```matlab
1 function fe2d_nd_fast ( alpha, beta, gamma, delta, T, delt, u0f, v0f, ...
2   g1uf, g1vf, g2uf, g2vf )
3 %*********************************************************************80
4 %
5 %% FE2D_ND_FAST applies Scheme 2 with Kinetics 1 to predator prey in a region.
6 %
7 %  Discussion:
8 %
9 %    FE2D_ND_FAST is a "fast" version of FE2D_ND.
10 %
11 %    FE2D_ND is a finite element Matlab code for Scheme 2 applied
12 %    to the predator-prey system with Kinetics 1 solved over a region
13 %    which has been triangulated.  The geometry and grid are read from
14 %    user-supplied files 't_triang.dat' and 'p_coord.dat' respectively.
15 %    as are the list of nodes on which Dirichlet and Neumann boundary
16 %    conditions are to be imposed (from 'bn1_nodes.dat' and 'bn2_nodes.dat'
17 %    respectively).
18 %
19 %    This function has 12 input parameters.  All, some, or none of them may
20 %    be supplied as command line arguments or as functional parameters.
21 %    Parameters not supplied through the argument list will be prompted for.
22 %
23 %    The parameters ALPHA, BETA, GAMMA and DELTA appear in the predator-prey
24 %    equations as follows:
25 %
26 %      dUdT =          nabla U +      U*V/(U+ALPHA) + U*(1-U)
27 %      dVdT = delta * nabla V + BETA*U*V/(U+ALPHA) - GAMMA * V
28 %
29 %  Licensing:
30 %
31 %    Copyright (C) 2014 Marcus R. Garvie.
32 %    See 'mycopyright.txt' for details.
33 %
34 %  Modified:
35 %
36 %    29 April 2014
37 %
38 %  Author:
39 %
40 %    Marcus R. Garvie and John Burkardt.
41 %
42 %  Reference:
43 %
44 %    Marcus R Garvie, John Burkardt, Jeff Morgan,
45 %    Simple Finite Element Methods for Approximating Predator-Prey Dynamics
46 %    in Two Dimensions using MATLAB,
47 %    Submitted to Bulletin of Mathematical Biology, 2014.
48 %
49 %  Parameters:
50 %
51 %    Input, real ALPHA, a parameter in the predator prey equations.
52 %    0 < ALPHA.
53 %
54 %    Input, real BETA, a parameter in the predator prey equations.
55 %    0 < BETA.
```

```
56 %
57 %    Input, real GAMMA, a parameter in the predator prey equations.
58 %    0 < GAMMA.
59 %
60 %    Input, real DELTA, a parameter in the predator prey equations.
61 %    0 < DELTA.
62 %
63 %    Input, real T, the maximum time.
64 %    0 < T.
65 %
66 %    Input, real DELT, the time step to use in integrating from 0 to T.
67 %    0 < DELT.
68 %
69 %    Input, string U0F or function pointer @U0F, a function for the initial
70 %    condition of U(X,Y).
71 %
72 %    Input, string V0F or function pointer @V0F, a function for the initial
73 %    condition of V(X,Y).
74 %
75 %    Input, string G1UF or function pointer @G1UF, a function for the Dirichlet
76 %    boundary condition of U(X,Y,T).
77 %
78 %    Input, string G1VF or function pointer @G1VF, a function for the Dirichlet
79 %    boundary condition of V(X,Y,T).
80 %
81 %    Input, string G2UF or function pointer @G2UF, a function for the Neumann
82 %    boundary condition of U(X,Y,T).
83 %
84 %    Input, string G2VF or function pointer @G2VF, a function for the Neumann
85 %    boundary condition of V(X,Y,T).
86 %
87 %****************************************************************************80
88 %  Enter data for mesh geometry.
89 %****************************************************************************80
90 %
91 %  Read in 'p(2,n)', the 'n' coordinates of the nodes.
92    load p_coord.dat -ascii
93    p = ( p_coord )';
94 %
95 %  Read in 't(3,no_elems)', the list of nodes for 'no_elems' elements,
96 %  and force the entries to be integers.
97 %
98    load t_triang.dat -ascii
99    t = ( round ( t_triang ) )';
100 %
101 %  Read in 'bn1(1,isn1)', the nodes on Gamma1.
102 %
103    load bn1_nodes.dat -ascii
104    bn1 = ( round( bn1_nodes ) )';
105 %
106 %  Read in 'bn2(1,isn2)', the nodes on Gamma2.
