МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра проектирования и безопасности компьютерных систем

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине

"ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА"

по теме:

«Линейные уравнения»

Выполнил: Нгуен Ле Минь
Группа: N3251
Преподаватель: Гришенцев А.Ю.

Санкт-Петербург
2021

Задание 5.1 : Метод Камера

1 : Задание

Наименование задачи: решение системы линейных уравнений методом Крамера на множестве вещественных чисел.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве вещественных чисел, методом Крамера. Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы. Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи.

2: Теория

Метод Крамера (правило Крамера) — способ решения систем линейных алгебраических уравнений с числом уравнений равным числу неизвестных с ненулевым главным определителем матрицы коэффициентов системы (причём для таких уравнений решение существует и единственно)

Пусть дана система n линейных уравнений с n неизвестными

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2n}x_n = b_2 \\ ..... \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + ... + a_{nn}x_n = b_n \end{cases}$$

Определитель матрицы системы δ, отличный от нуля, решение записывают в виде :

$$x_i = \frac{1}{\Delta}\begin{vmatrix} a_{11} & \dots & a_{1,i-1} & b_1 & a_{1,i+1} & \dots & a_{1n} \\ a_{21} & \dots & a_{2,i-1} & b_2 & a_{2,i+1} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n-1,1} & \dots & a_{n-1,i-1} & b_{n-1} & a_{n-1,i+1} & \dots & a_{n-1,n} \\ a_{n1} & \dots & a_{n,i-1} & b_n & a_{n,i+1} & \dots & a_{nn} \end{vmatrix}$$

($i-$ый столбец матрицы системы заменяется столбцом свободных членов)

В другой форме правило Крамера формулируется так: для любых коэффициентов $c_1, c_2, ..., c_n$ справедливо равенство:

$$(c_1x_1 + c_2x_2 + \dots + c_nx_n) \cdot \Delta = -\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \\ c_1 & c_2 & \dots & c_n & 0 \end{vmatrix}$$

В этой форме метод Крамера справедлив без предположения, что δ отличен от нуля, не нужно даже, чтобы коэффициенты системы были бы элементами целостного кольца (определитель системы может быть даже делителем нуля в кольце коэффициентов). Можно также считать, что либо наборы $b_1, b_2, ..., b_n$ и $x_1, x_2, ..., x_n$, либо набор $c_1, c_2, ..., c_n$ состоят не из элементов кольца коэффициентов системы, а какого-нибудь модуля над этим кольцом. В этом виде формула Крамера используется, например, при доказательстве формулы для определителя Грама и Леммы Накаямы.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 5
 */
#include <iostream>
using namespace std;

void j_swap(double** arr, int N, int li, int lj){
    for (int i = 0; i < N; i++){
        double tmp = arr[i][lj];
        arr[i][lj] = arr[i][li];
```

```
        arr[i][li] = tmp;
    }
}

void i_swap(double** arr, int N, int li, int lj){
    for (int j = 0; j < N; j++){
        double tmp = arr[lj][j];
        arr[lj][j] = arr[li][j];
        arr[li][j] = tmp;
    }
}

double det(double **arr, int N){
    int i, j, k = 0, swaps = 0;
    bool checker = true;
    double det = 1;
    while (k < N * N && checker) {
        checker = false;
        for (i = 0; i < N; i++)
            if (arr[i][i] == 0) {
                if (k / N % 2 == 0)
                    j_swap(arr, N, k % N, i);
                else
                    i_swap(arr, N, k % N, i);
                if (k % N != i) swaps++;
                k++;
                checker = true;
                break;
            }
    }
    if(k == N * N && checker) {
        return 0;
    }
    if (swaps % 2 == 1) {
        det *= -1;
    }
    for (i = 0; i < N; i++) {
        for (k = i + 1; k < N; k++)
            for (j = N - 1; j >= i; j--)
                arr[j][k] = arr[j][k] - arr[j][i] / arr[i][i] * arr[i][k];
        det = arr[i][i] * det;
    }
    return det;
}

