Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»

КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

Умножение матриц

Студент: _____ Мациевский И. М.

дата, подпись Ф.И.О.

Преподаватель: _____ Волкова Л. Л.

дата, подпись Ф.И.О.

Москва, 2023

Содержание

Введение **2**

1 Аналитическая часть **3**

2 Конструкторская часть **4**

2.1 Классический алгоритм умножения матриц 4

2.2 Метод Штрассена 4

2.3 Метод Винограда 5

2.4 Функции программы 5

3 Технологическая часть **6**

4 Исследовательская часть **20**

Заключение **21**

Список используемых источников **22**

Введение

Цель лабораторной работы: провести сравнительный анализ алгоритмов умножения матриц.

Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать три алгоритма умножения матриц — классический, Винограда и Штрассена.
2. Реализовать разработанные алгоритмы.
3. Выполнить тестирование реализации разработанных алгоритмов.
4. Дать оценку сложности алгоритмов.
5. Оценить эффективность каждого алгоритма (по памяти и времени).

1 Аналитическая часть

Матрица — упорядоченный математический объект, обычно записываемый в виде прямоугольной таблицы элементов кольца/поля или другой структуры. Количество строк и столбцов задает размер этой матрицы. С матрицами можно производить различные действия, такие как транспонирование, сложение, умножения и др.

2 Конструкторская часть

2.1 Классический алгоритм умножения матриц

Пусть у нас есть две матрицы A размерности $m \times n$ и B размерности $n \times p$. Результирующая матрица будет иметь размерность $m \times p$, и её элементы C будут вычисляться следующим образом:

$C(i,k) = \sum_{j=1}^{n} A(i,j) \cdot B(j,k), \quad 1 \le i \le m, \quad 1 \le k \le p.$

1. $A(i,j)$ - элемент матрицы A в i-й строке и j-м столбце.
2. $B(j,k)$ - элемент матрицы B в j-й строке и k-м столбце.
3. $C(i,k)$ - элемент результирующей матрицы C в i-й строке и k-м столбце.

Этот процесс повторяется для всех i, k, что приводит к вычислению всех элементов матрицы C.

2.2 Метод Штрассена

Метод Штрассена представляет собой более эффективный алгоритм умножения матриц. Идея заключается в том, чтобы заменить обычные умножения матриц на более эффективные линейные комбинации элементов.

Для умножения двух матриц A и B размерности $n \times n$, метод Штрассена предлагает следующие рекуррентные формулы:

$$
\begin{aligned}
P_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}),\\
P_2 &= (A_{21} + A_{22}) \cdot B_{11},\\
P_3 &= A_{11} \cdot (B_{12} - B_{22}),\\
P_4 &= A_{22} \cdot (B_{21} - B_{11}),\\
P_5 &= (A_{11} + A_{12}) \cdot B_{22},\\
P_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}),\\
P_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}),
\end{aligned}
$$

Результирующие блоки матрицы C:

$$
\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7,\\
C_{12} &= P_3 + P_5,\\
C_{21} &= P_2 + P_4,\\
C_{22} &= P_1 - P_2 + P_3 + P_6.
\end{aligned}
$$

Таким образом, результат умножения матриц A и B получается из линейных комбинаций блоков $P_1, P_2, \ldots, P_7$.

2.3 Метод Винограда

Алгоритм Винограда включает следующие шаги.

1. Создать новую матрицу C размерностью $m \times n$ и заполните ее нулями.

2. Создать вспомогательные массивы row_factor и col_factor для оптимизации вычислений. Их размерность равна числу строк матрицы A и числу столбцов матрицы B соответственно.

3. Вычислить $row\_factor[i]$ как сумму произведений пар элементов из $A[i]$, умноженных на соответствующие элементы из $A[i]$.

4. Вычислите $col\_factor[j]$ как сумму произведений пар элементов из $B[j]$, умноженных на соответствующие элементы из $B[j]$.

