Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1
Задание №1 по курсу «Численные методы»

Студент:	С. М. Власова
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Задание №1.1

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.
Вариант: 12

$$\begin{cases} -x_1 - 8 \cdot x_2 + 5 \cdot x_4 = -60, \\ 6 \cdot x_1 - 6 \cdot x_2 + 2 \cdot x_3 + 4 \cdot x_4 = -10, \\ 9 \cdot x_1 - 5 \cdot x_2 - 6 \cdot x_3 + 4 \cdot x_4 = 65, \\ -5 \cdot x_1 - 9 \cdot x_3 + x_4 = 18 \end{cases}$$

1 Описание метода решения

LU – разложение матрицы A представляет собой разложение матрицы A в произведение нижней и верхней треугольных матриц, т.е.

$$A = L \cdot U,$$

где L — нижняя треугольная матрица (матрица, у которой все элементы, находящиеся выше главное диагонали, раны нулю: $l_{ij} = 0, \quad i < j$), U — верхняя треугольная матрица (матрица, у которой все элементы, находящиеся ниже главной диагонали, равны нулю: $u_{ij} = 0, \quad i > j$).
$LU-$разложение может быть построено с использованием метода Гаусса. Рассмотрим $k-$й шаг метода Гаусса, на котором осуществляется обнуление поддиагональных элементов $k-$го столбца матрицы $A^{(k-1)}$. С этой целью используется следующая операция:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \mu_i^{(k)} \cdot a_{kj}^{(k-1)}, \qquad \mu_i^{(k)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \quad i = \overline{k+1, n}, j = \overline{k, n}.$$

В терминах матричных операций такая операция эквивалентна умножению $A^{(k)} = M_k \cdot A^{(k-1)}$, где элементы матрицы M_k определяются следующим образом:

$$m_{ij}^{(k)} = \begin{cases} 1, & i = j \\ 0, & i \neq j, j \neq k \\ -\mu_{k+1}^{(k)}, & i \neq j, j = k \end{cases}$$

Т.е. матрица M_k имеет вид $\begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\mu_{k+1}^{(k)} & 1 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & -\mu_n^{(k)} & 0 & 0 & 0 & 1 \end{pmatrix}$

При этом выражение для обратной матрицы запишется в виде $A^{(k-1)} = M_k^{-1} \cdot A^{(k)}$, где

$$M_k^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{k+1}^{(k)} & 1 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \mu_n^{(k)} & 0 & 0 & 0 & 1 \end{pmatrix}$$

В результате прямого хода метода Гаусса получим $A^{(n-1)} = U$,

$$A = A^{(0)} = M_1^{-1} \cdot A^{(1)} = M_1^{-1} \cdot M_2^{-1} \cdot A^{(2)} = M_1^{-1} \cdot M_2^{-1} \cdot \ldots \cdot M_{n-1}^{-1} \cdot A^{(n-1)},$$

где $A^{(n-1)} = U$ — верхняя треугольная матрица, а $L = M_1^{-1} \cdot M_2^{-1} \cdot \ldots \cdot M_{n-1}^{-1}$ — нижняя треугольная матрица, имеющая вид

$$L = \begin{pmatrix} 1 & 0 & \ldots & 0 & 0 & 0 & 0 \\ \mu_2^{(1)} & 1 & \ldots & 0 & 0 & 0 & 0 \\ \mu_3^{(1)} & \mu_3^{(2)} & 1 & 0 & 0 & 0 & 0 \\ \ldots & \ldots & \ldots \mu_{k+1}^{(k)} & 1 & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \mu_n^{(1)} & \mu_n^{(2)} & \mu_n^{(k)} & \mu_2^{(k+1)} & \ldots & \mu_n^{(n-1)} & \mu_n^{(n)} \end{pmatrix}$$

Таким образом, искомое разложение $A = L \cdot U$ получено.
В дальнейшем $LU-$разложение может быть эффективно использовано при решении систем линейных алгебраических уравнений вида $A \cdot x = b$. Действительно, подставляя $LU-$разложение в СЛАУ, получим $L \cdot U \cdot x = b$, или $U \cdot x = L^{-1} \cdot b$. Т.е. процесс решения СЛАУ сводится к двум простым этапам.
На первом этапе решается СЛАУ $L \cdot z = b$. Поскольку матрица системы — нижняя треугольная, решение можно записать в явном виде:

$$z_1 = b_1, \quad z_i = b_i - \sum_{j=1}^{i-1} l_{ij} \cdot z_j, \quad i = \overline{2, n}.$$

На втором этапе решается СЛАУ $U \cdot x = z$ с верхней треугольной матрицей. Здесь, как и на предыдущем этапе, решение представляется в явном виде:

$$x_n = \frac{z_n}{u_{nn}}, \quad x_i = \frac{1}{u_{ii}} \cdot (z_i - \sum_{j=i+1}^{n} u_{ij} \cdot x_j), \quad i = \overline{n-1, 1}.$$

Отметим, что второй этап эквивалентен обратному ходу методу Гаусса, тогда как первый соответствует преобразованию правой части СЛАУ в процессе прямого хода.

