**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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# Отчет по лабораторной работе №2 по дисциплине "Анализ алгоритмов"

**Тема** Умножение матриц

**Студент** Егорова П.А.

**Группа** ИУ7-54Б

**Преподаватели** Волкова Л.Л., Строганов Ю.В.

Москва — 2022 г.

# Оглавление

# Введение

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов, который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Матрицы встречаются не только в деятельности науки, но и в повседневной жизни, ведь любое представление информации в виде таблицы тоже является матрицей.

Целью работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения цели ставятся следующие задачи.

1. Изучить классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
2. Реализовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
3. Дать оценку трудоёмкости алгоритмов.
4. Замерить время работы алгоритмов.
5. Провести сравнительный анализ на основе полученных экспериментально данных.

# 1 Аналитическая часть

## 1.1 Стандартный алгоритм

К операциям, которые можно совершать над матрицами, относятся сложение, вычисание, умножение на число, возведение в степень и т.д. В один ряд с ними стоит операция умножения матриц.

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица $C$

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.3}$$

будет называться произведением матриц $A$ и $B$. Стандартный алгоритм реализует формулу 1.3.

## 1.2 Алгоритм Винограда

Можно заметить, что каждый элемент матрицы, полученной при умножении двух других, представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. А также – дабы выполнить часть работы заранее – такое умножение допускает предварительную обработку.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$.

Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (1.4)$$

Для примера, приведенного в формуле 1.4, в стандартном алгоритме производятся четыре умножения и три сложения, в алгоритме Винограда – шесть умножений и десять сложений [1]. Но, несмотря на увеличение количества операций, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с двумя предварительно вычисленными суммами соседних элементов текущих строк и столбцов. Как правило, в ЭВМ операция сложения быстрее операции умножения, поэтому алгоритм Винограда должен работать быстрее стандартного.

## 1.3 Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которого от классического алгоритма — наличие предварительной обработки, а также количество операций умножения и сложения.

# 2 Конструкторская часть

## 2.1 Схемы алгоритмов

Для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим – с чётным, из-за того что отпадает необходимость в последнем цикле.

Данный алгоритм можно оптимизировать.

- Замена выражения $x = x + k$ на $x+ = k$;
- Замена вызова функции вычисления длины строки на заранее вычисленное значение.
- Замена деления на 2 на побитовый сдвиг.

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

На рисунках 2.2 и 2.3 представлена схема алгоритма Винограда и схема оптимизированного алгоритма Винограда.

## 2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений.

1. Операции из списка (2.1) имеют трудоемкость 1.

$$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

2. Трудоемкость оператора выбора if условие then A else B рассчитывается по формуле (2.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

