

```

1 function fe2d_nr_fast ( alpha, beta, gamma, delta, T, delt, u0f, v0f, ...
2   k1, k2, g2uf, g2vf )
3 %*****80
4 %
5 %% FE2D_NR_FAST applies Scheme 2 with Kinetics 1 to predator prey in a region.
6 %
7 % Discussion:
8 %
9 %   FE2D_NR_FAST is a "fast" version of FE2D_NR.
10 %
11 %   FE2D_NR 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 Robin 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, real K1, the coefficient for the Robin boundary condition
76 %     to be applied to U:  $dU/dn = k1 * U$ .
77 %
78 %     Input, real K2, the coefficient for the Robin boundary condition
79 %     to be applied to V:  $dV/dn = k2 * V$ .
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 %
93   load p_coord.dat -ascii
94   p = ( p_coord )';
95 %
96 %   Read in 't(3,no_elems)', the list of nodes for 'no_elems' elements.
97 %
98   load t_triangu.dat -ascii
99   t = ( round ( t_triangu ) )';
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 Gamma1.
112 %

```

```

113 cpp1 = subsetconnectivity ( t', p', bn1' );
114 %
115 % Construct the connectivity for the nodes on Gamma2.
116 %
117 cpp2 = subsetconnectivity ( t', p', bn2' );
118 %
119 % E1 = number of edges on Gamma1.
120 %
121 [ e1, ~ ] = size ( cpp1 );
122 %
123 % E2 = number of edges on Gamma2.
124 %
125 [ e2, ~ ] = size ( cpp2 );
126 %
127 % N = degrees of freedom per variable.
128 %
129 [ ~, n ] = size ( p );
130 %
131 % NO_ELEMS = number of elements.
132 %
133 [ ~, no_elems ] = size ( t );
134 %
135 % Extract vector of 'x' and 'y' values.
136 %
137 x = p(1,:);
138 y = p(2,:);
139 %*****80
140 % Enter data for model.
141 %*****80
142 if ( nargin < 1 )
143     alpha = input ( 'Enter parameter alpha: ' );
144 elseif ( ischar ( alpha ) )
145     alpha = str2num ( alpha );
146 end
147 if ( nargin < 2 )
148     beta = input ( 'Enter parameter beta: ' );
149 elseif ( ischar ( beta ) )
150     beta = str2num ( beta );
151 end
152 if ( nargin < 3 )
153     gamma = input ( 'Enter parameter gamma: ' );
154 elseif ( ischar ( gamma ) )
155     gamma = str2num ( gamma );
156 end
157 if ( nargin < 4 )
158     delta = input ( 'Enter parameter delta: ' );
159 elseif ( ischar ( delta ) )
160     delta = str2num ( delta );
161 end
162 if ( nargin < 5 )
163     T = input ( 'Enter maximum time T: ' );
164 elseif ( ischar ( T ) )
165     T = str2num ( T );
166 end
167 if ( nargin < 6 )
168     delT = input ( 'Enter time-step delT: ' );
169 elseif ( ischar ( delT ) )

```

```

170     delt = str2num ( delt );
171 end
172 fprintf ( 1, ' Using ALPHA = %g\n', alpha );
173 fprintf ( 1, ' Using BETA = %g\n', beta );
174 fprintf ( 1, ' Using GAMMA = %g\n', gamma );
175 fprintf ( 1, ' Using DELTA = %g\n', delta );
176 fprintf ( 1, ' Using T = %g\n', T );
177 fprintf ( 1, ' Using DELT = %g\n', delt );
178 %
179 % Initial conditions.
180 %
181 if ( nargin < 7 )
182     u0_str = input ( 'Enter initial data function u0(x,y): ', 's' );
183     u0f = @(x,y) eval ( u0_str );
184 elseif ( ischar ( u0f ) )
185     u0_str = u0f;
186     u0f = @(x,y) eval ( u0_str );
187 end
188 u = ( arrayfun ( u0f, x, y ) )';
189 if ( nargin < 8 )
190     v0_str = input ( 'Enter initial data function v0(x,y): ', 's' );
191     v0f = @(x,y) eval ( v0_str );
192 elseif ( ischar ( v0f ) )
193     v0_str = v0f;
194     v0f = @(x,y) eval ( v0_str );
195 end
196 v = ( arrayfun ( v0f, x, y ) )';
197 %
198 % Boundary conditions.
199 %
200 if ( nargin < 9 )
201     k1 = input('Enter the parameter k1 in the Robin b.c. for u ');
202 elseif ( ischar ( k1 ) )
203     k1 = str2num ( k1 );
204 end
205 if ( nargin < 10 )
206     k2 = input('Enter the parameter k2 in the Robin b.c. for v ');
207 elseif ( ischar ( k2 ) )
208     k2 = str2num ( k2 );
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 %

