**Министерство науки и высшего образования РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Уфимский университет науки и технологий»**

**Кафедра** Высокопроизводительных вычислений и дифференциальных уравнений

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 100 |  |  |  |  |  |  |  |  |  |  |
| 90 |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |
| 0 |  |  |  |  |  |  |  |  |  |  |

**Приближение функций**

**Отчет по лабораторной работе №3**

По дисциплине

«Численные методы»

**2342.233212.000 ПЗ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа  МКН-318Б | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Нечаев Б.П. |  |  |  |
| Принял | Феоктистов Б.А |  |  |  |

Уфа 2024

**Содержание**

[Введение 3](#_Toc183367692)

[Теоретическая Часть 4](#_Toc183367693)

[1. Метод Гаусса с выбором главного элемента 4](#_Toc183367694)

[2. Рациональная интерполяция 5](#_Toc183367695)

[3. LU-разложение 5](#_Toc183367696)

[4. Метод прогонки СЛАУ с 5-диагональной матрицей 6](#_Toc183367697)

[Практическая Часть 9](#_Toc183367698)

[1. Задача 1 9](#_Toc183367699)

[2. Задача 2 10](#_Toc183367700)

[3. Задача 4 12](#_Toc183367701)

[Заключение 13](#_Toc183367702)

[Список литературы 14](#_Toc183367703)

[Приложение А 15](#_Toc183367704)

**Введение**

**Цель работы:** получить навык численного решения систем линейных алгебраических уравнений (СЛАУ) с использованием различных прямых методов.

**Теоретическая Часть**

## Метод Гаусса с выбором главного элемента

Наиболее известным из точных методов решения систем линейных уравнений является метод исключения Гаусса. В предположении, что , первое уравнение системы

делим на коэффициент , в результате получаем уравнение

Затем из каждого из остальных уравнений вычитается первое уравнение, умноженное на соответствующий коэффициент . В результате эти уравнения преобразуются к виду

Первое неизвестное оказалось исключенным из всех уравнений, кроме первого. Далее в предположении, что , делим второе уравнение на коэффициент и исключаем неизвестное из всех уравнений, начиная со второго и т.д. В результате последовательного исключения неизвестных система уравнений преобразуется в систему уравнений с треугольной матрицей

Совокупность проведенных вычислений называется прямым ходом метода Гаусса.

Из -го уравнения системы определяем , из -го – и т.д. до . Совокупность таких вычислений называют обратным ходом метода Гаусса.

Чтобы избежать катастрофического влияния вычислительной погрешности, применяют метод Гаусса с выбором главного элемента. Его отличие от описанной выше схемы метода Гаусса состоит в следующем. Пусть по ходу исключения неизвестных получена система уравнений

Найдем такое, что и переобозначим и ; далее произведем исключение неизвестной из всех уравнений, начиная с -го. Такое переобозначение приводит к изменению порядка исключения неизвестных и во многих случаях существенно уменьшает чувствительность решения к погрешностям округления при вычислениях.

## Рациональная интерполяция

При заданных приближение к ищется в виде

Коэффициенты находятся из совокупности соотношений , которые можно записать в виде

Уравнения (1) образуют систему линейных алгебраических уравнений относительно неизвестных.

## LU-разложение

Пусть – данная матрица, а и – соответственно нижняя (левая) и верхняя (правая) треугольные матрицы. Справедливо следующее утверждение.

Теорема. Если все главные миноры квадратной матрицы отличны от нуля, то существуют такие нижняя и верхняя треугольные матрицы, что . Если элементы диагонали одной из матриц или фиксированы (ненулевые), то такое разложение единственно.

Формулы в случае фиксирования диагонали нижней треугольной матрицы :

## Метод прогонки СЛАУ с 5-диагональной матрицей

Метод прогонки является частным случаем метода Гаусса и применяется к системам с трех-пятидиагональной матрицей. Такие системы часто встречаются при численном решении краевых задач для дифференциальных уравнений второго порядка, при моделировании некоторых инженерных задач.

Если при решении использовать метод Гаусса, то расчет можно организовать таким образом, чтобы не включать нулевые элементы матрицы. Этим самым экономится требуемая память и уменьшается объем вычислений.

Рассмотрим СЛАУ следующего вида:

|  |  |
| --- | --- |
|  | (1) |

Для решения системы (1) применим метод исключения Гаусса. Учитывая структуру системы (1), легко получим, что обратный ход метода Гаусса должен осуществляться по формулам:

|  |  |
| --- | --- |
|  | (2) |

Для реализации (2) необходимо задать , а также определить коэффициенты  используя первое уравнение (2), выразим  и . Получим:

|  |  |
| --- | --- |
|  | (3) |

для 

Подставляя (3) в третье уравнение (1), получим



при 

Сравнивая это выражение с первым уравнением (2), видим, что если положить

|  |  |
| --- | --- |
|  | (4) |

где обозначено , то уравнения системы (1) для  будут удовлетворены.