5. Используя предварительно вычисленные row_factor и col_factor, вычислите каждый элемент матрицы C следующим образом:

$$C[i][j] = rc\_f + \sum_{k=0}^{k-1}(A[i][k] + B[k][j]) \cdot (A[i][k+1] + B[k+1][j]), \text{ где}$$

$$rc\_f = -row\_factor[i] - col\_factor[j]$$

6. Если размерность матрицы нечетна, нужно добавить к соответствующему элементу C произведение соответствующих элементов из A и B, которые не участвовали в основных вычислениях.

2.4 Функции программы

Требуется разработать следующие функции.

— Функция “matrixSum” — функция, которая складывает матрицы.

— Функция “matrixMultiplication” — функция, которая перемножает две матрицы.

— Функция “strassenMatrixMultiply” — функция для умножения матриц методом Штрассена.

— Функция “classicMatrixMultiply” — функция для умножения матриц классическим способом.

— Функция “matrixMultiplyVinograd” — Функция для умножения матриц с использованием алгоритма Винограда.

— Функция “showMatrix” — функция для вывода матрицы.

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1)

Листинг 1 – Исходный код

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

// Функция для сложения матриц
vector<vector<int>> matrixSum(const vector<vector<int>>& A, const
   vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] + B[i][j];
        }
    }

    return result;
}

// Функция для вычитания матриц
vector<vector<int>> matrixMultiplication(const vector<vector<int>>& A,
   const vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> result(n, vector<int>(n, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result[i][j] = A[i][j] - B[i][j];
        }
    }

    return result;
}

// Функция для умножения матриц методом Штрассена
```

```
vector<vector<int>> strassenMatrixMultiply(const vector<vector<int>>& A
    , const vector<vector<int>>& B) {
    int n = A.size();

    if (n <= 2) {
        vector<vector<int>> result(n, vector<int>(n, 0));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        return result;
    }

    int newSize = n;
    if (n % 2 != 0) {
        newSize++;
    }

    vector<vector<int>> newA(newSize, vector<int>(newSize, 0));
    vector<vector<int>> newB(newSize, vector<int>(newSize, 0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            newA[i][j] = A[i][j];
            newB[i][j] = B[i][j];
        }
    }

    vector<vector<int>> A11(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> A12(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> A21(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> A22(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> B11(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> B12(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> B21(newSize / 2, vector<int>(newSize / 2, 0));
    vector<vector<int>> B22(newSize / 2, vector<int>(newSize / 2, 0));
    for (int i = 0; i < newSize / 2; i++) {
        for (int j = 0; j < newSize / 2; j++) {
```

```
            A11[i][j] = newA[i][j];
            A12[i][j] = newA[i][j + newSize / 2];
            A21[i][j] = newA[i + newSize / 2][j];
            A22[i][j] = newA[i + newSize / 2][j + newSize / 2];

            B11[i][j] = newB[i][j];
            B12[i][j] = newB[i][j + newSize / 2];
            B21[i][j] = newB[i + newSize / 2][j];
            B22[i][j] = newB[i + newSize / 2][j + newSize / 2];
        }
    }

    vector<vector<int>> M1 = strassenMatrixMultiply(matrixSum(A11, A22)
, matrixSum(B11, B22));
    vector<vector<int>> M2 = strassenMatrixMultiply(matrixSum(A21, A22)
, B11);
    vector<vector<int>> M3 = strassenMatrixMultiply(A11,
matrixMultiplication(B12, B22));
    vector<vector<int>> M4 = strassenMatrixMultiply(A22,
matrixMultiplication(B21, B11));
    vector<vector<int>> M5 = strassenMatrixMultiply(matrixSum(A11, A12)
, B22);
    vector<vector<int>> M6 = strassenMatrixMultiply(
matrixMultiplication(A21, A11), matrixSum(B11, B12));
    vector<vector<int>> M7 = strassenMatrixMultiply(
matrixMultiplication(A12, A22), matrixSum(B21, B22));

