Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	«ИУ »
Кафедра	«ИУ6 »
Группа	«ИУ6-64Б »

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчет по домашнему заданию N3:

Аппроксимация функций

Вариант N 12

Студент: _____ Кошенков Д.О.
дата, подпись Ф.И.О.

Преподаватель: _____ Орлова А.С.
дата, подпись Ф.И.О.

Москва, 2025

Содержание

1 Введение

Цель домашней работы: изучение интерполяционного полинома Лагранжа, метода построение кубических сплайнов.

Для достижения цели необходимо решить следующие задачи:

1. Реализовать на C++ указанные методы аппроксимации функции.
2. Провести аппросимацию двух заданных функций указанными методами.
3. Построить графики полученных полиномов на заданной сетке.

Отчет должен содержать:

1. Краткое описание реализуемых методов.
2. Постановку задачи и исходные данные.
3. Описание реализации и соответствующие части листинга программы.
4. Результаты вычислений и графики.
5. Анализ полученных результатов.
6. Сравнение значения кубического сплайна и заданной функции в точках, не совпадающих со значениями в узлах интерполяции (для соответствующей части).

2 Аппроксимация функции. Интерполяция по Лагранжу

2.1 Теоретические сведения

Интерполяция — это способ нахождения промежуточных значений функции по имеющемуся дискретному набору известных значений. Полином Лагранжа является одним из методов полиномиальной интерполяции.

Пусть задана таблица значений функции $y_i = f(x_i)$ в $n+1$ различных точках $x_0, x_1, \ldots, x_n$. Требуется построить полином $L_n(x)$ степени не выше n, который проходит через все заданные точки, то есть $L_n(x_i) = y_i$ для всех $i = 0, 1, \ldots, n$.

Полином Лагранжа имеет вид:

$$L_n(x) = \sum_{i=0}^{n} y_i \cdot l_i(x)$$

где $l_i(x)$ — базисные полиномы Лагранжа, определяемые как:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \frac{(x - x_0) \ldots (x - x_{i-1})(x - x_{i+1}) \ldots (x - x_n)}{(x_i - x_0) \ldots (x_i - x_{i-1})(x_i - x_{i+1}) \ldots (x_i - x_n)}$$

2.2 Постановка задачи и исходные данные

Необходимо построить интерполяционный полином Лагранжа $L(x)$ для функции, заданной таблично. Исходные данные (Вариант 12) приведены в Таблице 1.

Таблица 1 – Табличные данные для интерполяции Лагранжа

x_i	0	1	2	3	4	5	6	7	8	9
y_i	9.0	8.2	7.4	6.6	5.8	4.9	4.1	3.3	2.5	1.8

2.3 Описание реализации

Интерполяция по Лагранжу реализована в функции ‘lagrange_interpolation‘. Эта функция принимает на вход векторы узлов ‘x_nodes‘, значений ‘y_nodes‘ и точку ‘xp‘, в которой необходимо вычислить значение полинома. Функция вычисляет базисные полиномы $l_i(x)$ и суммирует их с весами y_i согласно формуле Лагранжа. Включены проверки на совпадение размеров входных векторов и на наличие совпадающих узлов x_i. Полный код программы находится в Листинге 1 .

2.4 Результаты вычислений и график

Полином Лагранжа был построен для данных из Таблицы 1. В Таблице 2 приведены значения полинома $L(x_p)$, вычисленные программой в некоторых промежуточных точках x_p.

Таблица 2 – Значения полинома Лагранжа $L(x_p)$ в некоторых точках

x_p	$L(x_p)$
0.100000	8.803726
0.200000	8.665446
0.500000	8.441740
1.500000	7.836812
2.500000	6.982127
3.500000	6.217238
4.500000	5.350053
5.500000	4.482625
6.500000	3.718175
7.500000	2.861879
8.500000	2.277898
8.600000	2.232181
8.800000	2.090666

На Рисунке 1 представлен график построенного полинома Лагранжа $L(x)$ (синяя пунктирная линия), проходящего через исходные точки данных (x_i, y_i) (красные точки).