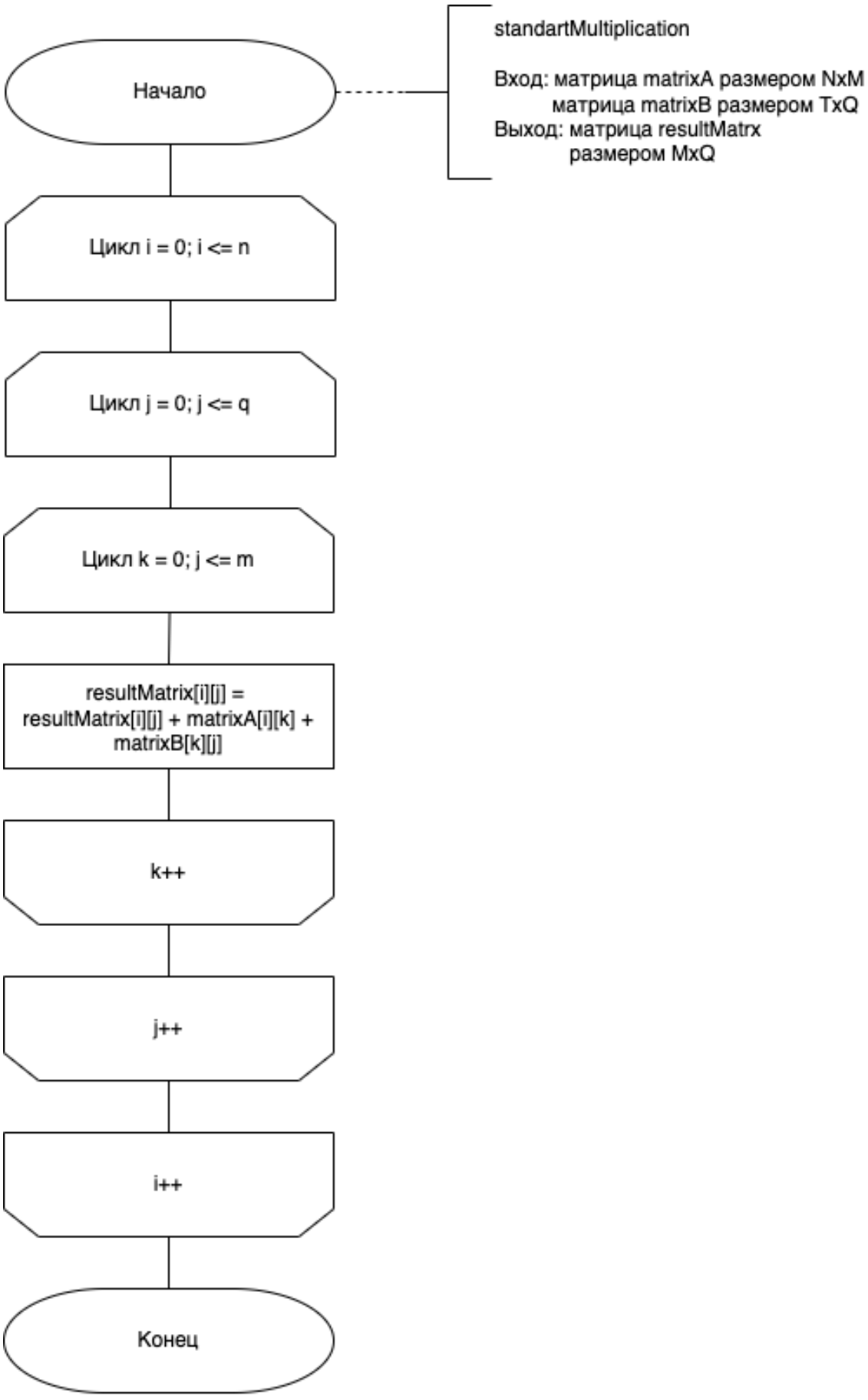


Рис. 2.1: Схема стандартного алгоритма умножения матриц

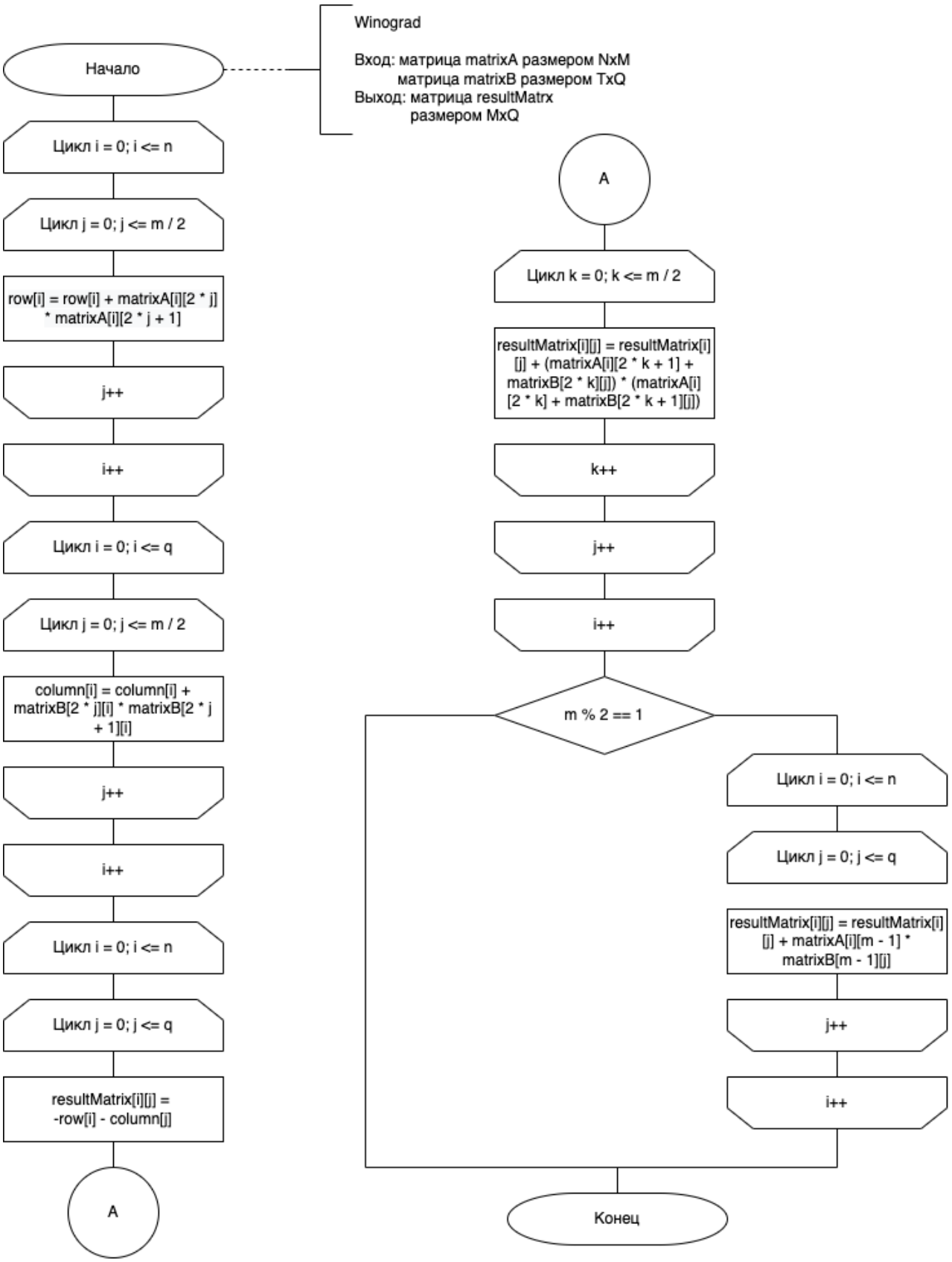


Рис. 2.2: Схема алгоритма Винограда

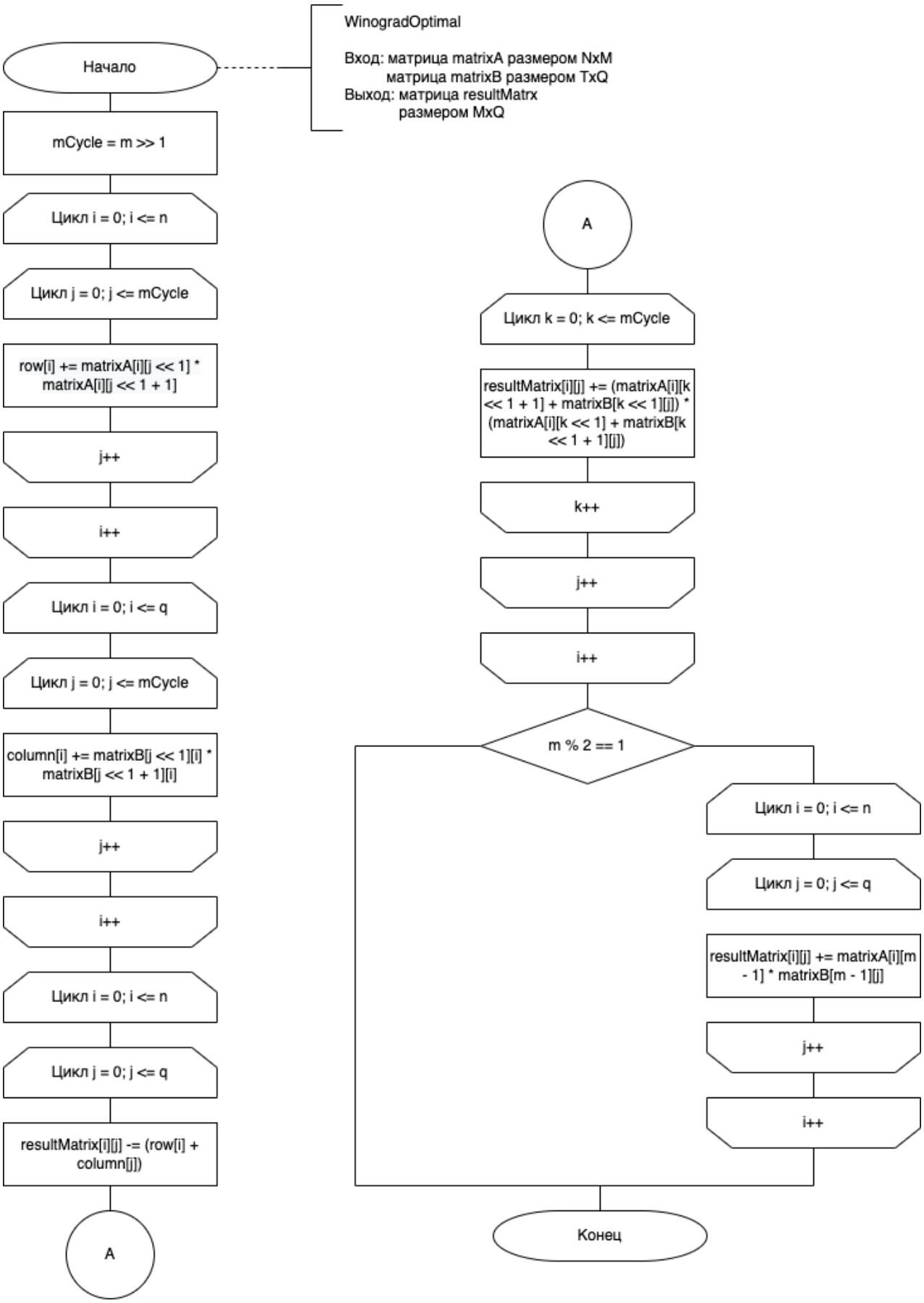


Рис. 2.3: Схема функций оптимизированного алгоритма Винограда

3. Трудоемкость цикла рассчитывается по формуле (2.3).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \quad (2.3)$$

4. Трудоемкость вызова функции равна 0.

## 2.3 Трудоёмкость алгоритмов

### 2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц можно вычислять поэтапно.

- Трудоёмкость внешнего цикла по $i \in [1..A]$ вычисляется как $f = 2 + A \cdot (2 + f_{body})$.
- Трудоёмкость цикла по $j \in [1..C]$ вычисляется как $f = 2 + C \cdot (2 + f_{body})$.
- Трудоёмкость скалярного умножения двух векторов – цикл по $k \in [1..B]$ вычисляется как $f = 2 + 10B$.

Трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела, что выполнено в формуле (2.4).

$$f_{base} = 2 + A \cdot (4 + C \cdot (4 + 10B)) = 2 + 4A + 4AC + 10ABC \approx 10ABC \quad (2.4)$$

### 2.3.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда можно вычислять поэтапно.

1. Трудоёмкость создания векторов rows и cols вычисляется как (2.5).

$$f_{create} = A + C. \quad (2.5)$$

2. Трудоёмкость заполнения вектора rows вычисляется как (2.6).

$$f_{rows} = 3 + \frac{B}{2} \cdot (5 + 12A). \tag{2.6}$$

3. Трудоёмкость заполнения вектора cols вычисляется как (2.7).

$$f_{cols} = 3 + \frac{B}{2} \cdot (5 + 12C). \tag{2.7}$$

4. Трудоёмкость цикла заполнения матрицы для чётных размеров вычисляется как (2.8).

$$f_{cycle} = 2 + A \cdot (4 + C \cdot (11 + \frac{25}{2} \cdot B)). \tag{2.8}$$

5. Трудоёмкость цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный вычисляется как (2.9):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + A \cdot (4 + 14C), & \text{иначе.} \end{cases} \tag{2.9}$$

Итого, для худшего случая (нечётный размер матриц):

$$f_{wino\_w} = A+C+12+8A+5B+6AB+6CB+25AC+\frac{25}{2}ABC \approx 12.5 \cdot MNK. \tag{2.10}$$

Для лучшего случая (чётный размер матриц):

$$f_{wino\_b} = A+C+10+4A+5B+6AB+6CB+11AC+\frac{25}{2}ABC \approx 12.5 \cdot MNK. \tag{2.11}$$

### 2.3.3 Оптимизированный алгоритм Винограда

Трудоёмкость улучшенного алгоритма Винограда можно вычислять поэтапно.

1. Трудоёмкость создания векторов rows и cols высчитывается по фор-

муле (2.12).

$$f_{init} = A + C. \tag{2.12}$$

2. Трудоёмкость заполнения вектора rows высчитывается по формуле (2.13).

$$f_{rows} = 2 + \frac{B}{2} \cdot (4 + 8A). \tag{2.13}$$

3. Трудоёмкость заполнения вектора cols высчитывается по формуле (2.14).

$$f_{cols} = 2 + \frac{B}{2} \cdot (4 + 8A). \tag{2.14}$$

4. Трудоёмкость цикла заполнения матрицы для чётных размеров высчитывается по формуле (2.15).

$$f_{cycle} = 2 + A \cdot (4 + C \cdot (8 + 9B)). \tag{2.15}$$

5. Трудоёмкость цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный высчитывается по формуле (2.16).

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + A \cdot (4 + 12C), & \text{иначе.} \end{cases} \tag{2.16}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем формулу (2.17).

$$f = A + C + 10 + 4B + 4BC + 4BA + 8A + 20AC + 9ABC \approx 9ABC. \tag{2.17}$$

Для лучшего случая (чётный общий размер матриц) имеем формулу (2.18).

$$f = A + C + 8 + 4B + 4BC + 4BA + 4A + 8AC + 9ABC \approx 9ABC. \tag{2.18}$$

## 2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы обоих алгоритмов умножения матриц. Оценены их трудоёмкости в лучшем и худшем случаях.

# 3 Технологическая часть

В данном разделе производится выбор средств реализации, а также приводятся требования к программному обеспечению (ПО), листинги реализованных алгоритмов.

## 3.1 Требования к ПО

На вход программе подаются две матрицы, на выходе должна быть получена результирующая матрица – произведение двух первых, вычисленная с помощью каждого реализованного алгоритма: стандартного, Винограда, Винограда с оптимизацией. Также необходимо вывести затраченное каждым алгоритмом процессорное время.

Интерфейс создаваемого приложения должен предоставлять возможность ввода двух матриц, задания размерностей и генерации по ним двух матриц и выбора алгоритма умножения. В качестве результата должна быть выведена результирующая матрица и наглядно представлен график работы алгоритмов: зависимость времени от количества повторений операции.

## 3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Swift [2]. Данный язык создан компанией Apple для разработки программного обеспечения для macOS, iOS, watchOS и tvOS и выступил альтернативой языку Objective-C. Он содержит в себе большое количество инструментов, которые позволяют быстро создавать интерфейс приложений, а также реализовывать различные алгоритмы. Программное обеспечение, созданное посредством swift, позволит наглядно продемонстрировать скорость работы алгоритмов, а также упростить тестирование.

Кроме того, в Swift есть фреймворк CoreFoundation [3], который предоставляет функции для замера процессорного времени.

В качестве среды разработки выбран XCode [4]. Альтернативой ему выступает среда AppCode от JetBrains [5]. Однако Xcode, являясь официальной средой разработки Apple, предоставляет возможность запуска симуляторов устройств Apple, что играет ключевую роль при создании масштабируемых приложений.

## 3.3 Листинги кода

В листингах 3.1 – 3.3 представлены реализации рассматриваемых алгоритмов.

В Листинге 3.1 приведена реализация стандартного алгоритма умножения матриц.

```
1  func standardMultiplication(_ matrixA: [[Int]], _ matrixB: [[Int]]) -> [[
       Int]] {
2
3      let n = matrixA.count
4      let m = matrixA[0].count
5      let q = matrixB[0].count
6
7      resultMatrix = createMatrix(n: n, m: q, fill: 0)
8
9      for i in 0..<n {
10         for j in 0..<q {
11             for k in 0..<m {
12                 resultMatrix[i][j] =  \
13                 resultMatrix[i][j] \
14                 + matrixA[i][k] * matrixB[k][j]
15             }
16         }
17     }
18
19
20     return resultMatrix
21 }
```

