МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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**

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«МАТРИЦЫ И ВЕКТОРА»**

**Выполнил(а):** студент(ка) группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Шпынов Н.А. /

Подпись

**Проверил:** к.т.н., доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д. /

Подпись

Нижний Новгород  
2023

**Оглавление**

[Введение 1](#_Toc27368)

[1 Постановка задачи 2](#_Toc28596)

[2 Руководство пользователя 3](#_Toc11513)

[2.1 Приложение для демонстрации работы векторов 3](#_Toc803)

[2.2 Приложение для демонстрации работы матриц 4](#_Toc17384)

[3 Руководство программиста 6](#_Toc12349)

[3.1 Использованные алгоритмы 6](#_Toc13821)

[3.2 Описание классов 6](#_Toc115)

[Заключение 13](#_Toc6690)

[Литература 14](#_Toc32083)

[Приложения 15](#_Toc12467)

# Введе**н**ие

Вектор – [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82" \o "Математический объект), характеризующийся величиной и направлением. Поскольку он однозначно задаётся набором своих координат, упорядоченный набор чисел часто тоже называют вектором. В программировании Вектор это динамический массив, который представляет собой последовательность элементов одного типа. В отличие от обычного массива, вектор может изменять свой размер в процессе выполнения программы.

Поскольку вектор представляет собой набор чисел, его можно использовать для реализации матриц, представляющих собой прямоугольные таблицы с заданными в них значениями. Соответственно ее можно представить упорядоченным набором векторов.

Верхнетреугольная матрица – это квадратная матрица, у которой все элементы, расположенные ниже главной диагонали равны нулю (ai,j = 0, i > j). Главная диагональ матрицы может содержать как нулевые, так и ненулевые элементы.

# Постановка задачи

Цель – реализовать представление верхнетреугольных матриц на языке C++ при помощи векторов

Задачи:

1. Разработка представления векторов. Для существования верхнетреугольных матриц на основе векторов, следует создать реализацию их самих.
2. Создать структуру данных вектора.
3. Реализовать необходимые для работы поля и методы, такие как сложение, векторное умножение, индексация и другие.
4. Разработка структуры данных представления матрицы. Определить, каким образом будут храниться вектора для матрицы. Потребуется создать специальную структуру данных, в которой будут содержаться последовательно уменьшающиеся вектора.
5. Определение операций над матрицами. Разработать алгоритмы и функции для выполнения операций над матрицами, таких как сложение, вычитание, умножение и другие.
6. Тестирование и отладка. Как и при любой разработке программного обеспечения, в конце необходимо провести тестирование и отладку кода. Следует обнаружить и исправить возможные ошибки и проблемы, а также убедиться в правильности функционирования.

# Руководство пользователя

## Приложение для демонстрации работы векторов

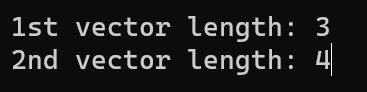
1. Запустить vector\_sample.exe. В результате появится следующее окно (рис. 1):



1. Основное окно приложения

На этом шаге потребуется ввести два числа соответственно: длина первого вектора, длина второго вектора

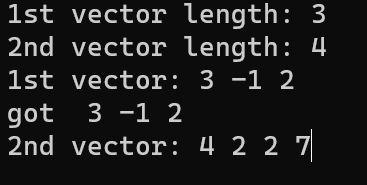
1. После ввода длин полей, следует их заполнить (рис. 2):



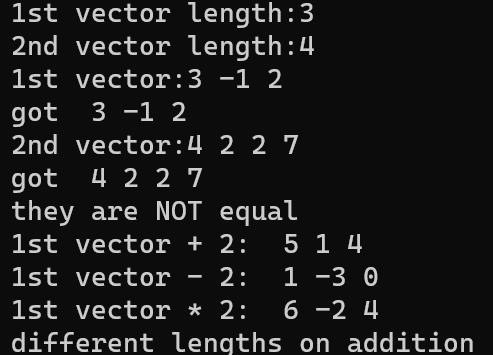
1. Ввод длин

Вводить значения надо по порядку через пробел или Enter, сначала для первого вектора, а затем для второго.