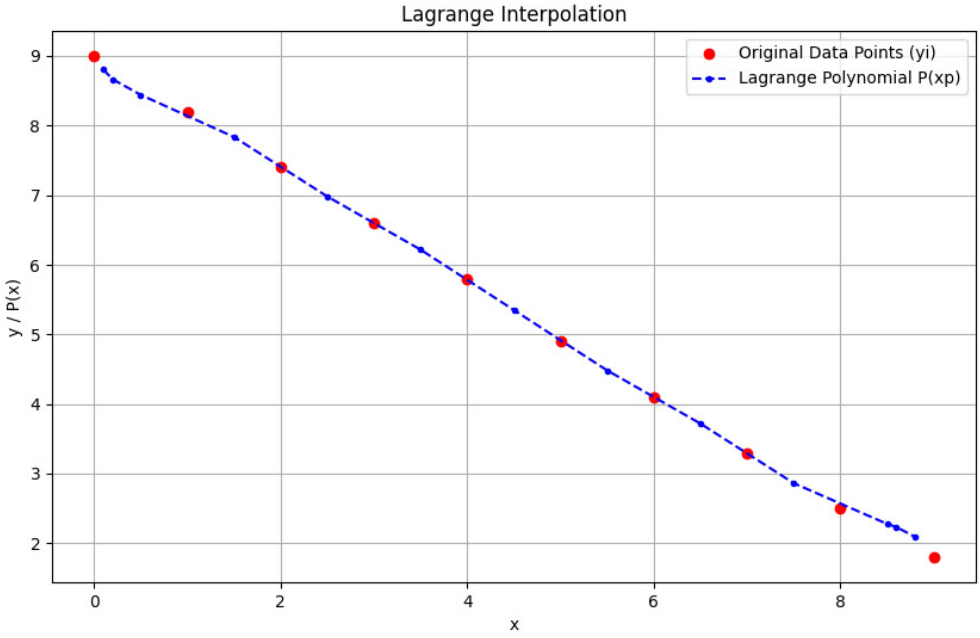

Рисунок 1 – График интерполяционного полинома Лагранжа $L(x)$

2.5 Анализ результатов

Как видно из Рисунка 1 и по определению метода, построенный полином Лагранжа точно проходит через все заданные узловые точки (x_i, y_i). Вычисленные значения в промежуточных точках (Таблица 2) показывают поведение интерполянта между узлами. Для большого числа узлов полином Лагранжа может сильно осциллировать между узлами, однако для данных точек, которые выглядят близкими к линейной зависимости, осцилляции не выражены ярко.

3 Построение кубических сплайнов

3.1 Теоретические сведения

Кубический сплайн — это гладкая функция, составленная из кубических полиномов на каждом отрезке $[x_i, x_{i+1}]$ сетки $x_0, x_1, \ldots, x_n$. Сплайн $S(x)$ удовлетворяет следующим условиям:

1. $S(x_i) = y_i$ для всех $i = 0, 1, \ldots, n$ (условие интерполяции).
2. На каждом отрезке $[x_i, x_{i+1}]$, $S(x)$ является кубическим полиномом $S_i(x)$.
3. $S(x)$, $S'(x)$ и $S''(x)$ непрерывны на всем интервале $[x_0, x_n]$.

Для однозначного определения сплайна требуются дополнительные граничные условия. В данной работе используется *естественный кубический сплайн*, для которого вторые производные на концах интервала равны нулю: $S''(x_0) = 0$ и $S''(x_n) = 0$.

На каждом отрезке $[x_i, x_{i+1}]$ кубический полином $S_i(x)$ можно представить в виде:

$$S_i(x) = M_i \frac{(x_{i+1} - x)^3}{6h_i} + M_{i+1} \frac{(x - x_i)^3}{6h_i} + \left(\frac{y_{i+1}}{h_i} - \frac{M_{i+1} h_i}{6} \right) (x - x_i) + \left(\frac{y_i}{h_i} - \frac{M_i h_i}{6} \right) (x_{i+1} - x)$$

где $h_i = x_{i+1} - x_i$ — шаг сетки (в данной работе сетка равномерная, $h_i = h$), а $M_i = S''(x_i)$ — значения вторых производных сплайна в узлах сетки.