int main(){
    int i, j, N = 0;
    cout << "Enter the number of unknowns";
    cin >> N;
    auto** arr = new double* [N];
    auto** tmparr = new double* [N];
    auto* dets = new double[N]();
    for (i = 0; i < N; i++) {
        arr[i] = new double[N];
        tmparr[i] = new double[N];
        for (j = 0; j < N; j++) {
            cout << "Please enter arr[" << i << "][" << j << "]: ";
            cin >> arr[i][j];
            tmparr[i][j] = arr[i][j];
        }
    }
```

```
    cout << "Enter the matrix " << endl;
    auto* B = new double[N]();
    for (i = 0; i < N; i++){
        cout << "B[" << i << "]:";
        cin >> B[i];
    }
    double detA = det(tmparr, N);
    if (detA == 0) {
        cout << "The determinant is zero! " << endl;
        return 1;
    }
    for (i = 0; i < N; i++){
        for (int b = 0; b < N; b++)
            for (int k = 0; k < N; k++)
                tmparr[b][k] = arr[b][k];
        for (int k = 0; k < N; k++)
            tmparr[k][i] = B[k];
        dets[i] = det(tmparr, N);
    }
    cout << "Determinant of the matrix of coefficients of the system: " << detA << endl;
    cout << "Roots are " << endl;
    for (i = 0; i < N; i++){
        cout << "det[" << i << "] " << dets[i]<< endl;
        cout << "x[" << i << "] = " << dets[i] / detA << endl;
    }
    for(i = 0; i < N; i++){
        delete [] arr[i], tmparr[i];
    }
    delete [] arr, tmparr, dets;
    return 0;
}
```

4: Вывод

