

```

1 //
2 // 9_1.cpp
3 //
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <math.h>
8
9 const int data = 2000;      // データ点数
10 const int M = 10;          // 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
13 typedef double Matrix[M+1][M+2]; // 行列の型を定義
14
15 typedef struct{
16     double x;      // 測定データ x
17     double y;      // 測定データ y
18 }input_data;
19
20 // ファイルを読み込む関数
21 int scan_file(const char* ifile, input_data* Indata){
22     int i = 0;
23     FILE *ifp = NULL;
24     if((ifp = fopen(ifile, "r")) == NULL){
25         fprintf(stderr, "Can't open file %s\n", ifile);
26         exit(-1);
27     }
28     while(fscanf(ifp, "%lf,%lf", &Indata[i].x, &Indata[i].y) != EOF){
29         i++;
30         if(i==data){
31             printf("Warning: Too many data points(n > %d). Stop the process.",data);
32             exit(0);
33         }
34     }
35     fclose(ifp);
36     return i; // ファイルのデータ点の個数を返す
37 }
38
39 // ベキ乗計算をする関数
40 double ipow(double p, int n){
41     double s = 1;
42     for (int k=0; k<n; k++){
43         s *= p;
44     }
45     return s;
46 }
47
48 // 最大値を検索し、その行数を返す関数
49 int search_max(Matrix A, int k, const int N){
50     double max = 0.0; // 最大値を格納する変数
51     int max_row = k; // 最大値がある行数を格納する変数
52     for(int i=0; i<N-k; i++){
53         if(fabs(max)<fabs(A[i+k][k]) && fabs(A[i+k][k])>EPS){
54             max = A[i+k][k];
55             max_row = i+k;
56         }
57     }
58     return max_row; // 最大値がある行数を返す
59 }
60
61 // 行を入れ替える関数
62 void change_row(Matrix A, int i, int max_row, const int N){
63     for(int j=0; j<N+1; j++){
64         double tmp = A[i][j]; // 一時的に格納する変数
65         A[i][j] = A[max_row][j];
66         A[max_row][j] = tmp;
67     }
68 }
69
70 // 前進消去
71 void forward_elimination(Matrix A, const int N){
72     for(int i=0; i<N; i++){
73         int max_row = search_max(A, i, N); // 最大値がある行数を検索
74         if(i != max_row){
75             change_row(A, i, max_row, N); // 最大値がある行数と行を入れ替え
76         }
77     }
78 }

```

```

77     double pivot = A[i][i];
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=i+1; k<N; k++){     // 掃き出し計算
83             double tmp = A[k][i];    // 一時的に値を保存する変数
84             for(int j=i; j<N+1; j++){
85                 A[k][j] -= tmp * A[i][j];
86             }
87         }
88     }
89 }
90 }
91
92 // 後退代入
93 void backward_substitution(Matrix A, const int N){
94     for(int i=N-2; i>=0; i--){
95         for(int j=N-1; j>i; j--){
96             A[i][N] -= A[i][j]*A[j][N];
97         }
98     }
99 }
100
101 // ガウス消去法
102 void gauss_elimination(Matrix A, const int N){
103     forward_elimination(A, N);       // 前進消去
104     backward_substitution(A, N);     // 後退代入
105 }
106
107 // 解を表示する関数
108 void print_ans(Matrix A, const int N){
109     printf("\n----- Answers : m = %d ----- \n",N-1);
110     for(int i=0; i<N; i++){
111         printf("a_%2d = %9.6f\n", i, A[i][N]); // 連立方程式の解が近似多項式の係数
112     }
113 }
114
115 // 最小二乗法を行う関数
116 void least_squares_method(input_data* Indata, Matrix A, int m, int datanum){
117     double s[2*M + 1] = {0.0};
118     double t[M + 1] = {0.0};
119
120     // s の計算
121     for(int i=0; i<2*m+1; i++){
122         for(int j=0; j<datanum; j++){
123             s[i] += ipow(Indata[j].x,i);
124         }
125     }
126     // t の計算
127     for(int i=0; i<m+1; i++){
128         for(int j=0; j<datanum; j++){
129             t[i] += Indata[j].y * ipow(Indata[j].x,i);
130         }
131     }
132     // 係数行列 A の作成
133     for(int i=0; i<m+1; i++){
134         A[i][m + 1] = t[i];
135         for(int j=0; j<m+1; j++){
136             A[i][j] = s[j + i];
137         }
138     }
139     // ガウス消去法
140     gauss_elimination(A, m+1);
141 }
142
143 double calc_cod(input_data* Indata, Matrix A, int m, int datanum){
144     double sst = 0.0; // 全変動の平方根
145     double sse = 0.0; // 残差変動の平方根
146     double y_m = 0.0; // yの平均
147
148     for(int i=0; i<datanum; i++){
149         y_m += Indata[i].y;
150     }
151     y_m /= datanum;
152
153     for(int i=0; i<datanum; i++){

```

```

154     double y_p = 0.0; // 近似多項式のy値
155     for(int j=0; j<m+1; j++){
156         y_p += A[j][m + 1] * ipow(Indata[i].x, j);
157     }
158     sse += ipow(Indata[i].y - y_p, 2);
159     sst += ipow(Indata[i].y - y_m, 2);
160 }
161 return 1.0 - (sse / sst); // 決定係数の算出
162 }
163
164 // 各次数における決定係数の計算結果を画面に表示する関数
165 void print_cod_calculation(int m_min, int m_max, const int datanum1, input_data* Data_mat1){
166     printf("\n----- R^2 of the calculation results <measurement data> -----\n");
167     for(int m = m_min; m<=m_max; m++){
168         Matrix A = {0.0}; // 拡大行列を定義
169         least_squares_method(Data_mat1, A, m, datanum1); // 最小二乗法によって近似多項式の係数を算出
170         double cod1 = calc_cod(Data_mat1, A, m, datanum1); // 計測データに対する 決定係数 を計算
171         printf("m =%2d\t%9.6f\n",m,cod1);
172     }
173 }
174
175 // メイン関数
176 int main(void){
177     input_data Data_mat1[data] = {0.0}; // 計測データ用の配列を定義
178
179     const char* ifile1 = "measurement_data.csv"; // 計測データのファイル名
180     const int datanum1 = scan_file(ifile1, Data_mat1); // ファイルを読み込みデータを配列に格納
181
182     const int m_min = 1; // 次数 m の最小値
183     const int m_max = 10; // 次数 m の最大値
184
185     print_cod_calculation(m_min,m_max,datanum1,Data_mat1); // 各次数における決定係数の計算結果を画面に表示する
186
187     return 0;
188 }

```