    vector<vector<int>> C11 = matrixSum(matrixMultiplication(matrixSum(
M1, M4), M5), M7);
    vector<vector<int>> C12 = matrixSum(M3, M5);
    vector<vector<int>> C21 = matrixSum(M2, M4);
    vector<vector<int>> C22 = matrixSum(matrixSum(matrixMultiplication(
M1, M2), M3), M6);

    vector<vector<int>> result(newSize, vector<int>(newSize));
    for (int i = 0; i < newSize / 2; i++) {
        for (int j = 0; j < newSize / 2; j++) {
            result[i][j] = C11[i][j];
            result[i][j + newSize / 2] = C12[i][j];
            result[i + newSize / 2][j] = C21[i][j];
            result[i + newSize / 2][j + newSize / 2] = C22[i][j];
```

```
        }
    }

    // Матрица без лишних нулей
    vector<vector<int>> finalResult(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            finalResult[i][j] = result[i][j];
        }
    }

    return finalResult;
}

// Функция для умножения матриц классическим способом
vector<vector<int>> classicMatrixMultiply(const vector<vector<int>>&
   matA, const vector<vector<int>>& matB) {

    if (matA[0].size() != matB.size()) {
        cout << "Невозможно умножить матрицы: неподходящие размеры" << endl;
        return {};
    }

    vector<vector<int>> result(matA.size(), vector<int>(matB[0].size(),
    0));

    for (int i = 0; i < matA.size(); i++) {
        for (int j = 0; j < matB[0].size(); j++) {
            for (int k = 0; k < matA[0].size(); k++) {
                result[i][j] += matA[i][k] * matB[k][j];
            }
        }
    }

    return result;
}

// Функция для умножения матриц с использованием алгоритма Винограда
vector<vector<int>> matrixMultiplyVinograd(const vector<vector<int>>&
   matA, const vector<vector<int>>& matB) {
    int rowsA = matA.size();
```

```
    int colsA = matA[0].size();
    int rowsB = matB.size();
    int colsB = matB[0].size();

    if (colsA != rowsB) {
        cout << "Невозможно умножить матрицы: неподходящие размеры" << endl;
        return {};
    }

    vector<vector<int>> result(rowsA, vector<int>(colsB, 0));

    // Предварительные вычисления для оптимизации
    vector<int> rowFactor(rowsA, 0);
    vector<int> colFactor(colsB, 0);
    for (int i = 0; i < rowsA; i++) {
        for (int k = 0; k < colsA - 1; k += 2) {
            rowFactor[i] += matA[i][k] * matA[i][k + 1];
        }
    }
    for (int j = 0; j < colsB; j++) {
        for (int k = 0; k < rowsB - 1; k += 2) {
            colFactor[j] += matB[k][j] * matB[k + 1][j];
        }
    }

    // Основной цикл умножения с использованием предварительных вычислений
    for (int i = 0; i < rowsA; i++) {
        for (int j = 0; j < colsB; j++) {
            result[i][j] = -rowFactor[i] - colFactor[j];
            for (int k = 0; k < colsA - 1; k += 2) {
                result[i][j] += (matA[i][k] + matB[k + 1][j]) * (matA[i
][k + 1] + matB[k][j]);
            }
            if (colsA % 2 != 0) {
                result[i][j] += matA[i][colsA - 1] * matB[rowsB - 1][j
];
            }
        }
    }

    return result;
```

```
}

// Функция для вывода матрицы
void showMatrix(const vector<vector<int>>& mat) {
    for (const auto& row : mat) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
}

int main(){
    setlocale(0, "");
    cout << "Каким методом Вы хотите перемножить матрицы?" << endl;
    cout << "Классический метод - 1" << endl;
    cout << "Метод Винограда - 2" << endl;
    cout << "Метод Штрассена - 3" << endl;
    int choose = 0;
    double medtime = 0;
    cin >> choose;

    if (choose == 1) {

        vector<vector<int>> matrixA = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}
    };
        vector<vector<int>> matrixB = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1}
    };

        cout << "Матрица A:" << endl;
        showMatrix(matrixA);
        cout << endl;

        cout << "Матрица B:" << endl;
        showMatrix(matrixB);
        cout << endl;

        clock_t start_time = clock();
        vector<vector<int>> result = classicMatrixMultiply(matrixA,
   matrixB);
        clock_t end_time = clock();
```

