Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра Систем Штучного Інтелекту

# Звіт

до лабораторної роботи № 2

з дисципліни

Чисельні методи

на тему:

“Прямі методи розв’язування СЛАР.”

## Варіант №24(LU)


Виконав: студент КН-217

**Ратушняк Денис**

Прийняла: доцент каф. СШІ

Мочурад Л. І.


Львів – 2022

**Мета роботи:** Набути навиків практичного використання прямих методів розв’язування СЛАР: методу LU-розкладу, методу квадратних коренів, методу ортогоналізації та методу поворотів.

**Завдання.** Складіть програму, яка методом **LU-розкладу(24 варіант)**, якщо Ваш номер у списку в журналі викладача кратний 4; або методом квадратного корення, якщо при ділені Вашого номера в журналі викладача на 4 отримується остача 1; або методом ортогоналізації, якщо остача 2; методом поворотів, якщо остача 3 – знаходить розв’язок системи лінійних рівнянь. Представте детально документовану програму з результатами знаходження розв’язків таких лінійних систем 3 і 4 порядків:

$$\begin{cases} 20x_1 - 4x_2 - 3x_3 + 8x_4 = 2, \\ -4x_1 - 26x_2 - 4x_3 + 2x_4 = -12, \\ -3x_1 - 4x_2 + 20x_3 + 2x_4 = -4, \\ 8x_1 + 2x_2 + 2x_3 - 26x_4 = 4, \end{cases}$$

$$\begin{cases} 13x_1 - 6x_2 + 2x_3 = 1, \\ -6x_1 + 22x_2 + 4x_3 = 3, \\ 2x_1 + 4x_2 - 14x_3 = 3, \end{cases}$$

$$\begin{cases} 13x_1 - 3x_2 + 4x_3 = 14, \\ -3x_1 - 4x_2 = -7, \\ 4x_1 - 5x_3 = -1. \end{cases}$$