107 %
108    load bn2_nodes.dat -ascii
109    bn2 = ( round ( bn2_nodes ) )';
110 %
111 %  Construct the connectivity for the nodes on Gamma2.
112 %
```

```matlab
113    cpp = subsetconnectivity ( t', p', bn2' );
114  %
115  %   E2 = number of edges on Gamma2.
116  %
117    [ e2, ~ ] = size ( cpp );
118  %
119  %   N = degrees of freedom per variable.
120  %
121    [ ~, n ] = size ( p );
122  %
123  %   NO_ELEMS = number of elements.
124  %
125    [ ~, no_elems ] = size ( t );
126  %
127  %   ISN1 = Number of nodes on boundary Gamma1.
128  %
129    [ ~, isn1 ] = size ( bn1 );
130  %
131  %   Extract vector of 'x' and 'y' values.
132  %
133    x = p(1,:);
134    y = p(2,:);
135  %*******************************************************************80
136  %   Enter data for model.
137  %*******************************************************************80
138    if ( nargin < 1 )
139      alpha = input ( 'Enter parameter alpha:  ' );
140    elseif ( ischar ( alpha ) )
141      alpha = str2num ( alpha );
142    end
143    if ( nargin < 2 )
144      beta = input ( 'Enter parameter beta:  ' );
145    elseif ( ischar ( beta ) )
146      beta = str2num ( beta );
147    end
148    if ( nargin < 3 )
149      gamma = input ( 'Enter parameter gamma:  ' );
150    elseif ( ischar ( gamma ) )
151      gamma = str2num ( gamma );
152    end
153    if ( nargin < 4 )
154      delta = input ( 'Enter parameter delta:  ' );
155    elseif ( ischar ( delta ) )
156      delta = str2num ( delta );
157    end
158    if ( nargin < 5 )
159      T = input ( 'Enter maximum time T:  ' );
160    elseif ( ischar ( T ) )
161      T = str2num ( T );
162    end
163    if ( nargin < 6 )
164      delt = input ( 'Enter time-step delt:  ' );
165    elseif ( ischar ( delt ) )
166      delt = str2num ( delt );
167    end
168    fprintf ( 1, '  Using ALPHA = %g\n', alpha );
169    fprintf ( 1, '  Using BETA = %g\n', beta );
```

```matlab
170    fprintf ( 1, '  Using GAMMA = %g\n', gamma );
171    fprintf ( 1, '  Using DELTA = %g\n', delta );
172    fprintf ( 1, '  Using T = %g\n', T );
173    fprintf ( 1, '  Using DELT = %g\n', delt );
174 %
175 %  Initial conditions.
176 %
177    if ( nargin < 7 )
178      u0_str = input ( 'Enter initial data function u0(x,y):  ', 's' );
179      u0f = @(x,y) eval ( u0_str );
180    elseif ( ischar ( u0f ) )
181      u0_str = u0f;
182      u0f = @(x,y) eval ( u0_str );
183    end
184    u = ( arrayfun ( u0f, x, y ) )';
185    if ( nargin < 8 )
186      v0_str = input ( 'Enter initial data function v0(x,y):  ', 's' );
187      v0f = @(x,y) eval ( v0_str );
188    elseif ( ischar ( v0f ) )
189      v0_str = v0f;
190      v0f = @(x,y) eval ( v0_str );
191    end
192    v = ( arrayfun ( v0f, x, y ) )';
193 %
194 %  Boundary conditions.
195 %
196    if ( nargin < 9 )
197      g1u_str = input('Enter the Dirichlet b.c. g1u(x,y,t) for u  ','s');
198      g1uf = @(x,y,t)eval(g1u_str);
199    elseif ( ischar ( g1uf ) )
200      g1u_str = g1uf;
201      g1uf = @(x,y,t)eval(g1u_str);
202    end
203    if ( nargin < 10 )
204      g1v_str = input('Enter the Dirichlet b.c. g1v(x,y,t) for v  ','s');
205      g1vf = @(x,y,t)eval(g1v_str);
206    elseif ( ischar ( g1vf ) )
207      g1v_str = g1vf;
208      g1vf = @(x,y,t)eval(g1v_str);
209    end
210    if ( nargin < 11 )
211      g2u_str = input('Enter the Neumann b.c. g2u(x,y,t) for u  ','s');
212      g2uf = @(x,y,t)eval(g2u_str);
213    elseif ( ischar ( g2uf ) )
214      g2u_str = g2uf;
215      g2uf = @(x,y,t)eval(g2u_str);
216    end
217    if ( nargin < 12 )
218      g2v_str = input('Enter the Neumann b.c. g2v(x,y,t) for v  ','s');
219      g2vf = @(x,y,t)eval(g2v_str);
220    elseif ( ischar ( g2vf ) )
221      g2v_str = g2vf;
222      g2vf = @(x,y,t)eval(g2v_str);
223    end
224 %
225 %  N = number of time steps.