```
        double cpu_time = (double(end_time - start_time)) /
CLOCKS_PER_SEC;
        medtime += cpu_time;
        cout << "Результат умножения классическим методом:" << endl;
        showMatrix(result);
        cout << "Процессорное время: " << cpu_time << endl;
        cout << endl;
    }
    else if (choose == 2) {

        vector<vector<int>> matrixA = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}
    };
        vector<vector<int>> matrixB = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1}
    };

        cout << "Матрица A:" << endl;
        showMatrix(matrixA);
        cout << endl;

        cout << "Матрица B:" << endl;
        showMatrix(matrixB);
        cout << endl;

        clock_t start_time = clock();
        vector<vector<int>> result = matrixMultiplyVinograd(matrixA,
matrixB);
        clock_t end_time = clock();
        double cpu_time = (double(end_time - start_time)) /
CLOCKS_PER_SEC;
        medtime += cpu_time;
        cout << "Результат умножения с помощью алгоритма Винограда:" << endl;
        showMatrix(result);
        cout << "Процессорное время: " << cpu_time << endl;
        cout << endl;
    }
    else if (choose == 3) {
        vector<vector<int>> matrixA = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}
    };
        vector<vector<int>> matrixB = { {9, 8, 7}, {6, 5, 4}, {3, 2, 1}
    };
        cout << "Матрица A:" << endl;
```

```
            showMatrix(matrixA);
            cout << endl;
            cout << "Матрица B:" << endl;
            showMatrix(matrixB);
            cout << endl;

            clock_t start_time = clock();
            vector<vector<int>> result = strassenMatrixMultiply(matrixA,
    matrixB);
            clock_t end_time = clock();
            double cpu_time = (double(end_time - start_time)) /
    CLOCKS_PER_SEC;
            medtime += cpu_time;
            cout << "Результат умножения с помощью алгоритма Штрассена:" << endl;
            showMatrix(result);
            cout << "Процессорное время: " << cpu_time << endl;
            cout << endl;
        }
        else {
            cout << "Неверный выбор" << endl;
        }
        return 0;
}
```

Примеры работы

На рисунках 1—7 представлены примеры работы описаннных выше алгоритмов.

1. На вход подаются две матрицы размерности 3×3, программа выводит их произведение, рассчитанное по классическому алгоритму. Результат приведён на рис. 1.

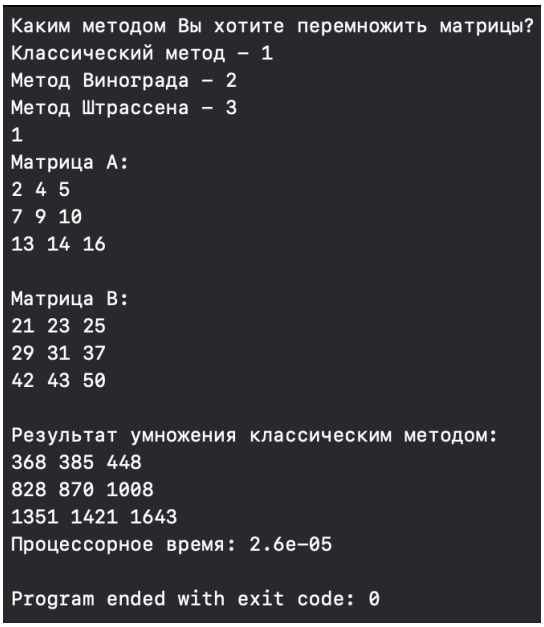

Рис. 1 – Пример работы 1

2. На вход подаются две матрицы размерности 3×3, программа выводит их произведение, рассчитанное по методу Винограда. Результат приведён на рис. 2.

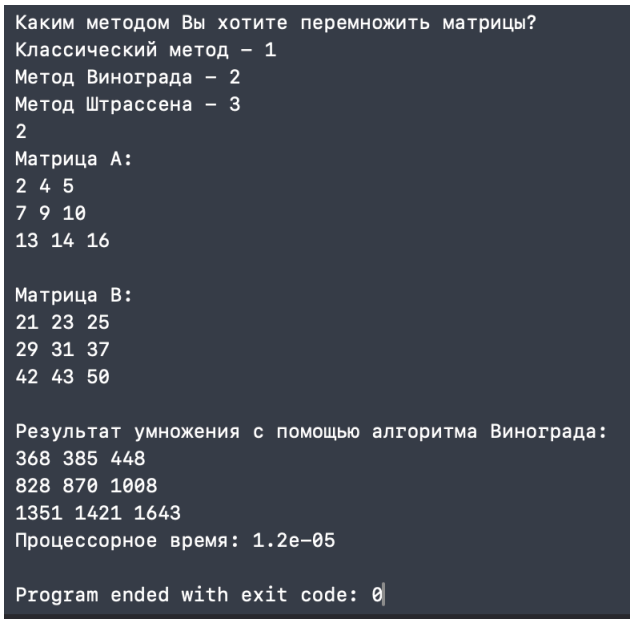

Рис. 2 – Пример работы 2

3. На вход подаются две матрицы размерности 3×3, программа выводит их произведение, рассчитанное по методу Штрассена. Результат приведён на рис. 3.

