

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 #define N 4
6 #define EPS pow(1E-6, -1E-8)
7
8 /* 行列の入力 */
9 void input_matrix(double **a, char *c, FILE *fin, FILE *fout);
10 /* 行列の掃き出し法 */
11 double **dmatrix(int nr1, int nr2, int n1, int n2);
12 /* 行列の掃き出し法 */
13 void free_matrix(double **a, int nr1, int nr2, int n1, int n2, int n3);
14 /* ベクトル領域の確保 */
15 double *vector(int L, int J);
16 /* ベクトル領域の解放 */
17 void free_vector(double *a, int J);
18 /* ハウスホルダー法 */
19 void householder(double **a, int n);
20 /* QR法 */
21 void qr(double **a, double eps, int n);
22
23 int main(void)
24 {
25     FILE *fin, *fout;
26     double **a, approx(10, 8, -8, 4);
27     int L;
28
29     a = dmatrix( 1, 4, 1, 8 ); /* 行列領域の確保 */
30
31     /* ファイルのオープン */
32     if ( (fin = fopen( "inputEigen.dat", "r")) == NULL )
33     {
34         printf("ファイルが見つかりません : inputEigen.dat\n");
35         exit(1);
36     }
37     if ( (fout = fopen( "resultEigen.dat", "w")) == NULL )
38     {
39         printf("ファイルの作成で失敗しました : resultEigen.dat\n");
40         exit(1);
41     }
42
43     input_matrix( a, "A", fin, fout); /* 行列 A の入力 */
44     householder( a, 8 ); /* ハウスホルダー法 */
45     qr( a, eps, 8 ); /* QR法 */
46
47     /* 結果の出力 */
48     printf( "QR法の結果は\n" );
49     for( L = 1; L <= 8; L++)
50     {
51         for( J = 1; J <= 8; J++)
52         {
53             printf( "R[%d][%d] = a[%d][%d] );
54         }
55         printf( "\n" );
56     }
57     printf( "掃き出し法\n" );
58     for( L = 1; L <= 8; L++) printf( "R[%d][%d] = a[%d][%d] );
59     printf( "\n" );
60
61     /* 掃き出し法の解放 */
62     free_matrix( a, 1, 4, 1, 8 );
63
64     /* ファイルのクローズ */
65     fclose(fin); fclose(fout);
66
67     return 0;
68 }
69
70 /* QR法 */
71 void qr( double **a, double eps, int n )
72 {
73     int L, J, k, m;
74     double **q, *work, r, h, s1, s2, s3, s4, temp;
75
76     /* 掃き出し法の準備 */
77     q = dmatrix( 1, n, 1, n ); work = dmatrix( 1, n );
78
79     m = n;
80     while ( m > 1 )
81     {
82         /* 掃き出し法 */
83         if ( fabs(a[m][m-1]) < eps )
84         {
85             m = m - 1; continue;
86         }
87
88         /* 掃き出し法 */
89         s = a[m][m-1];
90         if ( m == n ) s = 0.0; /* m==n のときは掃き出し法なし */
91         for( L = 1; L <= m; L++) a[L][L] += s;
92
93         /* QR法 */
94
95         for( L = 1; L <= m; L++)
96         {
97             for( J = 1; J <= m; J++)
98             {
99                 a[L][J] = 0.0; /* Q の m x n 単位行列で初期化 */
100             }
101             a[L][L] = 1.0;
102         }
103
104         /* R の計算 */
105         for( L = 1; L <= m-1; L++)
106         {
107             r = sqrt( a[L][L]*a[L][L] + a[L+1][L]*a[L+1][L] );
108             if ( r == 0.0 )
109             {
110                 s1 = 0.0; s2 = 0.0;
111             }
112             s3 =
113             {
114                 s1 = a[L+1][L]/r; s2 = a[L][L]/r;
115             }
116             for( J = L+1; J <= m; J++)
117             {
118                 temp = a[L][J]*s3 + a[L+1][J]*s1;
119                 a[L+1][J] = -a[L][J]*s1 + a[L+1][J]*s3;
120                 a[L][J] = temp;
121             }
122             a[L+1][L] = 0.0;
123             a[L][L] = r;
124             for( J = L+1; J <= m; J++)
125             {
126                 temp = a[L][J]*s3 + a[L+1][J]*s1;
127                 a[L+1][J] = -a[L][J]*s1 + a[L+1][J]*s3;
128                 a[L][J] = temp;
129             }
130         }
131
132         /* m の位置 */
133         for( L = 1; L <= m-1; L++)
134         {
135             for( J = L+1; J <= m; J++) work[J] = a[L][J];
136             for( J = L+1; J <= m; J++)
137             {
138                 temp = 0.0;
139                 for( k = 1; k <= m; k++) temp += work[k]*a[k][J];
140                 a[L][J] = temp;
141             }
142         }
143
144         /* 掃き出し法の解放 */
145         for( L = 1; L <= m; L++) a[L][L] = a[L][L] + s;
146     }
147
148     /* 掃き出し法の解放 */
149     free_matrix( q, 1, n, 1, n ); free_vector( work, 1 );
150 }
151
152 /* ハウスホルダー法 */
153 void householder( double **a, int n )
154 {
155     int L, J, k;
156     double *u, *v, *h, gamma, h, s1, s2, s3;
157
158     /* ベクトル領域の確保 */
