**ФГБОУ ВО**

**Уфимский государственный авиационный технический университет**

**Кафедра Информатики**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 90 |  |  |  |  |  |  |  |  |  |  |  |  |
| 80 |  |  |  |  |  |  |  |  |  |  |  |  |
| 70 |  |  |  |  |  |  |  |  |  |  |  |  |
| 60 |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

**Дисциплина:** Численные методы

**Отчет по лабораторной работе № 3**

**Тема:** «Прямые методы решения систем линейных алгебраических уравнений»

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Группа |  |  | Фамилия И. О. | Подпись | Дата | Оценка |
| МКН-318Б |  |
|  |  |
| Студент | | | Халитова А.А. |  |  |  |
| Преподаватель | | | Феоктистов Б.А. |  |  |  |
|  | | |  |  |  |  |

**Уфа 2024 г.**

**Содержание**

[1 Введение 2](#_Toc183168758)

[2 Теоретическая часть 3](#_Toc183168759)

[2.1 Метод Гаусса 3](#_Toc183168760)

[2.2 Метод LU-разложения. 4](#_Toc183168761)

[2.3 Метод квадратного корня. 5](#_Toc183168762)

[2.4 Метод прогонки 5](#_Toc183168763)

[3 Практическая часть 7](#_Toc183168764)

[3.1 Задание 1. 7](#_Toc183168765)

[3.2 Задание 2 8](#_Toc183168766)

[3.3 Задание 3 10](#_Toc183168768)

[3.4 Задание 4 12](#_Toc183168769)

[4 Заключение 14](#_Toc183168770)

[5 Список литературы 15](#_Toc183168771)

[6 Листинг программы 16](#_Toc183168772)

# Введение

Целью лабораторной работы является получение навыка приближенного вычисления определенных интегралов.

# Теоретическая часть

## Метод Гаусса

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.

## Метод LU-разложения.

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 квадратного корня.

Пусть – данная симметричная матрица, т.е. . Будем строить её представление в виде .

Матрица может быть определена совокупностью формул

## Метод прогонки

Дана система пятиточечных уравнений:

Text, letter

Description automatically generated

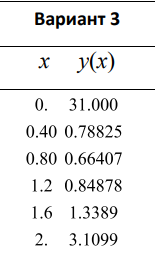
Запишем алгоритм правой прогонки для системы (1)-(5) в следующем виде:

Text, letter

Description automatically generated

Получили решение СЛАУ с пятидиагональной матрицей.

Вариант 3:



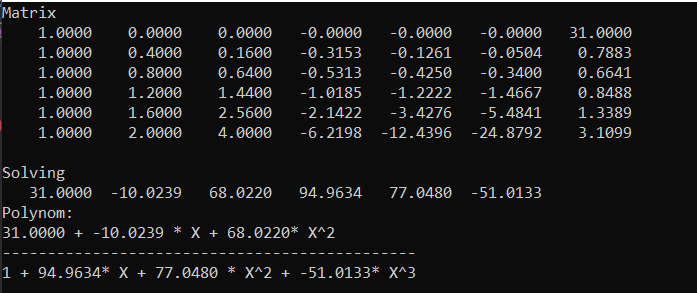
# Практическая часть

## Задание 1.

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ методом Гаусса с выбором ведущего элемента (рисунок 1).
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 2)

