

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
4
5 #define N 10 // 次元数式 */
6 #define EPS pow(10.0, -8.0) // epsilonの値を */
7 #define MAX 100 // 最大反復回数 */
8
9 /* 行列の入力 */
10 void input_matrix( double **a, char c, FILE *fin, FILE *fout);
11 /* ベクトルの入力 */
12 void input_vector( double *a, char c, FILE *fin, FILE *fout);
13 /* 行列の横断線画 */
14 double **dmatrix(int nr1, int nr2, int n1, int n2);
15 /* 行列の横断線図 */
16 void free_dmatrix(double **a, int nr1, int nr2, int n1, int n2);
17 /* ベクトル横断線の描画 */
18 double *dvector(int i, int j);
19 /* 横断線の解放 */
20 void free_dvector(double *a, int i);
21 /* 1 ノルムの計算 a[m...n] */
22 double vector_norm1( double *a, int m, int n );
23 /* A ベクトルa[m...n] と b[m...n] の内積を計算する */
24 double inner_product( int m, int n, double *a, double *b);
25 /* 行列a[1...N][1...N] とベクトル b[1...N] との内積 ca=ab */
26 void matrix_vector_product( double **a, double *b, double *c );
27 /* 行列a配法(2D法) */
28 double *cg( double **a, double *b, double *x );
29
30 int main(void)
31 {
32     FILE *fin, *fout;
33     double **a, *b, *c;
34     int i;
35
36     /* 行列およびベクトルの横断線図 */
37     a = dmatrix(1, N, 1, N); /* 行列 a[1...N][1...N] */
38     b = dvector(1,N); /* b[1...N] */
39     c = dvector(1,N); /* c[1...N] */
40
41     /* ファイルのオープン */
42     if ( (fin = fopen( "input_ap.dat", "r")) == NULL )
43     {
44         printf("ファイルが見つかりません : input_ap.dat\n");
45         exit(1);
46     }
47     if ( (fout = fopen( "output_ap.dat", "w")) == NULL )
48     {
49         printf("ファイルが作成できません : output_ap.dat\n");
50         exit(1);
51     }
52
53     input_matrix( a, 'a', fin, fout ); /* 行列 A の入出力 */
54     input_vector( b, 'b', fin, fout ); /* ベクトル b の入出力 */
55     input_vector( c, 'c', fin, fout ); /* 初期ベクトル c の入出力 */
56     c = cg( a, b, c ); /* 共役法配法(2D法) */
57
58     /* 結果の出力 */
59     fprintf( fout, "Aab の結果を次の通りです\n");
60     for( i = 1; i <= N; i++)
61     {
62         fprintf(fout, "%f\n", a[i]);
63     }
64
65     fclose(fin); fclose(fout); /* ファイルのクローズ */
66
67     /* 横断線の解放 */
68     free_dmatrix( a, 1, N, 1, N ); free_dvector( b, 1 ); free_dvector( c, 1 );
69
70     return 0;
71 }
72
73 /* 共役法配法(2D法) */
74 double *cg(double **a, double *b, double *x)
75 {
76     double eps, *r, *p, *tmp, alpha, beta, work;
77     int i, k=0;
78
79     r = dvector(1,N); /* r[1...N] */
80     p = dvector(1,N); /* p[1...N] */
81     tmp = dvector(1,N); /* tmp[1...N] */
82
83     matrix_vector_product( a, x, tmp ); /* tmp ← A b */
84
85     for( i = 1; i <= N; i++)
86     {
87         p[i] = a[i] - tmp[i] ; r[i] = p[i];
88     }
89
90     do
91     {
92         /* alpha の計算 */
93         matrix_vector_product( a, x, tmp ); /* tmp ← A x_k */
94         work = inner_product( 1, N, p, tmp); /* work ← (p,Ax_k) */
95         alpha = inner_product( 1, N, r, r ) / work ;
96
97         /* x_k(k=1) と r_k(k=1)の計算 */
98         for( i = 1; i <= N; i++) x[i] = x[i] + alpha*p[i];
99         for( i = 1; i <= N; i++) r[i] = r[i] - alpha*tmp[i];
100
101         /* 収束判定 */
102         eps = vector_norm1(r, 1, N);
103         k++; /* 反復回数の更新 */
104         if ( eps < EPS ) goto DONE;
105
106         /* beta と p_k(k=1) の計算 */
107         beta = - inner_product( 1, N, r, tmp) / work;
108         for( i = 1; i <= N; i++) p[i] = r[i] + beta*p[i];
109     }while( k < MAX );
110
111     OUTPUT:
112     /* 横断線の解放 */
113     free_dvector( r, 1 ); free_dvector( a, 1 ); free_dvector( tmp, 1 );
114     if ( k == MAX )
115     {
116         printf("答えが見つかりませんでした\n");
117         exit(1);
118     }
119     else
120     {
121         printf("反復回数%d回です\n", k); /* 反復回数を画面に表示 */
122         return x;
