

```

1 //
2 // 7_2.cpp
3 //
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <math.h>
8 #include <windows.h>
9
10 const int N = 100; // 行列の行数
11 const double EPS = 1e-6; // ピボットが 0 かどうか判定する値
12 typedef double Matrix[N][N+1]; // 行列の型を定義
13
14 // csv ファイルから A_mat と b_vec を読み込み、拡大行列を作る関数
15 void read_A_b(char* A_file, char* b_file, Matrix A1, Matrix A2){
16     FILE* A_fp = fopen(A_file, "rt"); // A_mat の入力ファイルポインタ
17     FILE* b_fp = fopen(b_file, "rt"); // b_vec の入力ファイルポインタ
18
19     if(!A_fp){ // A_mat が開けなかったらエラー
20         fprintf(stderr, "Can't open file %s\n", A_file);
21         exit(2);
22     }
23
24     if(!b_fp){ // b_vec が開けなかったらエラー
25         fprintf(stderr, "Can't open file %s\n", b_file);
26         exit(3);
27     }
28
29     for(int i=0; i<N; i++){
30         for(int j=0; j<N; j++){
31             fscanf(A_fp, "%lf,", &A1[i][j]); // A_mat は 1~N 列目
32             A2[i][j] = A1[i][j];
33         }
34         fscanf(b_fp, "%lf,", &A1[i][N]); // b_vec は N+1 列目
35         A2[i][N] = A1[i][N];
36     }
37     fclose(A_fp); // 入力ファイル(A_mat)を閉じる
38     fclose(b_fp); // 入力ファイル(b_vec)を閉じる
39 }
40
41 // 最大値を検索し、その行数を返す関数
42 int search_max(Matrix A, int k){
43     double max = 0.0; // 最大値を格納する変数
44     int max_row = k; // 最大値がある行数を格納する変数
45     for(int i=k; i<N; i++){
46         double val = fabs(A[i][k]);
47         if(val>EPS && val>fabs(max)){
48             max = A[i][k];
49             max_row = i;
50         }
51     }
52     return max_row; // 最大値がある行数を返す
53 }
54
55 // 行を入れ替える関数
56 void change_row(Matrix A, int i, int max_row){
57     for(int j=0; j<N+1; j++){
58         double tmp = A[i][j]; // 一時的に格納する変数
59         A[i][j] = A[max_row][j];
60         A[max_row][j] = tmp;
61     }
62 }
63
64 // ガウス・ジョルダン法
65 int gauss_jordan(Matrix A){
66     int rankcount = 0;
67     for(int i=0; i<N; i++){
68         int max_row = search_max(A, i); // 最大値がある行数を検索
69         if(fabs(A[i][i]) < EPS){
70             for(int j=i+1; j<N; j++){
71                 if(fabs(A[j][i]) > EPS){
72                     change_row(A, i, j);
73                     break;
74                 }
75             }
76         }

