МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Отчет по учебной практике**

**«Структура хранения матриц специального вида»**

**Выполнил:** студент группы 3822Б1ПР2

Соловьева Елизавета Андреевна

Нижний Новгород

2023.

**Содержание**

[1. Введение 3](#_Toc152163277)

[2. Постановка задачи 4](#_Toc152163278)

[3. Руководство пользователя 5](#_Toc152163279)

[4. Руководство программиста 6](#_Toc152163280)

[4.1. Описание структуры программы 6](#_Toc152163281)

[4.2. Описание кода 6](#_Toc152163282)

[4.3. Описание алгоритмов 9](#_Toc152163283)

[4.4 Код 10](#_Toc152163284)

[5. Эксперименты 19](#_Toc152163285)

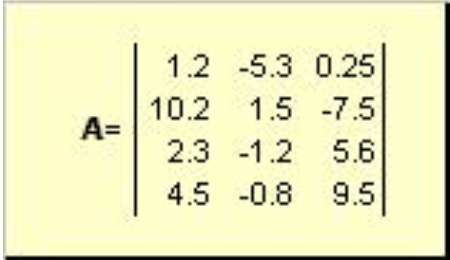
[6. Заключение 20](#_Toc152163286)

[7. Литература 21](#_Toc152163287)

# Введение

Матрицы и векторы являются основными понятиями в линейной алгебре и играют важную роль в различных областях науки и техники. Они представляют собой математические объекты, которые используются для описания и решения разнообразных задач, таких как системы линейных уравнений, анализ данных, оптимизация и многие другие.

Матрицей называется прямоугольная таблица чисел, например, (рис 1):



(Рис 1)

Матрицы обозначаются заглавными полужирными буквами (A), а их элементы — соответствующими строчными буквами с индексами, т.е. aij. Первый индекс нумерует строки, а второй — столбцы. В хемометрике принято обозначать максимальное значение индекса той же буквой, что и сам индекс, но заглавной. Поэтому матрицу A можно также записать как { aij, i = 1,..., I; j = 1,..., J}. Для приведенной в примере матрицы I = 4, J = 3 и a23 = −7.5.

Виды матриц:

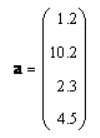
1. Квадратные матрицы - матрицы, у которых количество строк равно количеству столбцов.

2. Симметричные матрицы - матрицы, которые равны своему транспонированию.

3. Диагональные матрицы - матрицы, у которых все элементы вне главной диагонали равны нулю.

4. Единичные матрицы - квадратные матрицы, у которых на главной диагонали стоят единицы, а все остальные элементы равны нулю.

5. Нулевые матрицы - матрицы, у которых все элементы равны нулю. В математике рассматривается множество различных типов и видов матриц. Таковы, например, единичная, симметричная, кососимметричная, верхнетреугольная (нижнетреугольная) и т. п. матрицы.

Если матрица состоит только из одного столбца (J = 1), то такой объект называется вектором. Точнее говоря, вектором-столбцом. Например, (рис 2):

(Рис 2)

# Постановка задачи

В лабораторной работе необходимо выполнить следующее:

1. Разработать и реализовать шаблонный класс векторов TDynamicVector
   1. Конструктор копирования
   2. Конструктор перемещения
   3. Операции сравнения (равенство, неравенство)
   4. Скалярные операции (сложение, вычитание, умножение)
   5. Векторные операции (сложение, вычитание, умножение)
   6. Ввод и Вывод в поток
2. Разработать и реализовать шаблонный класс матриц TDynamicMatrix
   1. Операции сравнения (равенство, неравенство)
   2. Скалярные операции (сложение, вычитание, умножение)
   3. Матрично-скалярные операции
   4. Матрично-векторные операции
   5. Матрично-матричные операции
   6. Ввод и Вывод в поток
3. Написать пример программы, демонстрирующей работу данных двух классов
4. Написать наборы тестов с использованием Google C++ Testing Framework и проверить работоспособность методов классов.