226 %
```

```matlab
227    N = round ( T / delt );
228    fprintf ( 1, '  Taking N = %d time steps\n', N );
229 %*********************************************************************80
230 %  Assembly.
231 %*********************************************************************80
232    m_hat = zeros ( n, 1 );
233    K = sparse ( n, n );
234    for elem = 1 : no_elems
235 %
236 %  Identify nodes ni, nj and nk in element 'elem'.
237 %
238      ni = t(1,elem);
239      nj = t(2,elem);
240      nk = t(3,elem);
241 %
242 %  Identify coordinates of nodes ni, nj and nk.
243 %
244      xi = p(1,ni);
245      xj = p(1,nj);
246      xk = p(1,nk);
247      yi = p(2,ni);
248      yj = p(2,nj);
249      yk = p(2,nk);
250 %
251 %  Calculate the area of element 'elem'.
252 %
253      triangle_area = abs(xj*yk-xk*yj-xi*yk+xk*yi+xi*yj-xj*yi)/2;
254 %
255 %  Calculate some quantities needed to construct elements in K.
256 %
257      h1 = (xi-xj)*(yk-yj)-(xk-xj)*(yi-yj);
258      h2 = (xj-xk)*(yi-yk)-(xi-xk)*(yj-yk);
259      h3 = (xk-xi)*(yj-yi)-(xj-xi)*(yk-yi);
260      s1 = (yj-yi)*(yk-yj)+(xi-xj)*(xj-xk);
261      s2 = (yj-yi)*(yi-yk)+(xi-xj)*(xk-xi);
262      s3 = (yk-yj)*(yi-yk)+(xj-xk)*(xk-xi);
263      t1 = (yj-yi)^2+(xi-xj)^2;
264      t2 = (yk-yj)^2+(xj-xk)^2;
265      t3 = (yi-yk)^2+(xk-xi)^2;
266 %
267 %  Calculate local contributions to m_hat.
268 %
269      m_hat_i = triangle_area/3;
270      m_hat_j = m_hat_i;
271      m_hat_k = m_hat_i;
272 %
273 %  Calculate local contributions to K.
274 %
275      K_ki = triangle_area*s1/(h3*h1);
276      K_ik = K_ki;
277      K_kj = triangle_area*s2/(h3*h2);
278      K_jk = K_kj;
279      K_kk = triangle_area*t1/(h3^2);
280      K_ij = triangle_area*s3/(h1*h2);
281      K_ji = K_ij;
282      K_ii = triangle_area*t2/(h1^2);
283      K_jj = triangle_area*t3/(h2^2);
```

```matlab
284 %
285 %  Add contributions to vector m_hat.
286 %
287     m_hat(nk)=m_hat(nk)+m_hat_k;
288     m_hat(nj)=m_hat(nj)+m_hat_j;
289     m_hat(ni)=m_hat(ni)+m_hat_i;
290 %
291 %  Add contributions to K.
292 %
293     K=K+sparse(nk,ni,K_ki,n,n);
294     K=K+sparse(ni,nk,K_ik,n,n);
295     K=K+sparse(nk,nj,K_kj,n,n);
296     K=K+sparse(nj,nk,K_jk,n,n);
297     K=K+sparse(nk,nk,K_kk,n,n);
298     K=K+sparse(ni,nj,K_ij,n,n);
299     K=K+sparse(nj,ni,K_ji,n,n);
300     K=K+sparse(ni,ni,K_ii,n,n);
301     K=K+sparse(nj,nj,K_jj,n,n);
302   end
303 %
304 %  Construct matrix L.
305 %
306   ivec = 1 : n;
307   IM_hat = sparse(ivec,ivec,1./m_hat,n,n);
308   L = delt * IM_hat * K;
309 %
310 %  Construct matrices B1 and B2.
