Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 2

Перемножение матриц

Бутолин Александр Алексеевич
Студент группы ИУ7-52

2018 г.

Введение

В связи с возрастающей потребностью решать задачи, связанные с обработкой матриц, такие как расчет новых координат тела в пространстве, растет необходимость в эффективных алгоритмах по работе с ними. Перемножение матриц - одна из стандартных и наиболее используемых операций над матрицами, поэтому существует несколько алгоритмов, позволяющих произвести подобные вычисления. В данной работе требуется рассмотреть классический алгоритм и алгоритм Винограда для умножения матриц, а также провести их сравнительный анализ.

Цель работы: изучить алгоритмы умножения матриц (классический и Винограда), а также провести сравнительный анализ.

Задачи:

1. Реализовать алгоритмы: классическое умножение матриц, умножение алгоритмом Винограда, умножение улучшенным алгоритмом Винограда;
2. Провести замеры скорости выполнения алгоритмов;
3. Провести замеры памяти;
4. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитический раздел

В этом разделе описаны алгоритмы, использованные в данной лабораторной работе.

1.1 Описание алгоритмов

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется их произведением. Рассмотрим стандартный алгоритм перемножения двух матриц. Пусть есть две матрицы A и B размера $a \cdot b$ и $c \cdot d$ соответственно. Тогда, результатом из умножения будет матрица C размером $a \cdot d$, имеющая вид(1):

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1d} \\ c_{21} & c_{22} & \dots & c_{2d} \\ \dots & \dots & \dots & \dots \\ c_{a1} & c_{a2} & \dots & c_{ad} \end{bmatrix} \tag{1}$$

Каждый элемент матрицы (1) представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Часть вычислений можно просчитать заранее. Рассмотрим два вектора:

$$V = (v_1, v_2, v_3, v_4) \tag{2}$$

и

$$W = (w_1, w_2, w_3, w_4) \tag{3}$$

Их скалярное произведение:

$$V * W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4. \tag{4}$$

Это равенство можно переписать в виде:

$$V * W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{5}$$

Несмотря на то, что выражение (5) требует больше вычисления, чем (4), выражение в правой части последнего равенства (5) допускает предварительную обработку. Части этого выражения можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. В этом и заключается алгоритм Винограда.

В настоящее время умножение матриц активно применяется при решения задач:

1. Касающихся машинного обучения;
2. Преобразования координат тела на плоскости или в пространстве.

1.2 Вывод

Существует несколько возможных алгоритмов перемножения матриц. Это дает возможность произвести их сравнение для определения их преимуществ и недостатков.

2 Конструкторский раздел

Сложность стандартного перемножения матриц: O(n^3)
Для реализации алгоритмов были проведены следующие действия.

2.1 Схемы алгоритмов

На рисунках представлены схемы реализуемых алгоритмов.

