Отчет по лабораторной работе 1
По предмету “Анализ алгоритмов”
По теме “Умножение матриц”

Фирсова Дарья ИУ7-56

2018

Введение

В лабораторной работе изучаются алгоритмы умножения матриц. Рассмотрены алгоритмы: стандартный, алгоритм Винограда и улучшенный алгоритм Винограда.
Цель лабораторной работы: анализ, реализация и сравнительный анализ времени работы алгоритмов для различных размеров исходных матриц.
Задачи для лабораторной работы:

1. Ввести модель оценки трудоемкости
2. Реализовать стандартный алгоритм умножения матриц
3. Реализовать алгоритм Винограда
4. Релизовать улучшенный алгоритм Винограда, при этом произвести не менее 3 улучшений).
5. Провести временные замеры
6. Произвести расчет трудоемкости для реализованных алгоритмов.
7. Сравнительный анализ времени работы алгоритма для разных исходных матриц.

1 Аналитеская часть

В данном разделе приведены алгоритмы и составлена модель для вычисления трудоемкости.

1.1 Описание алгоритмов

1.1.1 Стандартный алгоритм умножения.

Имеем две матрицы A и B размерностями M x N и N x Q соответственно. Тогда результирующей матрицей будет матрица C размером M x Q, где $c_{ij} = \sum_{r=1}^{n} a_{ir} \cdot b_{rj}, (i = 0, 1, 2...m, j = 0, 1, 2...q)$.

1.1.2 Алгоритм Винограда.

Пусть i-я строка матрицы A - вектор $\vec{U}$, а j-й столбец матрицы B - вектор $\vec{V}$.

Тогда $C_{ij} = \boxed{u_1\ u_2\ u_3\ u_4} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 =$

$(u_1+v_2)(u_2v_1) + (u_3+v_4)(u_4+v_3)$ - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4.

"Хвост"для $\vec{U}$ вычисляется заранее и используется повтороно при умножении на каждый столбец матрицы B. Аналогично для вектора $\vec{V}$

Если вектора $\vec{U}$ и $\vec{V}$ нечетной длины, то к приведенным выше вычислениям, добавляем $C_{ij}\ += U_{N-1} \cdot V_{N-1}, \forall i, j$

1.2 Модель вычислений

Введем следующую модель вычислений: операции $+ - * / < > == != += == []$ имеют стоимость 1.

1.2.1 Оценка трудоемкости цикла for

Инициализация до цикла стоит 2, после выполнения тела цикла, инкрементируется итератор цикла, проверяется условие.
$F = 2 + N * (F_{body} + 2)$

1.2.2 Оценка трудоемкости оператора if

Переход по условию имеет стоимость 0, проверка условия зависит от выражения самой провеки согласно модели выше.
Для оператора без проверки условия: $F = 0$
Для оператора с проверкой условия: $F = 0 + body$