1. Для проверки правильности ввода программа выводит полученные ею поля (рис. 3):



1. Результат ввода полей векторов
2. После ввода второго вектора программа покажет примеры результатов операций над полями. После чего закроется (рис. 4).



1. Итог

## Приложение для демонстрации работы матриц

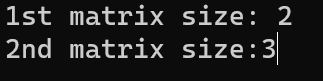
1. Запустить matrix\_sample.exe. В результате появится следующее окно   
   (рис. 5):



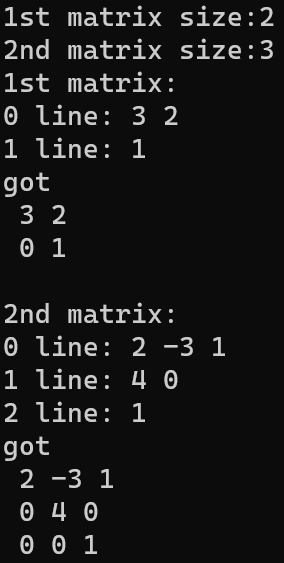
1. Основное окно приложения

Требуется ввести размеры обоих матриц по очереди. По одному значению, поскольку верхнетреугольная матрица всегда квадратная, ее длина и высота равны

1. После ввода длин матриц (рис. 6):



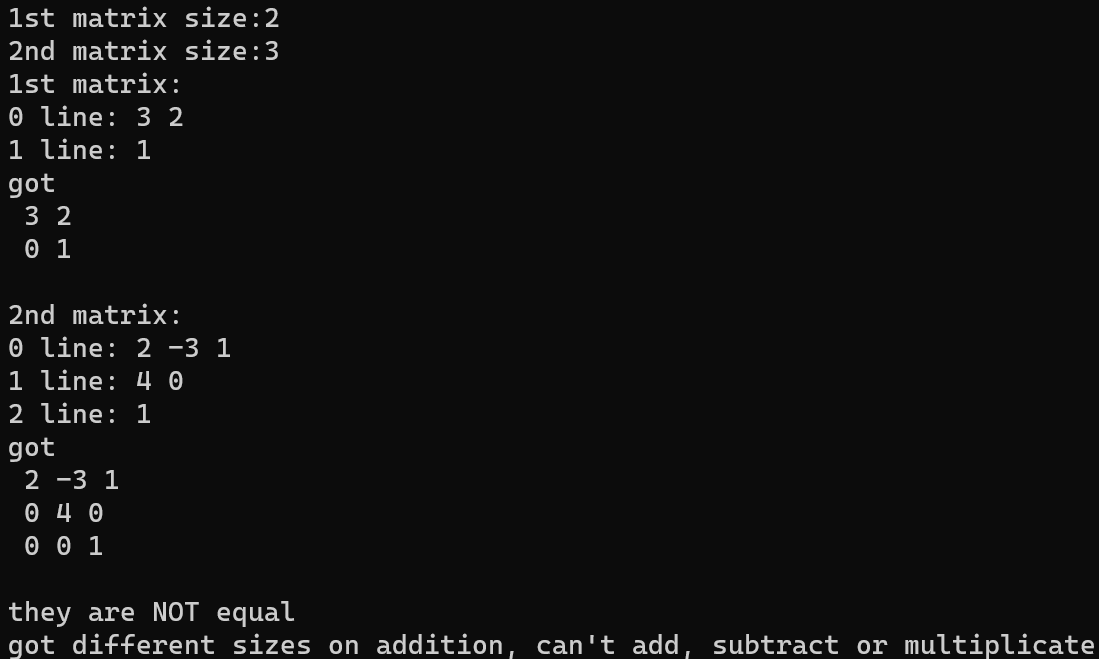
1. Ввод длин
2. Далее потребуется ввести матрицы соответственно (рис. 7):



1. результат ввода матриц

Сначала вводятся поля первой матрицы построчно (все значения первой строки, затем все для второй и так далее). Вводить нули под главной диагональю не надо (то есть каждая следующая строка требует на одно значение меньше)

1. Программа покажет примеры результатов операций над множествами. После чего завершится (рис. 8).