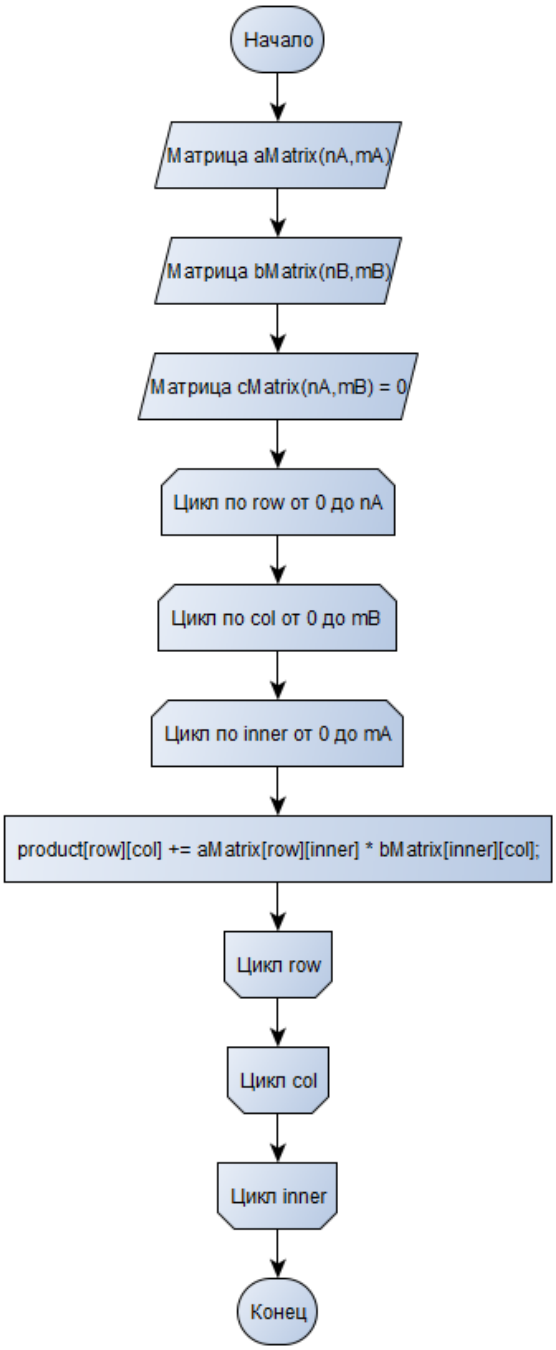
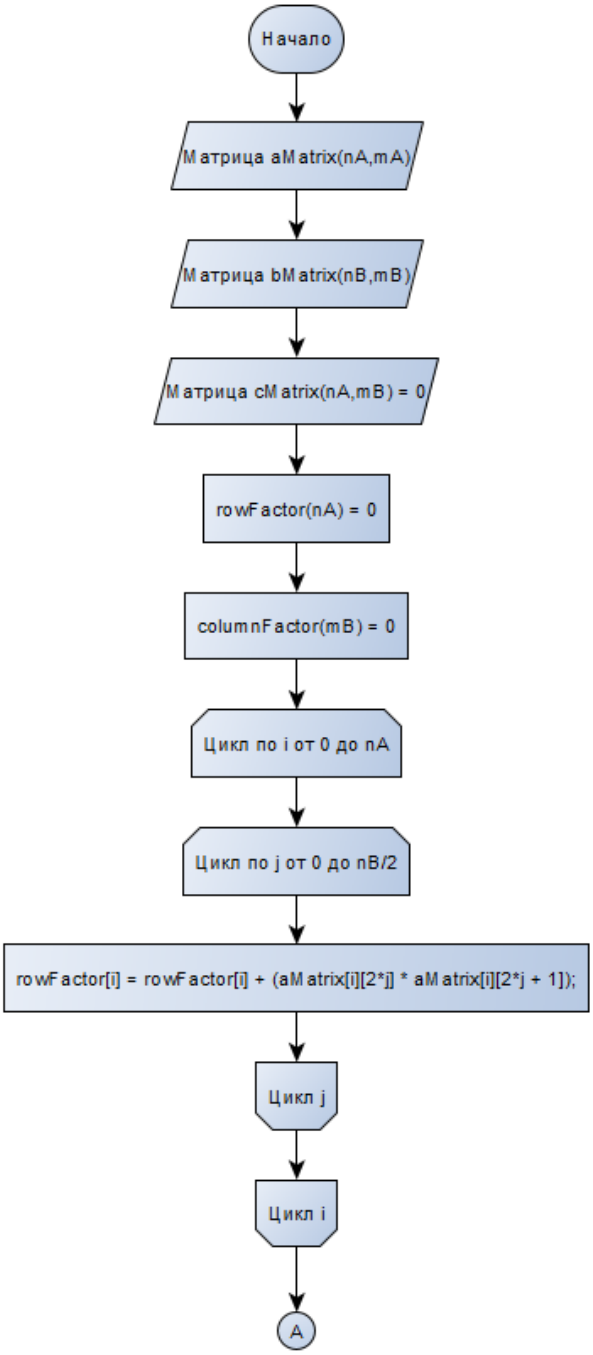
