Київський національний університет імені Тараса Шевченка
Факультет комп`ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем
Алгоритми та складність

# Лабораторна робота №1

“ Побудова оберненої матриці методом LU-розкладання”

Варіант №1

Виконав Студент 2-го курсу
Групи ІПС-21
Ляшенко Матвій Олексійович

Київ - 2024

## Завдання

Реалізувати алгоритм побудови оберненої дійснозначної матриці типу T** методом LU-розкладання. У процесі роботи використовується алгоритм Штрассена для множення матриць.

## Теорія

***LU-розкладання (LU decomposition)*** — це метод розкладання квадратної матриці A на добуток двох матриць: L та U, де:

- L (lower triangular matrix) — нижня трикутна матриця з одиницями на головній діагоналі;
- U (upper triangular matrix) — верхня трикутна матриця.

Тобто, A=LU.

LU-розкладання застосовується в чисельних методах для розв'язання систем лінійних алгебраїчних рівнянь, обчислення обернених матриць, визначників тощо.

Умови застосування LU-розкладу:

- Матриця A повинна бути квадратною та невиродженою (тобто мати ненульовий визначник).
- Матриця U не повинна мати нульових елементів на головній діагоналі, інакше неможливо виконати зворотну підстановку.

***Обернена матриця $A^{-1}$*** для квадратної матриці A визначається як така матриця, що:

$A \cdot A^{-1} = I$, де I — одинична матриця.

Для обчислення оберненої матриці за допомогою LU-розкладу виконуються такі кроки:

Виконується LU-розкладання матриці

$A = L \cdot U$.

Розв'язуються дві системи рівнянь для кожного стовпця одиничної матриці:

- $L \cdot Y = E_i$ (пряма підстановка),
- $U \cdot X = Y$ (зворотна підстановка).

Всі отримані X-вектори формують стовпці оберненої матриці $A^{-1}$.

***Алгоритм Штрассена*** — це ефективний алгоритм множення матриць, який дозволяє зменшити кількість необхідних операцій множення порівняно зі стандартним методом. Він був запропонований Фолькером Штрассеном у 1969 році і є першим відомим алгоритмом для множення матриць, який покращив асимптотичну складність цього завдання (через зменшення кількості операцій множення з 8 до 7).

## Алгоритм

1. Введення розмірності матриці та її елементів

**Опис:**

- Спочатку користувач вводить розмірність квадратної матриці n та заповнює її елементи.
- Якщо n є не додатним, програма завершує виконання.

**Код:**

```
int size;
cout << "Enter the size of the matrix: ";
cin >> size;

if (size <= 0) {
    cout << "Matrix size must be a positive integer." << endl;
    return 1;
}

Matrix<double> inputMatrix(size);
cout << "Enter elements of the matrix:\n";
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        cin >> inputMatrix.elements[i][j];
    }
}
```

2. Доповнення матриці до найближчого розміру, що є степенем двійки (при потребі)

**Опис:**

- Для забезпечення коректної роботи алгоритму Штрассена розмірність матриці збільшується до найближчого розміру, який є степенем двійки.

**Код:**

```
Matrix<double> paddedMatrix = padMatrix(inputMatrix);

// Функція доповнення матриці
template <typename T>
Matrix<T> padMatrix(Matrix<T>& original) {
    int newSize = nextPowerOf2(original.size);
    Matrix<T> padded(newSize);
    for (int i = 0; i < original.size; ++i) {
        for (int j = 0; j < original.size; ++j) {
            padded.elements[i][j] = original.elements[i][j];
        }
    }
    return padded;
}
```

3. Виконання LU-розкладання

**Опис:**

- Матриця розкладається на нижню трикутну матрицю L та верхню трикутну матрицю U.
- Під час обчислення перевіряється, чи є елементи на діагоналі майже нульовими для уникнення виродженості матриці.