1. Итог

# Руководство программиста

## Использованные алгоритмы

### Вектора

Вектор в программировании — это динамический массив, который представляет собой последовательность элементов одного типа. В отличие от обычного массива, вектор может изменять свой размер в процессе выполнения программы.

### Матрица

Матрица — [математический объект](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82" \o "Математический объект), записываемый в виде прямоугольной таблицы, который представляет собой совокупность [строк](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1" \o "Строка матрицы (страница отсутствует)) и [столбцов](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%BE%D0%BB%D0%B1%D0%B5%D1%86_%D0%BC%D0%B0%D1%82%D1%80%D0%B8%D1%86%D1%8B&action=edit&redlink=1" \o "Столбец матрицы (страница отсутствует)), на пересечении которых находятся его элементы.

Для реализации матриц удобно использовать упорядоченный набор векторов, то есть вектор из векторов. Это решение легко реализовать, так как многие операции, необходимые для матриц, уже определены для векторов.

### Верхнетреугольная матрица

Верхнетреугольная матрица – это квадратная матрица, у которой все элементы, расположенные ниже главной диагонали равны нулю (ai,j = 0, i > j). Главная диагональ матрицы может содержать как нулевые, так и ненулевые элементы.

При реализации таких матриц удобно использовать вектор из векторов, такой что каждый подвектор описывает строку этой матрицы, причём каждый последующий короче предыдущего на один элемент, т.к. хранить нули ниже главной диагонали смысла нет.

## Описание классов

### Класс TBitField

Объявление класса:

template <typename T> class TVector {

protected:

    int size;

    T\* elements;

    int start\_index;

public:

    TVector(int s = 10, int index = 0);

    TVector(const TVector<T>& v);

    ~TVector();

    TVector operator+(const TVector<T>& v);

    TVector operator+(const T& e);

    TVector operator-(const TVector<T>& v);

    TVector operator-(const T& e);

    T operator\*(const TVector<T>& v);

    TVector operator\*(const T& e);

    bool operator==(const TVector<T>& v) const;

    bool operator!=(const TVector<T>& v) const;

    const TVector& operator=(const TVector<T>& v);

    T& operator[](const int ind);

template<typename T> friend ostream& operator<<(ostream& out, const TVector<T>& v);

template<typename T> friend istream& operator>>(istream& in, TVector<T>& v);

};

template<typename T> ostream& operator<<(ostream& out, const TVector<T>& v) {

    for (int i = 0; i < v.size; i++) {

        out << " " << v.elements[i];

    }

    return out;

}

template<typename T> istream& operator>>(istream& in, TVector<T>& v) {

    for (int i = 0; i < v.size; i++) {

        in >> v[i];

    }

    return in;

}

*Поля:*

size – размер вектора.

elements – массив элементов вектора, того типа данных, которым задан вектор.

start\_index – индекс, с которого начинается вектор. Нужен для верхнетреугольных матриц, показывает, с какого столбца начинается вектор

*Конструкторы:*

TVector(int s = 10, int index = 0) – конструктор с параметром.

Назначение: создание вектором с заданной длиной и индексом, 10 и 0 по умолчанию

Входные данные: s – длина вектора, index – индекс.

TVector(const TVector<T>& v) – конструктор копирования.

Назначение: создание вектора из другого.

Входные данные: **v** – заданный вектор.

*Деструктор*:

~TVector();

*Методы:*

нет

*Операторы:*

bool operator==(const TVector<T>& v) const

Назначение: оператор сравнения.

Сравнить на равенство 2 вектора.

Входные данные: v – вектор, с которым идёт сравнение.

Выходные данные: true если равны, false если нет.

bool operator!=(const TVector<T>& v) const

Назначение: оператор сравнения на неравенство.

Сравнить на неравенство 2 вектора.

Входные данные: v – вектор, с которым идёт сравнение.

Выходные данные: false если равны, true если нет.

сonst TBitField& operator=(const TVector<T>& v)

Назначение: оператор присваивания. Присвоить вектору this вектор v.

Входные данные: v – вектор, который присваивается.

Выходные параметры: ссылка на вектор this равный v.

TVector operator+(const TVector<T>& v)

Назначение: оператор сложения. Прибавить к вектору this вектор v.

Входные данные: v – вектор, который прибавляется.