```
Каким методом Вы хотите перемножить матрицы?
Классический метод – 1
Метод Винограда – 2
Метод Штрассена – 3
3
Матрица A:
2 4 5
7 9 10
13 14 16

Матрица B:
21 23 25
29 31 37
42 43 50

Результат умножения с помощью алгоритма Штрассена:
368 385 448
828 870 1008
1351 1421 1643
Процессорное время: 0.000214

Program ended with exit code: 0
```

Рис. 3 – Пример работы 3

4. На вход подаются две матрицы размерности 6×6, программа выводит их произведение, рассчитанное по классическому алгоритму. Результат приведён на рис. 4.

```
Каким методом Вы хотите перемножить матрицы?
Классический метод – 1
Метод Винограда – 2
Метод Штрассена – 3
1
Матрица A:
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16

Матрица B:
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50

Результат умножения классическим методом:
896 1026 1183 2255 2385 2756
1584 1783 2055 3269 3468 4007
2386 2676 3083 4561 4851 5602
2255 2385 2756 896 1026 1183
3269 3468 4007 1584 1783 2055
4561 4851 5602 2386 2676 3083
Процессорное время: 3.4e-05

Program ended with exit code: 0
```

Рис. 4 – Пример работы 4

5. На вход подаются две матрицы размерности 6×6, программа выводит их произведение, рассчитанное по методу Винограда. Результат приведён на рис. 5.

```
Каким методом Вы хотите перемножить матрицы?
Классический метод – 1
Метод Винограда – 2
Метод Штрассена – 3
2
Матрица A:
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16

Матрица B:
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50

Результат умножения с помощью алгоритма Винограда:
896 1026 1183 2255 2385 2756
1584 1783 2055 3269 3468 4007
2386 2676 3083 4561 4851 5602
2255 2385 2756 896 1026 1183
3269 3468 4007 1584 1783 2055
4561 4851 5602 2386 2676 3083
Процессорное время: 2.9e-05

Program ended with exit code: 0
```

Рис. 5 – Пример работы 5

6. На вход подаются две матрицы размерности 6×6, программа выводит их произведение, рассчитанное по методу Штрассена. Результат приведён на рис. 6.

