Laboratorium 5

Artem Buhera
GĆ01 135678

19.04.2021

Dany laboratorium polega na zaimplementowaniu programu do rozwiązywania układów równań w postaci $A\vec{x}=\vec{b}$, realizujący dekompozycję LU przy zastosowaniu eliminacji Gaussa z częściowym wyborem elementu podstawowego.

Dekompozycja LU polega na rozkładzie podanej macierzy *A* na iloczyn dwóch macierzy *L* i *U*.

Przeprowadzamy ją poprzez eliminację Gaussa – czyli:

- od każdego wiersza macierzy *A*, zaczynając od drugiego, odejmujemy poprzedni wiersz dzielony przez wartość pierwszego elementu aktualnego wiersza
  - w wyniku tego otrzymujemy zera na pierwszych miejscach wiersza
- jednocześnie tworzymy macierz dolnotrójkątną *L*, która składa się z tych "dzielników"
- w przypadku, gdy dany element jest równy lub bliski zeru, przeprowadzamy zamiane miejscami aktualnego wiersza, a tego, który z poniższych ma największą wartość bezwzględną elementu o tej samej kolumnie
  - robimy tą zamianę wierzy o odpowiednich w macierzach *L*, *U* oraz wektorze $\vec{b}$
- powtarzamy prodecurę, zaczynając o jeden wiersz niżej

W wyniku tej dokompozycji macierz <u>A</u> zamienia się na macierz górnotrójkątną <u>U</u>.

Motywacją użycia dekompozycji LU zamiast samej eliminacji Gaussa jest przekształcenie równania *Ax=b* w dany sposób:

$$A\vec{x}=\vec{b} \Leftrightarrow LU\vec{x}=\vec{b} \Leftrightarrow L(U\vec{x})=\vec{b} \Leftrightarrow \begin{cases} L\vec{y}=\vec{b}, \\ U\vec{x}=\vec{y} \end{cases}$$

Korzystając się z tego faktu, że macierzy *L* i *U* są odpowiednio dolno- i górnotrójkątne, możemy w prosty sposób obliczyć z pierwszego równania wektor $\vec{y}$, a po tym szukany wektor rozwiązań $\vec{x}$ z drugiego. Także pozwala nam to rozwiązywać wiele równań w postaci $A\vec{x}_1=\vec{b}_1, A\vec{x}_2=\vec{b}_2, \ldots$, gdzie macierz *A* jest ta sama, bez przeprowadzenia obliczeń z jej powtórnym użyciem, co by skutkowało zwiększeniem kosztów obliczeniowych.

Do implementacji programu w języku C++ zostały stworzone dwie dodatkowe klasy *Matrix* oraz *Vector* na podstawie struktury danych *std::array* ze standardowej biblioteki szablonów.

Wybór danej struktury zamiast zwykłych list pozwala na użycie stworzonych metod i funkcji dla macierzy i wektorów o dynamicznym rozmiarze praktycznie bez żadnych obciążeń pamięciowych i obliczeniowych.

vector.h:

```cpp
#include <array>
#include <ostream>
#include <iostream>
#include <iomanip>

#ifndef VECTOR_H
#define VECTOR_H
using namespace std;

template <int N>
class Vector: public array<double,N> {
public:

    void swapElements(int a, int b) {
        std::swap((*this)[a], (*this)[b]);
    }

    void print(int width=10) {
        for (auto value : (*this)) {
            if (abs(value) < 1.0e-300) value = 0.0;
            std::cout << std::setw(width) << value << " ";
        }
        std::cout << std::endl;
    }

    void print(const std::string &text, int width=10) {
        std::cout << text << std::endl;
        print(width);
    }
};

#endif
```

matrix.h:

```cpp
#include "vector.h"

#ifndef MATRIX_H
#define MATRIX_H
using namespace std;

template<int N, int M>
class Matrix : array<Vector<N>, M> {
public:
    Matrix<N, M>() = default;
    Matrix<N, M>(array<Vector<N>, M> a) : array<Vector<N>, M> (a) {}

    using array<Vector<N>, M>::operator[];

    void swapRows(int a, int b) {
        swap((*this)[a], (*this)[b]);
    };

    void print(int width=8) const {
        for (Vector row : (*this))
            row.print(width);
        std::cout << std::endl;
    }

    void print(string text, int width=8) const {
        std::cout << text << std::endl;
        for (Vector row : (*this))
            row.print(width);
        std::cout << std::endl;
    }
};

template <int N, int M>
Vector<N> matrixTimesVector(const Matrix<N, M> &A, const Vector<M> &x) {
    Vector<N> result;

    for (int i = 0; i < N; i++) {
        double sum = 0.0;
        for (int j = 0; j < M; j++)
            sum += A[i][j] * x[j];
        result[i] = sum;
    }

    return result;
}

#endif
```