Рекуррентные соотношения (4) связывают  и  с  и . Поэтому, если будут заданы  и  для , то по формулам (4) последовательно можно найти коэффициенты  и  для 

Найдем  и  для . Из первого уравнения (1) и (2) для непосредственно получим

|  |  |
| --- | --- |
|  | (5) |

Далее подставляя значение (3) при  во вторую строчку (3.1), получим

|  |  |
| --- | --- |
|  | (6) |

Следовательно, второе уравнение (2.3.1) будет выполнено, если положить

|  |  |
| --- | --- |
|  | (7) |

Итак, используя (4) – (7) можно найти  и  для  Осталось определить  и , входящие в формулу (2).

Воспользуемся для этого четвертым и пятым уравнениями (1). Подставляя (3) при  в четвертое уравнение, найдем, что  и  определяются по формулам (4) для . Найдем теперь . Для этого подставим  или  где  определяется по формуле (4) при .

Объединяя полученные выше формулы, запишем алгоритм правой прогонки для системы (1) в следующем виде:





находятся прогоночные коэффициенты  и ;

неизвестные  находятся последовательно по формулам



**Практическая Часть**

## Задача 1

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ методом Гаусса с выбором ведущего элемента.
2. С использованием написанной программы решить задачу о рациональной интерполяции: выполнить приближение функции , заданной таблично, рациональной функцией вида

где и – многочлены степени и , соответственно. При этом требуется также определить значения и .

1. Построить график интерполирующей функции и исходных данных.

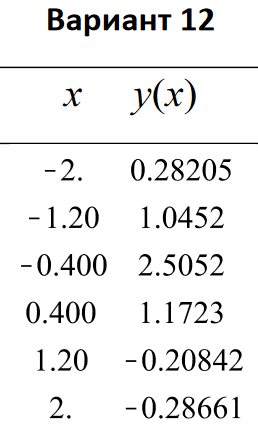


Рисунок 1. Исходные данные

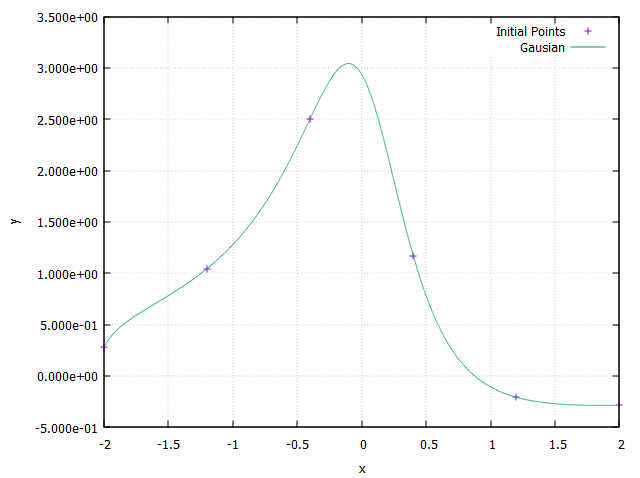
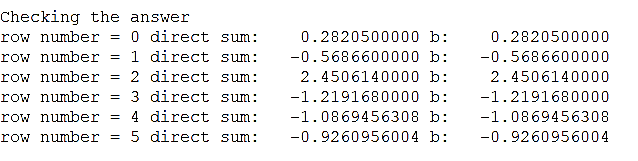
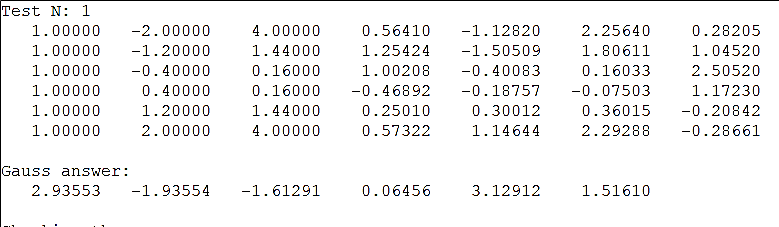


Рисунок 2. Решение СЛАУ методом Гаусса с выбором ведущего элемента

Рисунок 3. Проверка решения СЛАУ подстановкой

Рисунок 4. График интерполирующей функции и данных точек

**Вывод:** В ходе выполнения задания был реализован способ решения СЛАУ методом Гаусса с выбором ведущего элемента и проведена интерполяция таблично заданной функции с использованием рациональной функции.

## Задача 2

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ методом LU-разложения.
2. Выполнить п. 2), 3) Задачи 1

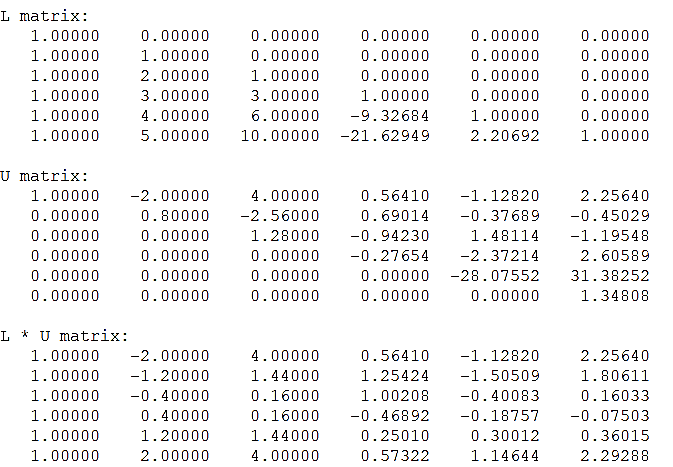


Рисунок 5. Матрицы LU разложения, а так же проверочная матрицы L\*U равная изначальной матрице.

