НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
Факультет “Компьютерных технологий в дизайне”

# Лабораторная работа № 5
по дисциплине “Вычислительная математика”
вариант №21

Выполнили:
Савельева Елизавета Юрьевна
Фурзикова Александра Евгеньевна
Группа: P3266

Преподаватель:
Машина Екатерина Алексеевна

г. Санкт-Петербург
2025

**Цель:** решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек. Лабораторная работа состоит из двух частей: вычислительной и программной.

**Порядок выполнения работы:**

1. Выбор исходных данных
2. Определение шага h (узлы равноотстоящие h=0,05).
3. Построение таблицы конечных разностей
4. Вычисление y(X1) с использованием первой формулы Ньютона
5. Вычисление y(X2) с использованием первой формулы Гаусса
6. Запись результатов и анализ точности

**Рабочие формулы методов:**

Формула Ньютона «вперед»:

$$N(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 + \ldots$$

Вторая формула Гаусса:

$$P(x) = y_0 + p\Delta y_{-1/2} + \frac{p(p+1)}{2!}\Delta^2 y_{-1} + \frac{p(p+1)(p-1)}{3!}\Delta^3 y_{-3/2} + \frac{p(p+1)(p-1)(p+2)}{4!}\Delta^4 y_{-2} + \ldots$$

**Вычислительная часть:**

1. *Построение таблицы конечных разностей*

| xi | yi | Δyi | Δ^2yi | Δ^3yi | Δ^4yi | Δ^5yi | Δ^6yi |
|---|---|---|---|---|---|---|---|
| 0.25 | 1,2557 | 0,9207 | 0,0247 | -0,0437 | 1,0756 | -4,1277 | 10,1917 |
| 0.30 | 2,1764 | 0,9454 | -0,019 | 1,0319 | -3,0521 | 6,064 | |
| 0.35 | 3,1218 | 0,9264 | 1,0129 | -2,0202 | 3,0119 | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.40 | 4,0482 | 1,9393 | -1,0073 | 0,9917 | | | |
| 0.45 | 5,9875 | 0,932 | -0,0156 | | | | |
| 0.50 | 6,9195 | 0,9164 | | | | | |
| 0.55 | 7,8359 | | | | | | |

2. *Вычисление значения функции при X1 =0,251 с использованием интерполяционной формулы Ньютона*

Шаг h = x_i+1 − xi = 0.05

Базовый узел x0 = 0.25

Нормированная координата t = (X1 − x0)/h = (0,251 -0.25)/0.05 = 0.02

Поскольку |t| ≤ 1 и точка лежит рядом с левым краем таблицы, применяем формулу Ньютона «вперед».

Формула Ньютона «вперед»:

$$N_4(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0$$

Пошаговое вычисление:

$y_0 = 1{,}2557$

$t\Delta y_0 = 0{,}02 \cdot 0{,}9207 = 0{,}018414$

$\frac{t(t-1)}{2!}\Delta^2 y_0 = \big(0{,}02 \cdot (0{,}02-1)\big)/2 \cdot 0{,}0247 = -0{,}00024206$

$\frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 = (0{,}02 \cdot (0{,}02-1)(0{,}02-2))/6 \cdot (-0{,}0437) = -0{,}000282652$

$\frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 = (0{,}02 \cdot (0{,}02-1)(0{,}02-2)(0{,}02-3))/24 \cdot (1{,}0756) = -0{,}00518295$

Итого $N_4(X1) = 1{,}2557 + 0{,}018414 - 0{,}00024206 - 0{,}000282652 - 0{,}00518295 \approx 1{,}26841$

3. *Вычисление значения функции при X2=0,445 с использованием интерполяционной формулы Гаусса*

Ближайший узел-«центр» xm = 0,40

Нормированное смещение p = (X2-xm)/h = (0,445-0,45)/0,05 = -0,1

p<0 – следовательно необходимо использовать вторую формулу Гаусса.

