

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define N 4 /* N 次正方行列 */
5
6 /* 行列の入力 */
7 void input_matrix( double **a, char c, FILE *fin, FILE *fout);
8 /* ベクトルの入力 */
9 void input_vector( double *b, char c, FILE *fin, FILE *fout);
10 /* 行列の領域確保 */
11 double **dmatrix(int nr1, int nr2, int n11, int n12);
12 /* 行列の領域解放 */
13 void free_dmatrix(double **a, int nr1, int nr2, int n11, int n12);
14 /* ベクトル領域の確保 */
15 double *dvector(int i, int j);
16 /* 領域の解放 */
17 void free_dvector(double *a, int i);
18 /* 修正コレスキー分解 */
19 double **cholesky_decomp( double **a );
20 /* 修正コレスキー分解を利用して連立一次方程式を解く */
21 double *cholesky_solve( double **a, double *b );
22
23 int main(void)
24 {
25     FILE *fin, *fout;
26     double **a, *b;
27     int i;
28
29     /* 行列およびベクトルの領域確保 */
30     a = dmatrix(1, N, 1, N); /* 行列 a[1...N][1...N] */
31     b = dvector(1,N); /* b[1...N] */
32
33     /* ファイルのオープン */
34     if ( (fin = fopen( "input_cho.dat", "r")) == NULL )
35     {
36         printf("ファイルが見つかりません : input_cho.dat\n");
37         exit(1);
38     }
39     if( (fout = fopen( "output_cho.dat", "w")) == NULL )
40     {
41         printf("ファイルが作成できません : output_cho.dat\n");
42         exit(1);
43     }
44
45     input_matrix( a, 'A', fin, fout ); /* 行列 A の入出力 */
46     input_vector( b, 'b', fin, fout ); /* ベクトル b の入出力 */
47     a = cholesky_decomp( a ); /* 修正コレスキー分解 */
48     b = cholesky_solve( a, b ); /* 前進代入・後退代入 */
49
50     /* 結果の出力 */
51     fprintf( fout, "Ax=b の解は次の通りです\n");
52     for( i = 1 ; i <= N ; i++)
53     {
54         fprintf(fout, "%f\t", b[i]);
55         fprintf( fout, "\n");
56     }
57
58     fclose(fin); fclose(fout); /* ファイルのクローズ */
59
60     /* 領域の解放 */
61     free_dmatrix( a, 1, N, 1, N ); free_dvector( b, 1 );
62
63     return 0;
64 }
65
66 /* 修正コレスキー分解 */
67 double **cholesky_decomp( double **a )
68 {
69     int i, j, k;
70     double tmp;
71
72     for( i = 2; i <= N; i++)
73     {
74         for( j = 1; j <= i-1; j++)
75         {
76             tmp = 0.0;
77             for ( k = 1; k <= j-1; k++)
78             {
79                 tmp += a[i][k]*a[k][k]*a[j][k];
80             }
81             a[i][j] = (a[i][j] - tmp) / a[j][j];
82         }
83         tmp = 0.0;
84         for ( k = 1; k <= j-1; k++)
85         {
86             tmp += a[i][k]*a[i][k]*a[k][k];
87         }
88         a[i][i] = a[i][i] - tmp;
89     }
90     return a;
91 }
92
93 /* 修正コレスキー分解を利用して連立一次方程式を解く */
94 double *cholesky_solve( double **a, double *b )
95 {
96     int i, j;
97     double tmp;
98
99     /* LDy = b */
100     b[1] = b[1]/a[1][1];
101     for( i = 2; i <= N; i++)
102     {
103         tmp = 0.0;
104         for( j = 1; j <= i-1; j++)
105         {
106             tmp += a[j][j]*a[i][j]*b[j];
107         }
108         b[i] = ( b[i] - tmp ) / a[i][i];
109     }
110
111     /* L^t x = y */
112     for( i = N-1; i >= 1; i--)
113     {
114         tmp = 0.0;
115         for( j = i+1; j <= N; j++)
116         {
117             tmp += a[j][i] * b[j];
118         }
119         b[i] = b[i] - tmp;
120     }
121
122     return b;
123 }
124
125 /* a[1...N][1...N] の入力 */
126 void input_matrix(double **a, char c, FILE *fin, FILE *fout)
127 {
128     int i, j;
129
130     fprintf(fout, "行列%c は次の通りです\n", c);
131     for (i = 1; i <= N; i++)
132     {
133         for (j = 1; j <= N; j++)
134         {
135             fscanf(fin, "%lf", &a[i][j]);
136             fprintf(fout, "%5.2f\t", a[i][j]);
137         }
138         fprintf(fout, "\n");
139     }
140 }
141
142 /* b[1...N]の入力 */
143 void input_vector(double *b, char c, FILE *fin, FILE *fout)
144 {
145     int i;
146
147     fprintf(fout, "ベクトル%c は次の通りです\n", c);
148     for(i = 1; i <= N; i++){
149         fscanf(fin, "%lf", &b[i]);
150         fprintf(fout, "%5.2f\t", b[i]);
151         fprintf(fout, "\n");
152     }
153 }
154
155 double **dmatrix(int nr1, int nr2, int n11, int n12)
156 {
157     int i, nrow, ncol;
158     double **a;
159
160     nrow = nr2 - nr1 + 1; /* 行の数 */
161     ncol = n12 - n11 + 1; /* 列の数 */
162
163     /* 行の確保 */
164     if((a = malloc(nrow * sizeof(double *))) == NULL){
165         printf("メモリが確保できません (行列 a)\n");
166         exit(1);
167     }
168     a = a - nr1; /* 行をずらす */
169
170     /* 列の確保 */
171     for(i = nr1; i <= nr2; i++) a[i] = malloc(ncol * sizeof(double));
172     for(i = nr1; i <= nr2; i++) a[i] = a[i] - n11; /* 列をずらす */
173
174     return (a);
175 }
176
177 void free_dmatrix(double **a, int nr1, int nr2, int n11, int n12)
178 {
179     int i;
180
181     /* メモリの解放 */
182     for(i = nr1; i <= nr2; i++) free((void *) (a[i] + n11));
183     free((void *) (a + nr1));
184 }
185
186 double *dvector(int i, int j)
187 {
188     double *a;
189
190     if((a = malloc( ((j - i + 1) * sizeof(double))) ) == NULL)
191     {
192         printf("メモリが確保できません (from dvector) \n");
193         exit(1);
194     }
195
196     return (a - i);
197 }
198
199 void free_dvector(double *a, int i)
200 {
201     free( (void *) (a + i) ); /* (void *) 型へのキャストが必要 */
202 }

```