Листинг 3.1: Функция стандартного умножения матриц

В Листинге 3.2 приведена реализация реализация алгоритма Винограда умножения матриц.

```
func Winograd(_ matrixA: [[Int]], _ matrixB: [[Int]]) -> [[Int]] {
    let n = matrixA.count
    let m = matrixA[0].count
    let q = matrixB[0].count

    resultMatrix = [[Int]](repeating: [Int](repeating: 0, count: q), count: n)
    var row = [Int](repeating: 0, count: n)
    for i in 0..<n {
        for j in 0..<(m / 2) {
            row[i] = row[i] + matrixA[i][2 * j] * matrixA[i][2 * j + 1]
        }
    }

    var column = [Int](repeating: 0, count: q)
    for i in 0..<q {
        for j in 0..<(m / 2) {
            column[i] = column[i] + matrixB[2 * j][i] * matrixB[2 * j + 1][i]
        }
    }
    for i in 0..<n {
        for j in 0..<q {
            resultMatrix[i][j] = -row[i] - column[j]
            for k in 0..<(m / 2) {
                resultMatrix[i][j] = resultMatrix[i][j] + (matrixA[i][2 * k + 1] + matrixB[2 * k][j]) * (matrixA[i][2 * k] + matrixB[2 * k + 1][j])
            }
        }
    }

    if m % 2 == 1 {
        for i in 0..<n {
            for j in 0..<q {
                resultMatrix[i][j] = resultMatrix[i][j] + matrixA[i][m - 1] * matrixB[m - 1][j]
            }
        }
    }

    return resultMatrix
}
```

Листинг 3.2: Функция алгоритма Винограда умножения матриц

В Листинге 3.3 приведена реализация оптимизированного алгоритма Винограда умножения матриц.

```
func WinogradOptimal(_ matrixA: [[Int]], _ matrixB: [[Int]]) -> [[Int]] {
    let n = matrixA.count
    let m = matrixA[0].count
    let mCycle = matrixA[0].count >> 1
    let q = matrixB[0].count

    resultMatrix = [[Int]](repeating: [Int](repeating: 0, count: q), count: n)
    var row = [Int](repeating: 0, count: n)

    for i in 0..<n {
        for j in 0..<mCycle {
            row[i] += matrixA[i][2 * j] * matrixA[i][2 * j + 1]
        }
    }

    var column = [Int](repeating: 0, count: q)
    for i in 0..<q {
        for j in 0..<mCycle {
            column[i] += matrixB[2 * j][i] * matrixB[2 * j + 1][i]
        }
    }
    for i in 0..<n {
        for j in 0..<q {
            resultMatrix[i][j] -= (row[i] + column[j])
            for k in 0..<mCycle {
                resultMatrix[i][j] += (matrixA[i][2 * k + 1] + matrixB[2 * k][j]) * (matrixA[i][2 * k] + matrixB[2 * k + 1][j])
            }
        }
    }
    if m % 2 == 1 {
        for i in 0..<n {
            for j in 0..<q {
                resultMatrix[i][j] += matrixA[i][m - 1] * matrixB[m - 1][j]
            }
        }
    }
    return resultMatrix
}
```

Листинг 3.3: Функция оптимизированного алгоритма Винограда умножения матриц

## 3.4 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы умножения матриц, рассматриваемых в данной лабораторной работе. Тесты для всех алгоритмов пройдены успешно.

Таблица 3.1: Функциональные тесты

| Матрица 1 | Матрица 2 | Ожидаемый результат |
|---|---|---|
| $\begin{pmatrix} \end{pmatrix}$ | $\begin{pmatrix} \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$ | $\begin{pmatrix} \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$ | $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 & a & 3 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ a \\ 1 \end{pmatrix}$ | Сообщение об ошибке |
| $\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ |
| $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ |
| $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ | $\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$ | $\begin{pmatrix} 6 & 6 \\ 15 & 15 \\ 24 & 24 \end{pmatrix}$ |
| $\begin{pmatrix} 2 \end{pmatrix}$ | $\begin{pmatrix} 2 \end{pmatrix}$ | $\begin{pmatrix} 4 \end{pmatrix}$ |

# Вывод

В данном разделе были разработаны исходные коды четырёх алгоритмов перемножения матриц: обычный алгоритм, алгоритм Винограда, Винограда с оптимизацией. Было проведено тестирование реализаций.

# 4 Исследовательская часть

## 4.1 Интерфейс приложения

На рисунках 4.1 – 4.3 приведено изображение интерфейса экранов приложения.

Главный экран приложения дает возможность ввести две матрицы, произведение которых будет вычисляться, а также предоставляется выбор метода, с помощью которого оно будет найдено. Также предусмотрена возможность сгенерировать матрицу: для этого необходимо задать размеры матриц в специальных текстовых полях и нажать на кнопку «Генерация». При нажатии на кнопку «Рассчитать», появляется второй экран, отображающий результирующую матрицу. При нажатии на кнопку «График», находящуюся на втором экране, отображается график зависимости времени (в секундах) от количества произведенных операций.

## 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: iOS 14.5;
- оперативная память: 4 Гб;
- процессор: Apple A14 Bionic 2990 МГц [6].

Во время тестирования iPad не был подключен к другим устройствам и был включен в сеть питания.

