Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Дисциплина: анализ алгоритмов

Лабораторная работа №2

Студент: Власова Екатерина, гр. ИУ7-54

2019 г.

Содержание

Введение

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае, количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами. В физике и других прикладных науках матрицы – являются средством записи данных и их преобразования. В программировании – в написании программ. И если сложение и вычитание матриц не представляет проблем, то умножение матриц представляет собой достаточно занимательную задачу. В данной лабораторной будут рассмотрены алгоритмы умножения матриц и их оптимизации.

1 Аналитическая часть

Ссылаясь на [1], дадим определение матрицы:

Матрицей A размера m×n называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. Условимся обозначать матрицы прописными буквами латинского алфавита: A, B, C, D, … Числа, функции или алгебраические выражения, образующие матрицу, называются матричными элементами. Будем обозначать их строчными буквами с двумя индексами. Первый индекс i=1,2, …, m указывает номер строки, а второй индекс j=1,2, …, n — номер столбца, в которых располагается соответствующий элемент. Таким образом,

$$A_{m\times n}\begin{pmatrix} a_{11} & a_{21} & .. & a_{1n} \\ a_{21} & a_{22} & .. & a_{2n} \\ .. & .. & .. & .. \\ a_{m1} & a_{m2} & .. & a_{mn} \end{pmatrix} \quad (1)$$

Здесь и в последующих формулах рядом с символом матрицы указан её размер.

Умножение матрицы A на матрицу B определено, лишь когда число столбцов первой матрицы в произведении равно числу строк второй.

