Grupa lab.3	Data wykonania 05.04.2024	Inżynieria Obliczeniowa 2023/2024
Temat ćwiczenia **Rozwiązywanie układów równań liniowych - rozkład LU**		
Imię i nazwisko Karolina Kurowska		Ocena i uwagi

Wstęp

Rozwiązanie układów równań liniowych jest fundamentalnym zagadnieniem w matematyce i naukach technicznych. Jedną z efektywnych metod ich rozwiązywania jest metoda rozkładu LU (Lower-Upper) metodą Doolittle'a. Ta technika opiera się na dekompozycji macierzy współczynników układu na iloczyn dwóch macierzy trójkątnych: dolnej (lower) i górnej (upper). Następnie, zamiast rozwiązywać pierwotny układ równań Ax = b, rozwiązuje się dwa nowe układy równań: Ly = b oraz Ux = y. Te dwa etapy są znacznie prostsze do przeprowadzenia, co przyspiesza obliczenia numeryczne. Dzięki temu przekształcamy oryginalny układ równań w równoważny, ale łatwiejszy do rozwiązania układ, co przyspiesza obliczenia numeryczne.

Przebieg ćwiczenia

Na wstępie przygotowujemy cztery macierze. Macierz *matrix_A* będzie przechowywać równania pobrane z pliku tekstowego. Zarówno *matrix_L*, jak i *matrix_U* są początkowo zerowane, jednak na głównej przekątnej *matrix_L* ustawiamy jedynki, aby później móc wypełnić ją prawidłowymi wartościami. Te macierze będą służyły do przechowywania obliczeń dla macierzy górno-trójkątnej (Upper) oraz dolno-trójkątnej (Lower). Ponadto tworzymy macierz *matrix_B*, która będzie przechowywać wyrazy wolne występujące w równaniach.

```
int quantity;
ifstream plik;
plik.open ("RURL_dane1.txt", ios::in);//opcja 1
//plik.open ("RURL_dane2.txt", ios::in);//opcja 2
if (plik.good () == false) {
    cout << "Plik nie istnieje\n";
    exit (0);
}
plik >> quantity;
cout << "Ilosc rownan: " << quantity << endl;

//stworzenie macierzy
double** matrix_A = new double* [quantity];
double** matrix_L = new double* [quantity];
double** matrix_U = new double* [quantity];
double* matrix_B = new double[quantity];
```

```
for (int i = 0; i < quantity; i++) {
    matrix_A[i] = new double[quantity + 1];
    matrix_L[i] = new double[quantity];
    matrix_U[i] = new double[quantity];
}

for (int i = 0; i < quantity; i++) {
    for (int j = 0; j < quantity + 1; j++) {
        plik >> matrix_A[i][j];
        if (j == quantity) {
            matrix_B[i] = matrix_A[i][j];
        }
    }
}
for (int i = 0; i < quantity; i++) {
    for (int j = 0; j < quantity; j++) {
        matrix_U[i][j] = 0;
        if (i == j) {
            matrix_L[i][j] = 1;
        }
        else {
            matrix_L[i][j] = 0;
        }
    }
}
plik.close ();
```

Posiadamy układ równań w postaci A * X = B. W tej metodzie musimy rozłożyć macierz A na dwie macierz trójkątne L i U, z którymi układ będzie łatwiejszy do policzenia. W tym celu implementujemy dwa wzory metodą Dolittle'a. Zgodnie z nimi, naprzemiennie wyliczają się kolumna z wierszem macierzy L i U.

- Wzór na macierz U:

$$\text{gdy } i \le j, \quad u_{i,j} = a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} \times u_{k,j}$$

- Wzór na macierz L:

$$\text{gdy } i > j, \quad l_{i,j} = \frac{1}{u_{j,j}} (a_{i,j} - \sum_{k=0}^{i-1} l_{i,k} \times u_{k,j})$$

```
void lower_and_upper (double** U, double** L, double** A, int quantity) {
    double suma = 0;
    for (int i = 0; i < quantity; i++) {
        for (int j = 0; j < quantity; j++) {
            if (i <= j) {
                for (int k = 0; k < i; k++) {
                    suma += (L[i][k] * U[k][j]);
                }
                U[i][j] = A[i][j] - suma;
                suma = 0;
            }
            else {
                for (int k = 0; k < j; k++) {
                    suma += (L[i][k] * U[k][j]);
                }
                L[i][j] = (1 / U[j][j]) * (A[i][j] - suma);
                suma = 0;
            }
        }
    }
```

Dzięki temu nasz układ możemy zapisać w postaci $L \times (U \times X) = B$. Teraz możemy zapisać U×X w formie wektora Y. Obliczamy go korzystając z macierzy L podstawiając w przód. Użyjemy następującego wzoru:

$$y_i = b_i - \sum_{j=0}^{i-1} l_{i,j} \times y_j$$

```cpp
void policz_y (double* B, double** L, double* Y, int quantity) {
    double suma = 0;
    Y[0] = B[0];
    for (int i = 0; i < quantity; i++) {
        for (int j = 0; j < i; j++) {
            suma += (L[i][j] * Y[j]);
        }
        Y[i] = B[i] - suma;
        suma = 0;
    }

    //wyświetl
    cout << "Macierz Y: " << endl;
    for (int i = 0; i < quantity; i++) {
        cout << Y[i] << endl;
    }
    cout << endl;
}
```

