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

«Решение систем линейных уравнений методом Гаусса»

Выполнил:
студент группы 382003-1
Ларин К.Д.

Проверил:
ассистент каф. МОСТ,
Волокитин В.Д.

Нижний Новгород
2021

Содержание

Постановка задачи

Реализовать шаблон класса vector, реализовать шаблон класса matrix как наследника от vector<vector>, реализовать метод Гаусса для решения системы линейных уравнений с выбором опорного элемента.

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: $x_i = \sum_{j=i+1}^{j=x_j * a_{ij} * (-1)}$ ❑, где x_i – значение соответствующего икса, а a_{ij} - 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

Файлы: BigInt.h, BigInt.cpp, menu.h, menu.cpp, rational.h, vector.h, matrix.h, linear_equations.h

BigInt: реализация целочисленной длинной арифметики. Для объектов типа BigInt доступны все арифметические операторы применимые к целым числам.

rational.h: реализация рациональных чисел

menu: функции для взаимодействия с пользователем средствами консоли

vector: реализация динамического массива. Предоставляет основную часть функционала std::vector, а так же реализует арифметические операции (сложение, вычитание векторов и умножение, деление на число).

matrix: реализация класса матрицы. Matrix является наследником от vector<vector> и сохраняет все функции доступные для вектора. В классе Matrix переопределены следующие функции:

- size() возвращает std::pair<size_t, size_t> - число строк и столбцов в матрице соответственно.
- resize(size_t, size_t) изменяет сразу 2 размерности.

В классе Matrix реализован метод gauss, который выполняет прямой ход алгоритма Гаусса.

```
template <typename pivotType = basePivot<T>, typename predType = isEqual<T>>
void gauss(pivotType = pivotType(), predType = predType());
```

- pivotType – тип функтора выбирающего опорный элемент в столбце матрицы. Оператор() для pivotType должен принимать объект типа matrix и целое число — номер столбца, возвращаемое значение — номер строки на которой находится опорное значение. По умолчанию выбирается максимальный по модулю элемент в строке, функция модуля имеет следующую реализацию (val < 0 ? -val, val).
- predType – тип функора реализующего сравнение двух элементов. Оператор() должен принимать два значения типа соответствующего типу элемента матрицы, возвращаемое значение true если значения аргументов равны, false в противном случае. По умолчанию для типов float, double, long double выполняется проверка std::abs(left - right) < std::numeric_limits<T>::epsilon() * 10000; для остальных типов используется оператор==.

Для работы функции gauss тип T должен удовлетворять следующим условиям:

- иметь конструктор от константы 0
- иметь операторы `/, *, -=`
- реализовать конструктор копирования или перемещения, реализовать оператор= (копирования или перемещения)
- функция модуля эквивалентна определению модуля действительного числа (для pivotType по умолчанию)
- для пользовательского типа данных результат false для a == b гарантирует, что a – b != 0 (для predType по умолчанию)

linear_equations: solveEquationSystem – функция решающая систему линейных уравнений.

```
template <typename T, typename pivotType = basePivot<T>, typename predType = isEqual<T>>

Matrix<T>  solveEquationSystem(Matrix<T> sys, vector<T>& ans, eqResults& err, pivotType
pivot = pivotType(), predType pred = predType(), bool isTriangle = false)
```

sys – расширенная матрица, соответствующая системе линейных уравнений

ans – решение системы уравнений ans[i] – значение x_{i+1}

err – используется для сообщений о несовместности системы или бесконечном числе решений, принимает одно из значений enum `eqResults` { OK, INCONSISTENT, INF_SOLUTIONS };

pivot – алгоритм выбора опорного элемента в матрицей

pred – проверка на равенство значений

isTriangle – приведена ли матрица к треугольному виду.

Возвращается матрица, переданная в качестве аргумента, после применение для нее функции gauss.