  
Рисунок 1 – решение СЛАУ методом Гаусса с выбором ведущего элемента

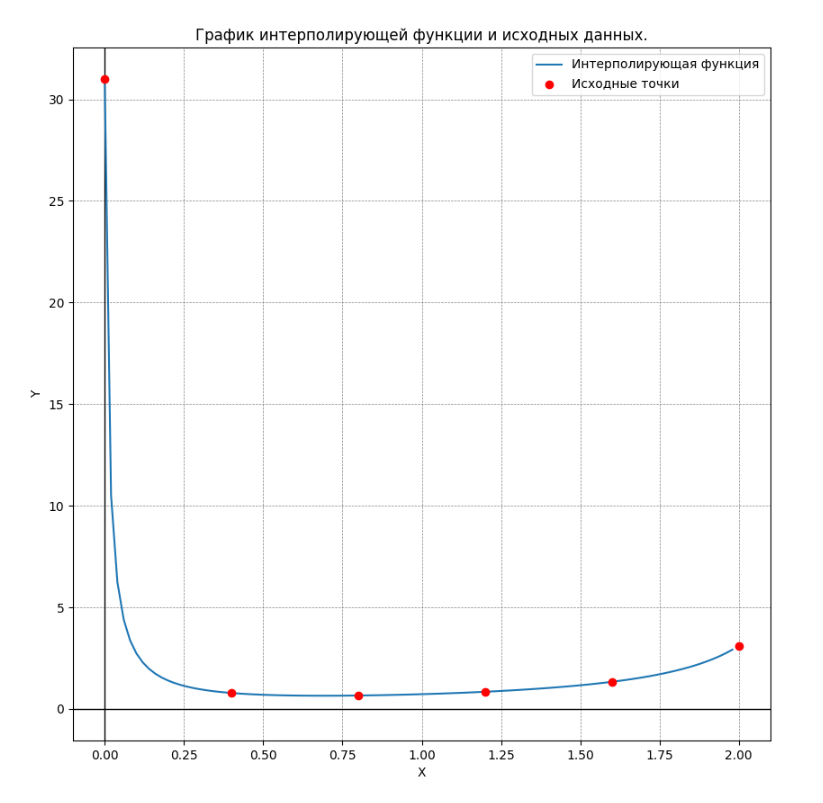


Рисунок 2 - график интерполирующей функции и исходных данных для задачи №1.

**Вывод:** В ходе выполнения задания 1 были изучены основные принципы решения СЛАУ методом Гаусса с выбором ведущего элемента и проведена интерполяция таблично заданной функции с использованием рациональной функции.

Построен график интерполирующей функции и исходных данных.

## Задание 2

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ методом LU-разложения (рисунок 3).
2. Выполнить п. 2), 3) Задачи 1 (рисунок 4).