## Програмна реалізація на мові програмування C++

```
#include <bits/stdc++.h>
using namespace std;
typedef long double ld;
typedef long long ll;
typedef vector< vector<ld > > matrix;
const ld eps = 1e-10;

ifstream tests("input.txt");
ofstream answers("output.txt");

void print(vector<ld> &A, ll cnt = 3)
{
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < A.size(); ++i) cout << A[i] << " ";
    cout << endl;
    cout << fixed << setprecision(0);
}

void print(vector<ld> &A, ofstream &answers, ll cnt = 3)
{
    answers << fixed << setprecision(cnt);
    for(int i = 0; i < A.size(); ++i) answers << A[i] << " ";
    answers << "\n\n";
    answers << fixed << setprecision(0);
}

void print(matrix &A, ofstream &answers, ll cnt=3)
{
    ll n = A.size();
```

```
    answers << fixed << setprecision(cnt);
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < A[i].size(); ++j) answers << A[i][j] << " ";
        answers << "\n";
    }
    answers << fixed << setprecision(0);
    answers << "\n";
}

void print(matrix &A, ll cnt=3)
{
    ll n = A.size();
    cout << fixed << setprecision(cnt);
    for(int i = 0; i < n; ++i)
    {
        for(int j = 0; j < A[i].size(); ++j) cout << A[i][j] << " ";
        cout << endl;
    }
    cout << fixed << setprecision(0);
    cout << endl;
}

matrix operator * (const matrix &A, const matrix &B)
{
    ll n = A.size();
    matrix res(n, vector<ld>(n, 0.0));
    for(int k = 0; k < n; ++k)
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                res[i][j] += A[i][k] * B[k][j];
    return res;
}

bool is_determinant_zero(matrix A)
{
    ll n = A.size();
    matrix AE;
    AE.resize(n);

    for(int i = 0; i < n; ++i)
        for(int j = 0; j < n; ++ j)
            AE[i].emplace_back(A[i][j]);

    for(int k = 0; k < n; ++k)
    {
        int row_max_el = k;

        /// finding maximum element
        for(int i = k + 1; i < n; ++i)
            if(fabs(AE[i][k]) > fabs(AE[row_max_el][k])) row_max_el = i;

        /// if maximum element goes to 0 -> inverse matrix does not exist
        if(fabs(AE[row_max_el][k]) < eps) return true;


        /// change of 2 rows for optimization of division by the maximum element
        if(k != row_max_el)
            for(int j = k; j < n; ++j) swap(AE[k][j], AE[row_max_el][j]);


        /// forming zeros under main diagonal
```

```
        for(int i = k + 1; i < n; ++i)
        {
            ld mik = -AE[i][k]/AE[k][k];

            AE[i][k] = 0;
            for(int j = k + 1; j < n; ++j) AE[i][j] += mik * AE[k][j];
        }
    }

    for(int i = 0; i < n; ++i)
    {
        if(fabs(AE[i][i]) < eps) return true;
    }
    return false;

}

matrix get_random_matrix(ll size_)
{
    ///randomize due to clock
    srand(clock());
    matrix res;
    res.resize(size_);
    for(int i = 0; i < size_; ++i)
    {
        res[i].resize(size_);
        for(int j = 0; j < size_; ++j) res[i][j] = rand() - rand()/2;
    }
    return res;
}


bool check_valid(matrix &A, ll &num)
{
    matrix B;
    ll n = A.size();
    B.resize(1);
    B[0].resize(1);
    B[0][0] = A[0][0];
    vector<ld> add(1);
    num = 0;
    if(fabs(A[0][0]) < eps) return false;
    for(int i = 1; i < n; ++i)
    {
        for(int num = 0; num < i; ++num) B[num].emplace_back(A[num][i]);
        for(int j = 0; j < i; ++j) add[j] = A[i][j];
        add.push_back(A[i][i]);
        B.push_back(add);
        if(is_determinant_zero(B)) {
            num = i;
            return false;
        }
    }
    return true;
}

void decompose_LU(matrix &A, matrix &L, matrix &U)
{
    ll n = A.size();
    vector<ld> null(n, 0);
    L.resize(n, vector<ld>(n, 0));
    U.resize(n, vector<ld>(n, 0));
```

```
    for(int s = 0; s < n; s++)
    {
        for(int j = s; j < n; j++)
        {
            U[s][j] = A[s][j];
            for(int k = 0; k < s; k++) U[s][j] -= L[s][k] * U[k][j];
        }
        for(int i = s; i < n; i++)
        {
