

```

1 function fe2d_d_fast ( alpha, beta, gamma, delta, T, delt, u0f, v0f, guf, gvf )
2 %*****80
3 %
4 %% FE2D_D_FAST applies Scheme 2 with Kinetics 1 to predator prey in a region.
5 %
6 % Discussion:
7 %
8 % FE2D_D_FAST is a "fast" version of FE2D_D.
9 %
10 % FE2D_D is a finite element Matlab code for Scheme 2 applied
11 % to the predator-prey system with Kinetics 1 solved over a region
12 % which has been triangulated. The geometry and grid are read from
13 % user-supplied files 't_triang.dat' and 'p_coord.dat' respectively.
14 %
15 % Dirichlet boundary conditions are applied.
16 %
17 % This function has 10 input parameters. All, some, or none of them may
18 % be supplied as command line arguments or as functional parameters.
19 % Parameters not supplied through the argument list will be prompted for.
20 %
21 % The parameters ALPHA, BETA, GAMMA and DELTA appear in the predator-prey
22 % equations as follows:
23 %
24 %      dUdT =          nabla U +          U*V/(U+ALPHA) + U*(1-U)
25 %      dVdT = delta * nabla V + BETA*U*V/(U+ALPHA) - GAMMA * V
26 %
27 % Licensing:
28 %
29 % Copyright (C) 2014 Marcus R. Garvie.
30 % See 'mycopyright.txt' for details.
31 %
32 % Modified:
33 %
34 % 29 April 2014
35 %
36 % Author:
37 %
38 % Marcus R. Garvie.
39 %
40 % Reference:
41 %
42 % Marcus R Garvie, John Burkardt, Jeff Morgan,
43 % Simple Finite Element Methods for Approximating Predator-Prey Dynamics
44 % in Two Dimensions using MATLAB,
45 % Submitted to Bulletin of Mathematical Biology, 2014.
46 %
47 % Parameters:
48 %
49 % Input, real ALPHA, a parameter in the predator prey equations.
50 % 0 < ALPHA.
51 %
52 % Input, real BETA, a parameter in the predator prey equations.
53 % 0 < BETA.
54 %
55 % Input, real GAMMA, a parameter in the predator prey equations.

```

```

56 %      0 < GAMMA.
57 %
58 %      Input, real DELTA, a parameter in the predator prey equations.
59 %      0 < DELTA.
60 %
61 %      Input, real T, the maximum time.
62 %      0 < T.
63 %
64 %      Input, real DELT, the time step to use in integrating from 0 to T.
65 %      0 < DELT.
66 %
67 %      Input, string U0F or function pointer @U0F, a function for the initial
68 %      condition of U(X,Y).
69 %
70 %      Input, string V0F or function pointer @V0F, a function for the initial
71 %      condition of V(X,Y).
72 %
73 %      Input, string GUF or function pointer @GUF, a function for the Dirichlet
74 %      boundary condition of U(X,Y,T).
75 %
76 %      Input, string GVF or function pointer @GVF, a function for the Dirichlet
77 %      boundary condition of V(X,Y,T).
78 %
79 %*****80
80 % Enter data for mesh geometry.
81 %*****80
82 %
83 % Read in 'p(2,n)', the 'n' coordinates of the nodes.
84 load p_coord.dat -ascii
85 p = ( p_coord )';
86 %
87 % Read in 't(3,no_elems)', the list of nodes for 'no_elems' elements,
88 % and force the entries to be integers.
89 %
90 load t_triang.dat -ascii
91 t = ( round ( t_triang ) )';
92 %
93 % Construct the connectivity for the nodes on Gamma.
94 %
95 edges = boundedges ( p', t' );
96 %
97 % BN = boundary nodes on Gamma.
98 %
99 bn = unique ( edges(:) );
100 %
101 % ISN = number of boundary nodes.
102 %
103 [~,isn] = size ( bn );
104 %
105 % N = degrees of freedom per variable.
106 %
107 [~,n] = size ( p );
108 %
109 % NO_ELEMS = number of elements.
110 %
111 [ ~, no_elems ] = size ( t );
112 %