**Код:**

```
Matrix<T> lower(size);
Matrix<T> upper(size);

// Ініціалізація
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        lower.elements[i][j] = (i == j) ? 1 : 0;
        upper.elements[i][j] = 0;
    }
}

// LU-розклад
for (int i = 0; i < originalSize; ++i) {
    for (int j = i; j < originalSize; ++j) {
        upper.elements[i][j] = inputMatrix.elements[i][j];
        for (int k = 0; k < i; ++k) {
            upper.elements[i][j] -= lower.elements[i][k] * upper.elements[k][j];
        }
    }

    if (abs(upper.elements[i][i]) < EPSILON) {
        throw runtime_error("Matrix is nearly singular and cannot be inverted.");
    }

    for (int j = i + 1; j < originalSize; ++j) {
        lower.elements[j][i] = inputMatrix.elements[j][i];
        for (int k = 0; k < i; ++k) {
            lower.elements[j][i] -= lower.elements[j][k] * upper.elements[k][i];
        }
        lower.elements[j][i] /= upper.elements[i][i];
    }
}
```

4. Використання алгоритму Штрассена для множення у ході обчислення

**Опис:**

- Для множення матриць використовується алгоритм Штрассена, що дозволяє прискорити обчислення для великих розмірів.
- Якщо матриця розміром 2×2, виконується просте множення.

**Код:**

```cpp
// Просте множення
template <typename T>
Matrix<T> simpleMultiply(Matrix<T> A, Matrix<T> B) {
    Matrix<T> C(A.size);
    for (int i = 0; i < A.size; ++i) {
        for (int j = 0; j < A.size; ++j) {
            for (int k = 0; k < A.size; ++k) {
                C.elements[i][j] += A.elements[i][k] * B.elements[k][j];
            }
        }
    }
    return C;
}

// Алгоритм Штрассена
template <typename T>
Matrix<T> strassenMultiply(Matrix<T> A, Matrix<T> B) {
    if (A.size == 2) {
        return simpleMultiply(A, B);
    }

    // Поділ матриць на підматриці та обчислення підматриць M1-M7
    // ...
}
```

5. Знаходження оберненої матриці за допомогою підстановок

**Опис:**

- Виконується обчислення оберненої матриці шляхом вирішення систем рівнянь з використанням прямих та зворотних підстановок.

**Код:**

```cpp
for (int i = 0; i < originalSize; ++i) {
    Matrix<T> identity(size);
    identity.elements[i][i] = 1;

    // Прямий хід для знаходження вектора y (Ly = e)
    Matrix<T> y(size);
    for (int j = 0; j < originalSize; ++j) {
        y.elements[j][0] = identity.elements[j][i];
        for (int k = 0; k < j; ++k) {
            y.elements[j][0] -= lower.elements[j][k] * y.elements[k][0];
        }
    }

    // Зворотний хід для знаходження вектора x (Ux = y)
    Matrix<T> x(size);
    for (int j = originalSize - 1; j >= 0; --j) {
        x.elements[j][0] = y.elements[j][0];
        for (int k = j + 1; k < originalSize; ++k) {
            x.elements[j][0] -= upper.elements[j][k] * x.elements[k][0];
        }
        x.elements[j][0] /= upper.elements[j][j];
    }

    // Заповнення стовпця оберненої матриці
    for (int j = 0; j < originalSize; ++j) {
        inverse.elements[j][i] = x.elements[j][0];
    }
}
```

## Складність алгоритму

Алгоритм знаходження оберненої матриці за допомогою LU-розкладу та алгоритму Штрассена має кілька етапів із різною обчислювальною складністю:

1. **LU-розкладання:**
   - Класичне LU-розкладання квадратної матриці розміром n×nn \times nn×n має обчислювальну складність $O(n^3)$. У нашому випадку ми обмежуємо обчислення розміром матриці перед виконанням оберненої підстановки.
   - У випадку використання доповнення матриці до найближчого розміру, що є степенем двійки, складність цього етапу залишається $O(n^3)$, де n — розмір доповненої матриці.
2. **Алгоритм Штрассена для множення:**
   - Алгоритм Штрассена має асимптотичну складність приблизно O(n^log27)≈O(n^2.81). Це забезпечує дещо кращу продуктивність порівняно з класичним множенням, яке має складність $O(n^3)$.
   - Для великих матриць використання алгоритму Штрассена дає приріст продуктивності, особливо для розмірів, що є степенем двійки.
3. **Знаходження оберненої матриці за допомогою підстановок:**
   - Виконання прямої та зворотної підстановки в обчисленні оберненої матриці має лінійну складність відносно кількості елементів матриці, тобто $O(n^2)$.