```
/Users/nguyenminh/CLionProjects/lab_computational/cmake-build-debug/lab51
Enter the number of unknowns3
Please enter arr[0][0]: 2
Please enter arr[0][1]: 1
Please enter arr[0][2]: 3
Please enter arr[1][0]: 4
Please enter arr[1][1]: 5
Please enter arr[1][2]: 6
Please enter arr[2][0]: 7
Please enter arr[2][1]: 9
Please enter arr[2][2]: 11
Enter the matrix
B[0]:12
B[1]:11
B[2]:3
Determinant of the matrix of coefficients of the system: 3
Roots are
det[0] 161
x[0] = 53.6667
det[1] -13
x[1] = -4.33333
det[2] -91
x[2] = -30.3333
```

Метод Крамера позволяет находить решение систем линейных алгебраических уравнений, если определитель основной матрицы отличен от нуля. По сути метод сводится к вычислению определителей матриц порядка n на n и применению соответствующих формул для нахождения неизвестных переменных. Если число уравнений в системе велико (больше трех), метод медленно выполнить, так как целесообразно искать решение методом Гаусса.

Задание 5.2 : Метод Гаусса

1 : Задание

Наименование задачи: решение системы линейных уравнений методом Гаусса на множестве вещественных чисел. Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве вещественных чисел, методом Гаусса. Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы.

Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи. Наименование задачи: решение системы линейных уравнений методом Гаусса с ведущим элементом на множестве комплексных чисел.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве комплексных чисел, методом Гаусса с ведущим элементом. Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы.

Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи.

2: Теория

Метод Гаусса это метод перехода от исходной системы линейных уравнений (при помощи эквивалентных преобразований) к системе, которая решается проще, чем исходная система. Рассмотрим систему линейных уравнений с действительными постоянными коэффициентами:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2n}x_n = b_2 \\ ..... \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + ... + a_{nn}x_n = b_n \end{cases}$$

или в матричной форме :

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$\boldsymbol{A \cdot X = Y}$$

Метод Гаусса решения системы линейных уравнений включает в себя 2 стадии: +) последовательное (прямое) исключение;

+) обратная подстановка.

3 : Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 5
 */
#include <iostream>

using namespace std;

// ----- MethodGauss -----
double* MethodGauss(double** matrix, double*y, int n) {
    double* x, dMax;
    int index, k = 0;
```

```
    x = new double[n];
    while (k < n) {
        // search string with max element a[i][k]
        dMax = abs(matrix[k][k]);
        index = k;
        for (int i = k; i < n; i++) {
            if (abs(matrix[i][k]) > dMax) {
                index = i;
                dMax = matrix[i][k];
            }
        }

        // swap 2 strings
        for (int j = 0; j < n; j++) {
            double tmp = matrix[k][j];
            matrix[k][j] = matrix[index][j];
            matrix[index][j] = tmp;
        }
        double tmpY = y[k];
        y[k] = y[index];
        y[index] = tmpY;

        // normalized equations
        for (int i = k; i < n; i++) {
            double normFactor = matrix[i][k];
            if (normFactor == 0) continue;
            for (int j = 0; j < n; j++) {
                matrix[i][j] = matrix[i][j] / normFactor;
            }
            y[i] = y[i] / normFactor;

            if (i == k) continue; // don't subtract the equation from itself

            for (int j = 0; j < n; j++) {
                matrix[i][j] = matrix[i][j] - matrix[k][j];
            }
            y[i] = y[i] - y[k];
        }
        k++;
    }

    // reverse motion
    for (k = n - 1; k >= 0; k--) {
        x[k] = y[k]; // last equation
        for (int i = 0; i < n; i++) {
            y[i] = y[i] - matrix[i][k] * x[k]; // matrix[i][k] - normFactor on this step
        }
    }

    return x;
}

// ----- PrintSystem -----
void PrintSystem(double** matrix, double* y, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << matrix[i][j] << "*x" << j;
            if (j < n - 1) cout << "+";
        }
        cout << " = " << y[i] << '\n';
    }
```

```
}


int main()
{
    int n; // count of equation
    double** matrixA, * vectorX, * vectorY;

    cout << "Please enter n: ";
    cin >> n;
    matrixA = new double*[n];
    vectorY = new double[n];
    // enter elements of matrix A
    for (int i = 0; i < n; i++) {
        matrixA[i] = new double[n];
        for (int j = 0; j < n; j++) {
            cout << "a[" << i << "][" << j << "] = ";
            cin >> matrixA[i][j];
        }
    }
    // enter elements of right side (vector Y)
    for (int k = 0; k < n; k++) {
        cout << "y[" << k << "] = ";
        cin >> vectorY[k];
    }

    PrintSystem(matrixA, vectorY, n);
    vectorX = MethodGauss(matrixA, vectorY, n);

    // print result - vector X
    for (int k = 0; k < n; k++) {
        cout << "x[" << k << "] = " << vectorX[k] << endl;
    }

    return 0;
}
```

4: Вывод

```
/Users/nguyenminh/CLionProjects/lab_computational/cmake-build-debug/lab52
Write count of equation: 3
a[0][0] = 3
a[0][1] = 4
a[0][2] = 5
a[1][0] = 6
a[1][1] = 7
a[1][2] = 9
a[2][0] = 11
a[2][1] = 12
a[2][2] = 15
y[0] = 16
y[1] = 2
y[2] = -9
3*x0+4*x1+5*x2 = 16
6*x0+7*x1+9*x2 = 2
11*x0+12*x1+15*x2 = -9
x[0] = -28.5
x[1] = 48.5
x[2] = -18.5
```

При выполнении лабораторной работы, я научился реализовывать программу решение системы линейных уравнений, на множестве вещественных чисел, методом Гаусса.

Задание 5.3 : Схема Халецкого

1: Задание

Наименование задачи: решение системы линейных уравнений с помощью метода схема Халецкого.
Вид решения: программа и отчёт.
Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве вещественных чисел, методом схема Халецкого.

Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы. Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи.

2: Теория

Для удобства рассуждений систему линейных уравнений запишем в матричном виде

$$A\bar{x} = \bar{b}, \qquad (1)$$

где $A = [a_{ij}]$ - квадратная матрица порядка n и

$$\bar{x} = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}, \ \bar{b} = \begin{bmatrix} a_{1,n+1} \\ \dots \\ a_{n,n+1} \end{bmatrix} \text{ - векторы-столбцы.}$$

Представим матрицы A в виде произведения нижней треугольной матрицы $B = [b_{ij}]$ и верхней треугольной матрицы $C = [c_{ij}]$ с единичной диагональю, т.е.

$$A = B \cdot C, \qquad (2)$$

где

$$B = \begin{bmatrix} b_{11} & 0 & \dots & 0 \\ b_{21} & b_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{bmatrix}, \ C = \begin{bmatrix} 1 & c_{12} & \dots & c_{1n} \\ 0 & 1 & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}.$$

Тогда элементы b_{ij} и c_{ij} определяются по формулам:

$$\begin{cases} b_{i1} = a_{i1} \\ b_{ij} = a_{ij} - \sum_{k=1}^{j-1} b_{ik} c_{kj} \end{cases}, \quad (1 < j \le i) \qquad (3)$$

и

$$\begin{cases} c_{1j} = \dfrac{a_{1j}}{b_{11}} \\ c_{ij} = \dfrac{1}{b_{ii}} (a_{ij} - \sum_{k=1}^{i-1} b_{ik} c_{kj}) \end{cases}, \quad (1 < i < j) \qquad (4)$$

Отсюда искомый вектор $\bar{x}$ может быть вычислен из цепи уравнений

$$B\bar{y} = \bar{b}, \quad C\bar{x} = \bar{y}. \qquad (5)$$

Так как матрицы B и C - треугольные, то системы (5) легко решаются, а именно:

$$\begin{cases} y_1 = \dfrac{a_{1,n+1}}{b_{11}} \\ y_i = \dfrac{1}{b_{ii}} (a_{i,n+1} - \sum_{k=1}^{i-1} b_{ik} y_k) \end{cases}, \quad (i > 1) \qquad (6)$$

и

$$\begin{cases} x_n = y_n \\ x_i = y_i - \sum_{k=i+1}^{n} c_{ik} x_k \end{cases}, \quad (i < n) \qquad (7)$$

Из формул (6) видно, что числа y_i выгодно вычислять вместе с коэффициентами c_{ij}. Этот метод получил название схемы Халецкого.
Заметим, что если матрица A симметрическая, т.е. $a_{ij} = a_{ji}$, то

$$c_{ij} = \frac{b_{ij}}{b_{ii}}, \quad (i < j).$$

Схема Халецкого удобна для работы на вычислительных машинах, т.к. в этом случае операции "накопления"(3) и (4) можно проводить без записи промежуточных результатов.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 5
 */
#include <iostream>
#include <vector>
#include <cmath>
#include <string>
using namespace std;