```

```

113 % Extract vector of 'x' and 'y' values.
114 %
115 x = p(1,:);
116 y = p(2,:);
117 %*****80
118 % Enter data for model.
119 %*****80
120 if ( nargin < 1 )
121     alpha = input ( 'Enter parameter alpha:  ' );
122 elseif ( ischar ( alpha ) )
123     alpha = str2num ( alpha );
124 end
125 if ( nargin < 2 )
126     beta = input ( 'Enter parameter beta:  ' );
127 elseif ( ischar ( beta ) )
128     beta = str2num ( beta );
129 end
130 if ( nargin < 3 )
131     gamma = input ( 'Enter parameter gamma:  ' );
132 elseif ( ischar ( gamma ) )
133     gamma = str2num ( gamma );
134 end
135 if ( nargin < 4 )
136     delta = input ( 'Enter parameter delta:  ' );
137 elseif ( ischar ( delta ) )
138     delta = str2num ( delta );
139 end
140 if ( nargin < 5 )
141     T = input ( 'Enter maximum time T:  ' );
142 elseif ( ischar ( T ) )
143     T = str2num ( T );
144 end
145 if ( nargin < 6 )
146     delt = input ( 'Enter time-step delt:  ' );
147 elseif ( ischar ( delt ) )
148     delt = str2num ( delt );
149 end
150 fprintf ( 1, ' Using ALPHA = %g\n', alpha );
151 fprintf ( 1, ' Using BETA = %g\n', beta );
152 fprintf ( 1, ' Using GAMMA = %g\n', gamma );
153 fprintf ( 1, ' Using DELTA = %g\n', delta );
154 fprintf ( 1, ' Using T = %g\n', T );
155 fprintf ( 1, ' Using DELT = %g\n', delt );
156 %
157 % Initial conditions.
158 %
159 if ( nargin < 7 )
160     u0_str = input ( 'Enter initial data function u0(x,y):  ', 's' );
161     u0f = @(x,y) eval ( u0_str );
162 elseif ( ischar ( u0f ) )
163     u0_str = u0f;
164     u0f = @(x,y) eval ( u0_str );
165 end
166 u = ( arrayfun ( u0f, x, y ) )';
167 if ( nargin < 8 )
168     v0_str = input ( 'Enter initial data function v0(x,y):  ', 's' );
169     v0f = @(x,y) eval ( v0_str );

```

```

170 elseif ( ischar ( v0f ) )
171     v0_str = v0f;
172     v0f = @(x,y) eval ( v0_str );
173 end
174 v = ( arrayfun ( v0f, x, y ) )';
175 %
176 % Boundary conditions.
177 %
178 if ( nargin < 9 )
179     gu_str = input('Enter the Dirichlet b.c. gu(x,y,t) for u ', 's');
180     guf = @(x,y,t)eval(gu_str);
181 elseif ( ischar ( guf ) )
182     gu_str = guf;
183     guf = @(x,y,t)eval(gu_str);
184 end
185 if ( nargin < 10 )
186     gv_str = input('Enter the Dirichlet b.c. gv(x,y,t) for v ', 's');
187     gv_f = @(x,y,t)eval(gv_str);
188 elseif ( ischar ( gv_f ) )
189     gv_str = gv_f;
190     gv_f = @(x,y,t)eval(gv_str);
191 end
192 %
193 % N = number of time steps.
194 %
195 N = round ( T / delt );
196 fprintf ( 1, ' Taking N = %d time steps\n', N );
197 %*****80
198 % Assembly.
199 %*****80
200 m_hat = zeros ( n, 1 );
201 K = sparse ( n, n );
202 for elem = 1 : no_elems
203 %
204 % Identify nodes ni, nj and nk in element 'elem'.
205 %
206     ni = t(1,elem);
207     nj = t(2,elem);
208     nk = t(3,elem);
209 %
210 % Identify coordinates of nodes ni, nj and nk.
211 %
212     xi = p(1,ni);
213     xj = p(1,nj);
214     xk = p(1,nk);
215     yi = p(2,ni);
216     yj = p(2,nj);
217     yk = p(2,nk);
218 %
219 % Calculate the area of element 'elem'.
220 %
221     triangle_area = abs(xj*yk-xk*yj-xi*yk+xk*yi+xi*yj-xj*yi)/2;
222 %
223 % Calculate some quantities needed to construct elements in K.
224 %
225     h1 = (xi-xj)*(yk-yj)-(xk-xj)*(yi-yj);
226     h2 = (xj-xk)*(yi-yk)-(xi-xk)*(yj-yk);