```

```

77     double pivot = A[i][i]; // ピボットを定義
78     if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
79         for(int j=i; j<N+1; j++){ // 各列をピボットで除算
80             A[i][j] /= pivot;
81         }
82         for(int k=0; k<N; k++){ // 掃き出し計算
83             if(k == i){
84                 continue;
85             }
86             double tmp = A[k][i]; // 一時的に値を保存する変数
87             for(int j=i; j<N+1; j++){
88                 A[k][j] -= tmp * A[i][j];
89             }
90         }
91     }else{
92         rankcount++; // ピボットが 0 であれば1つランク落ち
93     }
94 }
95 return N - rankcount;
96 }
97
98 // 前進消去
99 int forward_elimination(Matrix A){
100     int rankcount = 0;
101     for(int i=0; i<N; i++){
102         int max_row = search_max(A, i); // 最大値がある行数を検索
103         if(i != max_row){
104             change_row(A, i, max_row); // 最大値がある行数と行を入れ替え
105         }
106         double pivot = A[i][i];
107         if(fabs(pivot) > EPS){ // ピボットが 0 でなければ
108             for(int j=i; j<N+1; j++){ // 各列をピボットで除算
109                 A[i][j] /= pivot;
110             }
111             for(int k=i+1; k<N; k++){ // 掃き出し計算
112                 double tmp = A[k][i]; // 一時的に値を保存する変数
113                 for(int j=i; j<N+1; j++){
114                     A[k][j] -= tmp * A[i][j];
115                 }
116             }
117         }else{
118             rankcount++; // ピボットが 0 であれば1つランク落ち
119         }
120     }
121     return N - rankcount;
122 }
123
124 // 後退代入
125 void backward_substitution(Matrix A, int rank){
126     if(rank < N){ // 行数と階数が一致していれば後退代入
127         return;
128     }
129     for(int i=N-2; i>=0; i--){
130         for(int j=N-1; j>i; j--){
131             A[i][N] -= A[i][j]*A[j][N];
132         }
133     }
134 }
135
136 // ガウス消去法
137 void gauss_elimination(Matrix A, int& rank){
138     rank = forward_elimination(A); // 前進消去
139     backward_substitution(A, rank); // 後退代入
140 }
141
142 // 行列のコピー
143 void copy_matrix(Matrix A, Matrix copyA){
144     for(int i=0; i<N; i++){
145         for(int j=0; j<N+1; j++){
146             copyA[i][j] = A[i][j];
147         }
148     }
149 }
150
151 // ガウスジョルダン法およびガウス消去法の計算時間を求める関数
152 void check_culc_time(int culc_num, Matrix A1, Matrix A2){
153     DWORD time1_start, time1_end, time2_start, time2_end; // 計算時間を格納する変数を定義

```

```

154 int rank1 = 0; // 階数を定義
155 int rank2 = 0; // 階数を定義
156
157 Matrix copy_A1 = {0.0}; // 拡大行列を定義
158 Matrix copy_A2 = {0.0}; // 拡大行列を定義
159
160 // ガウスジョルダン法の計算時間
161 time1_start = timeGetTime();
162 for(int i=0; i<culc_num; i++){
163     copy_matrix(A1, copy_A1);
164     rank1 = gauss_jordan(copy_A1); // ガウスジョルダン法で連立方程式の解を計算
165 }
166 time1_end = timeGetTime();
167
168 // ガウス消去法の計算時間
169 time2_start = timeGetTime();
170 for(int i=0; i<culc_num; i++){
171     copy_matrix(A2, copy_A2);
172     gauss_elimination(copy_A2, rank2); // ガウス消去法で連立方程式の解を計算
173 }
174 time2_end = timeGetTime();
175
176 printf("Number of Repetitions : %d\n",culc_num);
177 printf("\n----- Gauss Jordan method -----\n");
178 for(int i=0; i<N; i++){
179     printf("x[%d] = %6.3f\n", i+1, copy_A1[i][N]);
180 }
181 printf("Calculation time : %ld [ms]\n", time1_end-time1_start);
182 printf("\n----- Gauss Elimination method -----\n");
183 for(int i=0; i<N; i++){
184     printf("x[%d] = %6.3f\n", i+1, copy_A2[i][N]);
185 }
186 printf("Calculation time : %ld [ms]\n", time2_end-time2_start);
187 }
188
189 // メイン関数
190 int main(int argc, char* argv[]){
191     if(argc!=3){ //実数時引数が2個なかったらエラーとする
192         fprintf(stderr, "Usage: %s inputfile outputfile\n", argv[0]);
193         exit(1);
194     }
195     char* A_file = argv[1]; //パラメータの1番目を入力ファイル名
196     char* b_file = argv[2]; //パラメータの2番目を入力ファイル名
197     Matrix A1 = {0.0}; // 拡大行列を定義
198     Matrix A2 = {0.0}; // 拡大行列を定義
199     read_A_b(A_file, b_file, A1, A2); // csv ファイルから A_mat と b_vec を読み込み、拡大行列を作成
200
201     int culc_num = 1000; // 反復回数
202     check_culc_time(culc_num, A1, A2);
203
204     return 0;
205 }

```