typedef vector<vector<double>> real_matrix;
void input_matrix(real_matrix &arr, int n) {
    cout << "Enter matrix" << endl; arr.resize(n);
    for (int i = 0; i < n; i++) {
        cout << "Enter coefficients of equation " << to_string(i) << endl;
        arr[i].resize(n);
        for (int j = 0; j < n; j++) cin >> arr[i][j];
    }
}
void transpose(real_matrix & from, real_matrix & to, int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            to[j][i] = from[i][j];
}
void output_matrix(real_matrix & a) {
    for (auto & i : a) {
        for (double j : i) cout << j << '\t';
        cout << endl;
    } cout << endl;
}
vector<double> choletsky_method(real_matrix a, vector<double> b, int n) {
    double pk, sum;
    real_matrix L(n, vector<double>(n, 0.0));
    real_matrix T(n, vector<double>(n, 0.0));
    vector<double> x(n), y(n);
    L[0][0] = sqrt(a[0][0]);
    for (int i = 1; i < n; i++) L[i][0] = a[i][0] / L[0][0];
    for (int i = 1; i < n; i++) {
        sum = 0;
        for (int k = 0; k < i; k++) sum += L[i][k] * L[i][k];
        pk = a[i][i] - sum; L[i][i] = sqrt(pk);
        for (int j = i + 1; j < n; j++) {
            sum = 0;
            for (int k = 0; k < i; k++) sum += L[j][k] * L[i][k];
            pk = a[j][i] - sum; L[j][i] = pk / L[i][i];
        }
    }
    transpose(L, T, n);
    cout << "Matrix L: " << endl;
    output_matrix(L);
    cout << "Matrix T: " << endl;
    output_matrix(T);
    // Решение L*y=b
    y[0] = b[0] / L[0][0];
    for (int i = 1; i < n; i++) {
```

```
        sum = 0;
        for (int k = 0; k < i; k++) sum += L[i][k] * y[k];
        pk = b[i] - sum; y[i] = pk / L[i][i];
    }
    // Решение T*x=y
    x[n - 1] = y[n - 1] / T[n - 1][n - 1];
    for (int i = n - 2; i >= 0; i--) {
        sum = 0;
        for (int k = i + 1; k < n; k++) sum += T[i][k] * x[k];
        pk = y[i] - sum; x[i] = pk / T[i][i];
    }
    return x;
}

void solver(){
    int n;
    cout << "Input number of equations in system: " << endl;
    cin >> n;
    real_matrix a;
    input_matrix(a, n);
    vector<double> b; b.resize(n);
    cout << "Input constant terms: " << endl;
    for (int i = 0; i < n; i++) cin >> b[i];
    vector<double> x = choletsky_method(a, b, n);
    for (int i = 0; i < x.size(); i++){
        cout << "x" << to_string(i) << " = " << x[i] << endl;
    }
}

int main() {
    solver();
    return 0;
}
```

4:Вывод

```
/Users/nguyenminh/CLionProjects/lab_computational/cmake-build-debug/lab53
Input number of equations in system:
3
Enter matrix
Enter coefficients of equation 0
6 7 8
Enter coefficients of equation 1
11 12 36
Enter coefficients of equation 2
23 40 105
Input constant terms:
3 9 12
Matrix L:
2.44949 0   0
4.49073 nan 0
9.38971 nan nan