Результаты экспериментов

```
G:\source\repos\Lab2\x64\Release\Lab2.exe
system of linear equations of real  3 x 3

-1.17934*x1 + -8.42746*x2 + -4.18692*x3 = -3.02091
-3.70212*x1 + -5.56313*x2 + -7.99581*x3 = -2.9436
0.0357645*x1 + -4.61831*x2 + -8.66976*x3 = 3.26489

        Main menu
1) create new matrix
2) reset matrix
3) set matrix element
4) run Gauss algorithm
5) solve a system of linear equations
6) print full matrix
7) stop print matrix
8) delete matrix
9) exit
>> 5
solution of the system of equations:
x1 = 1.43178; x2 = 0.465436; x3 = -0.618612
Для продолжения нажмите любую клавишу . . .
```

$$x=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}1.43177\\0.46543\\-0.61861\end{bmatrix}$$

```
G:\source\repos\Lab2\x64\Release\Lab2.exe
system of linear equations of real  3 x 3

1*x1 + 2*x2 + 3*x3 = 4
5*x1 + 6*x2 + 7*x3 = 8
9*x1 + 10*x2 + 11*x3 = 12

        Main menu
1) create new matrix
2) reset matrix
3) set matrix element
4) run Gauss algorithm
5) solve a system of linear equations
6) print full matrix
7) stop print matrix
8) delete matrix
9) exit
>> 5
infinite number of solutions
Для продолжения нажмите любую клавишу . . .
```

$$x_1=-2+1\cdot\lambda_1$$
$$x_2=3-2\cdot\lambda_1$$
$$x_3=\lambda_1.$$

```
G:\source\repos\Lab2\x64\Release\Lab2.exe
system of linear equations of rational  4 x 4

(1 / 1)*x1 + (2 / 1)*x2 + (5 / 1)*x3 + (9 / 1)*x4 = 79 / 1
(3 / 1)*x1 + (13 / 1)*x2 + (18 / 1)*x3 + (30 / 1)*x4 = 263 / 1
(2 / 1)*x1 + (4 / 1)*x2 + (11 / 1)*x3 + (16 / 1)*x4 = 146 / 1
(1 / 1)*x1 + (9 / 1)*x2 + (9 / 1)*x3 + (9 / 1)*x4 = 92 / 1

        Main menu
1) create new matrix
2) reset matrix
3) set matrix element
4) run Gauss algorithm
5) solve a system of linear equations
6) print full matrix
7) stop print matrix
8) delete matrix
9) exit
>> 5
solution of the system of equations:
x1 = 734 / 7; x2 = 53 / 7; x3 = -10 / 1; x4 = 1 / 1
Для продолжения нажмите любую клавишу . . .
```

$$x_1=104\frac{6}{7},\quad x_2=7\frac{4}{7},\quad x_3=-10,\quad x_4=1.$$

По данным экспериментов видно, что программа вычисляет ответ верно.

Заключение

Реализован метод Гаусса решения СЛАУ с выбором опорного элемента.

