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»

Лабораторная работа № 2
по дисциплине «Анализ алгоритмов»

Тема Умножение матриц

Студент Равашдех Ф.Х.

Группа ИУ7-55Б

Преподаватели Строганов Ю.В.

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Матричное умножение — одна из фундаментальных задач в математике и информатике.

Цель лабораторной работы — сравнение алгоритмов умножения матриц — стандартного, Винограда и оптимизированного Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) реализовать алгоритмы указанных методов;
2) провести замеры процессорного времени выполнения реализаций алгоритмов от размерности матриц;
3) провести сравнительный анализ реализаций алгоритмов по полученным экспериментальным данным;
4) описать результаты в отчете.

1 Аналитическая часть

1.1 Стандартный алгоритм

Пусть даны две матрицы A и B размерами $N \times M$ и $M \times K$ (1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mk} \end{pmatrix} \tag{1.1}$$

В результате умножения A на B получается матрица размером $N \times K$ (1.2), где каждый элемент выражается через элементы A и B (1.3).

$$C = A \times B = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nk} \end{pmatrix}, \tag{1.2}$$

$$c_{ij} = a_{i1} \cdot b_{1j} + ... + a_{im} \cdot b_{mj} = \sum_{l=1}^{m} a_{il} \cdot b_{lj} \ \ (i = \overline{1,n},\ j = \overline{1,k}). \tag{1.3}$$

1.2 Алгоритм Винограда

В стандартном алгоритме каждый элемент матрицы произведения представляется как скалярное произведение строки A и столбца B, тогда как в алгоритме Винограда это скалярное произведение преобразуется в эквивалентное выражение (1.4), которое имеет бóльшую трудоемкость, чем у скалярного произведения, но в случае матрицы можно выделить повторяющиеся слагаемые, которые можно посчитать заранее, что дает выигрыш в трудоемкости [3].

Для четного M (для нечетного M добавляется $a_{im} \cdot b_{mj}$):

$$\begin{aligned} c_{ij} = (a_{i1} + b_{2j}) \cdot (a_{i2} + b_{1j}) + (a_{i3} + b_{4j}) \cdot (a_{i4} + b_{3j}) + ... + (a_{im-1} + b_{mj}) \cdot (a_{im} + b_{m-1j}) - \\ - a_{i1} \cdot a_{i2} - a_{i3} \cdot a_{i4} - ... - a_{im-1} \cdot a_{im} - b_{1j} \cdot b_{2j} - b_{3j} \cdot b_{4j} - ... - b_{m-1j} \cdot b_{mj} = \\ = \sum_{l=1}^{m/2} ((a_{i2l-1} + b_{2lj}) \cdot (a_{i2l} + b_{2l-1j}) - a_{i2l-1} \cdot a_{i2l} - b_{2l-1j} \cdot b_{2lj}) \ \ (i = \overline{1,n},\ j = \overline{1,k}) \end{aligned} \tag{1.4}$$

Вывод

В данном разделе рассмотрены алгоритмы умножения матриц.

2 Конструкторская часть

2.1 Стандартный алгоритм

Схема стандартного алгоритма умножения матриц представлена на рисунке 2.1.

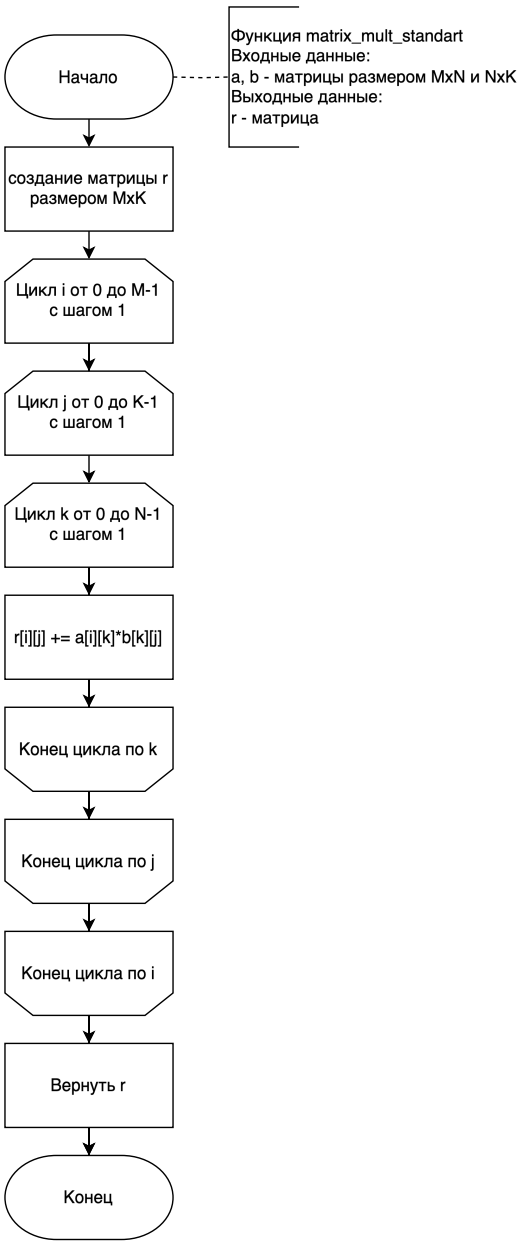

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

2.2 Алгоритм Винограда

Схема алгоритма Винограда умножения матриц представлена на рисунках 2.2 и 2.3.