Значения $M_1, \ldots, M_{n-1}$ находятся из системы линейных уравнений, которая выводится из условия непрерывности первой производной $S'(x)$ в узлах $x_1, \ldots, x_{n-1}$. Для равномерной сетки ($h_i = h$) система имеет вид:

$$hM_{i-1} + 4hM_i + hM_{i+1} = \frac{6}{h}(y_{i+1} - 2y_i + y_{i-1}), \quad i = 1, 2, \ldots, n-1$$

С учетом естественных граничных условий $M_0 = 0$ и $M_n = 0$, эта система является трехдиагональной и может быть эффективно решена методом прогонки (алгоритм Томаса).

3.2 Постановка задачи и исходные данные

Для функции $f(x) = \sin(x^2)e^{-x^2}$ на отрезке $[a, b] = [0, 3]$ с количеством узлов $n = 20$:

1. Вычислить значения функции $y_i = f(x_i)$ в узлах равномерной сетки $x_i = a + i \cdot h$, где $h = (b - a)/(n - 1)$.
2. Построить естественный кубический сплайн $S(x)$, интерполирующий эти значения.
3. Вычислить значения сплайна $S(x_{\text{mid}})$ в серединах отрезков $x_{\text{mid},i} = (x_i + x_{i+1})/2$.
4. Сравнить значения сплайна $S(x_{\text{mid}})$ с точными значениями функции $f(x_{\text{mid}})$ в этих точках, вычислив абсолютную погрешность $|S(x_{\text{mid}}) - f(x_{\text{mid}})|$.

Параметры задачи:

- Функция: $f(x) = \sin(x^2)e^{-x^2}$
- Интервал: $[a, b] = [0, 3]$
- Количество узлов: $n = 20$

Шаг сетки $h = (3 - 0)/(20 - 1) \approx 0.15789$.

3.3 Описание реализации

Построение и вычисление кубического сплайна реализовано с помощью нескольких функций:

- ‘func(x)‘: вычисляет значение исходной функции $f(x) = \sin(x^2)e^{-x^2}$.
- ‘solve_tridiagonal‘: решает систему линейных уравнений с трехдиагональной матрицей методом прогонки (алгоритм Томаса). Используется для нахождения вторых производных M_i.
- ‘build_natural_cubic_spline‘: формирует узлы x_i, вычисляет значения $y_i = f(x_i)$, составляет и решает систему для M_i с использованием ‘solve_tridiagonal‘, и возвращает структуру ‘SplineData‘, содержащую x, y, M, h.
- ‘evaluate_spline‘: вычисляет значение сплайна $S(x_p)$ в заданной точке x_p, используя найденные x_i, y_i, M_i и формулу для $S_i(x)$ на соответствующем интервале $[x_i, x_{i+1}]$.

Код программы находится в листинге 1.

3.4 Результаты вычислений и графики

Был построен естественный кубический сплайн $S(x)$ для функции $f(x) = \sin(x^2)e^{-x^2}$ на отрезке $[0, 3]$ с $n = 20$ узлами. Узлы x_i, значения функции $y_i = f(x_i)$ и вычисленные вторые производные сплайна $M_i = S''(x_i)$ приведены на рисунке 2.

```
Узлы (xi, yi=f(xi)) и вторые производные (Mi):
 xi      | yi=f(xi)   | Mi
---------|------------|---------------
0.00000000   0.00000000  0.00000000
0.15789474   0.02431437  2.30047174
0.31578947   0.09010857  0.78092507
0.47368421   0.17777964  -0.15913862
0.63157895   0.26063929  -1.30231793
0.78947368   0.31296939  -1.97903603
0.94736842   0.31863930  -2.01109010
1.10526316   0.27696776  -1.37011297
1.26315789   0.20273112  -0.34579065
1.42105263   0.11960309  0.61341366
1.57894737   0.04992436  1.12893439
1.73684211   0.00610352  1.09398133
1.89473684   -0.01196509     0.69284338
2.05263158   -0.01299247     0.23590031
2.21052632   -0.00743432     -0.05152681
2.36842105   -0.00228560     -0.12832798
2.52631579   0.00016731  -0.08395532
2.68421053   0.00059184  -0.02401223
2.84210526   0.00030270  0.00824722
3.00000000   0.00005086  0.00000000
```

