

```

1 function fe2dx_nr_alt_fast ( alpha, beta, gamma, delta, T, delt, u0f, v0f, ...
2   k1, k2, g2uf, g2vf )
3 %*****80
4 %
5 %% FE2DX_NR_ALT_FAST: version of FE2DX_NR_FAST with implicit Robin condition.
6 %
7 % Discussion:
8 %
9 %   FE2DX_NR_ALT_FAST is a "fast" version of FE2DX_NR_ALT.
10 %
11 %   FE2DX_NR_ALT is similar to FE2DX_NR, but uses an implicit approximation
12 %   of the Robin boundary condition. It is a finite element Matlab code
13 %   for Scheme 1 applied to the predator-prey system with Kinetics 1
14 %   solved over a region which has been triangulated. The geometry and grid
15 %   are read from user-supplied files 't_triang.dat' and 'p_coord.dat'
16 %   respectively, as are the list of nodes on which Robin and Neumann boundary
17 %   conditions are to be imposed (from 'bn1_nodes.dat' and 'bn2_nodes.dat'
18 %   respectively).
19 %
20 %   This function has 12 input parameters. All, some, or none of them may
21 %   be supplied as command line arguments or as functional parameters.
22 %   Parameters not supplied through the argument list will be prompted for.
23 %
24 %   The parameters ALPHA, BETA, GAMMA and DELTA appear in the predator-prey
25 %   equations as follows:
26 %
27 %       dUdT =          nabla U +          U*V/(U+ALPHA) + U*(1-U)
28 %       dVdT = delta * nabla V + BETA*U*V/(U+ALPHA) - GAMMA * V
29 %
30 % Licensing:
31 %
32 %   Copyright (C) 2014 Marcus R. Garvie.
33 %   See 'mycopyright.txt' for details.
34 %
35 % Modified:
36 %
37 %   29 April 2014
38 %
39 % Author:
40 %
41 %   Marcus R. Garvie and John Burkardt.
42 %
43 % Reference:
44 %
45 %   Marcus R Garvie, John Burkardt, Jeff Morgan,
46 %   Simple Finite Element Methods for Approximating Predator-Prey Dynamics
47 %   in Two Dimensions using MATLAB,
48 %   Submitted to Bulletin of Mathematical Biology, 2014.
49 %
50 % Parameters:
51 %
52 %   Input, real ALPHA, a parameter in the predator prey equations.
53 %   0 < ALPHA.
54 %
55 %   Input, real BETA, a parameter in the predator prey equations.

```

```

56 %      0 < BETA.
57 %
58 %      Input, real GAMMA, a parameter in the predator prey equations.
59 %      0 < GAMMA.
60 %
61 %      Input, real DELTA, a parameter in the predator prey equations.
62 %      0 < DELTA.
63 %
64 %      Input, real T, the maximum time.
65 %      0 < T.
66 %
67 %      Input, real DELT, the time step to use in integrating from 0 to T.
68 %      0 < DELT.
69 %
70 %      Input, string U0F or function pointer @U0F, a function for the initial
71 %      condition of U(X,Y).
72 %
73 %      Input, string V0F or function pointer @V0F, a function for the initial
74 %      condition of V(X,Y).
75 %
76 %      Input, real K1, the coefficient for the Robin boundary condition
77 %      to be applied to U:  $dU/dn = k1 * U$ .
78 %
79 %      Input, real K2, the coefficient for the Robin boundary condition
80 %      to be applied to V:  $dV/dn = k2 * V$ .
81 %
82 %      Input, string G2UF or function pointer @G2UF, a function for the Neumann
83 %      boundary condition of U(X,Y,T).
84 %
85 %      Input, string G2VF or function pointer @G2VF, a function for the Neumann
86 %      boundary condition of V(X,Y,T).
87 %
88 %*****80
89 % Enter data for mesh geometry.
90 %*****80
91 %
92 % Read in 'p(2,n)', the 'n' coordinates of the nodes.
93 %
94 load p_coord.dat -ascii
95 p = ( p_coord )';
96 %
97 % Read in 't(3,no_elems)', the list of nodes for 'no_elems' elements.
98 %
99 load t_triangu.dat -ascii
100 t = ( round ( t_triangu ) )';
101 %
102 % Read in 'bn1(1,isn1)', the nodes on Gamma1.
103 %
104 load bn1_nodes.dat -ascii
105 bn1 = ( round ( bn1_nodes ) )';
106 %
107 % Read in 'bn2(1,isn2)', the nodes on Gamma2.
108 %
109 load bn2_nodes.dat -ascii
110 bn2 = ( round ( bn2_nodes ) )';
111 %
112 % Construct the connectivity for the nodes on Gamma1.