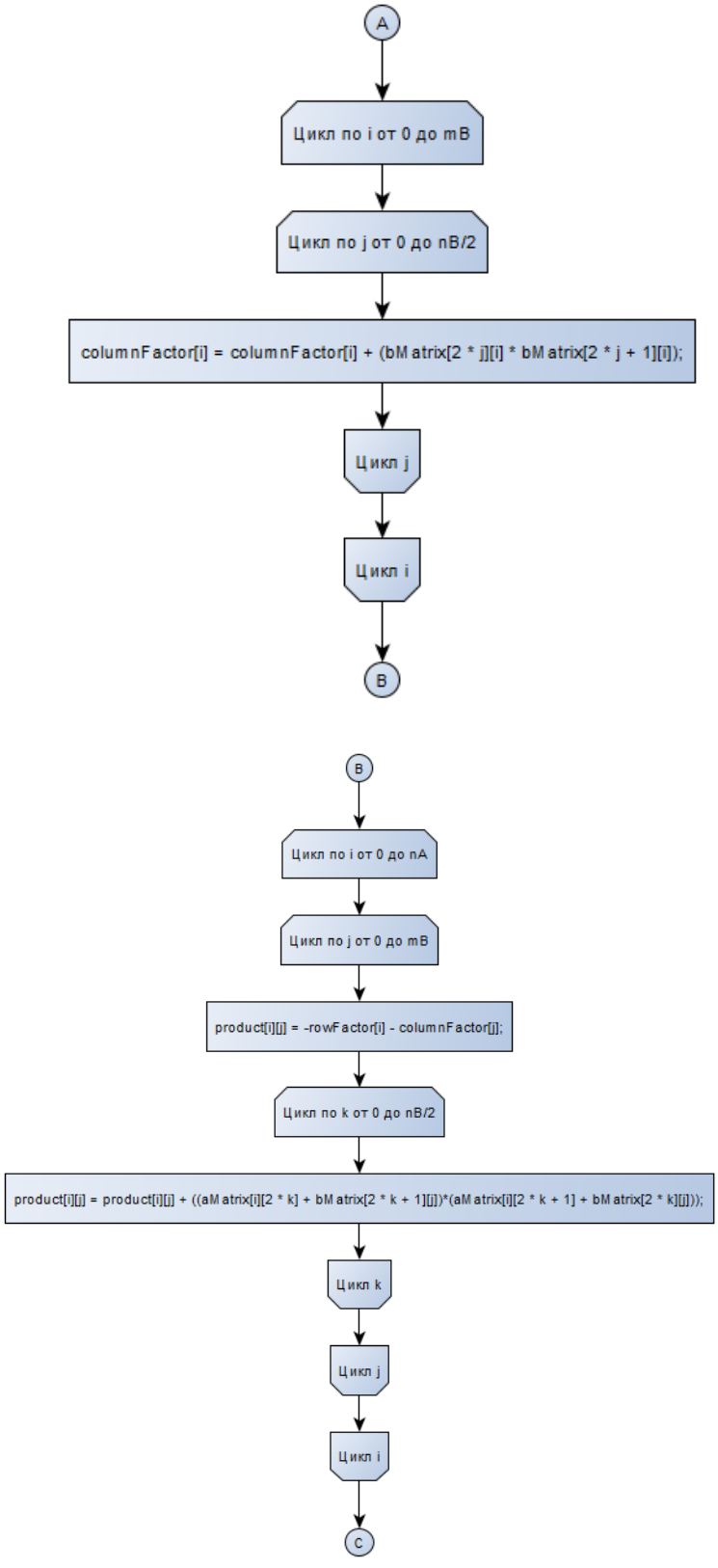


