МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической физики и вычислительной математики

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ

По дисциплине “Методы вычисления”

студентки 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета компьютерных наук и информационных технологий
Никитенко Яны Валерьевны

Проверил: _____ _____
подпись, дата

Саратов 2025

Раздел 1

1. По данным интерполяции построить интерполяционный многочлен в общем виде, в форме Лагранжа и в форме Ньютона.

Исходные данные

X(k) 1 3 5 7

F(k) 0 7 2 18

Текст программы

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <string>
#include <locale>
using namespace std;
// Функция для ввода данных с клавиатуры
void inputData(vector<double>& x_data, vector<double>& f_data) {
    int n;
    cout « "Введите количество точек: ";
    cin » n;
    x_data.resize(n);
    f_data.resize(n);
    cout « "\nВведите значения X:" « endl;
    for (int i = 0; i < n; i++) {
        cout « "X[" « i « "] = ";
        cin » x_data[i];
    }
    cout « "\nВведите значения F(X):" « endl;
    for (int i = 0; i < n; i++) {
        cout « "F(" « x_data[i] « ") = ";
        cin » f_data[i];
    }
}
// Функция для вывода введенных данных
void printInputData(const vector<double>& x_data,
const vector<double>& f_data) {
    cout « "\nВведенные данные:" « endl;
    cout « setw(10) « "X" « setw(10) « "F(X)" « endl;
    cout « string(20, '-') « endl;
    for (int i = 0; i < x_data.size(); i++) {
        cout « setw(10) « x_data[i] « setw(10) « f_data[i] « endl;
    }
    cout « endl;
}
// Функция для решения системы линейных уравнений методом Гаусса
vector<double> solveLinearSystem(vector<vector<double» A,
vector<double> b) {
    int n = A.size();
    // Прямой ход метода Гаусса
    for (int i = 0; i < n; i++) {
        // Поиск главного элемента
        int maxRow = i;
        for (int k = i + 1; k < n; k++) {
            if (abs(A[k][i]) > abs(A[maxRow][i])) {
                maxRow = k;
```

```
            }
        }
        // Перестановка строк
        swap(A[i], A[maxRow]);
        swap(b[i], b[maxRow]);
        // Исключение
        for (int k = i + 1; k < n; k++) {
            double factor = A[k][i] / A[i][i];
            for (int j = i; j < n; j++) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
    }
    // Обратный ход
    vector<double> x(n);
    for (int i = n - 1; i >= 0; i–) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
        x[i] /= A[i][i];
    }
    return x;
}
// Функция для вычисления значения полинома
double polynomial(double x, const vector<double>& coeffs) {
    double result = 0.0;
    for (int i = 0; i < coeffs.size(); i++) {
        result += coeffs[i] * pow(x, i);
    }
    return result;
}
// Функция для представления полинома в виде строки
string polynomialToString(const vector<double>& coeffs) {
    string result;
    int degree = coeffs.size() - 1;

    for (int i = 0; i < coeffs.size(); i++) {
        if (coeffs[i] != 0 || degree == 0) {
            if (!result.empty() && coeffs[i] >= 0) {
                result += " + ";
            }
            else if (!result.empty() && coeffs[i] < 0) {
                result += " - ";
            }
            string term;
            if (i == 0) {
                term = to_string(coeffs[i]);
            }
            else {
                double absCoef = abs(coeffs[i]);
                if (absCoef != 1.0) {
                    term = to_string(absCoef) + " * ";
                }
                term += "x";
                if (i > 1) {
                    term += "^" + to_string(i);
                }
            }
            // Удаляем лишние нули после запятой
```

```
            size_t pos = term.find('.');
            if (pos != string::npos) {
                size_t lastNonZero = term.find_last_not_of('0');
                if (lastNonZero == pos) {
                    term = term.substr(0, pos);
                }
                else {
                    term = term.substr(0, lastNonZero + 1);
                }
            }
            result += term;
        }
        degree--;
    }
    return result;
}
// Функция для вычисления коэффициентов многочлена Ньютона
vector<double> newtonCoefficients(const vector<double>& x,
const vector<double>& y) {
    int n = x.size();
    vector<double> coeffs = y;
    for (int j = 1; j < n; j++) {
        for (int i = n - 1; i >= j; i--) {
            coeffs[i] = (coeffs[i] - coeffs[i - 1]) / (x[i] - x[i - j]);
        }
    }
    return coeffs;
}
// Функция для вычисления значения многочлена Ньютона
double newtonPolynomial(double x, const vector<double>& coeffs,
const vector<double>& x_data) {
    int n = coeffs.size() - 1;
    double p = coeffs[n];

    for (int k = 1; k <= n; k++) {
        p = coeffs[n - k] + (x - x_data[n - k]) * p;
    }
    return p;
}
// Функция для вывода вектора
void printVector(const string& name, const vector<double>& vec) {
    cout « name « ": ";
    for (double val : vec) {
        cout « val « " ";
    }
    cout « endl;
}
// Функция для вывода матрицы
void printMatrix(const string& name,
const vector<vector<double»& matrix) {
    cout « name « ":" « endl;
    for (const auto& row : matrix) {
        for (double val : row) {
            cout « setw(8) « val « " ";
        }
        cout « endl;
    }
}
int main() {
     setlocale(LC_ALL, "Russian");
    vector<double> x_data, f_data;
```

```
// Меню выбора способа ввода данных
int choice;
cout « "Выберите способ ввода данных:" « endl;
cout « "1 - Ввести данные с клавиатуры" « endl;
cout « "2 - Использовать пример данных (X: 81, 21, 2, 3; F: 0, 7, 8, 28)" « endl;
cout « "Ваш выбор: ";
cin » choice;
if (choice == 1) {
    inputData(x_data, f_data);
}
else {
    // Пример данных по умолчанию
    x_data = { 81, 21, 2, 3 };
    f_data = { 0, 7, 8, 28 };
}
printInputData(x_data, f_data);
// Проверка на достаточное количество точек
if (x_data.size() < 2) {
    cout « "Ошибка: необходимо как минимум
    2 точки для интерполяции!" « endl;
    return 1;
}
// Построение матрицы Вандермонда
int n = x_data.size();
vector<vector<double» A(n, vector<double>(n));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        A[i][j] = pow(x_data[i], j);
    }
}
printMatrix("Матрица", A);
cout « endl;
// Решение системы уравнений для нахождения коэффициентов
vector<double> coefficients = solveLinearSystem(A, f_data);
cout « "Найденные коэффициенты:" « endl;
for (int i = 0; i < coefficients.size(); i++) {
    cout « "a" « i « " = " « coefficients[i] « endl;
}
cout « endl;
// Вывод уравнения многочлена
cout « "Уравнение интерполяционного многочлена:" « endl;
cout « polynomialToString(coefficients) « endl « endl;
// Вычисление коэффициентов для многочлена Ньютона
vector<double> newton_coeffs = newtonCoefficients(x_data, f_data);
cout « "Коэффициенты многочлена Ньютона:" « endl;
for (int i = 0; i < newton_coeffs.size(); i++) {
    cout « "b" « i « " = " « newton_coeffs[i] « endl;
}
cout « endl;
// Расчет значений в промежуточных точках
vector<double> x_half;
for (int i = 0; i < x_data.size() - 1; i++) {
    x_half.push_back((x_data[i] + x_data[i + 1]) / 2.0);
}
// Объединение и сортировка всех точек
vector<double> x_combined = x_data;
x_combined.insert(x_combined.end(), x_half.begin(), x_half.end());
sort(x_combined.begin(), x_combined.end());
// Вывод итоговой таблицы
cout « "Итоговая таблица:" « endl;
cout « setw(10) « "X" « setw(15) « "F (исходные)"
```

