```cpp
#include <iostream>
#include <math.h>
#include <stdio.h>
using namespace std;

#define ITERACJE 1000
#define E 0.00000001
int N; // rozmiar wczytanej macierzy
long double m[101][101]; // wczytana macierz
long double solution[101]; // wzorcowa odpowiedź
long double Gauss_X[ITERACJE][101]; // odpowiedź Gaussa
long double Jacob_X[ITERACJE][101]; // odpowiedź Jacoba
long double wektor_startowy[101] = {1};
int iterG, iterJ;

void OdwrocD() {
    for (int i = 0; i < N; i++)
        m[i][i] = (1.0 / m[i][i]);
}

void PomnozPrzezD() {
    for (int i = 0; i < N; i++) {
        m[i][N] = m[i][N] * m[i][i];

        for (int ii = 0; ii < N; ii++) {
            if (i != ii)
                m[i][ii] = (m[i][ii] * m[i][i]);
        }
    }
}

void Gauss() {
    iterG = ITERACJE;

    for (int i = 1; i < ITERACJE; i++) {
        for (int w = 0; w < N; w++) {

            Gauss_X[i][w] += m[w][N];

            for (int k = 0; k < N; k++) {
                if (w == k) {
                    continue;
                }
                if (w < k) {
                    Gauss_X[i][w] -= m[w][k] * Gauss_X[i - 1][k];
                }
                if (w > k) {
                    Gauss_X[i][w] -= m[w][k] * Gauss_X[i][k];
                }
            }
        }

        long double roznica = 0;
        for (int t = 0; t < N; t++) {
            if (fabs(Gauss_X[i][t] - Gauss_X[i - 1][t]) > roznica)
                roznica = fabs(Gauss_X[i][t] - Gauss_X[i - 1][t]);
        }
        if (roznica < E) {
            iterG = i;
            return;
        }
    }
```

```
 63 }
 64
 65 void Jacob() {
 66
 67     for (int i = 0; i < N; i++) {
 68         for (int ii = 0; ii < N; ii++) {
 69             if (i == ii) {
 70                 m[i][ii] = 0;
 71             } else
 72                 m[i][ii] = -m[i][ii];
 73         }
 74     }
 75
 76     for (int i = 1; i < ITERACJE; i++) {
 77         for (int w = 0; w < N; w++) {
 78
 79             Jacob_X[i][w] += m[w][N];
 80
 81             for (int q = 0; q < N; q++) {
 82                 Jacob_X[i][w] += m[w][q] * Jacob_X[i - 1][q];
 83             }
 84         }
 85
 86         long double roznica = 0;
 87         for (int t = 0; t < N; t++) {
 88             if (fabs(Jacob_X[i][t] - Jacob_X[i - 1][t]) > roznica)
 89                 roznica = fabs(Jacob_X[i][t] - Jacob_X[i - 1][t]);
 90         }
 91         if (roznica < E) {
 92             iterJ = i;
 93             return;
 94         }
 95     }
 96 }
 97
 98 void AnalizujWyniki() {
 99     cout << iterG << endl;
100     //cout << iterJ << endl;
101     for (int q = 0; q < N; q++) {
102         {
103
104             //printf("ITER: %d: Jacob: %.25Lf\n", iterJ, Jacob_X[iterJ][q])
    ;
105             //printf("ITER: %d: Gauss: %.25Lf\n", iterG, Gauss_X[iterG][q])
    ;
106
107         }
108     }
109 }
110
111 int main() {
112     int ilosc, nic;
113     cin >> ilosc;
114
115     while (ilosc--) {
116         cin >> N;
117         for (int i = 0; i < N; i++) {
118             cin >> solution[i];
119         }
120         cin >> nic;
121         for (int i = 0; i < N; i++)
122             for (int q = 0; q < N + 1; q++) {
```

```
123                     cin >> m[i][q];
124                     if (m[i][q] == 0) {
125                         throw;
126                     }
127                 }
128 
129             for (int q = 0; q < 101; q++) {
130                 Jacob_X[0][q] = solution[q]-wektor_startowy[q];
131                 Gauss_X[0][q] = solution[q]-wektor_startowy[q];
132             }
133 
134             OdwrocD();
135             PomnozPrzezD();
136 
137             Gauss();
138             Jacob();
139 
140             AnalizujWyniki();
141         }
142         return 0;
143 }
144 
145 
```