            L[i][s] = A[i][s];
            for(int k = 0; k < s; k++) L[i][s] -= (L[i][k] * U[k][s]);
            L[i][s] /= U[s][s];
        }
    }
}


void LY_eq_B(matrix &L, vector<ld> &Y, vector<ld> &B)
{
    ll n = L.size();
    for(int i = 0; i < n; ++ i)
    {
        ld h = 0;
        for(int j = 0; j < i; ++j)
        {
            h += L[i][j] * Y[j];
        }
        Y[i] = B[i] - h;
    }
    //cout << "LLLL" << endl; print(L,30);
    //cout << "YYYY "; for(auto to:Y) cout << to << " "; cout << endl;
    //cout << "BBBB "; for(auto to:B) cout << to << " "; cout << endl;
}

void UX_eq_Y(matrix &U, vector<ld> &X, vector<ld> &Y)
{
    ll n = U.size();
    for(int i = n - 1; i >= 0; -- i)
    {
        ld h = 0;
        for(int j = n - 1; j > i; --j)
        {
            h += U[i][j] * X[j];
        }
        X[i] = (Y[i] - h)/U[i][i];
    }
    //cout << "UUUU" << endl; print(U,30);
    //cout << "XXXX "; for(auto to:X) cout << to << " "; cout << endl;
    //cout << "YYYY "; for(auto to:Y) cout << to << " "; cout << endl;
}

void check_ans(matrix &A, vector<ld> &X, vector<ld> &B, ll cnt=0)
{
    answers << "HERE IS WHAT B SHOULD BE " << "\n";
    print(B, answers, cnt);
    answers << "HERE IS CALCULATED B" << "\n";
    ll n = A.size();
    vector<ld> ans(n,0);
    vector<ld> diff(n);
    ld euNorm = 0.0;
    for(int i = 0; i < n; ++i)
    {
```

```
            for(int j = 0; j < n; ++j)
            {
                ans[i] += A[i][j] * X[j];
            }
            diff[i] = B[i] - ans[i];
            euNorm += diff[i] * diff[i];
        }
        print(ans, answers, cnt);
        answers << "HERE is B - B(calculated)" << "\n";
        print(diff, answers, cnt);
        answers << "Norm of this vector = " << fixed << setprecision(cnt) << sqrt(euNorm) << "\n";
        answers << fixed << setprecision(0);
    }


    void solve(matrix &A, vector<ld> &B)
    {
        answers << "Size = " << A.size() << "\n";
        answers << "Matrix A:\n";
        print(A, answers);
        answers << "B:\n";
        print(B, answers);
        ll num;
        if(!check_valid(A, num))
        {
            answers << "Cannot do LU decomposition, because " << num << "th leading principal minors is equal to
0\n";
            return;
        }

        matrix L,U;
        decompose_LU(A, L, U);
        answers << "Matrix L:\n";
        print(L, answers, 5);
        answers << "Matrix U:\n";
        print(U, answers, 5);
        matrix mult = L * U;
        answers << "Matrix mult(L * U):\n";
        print(mult, answers, 5);
        ld euNorm = 0;
        answers << "Matrix A - mult(L * U):\n";
        answers << fixed << setprecision(22);
        ll n = A.size();
        for(int i = 0; i < n; ++ i)
        {
            for(int j = 0; j < n; ++j)
            {
                euNorm += (A[i][j] - mult[i][j]) * (A[i][j] - mult[i][j]);
                answers << A[i][j] - mult[i][j] << " ";
            }
            answers << "\n";
        }
        answers << fixed << setprecision(22) << "\nNorm of matrix A - mult = " << sqrt(euNorm) << "\n";
        answers << fixed << setprecision(0);
        vector<ld> Y(n);
        vector<ld> X(n);
        LY_eq_B(L, Y, B);
        UX_eq_Y(U, X, Y);
        answers << "Y: ";
        print(Y, answers, 22);
        answers << "X: ";
        print(X, answers, 22);
```

```
    check_ans(A, X, B, 22);
    answers << "-------------------------------------------------------\n";
}
int main()
{
    ll n;
    while(tests >> n)
    {
        matrix A(n, vector<ld>(n, 0));
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                tests >> A[i][j];
        vector<ld> B(n);
        for(int i = 0 ; i < n; ++i) tests >> B[i];
        solve(A, B);
    }
    ll sz = 300;
    matrix A(sz, vector<ld>(sz, 0));
    A = get_random_matrix(sz);
    vector<ld> B(sz);
    for(int i = 0; i < sz; ++i) B[i] = rand() - rand() / 2;
    solve(A, B);
    return 0;
}
```