Рисунок 2 – Узлы интерполяции x_i, значения функции $y_i = f(x_i)$ и вторые производные M_i для кубического сплайна

Сравнение значений сплайна $S(x_{\text{mid}})$ и точной функции $f(x_{\text{mid}})$ в серединах отрезков $x_{\text{mid},i} = (x_i + x_{i+1})/2$, а также абсолютная погрешность $|S(x_{\text{mid}}) - f(x_{\text{mid}})|$ приведены на рисунке 3.

```
Сравнение в серединах интервалов:
 x_mid      | S(x_mid)   | f(x_mid)   | |S-f|
--------------|--------------|--------------|--------------
0.07894737   0.00857266  0.00619392  0.00237873
0.23684211   0.05241012  0.05300644  0.00059632
0.39473684   0.13297525  0.13279630  0.00017895
0.55263158   0.22148667  0.22154651  0.00005984
0.71052632   0.29191725  0.29194449  0.00002724
0.86842105   0.32202165  0.32207564  0.00005399
1.02631579   0.30307202  0.30311199  0.00003997
1.18421053   0.24252311  0.24253519  0.00001208
1.34210526   0.16075010  0.16072851  0.00002159
1.50000000   0.08204885  0.08200831  0.00004054
1.65789474   0.02455026  0.02451452  0.00003574
1.81578947   -0.00571496     -0.00572884     0.00001388
1.97368421   -0.01392592     -0.01391786     0.00000806
2.13157895   -0.01050068     -0.01048367     0.00001701
2.28947368   -0.00457971     -0.00456725     0.00001247
2.44736842   -0.00072837     -0.00072478     0.00000359
2.60526316   0.00054781  0.00054491  0.00000290
2.76315789   0.00047183  0.00047168  0.00000015
2.92105263   0.00016393  0.00015332  0.00001061
```

Рисунок 3 – равнение значений кубического сплайна $S(x)$ и точной функции $f(x)$ в серединах отрезков

На Рисунке 4 показано сравнение графика кубического сплайна $S(x)$ (зеленые треуголь-

ники - значения в серединах отрезков), точной функции $f(x)$ (синяя сплошная линия) и узлов интерполяции (красные точки). На Рисунке 5 показана абсолютная погрешность $|S(x_{\text{mid}}) - f(x_{\text{mid}})|$ в логарифмическом масштабе.