```

```

227     h3 = (xk-xi)*(yj-yi)-(xj-xi)*(yk-yi);
228     s1 = (yj-yi)*(yk-yj)+(xi-xj)*(xj-xk);
229     s2 = (yj-yi)*(yi-yk)+(xi-xj)*(xk-xi);
230     s3 = (yk-yj)*(yi-yk)+(xj-xk)*(xk-xi);
231     t1 = (yj-yi)^2+(xi-xj)^2;
232     t2 = (yk-yj)^2+(xj-xk)^2;
233     t3 = (yi-yk)^2+(xk-xi)^2;
234 %
235 % Calculate local contributions to m_hat.
236 %
237     m_hat_i = triangle_area/3;
238     m_hat_j = m_hat_i;
239     m_hat_k = m_hat_i;
240 %
241 % Calculate local contributions to K.
242 %
243     K_ki = triangle_area*s1/(h3*h1);
244     K_ik = K_ki;
245     K_kj = triangle_area*s2/(h3*h2);
246     K_jk = K_kj;
247     K_kk = triangle_area*t1/(h3^2);
248     K_ij = triangle_area*s3/(h1*h2);
249     K_ji = K_ij;
250     K_ii = triangle_area*t2/(h1^2);
251     K_jj = triangle_area*t3/(h2^2);
252 %
253 % Add contributions to vector m_hat.
254 %
255     m_hat(nk)=m_hat(nk)+m_hat_k;
256     m_hat(nj)=m_hat(nj)+m_hat_j;
257     m_hat(ni)=m_hat(ni)+m_hat_i;
258 %
259 % Add contributions to K.
260 %
261     K=K+sparse(nk,ni,K_ki,n,n);
262     K=K+sparse(ni,nk,K_ik,n,n);
263     K=K+sparse(nk,nj,K_kj,n,n);
264     K=K+sparse(nj,nk,K_jk,n,n);
265     K=K+sparse(nk,nk,K_kk,n,n);
266     K=K+sparse(ni,nj,K_ij,n,n);
267     K=K+sparse(nj,ni,K_ji,n,n);
268     K=K+sparse(ni,ni,K_ii,n,n);
269     K=K+sparse(nj,nj,K_jj,n,n);
270 end
271 %
272 % Construct matrix L.
273 %
274     ivec = 1 : n;
275     IM_hat = sparse(ivec,ivec,1./m_hat,n,n);
276     L = delt * IM_hat * K;
277 %
278 % Construct matrices B1 and B2.
279 %
280     B1 = sparse(1:n,1:n,1,n,n) + L;
281     B2 = sparse(1:n,1:n,1,n,n) + delta * L;
282 %
283 % Modify B1 and B2 to impose Dirichlet boundary conditions.

```

```

284 %
285 for i = 1 : isn
286     node = bn(i);
287     B1(node,:) = 0;
288     B1(node,node) = 1;
289     B2(node,node) = 1;
290 end
291 %
292 % Do the incomplete LU factorisation of B1 and B2.
293 %
294 [ LB1, UB1 ] = ilu ( B1, struct('type','ilutp','droptol',1e-5) );
295 [ LB2, UB2 ] = ilu ( B2, struct('type','ilutp','droptol',1e-5) );
296 %*****80
297 % Time-stepping.
298 %*****80
299 for nt = 1 : N
300     tn = nt * delt;
301 %
302 % Evaluate modified functional response.
303 %
304 hhat = u ./ ( alpha + abs ( u ) );
305 %
306 % Update right-hand-side of linear system.
307 %
308 F = u - u .* abs ( u ) - v .* hhat;
309 G = beta * v .* hhat - gamma * v;
310 rhs_u = u + delt * F;
311 rhs_v = v + delt * G;
312 %
313 % Modify right hand sides to impose Dirichlet boundary conditions on Gamma
314 %
315 for i = 1 : isn
316     node = bn(i);
317     xx = p(1,node);
318     yy = p(2,node);
319     rhs_u(node) = guf ( xx, yy, tn );
320     rhs_v(node) = gvfv ( xx, yy, tn );
321 end
322 %
323 % Solve for u and v using GMRES.
324 %
325 [ u, flagu, relresu, iteru ] = gmres ( B1, rhs_u, [], 1e-6, [], LB1, UB1, u );
326 if flagu ~= 0
327     flagu
328     relresu
329     iteru
330     error('GMRES did not converge')
331 end
332 [ v, flagv, relresv, iterv ] = gmres ( B2, rhs_v, [], 1e-6, [], LB2, UB2, v );
333 if flagv ~= 0
334     flagv
335     relresv
336     iterv
337     error('GMRES did not converge')
338 end
339 end
340 %*****80

```

```

341 % Plot the solutions.
342 %*****80
343 %
344 % Plot U;
345 %
346 figure;
347 set(gcf,'Renderer','zbuffer');
348 trisurf(t',x,y,u,'FaceColor','interp','EdgeColor','interp');
349 colorbar;
350 axis off;
351 title('u');
352 view ( 2 );
353 axis equal on tight;
354 filename = 'fe2d_d_fast_u.png';
355 print ( '-dpng', filename );
356 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
357 %
358 % Plot V.
359 %
360 figure;
361 set(gcf,'Renderer','zbuffer');
362 trisurf(t',x,y,v,'FaceColor','interp','EdgeColor','interp');
363 colorbar;
364 axis off;
365 title('v');
366 view ( 2 );
367 axis equal on tight;
368 filename = 'fe2d_d_fast_v.png';
369 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
370 print ( '-dpng', filename );
371 return
372 end

```