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов

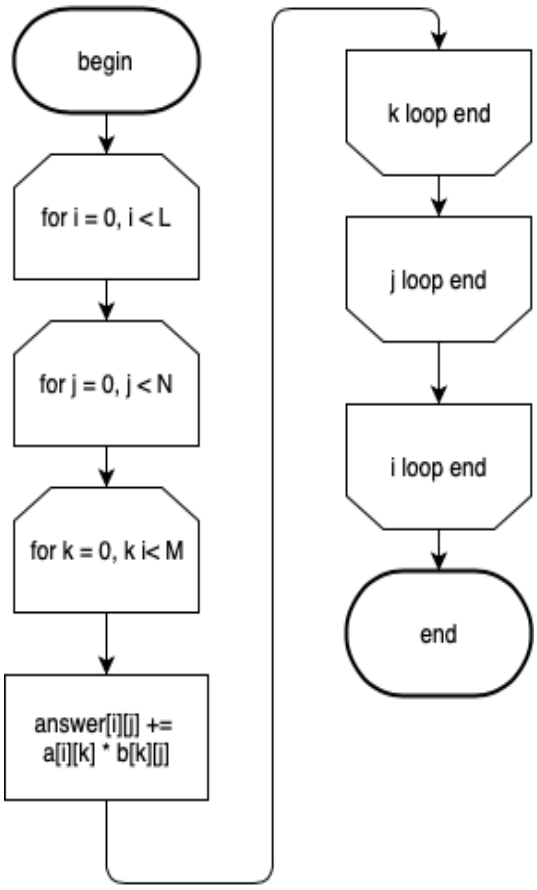


Рис. 1: Схема стандартного алгоритма

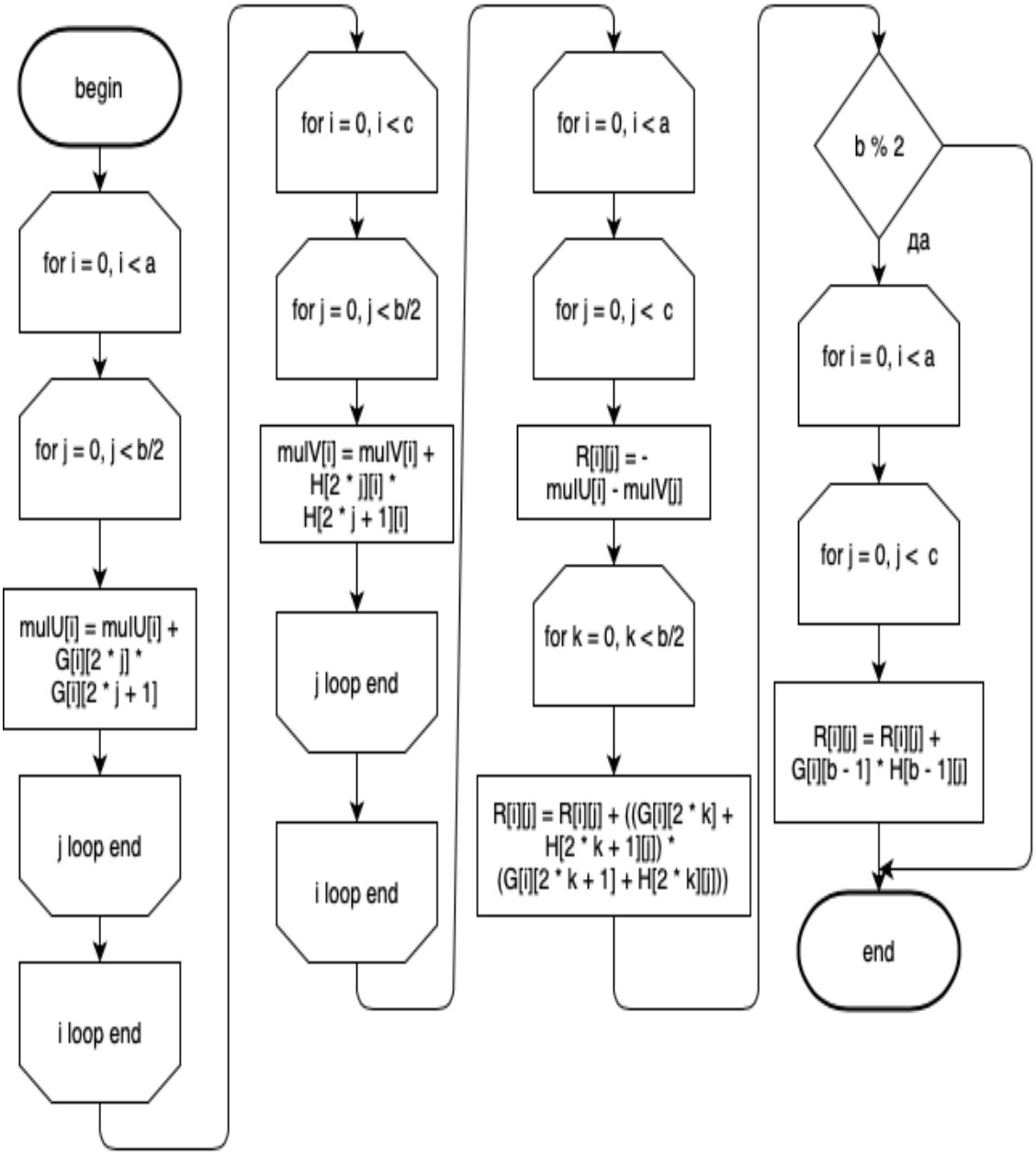


Рис. 2: Схема алгоритма Винограда

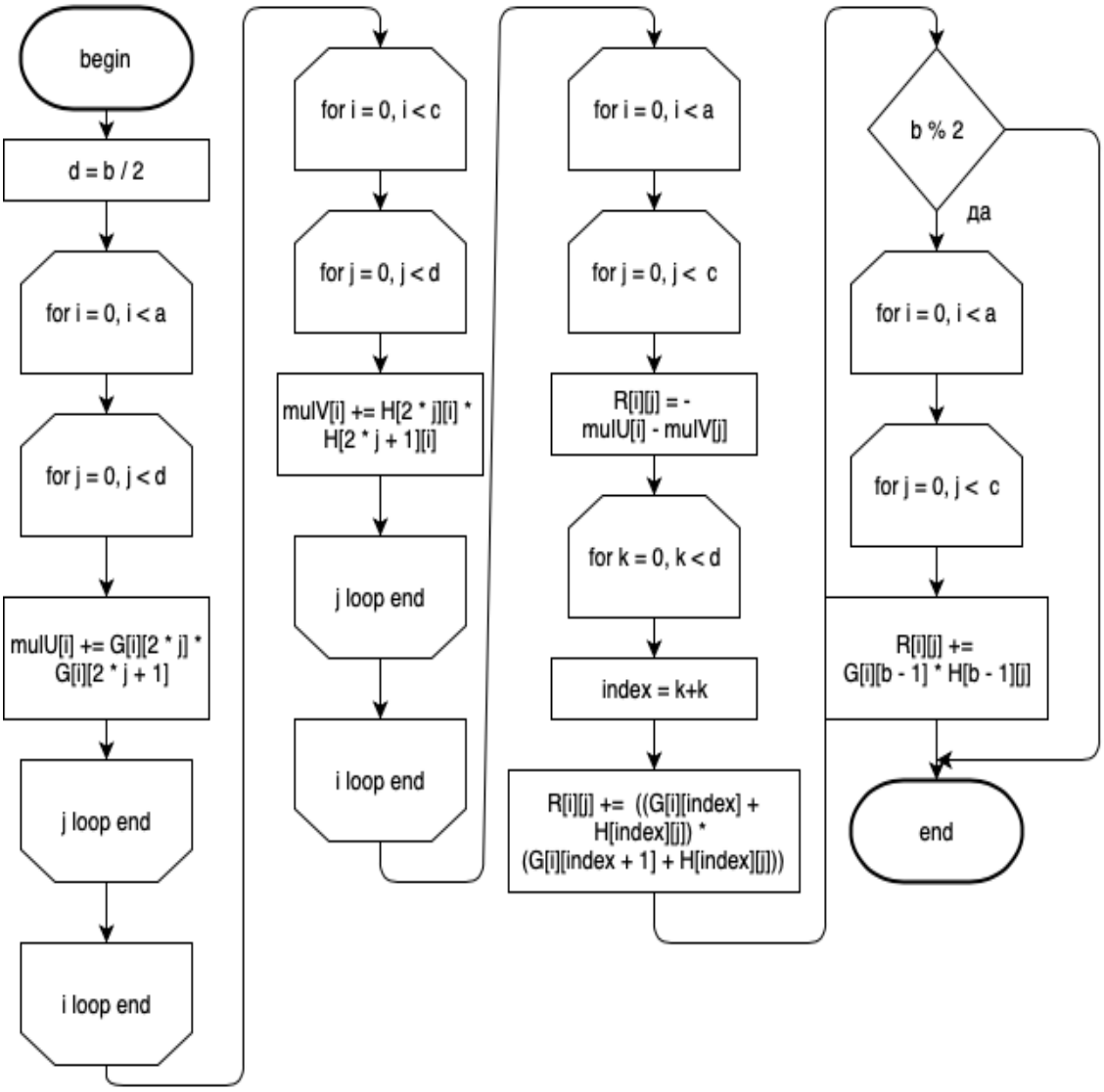


Рис. 3: Схема алгоритма улучшенного Винограда

3 Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

3.1 Средства реализации

В данной работе использовался язык Python 3.6, в среде Pycharm. Для измерения времени использовался модуль time, измерения проищводились в секундах.

