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

ФАКУЛЬТЕТ	Информатика и система управления
Кафедра	ИУ6 Компьютерные системы и сети
Группа	ИУ6-63Б

ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Отчет по домашнему заданию 3:

Аппроксимация функций

Вариант 10

Студент:		В.К. Залыгин
	дата, подпись	Ф.И.О.
Преподаватель:		Я.Ю. Павловский
	дата, подпись	Ф.И.О.

Москва, 2025

Оглавление

Цель домашней работы

Изучение интерполяционного полинома Лагранжа, метода построения кубических сплайнов.

Постановка задачи и исходные данные

x_i	1	2	3	4	5	6	7	8	9	10
y_i	178	182	190	199	200	213	220	231	235	242

Рисунок 1 – Условие домашней работы

Вариант	n	$[a,b]$	$f(x)$
10	25	[0,5]	$\frac{arctgx}{1+arctgx}$

Рисунок 2 – Условие домашней работы

Краткое описание реализуемых методов

Интерполяционный полином Лагранжа

Интерполяционный полином Лагранжа — это один из методов построения полинома, который проходит через заданный набор точек. Пусть даны $n+1$ различных узлов интерполяции $x_0, x_1, \ldots, x_n$ и соответствующие значения функции $f(x_0), f(x_1), \ldots, f(x_n)$. Тогда полином Лагранжа $L_n(x)$ имеет вид:

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

где базисные полиномы $l_i(x)$ определяются как:

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Каждый базисный полином $l_i(x)$ равен единице в узле x_i и нулю в остальных узлах x_j при $j \neq i$. Таким образом, полином Лагранжа обеспечивает точное совпадение с заданной функцией в узлах интерполяции.

Преимущества метода Лагранжа:

- Простота построения при небольшом числе точек.
- Гарантированное прохождение через заданные точки.

Недостатки:

- При большом числе точек возрастает степень полинома, что приводит к эффекту Рунге — сильным колебаниям между узлами.
- Неудобство пересчёта при добавлении новых узлов — требуется пересчитывать весь полином.

Кубические сплайны

Кубические сплайны — это способ аппроксимации функции с помощью набора кусочно-гладких полиномов третьей степени, обеспечивающих хорошую гладкость на границах отрезков. Для набора точек $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ кубический сплайн представляет собой набор функций $S_i(x)$, определённых на каждом промежутке $[x_i, x_{i+1}]$, вида:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Коэффициенты a_i, b_i, c_i, d_i подбираются так, чтобы обеспечить выполнение следующих условий:

1. $S_i(x_i) = y_i$ и $S_i(x_{i+1}) = y_{i+1}$ — интерполяция в узлах.
2. Непрерывность первых и вторых производных во внутренних узлах $x_1, x_2, \ldots, x_{n-1}$.
3. Дополнительные условия на краях интервала (например, нулевые вторые производные в крайних точках для естественного сплайна).

Решение задачи сводится к решению системы линейных уравнений относительно коэффициентов.

Преимущества кубических сплайнов:

- Обеспечивают высокую степень гладкости (не только функция, но и её первая и вторая производные непрерывны).
- Избегают эффекта Рунге даже при большом числе узлов.
- Удобны для добавления новых точек без перерасчёта всей конструкции.

Недостатки:

- Требуют решения системы уравнений для определения коэффициентов.
- Немного сложнее в реализации по сравнению с методом Лагранжа.

Текст программы

Ниже в листингах приведены листинги кода. Ввод данных (файл data_input.py) представлен в листинге 1.

Листинг 1 – Функции, инициализирующие данные

```
from math import cos
def get_tabulated_points():
    xi = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    yi = [178,182,190,199,200,213,220,231,235,242]
    return xi, yi
def f(x):
    return cos(x + cos(x)**3)

def get_spline_function_and_range():
    return 25, (-1, 4), f
```

Метод простых итераций (файл lagrange.py) представлен в листинге 2.