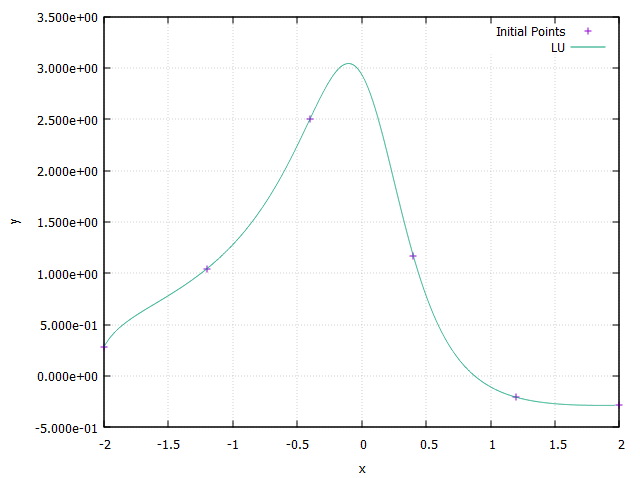
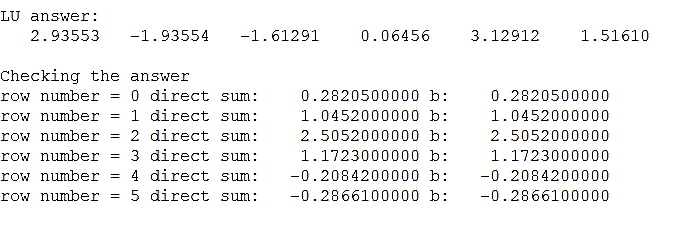


Рисунок 6.Решение СЛАУ и проверка подстановкой

Рисунок 7. График интерполирующей функции и данных точек

**Вывод:** в ходе выполнения задания было реализовано решение СЛАУ с помощью LU-разложения. Построен график интерполирующей функции и исходных данных, который совпадает с графиком в задании №1.

## Задача 4

1. Написать вычислительную программу на языке программирования C++ для решения методом прогонки СЛАУ с 5-диагональной матрицей.
2. Для отладки программы написать генератор случайных вещественных матриц данного вида с диагональным преобладанием

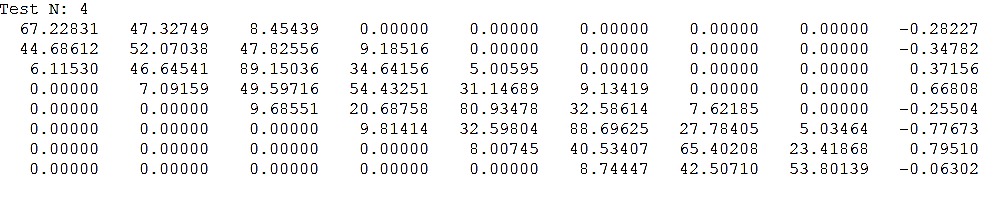


Рисунок 8. 5-диагональная матрица с диагональным преобладанием

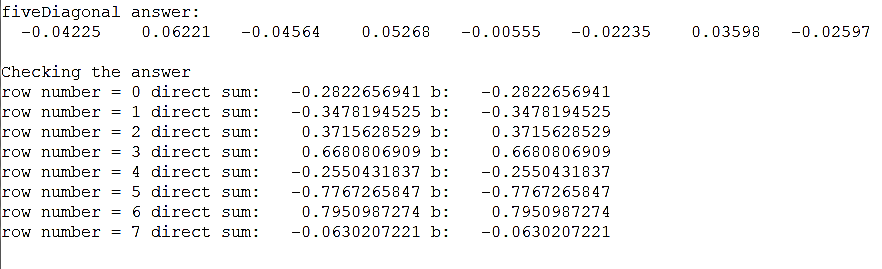


Рисунок 9. Решение 5-диагональной матрицы и проверка

**Вывод:** в ходе выполнения задания была разработана вычислительная программа на языке C++ для решения СЛАУ с 5-диагональной матрицей методом прогонки. Также был реализован генератор случайных вещественных матриц с диагональным преобладанием.

**Заключение**

В результате работы были получены навыки применения различных методов для решения СЛАУ, что полностью соответствует цели работы. Все алгоритмы были проверены и дали правильные результаты.

В ходе работы были реализованы несколько методов: метод Гаусса с выбором ведущего элемента, метод LU-разложения и метод прогонки для 5-диагональных матриц.

Метод Гаусса был использован для решения СЛАУ, для приближения функции использован метод рациональной интерполяции. Построенные графики показали, что программа работает правильно. Метод LU-разложения также успешно решал задачи, давая такие же результаты, как и метод Гаусса.