1.1 Описание алгоритмов

Описание рассматриваемых алгоритмов умножения матриц.

Базовое умножение матриц

Пусть даны две матрицы A и B. Согласно [1], умножение матрицы A на матрицу B определено, лишь когда число столбцов первой матрицы в произведении равно числу строк второй. Тогда произведением матриц A m×k B k×n называется матрица C m×n, каждый элемент которой cij равен сумме попарных произведений элементов i–й строки матрицы A на соответствующие элементы j–го столбца матрицы B, т. е.

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{jk} = \sum_{s=1}^{k} a_{is}b_{sj} \quad (1.1)$$

для всех i = 1, 2, …, m и j = 1, 2, …, n.

Алгоритм Винограда

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нём представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое

умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Используя [4], рассмотрим два вектора V = (v1, v2, v3, v4) и W = (w1, w2, w3, w4). Их скалярное произведение равно:

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4. \quad (1.2)$$

Это равенство можно переписать в виде:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \quad (1.3)$$

Выражение в правой части формулы 1.3 допускает предварительную обработку: его части 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.

1.2 Задание на выполнение лабораторной работы

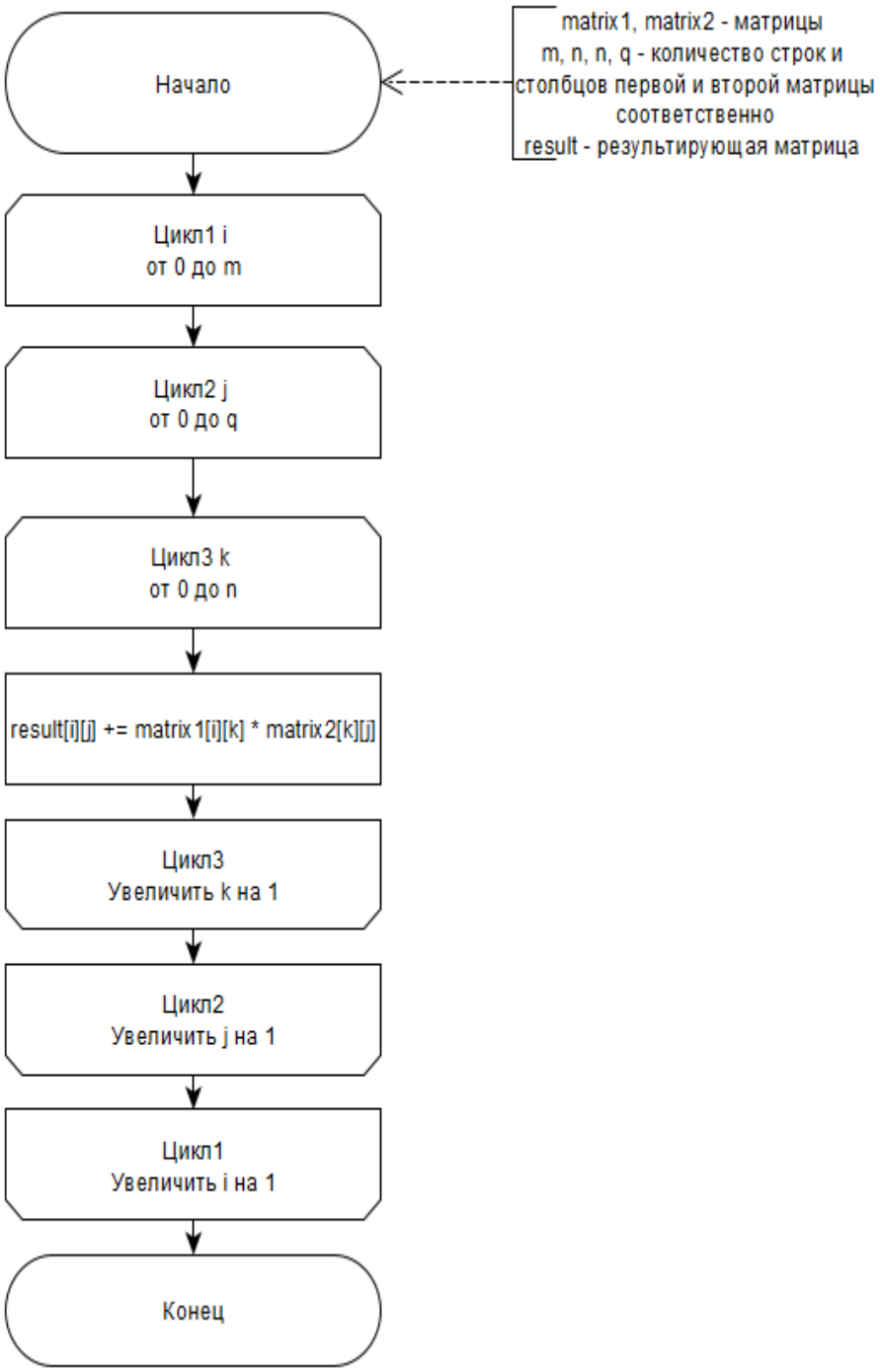
- Реализовать классический алгоритм умножения матриц и алгоритм Винограда;
- Провести оптимизацию алгоритма Винограда;
- Оценить трудоёмкость рассматриваемых алгоритмов;
- Сравнить эффективность алгоритмов по времени.