Каждый элемент складывается с соответствующим ему из другого вектора, результат записывается в итоговый вектор.

Выходные параметры: ссылка на вектор this равный this + v.

TVector operator+(const T& e)

Назначение: оператор сложения. Прибавить к вектору this элемент e.

Каждый элемент складывается с этим элементом, результат записывается в итоговый вектор.

Входные данные: v – элемент, который прибавляется.

Выходные параметры: ссылка на вектор this равный this + e.

TVector operator-(const TVector<T>& v)

Назначение: оператор вычитания. Вычесть из вектора this вектор v.

Из каждого элемента вычитается соответствующий ему из другого вектора, результат записывается в итоговый вектор.

Входные данные: v – вектор, который вычитается.

Выходные параметры: ссылка на вектор this равный this - v.

TVector operator-(const T& e)

Назначение: оператор вычитания. Вычесть из вектора this элемент e.

Из каждого элемента вычитается заданный элемент, результат записывается в итоговый вектор.

Входные данные: e – элемент, который вычитается.

Выходные параметры: ссылка на вектор this равный this - e.

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

Назначение: оператор умножения. Умножить вектор this на вектор v.

Входные данные: v – вектор, на который умножается.

Каждый элемент умножается на соответствующий из заданного вектора, результатом является сумма этих произведений.

Выходные параметры: ссылка на вектор this равный скалярному произведению этих векторов.

TVector operator\*(const T& e)

Назначение: оператор умножения. Умножить вектор this на элемент e.

Входные данные: e – элемент, на который умножается.

Каждый элемент умножается на заданный элемент, результат записывается в итоговый вектор.

Выходные параметры: ссылка на вектор this равный this \* e.

T& operator[](const int ind)

Назначение: индексация.

Входные данные: ind – индекс искомого элемента.

Выходные параметры: объект типа данных T под этим индексом.

friend istream &operator>>(istream &istr, TVector<T>& v)

Назначение: оператор **ввода** из терминала.

Входные данные: istr – поток ввода, bf – вектор.

Выходные данные: нет.

friend ostream &operator<<(ostream &ostr, const TVector<T>& v)

Назначение: оператор **вывода** в терминал.

Входные данные: ostr – поток вывода, v – вектор.

Выходные данные: переданный вектор выводится в терминал.

### Класс TMatrix

template<typename T> class TMatrix : public TVector <TVector<T>> {

public:

    TMatrix(int n);

    TMatrix(const TMatrix& m);

    TMatrix(const TVector<TVector<T>>& v);

    const TMatrix& operator=(const TMatrix& m);

    bool operator ==(const TMatrix& m) const;

    bool operator !=(const TMatrix& m) const;

    TMatrix operator+(const TMatrix& m);

    TMatrix operator-(const TMatrix& m);

    TMatrix operator\*(const TMatrix& m);

    template<typename T> friend ostream& operator<<(ostream& out, const TMatrix<T>& m);

    template<typename T> friend istream& operator>>(istream& in, TMatrix<T>& m);

};

template<typename T> ostream& operator<<(ostream& out, const TMatrix<T>& m)

{

    int tmp = 0;

    for (int i = 0; i < m.size; i++) {

        for (int j = 0; j < tmp; j++) {

            out << " 0";

        }

        tmp++;

        out << m.elements[i] << endl;

    }

    return out;

}

template<typename T> istream& operator>>(istream& in, TMatrix<T>& m)

{

    cout << endl;

    for (int i = 0; i < m.size; i++) {

        cout <<i << " line: ";

        in >> m.elements[i];

    }

    return in;

}

*Поля*:

Своих нет

*Конструкторы*:

TMatrix(int n) – конструктор с параметром.

Назначение: создание матрицы с заданном размером.

Входные данные: n – размер матрицы.

TMatrix(const TMatrix& m) – конструктор копирования.

Назначение: создание матрицы из другой.

Входные данные: m – заданная матрица.

TMatrix(const TVector<TVector<T>>& v) – конструктор присваивания.

Назначение: создание матрицы из вектора векторов.

Входные данные: v – заданный вектор.

*Методы*:

Нет

*Операторы:*

bool operator ==(const TMatrix& m) const

Назначение: оператор **сравнения**.