Листинг 2 – Функции lagrange_polynomial, plot_lagrange

```
import matplotlib.pyplot as plt
import numpy as np
from data_input import get_tabulated_points

def lagrange_polynomial(x, xi, yi):
    """Вычисляет значение полинома Лагранжа в точке x."""
    n = len(xi)
    result = 0
    for i in range(n):
        term = yi[i]
        for j in range(n):
            if i != j:
                term *= (x - xi[j]) / (xi[i] - xi[j])
        result += term
    return result

def plot_lagrange():
    xi, yi = get_tabulated_points()
    x = np.linspace(min(xi), max(xi), 500)
    y = [lagrange_polynomial(xi_val, xi, yi) for xi_val in x]

    plt.figure(figsize=(10, 6))
    plt.plot(x, y, label="Полином Лагранжа", color='blue')
    plt.scatter(xi, yi, color='red', label="Узлы интерполяции")
    plt.title("Интерполяция Полиномом Лагранжа")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid(True)
    plt.legend()
    plt.show()

if name == "__main__":
    plot_lagrange()
```

Отрисовка метода кубических сплайнов (файл spline.py) представлена в листинге 3.

Листинг 3 – Функция spline.py

```
import matplotlib.pyplot as plt
import numpy as np
from custom_spline import NaturalSpline
from data_input import get_spline_function_and_range

def build_cubic_spline():
    n, (a, b), f  = get_spline_function_and_range()
    xi = np.linspace(a, b, n)
    yi = [f(x) for x in xi]

    spline = NaturalSpline(xi, yi)

    # для отрисовки
```

```
    x_dense = np.linspace(a, b, n)
    x_exact = np.linspace(a, b, 500)
    y_dense = spline(x_dense)
    y_exact = [f(x) for x in x_exact]

    # Вывод значений в консоль (бонусом)
    print(f"{'x':>10} {'Spline(x)':>15} {'Exact f(x)':>15}
{'|Error|':>10}")
    print("-" * 55)
    for x_val, spline_val, exact_val in zip(x_dense, y_dense,
y_exact):
        error = abs(spline_val - exact_val)
        print(f"{x_val:10.5f} {spline_val:15.5f} {exact_val:15.5f}
{error:10.5e}")


    plt.figure(figsize=(10, 6))
    plt.plot(x_exact, y_exact, label="Точная функция", linestyle="-
-", color='green')
    plt.plot(x_dense, y_dense, label="Кубический сплайн",
color='blue')
    plt.scatter(xi, yi, color='red', label="Узлы интерполяции")
    plt.title("Интерполяция Кубическим Сплайном")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.grid(True)
    plt.legend()
    plt.show()

if name == "__main__":
    build_cubic_spline()
```

Метод кубических сплайнов (файл custom_spline.py) представлен в листинге 4.

Листинг 4 – Функция custom_spline.py

```
import numpy as np
from data_input import f

class NaturalCubicSpline:
    def __init__(self, x, y):
        self.n = len(x) - 1
        self.x = x
        self.y = y
        self.h = [x[i+1] - x[i] for i in range(self.n)]

        # Решаем систему для коэффициентов c
        self.a = y
        self.c = self._solve_c()
        self.b = [0] * self.n
        self.d = [0] * self.n

        for i in range(self.n):
            self.d[i] = (self.c[i+1] - self.c[i]) / (3 * self.h[i])
```

```
            self.b[i] = (self.a[i+1] - self.a[i]) / self.h[i] -
self.h[i] * (self.c[i+1] + 2*self.c[i]) / 3

    def _solve_c(self):
        """Решение трёхдиагональной системы методом прогонки"""
        n = self.n
        A = [0] + [self.h[i-1] for i in range(1, n)]
        B = [2 * (self.h[i-1] + self.h[i]) for i in range(1, n)]
        C = [self.h[i] for i in range(1, n)] + [0]
        F = [0] * (n-1)
        for i in range(1, n):
            F[i-1] = 3 * ((self.a[i+1] - self.a[i]) / self.h[i] -
(self.a[i] - self.a[i-1]) / self.h[i-1])

        # Прямой ход
        alpha = [0] * (n-1)
        beta = [0] * (n-1)
        alpha[0] = -C[0] / B[0]
        beta[0] = F[0] / B[0]

        for i in range(1, n-1):
            denom = B[i] + A[i] * alpha[i-1]
            alpha[i] = -C[i] / denom
            beta[i] = (F[i] - A[i] * beta[i-1]) / denom
        # Обратный ход
        c = [0] * (n+1)
        for i in reversed(range(1, n)):
            c[i] = alpha[i-1] * c[i+1] + beta[i-1]
        c[0] = c[n] = 0  # Свободные края

        return c

    def __call__(self, x_val):
        """Вычисляет значение сплайна в точке x_val"""
        # Находим нужный отрезок
        i = self._find_segment(x_val)
        dx = x_val - self.x[i]
        return self.a[i] + self.b[i]*dx + self.c[i]*dx**2 +
self.d[i]*dx**3

    def _find_segment(self, x_val):
        """Бинарный поиск интервала"""
        left = 0
        right = self.n
        while left <= right:
            mid = (left + right) // 2
            if self.x[mid] <= x_val <= self.x[mid+1]:
                return mid
            elif x_val < self.x[mid]:
                right = mid - 1
            else:
                left = mid + 1
        return self.n - 1  # если вдруг x_val == x[-1]
```

Результаты выполнения программы

Ниже представлены результаты интерполяции полиномом Лагранжа и кубическим сплайном.