Рис. 1: Классический алгоритм перемножения матриц


Начало
Матрица aMatrix(nA,mA)
Матрица bMatrix(nB,mB)
Матрица cMatrix(nA,mB) = 0
rowFactor(nA) = 0
columnFactor(mB) = 0
Цикл по i от 0 до nA
Цикл по j от 0 до nB/2
rowFactor[i] = rowFactor[i] + (aMatrix[i][2*j] * aMatrix[i][2*j + 1]);
Цикл j
Цикл i
A


A
Цикл по i от 0 до mB
Цикл по j от 0 до nB/2
columnFactor[i] = columnFactor[i] + (bMatrix[2 * j][i] * bMatrix[2 * j + 1][i]);
Цикл j
Цикл i
B
B
Цикл по i от 0 до nA
Цикл по j от 0 до mB
product[i][j] = -rowFactor[i] - columnFactor[j];
Цикл по k от 0 до nB/2
product[i][j] = product[i][j] + ((aMatrix[i][2 * k] + bMatrix[2 * k + 1][j])*(aMatrix[i][2 * k + 1] + bMatrix[2 * k][j]));
Цикл k
Цикл j
Цикл i
C

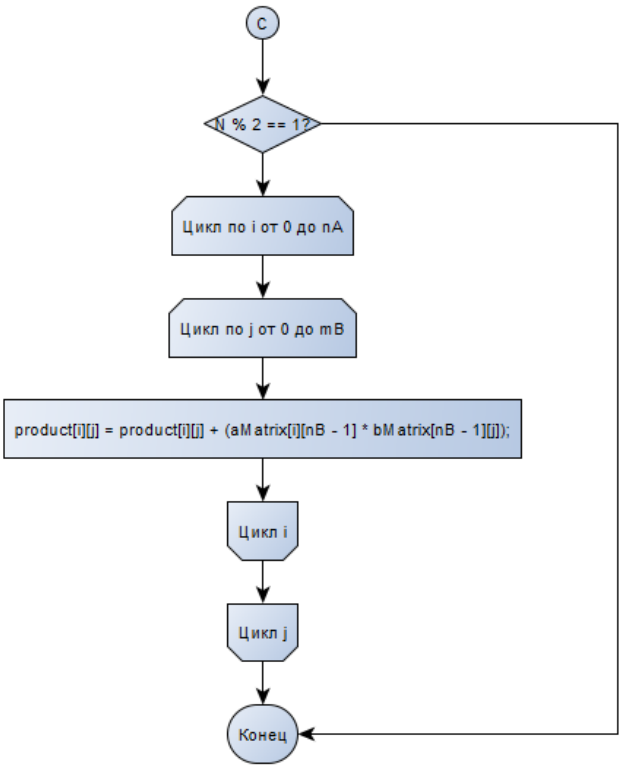


Рис. 2: Алгоритм Винограда перемножения матриц

2.2 Сравнительный анализ стандартного алгоритма перемножения матриц и алгоритма Винограда

Как видно из Рис. 2 алгоритм Винограда можно оптимизировать, так как некоторые операции можно вычислять заранее. Если же сравнивать с классическим алгоритмом, можно сделать предположение, что алгоритм Винограда будет работать медленнее, чем стандартный, из-за наличия дополнительных циклов. Так же, алгоритм Винограда использует дополнительные массивы, что должно увеличить объем потребляемой памяти. Однако можно улучшить алгоритм Винограда путём использования побитового сдвига при обращении к элементам массива, замены оператора a = a + c на a += c и объединения последних двух вложенных циклов в один условный оператор.

3 Технологический раздел

В данном разделе будет приведена информация о конкретной реализации приведенных выше алгоритмов, а также исходный код полученных методов.

3.1 Требования к программному обеспечению

К программному обеспечению предъявлены следующие требования:

1. Возможность ввода матриц;
2. Возможность вывода результатов всех трех алгоритмов;
3. Возможность вывода замеров времени, затраченного на работу алгоритмов.

3.2 Средства реализации

Лабораторная работа была выполнена в VisualStudio2017 на языке C++(c17) #. Замеры процессорного времени были произведены с помощью библиотеки intrin.h.