2 Конструкторская часть

В данном разделе представлены блок-схемы рассматриваемых алгоритмов и вычислена их трудоёмкость.

2.1 Разработка алгоритмов

2.1.1 Классическое умножение матриц



F body = 1 + F outer

F outer = 2 + M * (1 + F middle)

F middle = 2 + R * (1 + F inner)

F inner = 2 + N *

*(6 + 1 +1 +

+ 1 +

+ 1 +

+ 1)

Рисунок 1. Классическое умножение матриц

Трудоёмкость

Операция	**Стоимость**
Арифметические операторы	1
Операторы присваивания	1
[]	1
Другие операторы	1

Оператор	**Количество**
F body	
<	1
F outer	
+	M
<	M
F middle	
+	M * Q
<	M *Q
F inner	
[]	6 * M * Q * N
*	M * Q * N
+=	M *Q * N
<	M * Q * N

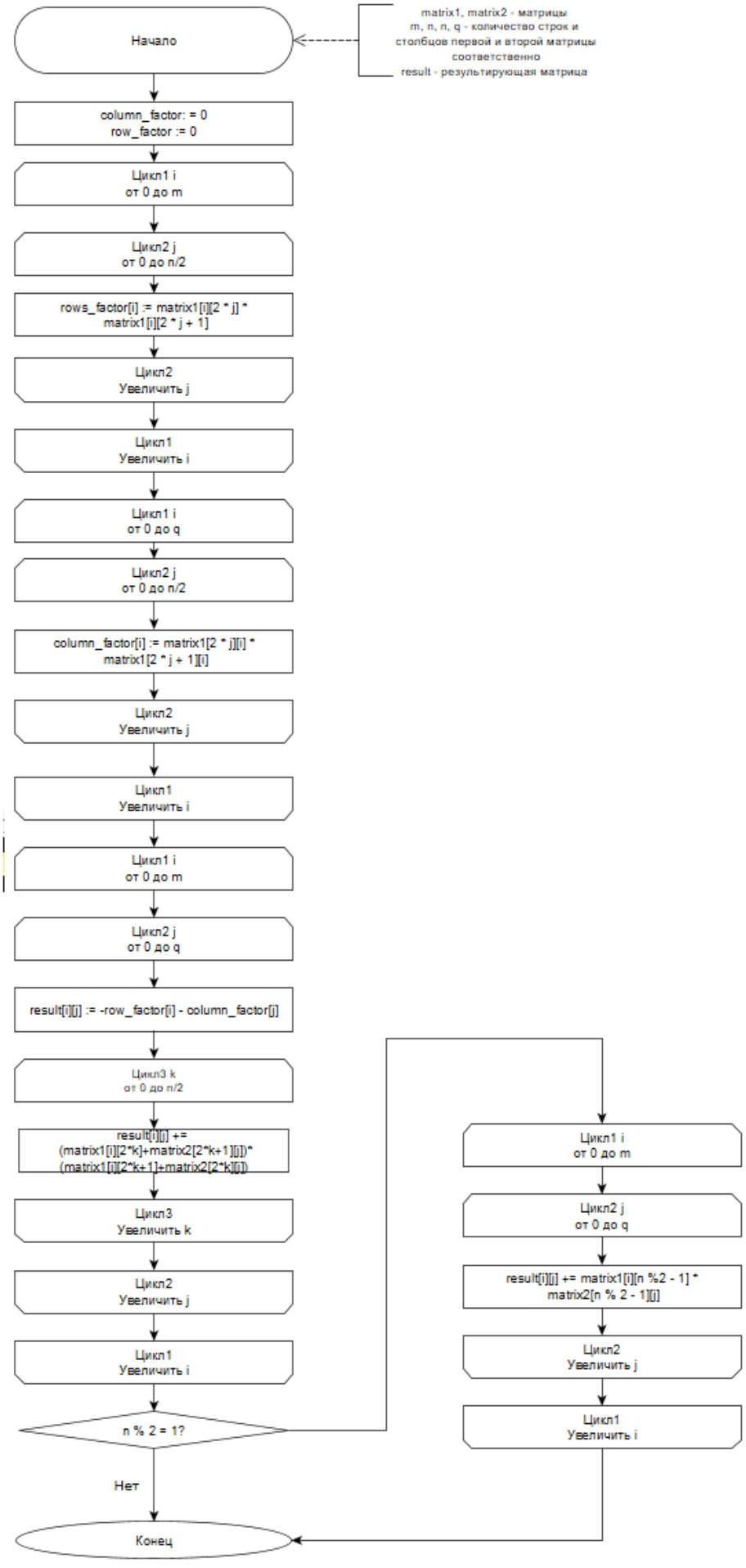
F inner = 2 + 11N

F middle = 2 + R * (1 + F inner) = 2 + 3 * R + 11 * R * N

F outer = 2 + M * (1 + F middle) = 2 + 3 * M + 3 * R * M + 11 * R * N * M

F body = 3 + 3 * M + 3 * R * M + 11 * R * N * M

2.1.2 Алгоритм Винограда



Fall=2+Frf+ Fcf+Fmult

Frf=2+M(1+ Frf_inner)

Frt_inner = 2 + D(5+ 3 +1 +1

+1 +

+ 1)

Fcf=2+Q(1+ Fcf_inner)

Fcf_inner=2+ D*(5 + 3 +1 +1

+1+

+1)

Fmult=2+M(1 +Fmouter)+ Fodd

Fmouter=2+Q (1+4+1 +1 + Fminner)

Fminner=2 +D(10+5+1+2

+1 +1 +1)

Fodd=4 +(Foddout)

Foddout=2+M (1+Foddiner)

Foddinner=1+ Q(6+1+2+1+2

+1 +1)

Оператор	Количество
Fall	
=	2
/	2
Frf	
<	1
=	1
Frf_inner	
[]	5
+	1
*	3
<	1
=	1
Fcf	
<	1
=	1
Fcf_inner	
*	3
[]	5
=	1
+	1
<	1
Fmult	
<	1
+=	1
Fmouter	
<	1
+=	2
[]	4
*	1
Fminner	
<	1
+=	1
[]	10
*	5
+	3
Fodd	
=	11
%	1
=	1
<	1

Foddout	
=	1
[]	6
%	2
-	1

Рассчитаем трудоёмкость алгоритма Винограда:

$$Foddinner = 1+14Q$$

$$Foddout = 1 + 2M + 14QM$$

$$Fodd = 5 + 2M + 14QM$$

$$Fminner = 2 + 21D$$

$$Fmouter = 2+9Q+21DQ$$

$$Fmult = 2+ 3\ M+9QM+21DQM+5 + 2M + 14QM=7+5M+23QM+21DQM$$

$$Fcfinner = 2 + 12D$$

$$Fcf = 2+3Q+12DQ$$

$$Frfinner = 2+12D$$

$$Frf = 2+3m+12DM$$

$$Fall = 2 +7+5M+23QM+21DQM+2+3Q+12DQ+2+3M+12DM=$$

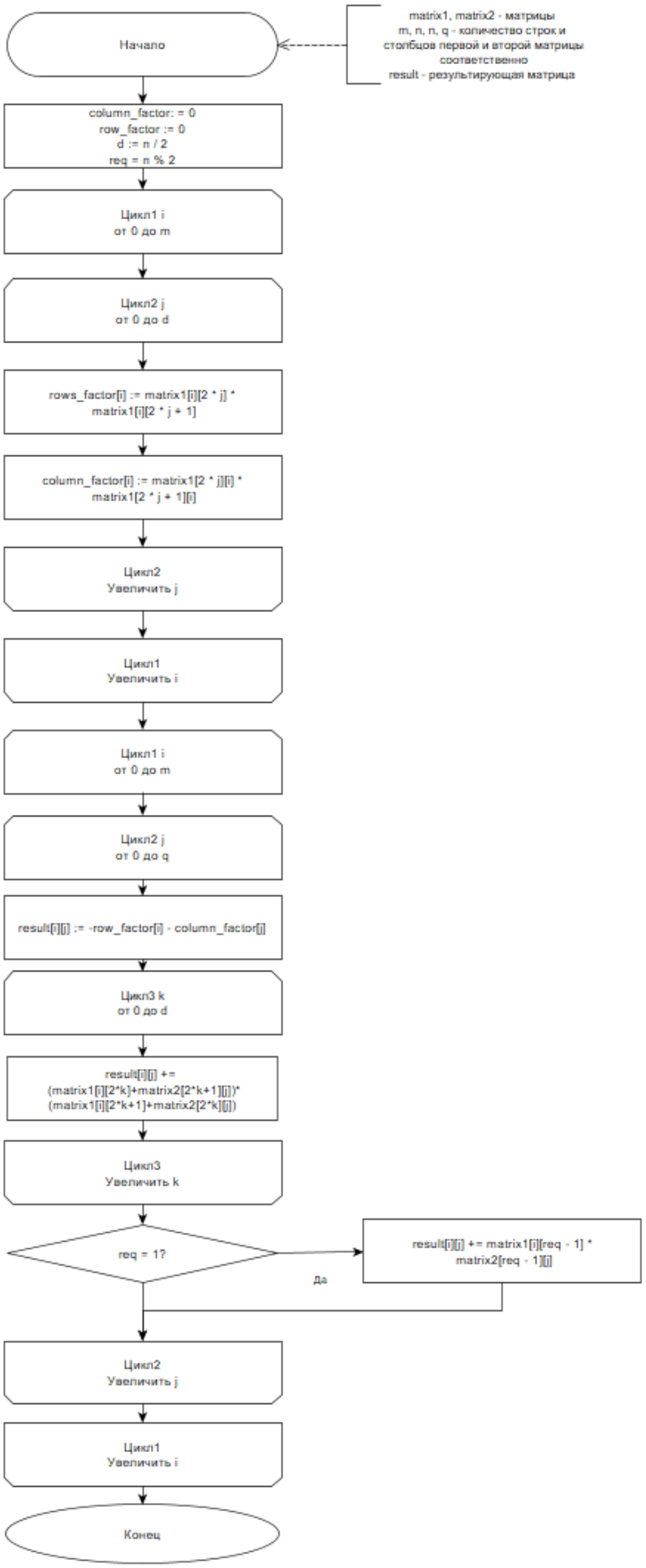
$$13+8M+23QM+21DQM+3Q+12DQ+12DM$$

2.1.3 Оптимизированный алгоритм Винограда

Чтобы оптимизировать алгоритм Винограда, в него было внесено несколько изменений:

1) избавление от деления в цикле;
2) цикл по вычислению элементов в случае нечётной матрицы был объединён с основным циклом алгоритма;
3) накопление результата, чтобы не записывать каждую итерацию результат в переменную, а только после завершения цикла.

Оператор	Количество
Fall	
=	4
/	11
%	
Fbody	
<	1
+=	1
Ffactor	
<	1
+=	1
Ffactor_body	
*	6
[]	10
=	2
+	2
Fmult	
<	1
+=	1
Fmult_outer	
<	1
+=	2
[]	3
Fmult_inner	
<	1
+=	1
Fmult_body	
[]	16
*	5
+	4
=	2
-	2



Fall = 6 + F body + Fmult

F body = 2 + M *(1 + Ffactor)

Ffactor = 2+M(1+Ffactor_body)

Ffactor_body=

(5+1 + 2+ 3 +

+ 5 +2 + 1 + 3+

+1

+1)

Fmult = 2 + M(1+Fmult_outer)

Fmult_outer = 2 +3 +1 +1 + Q(1 + Fmult_inner)

Fmult_inner = 2 + D(1+ Fmult_body)

Fmult_body = 10 + 1 + 2 + 5 +

+ 1 +

+ 1 + 6 + 1 + 2 + 1 +

+ 1 +

+1)

Рассчитаем трудоёмкость для оптимизированного алгоритма Винограда:

$$F_{mult\_body} = 26$$

$$F_{mult\_inner} = 2 + D + 26D$$

$$F_{mult\_outer} = 9 + 3Q + DQ + 26\,DQ$$

$$F_{mult} = 11 + 9M + 3QM + DQM + 26DQM$$

$$F_{factor} = 2 + 24M$$

$$F_{body} = 2 + 2M + 24M*M$$

$$F_{all} = 19 + 11M + 24MM + 3QM + DQM + 26DQM$$

2.2 Выводы по конструкторскому разделу

Были сделаны блок-схемы реализуемых алгоритмов. Рассмотрена оптимизация алгоритма Винограда. Для каждого алгоритма была оценена трудоёмкость.

