

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«Национальный исследовательский  
Нижегородский государственный университет им. Н.И. Лобачевского»  
(ННГУ)

**Институт информационных технологий, математики и механики**

Направление подготовки: «Программная инженерия»

**ОТЧЕТ**

по лабораторной работе №2

*Структуры хранения матриц специального вида*

**Выполнил:** студент группы  
3822Б1ПР2

\_\_\_\_\_ В. Е. Филатьев  
Подпись

**Проверила:**

\_\_\_\_\_ Я.В. Силенко  
Подпись

Нижний Новгород  
2023

## Содержание

1.ВВЕДЕНИЕ:.....	3
2. ПОСТАНОВКА ЗАДАЧИ:.....	4
3.ОПИСАНИЕ АЛГОРИТМОВ:.....	5
3.1. ОПЕРАЦИИ НАД ВЕКТОРАМИ: .....	5
3.2. ОПЕРАЦИИ НАД МАТРИЦАМИ: .....	5
4.ОПИСАНИЕ ПРОГРАММЫ: .....	6
4.1. КЛАСС ВЕКТОР:.....	6
4.2. КЛАСС МАТРИЦА: .....	7
4.3. КЛАСС ТАЙМЕР: .....	7
5.РЕЗУЛЬТАТЫ: .....	8
6.ЗАКЛЮЧЕНИЕ .....	9
7.ЛИТЕРАТУРА:.....	10
8.ПРИЛОЖЕНИЕ: .....	11

## 1. Введение:

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы. Матрицу можно также представить в виде функции двух дискретных аргументов. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

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.

## **2. Постановка задачи:**

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц и выполнение основных операций над ними:

- Сложение/вычитание;
- Умножение;
- Копирование;
- Сравнение.

Программные средства должны содержать:

- Класс вектор (на шаблонах);
- Класс матрица (на шаблонах);
- Тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними.

### 3.Описание алгоритмов:

#### 3.1. ОПЕРАЦИИ НАД ВЕКТОРАМИ:

Пусть заданы два вектора  $A = (a_1, a_2, \dots, a_n)$  и  $B = (b_1, b_2, \dots, b_n)$ . Рассмотрим следующие основные операции над векторами:

- Сравнение ( $A = B$ ). Вектора считаются равными тогда и только тогда, когда  $a_i = b_i$  при всех  $i = 1..n$ .
- Прибавление скаляра ( $A + a$ ). Результатом сложения вектора  $A$  и скаляра,  $a$  называется вектор  $A' = (a_1 + a, a_2 + a, \dots, a_n + a)$ .
- Вычитание скаляра ( $A - a$ ). Результатом вычитания вектора  $A$  и скаляра,  $a$  называется вектор  $A' = (a_1 - a, a_2 - a, \dots, a_n - a)$ .
- Умножение на скаляр ( $A * a$ ). Результатом умножения вектора  $A$  на скаляр,  $a$  называется вектор  $A' = (a_1 * a, a_2 * a, \dots, a_n * a)$ .
- Вычисление длины. Длиной вектора  $A$  называется скалярная величина

$$|\vec{a}| = \sqrt{\sum_{i=1}^n a_i^2}$$

- Сложение векторов ( $A + B$ ). Результатом сложения векторов  $A$  и  $B$  называется вектор  $C = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$ .
- Вычитание векторов ( $A - B$ ). Результатом вычитания векторов  $A$  и  $B$  называется вектор  $C = (a_1 - b_1, a_2 - b_2, \dots, a_n - b_n)$ .
- Скалярное произведение векторов ( $A * B$ ). Скалярным произведением векторов  $A$  и  $B$  называется скалярная величина  $c = \sqrt{\sum_{i=1}^n a_i \times b_i}$

#### 3.2. ОПЕРАЦИИ НАД МАТРИЦАМИ:

Пусть заданы две матрицы  $A = (a_{ij})$  и  $B = (b_{ij})$  и вектор  $V = (v_j)$ , где  $i = 1..m$ ;  $j = 1..n$ . Рассмотрим следующие основные операции над матрицами:

- Сравнение ( $A = B$ ). Матрицы считаются равными тогда и только тогда, когда  $a_{ij} = b_{ij}$  при всех  $i = 1..m$ ;  $j = 1..n$ .
- Умножение на скаляр ( $A * d$ ). Результатом умножения матрицы  $A$  на скаляр  $d$  называется матрица  $D = (d_{ij})$ , где  $d_{ij} = a_{ij} * d$ , при всех  $i = 1..m$ ;  $j = 1..n$ .
- Умножение на вектор ( $A * V$ ). Результатом умножения матрицы  $A$  на вектор  $V$  называется вектор  $D = (d_i)$ , где  $d_i = \sum_{j=1}^n a_{ij} \times v_j$ , при всех  $i = 1..m$ ;  $j = 1..n$ .
- Сложение матриц ( $A + B$ ). Результатом сложения матриц  $A$  и  $B$  называется матрица  $C = (c_{ij})$ , где  $c_{ij} = a_{ij} + b_{ij}$  при всех  $i = 1..m$ ;  $j = 1..n$ .
- Вычитание матриц ( $A - B$ ). Результатом вычитания матриц  $A$  и  $B$  называется матрица  $C = (c_{ij})$ , где  $c_{ij} = a_{ij} - b_{ij}$  при всех  $i = 1..m$ ;  $j = 1..n$ .
- Умножение матриц ( $A * B$ ). Результатом умножения матриц  $A = (a_{ij})$  и  $B = (b_{jk})$  называется матрица  $C = (c_{ik})$ , где  $c_{ik} = \sum_{j=1}^n a_{ij} * b_{jk}$  при всех  $i = 1..m$ ;  $j = 1..n$ ;  $k = 1..n$ .

## 4.Описание программы:

Программа состоит из двух основных классов и нескольких побочных:

### 4.1. 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 защиты
- const T& operator[](size\_t ind) const – Константный оператор индексации без защиты
- T& at(size\_t ind) Оператор индексирования с защитой
- 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

#### 4.2. 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 защитой
- `const T& at(size_t ind) const` – Константный оператор индексирования с защитой

#### 4.3. КЛАСС ТАЙМЕР:

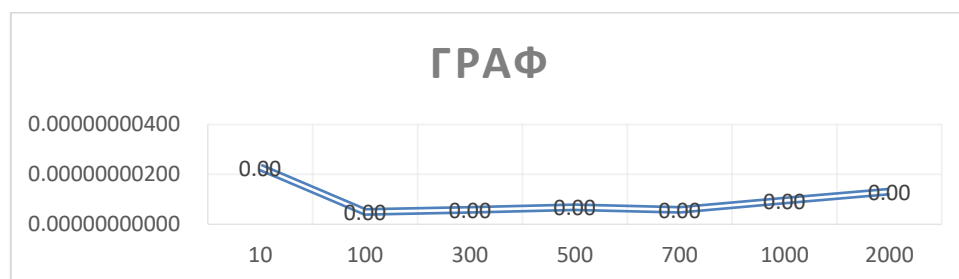
- `Timer()` – Конструктор инициализации таймера
- `void set()` – Метод получения времени работы программы

## 5.Результаты:

Передадим в функцию, которая умножает матрицы сгенерированный случайным образом и проверим результат с предположительной скоростью. Предположительная скорость умножения двух матриц  $O(n^3)$ .

	Время (T(n))	O(n)	T(n) / O(n)
10	$2.28 * 10^{-5}$	1000	$2.28 * 10^{-8}$
100	$4.934 * 10^{-3}$	$10^6$	$4.934 * 10^{-9}$
300	0.1553	$27 * 10^6$	$5.75 * 10^{-9}$
500	0.855	$125 * 10^6$	$6.84 * 10^{-9}$
700	1.99	$346 * 10^6$	$5.75 * 10^{-9}$
1000	9.473	$10^9$	$9.47 * 10^{-9}$
2000	105.417	$8 * 10^9$	$13.1 * 10^{-9}$

Построим график:



Получаем прямую, значит данная сложность совпадает с предположенной скоростью.



## **6. Заключение**

Мы реализовали класс вектор и класс матриц, их основные методы и действия над ним. Реализовали тесты для проверки работоспособности данной программы. Замерили время выполнения программ для различных размеров матриц и проверили с предположенной скоростью выполнения умножения матриц. Проверили работоспособность программы и проверили обработку ошибки при неверном вводе.