## 

Рисунок 3 – решение СЛАУ LU-разложением с проверкой

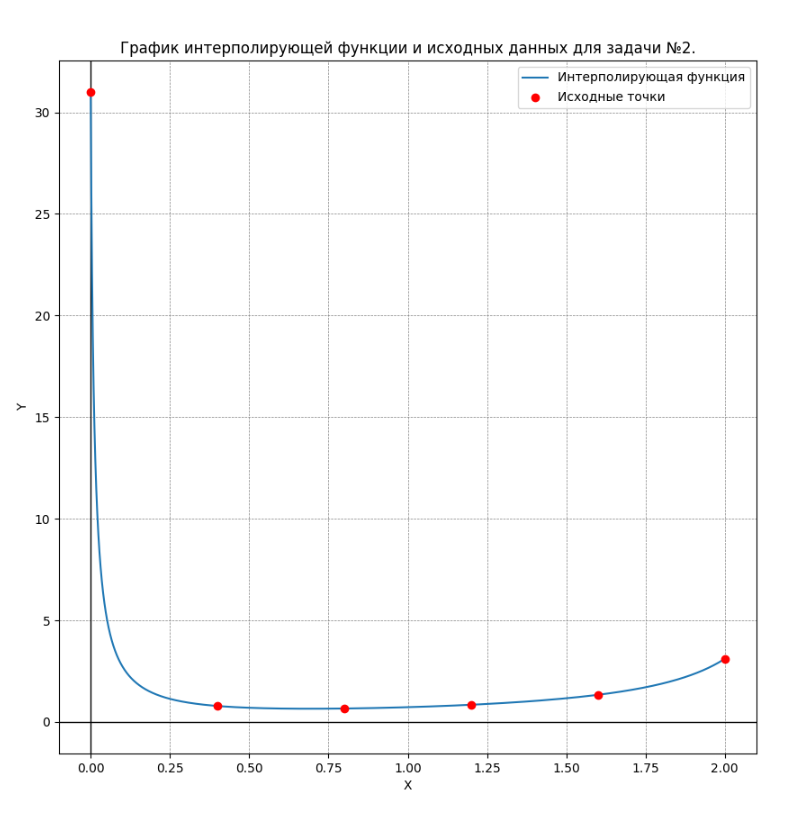


Рисунок 4 – график интерполирующей функции и исходных данных для задачи №2.

**Вывод:** В ходе выполнения задания 2 были изучены основные принципы решения СЛАУ с помощью LU-разложения. Также была реализована функция для проверки правильности выполнения разложения.

Построен график интерполирующей функции и исходных данных, который совпадает с графиком в задании №1, что подтверждает его эффективность.

## Задание 3

1. Написать вычислительную программу на языке программирования C++ для решения СЛАУ с симметричной матрицей методом квадратного корня.
2. С использованием написанной программы решить задачу об аппроксимации функции из первой лабораторной работы, заданной на равномерной сетке из 20 узлов, многочленами степени с использованием метода наименьших квадратов.
3. Построить графики аппроксимирующих многочленов и исходных данных (рисунок 5).
4. Определить степень многочлена, обеспечивающего наилучшее приближение (соответствующее наименьшему значению суммы квадратов отклонений значений многочлена в узлах сетки от исходных данных) ( рисунок 6).