## Вхідні дані

```
3
13 -3 4
-3 -4 0
4 0 -5

14 -7 -1

3
13 -6 2
-6 22 4
2 4 -14

1 3 3

4
20 -4 -3 8
-4 -26 -4 2
-3 -4 20 2
8 2 2 -26

2 -12 -4 4

3
1 1 2
3 3 5
6 7 8

2 3 2
```

# Вихідні дані

```
Size = 3
Matrix A:
13.000 -3.000 4.000
-3.000 -4.000 0.000
4.000 0.000 -5.000

B:
14.000 -7.000 -1.000

Matrix L:
1.00000 0.00000 0.00000
-0.23077 1.00000 0.00000
0.30769 -0.19672 1.00000

Matrix U:
13.00000 -3.00000 4.00000
0.00000 -4.69231 0.92308
0.00000 0.00000 -6.04918

Matrix mult(L * U):
13.00000 -3.00000 4.00000
-3.00000 -4.00000 0.00000
4.00000 0.00000 -5.00000

Matrix A - mult(L * U):
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000
0.0000000000000000000000 -0.0000000000000000002168 0.0000000000000000000000
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000

Norm of matrix A - mult = 0.0000000000000000002168
Y: 14.0000000000000000000000 -3.7692307692307692307526 -6.0491803278688524589737

X: 1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000

HERE IS WHAT B SHOULD BE
14.0000000000000000000000 -7.0000000000000000000000 -1.0000000000000000000000

HERE IS CALCULATED B
14.0000000000000000000000 -7.0000000000000000000000 -1.0000000000000000000000

HERE is B - B(calculated)
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000

Norm of this vector = 0.0000000000000000000000
```

```
Size = 3
Matrix A:
13.000 -6.000 2.000
-6.000 22.000 4.000
2.000 4.000 -14.000

B:
1.000 3.000 3.000

Matrix L:
1.00000 0.00000 0.00000
-0.46154 1.00000 0.00000
0.15385 0.25600 1.00000

Matrix U:
13.00000 -6.00000 2.00000
0.00000 19.23077 4.92308
0.00000 0.00000 -15.56800

Matrix mult(L * U):
13.00000 -6.00000 2.00000
-6.00000 22.00000 4.00000
2.00000 4.00000 -14.00000

Matrix A - mult(L * U):
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000

Norm of matrix A - mult = 0.0000000000000000000000
Y: 1.0000000000000000000000 3.4615384615384615384949 1.9600000000000000000347

X: 0.1942446043165467625886 0.2122302158273381294959 -0.1258992805755395683515

HERE IS WHAT B SHOULD BE
1.0000000000000000000000 3.0000000000000000000000 3.0000000000000000000000

HERE IS CALCULATED B
1.0000000000000000000000 3.0000000000000000002168 3.0000000000000000000000

HERE is B - B(calculated)
0.0000000000000000000000 -0.0000000000000000002168 0.0000000000000000000000

Norm of this vector = 0.0000000000000000002168
```

```
Size = 4
Matrix A:
20.000 -4.000 -3.000 8.000
-4.000 -26.000 -4.000 2.000
-3.000 -4.000 20.000 2.000
8.000 2.000 2.000 -26.000

B:
2.000 -12.000 -4.000 4.000

Matrix L:
1.00000 0.00000 0.00000 0.00000
-0.20000 1.00000 0.00000 0.00000
-0.15000 0.17164 1.00000 0.00000
0.40000 -0.13433 0.12695 1.00000

Matrix U:
20.00000 -4.00000 -3.00000 8.00000
0.00000 -26.80000 -4.60000 3.60000
0.00000 0.00000 20.33955 2.58209
0.00000 0.00000 0.00000 -29.04421

Matrix mult(L * U):
20.00000 -4.00000 -3.00000 8.00000
-4.00000 -26.00000 -4.00000 2.00000
-3.00000 -4.00000 20.00000 2.00000
8.00000 2.00000 2.00000 -26.00000

Matrix A - mult(L * U):
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000001084
0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000 0.0000000000000000000000
0.0000000000000000000000 0.0000000000000000001084 0.0000000000000000000000 0.0000000000000000000000

Norm of matrix A - mult = 0.0000000000000000001533
Y: 2.0000000000000000000000 -11.6000000000000000003469 -1.7089552238805970148412 1.8587415153182902218323

X: 0.2016675088428499242022 0.4372662961091460333599 -0.0758969176351692774115 -0.0639969681657402728617

HERE IS WHAT B SHOULD BE
2.0000000000000000000000 -12.0000000000000000000000 -4.0000000000000000000000 4.0000000000000000000000

HERE IS CALCULATED B
1.9999999999999999998916 -12.0000000000000000008674 -4.0000000000000000000000 4.0000000000000000000000

HERE is B - B(calculated)
0.0000000000000000001084 0.0000000000000000008674 0.0000000000000000000000 0.0000000000000000000000

Norm of this vector = 0.0000000000000000008741
```

```
Size = 3
Matrix A:
1.000 1.000 2.000
3.000 3.000 5.000
6.000 7.000 8.000

B:
2.000 3.000 2.000

Cannot do LU decomposition, because 1th leading principal minors is equal to 0
```

### Аналіз чисельних експериментів

LU-розклад є кращим способом реалізації алгоритму Гауса для повторного розв’язування ряду рівнянь з тією самою лівою частиною. Тобто для розв’язання рівняння Ax = b з різними значеннями b для одного і того ж A.(Метод Гауса буде постійно робити ~n^3/3 операції, в той час як з допомогою LU – розкладу це можна робити за ~n^2 операцій).

Час роботи програми рандомної матриці 300 × 300.

```
"A:\T\3 term\Numerical Methods\lab2LU\lab2LU\bin\Debug\lab2LU.exe"

Process returned 0 (0x0)   execution time : 7.191 s
Press any key to continue.
```

```
"A:\T\3 term\Numerical Methods\lab1.2\bin\Debug\lab1.exe"

Process returned 0 (0x0)   execution time : 0.833 s
Press any key to continue.
```

Час виконання для різних систем(різні, як в лівій, так і в правій частині) буде швидше в методу Гауса(O(n^3)), оскільки в методі LU розкладу потрібна ще перевірка на можливість самого розкладу(порядка n^4 операцій, знайти визначник матриці розміром 1 × 1, 2 × 2, 3 × 3, …, n × n, кожен визначник найшвидше можна знайти методом Гауса(n^3)).

Похибка є досить малою, отже алгоритм розв'язує СЛАР з високою точністю.

### Висновок

Я закріпив вміння та навички практичного використання прямих методу розв’язування СЛАР - методу LU-розкладу. Створив програму, яка зчитує СЛАР з файлу, виводить вектор-рішення СЛАР, а також додаткову інформацію (матриці A, L, U…, вектори B, X, Y… різні норми і тд). Також програма перевіряє чи взагалі можливо поточну матрицю розкласти на L та U матриці.