3 Технологическая часть

В данном разделе был сделан выбор используемого языка программирования, предоставлен код алгоритмов.

3.1 Требования к программному обеспечению

Программа должна поддерживать пользовательский режим, в котором пользователь может с клавиатуры задать размер матриц и значения их элементов; а также операционный режим, в котором пользователь вводит размер матриц, он автоматически заполняется, и для каждого алгоритма выводится время его работы.

3.2 Средства реализации

В качестве языка программирования в данной работе был выбран Forth – конкатентативный язык программирования, в котором программы записываются последовательностью лексем. Этот язык был выбран с целью ознакомления с особенностями его работы. Его стандарт приведён в [9].

3.3 Листинг кода

3.3.1 Классический алгоритм умножения матриц

Листинг 0

```
: [!] ( value index array -- ) swap cells + ! ;
: matrix_multi ( -- . outer to inner:  K J I)
 rows1 0 ?DO
 columns2 0 ?DO
 0
 columns1 0 ?DO
 matrix1 K columns1 * I + cells + @
 matrix2 I columns2 * J + cells + @
 *
 +
 dup K columns2 * J + result [!]
 LOOP
 drop
 LOOP LOOP  ;
```

Заметим, что реализация алгоритмов умножения матриц на Forth требует другой трудоёмкости, нежели было рассмотрено в конструкторском разделе. Рассчитаем трудоёмкость классического умножения матриц на Forth:

$$F_{inner} = 2 + 10 * N$$

$$F_{outer} = 2 + 2 * R + 10 * R * N$$

F body = 2 + 3 * M + 2 * R * M + 10 * R * N

3.3.2 Алгоритм Винограда

Листинг 1

```
: winograd
  rows1 0 ?DO
  rows2 2 / 0 ?DO
  row_factor J cells + @
  matrix1 J columns1 * 2 I * + cells + @
  matrix1 J columns1 * 2 I * 1 + + cells + @
  * +
  J row_factor [!]
  LOOP LOOP
  columns2 0 ?DO
  rows2 2 / 0 ?DO
  column_factor J cells + @
  matrix2 2 I columns2 * * J + cells + @
  matrix2 2 I columns2 * * J + columns2 + cells + @
  * +
  J column_factor [!]
  LOOP LOOP
  rows1 0 ?DO
  columns2 0 ?DO
  row_factor J cells + @ column_factor I cells + @ +
  -1 *
  J columns2 * I + result [!]
  Rows2 2 / 0 ?DO
  result K columns2 * J + cells + @
  matrix1 K columns1 * I 2 * + cells + @
  matrix2 I columns2 2 * * J + columns2 + cells + @
  +
  matrix1 K columns1 * I 2 * + 1 + cells + @
  matrix2 I columns2 2 * * J + cells + @
  + * +
  K columns2 * J + result [!]
  LOOP LOOP LOOP
  rows2 2 MOD 1 = IF
  rows1 0 ?DO
  columns2 0 ?DO
  result J columns2 * I + cells + @
  matrix1 J columns1 * rows2 1 - + cells + @
  matrix2 rows2 1 - columns2 * I + cells + @
```

```
 * +
 J columns2 * I + result [!]
 LOOP LOOP THEN ;
```

3.3.3 Оптимизированный алгоритм Винограда

Листинг 2

```
: opt_winograd
 ( ROW FACTOR )
 rows1 0 ?DO
 d 0 ?DO
 row_factor J cells + @
 matrix1 J columns1 * 2 I * + cells + @
 matrix1 J columns1 * 2 I * 1 + + cells + @
 *
 +
 J row_factor [!]
 column_factor J cells + @
 matrix2 2 I columns2 * * J + cells + @
 matrix2 2 I columns2 * * J + columns2 + cells + @
 *
 +
 J column_factor [!]
 LOOP LOOP

 ( MULTIPLICATION )
 rows1 0 ?DO
 columns2 0 ?DO
 row_factor J cells + @
 column_factor I cells + @
 +
 -1 *
 J columns2 * I + result [!]
 0
 d 0 ?DO

 matrix1 K columns1 * I 2 * + cells + @
 matrix2 I columns2 2 * * J + columns2 + cells + @
 +
 matrix1 K columns1 * I 2 * + 1 + cells + @
 matrix2 I columns2 2 * * J + cells + @
  + *  +
 LOOP
 result J columns2 * I + cells + @
 +
```

```
  J columns2 * I + result [!]
  req 1 = IF
  result J columns2 * I + cells + @
  matrix1 J columns1 * rows2 1 - + cells + @
  matrix2 rows2 1 - columns2 * I + cells + @
  *
  +
  J columns2 * I + result [!]
  THEN
  LOOP LOOP
;
```

