

```

1 // 6_2.cpp
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <math.h>
5
6 const int MAX_SIZE = 7;      // 行列の最大サイズ
7 const double EPS = 1e-6;    // ゼロ判定用のしきい値
8 const int STR_LEN = 200;    // 読み取り用バッファサイズ
9
10 typedef double Matrix[MAX_SIZE][MAX_SIZE * 2]; // A | I の拡大行列
11
12 // 行列をファイルから読み込む
13 int scan_mat(FILE* ifp, Matrix A) {
14     char line[STR_LEN]; // 読み込まれた文字列の保存先
15     int row = 0;
16
17     while (fgets(line, STR_LEN, ifp) != NULL && row < MAX_SIZE) {
18         char* ptr = line; // strtod() で使うために、文字列の先頭アドレスをポインタptrに保存。
19                             // C言語では、配列名（例えば line）は自動的に先頭要素のポインタに変換されるという仕様がある。&line[0]と同じ。
20         for (int j = 0; j < MAX_SIZE; j++) {
21             A[row][j] = strtod(ptr, &ptr); // strtod(ptr, &ptr)は現在のptrの位置からdouble型の数値を1つ読み取る。
22                                             // 同時にptrを次の数値の位置に自動で更新する。
23         }
24         row++;
25     }
26
27     return row; // 実際の行列サイズを返す
28 }
29
30 // サンプルプログラムの行列を読み込む関数の解答。このプログラムでは未使用。
31 int scan_mat_sample(FILE* ifp, Matrix A) {
32     int col[MAX_SIZE] = {0};
33
34     for (int i = 0; i < MAX_SIZE + 1; i++) {
35         char line[STR_LEN] = {0};
36         int str_end = 0;
37
38         if (fgets(line, STR_LEN, ifp) == NULL) {
39             for (int k = 0; k < i; k++) {
40                 if (i != col[k]) {
41                     printf("In row %d, number of columns is inconsistent.\n", k + 1);
42                     return 0;
43                 }
44             }
45             return i; // 行数を返す
46         }
47
48         for (int z = 0; z < STR_LEN + 1; z++) {
49             if (line[z] == '\0' || line[z] == '\n') {
50                 str_end = z;
51                 break;
52             }
53         }
54
55         char* ptr = line;
56         for (int j = 0; j < MAX_SIZE + 1; j++) {
57             A[i][j] = strtod(ptr, &ptr);
58             if (ptr == &line[str_end]) {
59                 col[i] = j;
60                 break;
61             }
62             ++ptr;
63         }
64     }
65
66     return 0;
67 }
68
69 // A に単位行列 I を右に結合
70 void make_combining_I_matrix(Matrix A, int size) {
71     for (int i = 0; i < size; i++) {
72         for (int j = size; j < size * 2; j++) {
73             A[i][j] = (i == j - size) ? 1.0 : 0.0;
74         }
75     }
76 }

```

```

77
78 // 行列の表示
79 void print_matrix(const Matrix A, int size) {
80     for (int i = 0; i < size; i++) {
81         for (int j = 0; j < size * 2; j++) {
82             if (j == size) {
83                 printf(" |");
84             }
85             printf("%8.3f", A[i][j]);
86         }
87         printf("\n");
88     }
89     printf("\n");
90 }
91
92 // ピボット選択：列の絶対値が最大の行と入れ替える
93 int partial_pivot(Matrix A, int size, int row) {
94     int max_row = row;
95     double max_val = fabs(A[row][row]);
96
97     for (int i = row + 1; i < size; i++) {
98         if (fabs(A[i][row]) > max_val) {
99             max_val = fabs(A[i][row]);
100             max_row = i;
101         }
102     }
103
104     if (max_val < EPS) {
105         printf("Error: Singular matrix.\n");
106         return 0;
107     }
108
109     if (max_row != row) {
110         for (int j = 0; j < size * 2; j++) {
111             double tmp = A[row][j];
112             A[row][j] = A[max_row][j];
113             A[max_row][j] = tmp;
114         }
115         printf("Swapped row %d and row %d.\n\n", row + 1, max_row + 1);
116         print_matrix(A, size);
117     }
118
119     return 1;
120 }
121
122 // 全要素がゼロの行があるか確認
123 int check_zero_row(const Matrix A, int size) {
124     for (int i = 0; i < size; i++) {
125         double sum = 0;
126         for (int j = 0; j < size; j++) {
127             sum += fabs(A[i][j]);
128         }
129         if (sum < EPS) {
130             return 0;
131         }
132     }
133     return 1;
134 }
135
136 // ガウス・ジョルダン法で逆行列を求める
137 int gauss_jordan(Matrix A, int size) {
138     for (int i = 0; i < size; i++) {
139         if (!partial_pivot(A, size, i)) {
140             return 0;
141         }
142
143         double pivot = A[i][i];
144         for (int j = 0; j < size * 2; j++) {
145             A[i][j] /= pivot;
146         }
147
148         for (int k = 0; k < size; k++) {
149             if (k == i) {
150                 continue;
151             }
152             double factor = A[k][i];
153             for (int j = 0; j < size * 2; j++) {

```

```

154         A[k][j] -= factor * A[i][j];
155     }
156 }
157
158 print_matrix(A, size);
159
160 if (!check_zero_row(A, size)) {
161     printf("Matrix A is singular (rank deficient).\n");
162     return 0;
163 }
164 }
165
166 return 1;
167 }
168
169 // 逆行列の出力 (右側だけ)
170 void print_inv_matrix(const Matrix A, int size) {
171     printf("----- Inverse Matrix -----\n");
172     for (int i = 0; i < size; i++) {
173         for (int j = size; j < size * 2; j++) {
174             printf("%.3f", A[i][j]);
175         }
176         printf("\n");
177     }
178 }
179
180 int main(void) {
181     const char* const ifile = "mat_A_rand.txt";
182     FILE* ifp = fopen(ifile, "rt");
183
184     if (!ifp) {
185         fprintf(stderr, "Can't open file %s\n", ifile);
186         return 1;
187     }
188
189     Matrix A = {0.0}; //初期化
190     int size = scan_mat_sample(ifp, A);
191     fclose(ifp);
192
193     if (size > 0) {
194         make_combining_I_matrix(A, size);
195         print_matrix(A, size);
196
197         if (gauss_jordan(A, size)) {
198             print_inv_matrix(A, size);
199         }
200     } else {
201         fprintf(stderr, "Failed to load matrix.\n");
202     }
203
204     return 0;
205 }

```