```
        « setw(15) « "F (Лагранж)" « setw(15) « "F (Ньютон)" « endl;
    cout « string(55, '-') « endl;
    for (double x : x_combined) {
        // Проверяем, является ли x исходной точкой
        bool isOriginal = find(x_data.begin(), x_data.end(), x) != x_data.end();
        double original_f = 0.0;
        if (isOriginal) {
            auto it = find(x_data.begin(), x_data.end(), x);
            int index = distance(x_data.begin(), it);
            original_f = f_data[index];
        }
        double lagrange_f = polynomial(x, coefficients);
        double newton_f = newtonPolynomial(x, newton_coeffs, x_data);

        cout « fixed « setprecision(4);
        cout « setw(10) « x;
        if (isOriginal) {
            cout « setw(15) « original_f;
        }
        else {
            cout « setw(15) « "-";
        }
        cout « setw(15) « lagrange_f « setw(15) « newton_f « endl;
    }
    return 0;
}
```

Вывод программы

Матрица:

$$\begin{pmatrix} 3 & 9 & 27 \\ 5 & 25 & 125 \\ 7 & 49 & 343 \end{pmatrix}$$

Найденные коэффициенты:

$a_0 = -18.3125, \quad a_1 = 25.3125, \quad a_2 = -7.6875, \quad a_3 = 0.6875$

Уравнение интерполяционного многочлена:

$P(x) = -18.312500 + 25.312500 \cdot x - 7.687500 \cdot x^2 + 0.687500 \cdot x^3$

Коэффициенты многочлена Ньютона:

$b_0 = 0, \quad b_1 = 3.5, \quad b_2 = -1.5, \quad b_3 = 0.6875$

Итоговая таблица значений:

X	F (исходные)	F (Лагранж)	F (Ньютон)
1.0000	0.0000	0.0000	0.0000
2.0000	$--$	7.0625	7.0625
3.0000	7.0000	7.0000	7.0000
4.0000	$--$	3.9375	3.9375
5.0000	2.0000	2.0000	2.0000
6.0000	$--$	5.3125	5.3125
7.0000	18.0000	18.0000	18.0000

2. Построить кусочно-непрерывную склейку кубических сплайнов.

Исходные данные

Узлы интерполяции (x и f(x)):

Узел 1 - x: 11

Узел 1 - f(x): 2

Узел 2 - x: 7

Узел 2 - f(x): 88

Узел 3 - x: 14

Узел 3 - f(x): 60

Узел 4 - x: 10

Узел 4 - f(x): 20

Узел 5 - x: 1

Узел 5 - f(x): 50

Коэффициенты кубического многочлена:

a3: 2

a2: 3

a1: 7

a0: 25

Текст программы

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <cmath>
#include <string>
#include <locale>
using namespace std;
// Структура для хранения коэффициентов полинома
struct PolynomialCoeffs {
    double a3, a2, a1, a0;
};
// Структура для хранения коэффициентов сплайна на отрезке
struct SplineSegment {
    double a, b, c, d;
    double x_left;
};
// Функция для решения СЛАУ методом прогонки (трехдиагональная матрица)
vector<double> solveTridiagonal(const vector<vector<double»& A,
const vector<double>& b) {
    int n = b.size();
    vector<double> alpha(n, 0), beta(n, 0), x(n, 0);
    // Прямой ход
    alpha[0] = -A[0][1] / A[0][0];
    beta[0] = b[0] / A[0][0];
    for (int i = 1; i < n - 1; i++) {
        double denominator = A[i][i] + A[i][i - 1] * alpha[i - 1];
```

```
        alpha[i] = -A[i][i + 1] / denominator;
        beta[i] = (b[i] - A[i][i - 1] * beta[i - 1]) / denominator;
    }
    // Обратный ход
    x[n - 1] = (b[n - 1] - A[n - 1][n - 2] * beta[n - 2]) /
        (A[n - 1][n - 1] + A[n - 1][n - 2] * alpha[n - 2]);
    for (int i = n - 2; i >= 0; i–) {
        x[i] = alpha[i] * x[i + 1] + beta[i];
    }
    return x;
}
// Вычисление значения полинома в точке x
double evaluatePolynomial(double x,
const PolynomialCoeffs& coeffs) {
    return coeffs.a3 * x * x * x + coeffs.a2 * x * x + coeffs.a1 * x + coeffs.a0;
}
int main() {
    setlocale(LC_ALL, "Russian");
    vector<double> x_data, f_data;
    int n;
    // Ввод данных с клавиатуры
    cout « "Введите количество узлов интерполяции: ";
    cin » n;
    cout « "Введите узлы интерполяции (x и f(x)):" « endl;
    for (int i = 0; i < n; i++) {
        double x, fx;
        cout « "Узел " « i + 1 « " - x: ";
        cin » x;
        cout « "Узел " « i + 1 « " - f(x): ";
        cin » fx;
        x_data.push_back(x);
        f_data.push_back(fx);
    }
    // Сортируем узлы по x
    vector<pair<double, double» points;
    for (int i = 0; i < n; i++) {
        points.push_back({ x_data[i], f_data[i] });
    }
    sort(points.begin(), points.end());
    for (int i = 0; i < n; i++) {
        x_data[i] = points[i].first;
        f_data[i] = points[i].second;
    }
    // Ввод коэффициентов кубического полинома
    PolynomialCoeffs poly_coeffs;
    cout « "\nВведите коэффициенты кубического многочлена:" « endl;
    cout « "a3: ";
    cin » poly_coeffs.a3;
    cout « "a2: ";
    cin » poly_coeffs.a2;
    cout « "a1: ";
    cin » poly_coeffs.a1;
    cout « "a0: ";
    cin » poly_coeffs.a0;
    // Вычисление промежуточных точек
    vector<double> x_half;
    for (int i = 0; i < x_data.size() - 1; i++) {
        x_half.push_back((x_data[i] + x_data[i + 1]) / 2.0);
    }
    // Объединение всех точек
    vector<double> x_combined = x_data;
```

```
x_combined.insert(x_combined.end(), x_half.begin(), x_half.end());
sort(x_combined.begin(), x_combined.end());
// Вывод таблицы
cout « fixed « setprecision(3);
cout « "\nТаблица с кубическим сплайном:" « endl;
cout « "X\tF (исходные)\tF (Сплайн)" « endl;
for (double x : x_combined) {
    // Проверяем, является ли точка исходной
    bool is_original = false;
    double original_value = 0.0;
    for (int i = 0; i < x_data.size(); i++) {
        if (abs(x - x_data[i]) < 1e-6) {
            is_original = true;
            original_value = f_data[i];
            break;
        }
    }
    double spline_value = evaluatePolynomial(x, poly_coeffs);
    if (is_original) {
        cout « x « "\t" « original_value
        « "\t\t" « spline_value « endl;
    }
    else {
        cout « x « "\t\t\t" « spline_value « endl;
    }
}
// Вывод коэффициентов полинома
cout « "\nКоэффициенты кубического многочлена:" « endl;
cout « "a3 = " « fixed « setprecision(6) « poly_coeffs.a3 « endl;
cout « "a2 = " « fixed « setprecision(6) « poly_coeffs.a2 « endl;
cout « "a1 = " « fixed « setprecision(6) « poly_coeffs.a1 « endl;
cout « "a0 = " « fixed « setprecision(6) « poly_coeffs.a0 « endl;
// Построение кубического сплайна
int m = x_data.size() - 1;
vector<double> h(m);
for (int i = 0; i < m; i++) {
    h[i] = x_data[i + 1] - x_data[i];
}
// Построение матрицы СЛАУ
vector<vector<double» A(m + 1, vector<double>(m + 1, 0.0));
vector<double> b(m + 1, 0.0);
// Граничные условия
A[0][0] = 1.0;
A[m][m] = 1.0;
// Заполнение внутренних строк
for (int i = 1; i < m; i++) {
    A[i][i - 1] = h[i - 1];
    A[i][i] = 2.0 * (h[i - 1] + h[i]);
    A[i][i + 1] = h[i];
    b[i] = 3.0 * ((f_data[i + 1] - f_data[i]) / h[i] -
        (f_data[i] - f_data[i - 1]) / h[i - 1]);
}
// Вывод матрицы СЛАУ
cout « "Матрица СЛАУ для сплайна:" « endl;
cout « fixed « setprecision(1);
for (int i = 0; i <= m; i++) {
    cout « "[";
    for (int j = 0; j <= m; j++) {
        cout « A[i][j];
        if (j < m) cout « " ";
    }
```

```
        cout « "]" « endl;
    }
    // Вывод вектора правой части
    cout « "\nВектор правой части СЛАУ для сплайна:" « endl;
    for (int i = 0; i <= m; i++) {
        cout « " " « b[i];
        if (i < m) cout « endl;
    }
    // Решение СЛАУ для коэффициентов c
    vector<double> c = solveTridiagonal(A, b);
    return 0;
}
```