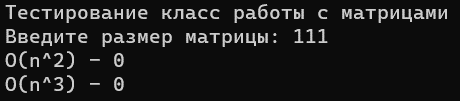
# Руководство пользователя

Как только запускается программа, мы видим окно (рис 3) и предлагается ввести произвольный размер матрицы:



(Рис 3)

Далее, пользователь вводит размер матрицы. После программа считает время выполнения сложения и умножения в секундах (рис 4):



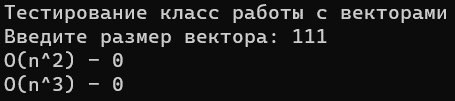
(Рис 4)

Далее, мы видим окно (рис 5) и предлагается ввести произвольный размер вектора:



(Рис 5)

Далее, пользователь вводит размер вектора. После программа считает время выполнения сложения и умножения в секундах (рис 6):



(Рис 6)

# Руководство программиста

## Описание структуры программы

Программа состоит из модулей:

* Модуль *sample\_utmatrix*. Включает в себя заголовочный файл *tmatrix.h*, в котором описаны методы с реализацией шаблонных классов *TDynamicMatrix* и *TDynamicVector*. И файл с реализацией sample\_matrix.cpp
* Модуль *test\_utmatrix*. Набор тестов для обоих классов. Включает в себя файлы *test\_tmatrix.cpp* и *test\_tvector.cpp.* Реализованы они с помощью использования фреймворка Google Test.

## Описание кода

Класс вектор:

Класс вектор – шаблонный класс с изменяемым типом хранения содержащий:

Поля:

* Size\_t sz – размер вектора
* T\* pMem – указатель шаблонный тип для выделения памяти для массива

Методы:

* TDynamicVector(size\_t size = 1) : sz(size) – Конструктор по умолчанию
* TDynamicVector(T\* arr, size\_t s) : sz(s) – Конструктор инициализации
* TDynamicVector(const TDynamicVector& v) – Конструктор копирования
* TDynamicVector(TDynamicVector&& v) noexcept – Конструктор перемещения
* TDynamicVector& operator=(const TDynamicVector& v) – Оператор присваивания
* TDynamicVector& operator=(TDynamicVector&& v) noexcept – Перемещающий оператор присваивания
* size\_t size() const noexcept – Метод возвращающий размер вектора
* T& operator[](size\_t ind) – Оператор индексирования без зашиты
* const T& operator[](size\_t ind) const – Константный оператор индексации без защиты
* T& at(size\_t ind) Оператор индексирования с зашитой
* const T& at(size\_t ind) const Константный оператор индексации с защиты
* bool operator==(const TDynamicVector& v) const noexcept - Оператор сравнения векторов
* bool operator!=(const TDynamicVector& v) const noexcept- Оператор сравнения векторов
* TDynamicVector operator+(T val) – Скалярный оператор сложения вектора с константой
* TDynamicVector operator-(T val) – Скалярный оператор вычитания вектора с константой
* TDynamicVector operator\*(T val) – Скалярный оператор умножения вектора с константой
* TDynamicVector operator+(const TDynamicVector& v) – Векторный оператор сложения двух векторов
* TDynamicVector operator-(const TDynamicVector& v) – Векторный оператор вычитания двух векторов
* T operator\*(const TDynamicVector& v) noexcept(noexcept(T())) – Векторный оператор умножения двух векторов
* friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept – Дружественная функция обменом значения между двумя векторами
* friend istream& operator>>(istream& istr, TDynamicVector& v) – Дружественная функция ввода
* friend ostream& operator<<(ostream& ostr, const TDynamicVector& v) – Дружественная функция вывода

Класс матрица:

* TDynamicMatrix(size\_t s = 1) : TDynamicVector<TDynamicVector<T>>(s) – Конструктор инициализации
* bool operator==(const TDynamicMatrix& m) const noexcept – Оператор сравнения двух матриц
* bool operator!=(const TDynamicMatrix& m) const – Оператор сравнения двух матриц
* TDynamicMatrix operator\*(const T& val) – Скалярный оператор умножения матрицы на скаляр
* TDynamicVector<T> operator\*(const TDynamicVector<T>& v) – Векторное операция умножения матрицы на вектор
* TDynamicMatrix operator+(const TDynamicMatrix& m) – Матричная операция сложения двух матриц
* TDynamicMatrix operator-(const TDynamicMatrix& m) – Матричная операция разности двух матриц
* TDynamicMatrix operator\*(const TDynamicMatrix& m) – Матричная операция умножения двух матриц
* friend istream& operator>>(istream& istr, TDynamicMatrix& v) – Дружественная функция ввода матрицы
* friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v) – Дружественная функция вывода матрицы
* size\_t size() const noexcept – Метод получения размера матрицы
* T& at(size\_t ind1, size\_t ind2) – Оператор индексирования с зашитой
* const T& at(size\_t ind) const – Константный оператор индексирования с зашитой

## Описание алгоритмов

Пусть заданы два вектора A = (a1, a2, …, an) и B = (b1, b2, …, bn).