3.2 Листинг кода

```
def multiply(a, b, l, m, n):
    answer = [[0 for i in range(n)] for j in range(l)]
    for i in range(l):
        for j in range(n):
            for k in range(m):
                answer[i][j] += a[i][k] * b[k][j]
    return answer

def winograd(G, H):
    a = len(G)
    b = len(H)
    c = len(H[0])
    mulU = [0 for i in range(a)]
    mulV = [0 for i in range(c)]
    R = [[0 for i in range(c)] for j in range(a)]

    for i in range(a):
        for j in range(0, b // 2, 1):
            mulU[i] = mulU[i] + G[i][2 * j] * G[i][2 * j + 1]

    for i in range(c):
        for j in range(0, b // 2, 1):
            mulV[i] = mulV[i] + H[2 * j][i] * H[2 * j + 1][i]

    for i in range(a):
        for j in range(c):
            R[i][j] = - mulU[i] - mulV[j]
            for k in range(0, b // 2, 1):
                R[i][j] = R[i][j] + ((G[i][2 * k] + H[2 * k + 1][
    j]) * (G[i][2 * k + 1] + H[2 * k][j]))
```

```
    if b % 2:
        for i in range(a):
            for j in range(c):
                R[i][j] = R[i][j] + G[i][b - 1] * H[b - 1][j]

    return R

def opt_winograd(G, H):
    a = len(G)
    b = len(H)
    c = len(H[0])
    d = b // 2
    mulU = [0 for i in range(a)]
    mulV = [0 for i in range(c)]
    R = [[0 for i in range(c)] for j in range(a)]

    for i in range(a):
        for j in range(d):
            mulU[i] += G[i][2 * j] * G[i][2 * j + 1]

    for i in range(c):
        for j in range(d):
            mulV[i] += H[2 * j][i] * H[2 * j + 1][i]

    for i in range(a):
        for j in range(c):
            R[i][j] = - mulU[i] - mulV[j]
            for k in range(d):
                index = 2 * k
                R[i][j] += ((G[i][index] + H[index + 1][j]) * (G[
    i][index + 1] + H[index][j]))
    if b % 2:
        for i in range(a):
            for j in range(c):
                R[i][j] += G[i][b - 1] * H[b - 1][j]

    return R
```

Листинг 1. Реализация алгоритмов.

3.3 Вычисление трудоемкости алгоритмов

Расчет производился по исходному коду, указанному на листинге 1. Разделение на части проводилось согласно логическим сегментам программы. Для сокращения времени работы алгоритма были сделаны следующие улучшения:

1. В цикле не вычисляется значение границы цикла. До начала работы алгоритма введена новая переменная.
2. Введен оператор $+=$ для сокращения количества операций.
3. Изменены общие индексы для взятия адреса.

Оценка трудоемкости стандартного алгоритма:
$10MNQ + 4MQ + 4M + 2$

Оценка трудоемкости алгоритма Винограда:
Первая часть: $\frac{13MN}{2} + 5M + 2$
Вторая часть: $\frac{13QN}{2} + 5M + 2$
Третья часть: $13MNQ + 9MQ + 2M + 2$
Четвертая часть: $15QM + 2M + 1$

Оценка трудоемкости улучшенного алгоритма Винограда:
Первая часть: $6MN + 2M + 2$
Вторая часть: $6QN + 2M + 2$
Третья часть: $10MNQ + 9QM + 2M + 2$
Четвертая часть: $12MQ + 2M + 1$

4 Экспериментальная часть

В данном разделе будут приведены примеры работы алгоритмов и произведены замеры времени. Тестирование производилось на компьютере с процессором Intel Core i5 (I5-6267U) и оперативной памятью 8 Гб.