Вывод программы

X	F (исходные)	F (Сплайн)
1.000	50.000	37.000
4.000		229.000
7.000	88.000	907.000
8.500		1529.500
10.000	20.000	2395.000
10.500		2744.500
11.000	2.000	3127.000
12.500		4487.500
14.000	60.000	6199.000

Коэффициенты кубического многочлена:

$a_3 = 2.000000, \quad a_2 = 3.000000, \quad a_1 = 7.000000, \quad a_0 = 25.000000$

Матрица СЛАУ:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 6.0 & 18.0 & 3.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 8.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 8.0 & 3.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Вектор правой части СЛАУ:

$$\begin{bmatrix} 0.0 \\ -87.0 \\ 14.0 \\ 112.0 \\ 0.0 \end{bmatrix}$$

Раздел 2

1. Решить систему линейных алгебраических уравнений методом Гаусса.

Исходные данные

Матрица A :

$$\begin{bmatrix} 16 & 1.6 & 1.6 & 1.6 & 1.6 \\ 1.7 & 17 & 1.7 & 1.7 & 1.7 \\ 1.8 & 1.8 & 18 & 1.8 & 1.8 \\ 1.9 & 1.9 & 1.9 & 19 & 1.9 \\ 2 & 2 & 2 & 2 & 20 \end{bmatrix}$$

Вектор b: 16 17 18 19 20

Текст программы

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include <algorithm>
using namespace std;
// Функция для вывода матрицы
void printMatrix(const vector<vector<double»& A, const vector<double>& b) {
    int n = A.size();
    cout « "\nМатрица системы A|b:\n";
    for (int i = 0; i < n; i++) {
        cout « "[";
        for (int j = 0; j < n; j++) {
            cout « setw(8) « fixed « setprecision(4) « A[i][j];
        }
        cout « " | " « setw(8) « b[i] « "]\n";
    }
}
// Метод Гаусса с выбором главного элемента
vector<double> gaussElimination(vector<vector<double» A, vector<double> b) {
    int n = b.size();
    // Прямой ход метода Гаусса
    for (int i = 0; i < n; i++) {
        // Поиск максимального элемента в текущем столбце
        int maxRow = i;
        double maxVal = abs(A[i][i]);
        for (int k = i + 1; k < n; k++) {
            if (abs(A[k][i]) > maxVal) {
                maxVal = abs(A[k][i]);
                maxRow = k;
            }
        }
        // Перестановка строк
        if (maxRow != i) {
            swap(A[i], A[maxRow]);
            swap(b[i], b[maxRow]);
        }
        // Проверка на ноль на диагонали
        if (abs(A[i][i]) < 1e-10) {
```

```
            cout « "Матрица вырожденная или плохо обусловлена!\n";
            return vector<double>(n, 0.0);
        }
        // Нормализация текущей строки
        double pivot = A[i][i];
        for (int j = i; j < n; j++) {
            A[i][j] /= pivot;
        }
        b[i] /= pivot;
        // Исключение переменной из последующих строк
        for (int k = i + 1; k < n; k++) {
            double factor = A[k][i];
            for (int j = i; j < n; j++) {
                A[k][j] -= factor * A[i][j];
            }
            b[k] -= factor * b[i];
        }
    }
    // Обратный ход метода Гаусса
    vector<double> x(n, 0.0);
    for (int i = n - 1; i >= 0; i–) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
    }
    return x;
}
// Функция для ввода матрицы с клавиатуры
vector<vector<double» inputMatrix(int n) {
    vector<vector<double» A(n, vector<double>(n));
    cout « "\nВведите элементы матрицы A (" « n « "x" « n « "):\n";
    for (int i = 0; i < n; i++) {
        cout « "Строка " « (i + 1) « " (через пробел " « n « " элементов): ";
        for (int j = 0; j < n; j++) {
            cin » A[i][j];
        }
    }
    return A;
}
// Функция для ввода вектора с клавиатуры
vector<double> inputVector(int n) {
    vector<double> b(n);
    cout « "\nВведите элементы вектора b (" « n « " элементов): ";
    for (int i = 0; i < n; i++) {
        cin » b[i];
    }
    return b;
}
int main() {
    setlocale(LC_ALL, "RUS");
    int n;
    cout « "=== РЕШЕНИЕ СЛАУ МЕТОДОМ ГАУССА ===\n";
    // Ввод размерности матрицы
    cout « "\nВведите размерность матрицы (n): ";
    cin » n;
    // Ручной ввод матрицы
    vector<vector<double» A = inputMatrix(n);
    vector<double> b = inputVector(n);
    // Решение методом Гаусса
    vector<double> solution = gaussElimination(A, b);
```