Рассмотрим следующие основные операции над векторами:

* Сравнение (A = B). Вектора считаются равными тогда и только тогда, когда ai = bi при всех i = 1..n.
* Прибавление скаляра (A + a). Результатом сложения вектора A и скаляра а называется вектор A’ = (a1 + a, a2 + a, …, an + a).
* Вычитание скаляра (A – a). Результатом вычитания вектора A и скаляра а называется вектор A’ = (a1 – a, a2 – a, …, an – a).
* Умножение на скаляр (A \* a). Результатом умножения вектора A на скаляр а называется вектор A’ = (a1 \* a, a2 \* a, …, an \* a).
* Сложение векторов (A + B). Результатом сложения векторов A и B называется вектор С = (a1 + b1, a2 + b2, …, an + bn ).
* Вычитание векторов (A – B). Результатом вычитания векторов A и B называется вектор С = (a1 – b1, a2 – b2, …, an – bn).
* Скалярное произведение векторов (A \* B). Скалярным произведением векторов A и B называется скалярная величина с =

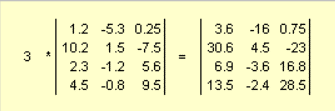
Пусть заданы две матрицы A = (aij) и B = (bij) и вектор V = (vj), где i = 1..m; j = 1..n.

Рассмотрим следующие основные операции над матрицами:

* Сравнение (A = B). Матрицы считаются равными тогда и только тогда, когда aij =bij при всех i = 1..m; j = 1..n.
* Умножение на скаляр (A \* d). Результатом умножения матрицы A на скаляр d называется матрица D = (dij), где dij = aij \* d, при всех i = 1..m; j = 1..n.
* Умножение на вектор (A \* V). Результатом умножения матрицы A на вектор V называется вектор D = (di), где di = , при всех i = 1..m; j = 1..n.
* Сложение матриц (A + B). Результатом сложения матриц A и B называется матрица C = (cij), где cij = aij + bij при всех i = 1..m; j = 1..n.
* Вычитание матриц (A – B). Результатом вычитания матриц A и B называется матрица C = (cij), где cij = aij - bij при всех i = 1..m; j = 1..n.
* Умножение матриц (A \* B). Результатом умножения матриц A и B

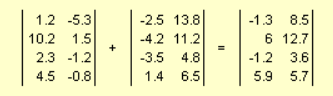
называется матрица C, где cij = aik \* bkj при всех i = 1..m; j = 1..n; k = 1..n.

Матрицы можно умножать на числа. При этом каждый элемент умножается на это число. Например (рис 7):



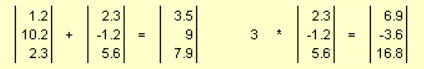
(Рис 7)

Две матрицы одинаковой размерности можно поэлементно складывать и вычитать. Например (рис 8):



(Рис 8)

Векторы можно складывать и умножать на числа так же, как это делается с матрицами. Например (рис 9):



(Рис 9)

## 4.4 Код

Tmatrix.h

#ifndef \_\_TDynamicMatrix\_H\_\_

#define \_\_TDynamicMatrix\_H\_\_

#include <iostream>

using namespace std;

const int MAX\_VECTOR\_SIZE = 100000000;

const int MAX\_MATRIX\_SIZE = 10000;

// Динамический вектор -

// шаблонный вектор на динамической памяти

template<typename T>

class TDynamicVector

{

protected:

size\_t sz;//размер

T\* pMem; //память

public:

TDynamicVector(size\_t size = 1) : sz(size) // конструктор с параметром

{

if (sz == 0)

throw out\_of\_range("Vector size should be greater than zero"); //вышел за пределы допустимого диапазона значений объекта

if (sz > MAX\_VECTOR\_SIZE)

throw out\_of\_range("Больше максимального размера");

pMem = new T[sz]{};// У типа T д.б. констуктор по умолчанию

}

TDynamicVector(T\* arr, size\_t s) : sz(s) // конструктор с массивом

{

if (sz == 0)

throw out\_of\_range("Vector size should be greater than zero"); //вышел за пределы допустимого диапазона значений объекта

if (sz > MAX\_VECTOR\_SIZE)

throw out\_of\_range("Больше максимального размера");

assert(arr != nullptr && "TDynamicVector ctor requires non-nullptr arg"); //ошибку выводит с abort и с конкретным местом где она находится

pMem = new T[sz];

std::copy(arr, arr + sz, pMem);