3.3 Листинг кода

Ниже приведены листинги реализованных методов.

```
#include "Header.h"
#include "ioMatrix.h"

using namespace std;

vector <vector<int>> multiplyStandart(vector <vector <int>> aMatrix, \
                                      vector <vector <int>> bMatrix, \
                                        int nA, int mA, int nB, int mB)
{
        vector <vector <int>> product(nA);

        for (int i = 0; i < nA; ++i)
        {
                product[i].resize(mB);
        }

        for (int row = 0; row < nA; row++)
        {
                for (int col = 0; col < mB; col++)
                {
                        for (int inner = 0; inner < mA; inner++)
                        {
                                product[row][col] += aMatrix[row][inner] * \
                                                     bMatrix[inner][col];
                        }
                }
        }
        return product;
```

```
}

vector <vector<int>> multiplyVinograd(vector <vector <int>> aMatrix, \
                                      vector <vector <int>> bMatrix, \
                                        int nA, int mA, int nB, int mB)
{
        int d;
        vector<int> rowFactor(nA);
        vector<int> columnFactor(mB);

        vector<vector<int>> product(nA);
        for (int i = 0; i < nA; ++i)
        {
                product[i].resize(mB);
        }

        d = nB / 2;

        for (int i = 0; i < nA; ++i)
        {
                for (int j = 0; j < d; ++j)
                {
                        rowFactor[i] = rowFactor[i] + (aMatrix[i][2*j] * \
                                                       aMatrix[i][2*j + 1]);
                }
        }

        for (int i = 0; i < mB; ++i)
        {
                for (int j = 0; j < d; ++j)
                {
                        columnFactor[i] = columnFactor[i] + \
                                          (bMatrix[2 * j][i] * \
                                            bMatrix[2 * j + 1][i]);
                }
        }

        for (int i = 0; i < nA; ++i)
        {
                for (int j = 0; j < mB; ++j)
                {
                        product[i][j] = -rowFactor[i] - \
                                                columnFactor[j];
                        for (int k = 0; k < d; ++k)
                        {
                                product[i][j] = product[i][j] + \
                                                ( \
                                                   (aMatrix[i][2 * k] + \
                                                   bMatrix[2 * k + 1][j]) \
                                                       * \
                                                   (aMatrix[i][2 * k + 1] + \
```

```
                                                        bMatrix[2 * k][j]) \
                                                    );
                        }
                }
        }

        if (nB % 2 != 0)
        {
                for (int i = 0; i < nA; ++i)
                {
                        for (int j = 0; j < mB; ++j)
                        {
                                product[i][j] = product[i][j] + \
                                                (aMatrix[i][nB - 1] * \
                                                    bMatrix[nB - 1][j]);
                        }
                }
        }

        return product;
}

vector <vector<int>> multiplyVinogradImprove(vector <vector <int>> aMatrix, \
                                             vector <vector <int>> bMatrix, \
                                              int nA, int mA, int nB, int mB)
{
        int d;
        vector<int> rowFactor(nA);
        vector<int> columnFactor(mB);

        vector<vector<int>> product(nA);
        for (int i = 0; i < nA; ++i)
        {
                product[i].resize(mB);
        }

        d = nB / 2;

        for (int i = 0; i < nA; ++i)
        {
                for (int j = 0; j < d; ++j)
                {
                        rowFactor[i] += (aMatrix[i][(j<<1)] * \
                                                aMatrix[i][(j<<1)|1]);
                }
        }

        for (int i = 0; i < mB; ++i)
        {
                for (int j = 0; j < d; ++j)
                {
```

```
                columnFactor[i] += (bMatrix[(j<<1)][i] * \
                                    bMatrix[(j<<1)|1][i]);
            }
        }

        if (nB % 2 == 0)
        {
            for (int i = 0; i < nA; ++i)
            {
                for (int j = 0; j < mB; ++j)
                {
                    product[i][j] = -rowFactor[i] - columnFactor[j];
                    for (int k = 0; k < d; ++k)
                    {
                        product[i][j] += ((aMatrix[i][(k << 1)] +
                          bMatrix[(k << 1) | 1][j])* \
                           (aMatrix[i][(k << 1) | 1] \
                                + bMatrix[(k << 1)][j]));
                    }
                }
            }
        }
        else
        {
            for (int i = 0; i < nA; ++i)
            {
                for (int j = 0; j < mB; ++j)
                {
                    product[i][j] = -rowFactor[i] - columnFactor[j];
                    for (int k = 0; k < d; ++k)
                    {
                        product[i][j] += ((aMatrix[i][(k << 1)] +
                          bMatrix[(k << 1) | 1][j])*\
                           (aMatrix[i][(k << 1) | 1] + \
                          bMatrix[(k << 1)][j]) + \
                                    (aMatrix[i][nB - 1] * \
                                      bMatrix[nB - 1][j]));
                    }
                }
            }
        }

        return product;
}
```

4 Экспериментальная часть

При реализации алгоритмов была произведена проверка правильности работы. Также был поставлен эксперимент для подтверждения утверждения о том, что алгоритм Винограда работает медленнее классического.

4.1 Примеры работы

Были проведены тесты всех трех алгоритмов для определения правильности их работы.
Тест 1. Умножение.
Матрицы для перемножения:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 10 \\ 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 & 0 \\ 5 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix}$$

Результаты, где (6) - классический алгоритм, (7) - алгоритм Винограда, и (8) - оптимизированный алгоритм Винограда:

$$Classic = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix} \quad (6) \qquad Winogr = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix} \quad (7) \qquad WinOpt = \begin{bmatrix} 12 & 7 & -5 \\ -2 & 6 & -10 \\ 0 & 0 & 0 \end{bmatrix} \quad (8)$$

Тест 2. Умножение на единичную матрицу.
Матрицы для перемножения:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix}$$