Сравнить на равенство 2 матрицы.

Входные данные: m – матрица, с которой идёт сравнение.

Выходные данные: 1 если равны, 0 если нет.

bool operator !=(const TMatrix& m) const

Назначение: оператор **сравнения на неравенство**.

Сравнить на неравенство 2 матрицы.

Входные данные: m – матрица, с которой идёт сравнение.

Выходные данные: 0 если равны, 1 если нет.

const TMatrix& operator=(const TMatrix& m)

Назначение: оператор **присваивания**.

Входные данные: m – матрица, которая присваивается.

Выходные параметры: ссылка на матрицу this, равную m.

TMatrix operator\*(const TMatrix& m)

Назначение: оператор **умножения**.

Входные данные: m – матрица, с которой проводится операция.

Умножение происходит по правилам умножения матриц в математике.

Каждый элемент матрицы (для стандартной матрицы это элемент аi+ind, j, где i – номер элемента соответствующего вектора, ind – его индекс, j – его номер в векторе векторов) является суммой таких ее элементов, что , где аi, j – текущий элемент матрицы, i – номер элемента соответствующего вектора, j – его номер в векторе векторов.

Выходные данные: матрица равная произведению данных.

TMatrix operator-(const TMatrix& m)

Назначение: оператор **вычитания**.

Входные данные: m – матрица, с которой проводится операция.

Из каждого элемента матрицы (вектор) вычитается соответствующий ему вектор из заданной, результат записывается в итоговую матрицу.

Выходные данные: матрица равная разности данных.

TMatrix operator+(const TMatrix& m)

Назначение: оператор **сложения**.

Входные данные: m – матрица, с которой проводится операция.

К каждому элементу матрицы (вектор) прибавляется соответствующий ему вектор из заданной, результат записывается в итоговую матрицу.

Выходные данные: матрица равная сумме данных.

friend istream &operator>>(istream &istr, const TMatrix& m)

Назначение: оператор **ввода** из терминала.

Входные данные: istr – поток ввода, m – матрица.

Выходные данные: нет.

friend ostream &operator<<(ostream &ostr, const TMatrix& m)

Назначение: оператор **вывода** в терминал.

Входные данные: ostr – поток вывода, m – матрица.

Выходные данные: переданная матрица выводится в терминал.

# Заключение

Реализация верхнетреугольных матриц при помощи векторов оказалась эффективным подходом, позволяющим удобно хранить и оперировать такими матрицами. Этот метод легко создать на основе векторов, большинство функционала класса которых было использовано и в матрицах.

# Литература

1. Вектора в математике [<https://en.wikipedia.org/wiki/Vector>].
2. Матрицы в математике [<https://en.wikipedia.org/wiki/Matrix_(mathematics)>]
3. Справочник по С++ [[https://prog-cpp.ru/cpp](https://prog-cpp.ru/cpp/)].
4. Справочник по С++ [[https://en.cppreference.com/w](https://en.cppreference.com/w/)].

# Приложения

### Реализация полей класса TVector:

template<typename T> TVector<T>::TVector(int s, int index) {

    if (s <= 0)

        throw "got negative size";

    size = s;

    start\_index = index;

    elements = new T[size];

}

template<typename T> TVector<T>::TVector(const TVector<T>& v) {

    size = v.size;

    start\_index = v.start\_index;

    elements = new T[size];

    for (int i = 0; i < size; i++)

        elements[i] = v.elements[i];

}

template<typename T> TVector<T>::~TVector()

{

    delete[] elements;

}

template<typename T> TVector<T> TVector<T>::operator+(const TVector<T>& v) {

    if (size != v.size)

        throw "different lengths on addition";

    else if (start\_index != v.start\_index)

        throw "different indexes on addition";

    TVector vectTmp(size, start\_index);

        for (int i = 0; i < size; i++)

            vectTmp[i] = elements[i] + v.elements[i];

        return vectTmp;

}

template<typename T> TVector<T> TVector<T>::operator+(const T& e) {

    TVector vectTmp(size, start\_index);

        for (int i = 0; i < size; i++)

            vectTmp[i] = elements[i] + e;

        return vectTmp;