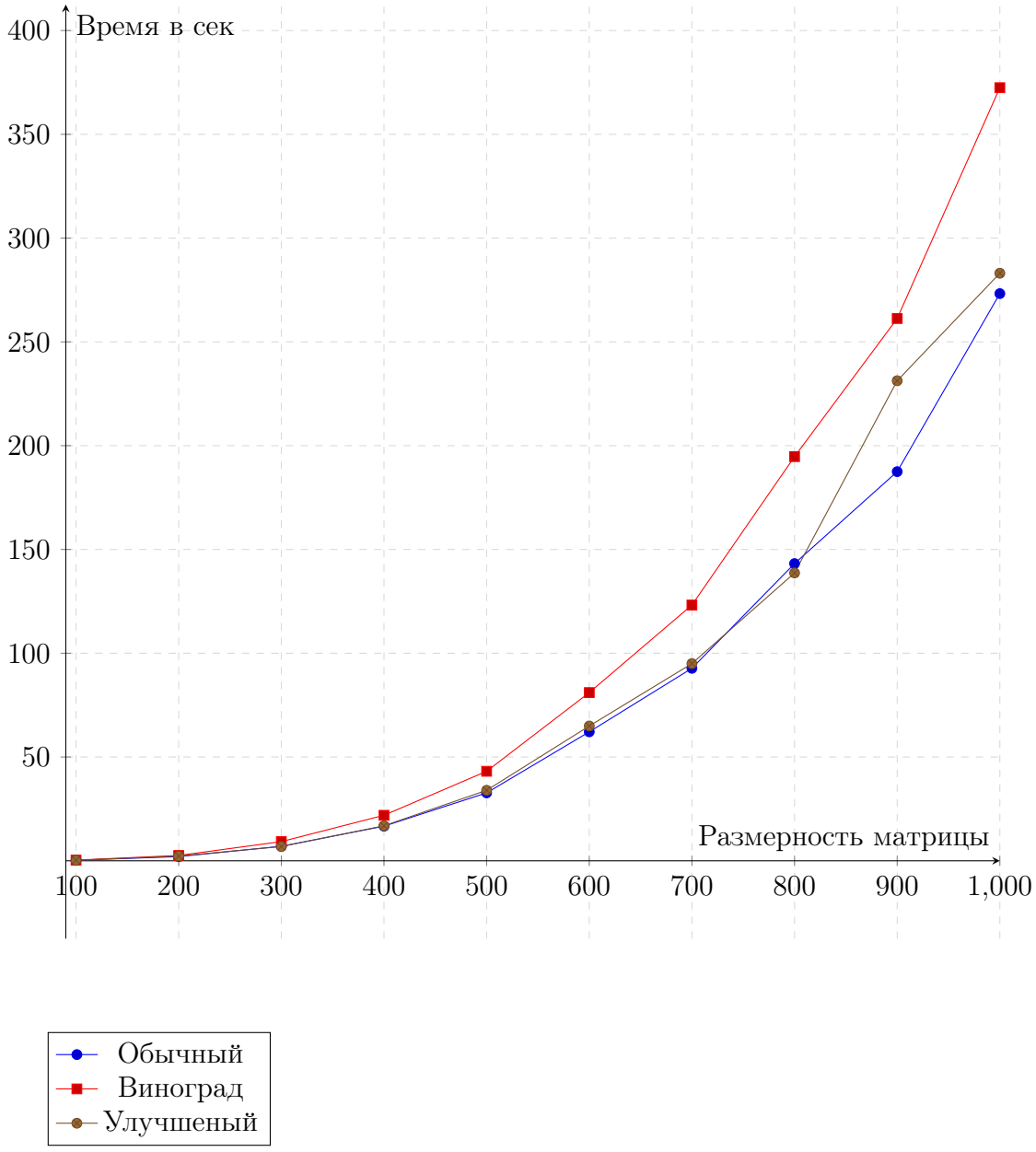
4.1 Примеры работы

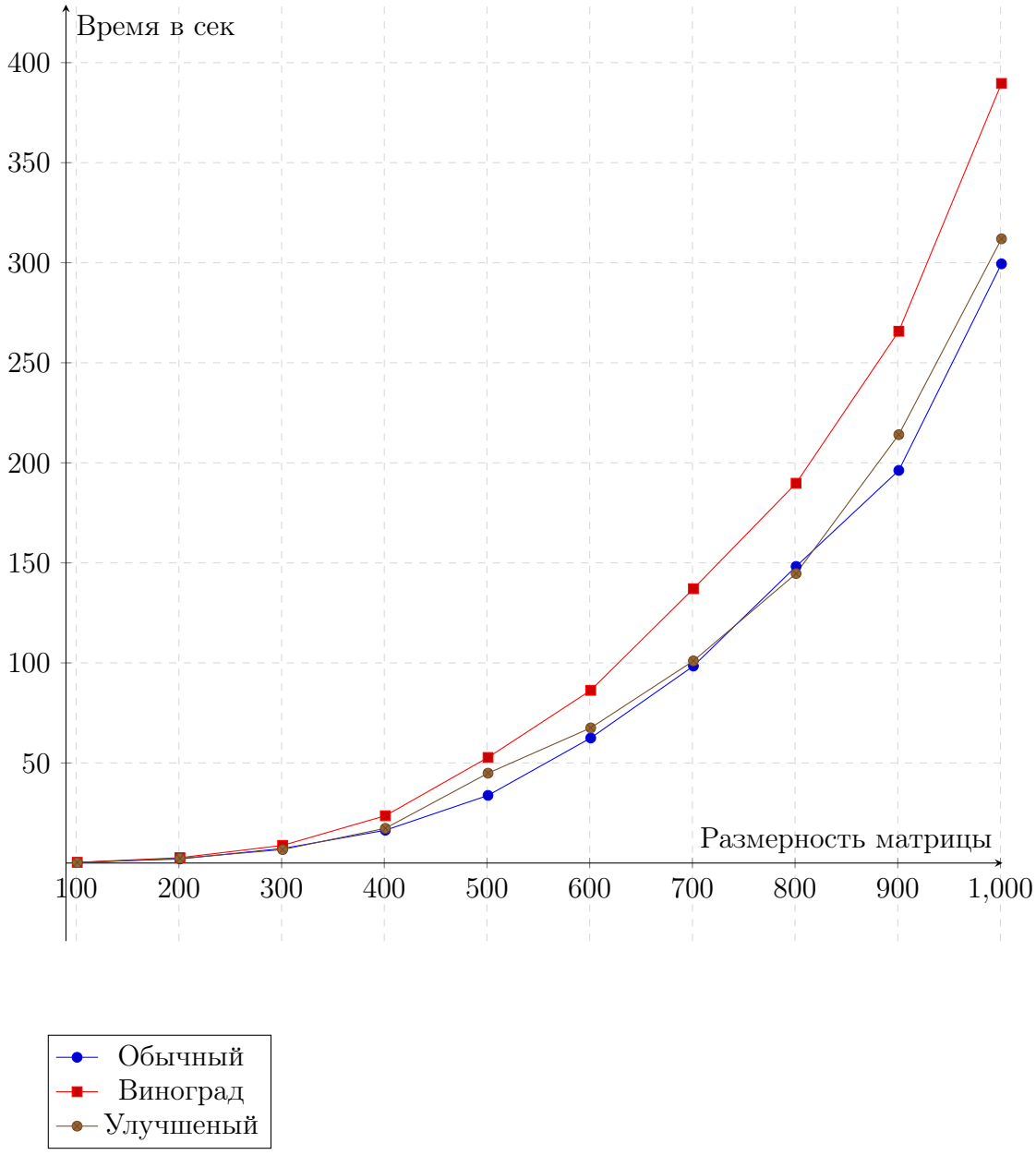
Пример результата работы умножения матриц. Так как в данной реализации генерируются случайные значения, то для проверки результата использовалась библиотека Numpy. При одинаковых входных данных алгоритмы выдают одинаковый результат, который сравнивается с результатом умножния с помощью функции numpy.matmul(). Для вычисления используются матрицы с размерностью $A = N * N + 1$ и $B = N + 1 * N$

$$\begin{bmatrix} 12 & 14 & 20 \\ 24 & 14 & 16 \end{bmatrix} + \begin{bmatrix} 11 & 15 \\ 8 & 15 \\ 14 & 15 \end{bmatrix} = \begin{bmatrix} 524 & 690 \\ 600 & 810 \end{bmatrix}$$

4.2 Сравнительный анализ

Сравнение алгоритмов стандартного умножения, алгоритма Винограда и улучшенного алгоритма Винограда. На графиках приведены замеры времени работы для матрицы четной и нечетной размерности. Первый график для лучшго случая - нечетной размерности, второй график для четной размерности. Каждый экперимент проводился два раза из-за большого времени работы алгоритма, результат - среднее арифметическое двух замеров времени.


Время в сек
400
350
300
250
200
150
100
50
Размерность матрицы
100
200
300
400
500
600
700
800
900
1,000
Обычный
Виноград
Улучшеный


Время в сек
400
350
300
250
200
150
100
50
Размерность матрицы
100
200
300
400
500
600
700
800
900
1,000
Обычный
Виноград
Улучшеный

4.3 Вывод

На малых размерностях исходной матрицы время работы стандартного и улучшенного алгоритма Винограда различаюся незначительно. Тогда как алгоритм Винограда всегда работает медленнее. На больших размерностях стандартный алгоритм умножения матриц работает быстрее. При размерности в 1000, разница между стандартным алгоритмом и улучшенным Винограда составляет 10 секунд, а обычный алгоритм Винограда работает на 100 секунд дольше.

5 Заключение

В данной лабораторной работе вычислены сложности алгоритмов для умножения матриц. Разработаны программы по этим алгоритмам, проведены тесты по времени, произведен сравнительный анализ алгоритмов. Для составления отчета использован Latex.