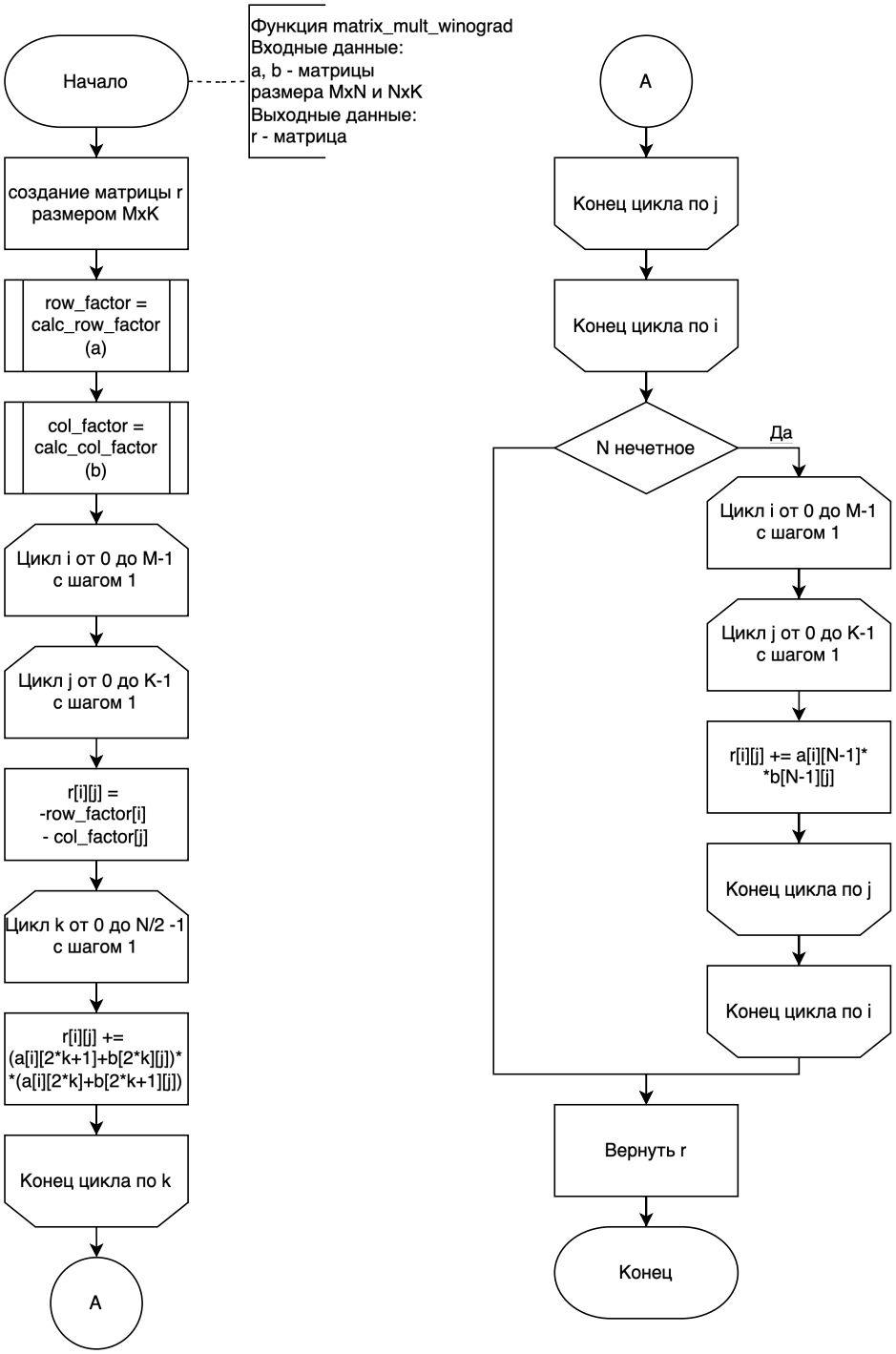

Рисунок 2.2 — Схема алгоритма Винограда умножения матриц

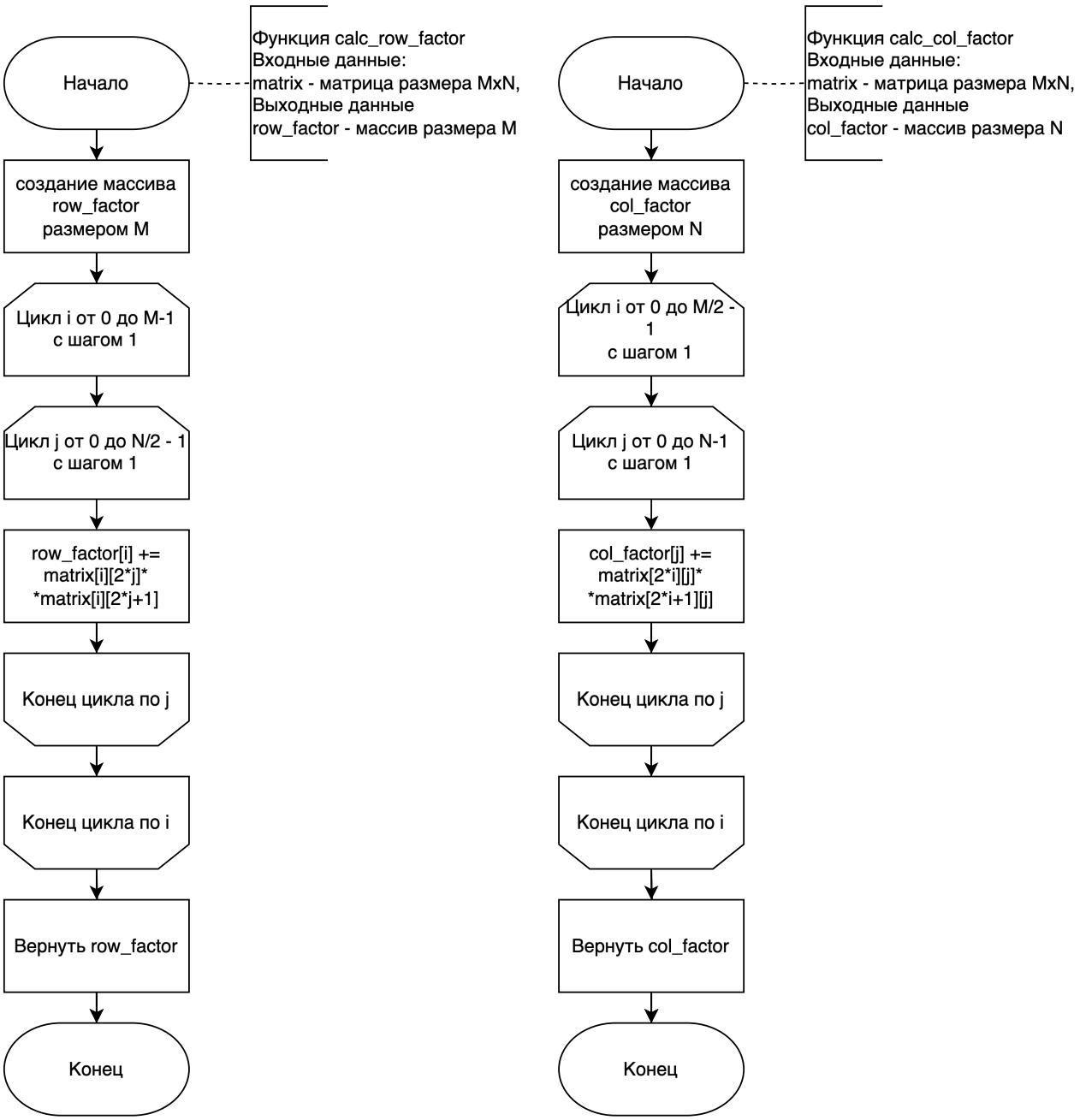

Рисунок 2.3 — Схема алгоритма Винограда умножения матриц

2.3 Оптимизированный алгоритм Винограда

Схема оптимизированного алгоритма Винограда умножения матриц представлена на рисунках 2.4 и 2.5.