}

template<typename T> TVector<T> TVector<T>::operator-(const TVector<T>& v) {

    if (size != v.size)

        throw "different lengths on subtraction";

    else if (start\_index != v.start\_index)

        throw "different indexes on subtraction";

    TVector vectTmp(size, start\_index);

        for (int i = 0; i < size; i++)

            vectTmp[i] = elements[i] - v.elements[i];

    return vectTmp;

}

template<typename T> TVector<T> TVector<T>::operator-(const T& e) {

    TVector vectTmp(size, start\_index);

        for (int i = 0; i < size; i++)

            vectTmp[i] = elements[i] - e;

        return vectTmp;

}

template<typename T> T TVector<T>::operator\*(const TVector<T>& v) {

    if (size != v.size)

        throw "different lengths on multiplication";

    else if (start\_index != v.start\_index)

        throw "different indexes on multiplication";

    int tmp = 0;

    for (int i = 0; i < size; i++)

        tmp += elements[i] \* v.elements[i];

    return tmp;

}

template<typename T> TVector<T> TVector<T>::operator\*(const T& e) {

    TVector vectTmp(size, start\_index);

        for (int i = 0; i < size; i++)

            vectTmp[i] = elements[i] \* e;

        return vectTmp;

}

template<typename T> bool TVector<T>::operator==(const TVector<T>& v) const {

    if ((start\_index != v.start\_index) || (size != v.size))

        return false;

    for (int i = 0; i < size; i++) {

        if (elements[i] != v.elements[i])

            return false;

    }

    return true;

}

template<typename T> bool TVector<T>::operator!=(const TVector<T>& v) const {

    return (!(\*this == v));

}

template<typename T> const TVector<T>& TVector<T>::operator=(const TVector<T>& v) {

    if (this == &v)

        return \*this;

    if (size != v.size) {

        delete[] elements;

        size = v.size;

        elements = new T[size];

    }

    start\_index = v.start\_index;

    for (int i = 0; i < size; i++)

        elements[i] = v.elements[i];

    return \*this;

}

template<typename T>  T& TVector<T>::operator[](int ind) {

    if ((ind > size) || (ind < 0))

        throw "out of range";

    return elements[ind];

}

### Реализация полей класса TMatrix:

template  <typename T> TMatrix<T>::TMatrix(int n) :TVector <TVector<T>>(n) {

    for (int i = 0; i < n; i++) {

        elements[i] = TVector<T>(n - i, i);

    }

}

template  <typename T> TMatrix<T>::TMatrix(const TMatrix& m) : TVector<TVector<T>>(m) {}

template  <typename T> TMatrix<T>::TMatrix(const TVector<TVector<T>>& v) : TVector<TVector<T>>(v) {}

template  <typename T> TMatrix<T> TMatrix<T>::operator+(const TMatrix<T>& m) {

    if (size != m.size)

        throw "got different sizes on addition, can't add, subtract or multiplicate";

    return TVector<TVector<T>>::operator+(m);

}

template  <typename T> TMatrix<T> TMatrix<T>::operator-(const TMatrix<T>& m) {

    if (size != m.size)

        throw "got different sizes on subtractoin, can't add, subtract or multiplicate";

    return TVector<TVector<T>>::operator-(m);

}

template  <typename T> TMatrix<T> TMatrix<T>::operator\*(const TMatrix<T>& m) {

    if (size != m.size)

        throw "got different sizes on multiplication, can't add, subtract or multiplicate";

    TMatrix<T> res(size);

    for (int i = 0; i < size; i++)

        for (int j = i; j < size; j++)

        {

            T sum = 0;

            for (int g = i; g <= j; g++)

                sum += elements[i][g - i] \* m.elements[g][j - g];

            res.elements[i][j - i] = sum;

        }

    return res;

}

template  <typename T> bool TMatrix<T>::operator==(const TMatrix& m) const {

    if (m.size != size)

        return false;

    return TVector<TVector<T>>::operator==(m);

}

template  <typename T> bool TMatrix<T>::operator!=(const TMatrix& m) const {

    return (!(\*this == m));

}

template  <typename T> const TMatrix<T>& TMatrix<T>::operator=(const TMatrix<T>& m)

{

    return  TVector<TVector<T>>::operator=(m);

}