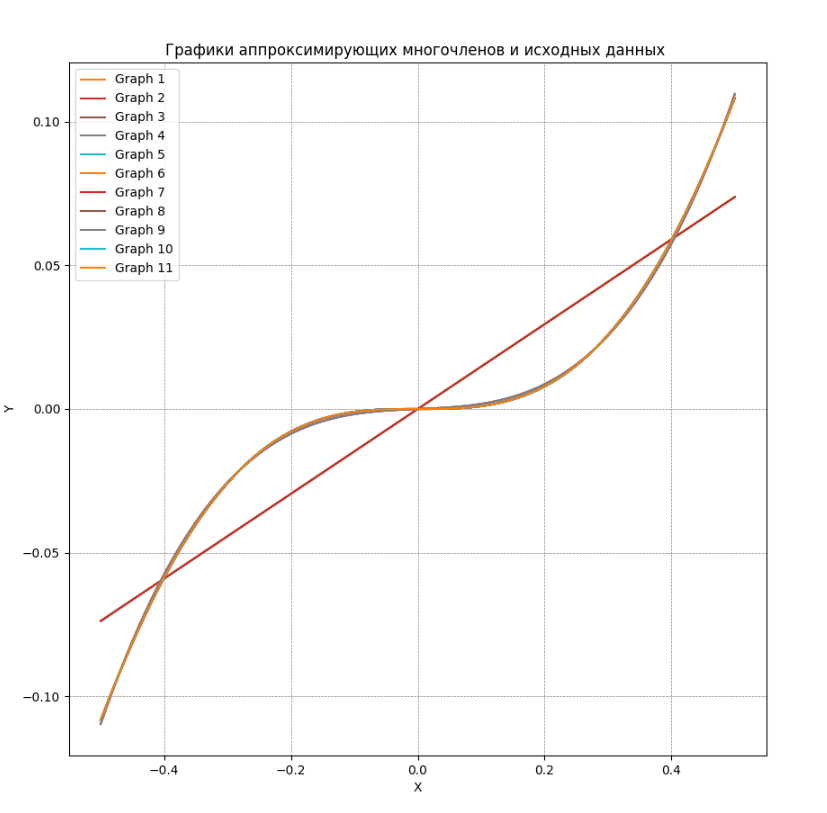


Рисунок 5 - графики аппроксимирующих многочленов и исходных данных

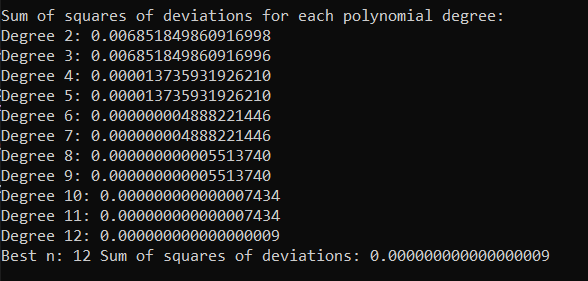


Рисунок 6 – выполнение программы

**Вывод:** В ходе выполнения задания 3 была разработана программа для решения СЛАУ методом квадратного корня и проведена аппроксимация функции с использованием метода наименьших квадратов.

Была определена степень многочлена n = 12, обеспечивающего наилучшее приближение (соответствующее наименьшему значению суммы квадратов отклонений значений многочлена в узлах сетки от исходных данных).

Построены графики аппроксимирующих многочленов и исходных данных.

## Задание 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 7).

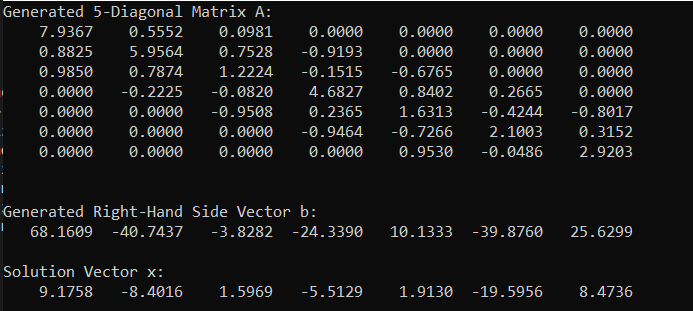


Рисунок 7 – пример работы программы

**Вывод:** В ходе выполнения задания 6   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 ходе выполнения лабораторной работы по теме “Прямые методы решения систем линейных алгебраических уравнений” были изучены и реализованы различные алгоритмы, позволяющие эффективно решать СЛАУ

Были созданы программы на языке C++ для решения СЛАУ с использованием различных методов.

Лабораторная работа позволила углубить понимание численных методов интегрирования и их применения в различных задачах.

# Список литературы

1. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. Численные методы. — Москва: Наука, 2000. — 320 с.
2. Калиткин Н.Н. Численные методы. — Москва: Высшая школа, 2005. — 256 с.

# Листинг программы

**first\_ex.cpp**

#include "Constants.h"

vector<vector<double>> FillMatrix() {

int p = 2;

int q = 3;

vector<vector<double>> A(dimension, vector<double>(p + q + 2));

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

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

A[i][j] = pow(x[i], j);

}

for (int j = p + 1; j <= p + q; j++) {

A[i][j] = -y[i] \* pow(x[i], j - p);

}

A[i][p + q + 1] = y[i];

}

return A;

}

void print(const vector<vector<double>>& q) {

cout << fixed << setprecision(4);

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

for (int j = 0; j < q[i].size(); j++) {

cout << setw(10) << q[i][j];

}

cout << endl;

}

cout << endl;

}

void print(const vector<double>& q) {

cout << fixed << setprecision(4);

for (int j = 0; j < q.size(); j++) {

cout << setw(10) << q[j];

}

cout << endl;

}

vector<double> gaussElimination(vector<vector<double>>& A, vector<double>& b) {

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

double maxEl = abs(A[i][i]);

int maxRow = i;

for (int k = i + 1; k < dimension; k++) {

if (abs(A[k][i]) > maxEl) {

maxEl = abs(A[k][i]);

maxRow = k;

}

}

swap(A[i], A[maxRow]);

swap(b[i], b[maxRow]);

for (int k = i + 1; k < dimension; k++) {

double factor = A[k][i] / A[i][i];