Для решения СЛАУ с 5-диагональной матрицей был использован метод прогонки, а также был написан генератор случайных матриц с диагональным преобладанием для тестирования программы.

**Список литературы**

1. Бахвалов Н. С., Жидков Н. П., Кобельков Г. М. Численные методы: учебник. — М.: Бином, 2018. — 636 с.
2. Калиткин Н. Н. Численные методы: учебник. — 2-е изд. — СПб.: БХВ-Петербург, 2014. — 592 с.
3. Вержбицкий В.М. Основы численных методов: учебник для студентов высших учебных заведений, обучающихся по направлению подготовки дипломированных специалистов «Прикладная математика». — 5-е изд. — М.; Берлин: Директ-Медиа, 2021. — 400 с.
4. Завьялов Ю.С., Квасов Б.И., Мирошниченко В.Л. Методы сплайнфункций. — М.: МГУ, 2010. — 280 с.

**Приложение А**

**Листинг программы**

**Сonstants.h**

#pragma once

#ifndef MATH\_CONSTANTS\_H

#define MATH\_CONSTANTS\_H

const double PI = 3.141592653589793;

const int nodes = 10000;

const double delta0 = 5 \* 1e-5;

#endif

**FiveDiagonals.h**

#pragma once

#include "RecordPoints.h"

inline double RandomDouble(double min, double max) {

return min + (max - min) \* (*rand*() / static\_cast<double>(*RAND\_MAX*));

}

inline int RandomInt(int min, int max) {

return min + *rand*() % (max - min + 1);

}

inline RecordVector FiveDiagonalMethod(RecordMatrix &matrix) {

*size\_t* dimension = matrix.data.*size*();

if (!matrix.determinant()) {

return RecordVector(dimension, -1);

}

RecordVector alpha(dimension + 3), beta(dimension + 3), gamma(dimension + 3);

alpha.setValue(1, -matrix.getValue(0, 1) / matrix.getValue(0, 0));

beta.setValue(1, matrix.getValue(0, 2) / matrix.getValue(0, 0));

gamma.setValue(1, matrix.getValue(0, dimension) / matrix.getValue(0, 0));

double delta1 =

matrix.getValue(1, 1) - (-matrix.getValue(1, 0) \* alpha.getValue(1));

alpha.setValue(2, (-matrix.getValue(1, 2) -

beta.getValue(1) \* (-matrix.getValue(1, 0))) /

delta1);

gamma.setValue(2, (matrix.getValue(1, dimension) +

(-matrix.getValue(1, 0) \* gamma.getValue(1))) /

delta1);

beta.setValue(2, matrix.getValue(1, 3) / delta1);

for (*size\_t* i = 2; i <= dimension - 1; ++i) {

double delta =

matrix.getValue(i, i) -

(i < 2 ? 0 : matrix.getValue(i, i - 2)) \* beta.getValue(i - 1) +

alpha.getValue(i) \*

((i < 2 ? 0 : matrix.getValue(i, i - 2)) \* alpha.getValue(i - 1) -

(i < 1 ? 0 : -matrix.getValue(i, i - 1)));

if (i <= dimension - 2) {

alpha.setValue(

i + 1,

((i > dimension - 2 ? 0 : -matrix.getValue(i, i + 1)) +

beta.getValue(i) \* ((i < 2 ? 0 : matrix.getValue(i, i - 2)) \*

alpha.getValue(i - 1) -

(i < 1 ? 0 : -matrix.getValue(i, i - 1)))) /

delta);

}

if (i <= dimension - 3) {

beta.setValue(i + 1, (i > dimension - 3 ? 0 : matrix.getValue(i, i + 2)) /

delta);

}

gamma.setValue(

i + 1,

(matrix.getValue(i, dimension) -

(i < 2 ? 0 : matrix.getValue(i, i - 2)) \* gamma.getValue(i - 1) -

gamma.getValue(i) \*

((i < 2 ? 0 : matrix.getValue(i, i - 2)) \* alpha.getValue(i - 1) -

(i < 1 ? 0 : -matrix.getValue(i, i - 1)))) /

delta);

}

RecordVector answer(dimension + 3);

answer.setValue(dimension - 1, gamma.getValue(dimension));

answer.setValue(dimension - 2, alpha.getValue(dimension - 1) \*

answer.getValue(dimension - 1) +

gamma.getValue(dimension - 1));

for (int i = static\_cast<int>(dimension) - 3; i >= 0; --i) {

answer.setValue(i, alpha.getValue(static\_cast<*size\_t*>(i + 1)) \*

answer.getValue(static\_cast<*size\_t*>(i + 1)) -

beta.getValue(static\_cast<*size\_t*>(i + 1)) \*

answer.getValue(static\_cast<*size\_t*>(i + 2)) +

gamma.getValue(static\_cast<*size\_t*>(i + 1)));

}

answer.data.*pop\_back*();

answer.data.*pop\_back*();

answer.data.*pop\_back*();

return answer;

}

**Gauss.h**

#pragma once

#include "IFunction.h"

