Анализ Алгоритмов

Лабораторная работа №2

«Алгоритмы умножения матриц»

Юмаев Артур Русланович

ИУ7-55

Преподаватель: Волкова Л.Л.

Оглавление

[Введение 3](#_Toc22826197)

[Постановка задачи 4](#_Toc22826198)

[1. Аналитическая часть 5](#_Toc22826199)

[Классический подход 5](#_Toc22826200)

[Алгоритм Винограда 5](#_Toc22826201)

[Оптимизированный алгоритм Винограда 6](#_Toc22826202)

[2. Конструкторская часть 7](#_Toc22826203)

[3. Технологическая часть 8](#_Toc22826204)

[4. Исследовательская часть 11](#_Toc22826205)

# Введение

Алгоритмы перемножение матриц имеют довольно обширное применение. В данное время они используются в области искусственного интеллекта. Перемножение матриц – одна из основных операций, на которых строятся алгоритмы искусственных нейронных сетей, которые могут анализировать изображения, находить скрытые взаимосвязи и классифицировать их, заранее обучившись на некой тренировочной выборки. Перемножения матриц также используются в прикладной физике, математике, математической статистике и многих других прикладных науках.

В данный момент существуют несколько алгоритмов перемножения матриц. Ниже в таблице 1 приведен их список с коэффициентом ω, который показывает сложность алгоритмов ).

Таблица 1

Сравнение эффективности по времени алгоритмов умножения матриц

|  |  |
| --- | --- |
| Алгоритм | ω |
| Классический (1950) | 3.0 |
| Алгоритм Пана (1978) | 2.78041 |
| Алгоритм Бини (1979) | 2.7799 |
| Алгоритм Шёнхаге (1981) | 2.695 |
| Алгоритм Копперсмита — Винограда (1990) | 2.3727 |

Алгоритм, который будет реализован в данной работе является одним из самых эффективных на данный момент.

# Постановка задачи

Реализовать классический алгоритм умножения матриц. Изучить и реализовать алгоритм Винограда. Разработать и реализовать оптимизированный вариант алгоритма Винограда. Выбрать модель оценки трудоёмкости и по ней дать оценку трудоёмкости классическому алгоритму умножения матриц, алгоритму Винограда (для лучшего и худшего случая) и оптимизированному алгоритму Винограда (для лучшего и худшего случая). Сделать замеры времени для алгоритмов. Результаты экспериментов сравнить с теоретическими оценками трудоёмкости. Сделать выводы.

# Аналитическая часть

В данном разделе будет приведено математическое описание алгоритмов перемножения матриц. Буду рассмотрены 3 подхода: классический, алгоритм Винограда и оптимизированный алгоритм Винограда.

## Классический подход

Предположим, что необходимо получить матрицу Для нахождения значений элементов матрицы используют следующее выражение:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Классический алгоритм напрямую реализует эту формулу.

## Алгоритм Винограда

Можно заметить, что элементы из суммы выражения 1 можно переписать как:

т. е. как сумму произведения сумм и двух произведений. Учитывая, что упомянутые два произведения можно рассчитать заранее для обработки двух элементов матрицы теперь нужно не сложение и два умножения, а умножение и два сложения, что проще с точки зрения вычислений. Таким образом, алгоритм Винограда состоит в том, что:

1. Совершить расчет заранее двух произведений для каждого ряда и столбца матрицы-результата (одно произведение считается для ряда, другое для столбца). Для хранения результатов используется промежуточный буфер;
2. По вышеприведённой формуле осуществить расчёт каждого элемента матрицы;
3. В случае, если «совпадающий край» исходных матриц имеет размерность, характеризующуюся нечётным числом, пройтись во второй раз по матрице, дополняя элементы недостающим элементом (который не был описан вышеописанной суммой).

Можно заметить, что пункт 3 необходимо выполнять только в некоторых случаях, но если это происходит, то получается существенное увеличение времени работы алгоритма.

## Оптимизированный алгоритм Винограда

Пункт 3 можно совместить с пунктом 2, уменьшив число обращений к памяти. Это позволяет уменьшить время работы алгоритма в случаях с нечётной общей стороной исходных матриц. Кроме того, для осуществления алгоритма Винограда используется два дополнительных буфера. От них можно избавиться за счёт небольших потерь производительности, поместив вычисление препроцессированных значений в основной этап (пункт 2). Доступ к данным, расположенным последовательно и близко в памяти, в среднем осуществляется быстрее, чем к расположенным далеко (из-за кэша процессора и других элементов процесса доступа к памяти). Поэтому вторая матрица предварительно транспонируется.