}

TDynamicVector(const TDynamicVector& v) //конструктор копирования

{

sz = v.sz;

if (v.pMem == nullptr) {

pMem = nullptr;

}

else {

pMem = new T[sz];

for (int i = 0; i < sz; i++) {

pMem[i] = v.pMem[i];

}

}

}

TDynamicVector(TDynamicVector&& v) noexcept //конструктор перемещения (вместо копирования данных они просто перемещаются из одной копии объекта в другую)

{

sz = v.sz;

pMem = v.pMem;

v.sz = 0;

v.pMem = nullptr;

}

~TDynamicVector()

{

if (pMem != nullptr) {

sz = 0;

delete[]pMem;

}

}

TDynamicVector& operator=(const TDynamicVector& v)

{

if (this == &v) { // если равен сам себе (что может глупенький сделать, то мы его же и возвращаем)

return \*this;

}

if (pMem != nullptr) { // если в квартире грязно, то ее надо почистить

delete[]pMem;

}

if (v.pMem == nullptr) {

sz = 0;

pMem = nullptr;

}

else {

sz = v.sz;

pMem = new T[sz];

for (int i = 0; i < sz; i++) {

pMem[i] = v.pMem[i];

}

}

return \*this;

}

TDynamicVector& operator=(TDynamicVector&& v) noexcept

{

if (this == &v) { // если равен сам себе (что может глупенький сделать, то мы его же и возвращаем)

return \*this;

}

if (pMem != nullptr) { // если в квартире грязно, то ее надо почистить

delete[]pMem;

}

if (v.pMem == nullptr) {

sz = 0;

pMem = nullptr;

}

else {

sz = v.sz;

pMem = v.pMem;

v.sz = 0;

v.pMem = nullptr;

}

return \*this;

}

size\_t size() const noexcept { return sz; }

// индексация

T& operator[](size\_t ind) //можно потом изменить

{

return pMem[ind];

}

const T& operator[](size\_t ind) const //нельзя потом изменить

{

return pMem[ind];

}

// индексация с контролем

T& at(size\_t ind)

{

if ((ind < 0) || (ind >= sz)) {

throw "Error";

}

if (pMem == nullptr) {

throw "Error";

}

return pMem[ind];

}

const T& at(size\_t ind) const

{

if ((ind < 0) || (ind >= sz)) {

throw "Error";

}

if (pMem == nullptr) {

throw "Error";

}

return pMem[ind];

}

// сравнение

bool operator==(const TDynamicVector& v) const noexcept

{

if (sz != v.sz) {

return 0;

}

for (int i = 0; i < sz; i++) {

if (pMem[i] != v.pMem[i]) {

return 0;

}

}

return 1;

}

bool operator!=(const TDynamicVector& v) const noexcept

{

if (sz != v.sz) {

return 1;

}

for (int i = 0; i < sz; i++) {

if (pMem[i] != v.pMem[i]) {

return 1;

}

}

return 0;

}

// скалярные операции

TDynamicVector operator+(T val)

{

TDynamicVector temp(\*this);

for (int i = 0; i < sz; i++) {

temp[i] = pMem[i] + val;

}

return temp;

}

TDynamicVector operator-(T val)

{

TDynamicVector temp(\*this);

for (int i = 0; i < sz; i++) {

temp[i] = pMem[i] - val;

}

return temp;

}

TDynamicVector operator\*(T val)

{

TDynamicVector<T> temp(\*this);

for (int i = 0; i < sz; i++) {

temp[i] = pMem[i]\*val;

}

return temp;

}

// векторные операции

TDynamicVector operator+(const TDynamicVector& v)

{

if (this->sz != v.sz)

throw exception("Error");

TDynamicVector temp(v);

for (int i = 0; i < sz; i++) {

temp[i] = pMem[i] + v.pMem[i];

}

return temp;

}

TDynamicVector operator-(const TDynamicVector& v)

{

if (this->sz != v.sz)

throw exception("Error");

TDynamicVector temp(v);

for (int i = 0; i < sz; i++) {

temp[i] = pMem[i] - v.pMem[i];

}

return temp;

}

T operator\*(const TDynamicVector& v) //noexcept(noexcept(T()))

{

if (sz != v.sz) {

throw "Error";

}

T temp{};

for (int i = 0; i < sz; i++) {

temp = temp +( pMem[i] \* v.pMem[i]);

}

return temp;

}

friend void swap(TDynamicVector& lhs, TDynamicVector& rhs) noexcept

{

std::swap(lhs.sz, rhs.sz);

std::swap(lhs.pMem, rhs.pMem);

