

```

1 function fe2d_p_fast ( alpha, beta, gamma, delta, a, b, h, T, delt, u0f, v0f )
2 %*****80
3 %
4 %% FE2D_P_FAST applies Scheme 2 with Kinetics 1 to predator prey in the square.
5 %
6 % Discussion:
7 %
8 % FE2D_P_FAST is a "fast" version of FE2D_P.
9 %
10 % FE2D_P is a finite element Matlab code for Scheme 2 applied
11 % to the predator-prey system with Kinetics 1 solved over the square.
12 % The geometry and grid are created within this function, so no external
13 % files need to be imported.
14 %
15 % Periodic boundary conditions are applied.
16 %
17 % This function has 11 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 A, B, the endpoints of the spatial interval.
62 %      The spatial region is a square [A,B]x[A,B].  A < B.
63 %
64 %      Input, real H, the spatial step size used to discretize [A,B].
65 %      0 < H.
66 %
67 %      Input, real T, the maximum time.
68 %      0 < T.
69 %
70 %      Input, real DELT, the time step to use in integrating from 0 to T.
71 %      0 < DELT.
72 %
73 %      Input, string U0F or function pointer @U0F, a function for the initial
74 %      condition of U(X,Y).
75 %
76 %      Input, string V0F or function pointer @V0F, a function for the initial
77 %      condition of V(X,Y).
78 %
79 %*****80
80 %  Enter model parameters.
81 %*****80
82  if ( nargin < 1 )
83      alpha = input ( 'Enter parameter alpha:  ' );
84  elseif ( ischar ( alpha ) )
85      alpha = str2num ( alpha );
86  end
87  if ( nargin < 2 )
88      beta = input ( 'Enter parameter beta:  ' );
89  elseif ( ischar ( beta ) )
90      beta = str2num ( beta );
91  end
92  if ( nargin < 3 )
93      gamma = input ( 'Enter parameter gamma:  ' );
94  elseif ( ischar ( gamma ) )
95      gamma = str2num ( gamma );
96  end
97  if ( nargin < 4 )
98      delta = input ( 'Enter parameter delta:  ' );
99  elseif ( ischar ( delta ) )
100      delta = str2num ( delta );
101  end
102  if ( nargin < 5 )
103      a = input ( 'Enter a in [a,b]^2:  ' );
104  elseif ( ischar ( a ) )
105      a = str2num ( a );
106  end
107  if ( nargin < 6 )
108      b = input ( 'Enter b in [a,b]^2:  ' );
109  elseif ( ischar ( b ) )
110      b = str2num ( b );
111  end
112  if ( nargin < 7 )

```

```

113     h = input ( 'Enter space-step h:  ' );
114 elseif ( ischar ( h ) )
115     h = str2num ( h );
116 end
117 if ( nargin < 8 )
118     T = input ( 'Enter maximum time T:  ' );
119 elseif ( ischar ( T ) )
120     T = str2num ( T );
121 end
122 if ( nargin < 9 )
123     delt = input ( 'Enter time-step delt:  ' );
124 elseif ( ischar ( delt ) )
125     delt = str2num ( delt );
126 end
127 fprintf ( 1, '  Using ALPHA = %g\n', alpha );
128 fprintf ( 1, '  Using BETA = %g\n', beta );
129 fprintf ( 1, '  Using GAMMA = %g\n', gamma );
130 fprintf ( 1, '  Using DELTA = %g\n', delta );
131 fprintf ( 1, '  Using A = %g\n', a );
132 fprintf ( 1, '  Using B = %g\n', b );
133 fprintf ( 1, '  Using H = %g\n', h );
134 fprintf ( 1, '  Using T = %g\n', T );
135 fprintf ( 1, '  Using DELT = %g\n', delt );
136 %
137 % Calculate and assign some constants.
138 %
139 mu = delt / (h^2);
140 J = round ( ( b - a ) / h );
141 dimJ = J + 1;
142 %
143 % N = number of nodes for each dependent variable.
144 %
145 n = dimJ ^ 2;
146 %
147 % N = number of time steps.
148 %
149 N = round ( T / delt );
150 fprintf ( 1, '\n' );
151 fprintf ( 1, '  1D grid size is %d\n', dimJ );
152 fprintf ( 1, '  2D grid size is %d\n', n );
153 fprintf ( 1, '  Taking N = %d time steps\n', N );
154 %
155 % Create the spatial grid.
156 %
157 indexI = 1:dimJ;
158 x = ( indexI - 1 ) * h + a;
159 [ X, Y ] = meshgrid ( x, x );
160 %
161 % Initial conditions.
162 %
163 if ( nargin < 10 )
164     u0_str = input ( 'Enter initial data function u0(x,y):  ', 's' );
165     u0f = @(x,y) eval ( u0_str );
166 elseif ( ischar ( u0f ) )
167     u0_str = u0f;
168     u0f = @(x,y) eval ( u0_str );
169 end