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

A[k][j] -= factor \* A[i][j];

}

b[k] -= factor \* b[i];

}

}

vector<double> x\_ans(dimension);

for (int i = dimension - 1; i >= 0; i--) {

x\_ans[i] = b[i];

for (int j = i + 1; j < dimension; j++) {

x\_ans[i] -= A[i][j] \* x\_ans[j];

}

x\_ans[i] /= A[i][i];

}

return x\_ans;

}

void InterpolationTask(vector<double>& res) {

ofstream first\_ex("first\_ex.txt");

int nodes = 10000;

double a = x[0];

double b = x[dimension - 1];

double h = (b - a) / nodes;

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

double X = a + i \* h;

double f = (res[0] + res[1] \* X + res[2] \* X \* X) / (1 + res[3] \* X + res[4] \* X \* X + res[5] \* X \* X \* X);

first\_ex << X << " " << f << endl;

}

}

void first() {

vector<vector<double>> A = FillMatrix();

cout << "Matrix" << endl;

print(A);

cout << "Solving " << endl;

vector<double>y\_temp = y;

vector<double> coefficients = gaussElimination(A, y\_temp);

print(coefficients);

InterpolationTask(coefficients);

}

**Second\_ex.cpp**

#include "Constants.h"

vector<vector<double>> FillMatrix();

void print(const vector<vector<double>>& q);

void print(const vector<double>& q);

void prove(const vector<vector<double>>& L, const vector<vector<double>>& U) {

size\_t n = L.size();

vector<vector<double>> result(n, vector<double>(n, 0));

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

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

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

result[i][j] += L[i][k] \* U[k][j];

}

}

}

cout << "Result of L \* U:" << endl;

print(result);

}

void LU\_Decomposition(vector<vector<double>>& A, vector<vector<double>>& L, vector<vector<double>>& U) {

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

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

U[i][j] = A[i][j];

}

}

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

L[i][i] = 1;

for (int j = i + 1; j < dimension; j++) {

L[j][i] = U[j][i] / U[i][i];

}

for (int k = i + 1; k < dimension; k++) {

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

U[k][j] -= L[k][i] \* U[i][j];

}

}

}

cout << "L:" << endl;

print(L);

cout << "U:" << endl;

print(U);

}

vector<double> forwardSubstitution(const vector<vector<double>>& L, const vector<double>& b) {

vector<double> y\_temp(dimension);

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

y\_temp[i] = b[i];

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

y\_temp[i] -= L[i][j] \* y\_temp[j];

}

}

return y\_temp;

}

vector<double> backwardSubstitution(const vector<vector<double>>& U, const vector<double>& y) {

vector<double> x\_temp(dimension);

for (int i = dimension - 1; i >= 0; i--) {

x\_temp[i] = y[i];

for (int j = i + 1; j < dimension; j++) {

x\_temp[i] -= U[i][j] \* x\_temp[j];

}

x\_temp[i] /= U[i][i];

}

return x\_temp;

}

vector<double> solveLU(vector<vector<double>>& A, const vector<double>& b) {

vector<vector<double>> L(dimension, vector<double>(dimension, 0));

vector<vector<double>> U(dimension, vector<double>(dimension, 0));

LU\_Decomposition(A, L, U);

prove(L, U);

vector<double> y\_temp = forwardSubstitution(L, b);

vector<double> x\_temp = backwardSubstitution(U, y\_temp);

return x\_temp;

}

void InterpolationTaskSecond(vector<double>& res) {

ofstream first\_ex("second\_ex.txt");

int nodes = 100;

double a = x[0];

double b = x[dimension - 1];

double h = (b - a) / nodes;

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

double X = a + i \* h;

double f = (res[0] + res[1] \* X + res[2] \* X \* X) / (1 + res[3] \* X + res[4] \* X \* X + res[5] \* X \* X \* X);

first\_ex << X << " " << f << endl;