```

```

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

```

```

170 elseif ( ischar ( delt ) )
171     delt = str2num ( delt );
172 end
173 fprintf ( 1, ' Using ALPHA = %g\n', alpha );
174 fprintf ( 1, ' Using BETA = %g\n', beta );
175 fprintf ( 1, ' Using GAMMA = %g\n', gamma );
176 fprintf ( 1, ' Using DELTA = %g\n', delta );
177 fprintf ( 1, ' Using T = %g\n', T );
178 fprintf ( 1, ' Using DELT = %g\n', delt );
179 %
180 % Initial conditions.
181 %
182 if ( nargin < 7 )
183     u0_str = input ( 'Enter initial data function u0(x,y): ', 's' );
184     u0f = @(x,y) eval ( u0_str );
185 elseif ( ischar ( u0f ) )
186     u0_str = u0f;
187     u0f = @(x,y) eval ( u0_str );
188 end
189 u = ( arrayfun ( u0f, x, y ) )';
190 if ( nargin < 8 )
191     v0_str = input ( 'Enter initial data function v0(x,y): ', 's' );
192     v0f = @(x,y) eval ( v0_str );
193 elseif ( ischar ( v0f ) )
194     v0_str = v0f;
195     v0f = @(x,y) eval ( v0_str );
196 end
197 v = ( arrayfun ( v0f, x, y ) )';
198 %
199 % Boundary conditions.
200 %
201 if ( nargin < 9 )
202     k1 = input('Enter the parameter k1 in the Robin b.c. for u ');
203 elseif ( ischar ( k1 ) )
204     k1 = str2num ( k1 );
205 end
206 if ( nargin < 10 )
207     k2 = input('Enter the parameter k2 in the Robin b.c. for v ');
208 elseif ( ischar ( k2 ) )
209     k2 = str2num ( k2 );
210 end
211 if ( nargin < 11 )
212     g2u_str = input('Enter the Neumann b.c. g2u(x,y,t) for u ', 's');
213     g2uf = @(x,y,t)eval(g2u_str);
214 elseif ( ischar ( g2uf ) )
215     g2u_str = g2uf;
216     g2uf = @(x,y,t)eval(g2u_str);
217 end
218 if ( nargin < 12 )
219     g2v_str = input('Enter the Neumann b.c. g2v(x,y,t) for v ', 's');
220     g2vf = @(x,y,t)eval(g2v_str);
221 elseif ( ischar ( g2vf ) )
222     g2v_str = g2vf;
223     g2vf = @(x,y,t)eval(g2v_str);
224 end
225 %
226 % N = number of time steps.

```

```

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

```

```

284     K_jj = triangle_area*t3/(h2^2);
285 %
286 % Add contributions to vector m_hat.
287 %
288     m_hat(nk)=m_hat(nk)+m_hat_k;
289     m_hat(nj)=m_hat(nj)+m_hat_j;
290     m_hat(ni)=m_hat(ni)+m_hat_i;
291 %
292 % Add contributions to K.
293 %
294     K=K+sparse(nk,ni,K_ki,n,n);
295     K=K+sparse(ni,nk,K_ik,n,n);
296     K=K+sparse(nk,nj,K_kj,n,n);
297     K=K+sparse(nj,nk,K_jk,n,n);
298     K=K+sparse(nk,nk,K_kk,n,n);
299     K=K+sparse(ni,nj,K_ij,n,n);
300     K=K+sparse(nj,ni,K_ji,n,n);
301     K=K+sparse(ni,ni,K_ii,n,n);
302     K=K+sparse(nj,nj,K_jj,n,n);
303 end
304 %
305 % Construct matrix L.
306 %
307     ivec = 1 : n;
308     IM_hat = sparse(ivec,ivec,1./m_hat,n,n);
309     L = delt * IM_hat * K;
310 %
311 % Construct fixed parts of matrices A_{n-1} and C_{n-1}.
312 %
313     A0 = L + sparse(1:n,1:n,1-delt,n,n);
314     C0 = delta * L + sparse(1:n,1:n,1+delt*gamma,n,n);
315 %
316 % Set up A1 and C1 matrices that impose Robin boundary condition on Gamma1.
317 %
318     A1 = sparse ( n, n );
319     C1 = sparse ( n, n );
320     for i = 1 : el
321         node1 = cpp1(i,1);
322         node2 = cpp1(i,2);
323         x1 = p(1,node1);
324         y1 = p(2,node1);
325         x2 = p(1,node2);
326         y2 = p(2,node2);
327         im_hat1 = 1/m_hat(node1);
328         im_hat2 = 1/m_hat(node2);
329         gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
330         A1(node1,node1) = A1(node1,node1) - delt * k1 * im_hat1 * gamma12 / 2;
331         A1(node2,node2) = A1(node2,node2) - delt * k1 * im_hat2 * gamma12 / 2;
332         C1(node1,node1) = C1(node1,node1) - delt * k2 * im_hat1 * gamma12 / 2;
333         C1(node2,node2) = C1(node2,node2) - delt * k2 * im_hat2 * gamma12 / 2;
334     end
335 %*****80
336 % Time-stepping.
337 %*****80
338     for nt = 1 : N
339         tn = nt * delt;
340 %

