Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Вычисление арифметических выражений»**

**Выполнил**:

студент группы 382003-1

Митин Р. А.

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Требовалось реализовать улучшенный метод Гаусса для матрицы от наследованной от шаблонного вектора векторов. Так же нужно было сделать и сам шаблонный вектор, со всеми необходимыми функциями и операторами.

# Метод решения

Решение СЛАУ используя метод Гаусса:

Метод Гаусса должен принимает на вход вектор свободных коэффициентов (далее вектор b), и с помощью элементарных преобразований над матрицей и вектором, возвращает в ответе вектор решений СЛАУ. Ведущий элемент на каждом шаге алгоритма выбирается как наибольший в столбце, элементы которого находятся не выше главной диагонали. После определения ведущего элемента, вся строка, содержащая его, перемешается так, чтобы элемент оказался на главной диагонали.

Прямой ход:

На каждом шаге алгоритм методом, описанным выше, выбирает ведущий элемент, после чего проверяет систему на линейную независимость (проверяет равен ли ведущий элемент нулю), если система линейно зависима, то алгоритм возвращает вектор нулевой длины, если нет, то строка, содержащая ведущий элемент, и значение в векторе b, соответствующее ему, делятся на ведущий элемент. Следующим шагом алгоритм обнуляет все элементы ведущего столбца, находящиеся ниже диагонали, путём вычитания из соответствующих строк ведущей строки, умноженной на необходимый коэффициент. Так происходит до тех пор, пока алгоритм не обработает все столбцы. После этого матрица имеет верхне-треугольный вид и алгоритм приступает к следующему шагу.

Обратный ход:

На i-том шаге данного этапа вычисляется значение i-того икса, причём вычисление начинается с последнего и продолжается по порядку до первого икса. Вычисление происходит по формуле: , где xi – значение соответствующего икса, а aij  - элемент матрицы стоящий на пересечении i – строки и j – столбца.

# Руководство пользователя

При запуске программа выводит меню, в котором написано, что нужно ввести в консоль, чтобы получить необходимый результат, если ввести символ, который не указан в меню, то программа попросит ввести символ снова.

# Описание программной реализации

Программа содержит 5 файлов, 1 Заголовочный: interface.h и 4 файла с исходным кодом: Matrix.cpp, Vector.cpp, interface.cpp, Main.cpp.

Vector.cpp:

Файл содержит реализацию шаблонного вектора со всеми необходимыми методами и операторами, так же в начале файла указана константа MIN=10-9, которая отвечает за погрешность вычислений, если два числа отличаются друг от друга менее чем на MIN то они считаются равными.

Matrix.cpp:

Файл содержит реализацию шаблонной матрицы, от наследованной от вектора векторов, со всеми необходимыми методами и операторами.

interface.cpp:

Файл содержит функции необходимые для взаимодействия с пользователем

interface.h:

Содержит объявление функций, лежащих в interface.cpp

Main.cpp:

Содержит функцию main.

# Подтверждение корректности

Для подтверждения корректности в программе используется специальный алгоритм проверки. Он генерирует случайную матрицу и вектор, после чего перемножает их и вызывает метод Гаусса для матрицы и результата произведения, если результат метода Гаусса совпадает с исходным вектором, то метод Гаусса реализован верно и погрешность вычислений приемлема, если результаты не совпали значит погрешность больше необходимой и программа выводит её на экран.

# Результаты экспериментов

|  |  |  |  |
| --- | --- | --- | --- |
| Номер эксперимента | Исходные матрицы: | Решение вручную | Программный расчёт: |
| 1: |  | X1= 2.05643  X2= 1.99799  X3= -1.93543 |  |
| 2: |  | X1= 0.148153  X2= -1.5847  X3= -1.36697  X4=7.58173 |  |
| 3: |  | X1= -8.3512  X2= -2.48037  X3= 2.93944 |  |
| 4: |  | X1= -4.30841  X2= -8.06077  X3= 0.64231 |  |

По данным экспериментов видно, что программа вычисляет ответ верно.

# Заключение

Реализован улучшенный метод Гаусса для матрицы от наследованной от шаблонного вектора векторов. Так же сделан и сам шаблонный вектор, со всеми необходимыми функциями и операторами. Реализован минимальный интерфейс для взаимодействия с пользователем.