## 7. Литература:

Википедия - [https://ru.wikipedia.org/wiki/Вектор\\_\(геометрия\)](https://ru.wikipedia.org/wiki/Вектор_(геометрия))

Википедия - [https://ru.wikipedia.org/wiki/Матрица\\_\(математика\)](https://ru.wikipedia.org/wiki/Матрица_(математика))

Разница между throw и assert - <https://www.programmersought.com/article/54774475923/>

Конструктор перемещения - <https://metanit.com/cpp/tutorial/14.2.php>

Обработка ошибок и исключений - <https://learn.microsoft.com/ru-ru/cpp/cpp/errors-and-exception-handling-modern-cpp?view=msvc-170>

## 8. Приложение:

Класс вектор:

```
template<typename T>
class TDynamicVector
{
protected:
    size_t sz;
    T* pMem;
public:
    TDynamicVector(size_t size = 1) : sz(size)
    {
        pMem = nullptr;
        if (size == 0)
            throw out_of_range("Vector size should be greater than zero");
        if (size > MAX_VECTOR_SIZE)
            throw length_error("The size of the vector must be less than MAX_VECTOR_SIZE");
        pMem = new T[size](); // {}; // У типа T д.б. конструктор по умолчанию
    }
    TDynamicVector(T* arr, size_t s) : sz(s){
        pMem = nullptr;
        if (sz == 0)
            throw out_of_range("Vector size should be greater than zero");
        if (sz > MAX_VECTOR_SIZE)
            throw length_error("The size of the vector must be less than MAX_VECTOR_SIZE")
        assert(arr != nullptr && "TDynamicVector ctor requires non-nullptr arg");
        pMem = new T[sz];
        std::copy(arr, arr + sz, pMem);
    }
    TDynamicVector(const TDynamicVector& v)
    {
        this->sz = v.sz;
        if (v.pMem == nullptr)
            this->pMem = nullptr;
        else {
            this->pMem = new T[sz];
            std::copy(v.pMem, v.pMem + sz, pMem);
        }
    }
    TDynamicVector(TDynamicVector&& v) noexcept
    {
        this->pMem = nullptr;
        swap(*this, v);
    }
    ~TDynamicVector()
    {
        if (pMem != nullptr)
            delete[] pMem;
        sz = 0;
        pMem = nullptr;
    }
    TDynamicVector& operator=(const TDynamicVector& v)
    {
        if (this == &v)
            return *this;
        this->sz = v.sz;
        if (v.pMem == nullptr)
            this->pMem = nullptr;
        else {
            this->pMem = new T[sz];
            std::copy(v.pMem, v.pMem + sz, pMem);
        }
    }
}
```

```

    return *this;
}

TDynamicVector& operator=(TDynamicVector&& v) noexcept
{
    swap(*this, v);
    return *this;
}

size_t size() const noexcept { return sz; }
// индексация
T& operator[](size_t ind)
{
    return pMem[ind];
}
const T& operator[](size_t ind) const
{
    return pMem[ind];
}
// индексация с контролем
T& at(size_t ind)
{
    if (ind < 0 || ind >= sz)
        throw out_of_range("Vector size should be greater than zero");
    return pMem[ind];
}
const T& at(size_t ind) const
{
    if (ind < 0 || ind >= sz)
        throw out_of_range("Vector size should be greater than zero");
    return pMem[ind];
}

// сравнение
bool operator==(const TDynamicVector& v) const noexcept
{
    if (this->sz != v.sz)
        return false;
    for (int i = 0; i < sz; i++)
        if (this->pMem[i] != v.pMem[i])
            return false;
    return true;
}
bool operator!=(const TDynamicVector& v) const noexcept
{
    return !this->operator==(v);
}

// скалярные операции
TDynamicVector operator+(T val)
{
    TDynamicVector temp(this->sz);
    if (pMem != nullptr) {
        for (int i = 0; i < sz; i++) {
            temp.pMem[i] = this->pMem[i] + val;
        }
    }
    return temp;
}
TDynamicVector operator-(T val)
{
    TDynamicVector temp(this->sz);
    if (pMem != nullptr) {