```

```

170 U0 = ( arrayfun ( u0f, X, Y ) )';
171 if ( nargin < 11 )
172     v0_str = input ( 'Enter initial data function v0(x,y): ', 's' );
173     v0f = @(x,y) eval ( v0_str );
174 elseif ( ischar ( v0f ) )
175     v0_str = v0f;
176     v0f = @(x,y) eval ( v0_str );
177 end
178 V0 = ( arrayfun ( v0f, X, Y ) )';
179 %
180 % Convert to 1-D vector.
181 %
182 % 11 21 becomes 11
183 % 12 22          12
184 %                21
185 %                22
186 %
187 u = U0(:);
188 v = V0(:);
189 %*****80
190 % Assembly.
191 %*****80
192 L = sparse ( n, n );
193 L(1,1)=3;
194 L(1,2)=-3/2;
195 L(J+1,J+1)=6;
196 L(J+1,J)=-3;
197 L=L+sparse(2:J,3:J+1,-1,n,n);
198 L=L+sparse(2:J,2:J,4,n,n);
199 L=L+sparse(2:J,1:J-1,-1,n,n);
200 L(1,J+2)=-3/2;
201 L(J+1,2*J+2)=-3;
202 L=L+sparse(2:J,J+3:2*J+1,-2,n,n);
203 L(n-J,n-J)=6;
204 L(n-J,n-J+1)=-3;
205 L(n,n)=3;
206 L(n,n-1)=-3/2;
207 L=L+sparse(n-J+1:n-1,n-J+2:n,-1,n,n);
208 L=L+sparse(n-J+1:n-1,n-J+1:n-1,4,n,n);
209 L=L+sparse(n-J+1:n-1,n-J:n-2,-1,n,n);
210 L(n-J,n-(2*J+1))=-3;
211 L(n,n-dimJ)=-3/2;
212 L=L+sparse(n-J+1:n-1,n-2*J:n-(J+2),-2,n,n);
213 L=L+sparse(J+2:n-dimJ,2*J+3:n,-1,n,n);
214 L=L+sparse(J+2:n-dimJ,1:n-2*dimJ,-1,n,n);
215 L=L+sparse(J+2:n-dimJ,J+2:n-dimJ,4,n,n);
216 L=L+sparse(J+2:n-(J+2),J+3:n-dimJ,-1,n,n);
217 L=L+sparse(J+2:dimJ:n-(2*J+1),J+3:dimJ:n-2*J,-1,n,n);
218 L=L+sparse(2*J+2:dimJ:n-2*dimJ,2*J+3:dimJ:n-(2*J+1),1,n,n);
219 L=L+sparse(J+3:n-dimJ,J+2:n-(J+2),-1,n,n);
220 L=L+sparse(2*J+2:dimJ:n-dimJ,2*J+1:dimJ:n-(J+2),-1,n,n);
221 L=L+sparse(2*J+3:dimJ:n-(2*J+1),2*J+2:dimJ:n-2*dimJ,1,n,n);
222 %
223 % Construct matrices B1 and B2.
224 % Modify B1 and B2 once to impose periodic boundary conditions.
225 %
226 B1 = sparse(1:n,1:n,1,n,n) + mu * L;