Matrix T:
2.44949 4.49073 9.38971
0   nan nan
0   0   nan

x0 = nan
x1 = nan
x2 = nan
```

Схема Халецкого может использоваться как эффективный метод решения систем линейных уравнений, однако только для узкого их круга ввиду ограничений, налагаемых методом на матрицу коэффициентов.

Задание 5.4 : Метод итераций

1: Задание

Наименование задачи: решение системы линейных уравнений с помощью метода итераций.
Вид решения: программа и отчёт.
Реализация решения: язык C или C++.
Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве вещественных чисел, методом итераций.
Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы. Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи.

2: Теория

Заметим, что систему линейных уравнений
$Ax = f$ (1)
можно преобразовать к такому виду :
$x = (E - \tau * A)x + \tau * f$ (2)
причем новое уравнение (2) равносильно исходному при любом значении t. Вообще, (1) многими способами можно заметить равносильной системой вида :
$x = Bx + \phi, x \in R^m, \phi \in R^m$ (3)
частным случаем которой является (2)
Итерационная схема при заданном произвольно x_0 имеет следующий вид
$x_{p+1} = (E - \tau A)x_p + \tau f$,$p = 0, 1, 2, ...$
при заданном произвольно x_0

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 5
 */
#include <iostream>
using namespace std;

class Matr{
private:
    int maxCounts = 20000;
    int n{};
    double **masA;
    double *masB;
public:
    // constructor, requires input of dimension,
    // matrices of coefficients of equations and
    // matrices of free elements
    Matr(){
        cout << "Enter the dimension:  ";
        cin >> n;
        masB = new double[n];
        masA = new double*[n];
        for(int i = 0; i < n; i++){
            masA[i] = new double[n];
            for(int j = 0; j < n; j++){
                cout << "Please enter a[" << i << "][" << j << "]: ";
                cin >> masA[i][j];
            }
        }
        for(int i = 0; i < n; i++){
            cout << "Please enter b[" << i << "]: ";
```

```
            cin >> masB[i];
        }
    }
    // the destructor cleans up memory from
    // matrices of free members
    // and matrices of coefficients of equations
    ~Matr(){
        for(int i = 0; i < n; i++)
            delete [] masA[i];
        delete [] masA, masB;
    }
    // iteration method for solving the СЛАУ
    void Itera(){
        auto *currentVariableValues = new double[n];
        auto *previousVariableValues = new double[n]();
        int counter = 0;
        do{
            // Calculate the values of the unknowns at the current iteration
            // according to theoretical formulas
            for (int i = 0; i < n; i++){
                // Initialize the i-th unknown value
                // free member of the i-th row of the matrix
                currentVariableValues[i] = masB[i];
                // Subtract the sum for all unknowns different from the i-th
                for (int j = 0; j < n; j++)
                    if (i != j)
                        currentVariableValues[i] -= masA[i][j] * previousVariableValues[j];
                // Divide by the coefficient for the i-th unknown
                currentVariableValues[i] /= masA[i][i];
            }
            previousVariableValues = currentVariableValues;
            counter++;
        }while (counter < maxCounts);
        for (int i = 0; i < n; i++)
            cout << "x[" << i << "] = " << previousVariableValues[i] << endl;
        delete [] currentVariableValues, previousVariableValues;
    }
};

void solver(){
    Matr *res = new Matr;
    res->Itera();
    delete res;
}

int main(){
    solver();
    return 0;
}
```

4: Вывод

```
/Users/nguyenminh/CLionProjects/lab_computational/cmake-build-debug/lab54
Enter the dimension:  2
Please enter a[0][0]: 5
Please enter a[0][1]: 3
Please enter a[1][0]: 6
Please enter a[1][1]: 7
Please enter b[0]: -1
Please enter b[1]: -6
x[0] = 0.647059
x[1] = -1.41176
```

При выполнении мы научились программными методами, с помощью языка программирования C++,находить решение системы уравнений с помощью метода итераций.

Задание 5.5: Метод Зейделя

1: Задание

Наименование задачи: решение системы линейных уравнений с помощью метода Зейделя.
Вид решения: программа и отчёт.
Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: решение системы линейных уравнений, на множестве вещественных чисел, методом Зейделя. Предусмотреть возможность решения уравнений различного порядка без перекомпиляции программы. Исходные данные (матрицу коэффициентов уравнения, вектор правых частей), вводить в программу из консоли или из файла. Оценить вычислительную сложность решения задачи.

2: Теория

Метод Зейделя представляет собой некоторую модификацию метода итераций. Основная его идея заключается в том, что при вычислении $(k+1)$-го приближения неизвестной x_i учитываются уже вычисленные ранее $(k+1)$-е приближения неизвестных $x_1, x_2, \ldots, x_{i-1}$.