}

}

void second() {

vector<vector<double>> A = FillMatrix();

cout << "Matrix A:" << endl;

print(A);

vector<vector<double>> L(dimension, vector<double>(dimension, 0));

vector<vector<double>> U(dimension, vector<double>(dimension, 0));

vector<double> solution = solveLU(A, y);

cout << "Solution x:" << endl;

print(solution);

InterpolationTaskSecond(solution);

}

**Third\_ex.cpp**

#include "Constants.h"

void print(const vector<vector<double>>& q);

void print(const vector<double>& q);

double f(double x) {

return (x \* x \* x) \* sqrt(1 - x \* x);

}

vector<double> SquareRootMethod(const vector<vector<double>>& matrix, const vector<double>& b) {

size\_t dim = matrix.size();

vector<vector<double>> U(dim, vector<double>(dim, 0.0));

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

U[i][i] = matrix[i][i];

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

U[i][i] -= U[k][i] \* U[k][i];

}

U[i][i] = sqrt(U[i][i]);

for (int j = i + 1; j < dim; ++j) {

U[i][j] = matrix[i][j];

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

U[i][j] -= U[k][i] \* U[k][j];

}

U[i][j] /= U[i][i];

}

}

vector<double> y(dim);

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

y[i] = b[i];

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

y[i] -= U[k][i] \* y[k];

}

y[i] /= U[i][i];

}

vector<double> x(dim);

for (int i = dim - 1; i >= 0; --i) {

x[i] = y[i];

for (int k = i + 1; k < dim; ++k) {

x[i] -= U[i][k] \* x[k];

}

x[i] /= U[i][i];

}

return x;

}

void third() {

int n = 12;

int nodes = 20;

vector<double> x\_temp(nodes + 1);

vector<double> y\_temp(nodes + 1);

double h = (b - a) / nodes;

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

double x\_t = a + h \* i;

x\_temp[i] = x\_t;

y\_temp[i] = f(x\_t);

}

vector<vector<double>> answers;

for (int m = 2; m <= n; m++) {

vector<vector<double>> A(m, vector<double>(m, 0.0));

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

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

double sum = 0.0;

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

sum += pow(x\_temp[l], i + j);

}

A[i][j] = sum;

}

}

cout << "Matrix" << endl;

print(A);

vector<double> b\_vector(m, 0.0);

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

double sum\_b = 0.0;

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

sum\_b += y\_temp[j] \* pow(x\_temp[j], i);

}

b\_vector[i] = sum\_b;

}

//print(b\_vector);

vector<double> answer = SquareRootMethod(A, b\_vector);

//cout << "Square root method answer: " << endl;

//print(answer);

answers.push\_back(answer);

}

print(answers);

ofstream third\_ex("third\_ex.txt");

int N = 100;

double h\_fine = (b - a) / N;

for (int e = 0; e < answers.size(); e++) {

third\_ex << N + 1 << endl;

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

double x\_t = a + i \* h\_fine;

double approxValue = 0.0;

for (int j = 0; j <= e + 2 && j < answers[e].size(); j++) {

approxValue += answers[e][j] \* pow(x\_t, j);

}

third\_ex << x\_t << " " << approxValue << endl;

}

}

third\_ex.close();

// Вычисление сумм квадратов отклонений

vector<double> sum\_squares\_errors(answers.size(), 0.0);

double min\_val = numeric\_limits<double>::max();

int best\_degree\_index = -1;

for (size\_t i = 0; i < x\_temp.size(); ++i) {

for (size\_t j = 0; j < answers.size(); ++j) {

double f\_val = 0.0;

for (size\_t e = 0; e <= j + 2 && e < answers[j].size(); e++) { // значение полинома для текущей точки

f\_val += answers[j][e] \* pow(x\_temp[i], e);

}

sum\_squares\_errors[j] += pow(f\_val - y\_temp[i], 2); // сумма квадратов отклонений