Листинг 3 Замер времени

```
: time: ( "word" -- )
utime 2>R ' EXECUTE
utime 2R> D-
<# # # # # # # [CHAR] . HOLD #S #> TYPE ." seconds" ;
```

3.4 Выводы по технологическому разделу

Выбран язык программирования Forth, средство реализации языка Gforth. Была определена функция для замера времени в секундах, а также реализованы алгоритмы умножения матриц. Описаны требования к ПО.

Трудоёмкость алгоритмов отличается от той, что была рассмотрена в конструкторской части, если реализовывать программу на Forth. Так, трудоёмкость классического алгоритма умножения матриц на Forth получается равной F body = 2 + 3 * M + 2 * R * M + 10 * R * N, в то время как трудоёмкость этого же алгоритма на «обычных» языках - F body = 3 + 3 * M + 3 * R * M + 11 * R * N * M.

4 Экспериментальная часть

В данном разделе предоставлены примеры работы программы и сравнительные анализы алгоритмов по времени.

4.1 Примеры работы

```
1. OPERATIONAL MODE.
2. USER MODE.
2
Number of rows of the first matrix:
2
Number of columns of the first matrix:
2
Number of rows of the second matrix:
2
Number of columns of the second matrix:
2
Multip. is possible
INPUT FIRST MATRIX:
Input: 1
Input: 2
Input: 3
Input: 4
INPUT SECOND MATRIX:
Input: 5
Input: 6
Input: 7
Input: 8


FIRST MATRIX:
1 2
3 4
SECOND MATRIX:
5 6
7 8

CLASSIC MATRIX MULT.:
19 22
43 50

WINOGRAD ALGORITHM:
19 22
43 50
rowfact:2 12 colfa:35 48
OPTIMIZED WINOGRAD ALGORITHM:
19 22
43 50
```

Рисунок 4.1 Пример работы программы

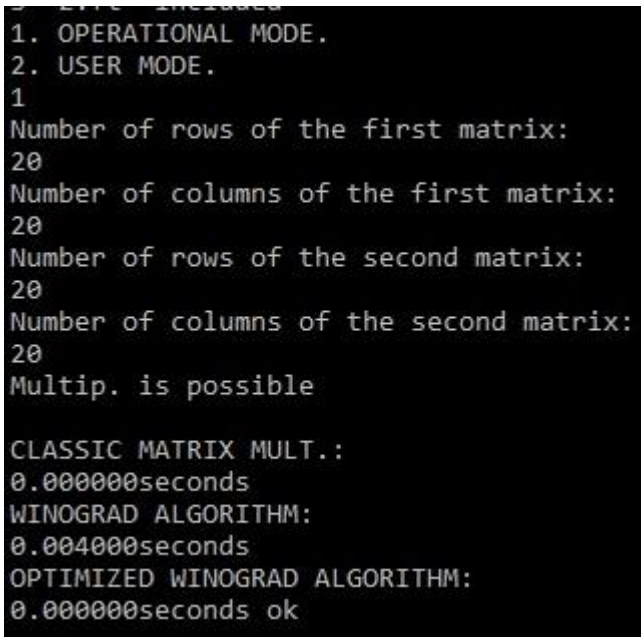


Рисунок 4.2 Пример работы программы

4.2 Постановка эксперимента

Алгоритмы тестируются по времени с помощью функции, возвращающей время работы в секундах, её код представлен в конструкторском разделе.