```

```

341 % Initialize right-hand-side functions.
342 %
343     rhs_u = u;
344     rhs_v = v;
345 %
346 % Update coefficient matrices of linear system.
347 %
348     diag = abs ( u );
349     diag_entries = u ./ ( alpha + abs ( u ) );
350     A = A0 +          delt * sparse(1:n,1:n,diag,n,n);
351     B =          delt * sparse(1:n,1:n,diag_entries,n,n);
352     C = C0 - beta * delt * sparse(1:n,1:n,diag_entries,n,n);
353 %
354 % Impose implicit Robin boundary condition on Gamma1.
355 %
356     A = A + A1;
357     C = C + C1;
358 %
359 % Do the incomplete LU factorisation of C and A.
360 %
361     [ LC, UC ] = ilu ( C, struct('type','ilutp','droptol',1e-5) );
362     [ LA, UA ] = ilu ( A, struct('type','ilutp','droptol',1e-5) );
363 %
364 % Impose Neumann boundary condition on Gamma2.
365 %
366     for i = 1:e2
367         node1 = cpp2(i,1);
368         node2 = cpp2(i,2);
369         x1 = p(1,node1);
370         y1 = p(2,node1);
371         x2 = p(1,node2);
372         y2 = p(2,node2);
373         im_hat1 = 1/m_hat(node1);
374         im_hat2 = 1/m_hat(node2);
375         gamma12 = sqrt((x1-x2)^2 + (y1-y2)^2);
376         rhs_u(node1) = rhs_u(node1) + delt * g2uf (x1,y1,tn) * im_hat1*gamma12/2;
377         rhs_u(node2) = rhs_u(node2) + delt * g2uf (x2,y2,tn) * im_hat2*gamma12/2;
378         rhs_v(node1) = rhs_v(node1) + delt * g2vf (x1,y1,tn) * im_hat1*gamma12/2;
379         rhs_v(node2) = rhs_v(node2) + delt * g2vf (x2,y2,tn) * im_hat2*gamma12/2;
380     end
381 %
382 % Solve for v using GMRES.
383 %
384     [v,flagv,relresv,iterv] = gmres ( C, rhs_v,[],1e-6,[],LC,UC,v );
385     if flagv ~= 0
386         flagv
387         relresv
388         iterv
389         error('GMRES did not converge')
390     end
391     r = rhs_u - B * v;
392 %
393 % Solve for u using GMRES.
394 %
395     [u,flagu,relresu,iteru] = gmres ( A,r,[],1e-6,[],LA,UA,u );
396     if flagu ~= 0
397         flagu

```

```

398     relresu
399     iteru
400     error('GMRES did not converge')
401 end
402 end
403 %*****80
404 % Plot the solutions.
405 %*****80
406 %
407 % Plot U;
408 %
409 figure;
410 set(gcf,'Renderer','zbuffer');
411 trisurf(t',x,y,u,'FaceColor','interp','EdgeColor','interp');
412 colorbar;
413 axis off;
414 title('u');
415 view ( 2 );
416 axis equal on tight;
417 filename = 'fe2dx_nr_alt_fast_u.png';
418 print ( '-dpng', filename );
419 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
420 %
421 % Plot V.
422 %
423 figure;
424 set(gcf,'Renderer','zbuffer');
425 trisurf(t',x,y,v,'FaceColor','interp','EdgeColor','interp');
426 colorbar;
427 axis off;
428 title('v');
429 view ( 2 );
430 axis equal on tight;
431 filename = 'fe2dx_nr_alt_fast_v.png';
432 fprintf ( 1, ' Saved graphics file "%s"\n', filename );
433 print ( '-dpng', filename );
434 return
435 end

```