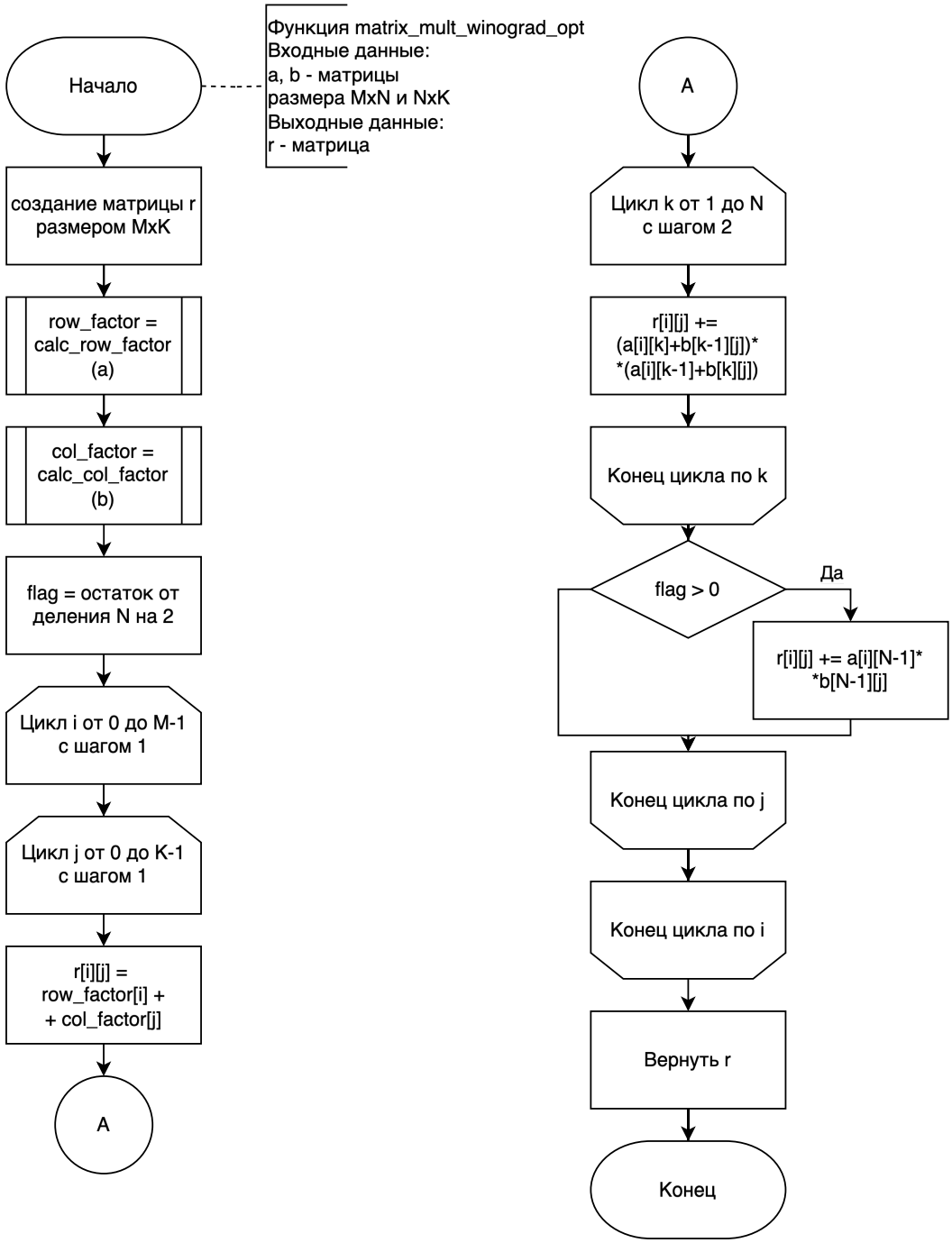

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда умножения матриц

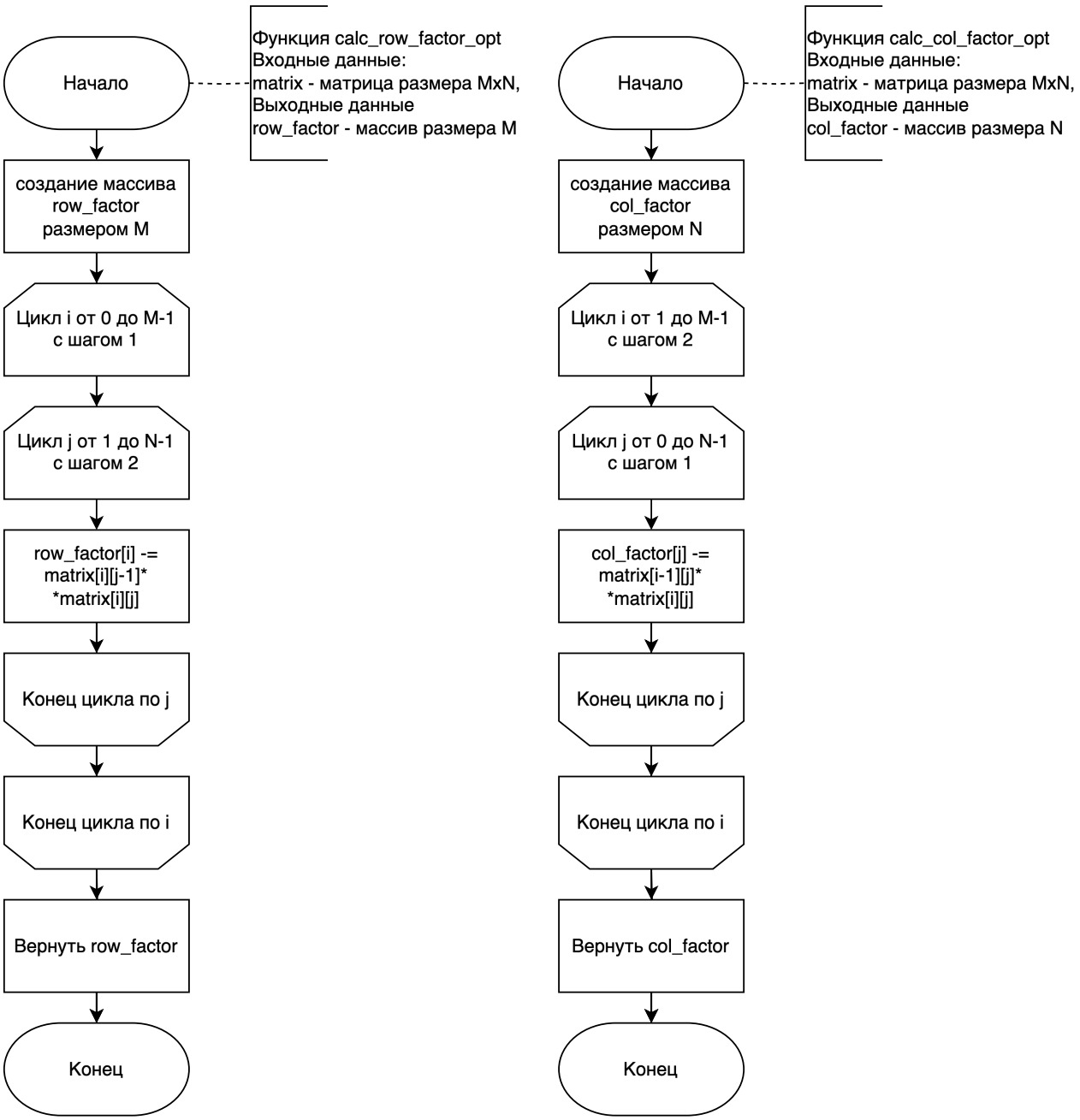

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда умножения матриц

2.4 Трудоемкость алгоритмов

Операции $+, -, +=, -=, []$ имеют трудоемкость 1.

Операции $*, /, \%$ имеют трудоемкость 2.

Трудоемкость условного оператора определяется формулой 2.1

$$f_{\text{усл.оп.}} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.1}$$

Трудоемкость цикла определяется формулой 2.2.

$$f_{\text{цикл}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.2}$$

2.4.1 Трудоемкость стандартного алгоритма

Стандартный алгоритм состоит из трех циклов. Третий по вложенности цикл имеет трудоемкость $2+11 \cdot N$. Тогда второй по вложенности имеет трудоемкость $2+K \cdot (2+(2+11 \cdot N))$. Тогда внешний цикл имеет трудоемкость $f = 2 + M \cdot (2 + (2 + K \cdot (2 + (2 + 11 \cdot N))))$. Т. е. трудоемкость всего алгоритма равна $f_{\text{std}} = 2 + 4 \cdot M + 4 \cdot M \cdot K + 11 \cdot M \cdot K \cdot N$.

2.4.2 Трудоемкость алгоритма Винограда

Трудоемкость заполнения вспомогательного массива row_factor равна $2+M \cdot (6+\frac{N}{2} \cdot 17)$. Трудоемкость заполнения вспомогательного массива col_factor равна $4 + \frac{N}{2} \cdot (6 + K \cdot 15)$. Трудоемкость наиболее вложенного цикла равна $4 + \frac{N}{2} \cdot 27$. Трудоемкость основного цикла равна $2 + M \cdot (2 + 2 + K \cdot (2 + 7 + (4 + \frac{N}{2} \cdot 29)))$. Трудоемкость условного оператора равна 2.3.