```

```

227 B2 = sparse(1:n,1:n,1,n,n) + delta * mu * L;
228 for s = 1 : dimJ
229     k1 = s*dimJ;
230     k2 = (s-1)*dimJ+1;
231     k3 = s;
232     k4 = s+J*dimJ;
233     B1(k1,:)=0;
234     B1(k3,:)=0;
235     B1(k1,k1)=1;
236     B1(k3,k3)=1;
237     B2(k1,:)=0;
238     B2(k3,:)=0;
239     B2(k1,k1)=1;
240     B2(k3,k3)=1;
241 end
242
243 fprintf ( 1, '\n' );
244 fprintf ( 1, ' Matrix size N = %d\n', n );
245 fprintf ( 1, ' B1 nonzeros = %d\n', nnz ( B1 ) );
246 fprintf ( 1, ' B2 nonzeros = %d\n', nnz ( B2 ) );
247 %
248 % Do the incomplete LU factorisation of B1 and B2 once.
249 %
250 [ LB1, UB1 ] = ilu ( B1, struct('type','ilutp','droptol',1e-5) );
251 [ LB2, UB2 ] = ilu ( B2, struct('type','ilutp','droptol',1e-5) );
252 %*****80
253 % Time-stepping.
254 %*****80
255 for nt = 1 : N
256 %
257 % Evaluate modified functional response.
258 %
259 hhat = u ./ ( alpha + abs ( u ) );
260 %
261 % Update right-hand-side of linear system.
262 %
263 F = u - u .* abs ( u ) - v .* hhat;
264 G = beta * v .* hhat - gamma * v;
265 y1 = u + delt * F;
266 y2 = v + delt * G;
267 %
268 % Modify right hand sides to impose periodic boundary conditions.
269 %
270 for s = 1 : dimJ
271     k1 = s*dimJ;
272     k2 = (s-1)*dimJ+1;
273     k3 = s;
274     k4 = s+J*dimJ;
275     y1(k1) = u(k2);
276     y1(k3) = u(k4);
277     y2(k1) = v(k2);
278     y2(k3) = v(k4);
279 end
280 %
281 % Solve for u and v using GMRES.
282 %
283 [ u, flagu, relresu, iteru ] = gmres ( B1, y1, [], 1e-6, [], LB1, UB1, u );

```

```

284     if flagu ~= 0
285         flagu
286         relresu
287         iteru
288         error('GMRES did not converge')
289     end
290     [ v, flagv, relresv, iterv ] = gmres ( B2, y2, [], 1e-6, [], LB2, UB2, v );
291     if flagv ~= 0
292         flagv
293         relresv
294         iterv
295         error('GMRES did not converge')
296     end
297 end
298 %*****80
299 % Plot solutions.
300 %*****80
301 %
302 % Re-order 1-D solution vectors into 2-D solution grids.
303 %
304 V_grid = reshape ( v, dimJ, dimJ );
305 U_grid = reshape ( u, dimJ, dimJ );
306 %
307 % Put solution grids into ij (matrix) orientation.
308 %
309 V_grid = V_grid';
310 U_grid = U_grid';
311 figure;
312 pcolor(X,Y,U_grid);
313 shading interp;
314 colorbar;
315 axis square xy;
316 title('u')
317 filename = 'fe2d_p_fast_u.png';
318 print ( '-dpng', filename );
319 fprintf ( 1, '\n' );
320 fprintf ( 1, ' U contours saved in "%s"\n', filename );
321 figure;
322 pcolor(X,Y,V_grid);
323 shading interp;
324 colorbar;
325 axis square xy;
326 title('v')
327 filename = 'fe2d_p_fast_v.png';
328 print ( '-dpng', filename );
329 fprintf ( 1, ' V contours saved in "%s"\n', filename );
330 return
331 end

```