#include "RecordPoints.h"

inline RecordVector Gauss(RecordMatrix &matrix) {

*size\_t* dim = matrix.*data*.*size*();

RecordVector result(dim);

for (*size\_t* i = 0; i < dim; ++i) {

double main = 0;

*size\_t* m\_index = i;

for (*size\_t* m = i; m < dim; ++m) {

if (*std*::*abs*(matrix.getValue(m, i)) > main) {

main = *std*::*abs*(matrix.getValue(m, i));

m\_index = m;

}

}

if (main == 0) {

*std*::*cout* << "There is no solution" << *std*::*endl*;

return RecordVector(dim, -1);

}

if (i != m\_index) {

for (*size\_t* j = 0; j < dim + 1; ++j) {

double temp = matrix.getValue(i, j);

matrix.setValue(i, j, matrix.getValue(m\_index, j));

matrix.setValue(m\_index, j, temp);

}

}

for (*size\_t* j = i + 1; j < dim; ++j) {

double coeff = -matrix.getValue(j, i) / matrix.getValue(i, i);

for (*size\_t* k = i; k < dim + 1; ++k) {

matrix.setValue(j, k,

matrix.getValue(j, k) + coeff \* matrix.getValue(i, k));

}

}

}

result.setValue(dim - 1, matrix.getValue(dim - 1, dim) /

matrix.getValue(dim - 1, dim - 1));

for (int i = static\_cast<int>(dim) - 2; i >= 0; --i) {

double sum = matrix.getValue(i, dim);

for (*size\_t* j = static\_cast<*size\_t*>(i + 1); j < dim; ++j) {

sum -= matrix.getValue(i, j) \* result.getValue(j);

}

result.setValue(i, sum / matrix.getValue(i, i));

}

return result;

}

inline RecordVector LUDecomposition(const RecordMatrix &matrix) {

*size\_t* dim = matrix.*data*.*size*();

RecordMatrix L(dim, dim);

RecordMatrix U(dim, dim);

for (*size\_t* i = 0; i < dim; ++i) {

L.setValue(i, i, 1);

for (*size\_t* j = 0; j < dim; ++j) {

if (i <= j) {

U.setValue(i, j, matrix.getValue(i, j));

for (*size\_t* k = 0; k < i; ++k) {

U.setValue(i, j,

U.getValue(i, j) - L.getValue(i, k) \* U.getValue(k, j));

}

} else {

L.setValue(i, j, matrix.getValue(i, j));

for (*size\_t* k = 0; k < j; ++k) {

L.setValue(i, j,

L.getValue(i, j) - L.getValue(i, k) \* U.getValue(k, j));

}

L.setValue(i, j, L.getValue(i, j) / U.getValue(j, j));

}

}

}

*std*::*cout* << "L matrix:" << *std*::*endl*;

L.printMatrix();

*std*::*cout* << "U matrix:" << *std*::*endl*;

U.printMatrix();

RecordVector y(dim);

for (*size\_t* i = 0; i < dim; ++i) {

y.setValue(i, matrix.getValue(i, dim));

for (*size\_t* k = 0; k < i; ++k) {

y.setValue(i, y.getValue(i) - L.getValue(i, k) \* y.getValue(k));

}

}

RecordVector result(dim);

for (int i = static\_cast<int>(dim - 1); i >= 0; --i) {

result.setValue(i, y.getValue(i));

for (*size\_t* k = static\_cast<*size\_t*>(i + 1); k < dim; ++k) {

result.setValue(i, result.getValue(i) -

U.getValue(i, k) \* result.getValue(k));

}

result.setValue(i, result.getValue(i) / U.getValue(i, i));

}

RecordMatrix composite(dim, dim, 0);

for (*size\_t* i = 0; i < dim; i++) {

for (*size\_t* j = 0; j < dim; j++) {

for (*size\_t* k = 0; k < dim; k++) {

double newValue =

composite.getValue(i, j) + L.getValue(i, k) \* U.getValue(k, j);

composite.setValue(i, j, newValue);

}

}

}

*std*::*cout* << "L \* U matrix:" << *std*::*endl*;

composite.printMatrix();

return result;

}

class RationalFunction : public IFunction {

public:

RationalFunction(const *std*::*string* &newName) { name = newName; }

void addAnswer(const RecordVector &answer) { vector = answer; }

double evaluate(double x) const override {

return (vector.getValue(0) + vector.getValue(1) \* x +

vector.getValue(2) \* *pow*(x, 2)) /

(1 + vector.getValue(3) \* x + vector.getValue(4) \* *pow*(x, 2) +

vector.getValue(5) \* *pow*(x, 3));

}

private:

RecordVector vector;

};

**IFunction.h**

#pragma once

#include "Constants.h"

#include <cmath>

#include <iostream>

#include <map>

#include <stdexcept>

#include <vector>

class IFunction {

protected:

*std*::*vector*<*std*::*pair*<double, double>> points;

*std*::*string* name;

public:

virtual double evaluate(double x) const = 0;

int degree() const { return static\_cast<int>(points.*size*()) - 1; }

const *std*::*string* &getName() const { return name; }

};

**Plotter.h**

#pragma once

#include "Constants.h"