## Загальна складність

Загальна складність алгоритму визначається поєднанням усіх етапів. Якщо використовується алгоритм Штрассена для множення матриць, загальна складність знаходження оберненої матриці наближається до O(n^2.81). У випадку використання класичного підходу для множення складність становить $O(n^3)$. Для малих матриць різниця у продуктивності незначна, але для великих матриць алгоритм Штрассена забезпечує виграш у швидкодії.

## Мова реалізації алгоритму

C++

## Модулі програми

**Matrix<T>** :

- Шаблонна структура для роботи з матрицями
- Описує матрицю розміром size на size з елементами типу T**.
- Реалізує конструктор для створення матриці, деструктор для звільнення пам'яті.

**int nextPowerOf2(int n)**:

- Визначає найближчий степінь 2, більший або рівний заданому числу n.

**Matrix<T> padMatrix(Matrix<T>& original)**:

- Доповнює матрицю до найближчого розміру, що є степенем двійки (якщо необхідно).

**Matrix<T> trimMatrix(Matrix<T>& padded, int originalSize)**:

- Видаляє зайві нульові доповнення з розширеної матриці.

**Matrix<T> simpleMultiply(Matrix<T> A, Matrix<T> B)**:

- Реалізує просте множення матриць розміром size на size.

**Matrix<T> strassenMultiply(Matrix<T> A, Matrix<T> B)**:

- Виконує множення матриць за алгоритмом Штрассена.

**Matrix<T> fillSubMatrix(Matrix<T> C, Matrix<T> subMatrix, int startRow, int startCol)**:

- Заповнює підматрицю subMatrix у більшу матрицю C, починаючи з позиції startRow, startCol.

**Matrix<T> computeInverseLU(Matrix<T> inputMatrix, int originalSize)**:

- Знаходить обернену матрицю за допомогою LU-розкладу.

**int main()**:

- Відповідає за введення даних, доповнення матриці, обчислення оберненої матриці, виведення результату або повідомлення про помилку, якщо матриця є майже сингулярною.

## Інтерфейс користувача

- Введення розміру матриці.
- Введення елементів матриці.
- Виведення оберненої матриці або повідомлення про неможливість обернення.

## Тестові приклади

1.
```
Enter the size of the matrix:2
 Enter elements of the matrix:
1
2
3
4

The inverse of the matrix is:
-2.000000 1.000000
1.500000 -0.500000
```

2.
```
Enter the size of the matrix:1
 Enter elements of the matrix:
3

The inverse of the matrix is:
0.333333
```
3.
```
Enter the size of the matrix:3
 Enter elements of the matrix:
1
2
3
4
5
6
7
8
3

The inverse of the matrix is:
-1.833333 1.000000 -0.166667
1.666667 -1.000000 0.333333
-0.166667 0.333333 -0.166667
```
4.
```
Enter the size of the matrix:2
 Enter elements of the matrix:
-1
0.3
55
-6.789

The inverse of the matrix is:
0.699104 0.030893
5.663680 0.102976
```

## Висновки

Реалізовано алгоритм обчислення оберненої матриці методом LU-розкладання з використанням алгоритму Штрассена для множення матриць. Алгоритм ефективно працює для квадратних матриць довільного розміру.

## Використані літературні джерела


https://en.wikipedia.org/wiki/LU_decomposition

https://www.tutorialspoint.com/cplusplus-program-to-perform-lu-decomposition-of-any-matrix