$$2 + \begin{cases} 0, & \text{размер четный,} \\ 2 + M \cdot (2 + 13K), & \text{иначе.} \end{cases} \tag{2.3}$$

Т. е. трудоемкость всего алгоритма равна 2.4.

$$f_{\text{Винограда}} = (2 + 6M + 8.5 \cdot M \cdot N) + (4 + 3N + 7.5 \cdot N \cdot K) +$$

$$+(2 + 4M + 13 \cdot M \cdot K + 14.5 \cdot M \cdot K \cdot N) + (2 + \begin{cases} 0, & \text{N четно} \\ 2 + 2M + 13 \cdot M \cdot K, & \text{иначе} \end{cases}) =$$

$$= \begin{cases} 10 + 10M + 3N + 8.5 \cdot M \cdot N + 7.5 \cdot N \cdot K + 13 \cdot M \cdot K + 14.5 \cdot M \cdot K \cdot N, & \text{л.с.} \\ 12 + 12M + 3N + 8.5 \cdot M \cdot N + 7.5 \cdot N \cdot K + 26 \cdot M \cdot K + 14.5 \cdot M \cdot K \cdot N, & \text{х. с.} \end{cases} \tag{2.4}$$

2.4.3 Трудоемкость оптимизированного алгоритма винограда

Трудоемкость заполнения вспомогательного массива row_factor равна $2+M\cdot(4+\frac{N}{2}\cdot 11)$. Трудоемкость заполнения вспомогательного массива col_factor равна $2+\frac{N}{2}\cdot(4+K\cdot 11)$. Трудоемкость вычисления флага для условия равна 3. Трудоемкость наиболее вложенного цикла равна $2+\frac{N}{2}\cdot 19$. Трудоемкость условного оператора равна 2.5.

$$1+\begin{cases}0, & \text{размер четный,}\\ 11, & \text{иначе.}\end{cases} \tag{2.5}$$

Трудоемкость основного цикла равна 2.6.

$$2+M\cdot(2+2+K\cdot(2+6+(2+\frac{N}{2}\cdot 19)+1+\begin{cases}0, & \text{размер четный,}\\ 11, & \text{иначе.}\end{cases})) \tag{2.6}$$

Т. е. трудоемкость всего алгоритма равна 2.7.