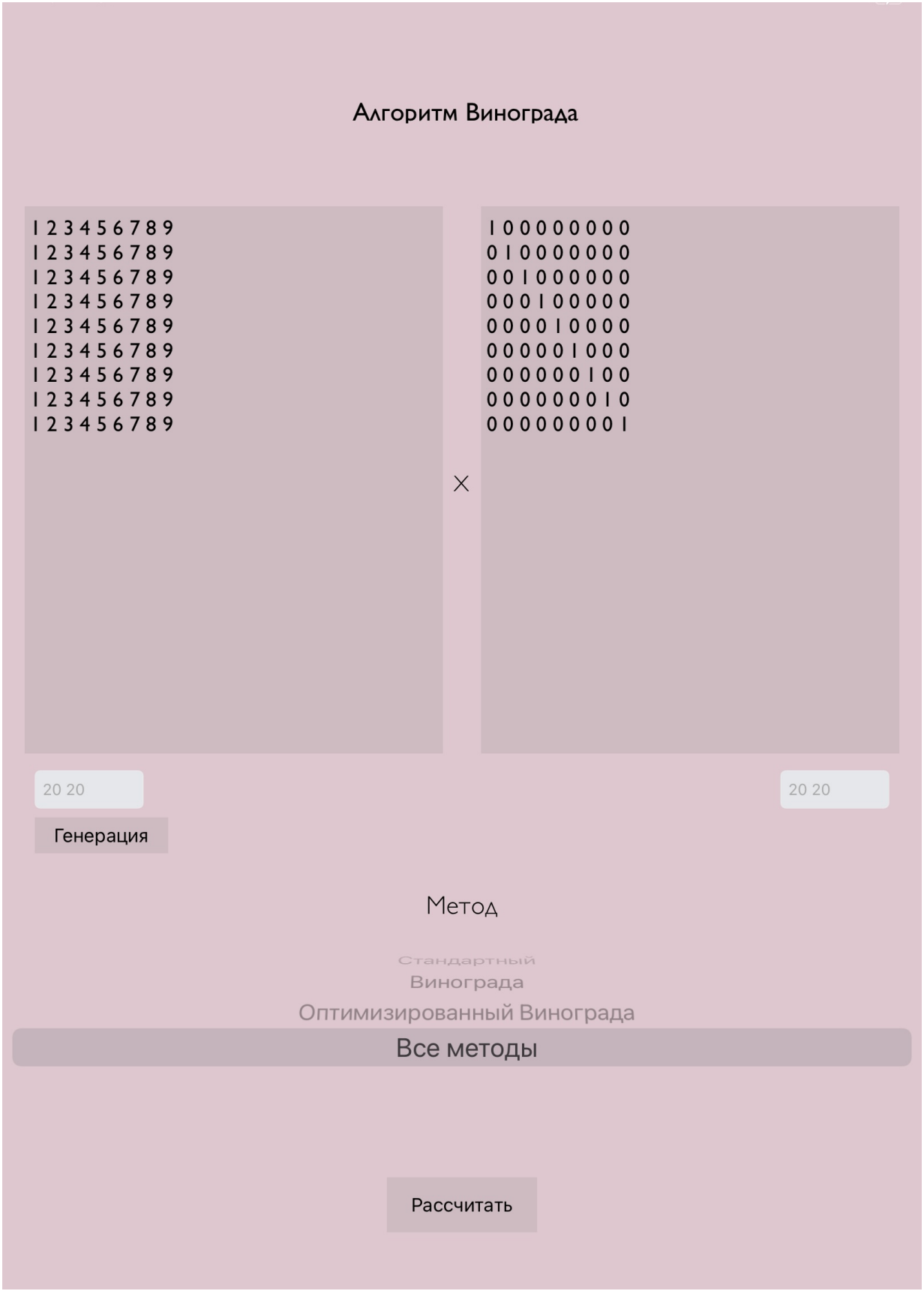


Рис. 4.1: Интерфейс

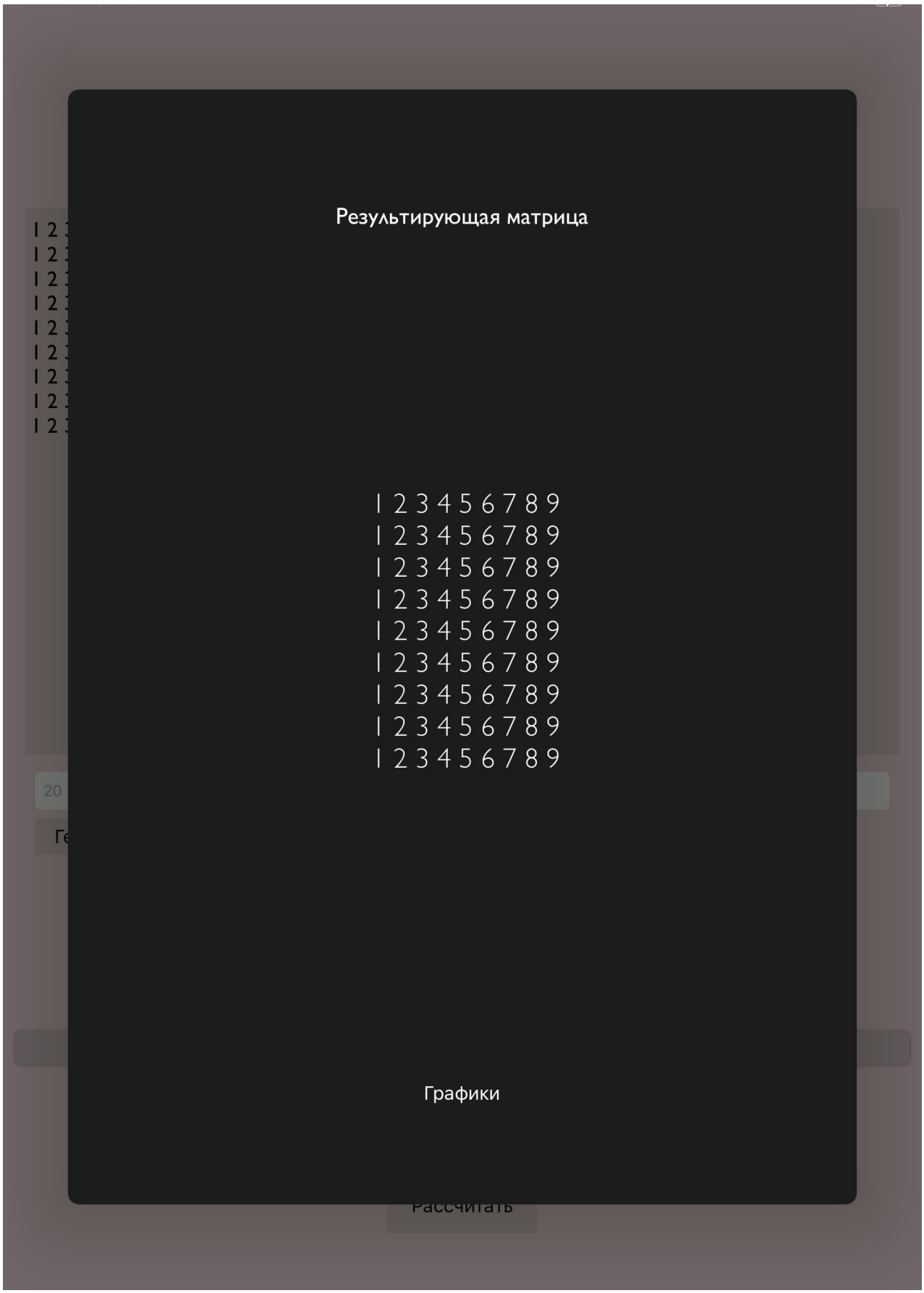


Рис. 4.2: Экран с результатом

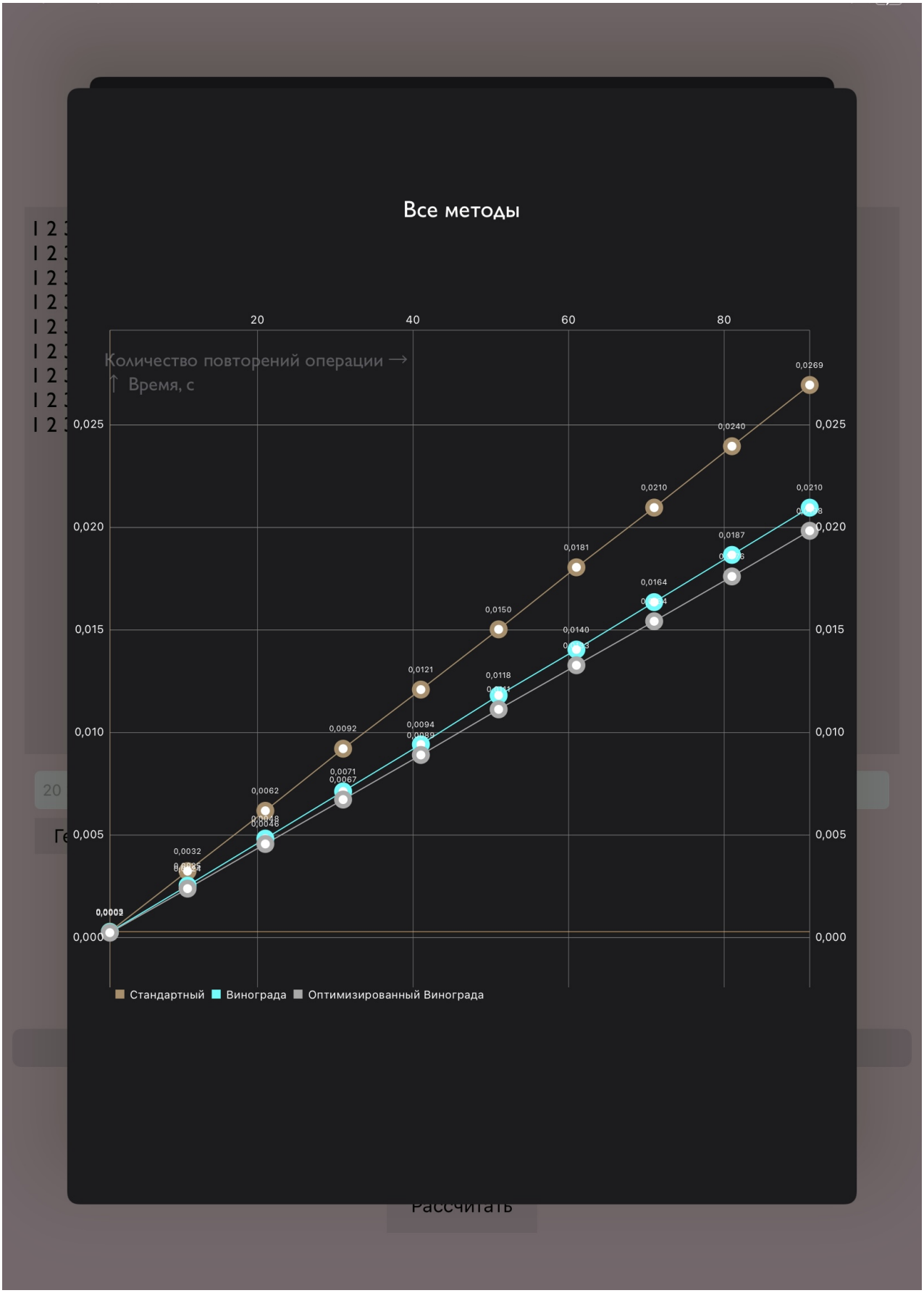


Рис. 4.3: Экран с графиками

## 4.3 Время выполнения реализаций алгоритмов

В таблице 4.1 представлены замеры времени работы для каждого из алгоритмов для квадратных матриц небольшого размера (1-9). Здесь и далее: СА — стандартный алгоритм, АВ — алгоритм Винограда, ОАВ – оптимизированный алгоритм Винограда. Время в микросекундах.

Таблица 4.1: Результаты замеров времени алгоритмов при малых размерах матриц (микросекунды)

| Размер | СА | АВ | ОАВ |
|---|---|---|---|
| 1 | 9.117 | 14.889 | 13.246 |
| 2 | 39.704 | 45.977 | 50.745 |
| 3 | 79.278 | 73.845 | 67.405 |
| 4 | 148.618 | 150.349 | 115.306 |
| 5 | 279.045 | 249.385 | 183.087 |
| 6 | 359.754 | 367.953 | 265.140 |
| 7 | 615.749 | 451.200 | 426.423 |
| 8 | 883.049 | 602.657 | 541.336 |
| 9 | 1194.958 | 825.197 | 799.224 |

Можно заметить, что при небольших размерах (1-9) матриц стандартный алгоритм эффективнее алгоритма Винограда по времени. Однако с увеличением размеров матриц растет и эффективность по времени работы алгоритма Винограда в сравнении со стандартным. Чтобы проследить разницу временных затрат стандартной реализации и реализации Винограда в двух вариациях, обратимся к матрицам большего размера.