Пусть получена эквивалентная система :

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Выберем произвольно начальные приближения корней $x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \ldots, x_n^{(0)}$. Далее, предполагая, что k-ые приближения $x_n^{(k)}$ корней известны, согласно Зейделю будем строить $(k+1)$-е приближения корней по формулам:

$$x_1^{(k+1)} = \beta_1 + \alpha_{12}x_2^{(k)} + \alpha_{13}x_3^{(k)} + \ldots + \alpha_{1n}x_n^{(k)},$$
$$x_2^{(k+1)} = \beta_2 + \alpha_{21}x_1^{(k+1)} + \alpha_{23}x_2^{(k)} + \ldots + \alpha_{2n}x_n^{(k)},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_n^{(k+1)} = \beta_n + \alpha_{n1}x_1^{(k+1)} + \alpha_{n2}x_2^{(k+1)} + \ldots + \alpha_{nn}x_n^{(k)} \quad (k = 0, 1, 2, \ldots).$$

Заметим, что указанные выше условия сходимости для простой итерации остается верной для итерации по методу Зейделя. Обычно метод Зейделя дает лучшую сходимость, чем метод простой итерации, но приводит к более громоздким вычислениям.

3: Код

```cpp
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 5
 */
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

void solver(){
    int size;
    cout << "Enter the number of equations:  \n";
    cin >> size;
    // We will store the matrix in a vector consisting of
    // vectors of real numbers
    vector <vector <long double> > matrix;
    // The matrix will have a size (size) x (size + 1),
    // taking into account the column of free members
    matrix.resize(size);
    cout << "Enter matrix element by element \n";
    for (int i = 0; i < size; i++){
```

```
        matrix[i].resize(size + 1);
        for (int j = 0; j < size + 1; j++){
            cout << "a[" << i << "][" << j << "]: ";
            cin >> matrix[i][j];
        }
    }
    cout << "Enter accuracy\n";
    // Read the required accuracy of the solution
    long double eps;
    cin >> eps;
    // Enter vector of values of unknowns at the previous iteration,
    // whose size is equal to the number of rows in the matrix, i.e. size,
    // and according to the method, we initially fill it with zeros
    vector <long double> previousVariableValues(size, 0.0);
    // Let's iterate until
    // until the required precision is achieved
    while (true){
        // Enter vector of values of unknowns at the current step
        vector <long double> currentVariableValues(size);
    // Calculate the values of the unknowns at the current iteration
        // according to theoretical formulas
        for (int i = 0; i < size; i++){
            // Initialize the i-th unknown value
            // free element of the i-th row of the matrix
            currentVariableValues[i] = matrix[i][size];
            // Subtract the sum for all unknowns different from the i-th
            for (int j = 0; j < size; j++){
                // For j <i we can use the already calculated
                // at this iteration, the values of the unknowns
                if (j < i)
                    currentVariableValues[i] -= matrix[i][j] * currentVariableValues[j];
                // For j> i, use the values from the previous iteration
                if (j > i)
                    currentVariableValues[i] -= matrix[i][j] * previousVariableValues[j];
            }
            // Divide by the coefficient for the i-th unknown
            currentVariableValues[i] /= matrix[i][i];
        }
        // Calculate the current error relative to the previous iteration
        long double error = 0.0;
        for (int i = 0; i < size; i++)
            error += abs(currentVariableValues[i] - previousVariableValues[i]);

        // If the required accuracy is achieved, then we end the process
        if (error < eps)
            break;
        // Go to the next iteration, so
        // that the current values are unknown
        // become the values at the previous iteration
        previousVariableValues = currentVariableValues;
    }

    // Display the found values of the unknowns with 8 digits of precision
    for (int i = 0; i < size; i++)
        printf("x[%d]: %.8Lf ", i, previousVariableValues[i]);
}

int main(){
    solver();
    return 0;
}
```

4: Вывод программы

Enter the number of equations
3
Enter matrix element by element
a[0][0]: 10
a[0][1]: -3
a[0][2]: 2
a[0][3]: 10
a[1][0]: 3
a[1][1]: -10
a[1][2]: -2
a[1][3]: -23
a[2][0]: 2
a[2][1]: -3
a[2][2]: 10
a[2][3]: 26
Enter accuracy
0.05
x[0]: 1.00748800 x[1]: 2.00757440 x[2]: 3.00077472

В этом задании мы научились программными методами, с помощью языка программирования C++,находить решение системы уравнений с помощью метода Зейделя.