2 Протокол

Входные данные я храню в файле *data1*: первая строка — размерность матрицы, на следующих строках — матрица.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data1
4
-1 -8 0 5
6 -6 2 4
9 -5 -6 4
-5 0 -9 1
-60 -10 65 18
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Выходные данные я записываю в файл *res1*.
Скриншот консоли:

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ 1.1.cpp -o 1.1
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.1 < data1 > res1
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res1
-1 -8 0 5 -60
6 -6 2 4 -10
9 -5 -6 4 65
-5 0 -9 1 18
LU-разложение:
Нижняя треугольная матрица L:
1 0 0 0
-6 1 0 0
-9 1.42593 1 0
5 -0.740741 0.849372 1

Верхняя треугольная матрица U:
-1 -8 0 5
0 -54 2 34
0 0 -8.85185 0.518519
0 0 0 0.74477

Решение системы с помощью LU-разложения:
x_1 = 7
x_2 = 6
x_3 = -6
x_4 = -1
Детерминант: -355

Обратная матрица:
-0.0280899 -0.0421348 0.0955056 -0.0730337
-0.904494 1.64326 -0.724719 0.848315
-0.123596 0.314607 -0.179775 0.0786517
-1.25281 2.62079 -1.14045 1.3427
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Результат работы программы следующий:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -6 & 1 & 0 & 0 \\ -9 & 1.42593 & 1 & 0 \\ 5 & -0.740741 & 0.849372 & 1 \end{pmatrix}; \quad U = \begin{pmatrix} -1 & -8 & 0 & 5 \\ 0 & -54 & 2 & 34 \\ 0 & 0 & -8.85185 & 0.518519 \\ 0 & 0 & 0 & 0.74477 \end{pmatrix};$$

Решение СЛАУ:

$$x_1 = 7; \quad x_2 = 6; \quad x_3 = -6; \quad x_4 = -1;$$

Определитель матрицы $A : detA = -355;$
Обратная матрица:

$$A^{-1} = \begin{pmatrix} -0.0280899 & -0.0421348 & 0.0955056 & -0.0730337 \\ -0.904494 & 1.64326 & -0.724719 & 0.848315 \\ -0.123596 & 0.314607 & -0.179775 & 0.0786517 \\ -1.25281 & 2.62079 & -1.14045 & 1.3427 \end{pmatrix}$$