Na koniec aby znaleźć rozwiązanie układu równań łączymy utworzoną macierz U i Y. Dzięki temu zabiegowi będziemy mogli wykorzystać funkcję zaimplementowaną przez nas na wcześniejszych laboratoriach.

```cpp
void policz_matrix_UandY (double ** U, double* Y, double** A, int quantity) {
    for (int i = 0; i < quantity; i++) {
        for (int j = 0; j < quantity + 1; j++) {
            if (j == quantity) {
                A[i][j] = Y[i];
            }
            else {
                A[i][j] = U[i][j];
            }
        }
    }

    cout << "Macierz U rozszerzona o macierz Y: " << endl;
    wypisz_macierz (quantity, A);
}
```

```cpp
void policz_rozwiazania (int quantity, double** table, double* wyniki) {
    for (int l = quantity - 1; l >= 0; l--) {
        wyniki[l] = table[l][quantity] / table[l][l]; //b_n/a_nn
        for (int k = l + 1; k < quantity; k++) {
            wyniki[l] -= (table[l][k] * wyniki[k]) / table[l][l]; //reszta wzoru
        }
    }
    for (int i = 0; i < quantity; i++) { //wypisanie roziwazań
        cout << "Wynik rownania " << i + 1 << ". to " << wyniki[i] << endl;
    }
}
```

Wyniki:

RURL_dane1.txt

```
Macierz glowna A:
 2 4 2 1 10
 2 2 3 3 6
 4 2 2 1 6
 0 2 1 1 4

Macierz Lower:
 1 0 0 0
 1 1 0 0
 2 3 1 0
 0 -1 -0.4 1

Macierz Upper:
 2 4 2 1
 0 -2 1 2
 0 0 -5 -7
 0 0 0 0.2

Macierz B:
10
6
6
4

Macierz Y:
10
-4
-2
-0.8
```

```
Macierz U rozszerzona o macierz Y:
 2 4 2 1 10
 0 -2 1 2 -4
 0 0 -5 -7 -2
 0 0 0 0.2 -0.8

Wynik rownania 1. to -1
Wynik rownania 2. to 1
Wynik rownania 3. to 6
Wynik rownania 4. to -4
```

RURL_dane2.txt

```
Macierz glowna A:
 1 1 -2 1 -2 -5 8
 2 -4 -1 2 3 3 1
 2 -2 6 -1 6 5 5
 0 2 1 1 4 5 5
 -5 0 4 -1 9 4 10
 7 -2 -4 5 3 -1 -5

Macierz Lower:
 1 0 0 0 0 0
 2 1 0 0 0 0
 2 0.666667 1 0 0 0
 0 -0.333333 0.25 1 0 0
 -5 -0.833333 -0.4375 1.53571 1 0
 7 1.5 0.6875 0.0357143 -5.18605 1

Macierz Upper:
 1 1 -2 1 -2 -5
 0 -6 3 0 7 13
 0 0 8 -3 5.33333 6.33333
 0 0 0 1.75 5 7.75
 0 0 0 0 -0.511905 -19.2976
 0 0 0 0 0 -90.2093

Macierz B:
8
1
5
5
10
-5

Macierz Y:
8
-15
-1
0.25
36.6786
```

```
Macierz U rozszerzona o macierz Y:
 1 1 -2 1 -2 -5 8
 0 -6 3 0 7 13 -15
 0 0 8 -3 5.33333 6.33333 -1
 0 0 0 1.75 5 7.75 0.25
 0 0 0 0 -0.511905 -19.2976 36.6786
 0 0 0 0 0 -90.2093 152.395

Wynik rownania 1. to -9.43387
Wynik rownania 2. to -1.49549
Wynik rownania 3. to 17.918
Wynik rownania 4. to 30.3857
Wynik rownania 5. to -7.96649
Wynik rownania 6. to -1.68935
```

Wnioski

Metoda Doolittle'a to skuteczne narzędzie do rozwiązywania skomplikowanych układów równań liniowych. Rozkład LU pozwala na efektywne przekształcenie układu, co znacznie przyspiesza obliczenia, zwłaszcza przy dużych macierzach. Dzięki niej możemy uniknąć bezpośredniego rozwiązywania układu, co może być czasochłonne. Zamiast tego, dzielimy proces na rozwiązanie układów dla macierzy dolnotrójkątnej L i górnotrójkątnej U, co jest bardziej efektywne obliczeniowo. Metoda ta znajduje zastosowanie zwłaszcza w praktycznych zastosowaniach, gdzie mamy do czynienia z dużymi zbiorami danych lub skomplikowanymi modelami matematycznymi.