}

}

for (size\_t j = 0; j < sum\_squares\_errors.size(); ++j) { // ищем полином с наименьшей суммой квадратов отклонений

if (sum\_squares\_errors[j] < min\_val) {

min\_val = sum\_squares\_errors[j];

best\_degree\_index = static\_cast<int>(j);

}

}

cout << "Sum of squares of deviations for each polynomial degree:" << endl;

for (size\_t j = 0; j < sum\_squares\_errors.size(); ++j) {

cout << "Degree " << (j + 2) << ": " << sum\_squares\_errors[j] << endl;

}

cout << "Best n: " << best\_degree\_index + 2 << " Sum of squares of deviations: " << sum\_squares\_errors[best\_degree\_index] << endl;

}

**Fourth\_ex.cpp**

#include "Constants.h"

void print(const vector<vector<double>>& q);

void print(const vector<double>& q);

double randomNonZeroDouble(double min, double max) {

random\_device rd;

mt19937 mt(rd());

uniform\_real\_distribution<double> dist(min, max);

double value;

do {

value = dist(mt);

} while (value == 0.0);

return value;

}

vector<vector<double>> generateFiveDiagonalMatrix(int size) {

vector<vector<double>> matrix(size, vector<double>(size, 0.0));

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

matrix[i][i] = randomNonZeroDouble(1.0, 10.0);

if (i > 0) {

matrix[i][i - 1] = randomNonZeroDouble(-1.0, 1.0);

}

if (i < size - 1) {

matrix[i][i + 1] = randomNonZeroDouble(-1.0, 1.0);

}

if (i > 1) {

matrix[i][i - 2] = randomNonZeroDouble(-1.0, 1.0);

}

if (i < size - 2) {

matrix[i][i + 2] = randomNonZeroDouble(-1.0, 1.0);

}

}

return matrix;

}

vector<double> generateRightHandSide(int size) {

vector<double> b(size);

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

b[i] = randomNonZeroDouble(-100.0, 100.0);

}

return b;

}

vector<double> solveTridiagonal(const vector<vector<double>>& A, const vector<double>& b) {

int n = b.size();

vector<double> c(n), d(n), x(n);

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

if (i == 0) {

c[i] = A[i][i + 1] / A[i][i];

d[i] = b[i] / A[i][i];

}

else if (i == n - 1) {

d[i] = (b[i] - A[i][i - 1] \* d[i - 1]) / A[i][i];

}

else {

c[i] = A[i][i + 1] / (A[i][i] - A[i][i - 1] \* c[i - 1]);

d[i] = (b[i] - A[i][i - 1] \* d[i - 1]) / (A[i][i] - A[i][i - 1] \* c[i - 1]);

}

}

x[n - 1] = d[n - 1];

for (int i = n - 2; i >= 0; --i) {

x[i] = d[i] - c[i] \* x[i + 1];

}

return x;

}

void fourth() {

int n = 7;

vector<vector<double>> A = generateFiveDiagonalMatrix(n);

vector<double> b = generateRightHandSide(n);

cout << "Generated 5-Diagonal Matrix A:" << endl;

print(A);

cout << "\nGenerated Right-Hand Side Vector b:" << endl;

print(b);

vector<double> solution = solveTridiagonal(A, b);

cout << "\nSolution Vector x:" << endl;

print(solution);

}

**Constants.h**

#pragma once

#include <vector>

#include <iostream>

#include <iomanip>

#include <fstream>

#include <random>

using namespace std;

const vector<double> x = { 0., 0.4, 0.8, 1.2, 1.6, 2. };

const vector<double> y = { 31., 0.78825, 0.66407, 0.84878, 1.3389, 3.1099 };

const int dimension = 6;

const double a = -0.5;

const double b = 0.5;

**main.cpp**

void first();

void second();

void third();

void fourth();

int main(){

first();

}