$$\begin{aligned} f_{\text{Опт. Винограда}} &= (2+4M+5.5\cdot M\cdot N)+(2+2N+5.5\cdot N\cdot K)+\\ &+(2+4M+11\cdot M\cdot K+9.5\cdot M\cdot K\cdot N+\begin{cases}0, & \text{N четно}\\ 11\cdot M\cdot K, & \text{иначе}\end{cases}= \\ &= \begin{cases}6+8M+2N+5.5\cdot M\cdot N+5.5\cdot N\cdot K+11\cdot M\cdot K+9.5\cdot M\cdot K\cdot N, & \text{л.с.}\\ 6+8M+2N+5.5\cdot M\cdot N+5.5\cdot N\cdot K+22\cdot M\cdot K+9.5\cdot M\cdot K\cdot N, & \text{х. с.}\end{cases}\end{aligned} \tag{2.7}$$

Вывод

В данной части работы были описаны стандартный алгоритм, алгоритм Винограда, оптимизированный алгоритм Винограда умножения матриц. Оценены их трудоемкости.

3 Технологическая часть

3.1 Требования к программному обеспечению

Входные данные: две матрицы и размеры матриц.

Выходные данные: матрица и размеры матрицы.

3.2 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C с использованием функции $clock$ [1] для замера процессорного времени. Графики были созданы с помощью библиотеки для визуализации данных $Matplotlib$ [2].

3.3 Реализация алгоритмов

Реализация алгоритмов умножения матриц стандартным методом, методом Винограда и оптимизированным методом Винограда представлена в листингах 3.1, 3.2 и 3.3.

Листинг 3.1 — Функция умножения матриц стандартным методом

```
matrix_error_t matrix_mult_standart(const matrix_t *a, const matrix_t *
   b, matrix_t **r)
{
  matrix_error_t rc = MATRIX_OK;
  if (!a || !b || a->cols != b->rows)
    rc = MATRIX_INVALID_PARAM;
  else if (!( *r = matrix_create(a->rows, b->cols)))
    rc = MATRIX_MEM;
  else
  {
    for (size_t i = 0; i < a->rows; i++)
      for (size_t j = 0; j < b->cols; j++)
        for (size_t k = 0; k < a->cols; k++)
          ( *r)->data[i][j] += a->data[i][k] * b->data[k][j];
  }
  return rc;
}
```

Листинг 3.2 — Функция умножения матриц методом Винограда

```
matrix_error_t calc_row_factor(const matrix_t *matrix, double **
   row_factor)
{
  matrix_error_t rc = MATRIX_OK;
  if (matrix == NULL || matrix->data == NULL || row_factor == NULL || *
     row_factor != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!( *row_factor = calloc(matrix->rows, sizeof(double))))
    rc = MATRIX_MEM;
  else
    for (size_t i = 0; i < matrix->rows; i++)
      for (size_t j = 0; j < matrix->cols / 2; j++)
        ( *row_factor)[i] += matrix->data[i][2 * j] * matrix->data[i][2
            * j + 1];
  return rc;
}

matrix_error_t calc_col_factor(const matrix_t *matrix, double **
   col_factor)
{
  matrix_error_t rc = MATRIX_OK;
  if (matrix == NULL || matrix->data == NULL || col_factor == NULL || *
     col_factor != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!( *col_factor = calloc(matrix->cols, sizeof(double))))
    rc = MATRIX_MEM;
  else
    for (size_t i = 0; i < matrix->rows / 2; i++)
      for (size_t j = 0; j < matrix->cols; j++)
        ( *col_factor)[j] += matrix->data[2 * i][j] * matrix->data[2 *
           i + 1][j];
  return rc;
}

matrix_error_t matrix_mult_winograd(const matrix_t *a, const matrix_t *
   b, matrix_t **r)
{
  matrix_error_t rc = MATRIX_OK;
  double *row_factor = NULL, *col_factor = NULL;
  if (a == NULL || a->data == NULL || b == NULL || b->data == NULL || a
```

```
    ->cols != b->rows || r == NULL || *r != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!( *r = matrix_create(a->rows, b->cols)))
    rc = MATRIX_MEM;
  else if ((rc = calc_row_factor(a, &row_factor)) != MATRIX_OK)
    matrix_free(r);
  else if ((rc = calc_col_factor(b, &col_factor)) != MATRIX_OK)
    matrix_free(r), free(row_factor);
  else
  {
    for (size_t i = 0; i < a->rows; i++)
    {
      for (size_t j = 0; j < b->cols; j++)
      {
        ( *r)->data[i][j] = -row_factor[i] - col_factor[j];
        for (size_t k = 0; k < a->cols / 2; k++)
          ( *r)->data[i][j] += (a->data[i][2 * k + 1] + b->data[2 * k][
             j]) * (a->data[i][2 * k] + b->data[2 * k + 1][j]);
      }
    }
    if (a->cols % 2)
      for (size_t i = 0; i < a->rows; i++)
        for (size_t j = 0; j < b->cols; j++)
          ( *r)->data[i][j] += a->data[i][a->cols - 1] * b->data[b->
             rows - 1][j];
    free(row_factor);
    free(col_factor);
  }
  return rc;
}
```