Główny program:

```cpp
#include "vector.h"
#include "matrix.h"

using namespace std;


template <int N, int M> int getPartialPivot(const Matrix<N, M> &A, int from);
template <int N, int M> pair<Matrix<N, M>, Matrix<N, M>> LU_decomposition(const
Matrix<N, M> &A, Vector<N> &indexes);
template <int N> void rearrangeIndexes(Vector<N> &vect, const Vector<N>
&indexes);
template <int N, int M> Vector<N> solveL(Matrix<N, M> &L, Vector<N> &b);
template <int N, int M> Vector<N> solveU(Matrix<N, M> &U, Vector<N> &y);
template <int N, int M> Vector<N> solveAxb(const Matrix<N, M> &A, Vector<N> &b);


int main() {
    Matrix<4, 4> A = {{
        1.0, -20.0, 30.0, -4.0,
        2.0, -40.0, -6.0, 50.0,
        9.0, -180.0, 11.0, -12.0,
        -16.0, 15.0, -140.0, 13.0
    }};
    Vector<4> b = {{35.0, 104.0, -366.0, -354.0}};

    cout << "Rozwiązujemy układ równań w postaci Ax = b metodą dekompozycji" <<
endl;
    A.print("Macierz A: ", 5);
    b.print("Wektor b: ", 5);

    Vector<4> x = solveAxb(A, b);

    cout << endl << "Na koniec sprawdźmy czy otrzymany wektor x faktycznie
zawiera rozwiązania do pierwotnego równania Ax = b" << endl;

    Vector<4> expected_b = matrixTimesVector(A, x);
    expected_b.print();
    cout << "Po wymnożeniu macierzy A i obliczonego wektora x ewidentnie
otrzymaliśmy wektor b" << endl;
}
```

```cpp
template <int N, int M>
Vector<N> solveAxb(const Matrix<N, M> &A, Vector<N> &b) {
    Vector<N> x;
    Vector<N> indexes;
    for (int i = 0; i < 4; i++)
        indexes[i] = i;

    cout << endl << endl << "Zaczynamy dekompozycję LU" << endl;
    auto [L, U] = LU_decomposition(A, indexes);

    cout << "Po dekompozycji otrzymaliśmy macierze L oraz U" << endl;
    L.print("Macierz L: ");
    U.print("Macierz U: ");
    indexes.print("Nowa kolejność wierszy w macierzach L, U oraz wektorze b,\n
która powstała w wyniku zamian wierszy po wyborze częściowym:", 2);

    rearrangeIndexes(b, indexes);
    b.print("Wektor b po zmianie kolejności wierszy: ", 8);


    cout << endl << "Rozwiązujemy układ równań Ly = b, gdzie L - macierz
dolnotrójkątna" << endl;
    Vector<N> y = solveL(L, b);
    y.print("Wektor y: ", 8);


    cout << endl << "Rozwiązujemy układ równań Ux = y, gdzie U - macierz
górnotrójkątna" << endl;
    x = solveU(U, y);
    x.print("Wektor x:", 8);
    cout << "Znaleźliśmy rozwiązanie dla układu równań Ax = b" << endl;

    return x;
}

template <int N, int M>
int getPartialPivot(const Matrix<N, M> &A, int from) {
    int pivot = from;
    double max = A[from][from];

    for (int i = from; i < N; i++) {
        double value = std::abs(A[i][from]);
        if (value > max) {
            max = value;
            pivot = i;
        }
    }

    return pivot;
}
```

```cpp
template <int N, int M>
pair<Matrix<N, M>, Matrix<N, M>> LU_decomposition(const Matrix<N, M> &A,
Vector<N> &indexes) {
    Matrix<N, M> L;
    Matrix<N, M> U = A;

    for (int k = 0; k < N-1; k++) {
        double divisor = U[k][k];
        if (divisor <= 1.0e-6 && divisor >= -1.0e-6) {
            int pivot = getPartialPivot(U, k);
            U.swapRows(k, pivot);
            L.swapRows(k, pivot);
            indexes.swapElements(k, pivot);
            divisor = U[k][k];

            cout << "Zamiana kolumn " << k << " oraz " << pivot << ": " << endl;
            U.print();
        }

        L[k][k] = 1;

        Vector<N> firstRow = U[k];
        for (int row = k+1; row < N; row++) {
            double firstInCol = U[row][k];
            double multiplier = firstInCol / divisor;

            L[row][k] = multiplier;

            for (int col = k; col < N; col++)
                U[row][col] -= firstRow[col] * multiplier;
        }

        cout << "Po iteracji " << k+1 << ": " << endl;
        U.print();
    }

    L[N-1][N-1] = 1;

    return make_pair(L, U);
}

template <int N>
void rearrangeIndexes(Vector<N> &vect, const Vector<N> &indexes) {
    Vector tmp = vect;
    for (int i = 0; i < N; i++) {
        double val = tmp[i];
        int index = indexes[i];
        vect[index] = val;
    }
};
```

```cpp
template <int N, int M>
Vector<N> solveL(Matrix<N, M> &L, Vector<N> &b) {
    Vector<N> x;
    for (int n = 0; n < N; n++) {
        double sum = 0.0;
        for (int m = 0; m <= n-1; m++)
            sum += L[n][m] * x[m];
        x[n] = b[n] - sum;
    }
    return x;
}

template <int N, int M>
Vector<N> solveU(Matrix<N, M> &U, Vector<N> &y) {
    Vector<N> x;
    int sumElementsCount = 0;
    for (int n = N-1; n >= 0; n--) {
        double sum = 0.0;
        int m = N-1;
        for (int _ = 0; _ < sumElementsCount; _++) {
            sum += U[n][m] * x[m];
            m--;
        }
        sumElementsCount++;
        x[n] = (y[n] - sum) / U[n][n];
    }
    return x;
}
```

