# Метод квадратних коренів
# (схема Холецького)

Підготувала:
Студентка ПМІ-23
Шувар Софія

**Мета**: Реалізувати алгоритм схеми Холецького для розв`язування симетричних СЛАР. Продемонструвати роботу програми на конкретному прикладі.

**Завдання**:

(22) $$\begin{cases} 1{,}17x_1 - 0{,}65x_2 + 1{,}54x_3 = -1{,}43 \\ -0{,}65x_1 + 1{,}16x_2 - 1{,}73x_3 = 0{,}68 \\ 1{,}54x_1 - 1{,}73x_2 + 2{,}15x_3 = 1{,}87 \end{cases}$$

**Хід роботи**:

1. Розв’язати систему лінійних рівнянь методом квадратних коренів (схемою Холецького).
2. Реалізувати алгоритм методу квадратних коренів на довільний мові програмування.
3. Продемонструвати результати роботи алгоритму на конкретному прикладі.

**Розв’язок системи**:

22. $$\begin{cases} 1{,}17x_1 - 0{,}65x_2 + 1{,}54x_3 = 1{,}45 \\ -0{,}65x_1 + 1{,}16x_2 - 1{,}73x_3 = 0{,}68 \\ 1{,}54x_1 - 1{,}73x_2 + 2{,}15x_3 = 1{,}87 \end{cases}$$

$$u_{11} = \frac{3\sqrt{13}}{10}$$

$$u_{12} = -\frac{\sqrt{13}}{6}$$

$$u_{13} = \frac{77\sqrt{13}}{195}$$

$$u_{22} = \frac{\sqrt{719}}{30}$$

$$u_{23} = -\frac{787\sqrt{719}}{21570}$$

$$u_{33} = \frac{i \cdot \sqrt{1821926587}}{46735}$$

$$U = \begin{pmatrix} \frac{3\sqrt{13}}{10} & -\frac{\sqrt{13}}{6} & \frac{77\sqrt{13}}{195} \\ 0 & \frac{\sqrt{719}}{30} & -\frac{787\sqrt{719}}{21570} \\ 0 & 0 & \frac{i\sqrt{1821926587}}{46735} \end{pmatrix}$$

$$U^T = \begin{pmatrix} \frac{3\sqrt{13}}{10} & 0 & 0 \\ -\frac{\sqrt{13}}{6} & \frac{\sqrt{719}}{30} & 0 \\ \frac{77\sqrt{13}}{195} & -\frac{787\sqrt{719}}{21570} & \frac{i\sqrt{1821926587}}{46735} \end{pmatrix}$$

Оскільки у результаті отримано комплексне число, то порахувати невідомі змінні даного СЛАР неможливо

## Реалізація алгоритму:

Для реалізації алгоритму я обрала мову програмування С++.

### Функція user_input(вхідні дані):

Функція наповнює матрицю a та послідовність вільних членів answ введеними користувачем з клавіатури значеннями.

```cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;


void user_input(unsigned int n, vector<vector<double>> &a, vector<double> &answ){
    for (int i = 0; i < n; ++i) {
        a[i].resize(n);
        for (int j = 0; j < n; ++j) {
            cin >> a[i][j];
        }
        cin >> answ[i];
    }
}
```

### Функція user_input(вхідні дані):

Аргументи: n – розмір матриці;

Функція створює матрицю розміру n*n заповнену нулями.

```cpp
vector<vector<double>> zeroed_matrix(unsigned int n) {
    vector<vector<double>> p(n);
    for (int i = 0; i < n; ++i) {
        p[i].resize(n);
        for (int j = 0; j < n; ++j) {
            p[i][j] = 0;
        }
    }
    return p;
}
```

### Функція Cholesky_Decomposition (реалізація алгоритму):

Аргументи: p – початкова матриця; n – розміри матриці; L та U результуючі матриці заповнені нулями U та U*

Мета алгоритму: представлення симетричної додатньоозначеної матриці у вигляді A = UU* де U — нижня трикутна матриця з додатніми елементами на діагоналі.

Елементи матриці U можна обчислити, починаючи з верхнього лівого кута, за формулами:

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \qquad (1)$$

$$L_{ij} = \frac{1}{L_{jj}} \left( A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right), \qquad (2)$$

якщо $j < i$.