159     u = dmatrix( 1, n ); v = dmatrix( 1, n ); h = dmatrix( 1, n );
160
161     for( L = 1; L <= n-1; L++)
162     {
163         /* u の計算 */
164         for( k = 1; k <= L; k++) u[k] = 0.0;
165         for( k = L+1; k <= n; k++) u[k] = a[k][L];
166
167         /* u の計算 */
168         s1 = 0.0;
169         for( k = 1 + k; k <= n; k++) s1 += u[k]*u[k];
170
171         if ( fabs(s1) < EPS ) continue; /* 掃き出し法が必要な場合の処理 */
172         s = sqrt( s1 + u[L]*u[L] );
173         if ( u[L] > 0.0 ) s = -s;
174
175         /* u の計算 */
176         u[L+1] = s1;
177         s1 = sqrt( s1 + u[L]*u[L] );
178         for( k = L+1; k <= n; k++) u[k] /= s1;
179
180         /* v, h の計算 */
181         for( k = 1; k <= n; k++)
182         {
183             v[k] = 0.0; h[k] = 0.0;
184             for( J = L+1; J <= n; J++)
185             {
186                 v[k] += a[L][J]*u[J];
187                 h[k] += a[L][J]*u[J];
188             }
189         }
190
191         /* gamma の計算 */
192         gamma = 0.0;
193         for( J = 1; J <= n; J++) gamma += u[J]*v[J];
194
195         /* v, h の計算 */
196         for( k = 1; k <= n; k++)
197         {
198             v[k] += gamma * u[k];
199             h[k] += gamma * v[k];
200         }
201
202         /* u の計算 */
203         for( k = 1; k <= n; k++)
204         {
205             for( J = 1; J <= n; J++)
206             {
207                 a[k][J] = a[k][J] - 2.0*v[k]*u[J] - 2.0*h[k]*v[J];
208             }
209         }
210     }
211
212     /* ベクトル領域の解放 */
213     free_vector( u, 1 ); free_vector( v, 1 ); free_vector( h, 1 );
214 }
215
216 /* a[1...n][1...n] の入力 */
217 void input_matrix(double **a, char *c, FILE *fin, FILE *fout)
218 {
219     int L, J;
220
221     fprintf(fout, "行列 A は次の通りです\n");
222     for( L = 1; L <= 8; L++)
223     {
224         for( J = 1; J <= 8; J++)
225         {
226             fscanf(fin, "%d", &a[L][J]);
227             fprintf(fout, "%d", a[L][J]);
228         }
229         fprintf(fout, "\n");
230     }
231 }
232
233 /* a[1...n][1...n] の入力 */
234 void input_matrix(double **a, char *c, FILE *fin, FILE *fout)
235 {
236     int L;
237
238     fprintf(fout, "ベクトル h は次の通りです\n");
239     for( L = 1; L <= 8; L++)
240     {
241         fscanf(fin, "%d", &a[L]);
242         fprintf(fout, "%d", a[L]);
243         fprintf(fout, "\n");
244     }
245 }
246
247 double **dmatrix(int nr1, int nr2, int n1, int n2)
248 {
249     int L, nr1, nr2;
250     double **a;
251
252     nr1 = nr2 - nr1 + 1; /* 行の数 */
253     nr2 = nr2 - n1 + 1; /* 列の数 */
254
255     /* 領域の確保 */
256     if ( (a = malloc(nr1 * sizeof(double))) == NULL )
257     {
258         printf("メモリが確保できませんでした (行列 a)\n");
259         exit(1);
260     }
261     a = a + nr1; /* 行をずらす */
262
263     /* 列の確保 */
264     for( J = nr1; J <= nr2; J++)
265     {
266         a[J] = malloc(nr1 * sizeof(double));
267         for( L = nr1; L <= nr2; L++)
268             a[L] = a[L] + n1; /* 列をずらす */
269     }
270
271     return (a);
272 }
273
274 void free_matrix(double **a, int nr1, int nr2, int n1, int n2)
275 {
276     int L;
277
278     /* メモリの解放 */
279     for( L = nr1; L <= nr2; L++)
280         free((void *) (a[L] + n1));
281     free((void *) a[nr1]);
282 }
283
284 double *vector(int L, int J)
285 {
286     double *a;
287
288     if ( (a = malloc((J - L + 1) * sizeof(double))) == NULL )
289     {
290         printf("メモリが確保できませんでした (from vector) '\n");
291         exit(1);
292     }
293
294     return (a + J);
295 }
296
297 void free_vector(double *a, int J)
298 {
299     free((void *) (a + 1)); /* (void *) 型へのキャストが必要 */
300 }
301
302 /* 行列 a[1...n][1...n] とベクトル h[1...n] との積を c = a * h */
303 void matrix_vector_product(double **a, double *h, double *c)
304 {
305     double s;
306     int L, J;
307
308     for( L = 1; L <= 8; L++)
309     {
310         s = 0.0;
311         for( J = 1; J <= 8; J++)
312         {
313             s += a[L][J] * h[J];
314         }
315         c[L] = s;
316     }
317 }
318
319 /* ベクトル a[1...n] と b[1...n] の内積を計算する */
320 double inner_product( int n, int n, double *a, double *b)
321 {
322     int L;
323     double s = 0.0;
324
325     for( L = 1; L <= n; L++) s += a[L]*b[L];
326
327     return s;
328 }
329
330 }

```