В таблицах 4.2 и 4.3 представлены замеры времени работы для каждого из алгоритмов. Замеры проводились для четных и нечетных квадратных матриц размером от 10 до 201. Высчитывалось среднее время при количестве повторений равном 50. Время в секундах.

Таблица 4.2: Результаты замеров времени алгоритмов при четных размерах матриц (сек)

| Размер | CA | AB | OAB |
|---|---|---|---|
| 10 | 0.0017 | 0.0012 | 0.0010 |
| 20 | 0.0118 | 0.0076 | 0.0068 |
| 40 | 0.0953 | 0.0643 | 0.0616 |
| 80 | 0.7289 | 0.4340 | 0.4054 |
| 100 | 1.3709 | 0.9202 | 0.7693 |
| 150 | 4.7042 | 2.6902 | 2.5413 |
| 200 | 11.6528 | 6.3241 | 5.9507 |

Таблица 4.3: Результаты замеров времени алгоритмов при нечетных размерах матриц (сек)

| Размер | CA | AB | OAB |
|---|---|---|---|
| 11 | 0.0020 | 0.0013 | 0.0014 |
| 21 | 0.0133 | 0.0088 | 0.0079 |
| 41 | 0.0915 | 0.0540 | 0.0518 |
| 81 | 0.7135 | 0.4110 | 0.3831 |
| 101 | 1.6259 | 0.9673 | 1.1453 |
| 151 | 5.5601 | 2.9469 | 2.7519 |
| 201 | 12.1178 | 7.2646 | 6.7645 |

# Вывод

При увеличении размера матриц, увеличивается и эффективность работы алгоритма Винограда в сравнении со стандартным алгоритмом. В среднем реализация по Винограду работает быстрее в 1.7 раз. Оптимизированная реализация алгоритма Винограда также позволяет уменьшить время работы при больших размерностях: например, для размерности матрицы, равной 200, эксперимент показал, что алгоритм Винограда быстрее стандартного алгоритма чуть менее, чем в 2 раза, когда оптимизированная версия выигрывает у обычного умножения более, чем в 2 раза.

# Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- были реализованы 3 алгоритма умножения матриц: обычный, Винограда, оптимизированный Винограда;
- был произведен анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- алгоритмы проанализированы на основе экспериментальных данных: выявлено, что реализация по Винограду работает в среднем в 1.7 раз быстрее стандартной, этот показатель улучшается при увеличении размера матриц;
- подготовлен отчет о лабораторной работе.

# Литература


[1] Алгоритмы умножения матриц [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html

[2] Swift: Swift | Мультипарадигмальный компилируемый язык [Электронный ресурс]. Режим доступа: https://www.apple.com/swift/

[3] Core Foundation: Core Foundation | Time Utilities [Электронный ресурс]. Режим доступа: https://developer.apple.com/documentation/corefoundation/time_utilities

[4] Xcode: Xcode [Электронный ресурс]. Режим доступа: https://developer.apple.com/xcode/

[5] AppCode: AppCode | Smart Swift/Objective-C IDE for iOS [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/objc/

[6] iPad: iPad | Apple A14 Bionic [Электронный ресурс]. Режим доступа: https://developer.apple.com/xcode/