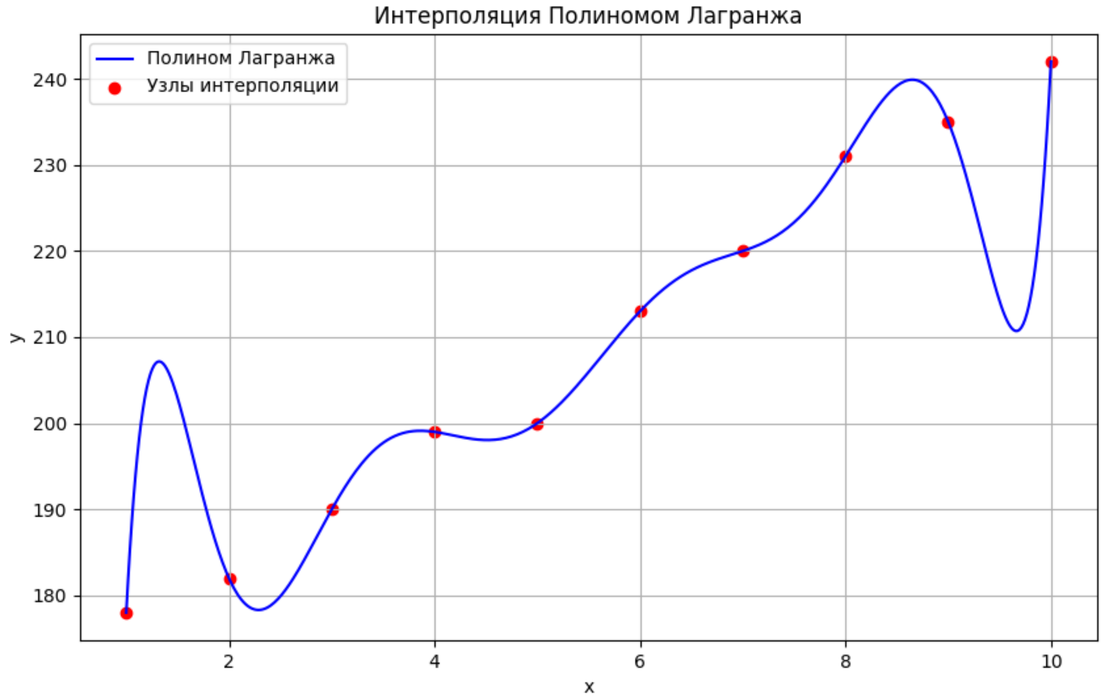

Рисунок 3 – Результаты выполнения программы

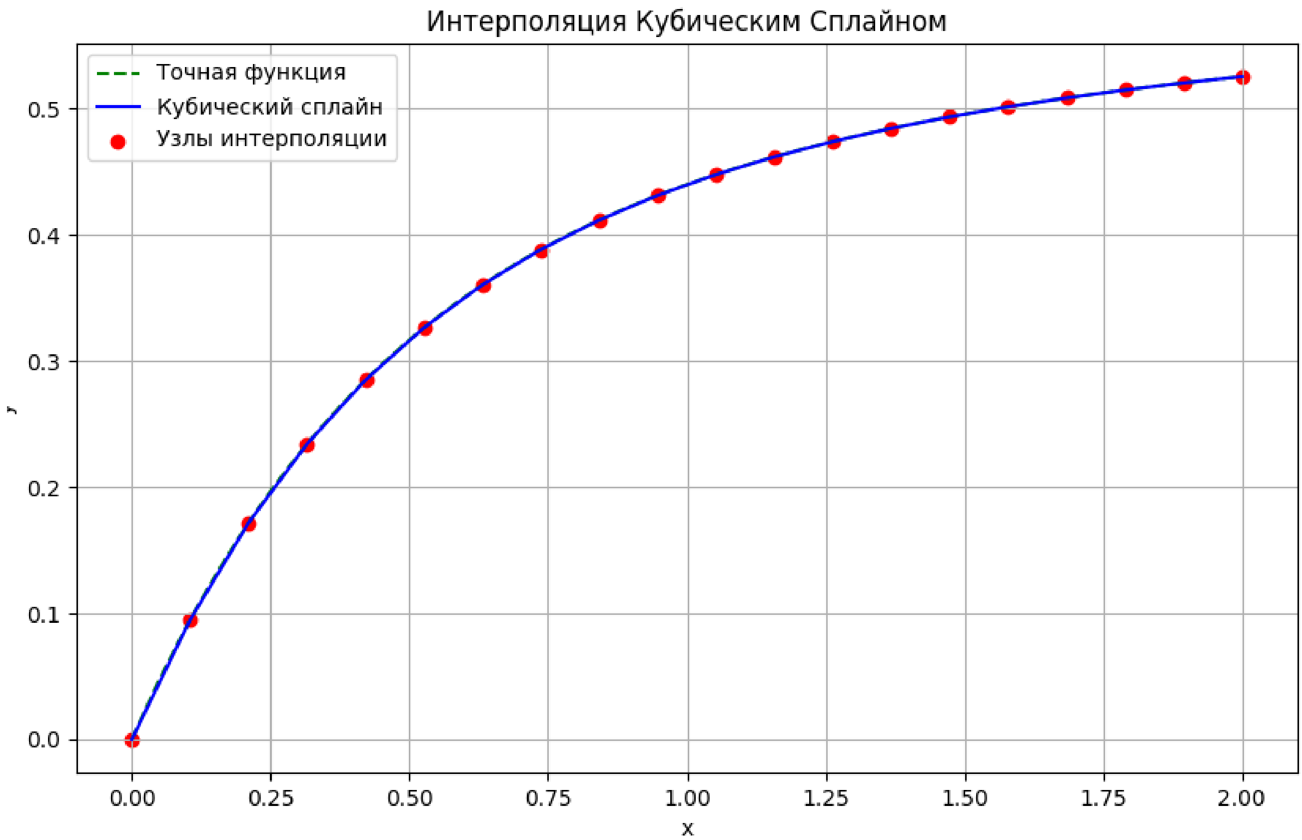

Рисунок 4 – Результаты выполнения программы

x	Spline(x)	Exact f(x)	|Error|
0.00000	0.00000	0.00000	0.00000e+00
0.10526	0.09492	0.00399	9.09298e-02
0.21053	0.17184	0.00795	1.63888e-01
0.31579	0.23423	0.01188	2.22352e-01
0.42105	0.28496	0.01578	2.69182e-01
0.52632	0.32636	0.01964	3.06719e-01
0.63158	0.36033	0.02348	3.36855e-01
0.73684	0.38839	0.02728	3.61106e-01
0.84211	0.41173	0.03106	3.80670e-01
0.94737	0.43129	0.03480	3.96492e-01
1.05263	0.44783	0.03852	4.09313e-01
1.15789	0.46191	0.04220	4.19714e-01
1.26316	0.47400	0.04586	4.28149e-01
1.36842	0.48446	0.04948	4.34980e-01
1.47368	0.49357	0.05308	4.40489e-01
1.57895	0.50155	0.05665	4.44905e-01
1.68421	0.50860	0.06019	4.48410e-01
1.78947	0.51485	0.06370	4.51149e-01
1.89474	0.52042	0.06718	4.53243e-01
2.00000	0.52543	0.07064	4.54788e-01

Рисунок 5 – Результаты выполнения программы

Анализ результатов

Все методы дали корректные результаты.

1) При сравнении значений функций в промежуточных точках (не совпадающих с узлами интерполяции) установлено, что кубический сплайн более точно воспроизводит поведение исходной функции по сравнению с полиномом Лагранжа. Ошибки сплайна в промежуточных точках оказались значительно меньше.

2) На практике метод кубических сплайнов оказался предпочтительнее для большого числа узлов, поскольку избегает резких колебаний и обеспечивает высокую точность аппроксимации. Метод Лагранжа целесообразно применять при малом количестве узлов или при необходимости точного восстановления значения функции в ограниченном числе точек.

3) Графики аппроксимации на заданной сетке продемонстрировали, что обе методики корректно выполняют поставленную задачу интерполяции. Однако различия в поведении между узлами становятся критичными при увеличении

сложности функции.