Листинг 3.3 — Функция умножения матриц оптимизированным методом Винограда

```
matrix_error_t calc_row_factor_opt(const matrix_t *matrix, double **
   row_factor)
{
  matrix_error_t rc = MATRIX_OK;
  if (matrix == NULL || matrix->data == NULL || row_factor == NULL || *
     row_factor != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!( *row_factor = calloc(matrix->rows, sizeof(double))))
    rc = MATRIX_MEM;
```

```
  else
    for (size_t i = 0; i < matrix->rows; i++)
      for (size_t j = 1; j < matrix->cols; j += 2)
        (*row_factor)[i] -= matrix->data[i][j - 1] * matrix->data[i][j
          ];
  return rc;
}

matrix_error_t calc_col_factor_opt(const matrix_t *matrix, double **
   col_factor)
{
  matrix_error_t rc = MATRIX_OK;
  if (matrix == NULL || matrix->data == NULL || col_factor == NULL || *
     col_factor != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!(*col_factor = calloc(matrix->cols, sizeof(double))))
    rc = MATRIX_MEM;
  else
    for (size_t i = 1; i < matrix->rows; i += 2)
      for (size_t j = 0; j < matrix->cols; j++)
        (*col_factor)[j] -= matrix->data[i - 1][j] * matrix->data[i][j
          ];
  return rc;
}

matrix_error_t matrix_mult_winograd_opt(const matrix_t *a, const
   matrix_t *b, matrix_t **r)
{
  matrix_error_t rc = MATRIX_OK;
  double *row_factor = NULL, *col_factor = NULL;
  if (a == NULL || a->data == NULL || b == NULL || b->data == NULL || a
     ->cols != b->rows || r == NULL || *r != NULL)
    rc = MATRIX_INVALID_PARAM;
  else if (!(*r = matrix_create(a->rows, b->cols)))
    rc = MATRIX_MEM;
  else if ((rc = calc_row_factor_opt(a, &row_factor)) != MATRIX_OK)
    matrix_free(r);
  else if ((rc = calc_col_factor_opt(b, &col_factor)) != MATRIX_OK)
    matrix_free(r), free(row_factor);
  else
  {
```

```
    int flag = a->cols % 2;
    for (size_t i = 0; i < a->rows; i++)
    {
      for (size_t j = 0; j < b->cols; j++)
      {
        ( *r)->data[i][j] = row_factor[i] + col_factor[j];
        for (size_t k = 1; k < a->cols; k += 2)
          ( *r)->data[i][j] += (a->data[i][k] + b->data[k - 1][j]) * (a
             ->data[i][k - 1] + b->data[k][j]);
        if (flag)
          ( *r)->data[i][j] += a->data[i][a->cols - 1] * b->data[b->
             rows - 1][j];
      }
    }
    free(row_factor);
    free(col_factor);
  }
  return rc;
}
```

3.4 Тестирование

В таблице 3.1 представлены данные о результатах тестирования реализации алгоритмов умножения матриц.

Таблица 3.1 — Модульные тесты

Матрица A	Матрица B	Результат
$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$	$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$	$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$
$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$	$\begin{pmatrix}9&8&7\\6&5&4\\3&2&1\end{pmatrix}$	$\begin{pmatrix}30&24&18\\84&69&54\\138&114&90\end{pmatrix}$
$\begin{pmatrix}1&1&1\end{pmatrix}$	$\begin{pmatrix}1\\1\\1\end{pmatrix}$	$\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix}$

Все тесты пройдены успешно.

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, используемые средства реализации, приведена реализация алгоритмов и результаты тестирования.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики устройства, на котором выполнялись замеры:

1) операционная система — macOS Sonoma 14.1 (23B2073);

2) процессор — Apple M3;

3) оперативная память — 16 Гб.

4.2 Типы данных

Матрица — $matrix\_t$; Тип ошибок — $matrix\_error\_t$.

4.3 Время выполнения реализаций алгоритмов

Замеры времени работы реализаций алгоритмов для каждого размера матриц проводились 100 раз, значение времени усреднялось. Данные замеров представлены в таблице 4.1.

Таблица 4.1 — Таблица замеров времени работы алгоритмов (в мс)

Размер	Стандартный	Винограда	Оптимизированный Винограда
10	0.00934	0.00978	0.00626
20	0.04994	0.03692	0.02672
30	0.11589	0.07702	0.05016
40	0.21056	0.12809	0.11678
50	0.34473	0.22543	0.20771
60	0.58648	0.37270	0.35031
70	0.93209	0.59764	0.54972
80	1.37477	0.92047	0.82010
90	1.94044	1.32322	1.19331
100	2.65325	1.76405	1.64636
110	3.53173	2.36100	2.16105
120	4.54108	2.99517	2.79426
130	5.91079	3.87216	3.53079
140	7.36001	4.76333	4.34239
150	9.18645	5.93133	5.41438
160	10.95094	7.06727	6.52982
170	13.37995	8.55917	7.91273
180	15.92595	9.97375	9.30066
190	18.92719	11.87942	10.79745
200	21.91092	13.64643	12.61243
210	25.83687	15.94249	14.57796
220	29.91168	18.10955	16.70312
230	34.30373	20.82617	19.24681
240	38.81086	23.48579	21.71082
250	47.14721	26.66110	24.61888
260	50.02556	29.69702	27.84935
270	59.25041	33.38113	31.85158
280	62.57422	37.20468	35.80270
290	69.20933	41.30339	40.40267
300	80.53249	45.57732	44.66144