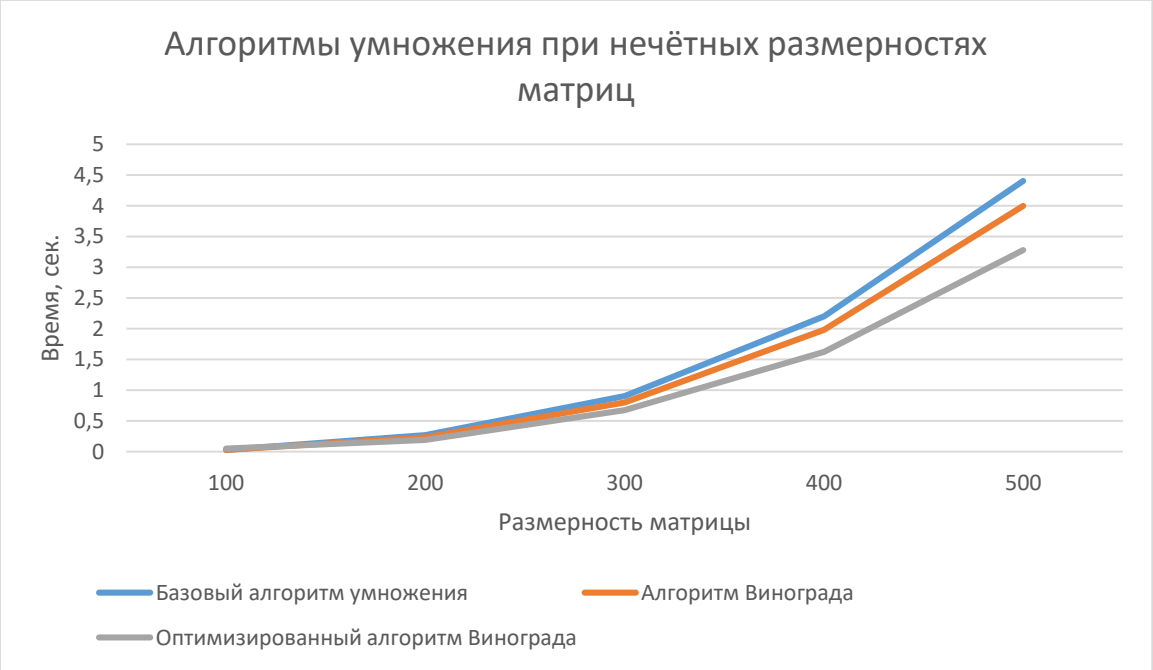
Алгоритмы умножения матриц были протестированы на матрицах размером 100, …, 500 с шагом 100 и на матрицах размером 101, …, 501 с шагом 100.

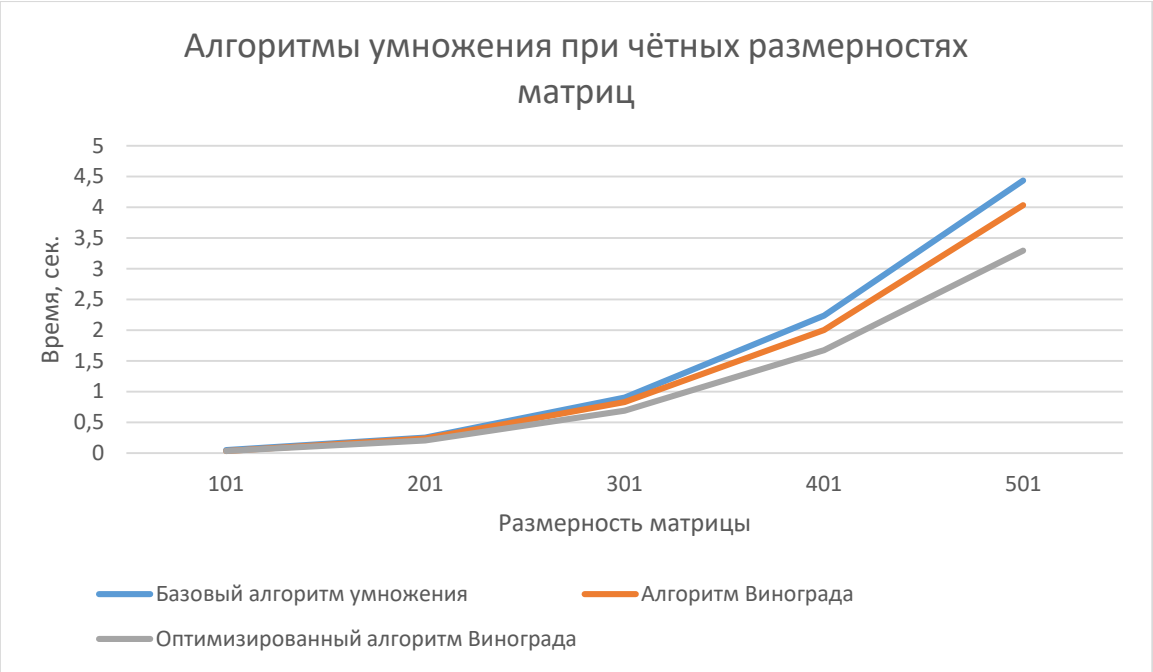
4.2.1 Тестирование времени работы функций

Размерность матриц	Классический алгоритм умножения матриц, сек	Алгоритм Винограда, сек.	Оптимизированный алгоритм Винограда, сек.
100 × 100	0.030	0.030	0.029
200 × 200	0.266	0.234	0.188
300 × 300	0.900	0.797	0.672
400 × 400	2.203	1.984	1.625
500 × 500	4.405	4.000	3.281

Размерность матриц	Классический алгоритм умножения матриц, сек	Алгоритм Винограда, сек.		Оптимизированный алгоритм Винограда, сек.
101 × 101	0.046	0.030		0.030
201 × 201	0.249	0.234		0.204
301 × 301	0.921	0.828		0.703
401 × 401	2.234	2.000		1.671
501 × 501	4.437	4.031		3.312

4.3 Сравнительный анализ на основе экспериментальных данных





4.4 Вывод

Алгоритмы умножения матриц были протестированы на матрицах чётных размеров 100, …, 501 с шагом 100 и на матрицах нечётных размеров 101, …, 501 с шагом 100. Рассмотрены матрицы со случайными значениями.