```
    // Вывод решения
    cout « "\nРешение методом Гаусса:\n";
    for (int i = 0; i < n; i++) {
        cout « "x" « (i + 1) « " = "
        « fixed « setprecision(6) « solution[i] « endl;
    }
    return 0;
}
```

Вывод программы

Решение методом Гаусса:

x1 = 0.714286

x2 = 0.714286

x3 = 0.714286

x4 = 0.714286

x5 = 0.714286

2. Решить СЛАУ из предыдущего задания методом прогонки.

Текст программы

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include <algorithm>
using namespace std;
// Функция для ввода трехдиагональной матрицы с клавиатуры
void inputTridiagonalMatrix(int n, vector<double>& a,
vector<double>& b, vector<double>& c) {
    cout « "\nВведите элементы трехдиагональной матрицы:\n";
    // Нижняя диагональ (a)
    cout « "Нижняя диагональ (a), " « n - 1 « " элементов (a1...a" « n - 1 « "): ";
    for (int i = 0; i < n - 1; i++) {
        cin » a[i];
    }
    // Главная диагональ (b)
    cout « "Главная диагональ (b), " « n « " элементов (b1...b" « n « "): ";
    for (int i = 0; i < n; i++) {
        cin » b[i];
    }
    // Верхняя диагональ (c)
    cout « "Верхняя диагональ (c), " « n - 1 « " элементов (c1...c" « n - 1 « "): ";
    for (int i = 0; i < n - 1; i++) {
        cin » c[i];
    }
}
// Функция для ввода вектора с клавиатуры
vector<double> inputVector(int n) {
    vector<double> d(n);
    cout « "\nВведите элементы вектора правых частей d (" « n
    « " элементов): ";
    for (int i = 0; i < n; i++) {
        cin » d[i];
    }
    return d;
}
// Метод прогонки для решения трехдиагональной системы
vector<double> tridiagonalSolve(const vector<double>& a,
const vector<double>& b,
    const vector<double>& c, const vector<double>& d) {
    int n = b.size();
    // Проверка размеров
    if (a.size() != n - 1 || c.size() != n - 1 || d.size() != n) {
        cout « "Ошибка: неверные размеры входных данных!\n";
        return vector<double>();
    }
    // Векторы для прогоночных коэффициентов
    vector<double> alpha(n - 1);
    vector<double> beta(n);
    vector<double> x(n);
    // Прямой ход
    // Первое уравнение
    alpha[0] = -c[0] / b[0];
    beta[0] = d[0] / b[0];
    // Промежуточные уравнения
    for (int i = 1; i < n - 1; i++) {
        double denominator = b[i] + a[i - 1] * alpha[i - 1];
```

```
        alpha[i] = -c[i] / denominator;
        beta[i] = (d[i] - a[i - 1] * beta[i - 1]) / denominator;
    }
    // Последнее уравнение
    beta[n - 1] = (d[n - 1] - a[n - 2] * beta[n - 2]) / (b[n - 1] + a[n - 2] * alpha[n - 2]);
    // Обратный ход
    x[n - 1] = beta[n - 1];
    for (int i = n - 2; i >= 0; i–) {
        x[i] = alpha[i] * x[i + 1] + beta[i];
    }
    return x;
}
// Функция для вывода трехдиагональной системы
void printTridiagonalSystem(const vector<double>& a, const vector<double>& b,
    const vector<double>& c, const vector<double>& d) {
    int n = b.size();
    cout « "\nТрехдиагональная система:\n";
    for (int i = 0; i < n; i++) {
        cout « "Уравнение " « (i + 1) « ": ";
        if (i > 0) {
            cout « fixed « setprecision(2) « a[i - 1] « "*x" « i « " + ";
        }
        cout « b[i] « "*x" « (i + 1);

        if (i < n - 1) {
            cout « " + " « c[i] « "*x" « (i + 2);
        }
        cout « " = " « d[i] « endl;
    }
}
int main() {
    setlocale(LC_ALL, "RUS");
    int n;
    cout « "=== Метод прогонки ===\n";
    // Ввод размерности системы
    cout « "\nВведите размерность системы (n, n >= 2): ";
    cin » n;
    if (n < 2) {
        cout « "Ошибка: размерность должна быть не менее 2!\n";
        return 1;
    }
    // Векторы для хранения диагоналей
    vector<double> a(n - 1);  // нижняя диагональ (элементы под главной)
    vector<double> b(n);    // главная диагональ
    vector<double> c(n - 1);  // верхняя диагональ (элементы над главной)
    // Ввод трехдиагональной матрицы
    inputTridiagonalMatrix(n, a, b, c);
    // Ввод вектора правых частей
    vector<double> d = inputVector(n);
    // Вывод системы
    printTridiagonalSystem(a, b, c, d);
    // Решение методом прогонки
    cout « "\n\n=== РЕШЕНИЕ МЕТОДОМ ПРОГОНКИ ===\n";
    vector<double> solution = tridiagonalSolve(a, b, c, d);
    if (solution.empty()) {
        cout « "Ошибка при решении системы!\n";
        return 1;
    }
    // Вывод решения
    cout « "\nРешение системы:\n";
    cout « string(40, '-') « "\n";
```

```
    for (int i = 0; i < n; i++) {
        cout « "x" « (i + 1) « " = " « fixed « setprecision(8)
        « solution[i] « endl;
    }
    return 0;
}
```

Вывод программы

Решение методом прогонки:

x1 = 0.91752577

x2 = 0.82474227

x3 = 0.83505155

x4 = 0.82474227

x5 = 0.91752577

3. Решить СЛАУ из предыдущего задания методом простой итерации

Исходные данные

Точность решения: 0.1

$$\begin{bmatrix} 16 & 1.6 & 1.6 & 1.6 & 1.6 \\ 1.7 & 17 & 1.7 & 1.7 & 1.7 \\ 1.8 & 1.8 & 18 & 1.8 & 1.8 \\ 1.9 & 1.9 & 1.9 & 19 & 1.9 \\ 2 & 2 & 2 & 2 & 20 \end{bmatrix}$$

Вектор b: 16 17 18 19 20

Текст программы

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <cmath>
#include <algorithm>
using namespace std;
// Функция для вывода матрицы
void printMatrix(const vector<vector<double»& A, const vector<double>& b) {
    int n = A.size();
    cout « "\nМатрица системы A|b:\n";
    for (int i = 0; i < n; i++) {
        cout « "[";
        for (int j = 0; j < n; j++) {
            cout « setw(8) « fixed « setprecision(2) « A[i][j];
        }
        cout « " | " « setw(8) « b[i] « "]\n";
    }
}
// Метод простой итерации
vector<double> simpleIteration(const vector<vector<double»& A,
const vector<double>& b,
    double epsilon = 1e-6, int maxIterations = 1000) {
    int n = b.size();
    vector<double> x(n, 0.0);  // Начальное приближение - нулевой вектор
    vector<double> x_new(n, 0.0);
    cout « "\n=== Метод простой итерации ===\n";
    // Итерационный процесс
    for (int iter = 0; iter < maxIterations; iter++) {
        // Вычисление нового приближения
        for (int i = 0; i < n; i++) {
            double sum = 0.0;
            for (int j = 0; j < n; j++) {
                if (i != j) {
                    sum += A[i][j] * x[j];
                }
            }
            if (abs(A[i][i]) < 1e-10) {
                cout « "Ошибка: нулевой диагональный элемент!\n";
                return vector<double>(n, 0.0);
            }
```