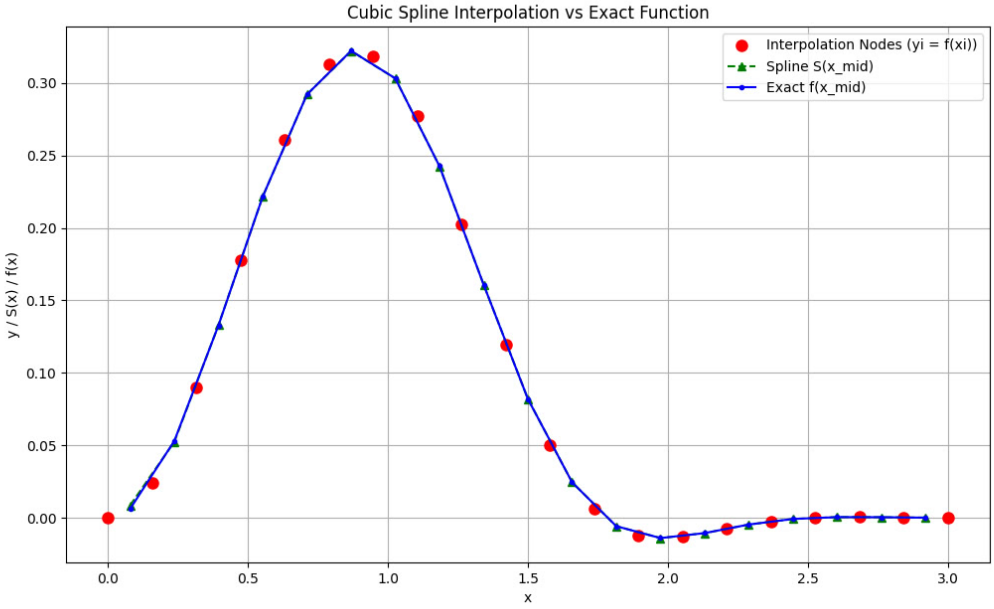

Рисунок 4 – Сравнение кубического сплайна $S(x)$ и точной функции $f(x)$

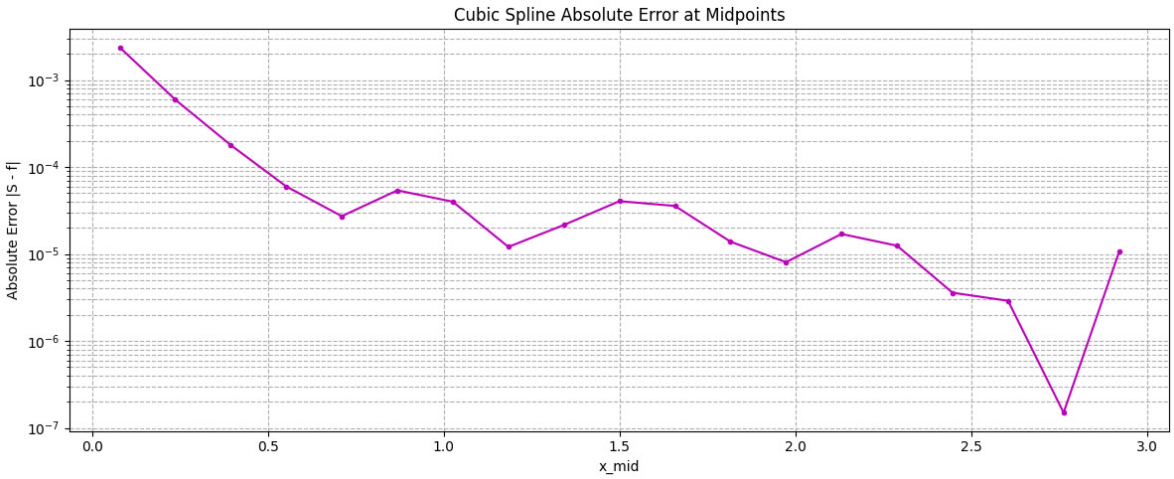

Рисунок 5 – Абсолютная погрешность $|S(x_{\text{mid}}) - f(x_{\text{mid}})|$ кубического сплайна в серединах отрезков (логарифмическая шкала по оси Y)

3.5 Анализ результатов

Результаты показывают, что кубический сплайн обеспечивает хорошее приближение функции $f(x) = \sin(x^2)e^{-x^2}$ на отрезке $[0, 3]$ при использовании 20 узлов.

Из Рисунка 4 видно, что график сплайна визуально очень близок к графику точной функции. Сплайн точно проходит через все узловые точки (красные кружки).

Анализ погрешности (Рисунок 3 и Рисунок 5) показывают, что абсолютная ошибка в серединах отрезков невелика. Наибольшая погрешность ($\approx 2.4 \times 10^{-3}$) наблюдается в самом первом интервале, что может быть связано с нулевыми граничными условиями для $S''(0) = 0$ и быстрым изменением функции вблизи нуля. На остальной части интервала погрешность значительно меньше, в основном порядка $10^{-5} - 10^{-7}$. Это свидетельствует о высокой точности аппроксимации кубическим сплайном для данной функции и выбранного числа узлов. Локальные пики погрешности на Рисунке 5 могут соответствовать областям, где кривизна функции $f(x)$ изменяется наиболее резко между узлами.