В результате тестирования было получено, что алгоритмы работают приблизительно одинаково по времени на матрицах маленькой размерности (< 100×100): так, время для классического алгоритма умножения матриц равняется 0.030 для матриц чётной размерности и 0.040 сек. для матриц нечётной размерности; алгоритм Винограда – 0.030 сек. и 0.032 сек. соответственно и оптимизированный алгоритм Винограда 0.029 сек. и 0.030 сек. соответственно. Алгоритмы работают медленнее для матриц нечётных размерностей: уже при размерности 301×301 классический алгоритм отрабатывает за 0.928 сек, алгоритм Винограда за 0.828 сек., а оптимизированный алгоритм Винограда – за 0.703 сек. При этом время работы алгоритмов для матрицы чётной размерности 300×300 равняется 0.900 сек., 0.797 сек и 0.672 сек. Соответственно.

При увеличении размерности матриц увеличивается и время выполнения алгоритмов. Лучшие результаты показывает оптимизированный алгоритм Винограда: при матрицах размерностями 500×500 и 501×501 алгоритм отрабатывает за 3.281 сек. и 3.312 сек., в то время как обычный алгоритм Винограда за 4.000 сек. И 4.031 сек., а классический алгоритм умножения матриц – за 4.405 сек. И 4.437 сек. Соответственно.

Заключение

В результате выполнения данной лабораторной работы были реализованы классический алгоритм умножения матриц и алгоритм Винограда. Была проведена оптимизация алгоритма Винограда, благодаря чему уменьшилось его время работы. Проведено сравнение эффективности алгоритмов по времени: в результате тестирования было получено, что алгоритмы работают приблизительно одинаково по времени на матрицах маленькой размерности (менее 100×100): так, время для классического алгоритма умножения матриц равняется 0.030 для матриц чётной размерности и 0.040 сек. для матриц нечётной размерности; алгоритм Винограда – 0.030 сек. и 0.032 сек. соответственно и оптимизированный алгоритм Винограда 0.029 сек. и 0.030 сек. соответственно. Алгоритмы работают медленнее для матриц нечётных размерностей: уже при размерности 301×301 классический алгоритм отрабатывает за 0.928 сек, алгоритм Винограда за 0.828 сек., а оптимизированный алгоритм Винограда – за 0.703 сек. При этом время работы алгоритмов для матрицы чётной размерности 300×300 равняется 0.900 сек., 0.797 сек и 0.672 сек. Соответственно.

При увеличении размерности матриц увеличивается и время выполнения алгоритмов. Лучшие результаты показывает оптимизированный алгоритм Винограда: при матрицах размерностями 500×500 и 501×501 алгоритм отрабатывает за 3.281 сек. и 3.312 сек., в то время как обычный алгоритм Винограда за 4.000 сек. И 4.031 сек., а классический алгоритм умножения матриц – за 4.405 сек. И 4.437 сек. Соответственно.

Список использованной литературы

1. Белоусов И. В. Матрицы и определители, 2006 г.
2. Корнг Г., Корн Т., Справочник по математике, 1973 г.
3. Хорн Р., Джонсон Ч., Матричный анализ, 2013 г.
4. Погорелов Д. А., Таразанов А. М., Волкова Л. Л., Оптимизация классического алгоритма Винограда для перемножения матриц, 2019 г.
5. Беллман Р. Введение в теорию матриц, 1969.
6. Голуб Дж., Ван Лоун Ч., Матричные вычисления, 1999.
7. Ланкастер П., Теория матриц, 1973.
8. Курош А. Г. Курс высшей алгебры, 1968.
9. Стандарт Forth, [Электронный ресурс], URL: https://forth-standard.org/
10. Л. Броуди, Начальный курс программирования на языке ФОРТ, 1990.
11. Stephen Pelc, Programming Forth Stephen Pelc, 2005.
12. Leo Brodie, Thinking Forth, 1984.
13. J. L. Bezemer, And so forth, 2004.
14. С. Н. Баранов, Н. Р. Ноздрунов, Язык Форт и его реализации, 1988.
15. Язык Форт [Электронный ресурс] URL: https://www.forth.org.ru/
16. Gforth Manual [Электронный ресурс] URL: http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/