123     }
124 }
125
126 /* 行列 a[1...N][1...N] とベクトル b[1...N] との内積 c ← ab */
127 void matrix_vector_product(double **a, double *b, double *c)
128 {
129     double wk;
130     int i, j;
131
132     for ( i = 1; i <= N; i++)
133     {
134         wk = 0.0;
135         for ( j = 1; j <= N; j++)
136         {
137             wk += a[i][j]*b[j];
138         }
139         c[i] = wk;
140     }
141 }
142
143 /* a[1...N][1...N] の入力 */
144 void input_matrix(double **a, char c, FILE *fin, FILE *fout)
145 {
146     int i, j;
147
148     fprintf(fout, "行列a (2D法の通りです)\n, c);
149     for ( i = 1; i <= N; i++)
150     {
151         for ( j = 1; j <= N; j++)
152         {
153             fscanf(fin, "%f", &a[i][j]);
154             fprintf(fout, "%8.2f\t", a[i][j]);
155         }
156         fprintf(fout, "\n");
157     }
158 }
159
160 /* b[1...N]の入力 */
161 void input_vector(double *b, char c, FILE *fin, FILE *fout)
162 {
163     int i;
164
165     fprintf(fout, "ベクトルb (2D法の通りです)\n, c);
166     for ( i = 1; i <= N; i++)
167     {
168         fscanf(fin, "%f", &b[i]);
169         fprintf(fout, "%8.2f\t", b[i]);
170         fprintf(fout, "\n");
171     }
172 }
173
174 double **dmatrix(int nr1, int nr2, int n1, int n2)
175 {
176     int i, mrow, ncol;
177     double **a;
178
179     mrow = nr2 - nr1 + 1; /* 行の数 */
180     ncol = n2 - n1 + 1; /* 列の数 */
181
182     /* 行の確保 */
183     if ((a = malloc(mrow * sizeof(double))) == NULL)
184     {
185         printf("メモリが確保できません (行列 a)\n");
186         exit(1);
187     }
188     a = a + nr1; /* 行をずらす */
189
190     /* 列の確保 */
191     for ( i = nr1; i <= nr2; i++)
192         a[i] = malloc(ncol * sizeof(double));
193     for ( i = nr1; i <= nr2; i++)
194         a[i] = a[i] - n1; /* 列をずらす */
195
196     return (a);
197 }
198
199 void free_dmatrix(double **a, int nr1, int nr2, int n1, int n2)
200 {
201     int i;
202
203     /* メモリの解放 */
204     for ( i = nr1; i <= nr2; i++)
205         free((void *) (a[i] - n1));
206     free((void *) (a - nr1));
207 }
208
209 double *dvector(int i, int j)
210 {
211     double *a;
212
213     if ((a = malloc((j) - i + 1) * sizeof(double))) == NULL)
214     {
215         printf("メモリが確保できません (from dvector) \n");
216         exit(1);
217     }
218
219     return (a - i);
220 }
221
222 void free_dvector(double *a, int i)
223 {
224     free((void *) (a + i)); /* (void *) 型へのキャストが必要 */
225 }
226
227 /* 比較演算 (共通) */
228 int double_comp(const void *a1, const void *a2)
229 {
230     const double a1 = *((double *)a1); /* (double *) へキャスト */
231     const double a2 = *((double *)a2); /* (double *) へキャスト */
232     if (a1 < a2)
233     {
234         return -1;
235     }
236     else if (a1 == a2)
237     {
238         return 0;
239     }
240     else
241     {
242         return 1;
243     }
244 }
245
246 /* 最大値ノルムの計算 a[m...n] */
247 double vector_norm_max(double *a, int m, int n)
248 {
249     int i, tmp;
250     tmp = a - m + 1; /* 全要素の計算 */
251     for ( i = m; i <= n; i++)
252         a[i] = fabs(a[i]);
253     /* 最大値: 関数アレイが a 型に作られていることに注意 */
254     qsort(a + m, tmp, sizeof(a[i]), double_comp);
255     return a[i];
256 }
257
258 /* ベクトル a[m...n] と b[m...n] の内積を計算する */
259 double inner_product( int m, int n, double *a, double *b)
260 {
261     int i;
262     double s = 0.0;
263
264     for( i = m; i <= n; i++) s += a[i]*b[i];
265
266     return s ;
267 }
268
269 /* 1 ノルムの計算 a[m...n] */
270 double vector_norm1(double *a, int m, int n)
271 {
272     int i;
273     double norm = 0.0;
274     for ( i = m; i <= n; i++)
275     {
276         norm += fabs(a[i]);
277     }
278     return norm;
279 }
```