3 Исходный код

Листинг 1: LU-метод

```
#include <iostream>
#include <vector>

void PrintMatrix(std::vector<std::vector<double>>& Matrix)
{
    int n = Matrix.size();
    int m = Matrix[0].size();
    for( int i = 0; i < n; i++)
    {
        for( int j = 0; j < m; j++)
            std::cout << Matrix[i][j] << " ";
        std::cout << std::endl;
    }
}

void SwapStrings(std::vector<std::vector<double>>& Matrix, int i, int k)
{
    int n = Matrix.size();
    std::vector<double> H = Matrix[i];
    Matrix[i] = Matrix[k];
    Matrix[i] = H;
}

void ChangeWithMax(std::vector<std::vector<double>>& Matrix, int i)
{
    int n = Matrix.size();
    double max = Matrix[i][i]*Matrix[i][i];
    int max_id = i;
    for(int j = i + 1; j < n; j++)
    {
        if( Matrix[j][i]*Matrix[j][i] > max)
        {
            max = Matrix[j][i]*Matrix[j][i];
            max_id = j;
        }
    }
    if(max_id != i)
        SwapStrings(Matrix, i, max_id);
}
```

```
void FindX(std::vector<std::vector<double>>& Matrix, std::vector<std::vector<double>>& x,
    std::vector<std::vector<double>>& b)
{
    int n = Matrix.size();
    int m = x[0].size();
    int j;

    for(int k = 0; k < m; k++)
    {
        for(int i = 0; i < n; i++) //Ly = b ---> y
        {
            j = i - 1;
            while(j >= 0)
            {
                b[i][k] -= Matrix[i][j] * b[j][k];
                j--;
            }
        }
    }
    for(int k = 0; k < m; k++)
    {
        for( int i = n - 1; i >= 0; i--) //Ux = b
        {
            j = i + 1;
            while(j < n){
                b[i][k] -= Matrix[i][j] * x[j][k];
                j++;
            }
            x[i][k] = b[i][k] / Matrix[i][i];
        }
    }
}

void LU(std::vector<std::vector<double>>& Matrix)
{
    int n = Matrix.size();
    int main_id = 0;
    PrintMatrix(Matrix);
    ChangeWithMax(Matrix, 0);

    for( int j = 1; j < n; j++) // i = 0
        Matrix[j][0] = Matrix[j][0]/Matrix[0][0];
    for( int i = 1; i < n; i++) // i = 1..n
```

```
    {
        for( int j = i; j < n; j++) // u[i][j]
        {
            for( int k = 0; k < i; k++)
                Matrix[i][j] -= Matrix[i][k]*Matrix[k][j];
        }
        ChangeWithMax(Matrix, i);
        for( int j = i + 1; j < n; j++) //l[j][i]
        {
            for( int k = 0; k < i; k++)
                Matrix[j][i] -= Matrix[j][k] * Matrix[k][i];
            Matrix[j][i] /= Matrix[i][i];
        }
    }
}

double Determinant(std::vector<std::vector<double>>& Matrix)
{
    int det = 1;
    for( int i = 0; i < Matrix.size(); i++)
        det *= Matrix[i][i];
    return det;
}

void L(std::vector<std::vector<double>>& Matrix)
{
    int n = Matrix.size();
    int count = 0;
    for( int i = 0; i < n; i++)
    {
        count = i;
        for( int j = 0; j < n; j++)
        {
            if( count > 0)
            {
                std::cout << Matrix[i][j] << " ";
                count--;
            }
            else if( count == 0)
            {
                std::cout << "1 ";
                count--;
            }
            else
```

```
                std::cout << "0 ";
        }
        std::cout << std::endl;
    }
}

void U(std::vector<std::vector<double>>& Matrix)
{
    int n = Matrix.size();
    int count = 0;
    for( int i = 0; i < n; i++)
    {
        count = i;
        for( int j = 0; j < n; j++)
        {
            if( count > 0)
            {
                std::cout << "0 ";
                count--;
            }
            else
                std::cout << Matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

int main()
{
    int n;
    double element;
    std::cin >> n;
    std::vector<std::vector<double>> A(n, std::vector<double>(n + 1));
    std::vector<std::vector<double>> x(n, std::vector<double>(1));
    std::vector<std::vector<double>> b(n, std::vector<double>(1));
    std::vector<std::vector<double>> ReversedA(n, std::vector<double>(n, 0));
    std::vector<std::vector<double>> ZeroMatrix(n, std::vector<double>(n ,0));

    for( int i = 0; i < n; i++)
        for( int j = 0; j < n; j++)
        {
            std::cin >> A[i][j];
            if( i == j)
                ZeroMatrix[i][j] = 1;
```

```
        }
    for(int i = 0; i < n; i++ )
        std::cin >> A[i][n];
    LU(A);
    std::cout << "LU-decomposition:" << std::endl;
    std::cout << "L:" << std::endl;
    L(A);
    std::cout << std::endl;
    std::cout << "U:" << std::endl;
    U(A);
    std::cout << std::endl;
    std::cout << "The solution of the SLAE: " << std::endl;
    for( int i = 0; i < n; i++)
        b[i][0] = A[i][n];
    FindX(A, x, b);
    for(int i = 0; i < n; i++)
        std::cout << "x_" << i + 1 << " = " << x[i][0] << std::endl;
    std::cout << "Determinant: " << Determinant(A) << std::endl;
    std::cout << std::endl;
    FindX(A, ReversedA, ZeroMatrix);
    std::cout << "Reversed matrix: " << std::endl;
    PrintMatrix(ReversedA);

    return 0;
}
```

4 Выводы

Выполнив первое задание первой лабораторной работы, я познакомилась с LU-разложением, которое эффективно используется для поиска решений СЛАУ. Этот метод является модификацией метода Гаусса. Сначала исходная матрица СЛАУ представляется в виде произведения двух матриц — нижней треугольной и верхней треугольной. Далее, когда разложение найдено, последовательно решаются две системы уравнений. Матрица первой системы — нижняя треугольная (U), второй системе соответствует верхняя треугольная матрица (L) — ее решение, эквивалентное обратному ходу Гаусса, и будет решением нашей СЛАУ.