( C0 ) );
%*****************************************************************************80
%  Time-stepping.
%*****************************************************************************80
  for nt = 1 : N
%
%  Form the coefficient matrices A, B, and C.
%  Zero out entries in DIAG and DIAG_ENTRIES that would otherwise
%  upset equations associated with boundary conditions.
%
    diag = abs ( u );
    diag_entries = u ./ ( alpha + abs ( u ) );
    for s = 1 : dimJ
      k1 = s * dimJ;
      k3 = s;
      diag(k1) = 0.0;
      diag(k3) = 0.0;
      diag_entries(k1) = 0.0;
      diag_entries(k3) = 0.0;
    end
    A = A0 +        delt * sparse ( 1:n, 1:n, diag, n, n );
    B =             delt * sparse ( 1:n, 1:n, diag_entries, n, n );
    C = C0 - beta * delt * sparse ( 1:n, 1:n, diag_entries, n, n );
%
%  Set the right hand sides of equations that impose the boundary conditions.
%
    for s = 1 : dimJ
      k1 = s*dimJ;
      k2 = (s-1)*dimJ+1;
      k3 = s;
      k4 = s+J*dimJ;
      v(k1) = v(k2);
      v(k3) = v(k4);
      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
%*****************************************************************************80
%  Plot solutions.
%*****************************************************************************80
%
% 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';
 figure;
 pcolor(X,Y,U_grid);
 shading interp;
 colorbar;
 axis square xy;
 title('u')
 filename = 'fe2dx_p_fast_u.png';
 print ( '-dpng', filename );
 fprintf ( 1, '\n' );
 fprintf ( 1, '  U contours saved in "%s"\n', filename );
 figure;
 pcolor(X,Y,V_grid);
 shading interp;
 colorbar;
 axis square xy;
 title('v')
 filename = 'fe2dx_p_fast_v.png';
 print ( '-dpng', filename );
 fprintf ( 1, '  V contours saved in "%s"\n', filename );
```

```
  return
end
```

Published with MATLAB® R2013b