```

```

        for (int i = 0; i < sz; i++) {
            temp.pMem[i] = this->pMem[i] - val;
        }
    }
    return temp;
}
TDynamicVector operator*(T val)
{
    TDynamicVector temp(this->sz);
    if (pMem != nullptr) {
        for (int i = 0; i < sz; i++) {
            temp.pMem[i] = this->pMem[i] * val;
        }
    }
    return temp;
}

// векторные операции
TDynamicVector operator+(const TDynamicVector& v)
{
    if (this->sz != v.sz)
        throw exception("Вектора различного размера");
    TDynamicVector temp(this->sz);
    for (int i = 0; i < this->sz; i++)
        temp.pMem[i] = this->pMem[i] + v.pMem[i];
    return temp;
}
TDynamicVector operator-(const TDynamicVector& v)
{
    if (this->sz != v.sz)
        throw exception("Вектора различного размера");
    TDynamicVector temp(this->sz);
    for (int i = 0; i < this->sz; i++)
        temp.pMem[i] = this->pMem[i] - v.pMem[i];
    return temp;
}
T operator*(const TDynamicVector& v) noexcept(noexcept(T{}))
{
    if (this->sz != v.sz)
        throw exception("Вектора различного размера");
    T temp{};
    for (int i = 0; i < this->sz; i++)
        temp += pMem[i] * v[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;
}
};
```

## Класс матрица:

```
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)
            throw out_of_range("Matrix size should be greater than zero");
        if (sz > MAX_MATRIX_SIZE)
            throw length_error("The size of the matrix must be less than 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 false;
        for (int i = 0; i < m.sz; i++) {
            if (pMem[i] != m.pMem[i])
                return false;
        }
        return true;
    }
    bool operator!=(const TDynamicMatrix& m) const noexcept {
        return !this->operator==(m);
    }
    // матрично-скалярные операции
    TDynamicMatrix operator*(const T& val)
    {
        TDynamicMatrix temp(*this);
        for (int i = 0; i < sz; i++)
            temp.pMem[i] = temp.pMem[i] * val;
        return temp;
    }

    // матрично-векторные операции
    TDynamicVector<T> operator*(const TDynamicVector<T>& v)
    {
        if (this->sz != v.size())
            throw exception("Различные линейные размеры вектора и матрицы");
        TDynamicVector<T> temp(sz);
        for (int i = 0; i < sz; i++) {
            temp[i] = this->pMem[i] * v;
        }
        return temp;
    }

    // матрично-матричные операции
    TDynamicMatrix operator+(const TDynamicMatrix& m)
    {
        if (sz != m.sz)
            throw exception("Матрицы различного размера");
        TDynamicMatrix temp(sz);
        for (int i = 0; i < sz; i++) {
```

```

    temp[i] = pMem[i] + m[i];
}
return temp;
}
TDynamicMatrix operator-(const TDynamicMatrix& m)
{
    if (sz != m.sz)
        throw exception("Матрицы различного размера");
    TDynamicMatrix temp(sz);
    for (int i = 0; i < sz; i++) {
        temp[i] = pMem[i] - m[i];
    }
    return temp;
}
TDynamicMatrix operator*(const TDynamicMatrix& m)
{
    if (sz != m.sz)
        throw exception("Матрицы различного размера");
    TDynamicMatrix temp(sz);
    for (int i = 0; i < sz; i++) {
        for (int j = 0; j < sz; j++) {
            for (int k = 0; k < sz; k++)
                temp[i][j] += this->pMem[i][k] * m.pMem[k][j];
        }
    }
    return temp;
}

// ВВОД/ВЫВОД
friend istream& operator>>(istream& istr, TDynamicMatrix& v)
{
    for (int i = 0; i < v.sz; i++) {
        for (int j = 0; j < v.sz; j++) {
            istr >> v[i][j];
        }
    }
    return istr;
}
friend ostream& operator<<(ostream& ostr, const TDynamicMatrix& v)
{
    for (int i = 0; i < v.sz; i++) {
        for (int j = 0; j < v.sz; j++) {
            ostr << v[i][j] << " ";
        }
        ostr << endl;
    }
    return ostr;
}

size_t size() const noexcept { return sz; }
//Индексация
T& at(size_t ind1, size_t ind2)
{
    if (ind1 < 0 || ind1 >= sz || ind2 < 0 || ind2 >= sz)
        throw out_of_range("Vector size should be greater than zero");
    return pMem[ind1][ind2];
}
const T& at(size_t ind) const
{
    if (ind < 0 || ind >= sz)
        throw out_of_range("Vector size should be greater than zero");
    return pMem[ind][ind2];
}

```



| }

---

Класс таймер:

```
class Timer {  
private:  
    chrono::time_point<chrono::steady_clock> start, end;  
public:  
    Timer() {  
        start = chrono::high_resolution_clock::now();  
    }  
    void set() {  
        end = chrono::high_resolution_clock::now();  
        chrono::duration<float> dur = end - start;  
        cout << "Время: " << dur.count() << " second" << endl;  
    }  
};
```