Wynik działania programu:

```
Rozwiązujemy układ równań w postaci Ax = b metodą dekompozycji
Macierz A:
    1   -20    30    -4
    2   -40    -6    50
    9  -180    11   -12
  -16    15  -140    13

Wektor b:
   35   104  -366  -354


Zaczynamy dekompozycję LU
Po iteracji 1:
       1      -20       30       -4
       0        0      -66       58
       0        0     -259       24
       0     -305      340      -51

Zamiana kolumn 1 oraz 3:
       1      -20       30       -4
       0     -305      340      -51
       0        0     -259       24
       0        0      -66       58

Po iteracji 2:
       1      -20       30       -4
       0     -305      340      -51
       0        0     -259       24
       0        0      -66       58

Po iteracji 3:
       1      -20       30       -4
       0     -305      340      -51
       0        0     -259       24
       0        0        0  51.8842

Po dekompozycji otrzymaliśmy macierze L oraz U
Macierz L:
       1        0        0        0
     -16        1        0        0
       9        0        1        0
       2        0 0.254826        1

Macierz U:
       1      -20       30       -4
       0     -305      340      -51
       0        0     -259       24
       0        0        0  51.8842

Nowa kolejność wierszy w macierzach L, U oraz wektorze b,
która powstała w wyniku zamian wierszy po wyborze częściowym:
 0  3  2  1
Wektor b po zmianie kolejności wierszy:
      35     -354     -366      104

Rozwiązujemy układ równań Ly = b, gdzie L - macierz dolnotrójkątna
Wektor y:
      35      206     -681  207.537
```

Rozwiązujemy układ równań Ux = y, gdzie U - macierz górnotrójkątna
Wektor x:
1 2 3 4
Znaleźliśmy rozwiązanie dla układu równań Ax = b

Na koniec sprawdźmy czy otrzymany wektor x faktycznie zawiera rozwiązania do pierwotnego równania Ax = b
35 104 -366 -354
Po wymnożeniu macierzy A i obliczonego wektora x ewidentnie otrzymaliśmy wektor b