}

// ввод/вывод

friend istream& operator>>(istream& istr, TDynamicVector& v)

{

for (size\_t i = 0; i < v.sz; i++)

istr >> v.pMem[i]; // требуется оператор>> для типа T

return istr;

}

friend ostream& operator<<(ostream& ostr, const TDynamicVector& v)

{

for (size\_t i = 0; i < v.sz; i++)

ostr << v.pMem[i] << ' '; // требуется оператор<< для типа T

return ostr;

}

};

// Динамическая матрица -

// шаблонная матрица на динамической памяти

template<typename T>

class TDynamicMatrix : private TDynamicVector<TDynamicVector<T>>

{

using TDynamicVector<TDynamicVector<T>>::pMem;

using TDynamicVector<TDynamicVector<T>>::sz;

public:

TDynamicMatrix(size\_t s = 1) : TDynamicVector<TDynamicVector<T>>(s)

{

if (sz == 0 || sz > MAX\_MATRIX\_SIZE)

throw out\_of\_range("Размер матрицы больше нуля или больше MAX\_MATRIX\_SIZE");

for (size\_t i = 0; i < sz; i++)

pMem[i] = TDynamicVector<T>(sz);

}

using TDynamicVector<TDynamicVector<T>>::operator[];

// сравнение

bool operator==(const TDynamicMatrix& m) const noexcept

{

if (this->sz != m.sz)

return 0;

for (size\_t i = 0; i < sz; i++)

if (pMem[i] != m.pMem[i])

return 0;

return 1;

}

//неравенство

bool operator!=(const TDynamicMatrix& m) const noexcept

{

if (this->sz != m.sz)

return 1;

for (size\_t i = 0; i < sz; i++)

if (pMem[i] != m.pMem[i])

return 1;

return 0;

}

// матрично-скалярные операции

TDynamicMatrix operator\*(const T& val)

{

TDynamicMatrix res(\*this);

for (size\_t i = 0; i < sz; i++)

res.pMem[i] = res.pMem[i] \* val;

return res;

}

// матрично-векторные операции

TDynamicVector<T> operator\*(const TDynamicVector<T>& v)

{

if (sz != v.sz)

throw exception("Error");

TDynamicVector<T> res(sz);

for (size\_t i = 0; i < sz; i++)

res[i] = pMem[i] \* v;

return res;

}

// матрично-матричные операции

TDynamicMatrix operator+(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw "Разный размер матриц!";

TDynamicMatrix res(\*this);

for (size\_t i = 0; i < sz; i++)

res.pMem[i] = pMem[i] + m.pMem[i];

return res;

}

TDynamicMatrix operator-(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw "Разный размер матриц!";

TDynamicMatrix res(\*this);

for (size\_t i = 0; i < sz; i++)

res.pMem[i] = pMem[i] - m.pMem[i];

return res;

}

TDynamicMatrix operator\*(const TDynamicMatrix& m)

{

if (sz != m.sz)

throw "Разный размер матриц!";

TDynamicMatrix res(\*this);

for (size\_t i = 0; i < sz; i++)

for (size\_t j = 0; j < sz; j++)

for (size\_t k = 0; k < sz; k++)

res.pMem[i][j] += pMem[i][k] \* m.pMem[k][j];

return res;

}

// ввод/вывод

friend istream& operator>>(istream& istr, TDynamicMatrix& v)

{

for (size\_t i = 0; i < v.sz; i++)

for (size\_t j = 0; j < v.sz; j++)

istr >> v[i][j];

return istr;

}

friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v)

{

for (size\_t i = 0; i < v.sz; i++)

{

for (size\_t j = 0; j < v.sz; j++)

{

ostr << v[i][j] << " ";

}

ostr << endl;

}

return ostr;

}

size\_t size()

{

return sz;

}

T& at(int a, int b)

{

if (((a < 0) || (b < 0))||((a>sz)||(b>sz)))

throw "Error";

return pMem[a][b];

}

};

#endif

Sample\_matrix.cpp

// Тестирование матриц

#include <iostream>

#include "tmatrix.h"

//---------------------------------------------------------------------------

//посчитать по времени и сравнить их операции при разном размере.