4 Программная реализация

Листинг 1 – Код для решения домашнего задания

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <numeric>   // Для std::iota
#include <stdexcept> // Дляобработкиошибок
#include <algorithm> // Для std::lower_bound


// Вычисляетзначениеполиномалагранжавточке       xp
double lagrange_interpolation(const std::vector<double>& x_nodes,
                              const std::vector<double>& y_nodes,
                              double xp) {
    if (x_nodes.size() != y_nodes.size() || x_nodes.empty()) {
        throw std::invalid_argument("Некорректные
           узлыдляинтерполяцииЛагранжа    .");
    }
    size_t n = x_nodes.size();
    double interpolated_value = 0.0;

    for (size_t i = 0; i < n; ++i) {
        double basis_polynomial = 1.0;
        for (size_t j = 0; j < n; ++j) {
            if (i != j) {
                // Проверканасовпадениеузлов
                if (std::abs(x_nodes[i] - x_nodes[j]) < 1e-9) {
                    throw std::runtime_error(
                        "Обнаружены дублирующиеся-узлы x,
                           дляинтерполяцииЛагранжатребуютсяразличныеузлы      .")
                           ;
                }
                basis_polynomial *= (xp - x_nodes[j]) / (x_nodes[i] -
                   x_nodes[j]);
            }
        }
        interpolated_value += y_nodes[i] * basis_polynomial;
    }
    return interpolated_value;
```

```
}

void lagrange_method() {
    std::cout << "=============================================" << std::
       endl;
    std::cout << " ПримеринтерполяцииЛагранжа                  " << std::endl;
    std::cout << "=============================================" << std::
       endl;

    // ДанныеизВарианта  12, Таблица1
    std::vector<double> x_nodes = {0.0, 1.0, 2.0, 3.0, 4.0,
                                   5.0, 6.0, 7.0, 8.0, 9.0};
    std::vector<double> y_nodes = {9.0, 8.2, 7.4, 6.6, 5.8,
                                   4.9, 4.1, 3.3, 2.5, 1.8};

    std::cout << "Входные точки:\n";
    std::cout << std::fixed << std::setprecision(1);
    std::cout << "xi\t| yi\n";
    std::cout << "--------|-------\n";
    for (size_t i = 0; i < x_nodes.size(); ++i) {
        std::cout << x_nodes[i] << "\t " << y_nodes[i] << std::endl;
    }
    std::cout << std::endl;

    // Точки, вкоторыхоцениваемполиномЛагранжа
    std::vector<double> x_eval = {0.1, 0.2, 0.5, 1.5, 2.5, 3.5,
                                  4.5, 5.5, 6.5, 7.5, 8.5, 8.6, 8.8};

    std::cout << "Значения полиномаЛагранжа :\n";
    std::cout << std::fixed << std::setprecision(6);
    std::cout << "xp\t| P(xp)\n";
    std::cout << "--------|-------------\n";
    try {
        for (double xp : x_eval) {
            double yp = lagrange_interpolation(x_nodes, y_nodes, xp);
            std::cout << xp << "\t " << yp << std::endl;
        }
    } catch (const std::exception& e) {
        std::cerr << "Ошибка интерполяцииЛагранжа : " << e.what() << std::
           endl;
    }
    std::cout << std::endl;
```

```
}

// Функциядляинтерполяции
double func(double x) {
    // f(x) = sin(x^2) * e^(x^2)
    return sin(x * x) * exp(-x * x);
}

// Структурадляхраненияданныхсплайна
struct SplineData {
    std::vector<double> x; // Узлы xi
    std::vector<double> y; // Значения yi = f(xi)
    std::vector<double> M; // Вторыепроизводные  Mi вузлах
    double h;              // Шагпредполагается ( равномернаясетка )
};

// РешениетрёхдиагональнойСЛАУметодомТомаса
void solve_tridiagonal(const std::vector<double>& a,
                       const std::vector<double>& b,
                       const std::vector<double>& c,
                       const std::vector<double>& d,
                       std::vector<double>& x,
                       int n) {
    if (n <= 0) return;

    std::vector<double> c_prime(n);
    std::vector<double> d_prime(n);