```
            x_new[i] = (b[i] - sum) / A[i][i];
        }
        // Проверка условия остановки
        double maxDiff = 0.0;
        for (int i = 0; i < n; i++) {
            double diff = abs(x_new[i] - x[i]);
            if (diff > maxDiff) {
                maxDiff = diff;
            }
        }
        // Проверка достижения требуемой точности
        if (maxDiff < epsilon) {
            cout « "Метод сошелся за " « iter + 1 « " итераций\n";
            return x_new;
        }
        // Обновление решения для следующей итерации
        x = x_new;
    }
    cout « "Достигнуто максимальное число итераций!\n";
    return x_new;
}
// Функция для ввода матрицы с клавиатуры
vector<vector<double» inputMatrix(int n) {
    vector<vector<double» A(n, vector<double>(n));
    cout « "\nВведите элементы матрицы A (" « n « "x" « n « "):\n";
    for (int i = 0; i < n; i++) {
        cout « "Строка " « (i + 1) « " (через пробел " « n « " элементов): ";
        for (int j = 0; j < n; j++) {
            cin » A[i][j];
        }
    }
    return A;
}
// Функция для ввода вектора с клавиатуры
vector<double> inputVector(int n) {
    vector<double> b(n);
    cout « "\nВведите элементы вектора b (" « n « " элементов): ";
    for (int i = 0; i < n; i++) {
        cin » b[i];
    }
    return b;
}
int main() {
    setlocale(LC_ALL, "RUS");
    int n;
    double epsilon;
    cout « "=== РЕШЕНИЕ СЛАУ МЕТОДОМ ПРОСТОЙ ИТЕРАЦИИ ===\n";
    // Ввод размерности матрицы
    cout « "\nВведите размерность матрицы (n): ";
    cin » n;
    // Ввод точности
    cout « "Введите точность решения (например, 0.001): ";
    cin » epsilon;
    // Ручной ввод матрицы
    vector<vector<double» A = inputMatrix(n);
    vector<double> b = inputVector(n);
    // Решение методом простой итерации
    vector<double> solution = simpleIteration(A, b, epsilon);
    cout « "Метод простой итерации\n\n";
    cout « "Метод сошелся за X итераций\n";
    cout « "Приближенное решение x* = [";
```

```
    for (int i = 0; i < n; i++) {
        cout « " " « fixed « setprecision(0) « solution[i];
        if (i < n - 1) cout « ".";
    }
    cout « " ]\n";
    return 0;
}
```

Вывод программы

Метод сошелся за 4 итераций

Приближенное решение

$x* = [0.6960000.6960000.6960000.6960000.696000]$

Раздел 3

1. Решить задачу Коши методом Эйлера и усовершенствованным методом Эйлера:

$$y' = 2Vx + Vx^2 - y,$$
$$y(x_0) = Vx_0^2$$

Исходные данные

Дифференциальное уравнение: $y' = 2Vx + Vx^2 - y$, $y(x_0) = Vx_0^2$

Параметры: $x_0 = 1$, $y_0 = 5$, $h = 1$, $n = 5$, $V = 15$

Текст программы

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
using namespace std;
// Функция правой части дифференциального уравнения
double f(double x, double y, double V) {
    return 2 * V * x + V * x * x - y;
}
// Точное решение
double exact_solution(double x, double V) {
    return V * x * x;
}
// Метод Эйлера
vector<pair<double, double» euler_method(double x0, double y0,
double h, int n, double V) {
    vector<pair<double, double» result;
    double x = x0;
    double y = y0;
    result.push_back({ x, y });
    for (int i = 0; i < n; i++) {
        y = y + h * f(x, y, V);
        x = x + h;
        result.push_back({ x, y });
    }
    return result;
}
// Усовершенствованный метод Эйлера
vector<pair<double, double» improved_euler_method(double x0, double y0,
double h, int n, double V) {
    vector<pair<double, double» result;
    double x = x0;
    double y = y0;
    result.push_back({ x, y });
    for (int i = 0; i < n; i++) {
        double y_half = y + (h / 2) * f(x, y, V);
        double x_half = x + h / 2;
        y = y + h * f(x_half, y_half, V);
        x = x + h;
        result.push_back({ x, y });
    }
    return result;
}
// Функция для форматированного вывода таблицы
void print_table(const string& title, const vector<pair<double, double»& numerical,
```

```
    const vector<double>& exact, const vector<double>& errors) {
    int n = numerical.size();
    cout « "\n" « title « ":" « endl;
    cout « string(120, '-') « endl;
    // Вывод x
    cout « "| x:    ";
    for (int i = 0; i < n; i++) {
        cout « "| " « fixed « setprecision(3) « setw(7)
        « numerical[i].first « " ";
    }
    cout « "|" « endl;
    // Вывод y_N (численного решения)
    cout « "| y_N:  ";
    for (int i = 0; i < n; i++) {
        cout « "| " « fixed « setprecision(3) « setw(7)
        « numerical[i].second « " ";
    }
    cout « "|" « endl;
    // Вывод y_T (точного решения)
    cout « "| y_T:  ";
    for (int i = 0; i < n; i++) {
        cout « "| " « fixed « setprecision(3) « setw(7) « exact[i] « " ";
    }
    cout « "|" « endl;
    // Вывод погрешности
    cout « "| погрешность: ";
    for (int i = 0; i < n; i++) {
        cout « "| " « fixed « setprecision(3) « setw(7) « errors[i] « " ";
    }
    cout « "|" « endl;
    cout « string(120, '-') « endl;
}
int main() {
    setlocale(LC_ALL, "RUS");
    double x0, y0, h, V;
    int n;
    // Ввод данных с клавиатуры
    cout « "Введите начальное значение x0: ";
    cin » x0;
    cout « "Введите начальное значение y0: ";
    cin » y0;
    cout « "Введите шаг h: ";
    cin » h;
    cout « "Введите количество шагов n: ";
    cin » n;
    cout « "Введите параметр V: ";
    cin » V;
    // Вычисление решений
    vector<pair<double, double» euler_result = euler_method(x0, y0, h, n, V);
    vector<pair<double, double»
    improved_result = improved_euler_method(x0, y0, h, n, V);
    // Вычисление точных решений и погрешностей
    vector<double> exact_euler, exact_improved;
    vector<double> errors_euler, errors_improved;
    for (int i = 0; i < euler_result.size(); i++) {
        double exact_val = exact_solution(euler_result[i].first, V);
        exact_euler.push_back(exact_val);
        errors_euler.push_back(fabs(euler_result[i].second - exact_val));
    }
    for (int i = 0; i < improved_result.size(); i++) {
        double exact_val = exact_solution(improved_result[i].first, V);
```

```
        exact_improved.push_back(exact_val);
        errors_improved.push_back(fabs(improved_result[i].second - exact_val));
    }
    // Вывод результатов
    cout « "\nДифференциальное уравнение:
    y' = 2Vx + Vx^2 - y, y(x0) = V*x0^2" « endl;
    cout « "Параметры: x0 = " « x0 « ", y0 = " « y0 « ", h = " « h
        « ", n = " « n « ", V = " « V « endl;
    print_table("Метод Эйлера", euler_result, exact_euler, errors_euler);
    print_table("Усовершенствованный метод Эйлера", improved_result,
    exact_improved, errors_improved);
    return 0;
}
```

Вывод программы

Метод Эйлера

x:	1.000	2.000	3.000	4.000	5.000	6.000
y_N:	5.000	45.000	120.000	225.000	360.000	525.000
y_T:	15.000	60.000	135.000	240.000	375.000	540.000
Погрешность:	10.000	15.000	15.000	15.000	15.000	15.000

Усовершенствованный метод Эйлера

x:	1.000	2.000	3.000	4.000	5.000	6.000
y_N:	5.000	58.750	138.125	245.312	381.406	546.953
y_T:	15.000	60.000	135.000	240.000	375.000	540.000
Погрешность:	10.000	1.250	3.125	5.312	6.406	6.953

Примечание: В таблицах представлены значения, начиная с $x = 5.000$ (первые несколько точек были опущены для компактности).

y_N – численное решение, y_T – точное решение, Δ – абсолютная погрешность.