# Приложение

Vector.cpp:

const double MIN = 1e-9;

template<typename T>

class vector

{

protected:

T\* data;

int size;

public:

vector(int nsize = 0, T\* ndata = NULL)

:size(nsize)

{

if (size < 0)

{

cout << "Size is negative\n";

exit(1);

}

data = new T[size];

if (ndata != NULL)

{

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

{

data[i] = ndata[i];

}

}

}

vector(const vector& a)

:size(a.size)

{

data = new T[size];

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

{

data[i] = a[i];

}

}

int getsize() const { return size; }

void resize(int newsize)

{

if (size != newsize)

{

delete[] data;

data = new T[newsize];

size = newsize;

}

}

void swap(int a, int b)

{

if (a != b)

{

T tmp = this->data[a];

this->data[a] = this->data[b];

this->data[b] = tmp;

}

}

void fillrandom()

{

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

{

this->data[i] = (T)(rand() % 20001) / 20.0 - 500;

}

}

vector<T>& operator=(const vector& second)

{

if (size != second.size)

{

size = second.size;

delete[] data;

data = new T[size];

}

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

{

this[0][i] = second[i];

}

return \*this;

}

bool operator!=(const vector& second)

{

if (size != second.size) { return true; }

if (this == &second) { return false; }

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

{

if (abs(data[i] - second.data[i])>MIN)

return true;

}

return false;

}

friend vector operator\*(const vector<T>& a, T b)

{

vector<T> result(a.size);

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

{

result[i] = a[i] \* b;

}

return result;

}

friend vector operator\*(T b,const vector<T>& a)

{

vector<T> result(a.size);

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

{

result[i] = a[i] \* b;

}

return result;

}

vector<T>& operator/=(const T a)

{

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

{

data[i] /= a;

}

return \*this;

}

vector<T>& operator+=(const vector<T> a)

{

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

{

data[i] += a[i];

}

return \*this;

}

T& operator[](int i)

{

if (i < 0 || i >= size)

{

cout << "Access error in vector";

exit(2);

}

return data[i];

}

const T& operator[](int i) const

{

if (i < 0 || i >= size)

{

cout << "Access error in vector";

exit(2);

}

return data[i];

}

friend ostream& operator<<(ostream& str, const vector<T>& vec)

{

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

{

str << vec[i] << ' ';

}

str << '\n';

return str;

}

friend istream& operator>>(istream& str, vector<T>& vec)

{

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

{

str >> vec[i];

}

return str;

}

template<typename T>

friend double inaccuracy(vector<T> a, vector<T> b);

~vector()

{

delete[] data;

}

};

template<typename T>

double inaccuracy(vector<T> a,vector<T> b)

{

double result = 100;

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

{

if (abs(b[i] - a[i]) >= MIN && abs(b[i] - a[i]) < result)

{

result = abs(b[i] - a[i]);

}

}

return result;

}

Matrix.cpp:

template<typename T>

class Matrix :public vector<vector<T>>

{

private:

void swaprow(int i, int j)

{

if (i != j)

{

vector<T> tmp = this->data[i];

this->data[i] = this->data[j];

this->data[j] = tmp;

}

}

int maxincol(int col)

{

T max = this->data[col][col];

int num = col;

for (int i = col + 1; i < this->size; i++)

{

if (abs(max) < abs(this->data[i][col]))

{

max = this->data[i][col];

num = i;

}

}

return num;

}

public:

Matrix(int nsize)

:vector<vector<T> >(nsize)

{

for (int i = 0; i < this->size; i++)

{

this->data[i].resize(this->size);

}

}

vector<T> operator\*(const vector<T>& b)

{

if (this->size != b.getsize())

{

cout << "Size is different\n";

exit(0);

}

vector<T> result(this->size);

for (int i = 0; i < this->size; i++)

{

result[i] = 0;

for (int j = 0; j < this->size; j++)

{

result[i] += this->data[i][j] \* b[j];

}

}

return result;

}

vector<T> operator()(const vector<T>& a)

{

vector<T> b = a;

for (int i = 0; i < this->size; i++)

{

int imax = maxincol(i);

swaprow(imax, i);

b.swap(imax, i);

if (abs(this->data[i][i]) < MIN)

{

cout << "System is linearly dependent\n\n";

return vector<T>(0);

}

b[i] /= this->data[i][i];

this->data[i] /= this->data[i][i];

for (int j = i+1; j < this->size; j++)

{

b[j] += b[i] \* this->data[j][i] \* (-1);

this->data[j] += this->data[i]\*this->data[j][i]\*(-1);

}

}

for (int i = this->size - 1; i >= 0; i--)

{

for (int j = i + 1; j < this->size; j++)

{

b[i] += b[j] \* (this->data[i][j]) \* (-1);

}

if (abs(b[i]) < MIN)

{

b[i] = 0;

}

}

return b;

}

void fillrandom()

{

for (int i = 0; i < this->size; i++)

{

for (int j = 0; j < this->size; j++)

{

this->data[i][j] = (T)(rand() % 20001)/20.0 - 500;

}

}

}

};