```
Каким методом Вы хотите перемножить матрицы?
Классический метод – 1
Метод Винограда – 2
Метод Штрассена – 3
3
Матрица A:
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16

Матрица B:
21 23 25 2 4 5
29 31 37 7 9 10
42 43 50 13 14 16
2 4 5 21 23 25
7 9 10 29 31 37
13 14 16 42 43 50

Результат умножения с помощью алгоритма Штрассена:
896 1026 1183 2255 2385 2756
1584 1783 2055 3269 3468 4007
2386 2676 3083 4561 4851 5602
2255 2385 2756 896 1026 1183
3269 3468 4007 1584 1783 2055
4561 4851 5602 2386 2676 3083
Процессорное время: 0.000853

Program ended with exit code: 0
```

Рис. 6 – Пример работы 6

7. На вход подаются две матрицы, одна размерности 6×6, другая размерности 6×3, программа выводит ошибку. Результат приведён на рис. 7.

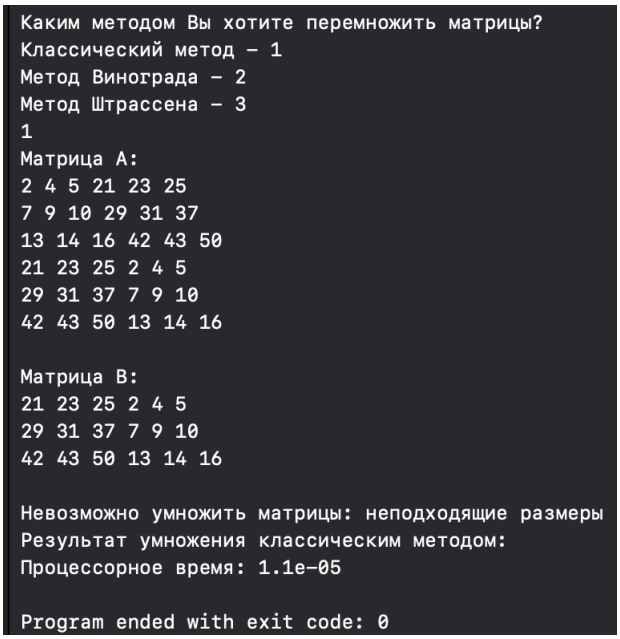

Рис. 7 – Пример работы 7

4 Исследовательская часть

Анализ эффективности. Классический способ умножения матриц имеет временную сложность $O(n^3)$, где n — размерность матрицы. Это означает, что время выполнения операции умножения матриц линейно зависит от куба размера матрицы. Как результат, общее количество операций для умножения двух матриц размером $m \times p$ и $p \times n$ составляет $m \cdot p \cdot n$ умножений и $m \cdot n \cdot (p-1)$ сложений.

Умножение матриц с помощью метода Винограда дает больший выигрыш во временной сложности. Она составляет $O(n^{2,3755})$, где n — размерность матрицы.

Метод Штрассена умножает матрицы за время $O(n^{\log_2 7}) = O(n^{2.81})$, что даёт выигрыш на больших плотных матрицах. Если обозначить количество операций умножения как $M(n)$, то для обычного метода умножения матриц размера $2^n \times 2^n$ имеем $M(n) = 8^n$. Алгоритм Штрассена снижает количество операций умножения до $M(n) = 7^n$, что означает уменьшение сложности. Он достигает этого за счет уменьшения числа стандартных умножений матриц.

Заключение

Цель лабораторной работы достигнута. Проведён сравнительный анализ алгоритмов умножения матриц.

В результате выполнения лабораторной работы были выполнены все задачи.

1. Описаны три алгоритма умножения матриц — классический, Винограда и Штрассена.
2. Реализованы разработанные алгоритмы.
3. Выполнено тестирование реализации разработанных алгоритмов.
4. Дана оценка сложности алгоритмов.
5. Оценена эффективность каждого алгоритма (по памяти и времени).

Список литературы

1. Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн - Алгоритмы. Построение и анализ. Издание 3-е. Издательство: Диалектика, 2020 г. - 1328 с.