# 2. Конструкторская часть

# 3. Технологическая часть

В данном разделе будут приведены листинги для каждого из алгоритмов на языке Python.

Листинг 1 Классический алгоритм умножения матриц

|  |
| --- |
| def classic\_matrix\_mult(A, B):  A, B = np.array(A), np.array(B)    if len(B) != len(A[0]):  print("Error! Different dimension!")  return None  n = len(A)  m = len(A[0])  t = len(B[0])  C = np.zeros((A.shape[0], B.shape[1]))  for i in range(n):  for j in range(m):  for k in range(t):  C[i][k] += A[i][j] \* B[j][k]  return C |

Листинг 2 Алгоритм Винограда

|  |
| --- |
| def classic\_winograd(A, B):  M = len(A)  N = len(B)  Q = len(B[0])    C = np.zeros((M, Q)) # Will be result  if N != len(A[0]):  print("Different dimension of the matrics")  return None  MulB = np.zeros((M))  MulV = np.zeros((Q))  for i in range(M):  for j in range(N // 2):  MulB[i] += A[i][2 \* j] \* A[i][2 \* j + 1]  for i in range(Q):  for j in range(N // 2):  MulV[i] += B[2 \* j][i] \* B[2 \* j + 1][i]  for i in range(M):  for j in range(Q):  C[i][j] = - MulB[i] - MulV[j]  for k in range(N // 2):  C[i][j] += ((A[i][2 \* k] + B[2 \* k + 1][j]) \* (A[i][2 \* k + 1] + B[2 \* k][j]))  if N % 2:  for i in range(M):  for j in range(Q):  C[i][j] += A[i][N - 1] \* B[N - 1][j]  return np.array(C) |

Листинг 3 Оптимизированный алгоритм Винограда

|  |
| --- |
| def optimized\_winograd(A, B):  M = len(A)  N = len(B)  Q = len(B[0])    C = np.zeros((M, Q)) # Will be result  if N != len(A[0]):  print("Different dimension of the matrics")  return None  d = N // 2  MulB = np.zeros((M))  MulV = np.zeros((Q))  for i in range(M):  MulB[i] = sum(A[i][2 \* j] \* A[i][2 \* j + 1]  for j in range(d))  for i in range(Q):  MulV[i] = sum(B[2 \* j][i] \* B[2 \* j + 1][i]  for j in range(d))  for i in range(M):  for j in range(Q):  C[i][j] = sum((A[i][2 \* k] + B[2 \* k + 1][j]) \*  (A[i][2 \* k + 1] + B[2 \* k][j])  for k in range(d))  - MulB[i] - MulV[j]  if N % 2:  for i in range(M):  C[i][j] = sum(A[i][N - 1] \* B[N - 1][j]  for j in range(Q))  return C |

# 4. Исследовательская часть

В данном разделе будет приведено сравнение классического алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Тестирование проводилась на сначала квадратных матрицах четного размера от *50* до *400* в шагом *50*, далее также проводились измерения на матрицах нечетного размера:

Замер времени проводился с помощью библиотеки time в Python 3.7 и метода process\_time().