#include "IFunction.h"

#include "RecordPoints.h"

#include <cmath>

#include <fstream>

#include <iomanip>

#include <iostream>

#include <stdexcept>

#include <vector>

class Plotter {

private:

*std*::*string* plotterName;

*std*::*vector*<*std*::*shared\_ptr*<IFunction>> functions;

*std*::*vector*<*std*::*string*> labels;

public:

Plotter(const *std*::*string* &name);

void plotDeltaPoints(const *std*::*vector*<RecordPoints> &functions,

const *std*::*string* &name, bool toLogscale) const;

void plotPointsAndFunction(const RecordPoints &points,

*std*::*shared\_ptr*<IFunction> func, double start,

double end, bool toLogscale) const;

*std*::*string* getLabel(*size\_t* i);

};

**RecordPoints.h**

#pragma once

#include <iomanip>

#include <iostream>

#include <map>

#include <string>

#include <vector>

class RecordPoints {

public:

RecordPoints(const *std*::*string* &name) : name(name) {}

RecordPoints(const *std*::*string* &name,

const *std*::*vector*<*std*::*pair*<double, double>> &points)

: name(name), points(points) {}

void addPoint(double x, double y) { points.*emplace\_back*(x, y); }

void clearPoints() { points.clear(); }

const *std*::*vector*<*std*::*pair*<double, double>> &getVectorPoints() const {

return points;

}

const *size\_t* getSize() const { return points.*size*(); }

const *std*::*pair*<double, double> &getPoint(*size\_t* index) const {

return points[index];

}

const *std*::*string* &getName() const { return name; }

private:

*std*::*vector*<*std*::*pair*<double, double>> points;

*std*::*string* name;

};

class RecordMatrix {

public:

*std*::*vector*<*std*::*vector*<double>> data;

RecordMatrix(*size\_t* rows, *size\_t* cols, double initialValue = 0.0)

: data(rows, *std*::*vector*<double>(cols, initialValue)) {}

void printMatrix() const {

for (const auto &row : data) {

for (double val : row) {

*std*::*cout* << *std*::*fixed* << *std*::*setw*(10) << *std*::*setprecision*(5) << val

<< ' ';

}

*std*::*cout* << "\n";

}

*std*::*cout* << "\n";

}

double getValue(*size\_t* row, *size\_t* col) const {

if (row < 0 || row >= data.*size*() || col < 0 || col >= data[0].*size*()) {

throw *std*::*out\_of\_range*("Index out of matrix");

}

return data[row][col];

}

void setValue(*size\_t* row, *size\_t* col, double value) {

if (row < 0 || row >= data.*size*() || col < 0 || col >= data[0].*size*()) {

throw *std*::*out\_of\_range*("Index out of matrix");

}

data[row][col] = value;

}

double determinant() const {

*size\_t* dim = data.*size*();

RecordMatrix mat(\*this);

for (*size\_t* i = 0; i < dim; ++i) {

*size\_t* maxRow = i;

for (*size\_t* k = i + 1; k < dim; ++k) {

if (*std*::*abs*(mat.getValue(k, i)) > *std*::*abs*(mat.getValue(maxRow, i))) {

maxRow = k;

}

}

if (i != maxRow) {

for (*size\_t* j = 0; j < dim; ++j) {

double temp = mat.getValue(i, j);

mat.setValue(i, j, mat.getValue(maxRow, j));

mat.setValue(maxRow, j, temp);

}

}

if (mat.getValue(i, i) == 0) {

return 0.0;

}

for (*size\_t* j = i + 1; j < dim; ++j) {

double coeff = -mat.getValue(j, i) / mat.getValue(i, i);

for (*size\_t* k = i; k < dim; ++k) {

mat.setValue(j, k, mat.getValue(j, k) + coeff \* mat.getValue(i, k));

}

}

}

double det = 1.0;

for (*size\_t* i = 0; i < dim; ++i) {

det \*= mat.getValue(i, i);

}

return det;

}

};

class RecordVector {

public:

*std*::*vector*<double> data;

RecordVector() = default;

RecordVector(*size\_t* size, double initialValue = 0.0)

: data(size, initialValue) {}

void printVector() const {

for (double val : data) {

*std*::*cout* << *std*::*fixed* << *std*::*setw*(10) << *std*::*setprecision*(5) << val

<< ' ';

}

*std*::*cout* << "\n";

}

double getValue(*size\_t* index) const {

if (index < 0 || index >= data.*size*()) {

throw *std*::*out\_of\_range*("Index out of vector");

}

return data[index];

}

void setValue(*size\_t* index, double value) {

if (index < 0 || index >= data.*size*()) {

throw *std*::*out\_of\_range*("Index out of vector");

}

data[index] = value;

}

*size\_t* getSize() const { return data.*size*(); }

void addValue(double value) { data.*push\_back*(value); }

void clear() { data.clear(); }

};

inline void checkAnswer(RecordMatrix &matrix, const RecordVector &answer) {

*std*::*cout* << "Checking the answer\n";