```

```

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);

```

```

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 % Do the incomplete LU factorization of B1 and B2.
316 %
317     [ LB1, UB1 ] = ilu ( B1, struct('type','ilutp','droptol',1e-5) );
318     [ LB2, UB2 ] = ilu ( B2, struct('type','ilutp','droptol',1e-5) );
319 %*****80
320 % Time-stepping.
321 %*****80
322     for nt = 1 : N
323         tn = nt * delt;
324 %
325 % Evaluate modified functional response.
326 %
327         hhat = u ./ ( alpha + abs ( u ) );
328 %
329 % Update right-hand-side of linear system.
330 %
331         F = u - u .* abs ( u ) - v .* hhat;
332         G = beta * v .* hhat - gamma * v;
333         rhs_u = u + delt * F;
334         rhs_v = v + delt * G;
335 %
336 % Impose Robin boundary conditions on Gamma1.
337 %
338     for i = 1 : e1
339         node1 = cpp1(i,1);
340         node2 = cpp1(i,2);

```

```

341     x1 = p(1,node1);
342     y1 = p(2,node1);
343     x2 = p(1,node2);
344     y2 = p(2,node2);
345     im_hat1 = 1/m_hat(node1);
346     im_hat2 = 1/m_hat(node2);
347     gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
348     rhs_u(node1) = rhs_u(node1) + delt*k1*u(node1)*im_hat1*gamma12/2;
349     rhs_u(node2) = rhs_u(node2) + delt*k1*u(node2)*im_hat2*gamma12/2;
350     rhs_v(node1) = rhs_v(node1) + delt*k2*v(node1)*im_hat1*gamma12/2;
351     rhs_v(node2) = rhs_v(node2) + delt*k2*v(node2)*im_hat2*gamma12/2;
352 end
353 %
354 % Impose Neumann boundary condition on Gamma2.
355 %
356 for i = 1 : e2
357     node1 = cpp2(i,1);
358     node2 = cpp2(i,2);
359     x1 = p(1,node1);
360     y1 = p(2,node1);
361     x2 = p(1,node2);
362     y2 = p(2,node2);
363     im_hat1 = 1/m_hat(node1);
364     im_hat2 = 1/m_hat(node2);
365     gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
366     rhs_u(node1) = rhs_u(node1) + delt * g2uf (x1,y1,tn) * im_hat1*gamma12/2;
367     rhs_u(node2) = rhs_u(node2) + delt * g2uf (x2,y2,tn) * im_hat2*gamma12/2;
368     rhs_v(node1) = rhs_v(node1) + delt * g2vf (x1,y1,tn) * im_hat1*gamma12/2;
369     rhs_v(node2) = rhs_v(node2) + delt * g2vf (x2,y2,tn) * im_hat2*gamma12/2;
370 end
371 %
372 % Solve for u and v using GMRES.
373 %
374 [u,flagu,relresu,iteru] = gmres ( B1,rhs_u,[],1e-6,[],LB1,UB1,u );
375 if flagu ~= 0
376     flagu
377     relresu
378     iteru
379     error('GMRES did not converge')
380 end
381 [v,flagv,relresv,iterv] = gmres ( B2,rhs_v,[],1e-6,[],LB2,UB2,v );
382 if flagv ~= 0
383     flagv
384     relresv
385     iterv
386     error('GMRES did not converge')
387 end
388 end
389 %*****80
390 % Plot the solutions.
391 %*****80
392 %
393 % Plot U;
394 %
395 figure;
396 set(gcf,'Renderer','zbuffer');
397 trisurf(t',x,y,u,'FaceColor','interp','EdgeColor','interp');

```

```
398 colorbar;
399 axis off;
400 title('u');
401 view ( 2 );
402 axis equal on tight;
403 filename = 'fe2d_nr_fast_u.png';
404 print ( '-dpng', filename );
405 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
406 %
407 % Plot V.
408 %
409 figure;
410 set(gcf,'Renderer','zbuffer');
411 trisurf(t',x,y,v,'FaceColor','interp','EdgeColor','interp');
412 colorbar;
413 axis off;
414 title('v');
415 view ( 2 );
416 axis equal on tight;
417 filename = 'fe2d_nr_fast_v.png';
418 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
419 print ( '-dpng', filename );
420 return
421 end
```