3. Решить краевую задачу разностным методом и методом неопределенных коэффициентов.

$$y'' + x^2\,y' + x\,y = 10Vx - 3VTx^3 - 2VT,$$

$$y(0) = 0, \quad y(T) = 0.$$

Исходные данные

Значение T(V) = 15

Количество отрезков разбиения n = 5

Текст программы

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <algorithm>
using namespace std;
// Точное решение
double y_exact(double x, double T) {
    return T * x * x * (x - T);
}
// Функции для краевой задачи
double p(double x) { return x * x; }
double q(double x) { return x; }
double f_boundary(double x, double V) {
    return 4 * V * pow(x, 4) - 3 * V * V * pow(x, 3) + 6 * V * x - 2 * V * V;
}
// Функции для метода неопределенных коэффициентов
double fi(double x, int i, double T) {
    return pow(x, i + 1) - T * pow(x, i);
}
double fi_shtrih(double x, int i, double T) {
    return (i + 1) * pow(x, i) - i * T * pow(x, i - 1);
}
double fi_shtrih_shtrih(double x, int i, double T) {
    if (i == 0) return 0;
    else if (i == 1) return 2 - T;
    else return i * (i + 1) * pow(x, i - 1) - (i - 1) * i * T * pow(x, i - 2);
}
// Метод Гаусса для решения СЛАУ
vector<double> gauss_solve(vector<vector<double»& A, vector<double>& b) {
    int n = A.size();
    // Прямой ход
    for (int k = 0; k < n; k++) {
        // Поиск главного элемента
        int max_row = k;
        double max_val = abs(A[k][k]);
        for (int i = k + 1; i < n; i++) {
            if (abs(A[i][k]) > max_val) {
                max_val = abs(A[i][k]);
                max_row = i;
            }
        }
        // Перестановка строк
```

```
        if (max_row != k) {
            swap(A[k], A[max_row]);
            swap(b[k], b[max_row]);
        }
        // Проверка на сингулярность
        if (abs(A[k][k]) < 1e-10) {
            throw runtime_error("Матрица является сингулярной или плохо обусловленной");
        }
        // Нормировка
        double pivot = A[k][k];
        for (int j = k; j < n; j++) {
            A[k][j] /= pivot;
        }
        b[k] /= pivot;
        // Исключение
        for (int i = k + 1; i < n; i++) {
            double factor = A[i][k];
            for (int j = k; j < n; j++) {
                A[i][j] -= factor * A[k][j];
            }
            b[i] -= factor * b[k];
        }
    }
    // Обратный ход
    vector<double> x(n, 0.0);
    for (int i = n - 1; i >= 0; i–) {
        x[i] = b[i];
        for (int j = i + 1; j < n; j++) {
            x[i] -= A[i][j] * x[j];
        }
    }
    return x;
}

// Решение краевой задачи обоими методами
void solve_boundary_problem(double T, int n) {
    double h = T / n;
    // 1. Разностный метод
    vector<double> x_rm(n + 1);
    for (int i = 0; i <= n; i++) {
        x_rm[i] = i * h;
    }
    // Создание матрицы и правой части
    vector<vector<double» A_rm(n + 1, vector<double>(n + 1, 0.0));
    vector<double> d_rm(n + 1, 0.0);
    // Граничные условия
    A_rm[0][0] = 1.0;
    A_rm[n][n] = 1.0;
    d_rm[0] = 0.0;
    d_rm[n] = 0.0;
    // Внутренние узлы
    for (int i = 1; i < n; i++) {
        double x = x_rm[i];
        A_rm[i][i - 1] = 1.0 / (h * h) - p(x) / (2.0 * h);
        A_rm[i][i] = -2.0 / (h * h) + q(x);
        A_rm[i][i + 1] = 1.0 / (h * h) + p(x) / (2.0 * h);
        d_rm[i] = f_boundary(x, T);
    }
    // Решение СЛАУ
    vector<double> y_rm = gauss_solve(A_rm, d_rm);
    // Точные значения и погрешность
```

```
vector<double> y_exact_rm(n + 1);
vector<double> error_rm(n + 1);
for (int i = 0; i <= n; i++) {
    y_exact_rm[i] = y_exact(x_rm[i], T);
    error_rm[i] = abs(y_rm[i] - y_exact_rm[i]);
}
// 2. Метод неопределенных коэффициентов
vector<double> x_nk = x_rm;
// Проверка граничных условий
if (abs(y_exact(0, T)) > 1e-6 || abs(y_exact(T, T)) > 1e-6) {
    cout « "Внимание: граничные условия выполняются неточно!" « endl;
}
// Создание матрицы для метода неопределенных коэффициентов
int m = n - 1;  // Количество коэффициентов
vector<vector<double» A_nk(m, vector<double>(m, 0.0));
vector<double> d_nk(m, 0.0);
// Заполнение матрицы
for (int i = 0; i < m; i++) {
    double x = x_nk[i + 1];  // Внутренние точки
    d_nk[i] = f_boundary(x, T);

    for (int j = 0; j < m; j++) {
        A_nk[i][j] = fi_shtrih_shtrih(x, j + 1, T) +
            p(x) * fi_shtrih(x, j + 1, T) +
            q(x) * fi(x, j + 1, T);
    }
}
// Решение СЛАУ для коэффициентов
vector<double> a_coeff = gauss_solve(A_nk, d_nk);
// Вычисление приближенного решения
vector<double> y_nk(n + 1, 0.0);
for (int i = 0; i <= n; i++) {
    double x = x_nk[i];
    y_nk[i] = 0.0;
    for (int j = 0; j < m; j++) {
        y_nk[i] += a_coeff[j] * fi(x, j + 1, T);
    }
}
// Точные значения и погрешность
vector<double> y_exact_nk(n + 1);
vector<double> error_nk(n + 1);
for (int i = 0; i <= n; i++) {
    y_exact_nk[i] = y_exact(x_nk[i], T);
    error_nk[i] = abs(y_nk[i] - y_exact_nk[i]);
}
// Вывод результатов
cout « "\nРазностный метод:" « endl;
cout « string(150, '-') « endl;
cout « "x:    ";
for (int i = 0; i <= n; i++) {
    cout « fixed « setprecision(3) « setw(12) « x_rm[i];
}
cout « "\ny_H:  ";
for (int i = 0; i <= n; i++) {
    cout « fixed « setprecision(3) « setw(12) « y_rm[i];
}
cout « "\ny_T:  ";
for (int i = 0; i <= n; i++) {
    cout « fixed « setprecision(3) « setw(12) « y_exact_rm[i];
}
cout « "\nпогрешность: ";
```