*size\_t* dimension = matrix.*data*.*size*();

for (*size\_t* i = 0; i < dimension; ++i) {

double sum = 0;

for (*size\_t* j = 0; j < dimension; ++j) {

sum += matrix.getValue(i, j) \* answer.getValue(j);

}

*std*::*cout* << "row number = " << i << " direct sum: " << *std*::*fixed*

<< *std*::*setw*(15) << *std*::*setprecision*(10) << sum

<< " b: " << *std*::*fixed* << *std*::*setw*(15) << *std*::*setprecision*(10)

<< matrix.getValue(i, dimension) << *std*::*endl*;

}

}

**Tester.h**

#include "Constants.h"

#include "FiveDiagonal.h"

#include "Gauss.h"

#include "Plotter.h"

#include "RecordPoints.h"

#include <iomanip>

#include <iostream>

#include <sstream>

#pragma once

class Tester {

private:

static bool first();

static bool second();

static bool third();

static bool fourth();

static const RecordPoints initialRecord;

public:

static bool makeTest(*size\_t* number);

};

**main.cpp**

#include "Tester.h"

int main() {

*srand*(static\_cast<unsigned int>(*time*(0)));

while (1) {

*size\_t* n;

*std*::*cout* << "Test N: ";

*std*::*cin* >> n;

if (Tester::makeTest(n)) {

*std*::*cout* << "\n\n";

}

}

return 0;

}

**Plotter.cpp**

#include "Plotter.h"

Plotter::Plotter(const *std*::*string* &name) : plotterName(name) {

labels = *std*::*vector*<*std*::*string*>();

functions = *std*::*vector*<*std*::*shared\_ptr*<IFunction>>();

}

void Plotter::plotDeltaPoints(const *std*::*vector*<RecordPoints> &points,

const *std*::*string* &name, bool toLogscale) const {

*std*::*ofstream* dataFile(name + ".dat");

if (!dataFile.*is\_open*()) {

throw *std*::*runtime\_error*("Error creating data file!");

}

for (*size\_t* i = 0; i < points[0].getSize(); i++) {

dataFile << points[0].getPoint(i).*first* << " ";

for (const auto &function : points) {

dataFile << *std*::*setprecision*(*std*::*numeric\_limits*<double>::*max\_digits10*)

<< function.getPoint(i).*second* << " ";

}

dataFile << "\n";

}

dataFile.*close*();

*std*::*ofstream* gnuplotScript(plotterName + "\_point.gp");

if (!gnuplotScript.*is\_open*()) {

throw *std*::*runtime\_error*("Error creating gnuplot script!");

}

gnuplotScript << "set title '" + name + "'\n";

gnuplotScript << "set grid\n";

if (toLogscale) {

gnuplotScript << "set logscale y\n";

}

gnuplotScript << "set xlabel 'n'\n";

gnuplotScript << "set ylabel 'delta'\n";

gnuplotScript << "set format y '%.3e'\n";

gnuplotScript << "set pointsize 1\n";

gnuplotScript << "plot";

for (*size\_t* i = 0; i < points.*size*(); ++i) {

gnuplotScript << " '" + name + ".dat' using 1:" << (i + 2)

<< " with points title '" << points[i].getName() << "'";

if (i != points.*size*() - 1) {

gnuplotScript << ", \\\n";

}

}

gnuplotScript << "\n";

gnuplotScript.*close*();

*std*::*string* command = "gnuplot -p " + plotterName + "\_point.gp";

*system*(command.*c\_str*());

}

void Plotter::plotPointsAndFunction(const RecordPoints &points,

*std*::*shared\_ptr*<IFunction> func,

double start, double end,

bool toLogscale) const {

*std*::*ofstream* dataFileOver(plotterName + "\_points\_" + points.getName() +

".dat");

if (!dataFileOver.*is\_open*()) {

throw *std*::*runtime\_error*("Error creating data file!");

}

for (*size\_t* i = 0; i < points.getSize(); i++) {

dataFileOver << points.getPoint(i).*first* << " ";

dataFileOver << *std*::*setprecision*(*std*::*numeric\_limits*<double>::*max\_digits10*)

<< points.getPoint(i).*second* << " ";

dataFileOver << "\n";

}

dataFileOver.*close*();

*std*::*ofstream* gnuplotScript(plotterName + ".gp");

if (!gnuplotScript.*is\_open*()) {

throw *std*::*runtime\_error*("Error creating gnuplot script!");

}

gnuplotScript << "set grid\n";

if (toLogscale) {

gnuplotScript << "set logscale y\n";

}

gnuplotScript << "set xlabel 'x'\n";

gnuplotScript << "set ylabel 'y'\n";

gnuplotScript << "set format y '%.3e'\n";

gnuplotScript << "set pointsize 1\n";

gnuplotScript << "plot";

gnuplotScript << " '" << plotterName << "\_points\_" << points.getName()

<< ".dat' using 1 : 2 with points title '" << points.getName()

<< "',\ ";

*std*::*ofstream* dataFile2(plotterName + "\_function\_" + func->getName() +

".dat");

if (!dataFile2.*is\_open*()) {

throw *std*::*runtime\_error*("Error creating data file!");