Приложение

```
template <typename T>
	class vector
	{
	public:
		vector();
		vector(size_t, const T & = T());
		vector(std::initializer_list<T>);
		vector(const vector&);
		vector(vector&&) noexcept;

		~vector();

		vector& operator=(const vector&);
		vector& operator=(vector&&) noexcept;

		T& operator[](size_t);
		const T& operator[](size_t) const;

		template <typename U>
		void push_back(U&&);
		void pop_back();

		void resize(size_t);
		void reserve(size_t);
		void clear();

		size_t size() const;

		friend bool operator== <>(const vector&, const vector&);
		friend bool operator!= <>(const vector&, const vector&);

		friend vector operator+ <>(vector, const vector&);
		friend vector operator- <>(vector, const vector&);
		friend vector operator* <>(const T&, vector);
		friend vector operator* <>(vector, const T&);
		friend vector operator/ <>(vector, const T&);

		vector& operator+=(const vector&);
		vector& operator-=(const vector&);
```

```
		vector& operator*=(const T&);
		vector& operator/=(const T&);

		vector operator-() const;

		T* begin();
		const T* begin() const;
		T* end();
		const T* end() const;

		T& front();
		const T& front() const;
		T& back();
		const T& back() const;

	protected:
		size_t _capacity, _size;
		T* _data;
	};

template <typename T>
	struct isEqual
	{
		bool operator()(const T& left, const T& right)
		{
			if constexpr (is_floating_point_v<T>) {
				return std::abs(left - right) < std::numeric_limits<T>::epsilon()
* 10000;
			}
			else {
				return left == right;
			}
		}
	};

	template <typename T>
	struct basePivot
	{
		size_t operator()(const Matrix<T>& matrix, size_t colum)
		{
			static auto abs = [](const T& val) {
				if (val < 0) return -val;
```

```
                return val;
        };

        size_t maxI = colum;

        for (size_t i = colum + 1; i < matrix.size().first; i++)
        {
                if (abs(matrix[maxI][colum]) < abs(matrix[i][colum])) {
                        maxI = i;
                }
        }

        return maxI;
    }
};

template <typename T>
class Matrix : public vector<vector<T>>
{
public:
    Matrix() = default;
    Matrix(const Matrix&) = default;
    Matrix(Matrix&&) = default;
    Matrix(size_t h, size_t w) : vector<vector<T>>(h, vector<T>(w)) {}
    Matrix(size_t h, size_t w, const T& val) : vector<vector<T>>(h, vector<T>(w,
val)) {}
    Matrix(std::initializer_list<vector<T>> iList) : vector<vector<T>>(iList) {}

    ~Matrix() = default;

    Matrix& operator=(const Matrix&) = default;
    Matrix& operator=(Matrix&&) = default;

    std::pair<size_t, size_t> size() const
    {
        if (this->_size == 0) {
            return { 0, 0 };
        }
        return { this->_size, this->_data[0].size() };
    }

    void resize(size_t h, size_t w)
```

```
	{
		this->vector<vector<T>>::resize(h);
		for (size_t i = 0; i < this->_size; i++)
		{
			this->_data[i].resize(w);
		}
	}

	template <typename pivotType = basePivot<T>, typename predType = isEqual<T>>
	void gauss(pivotType = pivotType(), predType = predType());
};



template <typename T>
template <typename pivotType, typename predType>
void Matrix<T>::gauss(pivotType pivot, predType pred)
{
	auto& matrix = *this;

	for (size_t i = 0; i < matrix.size().first && i < matrix.size().second; i++)
	{
		size_t maxValRow = pivot(*this, i);
		if (maxValRow != i) {
			std::swap(matrix[i], matrix[maxValRow]);
		}

		if (matrix[i][i] != T(0)) {
			for (size_t j = i + 1; j < matrix.size().first; j++)
			{
				bool nullRow = true;

				T tmp = matrix[j][i] / matrix[i][i];
				for (size_t k = i + 1; k < matrix.size().second; k++)
				{
					if (pred(matrix[j][k], tmp * matrix[i][k])) {
						matrix[j][k] = T(0);
					}
					else {
						matrix[j][k] -= tmp * matrix[i][k];
						nullRow = false;
					}
				}
```

```
                                matrix[j][i] = T(0);

                                if (nullRow) {
                                        std::swap(matrix[j], matrix.back());
                                        matrix.pop_back();
                                        j--;
                                }
                        }
                }
        }
```

```
enum equationsResults { OK, INCONSISTENT, INF_SOLUTIONS };

        template <typename T, typename pivotType = basePivot<T>, typename predType =
isEqual<T>>
        Matrix<T> solveEquationSystem(Matrix<T> sys, vector<T>& ans, int& err,
                                                        pivotType pivot = pivotType(),
predType pred = predType(),
                                                        bool isTriangle = false)
        {
                err = equationsResults::OK;

                if (!isTriangle) {
                        sys.gauss(pivot, pred);
                }

                if (sys.size().first + 1 != sys.size().second) {
                        err = equationsResults::INF_SOLUTIONS;

                        if (sys.size().first + 1 > sys.size().second) {
                                bool flag = true;
                                for (size_t i = 0; i + 1 < sys.size().second; i++)
                                {
                                        if (sys.back()[i] != 0) {
                                                flag = false;
                                                break;
                                        }
                                }
```

```
                        if (flag) {
                                err = equationsResults::INCONSISTENT;
                        }
                }
        }
        else {
                ans.resize(sys.size().first);
                T sum;
                for (size_t i = sys.size().first; i > 0;)
                {
                        --i;

                        sum = 0;
                        for (size_t j = i + 1; j < sys.size().first; j++)
                        {
                                sum += sys[i][j] * ans[j];
                        }

                        bool ansIsNull = pred(sys[i].back(), sum);
                        ans[i] = sys[i].back() - sum;
                        if (sys[i][i] == 0) {
                                if (ansIsNull) {
                                        err = equationsResults::INF_SOLUTIONS;
                                }
                                else {
                                        err = equationsResults::INCONSISTENT;
                                }

                                break;
                        }

                        ans[i] /= sys[i][i];
                }
        }

        return sys;
}
```