    // Прямойход
    c_prime[0] = c[0] / b[0];
    d_prime[0] = d[0] / b[0];
    for (int i = 1; i < n; ++i) {
        double m = 1.0 / (b[i] - a[i] * c_prime[i - 1]);
        c_prime[i] = (i < n - 1) ? (c[i] * m) : 0.0;
        d_prime[i] = (d[i] - a[i] * d_prime[i - 1]) * m;
    }

    // Обратныйход
    x[n - 1] = d_prime[n - 1];
    for (int i = n - 2; i >= 0; --i) {
        x[i] = d_prime[i] - c_prime[i] * x[i + 1];
    }
```

```
}

// Построениеестественногокубическогосплайна
SplineData build_natural_cubic_spline(double a, double b, int n) {
    if (n < 3) {
        throw std::invalid_argument("Для кубическогосплайнанужноминимум     3
           точки.");
    }

    SplineData spline;
    spline.x.resize(n);
    spline.y.resize(n);
    spline.M.resize(n);
    spline.h = (b - a) / (n - 1);

    // Генерацияузловизначенийфункции
    for (int i = 0; i < n; ++i) {
        spline.x[i] = a + i * spline.h;
        spline.y[i] = func(spline.x[i]);
    }

    // Настройкатрёхдиагональнойсистемыдля     M1 ... M_{n-2}
    int system_size = n - 2; // Решаемдля  M1..M_{n-2}
    if (system_size <= 0) {
        spline.M[0] = 0.0;
        if (n > 1) spline.M[n - 1] = 0.0;
        if (n == 3) { // Отдельныйслучай  n=3
            double d1 = (6.0 / spline.h) *
                        ((spline.y[2] - spline.y[1]) / spline.h -
                         (spline.y[1] - spline.y[0]) / spline.h);
            double b1 = 4.0 * spline.h;
            spline.M[1] = d1 / b1;
        }
        return spline;
    }

    // Коэффициентысистемы
    std::vector<double> sub_diag(system_size);
    std::vector<double> main_diag(system_size);
    std::vector<double> super_diag(system_size);
    std::vector<double> rhs(system_size);
```

```
    // Формулыдляестественногосплайна
    for (int k = 0; k < system_size; ++k) {
        int i = k + 1;
        main_diag[k] = 4.0 * spline.h;
        rhs[k] = (6.0 / spline.h) *
                 (spline.y[i + 1] - 2.0 * spline.y[i] + spline.y[i - 1]);
        if (k > 0) sub_diag[k] = spline.h;
        if (k < system_size - 1) super_diag[k] = spline.h;
    }

    // Подготовкак  TDMA
    std::vector<double> tdma_a(system_size), tdma_b(system_size),
        tdma_c(system_size), tdma_d(system_size), solution_M(system_size)
           ;

    for (int k = 0; k < system_size; ++k) {
        tdma_b[k] = main_diag[k];
        tdma_d[k] = rhs[k];
        tdma_a[k] = (k > 0) ? sub_diag[k] : 0;
        tdma_c[k] = (k < system_size - 1) ? super_diag[k] : 0;
    }

    solve_tridiagonal(tdma_a, tdma_b, tdma_c, tdma_d, solution_M,
       system_size);

    // Записьполученных  Mi
    spline.M[0] = 0.0;
    for (int k = 0; k < system_size; ++k) {
        spline.M[k + 1] = solution_M[k];
    }
    spline.M[n - 1] = 0.0;

    return spline;
}

// Оценказначениясплайна   S(xp) вточке  xp
double evaluate_spline(const SplineData& spline, double xp) {
    size_t n = spline.x.size();
    if (n < 2) {
        throw std::runtime_error("Сплайн
           непостроенилисодержитслишкоммалоточек       .");
    }
```