Алгоритм складається з наступних частин:

**Рядок 32** – оголошуємо фіктивні змінні необхідні для подальшого розвитку алгоритму;

**Рядки 34, 35** – розпочинаємо два цикли довжинами n по усіх елементах матриці;

**Рядки 36-39** – у змінній var2 обчислюємо суму необхідну у формулі (2);

**Рядок 42-45** – у змінній var обчислюємо суму необхідну у формулі (1);

**Рядки 40, 44** – заповнюємо відповідні клітинки матриці L згідно формул вище;

**Рядок 47** – верхню трикутну матрицю L заповнюємо нулями.

**Рядок 15** – заповнюємо діагональ матриці L одиницями.

**Рядок 50** – заповнюємо матрицю U.

```
30
31  void Cholesky_Decomposition(vector<vector<double>> p, unsigned int n, vector<vector<double>> &L, vector<vector<double>> &U) {
32      long i, j, k; double val = 0, val2 = 0;
33
34      for (i = 0; i < n; i++)
35          for (j = 0; j < n; j++) {
36              val = 0; val2 = 0;
37              if (i > j) {
38                  if (j > 0)
39                      for (k = 1; k < j + 1; k++) val2 += (L[i][k - 1] * L[j][k - 1]);
40                  L[i][j] = (p[i][j] - val2) / L[j][j];
41              }
42              else if (i == j) {
43                  for (k = 0; k < i; k++) val += pow(L[i][k], 2);
44                  L[i][j] = sqrt(p[i][j] - val);
45              }
46              else
47                  L[i][j] = 0;
48          }
49
50      for (i = 0; i < n; i++) for (j = 0; j < n; j++) U[i][j] = L[j][i];
51  }
52
```

## Функція result(обчислення змінних x):

Аргументи: U – знайдена матриця U*; answ – послідовність вільних членів; n – розмір матриці; x – шуканий вектор невідомих змінних.

Функція наповнює вектор невідомих змінним знайденими значеннями.

```cpp
void result(vector<vector<double>> &U, vector<double> &answ, unsigned int n, vector<double> &x){
    vector<double> y(n);
    for (int i = 0; i < n; ++i) {
        double val = 0;
        for (int j = 0; j < i; ++j)
            val += U[j][i] * y[j];
        y[i] = (answ[i] - val) / U[i][i];
    }
    for (int k = n - 1; k >= 0; --k) {
        double val = 0;
        for (int i = k + 1; i < n; ++i)
            val += U[k][i] * x[i];
        x[k] = (y[k] - val) / U[k][k];
    }
}
```

## Функція output_matrix(вивід):

Аргументи: a – квадратна матриця, n – розмір матриці.

Функція виводить матрицю у гарному вигляді

```cpp
void output_matrix(vector<vector<double>> &a, long n) {
    for (long i = 0; i < n; i++) {
        for (long j = 0; j < n; j++)
            cout << a[i][j] << "\t";
        cout << "\n";
    }
    cout << "\n";
}
```

## main():

Вивід усіх знайдених результатів.

```cpp
int main() {
    unsigned int n;
    cout << "Enter amount of variables: ";
    cin >> n;
    vector<vector<double>> a(n);
    vector<double> answ(n);
    cout << "Enter matrix: \n";
    user_input(n, a, answ);

    vector<vector<double>> L = zeroed_matrix(n); vector<vector<double>> U = zeroed_matrix(n);
    Cholesky_Decomposition(a, n, L, U);
    vector<double> res(n);
    result(U, answ, n, res);

    cout << "\nRESULTS: \n";
    for (int i = 1; i < n + 1; ++i) cout << "x" << i << " = " << res[i] << "\n";
    cout << "\n";

    cout << "L matrix: \n";
    output_matrix(L, n);
    cout << "U matrix: \n";
    output_matrix(U, n);
}
```

Результати роботи програми на конкретному прикладі:

```
Enter amount of variables: 4
Enter matrix:
4 2 2 1 9
2 5 1 2 10
2 1 5 1 9
1 2 1 4.875 8.875

RESULTS:
x1 = 1
x2 = 1
x3 = 1
x4 = 0

L matrix:
2   0   0   0
1   2   0   0
1   0   2   0
0.5 0.75    0.25    2

U matrix:
2   1   1   0.5
0   2   0   0.75
0   0   2   0.25
0   0   0   2


Process finished with exit code 0
```

```
Enter amount of variables: 3
Enter matrix:
1.17 -0.65 1.54 -1.43
 -0.65 1.16 -1.73 0.68
1.54 -1.73 2.15 1.87

RESULTS:
x1 = nan
x2 = nan
x3 = 0

L matrix:
1.08167 0   0
-0.600925   0.893806    0
1.42373 -0.978338   nan

U matrix:
1.08167 -0.600925   1.42373
0   0.893806    -0.978338
0   0   nan


Process finished with exit code 0
```

Розв’язки набувають значення nan оскільки при виконанні алгоритму елементом результуючої матриці стає комплексне число.

## Висновок

Виконуючи дану практичну роботу, я навчилась розв`язувати симетричні СЛАР за допомогою схеми Холецького та реалізувала алгоритм розв`язання на мові С++.