}

double step = (end - start) / ((nodes)-1);

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

double x = start + i \* step;

dataFile2 << x << " " << func->evaluate(x);

dataFile2 << "\n";

}

gnuplotScript << " '" << plotterName << "\_function\_" << func->getName()

<< ".dat' using 1:2 with lines title '" << func->getName()

<< "'";

gnuplotScript.*close*();

*std*::*string* command = "gnuplot -p " + plotterName + ".gp";

*system*(command.*c\_str*());

}

*std*::*string* Plotter::getLabel(*size\_t* i) { return labels[i]; }

**Tester.cpp**

#include "Tester.h"

const RecordPoints Tester::initialRecord =

RecordPoints("Initial Points", {

{-2, 0.28205},

{-1.20, 1.0452},

{-0.400, 2.5052},

{0.400, 1.1723},

{1.20, -0.20842},

{2., -0.28661},

});

bool Tester::first() {

try {

Plotter plotter("1");

*size\_t* dimension = initialRecord.getSize();

RecordMatrix matrix(dimension, dimension + 1, -1);

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

int p = 2;

int q = 3;

double x\_i = initialRecord.getPoint(i).first;

double y\_i = initialRecord.getPoint(i).second;

for (*size\_t* j = 0; j < dimension; ++j) {

if (j <= p) {

matrix.setValue(i, j, *std*::*pow*(x\_i, j));

} else {

matrix.setValue(i, j, -y\_i \* *std*::*pow*(x\_i, j - p));

}

}

matrix.setValue(i, dimension, y\_i);

}

matrix.printMatrix();

auto gaussAnswer = Gauss(matrix);

*std*::*cout* << "Gauss answer:\n";

gaussAnswer.printVector();

*std*::*cout* << "\n";

RationalFunction gauss("Gausian");

gauss.addAnswer(gaussAnswer);

plotter.plotPointsAndFunction(

initialRecord, *std*::*make\_shared*<RationalFunction>(gauss), -2, 2, false);

checkAnswer(matrix, gaussAnswer);

} catch (const *std*::*exception* &e) {

*std*::*cerr* << "Error: " << e.*what*() << "\n";

return false;

}

return true;

}

bool Tester::second() {

try {

Plotter plotter("2");

*size\_t* dimension = initialRecord.getSize();

RecordMatrix matrix(dimension, dimension + 1, -1);

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

int p = 2;

int q = 3;

double x\_i = initialRecord.getPoint(i).first;

double y\_i = initialRecord.getPoint(i).second;

for (*size\_t* j = 0; j < dimension; ++j) {

if (j <= p) {

matrix.setValue(i, j, *std*::*pow*(x\_i, j));

} else {

matrix.setValue(i, j, -y\_i \* *std*::*pow*(x\_i, j - p));

}

}

matrix.setValue(i, dimension, y\_i);

}

matrix.printMatrix();

auto LuAnswer = LUDecomposition(matrix);

*std*::*cout* << "LU answer:\n";

LuAnswer.printVector();

*std*::*cout* << "\n";

RationalFunction LUrian("LU");

LUrian.addAnswer(LuAnswer);

plotter.plotPointsAndFunction(initialRecord,

*std*::*make\_shared*<RationalFunction>(LUrian),

-2, 2, false);

checkAnswer(matrix, LuAnswer);

} catch (const *std*::*exception* &e) {

*std*::*cerr* << "Error: " << e.*what*() << "\n";

return false;

}

return true;

}

bool Tester::third() {

try {

} catch (const *std*::*exception* &e) {

*std*::*cerr* << "Error: " << e.*what*() << "\n";

return false;

}

return true;

}

bool Tester::fourth() {

try {

Plotter plotter("4");

*size\_t* dimension = 8;

RecordMatrix matrix(dimension, dimension, 0);

matrix.*data*.*resize*(dimension);

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

matrix.*data*[i].*resize*(dimension + 1);

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

if (*std*::*abs*(i - j) == 0) {

double value = RandomDouble(50, 90);

matrix.setValue(i, j, value);

} else if (*std*::*abs*(i - j) == 1) {

double value = RandomDouble(20, 50);

matrix.setValue(i, j, value);

} else if (*std*::*abs*(i - j) == 2) {

double value = RandomDouble(5, 10);

matrix.setValue(i, j, value);

} else {

matrix.setValue(i, j, 0);

}

}

matrix.setValue(i, dimension, RandomDouble(-1, 1));

}

matrix.printMatrix();

auto fiveDiagonalAnswer = FiveDiagonalMethod(matrix);

*std*::*cout* << "fiveDiagonal answer:\n";

fiveDiagonalAnswer.printVector();

*std*::*cout* << "\n";

checkAnswer(matrix, fiveDiagonalAnswer);

} catch (const *std*::*exception* &e) {

*std*::*cerr* << "Error: " << e.*what*() << "\n";

return false;

}

return true;

}

bool Tester::makeTest(*size\_t* number) {

switch (number) {

case 1:

return first();

case 2:

return second();

case 3:

return third();

case 4:

return fourth();

default:

return false;

}

}