Результаты, где (9) - классический алгоритм, (10) - алгоритм Винограда, и (11) - оптимизированный алгоритм Винограда:

$$Classic = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix} \quad (9) \qquad Winogr = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix} \quad (10) \qquad WinOpt = \begin{bmatrix} 2 & 4 \\ 5 & 0 \end{bmatrix} \quad (11)$$

Тест 3. Умножение на нулевую матрицу.
Матрицы для перемножения:

$$A = \begin{bmatrix} -1 & 10 \\ 2 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Результаты, где (12) - классический алгоритм, (13) - алгоритм Винограда, и (14) - оптимизированный алгоритм Винограда:

$$Classic = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (12) \qquad Winogr = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (13) \qquad WinOpt = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad (14)$$

Тест 4. Умножение матриц 1x1.
Матрицы для перемножения:

$$A = \begin{bmatrix} -1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \end{bmatrix}$$

Ожидаемый результат:

$$C = \begin{bmatrix} -2 \end{bmatrix}$$

Результаты, где (15) - классический алгоритм, (16) - алгоритм Винограда, и (17) - оптимизированный алгоритм Винограда:

$$Classic = \begin{bmatrix} -2 \end{bmatrix} \quad (15) \qquad Winogr = \begin{bmatrix} -2 \end{bmatrix} \quad (16) \qquad WinOpt = \begin{bmatrix} -2 \end{bmatrix} \quad (17)$$

4.2 Постановка эксперимента

Замеры времени выполнялся на квадратных матрицах размера от 100x100 до 1000x1000 с шагом 100 для четной совпадающей размерности матриц, и от 101x101 до 1001x1001 для нечетной. Числа в матрицах генерировались случайным образом.

4.3 Сравнительный анализ на основе экспериментальных данных

В ходе эксперимента были получены следующие данные, представленные в таблицах 1 и 2.

Таблица 1

Время выполнения алгоритмов (тики) для четной совпадающей размерности матриц

	Классический	Виноград	Оптимизированный Виноград
100	2265522496	2191283674	1839756105
200	18050721239	17157721168	14385504825
300	61085438711	57864873949	48365939262
400	144404496803	136729162879	114924244078
500	280343572072	265272323038	222117101203
600	454419916713	468634129865	401878517779
700	727826428503	749552179893	616871235096
800	1077996503537	1108881350759	920487039006
900	1537402124157	1579516929324	1311851684624
1000	2116661239335	2174911323641	1808386641669

Таблица 2

Время выполнения алгоритмов (тики) для нечетной совпадающей размерности матриц

	Классический	Виноград	Оптимизированный Виноград
101	2386501394	2227751207	1858600569
201	18301399540	17475126873	14866810382
301	61521489095	58313321263	48771905492
401	145427694448	139823847963	116094472809
501	265299403461	272700662438	226141044118
601	457003063880	470818479006	390446446278
701	724576583594	745894605863	619565551412
801	1082851897339	1113222144388	925093898256
901	1540592533368	1586115321978	1315880047558

По полученным данным были построены графики, представленные на рисунках 4 и 3.