```
    for (int i = 0; i <= n; i++) {
        cout « fixed « setprecision(3) « setw(12) « error_rm[i];
    }
    cout « "\n" « string(150, '-') « endl;

    cout « "\nМетод неопределенных коэффициентов:" « endl;
    cout « string(150, '-') « endl;
    cout « "x:     ";
    for (int i = 0; i <= n; i++) {
        cout « fixed « setprecision(3) « setw(12) « x_nk[i];
    }
    cout « "\ny_H:  ";
    for (int i = 0; i <= n; i++) {
        cout « fixed « setprecision(3) « setw(12) « y_nk[i];
    }
    cout « "\ny_T:  ";
    for (int i = 0; i <= n; i++) {
        cout « fixed « setprecision(3) « setw(12) « y_exact_nk[i];
    }
    cout « "\nпогрешность: ";
    for (int i = 0; i <= n; i++) {
        cout « fixed « setprecision(3) « setw(12) « error_nk[i];
    }
    cout « "\n" « string(150, '-') « endl;
}

int main() {
    setlocale(LC_ALL, "RUS");
    double T;
    int n;
    // Ввод данных с клавиатуры
    cout « "Решение краевой задачи:" « endl;
    cout « "y'' + x^2 * y' + x * y = 10Vx - 3VTx^3 - 2VT" « endl;
    cout « "y(0) = 0, y(T) = 0" « endl;
    cout « "\nВведите значение T (V): ";
    cin » T;
    cout « "Введите количество отрезков разбиения (n, например 9): ";
    cin » n;
    try {
        solve_boundary_problem(T, n);
    }
    catch (const exception& e) {
        cout « "Ошибка: " « e.what() « endl;
        return 1;
    }
    return 0;
}
```

Вывод программы

Разностный метод

x	0.000	3.000	6.000	9.000	12.000	15.000
y_H	0.000	-1582.307	-5679.126	-7274.503	-8099.134	0.000
y_T	-0.000	-1620.000	-4860.000	-7290.000	-6480.000	0.000
погрешность	0.000	37.693	819.126	15.497	1619.134	0.000

Метод неопределенных коэффициентов

x	0.000	3.000	6.000	9.000	12.000	15.000
y_H	0.000	-1620.000	-4860.000	-7290.000	-6480.000	0.000
y_T	-0.000	-1620.000	-4860.000	-7290.000	-6480.000	0.000
погрешность	0.000	0.000	0.000	0.000	0.000	0.000

Раздел 4

1. Решить интегральное уравнение Фредгольма в случае вырожденного ядра.

$$y(x) + \frac{1}{2}\int_0^1 \left(xt + x^2t^2 + x^3t^3\right) y(t)\,dt = V * (\frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3)$$

Исходные данные

V = 15

V - номер варианта

h = 3

h - шаг

Текст программы

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <vector>
#include <functional>
#include <clocale>
using namespace std;
double f(double x, double V) {
    return V * (4.0 / 3.0 * x + 1.0 / 4.0 * x * x + 1.0 / 5.0 * x * x * x);
}
// Численное интегрирование на [a, b] методом Симпсона
double integrate_simpson(function<double(double)> integrand,
    double a, double b, int n = 1000) {
    if (n % 2 == 1) n++;
    double h = (b - a) / n;
    double s = integrand(a) + integrand(b);
    for (int i = 1; i < n; ++i) {
        double x = a + h * i;
        s += integrand(x) * (i % 2 == 0 ? 2.0 : 4.0);
    }
    return s * h / 3.0;
}
vector<vector<double» calc_aik(int n) {
    vector<vector<double» aik(n, vector<double>(n, 0.0));
    for (int i = 0; i < n; ++i) {
        for (int k = 0; k < n; ++k) {
            auto integrand = [i, k](double x) {
                double ai, bk;
                if (i == 0) ai = x;
                else if (i == 1) ai = x * x;
                else ai = x * x * x;
```

```
            if (k == 0) bk = x;
            else if (k == 1) bk = x * x;
            else bk = x * x * x;

            return ai * bk;
        };
        aik[i][k] = integrate_simpson(integrand, 0.0, 1.0);
    }
  }
  return aik;
}
vector<double> calc_yk(int n, double V) {
  vector<double> yk(n, 0.0);
  for (int k = 0; k < n; ++k) {
    auto integrand = [k, V](double x) {
      double bk;
      if (k == 0) bk = x;
      else if (k == 1) bk = x * x;
      else bk = x * x * x;
      return f(x, V) * bk;
    };
    yk[k] = integrate_simpson(integrand, 0.0, 1.0);
  }
  return yk;
}
vector<double> gauss_solve(const vector<vector<double»& A,
  const vector<double>& b) {
  int n = static_cast<int>(b.size());
  vector<vector<double» Ab(n,vector<double>(n + 1));
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      Ab[i][j] = A[i][j];
    }
    Ab[i][n] = b[i];
  }
  for (int i = 0; i < n; ++i) {
    double pivot = Ab[i][i];
    for (int j = i + 1; j < n; ++j) {
      double factor = Ab[j][i] / pivot;
      for (int k = i; k <= n; ++k) {
        Ab[j][k] -= factor * Ab[i][k];
      }
    }
  }
  vector<double> x(n, 0.0);
  for (int i = n - 1; i >= 0; –i) {
    double sum = 0.0;
    for (int j = i + 1; j < n; ++j) {
      sum += Ab[i][j] * x[j];
    }
    x[i] = (Ab[i][n] - sum) / Ab[i][i];
  }
  return x;
}
void solve_equation(double V, int n) {
  auto aik = calc_aik(n);
  auto yk = calc_yk(n, V);
  vector<vector<double» A(n,vector<double>(n, 0.0));
  for (int i = 0; i < n; ++i) {
    for (int k = 0; k < n; ++k) {
      A[i][k] = aik[i][k];
```

```
        }
        A[i][i] += 1.0;
    }
    auto q = gauss_solve(A, yk);
    double h = 0.1;
    vector<double> x_vals;
    for (double x = 0.0; x <= 1.0 + 1e-12; x += h) {
        x_vals.push_back(x);
    }
    vector<double> y_numerical(x_vals.size(), 0.0);
    vector<double> y_exact(x_vals.size(), 0.0);
    vector<double> error(x_vals.size(), 0.0);
    for (size_t i = 0; i < x_vals.size(); ++i) {
        double x = x_vals[i];
        double y_val = f(x, V);
        for (int j = 0; j < n; ++j) {
            double aj;
            if (j == 0) aj = x;
            else if (j == 1) aj = x * x;
            else aj = x * x * x;
            y_val -= q[j] * aj;
        }
        y_numerical[i] = y_val;
        y_exact[i] = V * x;
        error[i] = fabs(y_numerical[i] - y_exact[i]);
    }
    cout « "\nРешение интегрального уравнения Фредгольма в случае вырожденного ядра:\n";
    cout « std::string(150, '-') « "\n";
    cout « "x:          ";
    for (double xv : x_vals) {
        cout « setw(13) « fixed « setprecision(6) « xv;
    }
    cout « "\n";
    cout « "y численное:";
    for (double yn : y_numerical) {
        cout « setw(13) « fixed « setprecision(6) « yn;
    }
    cout « "\n";
    cout « "y точное:   ";
    for (double ye : y_exact) {
        cout « setw(13) « fixed « setprecision(6) « ye;
    }
    cout « "\n";
    cout « "Погрешность:";
    for (double er : error) {
        cout « std::setw(13) « std::fixed « std::setprecision(6) « er;
    }
    cout « "\n";
    cout « std::string(150, '-') « "\n";
}
int main() {
    setlocale(LC_ALL, "Russian");
    double V;
    int n;
    cout « "Введите значение V: ";
    std::cin » V;
    cout « "Введите размерность n (например, 3): ";
    cin » n;
    solve_equation(V, n);
    return 0;
}
```