На рисунке 4.1 показаны графики зависимости времени работы реализаций алгоритмов умножения матриц от размера матриц. На рисунке 4.2 показаны графики зависимости времени работы реализаций алгоритмов умножения матриц методом Винограда от размера матриц.

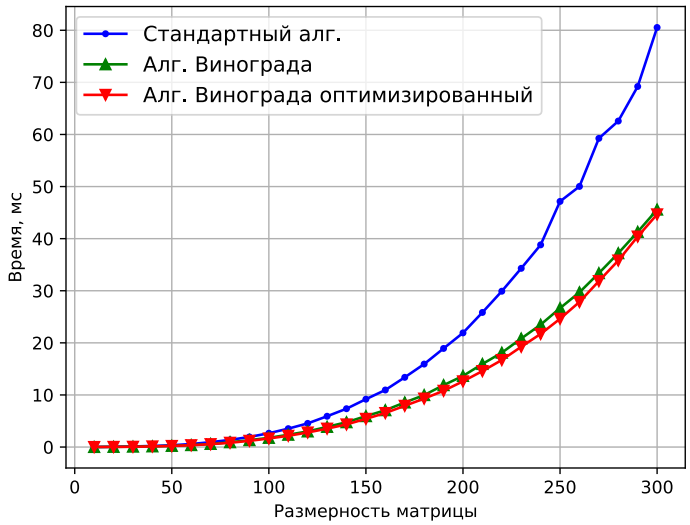

Рисунок 4.1 — Графики сравнения выполнения реализаций алгоритмов по времени

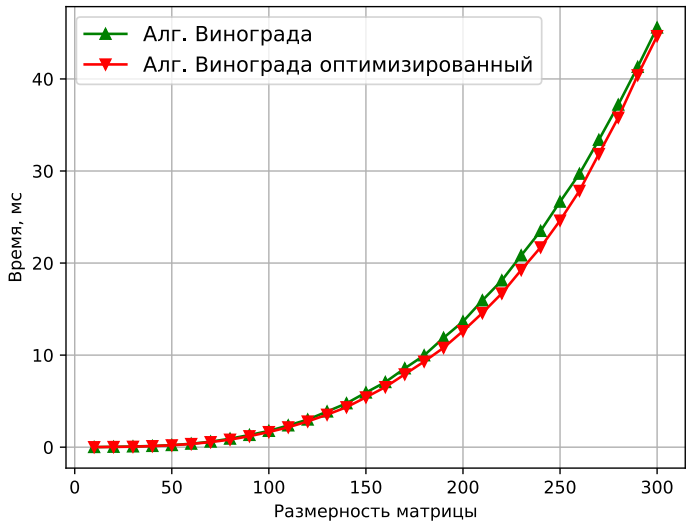

Рисунок 4.2 — Графики сравнения выполнения реализаций алгоритмов метода Винограда по времени

4.4 Вывод

В данном разделе были приведены технические характеристики и время выполнения реализаций алгоритмов. Стандартный алгоритм умножения матриц выполнялся дольше других алгоритмов. Неоптимизированный алгоритм Винограда выполнялся дольше оптимизированного.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: сравнение алгоритмов умножения матриц стандартным методом, методом Винограда и оптимизированным методом Винограда было проведено.

В ходе выполнения лабораторной работы были решены все задачи:

1) реализованы алгоритмы указанных методов;

2) провести замеры процессорного времени выполнения реализаций алгоритмов от размерности матриц;

3) проведен сравнительный анализ реализаций алгоритмов по полученным экспериментальным данным;

4) описаны результаты в отчете.

Сравнительный анализ алгоритмов показал:

1) самый быстрый — алгоритм умножения матриц оптимизированным методом Винограда;

2) самый медленный — алгоритм умножения матриц стандартным методом;

3) алгоритм умножения матриц неоптимизированным методом Винограда медленнее оптимизированного метода Винограда.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Стандарт C99 языка Си: 7.23.2.1 Функция замера процессорного времени clock [Электронный ресурс]. Режим доступа: http://www.man6.org/lib/pdfjs/web/viewer.html?file=/blog/PdfFile/c99InternationalStandard.pdf (дата обращения 11.12.24)

2. Документация библиотеки Matplotlib [Электронный ресурс]. Режим доступа: https://matplotlib.org/ (дата обращения 13.10.24)

3. Анисимов Н.С., Строганов Ю.В. Реализация алгоритма умножения матриц по Винограду на языке Haskell // Новые информационные технологии в автоматизированных системах – 2018 – С. 390-395.