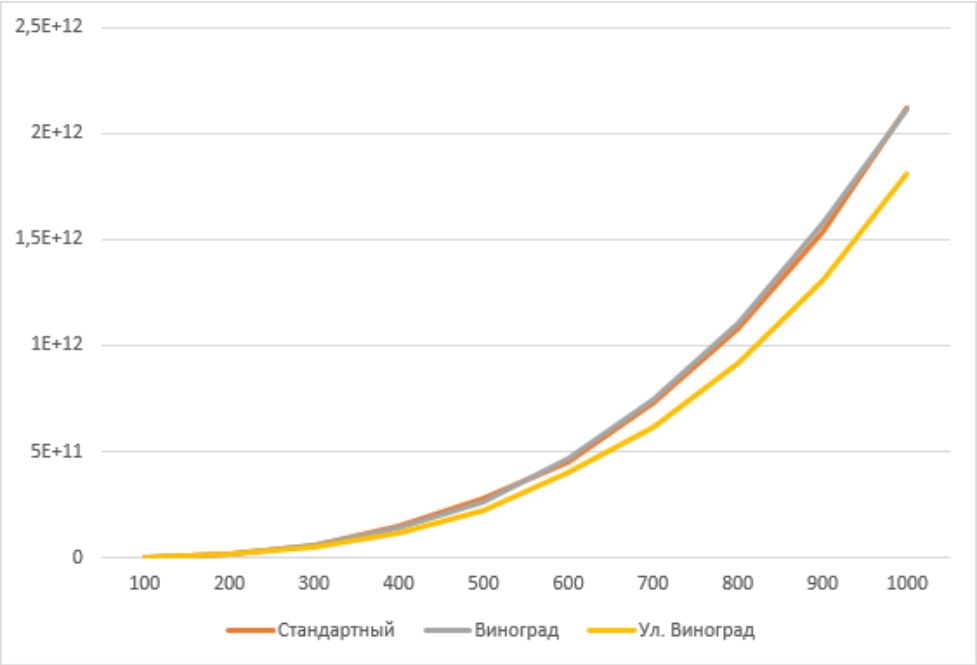


Рис. 3: Зависимость времени выполнения(тики) от размеров таблицы для четной размерности

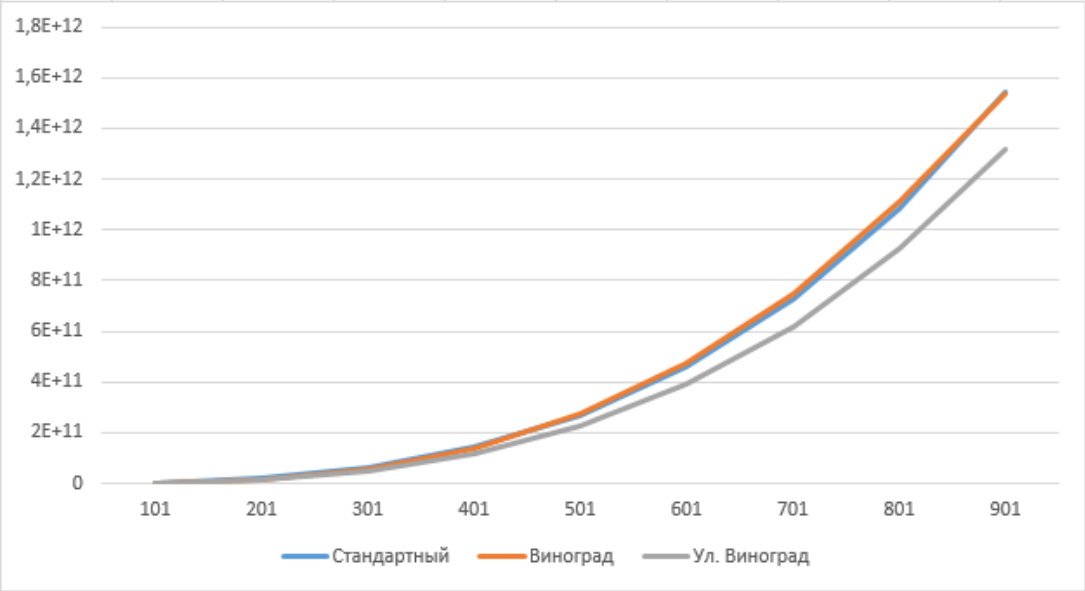


Рис. 4: Зависимость времени выполнения(тики) от размеров таблицы для нечетной размерности

4.4 Вывод

Из данных графиков можно сделать вывод, что алгоритм Винограда и классический алгоритм работают приерно одинаково. Также было показано, что произведенные оптимизации алгоритма Винограда улучшили показатели времени работы.

Заключение

В ходе выполнения лабораторной работы было реализовано два алгоритма перемножения матриц, а также была предложена оптимизация алгоритма Винограда. Была произведена оценка трудоемкости, показавшая, что алгоритм Винограда более трудоемкий алгоритм, нежели классический алгритм. Над реализованными методами были проведены ряды тестов, продемонстрировавших правильность их работы. Результаты, полученные в экспериментальной части отчета показали, что оптимизация улучшила показатели алгоритма Винограда по времени.

Список литературы

[1] Coppersmith and Shmuel Winograd. «Journal of Symbolic Computation» - М.: Доклады Академий Наук СССР, 1965.

[2] «Алгоритм Копперсмита — Винограда» [Электронный ресурс]. Journal of Symbolic Computation. – Режим доступа: http://ru.math.wikia.com/wiki/Алгоритм Копперсмита — Винограда, свободный.

[3] Корн Г., Корн Т. Алгебра матриц и матричное исчисление // Справочник по математике. — 4-е издание. — М: Наука, 1978.