311 %
312   B1 = sparse(1:n,1:n,1,n,n) + L;
313   B2 = sparse(1:n,1:n,1,n,n) + delta * L;
314 %
315 %  Modify B1 and B2 as part of the imposition of Dirichlet boundary conditions.
316 %
317     for i = 1 : isn1
318       node = bn1(i);
319       B1(node,:)=0;
320       B1(node,node)=1;
321       B2(node,:)=0;
322       B2(node,node)=1;
323     end
324 %
325 %  Do the LU factorization of B1 and B2.
326 %
327     [ LB1, UB1 ] = ilu ( B1, struct('type','ilutp','droptol',1e-5) );
328     [ LB2, UB2 ] = ilu ( B2, struct('type','ilutp','droptol',1e-5) );
329 %****************************************************************************80
330 %  Time-stepping.
331 %****************************************************************************80
332   for nt = 1 : N
333     tn = nt * delt;
334 %
335 %  Evaluate modified functional response.
336 %
337     hhat = u ./ ( alpha + abs ( u ) );
338 %
339 %  Update right-hand-side of linear system.
340 %
```

```matlab
341      F = u - u .* abs ( u ) - v .* hhat;
342      G = beta * v .* hhat - gamma * v;
343      rhs_u = u + delt * F;
344      rhs_v = v + delt * G;
345 %
346 %   Impose Neumann boundary condition on Gamma2.
347 %
348      for i = 1 : e2
349        node1 = cpp(i,1);
350        node2 = cpp(i,2);
351        x1 = p(1,node1);
352        y1 = p(2,node1);
353        x2 = p(1,node2);
354        y2 = p(2,node2);
355        im_hat1 = 1/m_hat(node1);
356        im_hat2 = 1/m_hat(node2);
357        gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
358        rhs_u(node1) = rhs_u(node1) + delt * g2uf (x1,y1,tn) * im_hat1*gamma12/2;
359        rhs_u(node2) = rhs_u(node2) + delt * g2uf (x2,y2,tn) * im_hat2*gamma12/2;
360        rhs_v(node1) = rhs_v(node1) + delt * g2vf (x1,y1,tn) * im_hat1*gamma12/2;
361        rhs_v(node2) = rhs_v(node2) + delt * g2vf (x2,y2,tn) * im_hat2*gamma12/2;
362      end
363 %
364 %   Set right hand sides of Dirichlet boundary conditions.
365 %
366      for i = 1 : isn1
367        node = bn1(i);
368        xx = p(1,node);
369        yy = p(2,node);
370        rhs_u(node) = g1uf ( xx, yy, tn );
371        rhs_v(node) = g1vf ( xx, yy, tn );
372      end
373 %
374 %   Solve for u and v using GMRES.
375 %
376      [u,flagu,relresu,iteru] = gmres ( B1,rhs_u,[],1e-6,[],LB1,UB1,u );
377      if flagu ~= 0
378        flagu
379        relresu
380        iteru
381        error('GMRES did not converge')
382      end
383      [v,flagv,relresv,iterv] = gmres ( B2,rhs_v,[],1e-6,[],LB2,UB2,v );
384      if flagv ~= 0
385        flagv
386        relresv
387        iterv
388        error('GMRES did not converge')
389      end
390
391    end
392 %*******************************************************************80
393 %   Plot the solutions.
394 %*******************************************************************80
395 %
396 %   Plot U;
397 %
```

```matlab
398    figure;
399    set(gcf,'Renderer','zbuffer');
400    trisurf(t',x,y,u,'FaceColor','interp','EdgeColor','interp');
401    colorbar;
402    axis off;
403    title('u');
404    view ( 2 );
405    axis equal on tight;
406    filename = 'fe2d_nd_fast_u.png';
407    print ( '-dpng', filename );
408    fprintf ( 1, '  Saved graphics file "%s"\n', filename );
409 %
410 %   Plot V.
411 %
412    figure;
413    set(gcf,'Renderer','zbuffer');
414    trisurf(t',x,y,v,'FaceColor','interp','EdgeColor','interp');
415    colorbar;
416    axis off;
417    title('v');
418    view ( 2 );
419    axis equal on tight;
420    filename = 'fe2d_nd_fast_v.png';
421    fprintf ( 1, '  Saved graphics file "%s"\n', filename );
422    print ( '-dpng', filename );
423    return
424 end
```