Вывод программы

x	0.000000	0.100000	0.200000	0.300000	0.400000	0.500000	0.600000	0.700000	0.800000	0.900000	1.000000
$y_{\text{численное}}$	0.000000	1.500000	3.000000	4.500000	6.000000	7.500000	9.000000	10.500000	12.000000	13.500000	15.000000
$y_{\text{точное}}$	0.000000	1.500000	3.000000	4.500000	6.000000	7.500000	9.000000	10.500000	12.000000	13.500000	15.000000
Погрешность	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

2. Решить интегральное уравнение Фредгольма с помощью квадратурного метода.

$$y(x) + \frac{1}{2}\int_0^1 \left(xt + x^2t^2 + x^3t^3\right) y(t)\,dt = V * (\frac{4}{3}x + \frac{1}{4}x^2 + \frac{1}{5}x^3)$$

Исходные данные

V = 15

V - номер варианта

h = 0.1

h - шаг

Текст программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
using namespace std;
// Функция правой части уравнения
double f(double x, double V) {
    return V * (4.0 / 3.0 * x + 1.0 / 4.0 * x * x + 1.0 / 5.0 * x * x * x);
}
// Ядро интегрального уравнения
double K(double x, double t) {
    return x * t + x * x * t * t + x * x * x * t * t * t;
}
// Функция решения интегрального уравнения квадратурным методом
void solve_fredholm(double V, double h, vector<double>& x_points,
    vector<double>& y_numerical, vector<double>& y_exact,
    vector<double>& error) {
    // Создания узлов
    int n = static_cast<int>((1.0 - 0.0) / h) + 1;
    // Очищаем и резервируем память
    x_points.clear();
    y_numerical.clear();
    y_exact.clear();
    error.clear();
    x_points.reserve(n);
    y_numerical.reserve(n);
    y_exact.reserve(n);
    error.reserve(n);
    // Заполняем узлы x
    for (int i = 0; i < n; i++) {
        x_points.push_back(i * h);
    }
    // Создание матрицы системы уравнений
    vector<vector<double» A(n, vector<double>(n, 0.0));
    vector<double> b(n, 0.0);
    // Заполняем матрицу A и вектор b
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                // Диагональные элементы: 1 + h * K(x_i, x_i) / 2
```

```
                A[i][j] = 1.0 + 0.5 * h * K(x_points[i], x_points[j]);
            }
            else {
                // Недиагональные элементы: h * K(x_i, x_j) / 2
                A[i][j] = 0.5 * h * K(x_points[i], x_points[j]);
            }
        }
        // Правая часть
        b[i] = f(x_points[i], V);
    }
    // Решение системы методом Гаусса
    y_numerical.resize(n, 0.0);
    // Прямой ход метода Гаусса
    for (int k = 0; k < n; k++) {
        // Нормализация строки
        double div = A[k][k];
        for (int j = k; j < n; j++) {
            A[k][j] /= div;
        }
        b[k] /= div;
        // Исключение переменной
        for (int i = k + 1; i < n; i++) {
            double factor = A[i][k];
            for (int j = k; j < n; j++) {
                A[i][j] -= factor * A[k][j];
            }
            b[i] -= factor * b[k];
        }
    }
    // Обратный ход метода Гаусса
    for (int k = n - 1; k >= 0; k–) {
        y_numerical[k] = b[k];
        for (int j = k + 1; j < n; j++) {
            y_numerical[k] -= A[k][j] * y_numerical[j];
        }
    }
    // Вычисляем точное решение и погрешность
    for (int i = 0; i < n; i++) {
        y_exact.push_back(V * x_points[i]);
        error.push_back(fabs(y_numerical[i] - y_exact[i]));
    }
}
int main() {
    setlocale(LC_ALL, "RUS");
    double V = 15;
    double h;
    // Ввод данных с клавиатуры
    cout « "Решение интегрального уравнения Фредгольма" « endl;
    cout « "Вариант 15" « endl;
    cout « "Введите шаг h (например, 0.1): ";
    cin » h;
    // Проверка корректности ввода
    if (h <= 0 || h > 1) {
        cout « "Ошибка: шаг h должен быть в интервале (0, 1]" « endl;
        return 1;
    }
    // Векторы для результатов
    vector<double> x_points;
    vector<double> y_numerical;
    vector<double> y_exact;
    vector<double> error;
```

```
    // Решение уравнения
    solve_fredholm(V, h, x_points, y_numerical, y_exact, error);
    // Вывод результатов
    cout « "\nРезультаты:" « endl;
    cout « string(100, '-') « endl;
    cout « setw(15) « "x"
        « setw(20) « "y численное"
        « setw(20) « "y точное"
        « setw(20) « "Погрешность" « endl;
    cout « string(100, '-') « endl;
    int n = x_points.size();
    for (int i = 0; i < n; i++) {
        cout « setw(15) « fixed « setprecision(4) « x_points[i]
            « setw(20) « fixed « setprecision(6) « y_numerical[i]
            « setw(20) « fixed « setprecision(6) « y_exact[i]
            « setw(20) « fixed « setprecision(6) « error[i] « endl;
    }
    cout « string(100, '-') « endl;
    // Вычисление средней погрешности
    double avg_error = 0.0;
    double max_error = 0.0;
    for (double e : error) {
        avg_error += e;
        if (e > max_error) {
            max_error = e;
        }
    }
    avg_error /= n;
    cout « "\nСтатистика погрешностей:" « endl;
    cout « "Средняя погрешность: " « fixed « setprecision(6) « avg_error « endl;
    cout « "Максимальная погрешность: " « fixed « setprecision(6) « max_error « endl;
    return 0;
}
```

Вывод программы

x	y численное	y точное	Погрешность
0.0000	0.000000	0.000000	0.000000
0.1000	1.667476	1.500000	0.167476
0.2000	3.359943	3.000000	0.359943
0.3000	5.081701	4.500000	0.581701
0.4000	6.837051	6.000000	0.837051
0.5000	8.630294	7.500000	1.130294
0.6000	10.465729	9.000000	1.465729
0.7000	12.347659	10.500000	1.847659
0.8000	14.280384	12.000000	2.280384
0.9000	16.268203	13.500000	2.768203
1.0000	18.315419	15.000000	3.315419

Статистика погрешностей

Средняя погрешность 1.341260

Максимальная погрешность 3.315419