```
    // Поискинтервала [xi, x_{i+1}], вкоторомнаходится xp
    auto it = std::lower_bound(spline.x.begin(), spline.x.end(), xp);
    int i = std::distance(spline.x.begin(), it);

    if (i == n) i = n - 1;
    if (i > 0 && (xp < spline.x[i] || std::abs(xp - spline.x[i]) < 1e-9))
        --i;

    i = std::max(0, std::min((int)n - 2, i));

    double xi = spline.x[i];
    double xi1 = spline.x[i + 1];
    double yi = spline.y[i];
    double yi1 = spline.y[i + 1];
    double Mi = spline.M[i];
    double Mi1 = spline.M[i + 1];
    double h = spline.h;

    if (std::abs(h) < 1e-9) {
        throw std::runtime_error("Нулевая ширинаинтерваласплайна .");
    }

    // Формулакубическогосплайнана [xi, xi+1]
    double term1 = Mi * pow(xi1 - xp, 3) / (6.0 * h);
    double term2 = Mi1 * pow(xp - xi, 3) / (6.0 * h);
    double term3 = (yi / h - Mi * h / 6.0) * (xi1 - xp);
    double term4 = (yi1 / h - Mi1 * h / 6.0) * (xp - xi);

    return term1 + term2 + term3 + term4;
}

void cubic_spline_method() {
    std::cout << "=============================================" << std::
       endl;
    std::cout << " Примеркубическойсплайн-интерполяции          " << std::endl;
    std::cout << "=============================================" << std::
       endl;

    double a = 0.0, b = 3.0;
    int n = 20;
    std::cout << "Функция: f(x) = sin(x^2) * exp(-x^2)\n";
```

```
std::cout << "Интервал: [" << a << ", " << b << "]\n";
std::cout << "Число узлов n: " << n << "\n\n";

SplineData spline;
try {
    spline = build_natural_cubic_spline(a, b, n);
} catch (const std::exception& e) {
    std::cerr << "Ошибка построениясплайна : " << e.what() << std::endl;
    return;
}

std::cout << "Узлы (xi, yi=f(xi)) ивторыепроизводные   (Mi):\n";
std::cout << std::fixed << std::setprecision(8);
std::cout << " xi\t\t| yi=f(xi)\t| Mi\n";
std::cout << "--------|---------------|---------------\n";
for (int i = 0; i < n; ++i) {
    std::cout << spline.x[i] << "\t " << spline.y[i] << "\t " <<
       spline.M[i]
              << std::endl;
}
std::cout << std::endl;

std::cout << "Сравнение всерединахинтервалов  :\n";
std::cout << " x_mid\t\t| S(x_mid)\t| f(x_mid)\t| |-Sf|\n";
std::cout << "
   ---------------|---------------|---------------|---------------\n"
   ;

for (int i = 0; i < n - 1; ++i) {
    double x_mid = (spline.x[i] + spline.x[i + 1]) / 2.0;
    double spline_val = 0.0;
    double exact_val = func(x_mid);
    try {
        spline_val = evaluate_spline(spline, x_mid);
    } catch (const std::exception& e) {
        std::cerr << "Ошибка вычислениясплайнав   x=" << x_mid << ": "
                  << e.what() << std::endl;
        spline_val = NAN;
    }
    double error = std::abs(spline_val - exact_val);

    std::cout << x_mid << "\t " << spline_val << "\t " << exact_val
```

```
            << "\t "
                    << error << std::endl;
    }
    std::cout << std::endl;
}

int main() {
    lagrange_method();

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

    ubic_spline_method();

    return 0;
}
```

5 Заключение

В ходе выполнения домашней работы были изучены, реализованы и применены два метода аппроксимации функций: интерполяционный полином Лагранжа и кубические сплайны.

Для таблично заданной функции был успешно построен полином Лагранжа, точно проходящий через заданные узлы.

Для функции $f(x) = \sin(x^2)e^{-x^2}$ на отрезке $[0, 3]$ был построен естественный кубический сплайн с использованием 20 узлов. Проведено сравнение значений сплайна с точными значениями функции в промежуточных точках (серединах отрезков). Анализ абсолютной погрешности показал высокую точность аппроксимации кубическим сплайном (порядка $10^{-5} - 10^{-7}$ на большей части интервала), что подтверждается графиками.