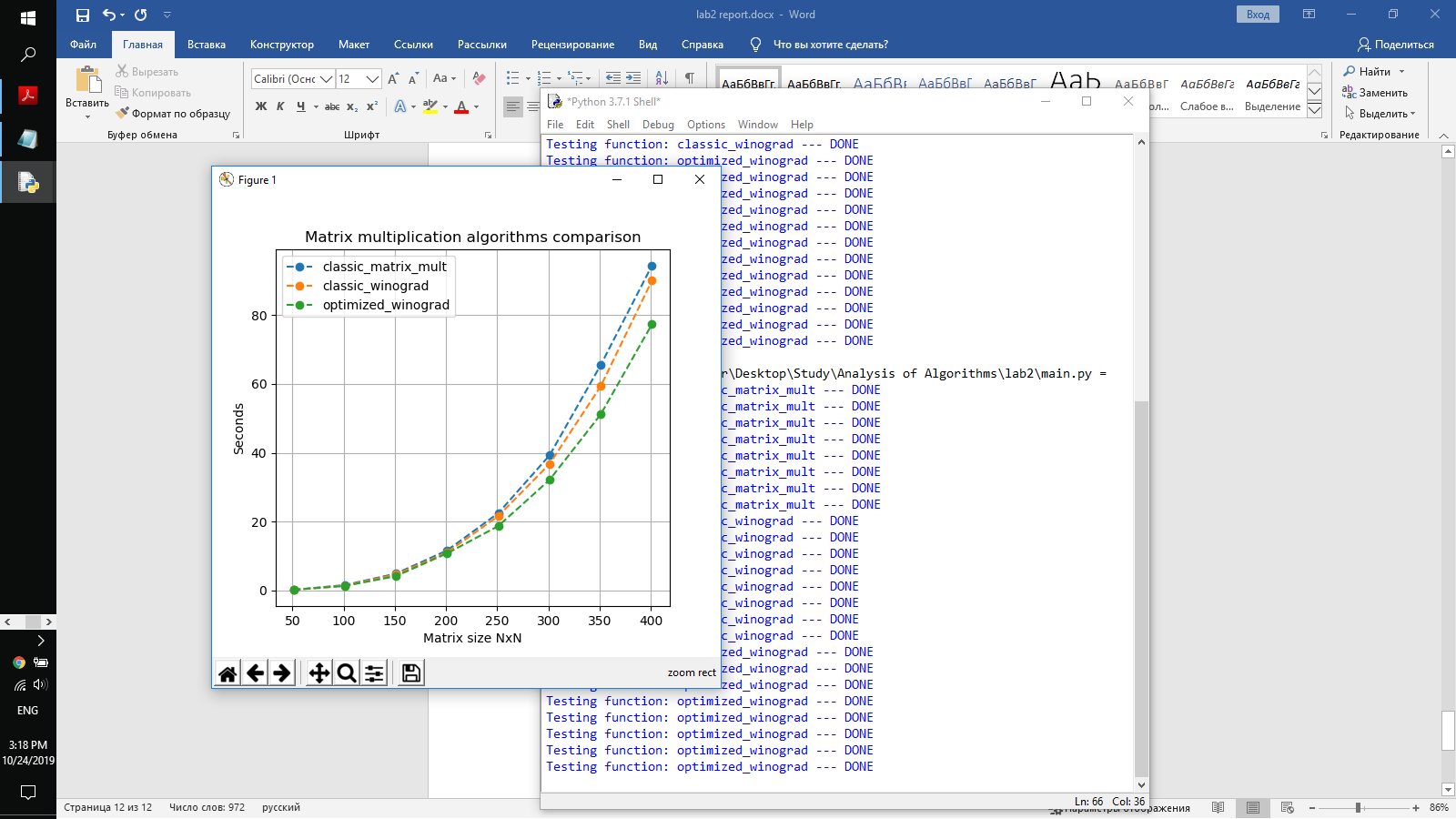


Рисунок 1 - Сравнение алгоритмов по скорости работы

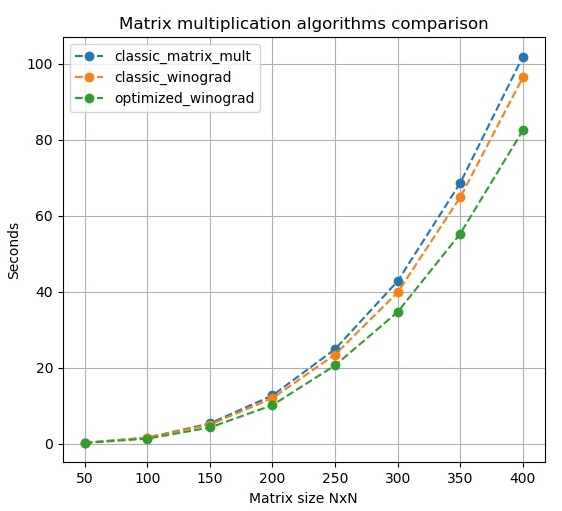


Рисунок 2 - Сравнение алгоритмов по скорости работы на матрицах нечетных размеров

По опыту видно, что матрицы с четным размером в среднем считаются быстрее, чем матрицы с нечетным размером.

# Заключение

Необходимо учитывать влияние размера входных данных на скорость выполнения алгоритмов. Трудоёмкость позволяет примерно оценить изменение времени, необходимого на выполнение алгоритма, в зависимости от входных параметров (размеры входных данных), но не позволяет оценить относительную скорость выполнения алгоритмов относительно друг друга.

В результате выполнения данной работы был реализован классический алгоритм умножения матриц. Был изучен и реализован алгоритм Винограда. Был разработан и реализован оптимизированный вариант алгоритма Винограда. Была выбрана модель оценки трудоёмкости и по ней были даны оценки трудоёмкости классическому алгоритму умножения матриц, алгоритму Винограда и оптимизированному алгоритму Винограда. Проведены замеры времени для алгоритмов. Было проведено сравнение результатов экспериментов с теоретическими оценками трудоёмкости. По итогам проведения работы были сделаны выводы.