TDynamicVector<int> rand\_vector(TDynamicVector<int> a, int n) {

for (int i = 0; i < n; i++) {

a[i] = 0 + rand() % 11;

}

return a;

}

TDynamicMatrix<int> rand\_matrix(TDynamicMatrix<int> b, int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++)

{

b[i][j] = 0 + rand() % 12;

}

}

return b;

}

void main()

{

/\*TDynamicMatrix<int> a(5), b(5), c(5);

int i, j;

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

cout << "Тестирование класс работы с матрицами"

<< endl;

for (i = 0; i < 5; i++)

for (j = i; j < 5; j++ )

{

a[i][j] = i \* 10 + j;

b[i][j] = (i \* 10 + j) \* 100;

}

c = a + b;

cout << "Matrix a = " << endl << a << endl;

cout << "Matrix b = " << endl << b << endl;

cout << "Matrix c = a + b" << endl << c << endl;\*/

setlocale(LC\_ALL, "Russian");

int n;

time\_t start, end;

cout << "Тестирование класс работы с матрицами"<< endl;

cout << "Введите размер матрицы: ";

cin >> n;

double x = 0;

TDynamicMatrix<int> a(n), b(n), c(n);

a = rand\_matrix(a, n);

b = rand\_matrix(b, n);

time(&start);

c = a + b;

time(&end);

x = (x + (double)(end - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "O(n^2) - " << x << endl;

x = 0;

TDynamicMatrix<int> aa(n), bb(n), cc(n);

aa = rand\_matrix(a, n);

bb = rand\_matrix(b, n);

time(&start);

cc = aa \* bb;

time(&end);

x = (x + (double)(end - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "O(n^3) - " << x << endl;

cout << "\n";

int k;

cout << "Тестирование класс работы с векторами" << endl;

cout << "Введите размер вектора: ";

cin >> k;

double g = 0;

TDynamicVector<int> arr1(k), arr2(k), arr3(k);

arr1 = rand\_vector(arr1, k);

arr2 = rand\_vector(arr2, k);

time(&start);

arr3 = arr1 + arr2;

time(&end);

g = (g + (double)(end - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "O(n^2) - " << g << endl;

g = 0;

TDynamicVector<int> arr11(k), arr22(k), arr33(k);

arr11 = rand\_vector(arr1, k);

arr22 = rand\_vector(arr2, k);

time(&start);

arr33 = arr11 \* arr22;

time(&end);

g = (g + (double)(end - start) / CLOCKS\_PER\_SEC) \* 1000;

cout << "O(n^3) - " << g << endl;

}

//---------------------------------------------------------------------------

# Эксперименты

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10
2. Процессор: AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
3. Версия Visual Studio: 2022

Матрицы:

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | Время работы оператора сложения (в с)  O(n2) | Время работы оператора умножения (в с)  O(n3) |
| 1000 | 0 | 15 |
| 2 500 | 0 | 240 |
| 5000 | 1 |  |

Были использованы классические методы для сложения и умножения матриц. Их сложности соответственно O(n2) и O(n3). Но на практике это не совсем так. Почему же?

Поскольку мы работаем только с верхнетреугольными матрицами и при этом используем метод хранения данных как вектор векторов, то по факту, мы храним только значения выше главной диагонали, никак не используя нижнюю половину. Тем самым отсекая практически половину значений, а значит экономя время примерно в 2 раза. Храня только заполненную часть матрицы, мы занимаем не памяти, а . За счет как раз этого и уменьшается время работы программы.

Вместо того чтобы хранить матрицу полностью используя память n2, мы храним только необходимую часть матрицы, занимая всего n\*(n+1)/2 памяти. За счет уменьшения объема используемой память и возрастет скорость работы программы, так как самой долгой операцией является операция доступа к данным.

# Заключение

В ходе выполнения этой лабораторной работы, я успешно справилась со всеми поставленными задачами. После изучения различных методов хранения матриц, была выбрана одна из наиболее эффективных форм представления матриц - вектор векторов. Для этого были реализованы классы TDynamicMatrix и TDynamicVector. Глубокое понимание устройства автоматических тестов пришло мне в результате необходимости практически ручного написания их. Также была проведена оценка производительности метода хранения верхнетреугольных матриц, и были сделаны выводы из отличия полученных значений от предполагаемых (в лучшую сторону).

# 7. Литература

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [<http://www.itmm.unn.ru/files/2018/10/Primer-1.2.-Struktury-hraneniya-matrits-spetsialnogo-vida.pdf>], 2015.
2. Википедия: свободная электронная энциклопедия: на русском языке [https://ru.wikipedia.org/wiki/Матрица\_(математика)]

[https://ru.wikipedia.org/wiki/Треугольная\_матрица]