

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 #define ROW    3 /* 行の要素数 */
6 #define COLUMN 4 /* 列の要素数 */
7
8 /* 行列の領域確保 */
9 double **dmatrix(int nr1, int nr2, int n11, int n12);
10 /* 行列の領域解放 */
11 void free_dmatrix(double **a, int nr1, int nr2, int n11, int n12);
12 double *dvector(int i, int j); /* ベクトル領域の確保 */
13 void free_dvector(double *a, int i); /* 領域の解放 */
14
15 /* 比較関数 */
16 int double_comp(const void *s1, const void *s2);
17 /* 1 ノルムの計算 a[m1...m2][n1...n2] */
18 double matrix_norm1(double **a, int m1, int m2, int n1, int n2);
19 /* 最大値ノルムの計算 a[m1...m2][n1...n2] */
20 double matrix_norm_max(double **a, int m1, int m2, int n1, int n2);
21
22 int main(void)
23 {
24     int i, j;
25     double **a;
26
27     a = dmatrix(1, ROW, 1, COLUMN); /* 行列 a[1...ROW][1...COLUMN] */
28     /* 行列の定義 */
29     printf("A=\n");
30     for(i = 1; i <= ROW; i++){
31         for(j = 1; j <= COLUMN; j++){
32             a[i][j] = 2.0 * (i + j) * pow(-1.0, j);
33             printf("%f\t", a[i][j]);
34         }
35         printf("\n");
36     }
37
38     printf("A の 1 ノルムは%f\n", matrix_norm1(a, 1, ROW, 1, COLUMN));
39     printf("A の最大値ノルムは%f\n", matrix_norm_max(a, 1, ROW, 1, COLUMN));
40
41     /* 行列領域の解放 */
42     free_dmatrix(a, 1, ROW, 1, COLUMN);
43
44     return 0;
45 }
46
47 double **dmatrix(int nr1, int nr2, int n11, int n12)
48 {
49     int i, nrow, ncol;
50     double **a;
51
52     nrow = nr2 - nr1 + 1; /* 行の数 */
53     ncol = n12 - n11 + 1; /* 列の数 */
54
55     /* 行の確保 */
56     if ((a = malloc(nrow * sizeof(double *))) == NULL)
57     {
58         printf("メモリが確保できません (行列 a)\n");
59         exit(1);
60     }
61     a = a - nr1; /* 行をずらす */
62
63     /* 列の確保 */
64     for (i = nr1; i <= nr2; i++)
65         a[i] = malloc(ncol * sizeof(double));
66     for (i = nr1; i <= nr2; i++)
67         a[i] = a[i] - n11; /* 列をずらす */
68
69     return (a);
70 }
71
72 void free_dmatrix(double **a, int nr1, int nr2, int n11, int n12)
73 {
74     int i;
75
76     /* メモリの解放 */
77     for (i = nr1; i <= nr2; i++)
78         free((void *) (a[i] + n11));
79     free((void *) (a + nr1));
80 }
81
82 double *dvector(int i, int j) /* a[i]~a[j] の領域を確保 */
83 {
84     double *a;
85
86     if ((a = malloc(((j - i + 1) * sizeof(double)))) == NULL)
87     {
88         printf("メモリが確保できません (from dvector) \n");
89         exit(1);
90     }
91
92     return (a - i);
93 }
94
95 void free_dvector(double *a, int i)
96 {
97     free((void *) (a + i)); /* (void *) 型へのキャストが必要 */
98 }
99
100 /* 比較関数 (昇順) */
101 int double_comp(const void *s1, const void *s2)
102 {
103     const double a1 = *((double *)s1); /* (double *) へキャスト */
104     const double a2 = *((double *)s2); /* (double *) へキャスト */
105     if (a1 < a2)
106     {
107         return -1;
108     }
109     else if (a1 == a2)
110     {
111         return 0;
112     }
113     else
114     {
115         return 1;
116     }
117 }
118
119 /* 1 ノルムの計算 */
120 double matrix_norm1(double **a, int m1, int m2, int n1, int n2)
121 {
122     int i, j, tmp;
123     double *work, norm;
124     work = dvector(n1, n2); /* ベクトル work[n1...n2] */
125
126     /* 列和の計算 */
127     for(j = n1; j <= n2; j++){
128         work[j] = 0.0;
129         for(i = m1; i <= m2; i++){
130             work[j] += fabs(a[i][j]);
131         }
132     }
133
134     tmp = n2 - n1 + 1; /* 列数の計算 */
135     /* 並べ替え:先頭アドレスがn1 だけずれていることに注意 */
136     qsort(work + n1, tmp, sizeof(work[0]), double_comp);
137
138     norm = work[n2];
139     free_dvector(work, n1);
140
141     return norm;
142 }
143
144 /* 最大値ノルムの計算 */
145 double matrix_norm_max(double **a, int m1, int m2, int n1, int n2)
146 {
147     int i, j, tmp;
148     double *work, norm;
149     work = dvector(m1, m2); /* ベクトル work[m1...m2] */
150
151     /* 行和の計算 */
152     for(i = m1; i <= m2; i++){
153         work[i] = 0.0;
154         for(j = n1; j <= n2; j++){
155             work[i] += fabs(a[i][j]);
156         }
157     }
158
159     tmp = m2 - m1 + 1; /* 行数の計算 */
160
161     /* 並べ替え:先頭アドレスがm1だけずれていることに注意 */
162     qsort(work + m1, tmp, sizeof(work[0]), double_comp);
163
164     norm = work[m2];
165     free_dvector(work, m1);
166
167     return norm;
168 }

```