Вторая формула Гаусса:

$$P_4(x) = y_0 + p\Delta y_{-1/2} + \frac{p(p+1)}{2!}\Delta^2 y_{-1} + \frac{p(p+1)(p-1)}{3!}\Delta^3 y_{-3/2} + \frac{p(p+1)(p-1)(p+2)}{4!}\Delta^4 y_{-2}$$

Где

$y_0 = y_4 = 4{,}0482$

$$\Delta y_{-1/2} = \frac{\Delta y_2 + \Delta y_3}{2} = 1{,}0049$$

$$\Delta^2 y_{-1} = \Delta^2 y_2 = -0{,}0014$$

$$\Delta^3 y_{-3/2} = \frac{\Delta^3 y_1 + \Delta^3 y_2}{2} = 0{,}00035$$

$$\Delta^4 y_{-2} = \Delta^4 y_1 = 0{,}0047$$

Пошаговое вычисление:

$$p\Delta y_{-1/2} = -0.10 \cdot 1{,}0049 = -0.10049$$

$$\frac{p(p+1)}{2!}\Delta^2 y_{-1} = (-0.10 \cdot 0.90)/2 \cdot (-0.0014) = 0.000063$$

$$\frac{p(p+1)(p-1)}{3!}\Delta^3 y_{-3/2} = (-0.10 \cdot 0.90 \cdot (-1.10))/6 \cdot (0.00035) = 0.0000058$$

## Листинг программы:

```
import math
import matplotlib.pyplot as plt
import numpy as np


def read_input_data():
    """Основная функция для выбора способа ввода данных"""
    while True:
        print("1. Ручной ввод")
        print("2. Загрузка из файла")
        print("3. Генерация по математической функции")

        while True:
            try:
                choice = int(input("Выберите вариант (1-3): "))
                if 1 <= choice <= 3:
                    break
                print("Ошибка: Введите число от 1 до 3")
            except ValueError:
                print("Ошибка: Введите целое число")

        if choice == 1:
            return input_from_keyboard()
        elif choice == 2:
            return input_from_file_with_retry()
        else:
            return generate_data_from_function_with_retry()


def input_from_keyboard():
    """Ввод данных с клавиатуры"""
    while True:
        try:
            point_count = int(input("Введите количество точек: "))
            if point_count > 0:
                break
            print("Ошибка: Количество точек должно быть положительным")
        except ValueError:
            print("Ошибка: Введите целое число")

    x_values = []
    y_values = []
    print(f"Введите {point_count} точек в формате x y:")

    for i in range(point_count):
        while True:
            try:
                data = input(f"Точка {i + 1}: ").strip()
                if not data:
                    print("Ошибка: Пустой ввод")
                    continue

                x, y = map(float, data.split())
                x_values.append(x)
                y_values.append(y)
                break
            except ValueError:
                print("Ошибка: Точка должна состоять из двух чисел. Попробуйте снова")
            except Exception:
```

```
            print("Ошибка ввода. Попробуйте снова")

    while len(set(x_values)) != len(x_values):
        print("Ошибка: Узлы интерполяции не должны совпадать. Введите точки заново")
        x_values.clear()
        y_values.clear()
        for i in range(point_count):
            while True:
                try:
                    data = input(f"Точка {i + 1}: ").strip()
                    if not data:
                        print("Ошибка: Пустой ввод")
                        continue

                    x, y = map(float, data.split())
                    x_values.append(x)
                    y_values.append(y)
                    break
                except ValueError:
                    print("Ошибка: Точка должна состоять из двух чисел. Попробуйте снова")
                except Exception:
                    print("Ошибка ввода. Попробуйте снова")

    return x_values, y_values


def input_from_file_with_retry():
    """Чтение данных из файла с обработкой ошибок"""
    while True:
        filename = input("Введите имя файла: ")
        try:
            x_data, y_data = read_data_from_file(filename)

            if len(set(x_data)) != len(x_data):
                print("Ошибка: Узлы интерполяции не должны совпадать")
                continue

            return x_data, y_data
        except FileNotFoundError:
            print("Ошибка: Файл не найден. Проверьте имя файла и попробуйте снова")
        except ValueError as e:
            print(f"Ошибка в файле: {e}. Попробуйте другой файл")
        except Exception as e:
            print(f"Ошибка: {e}. Попробуйте другой файл")


def read_data_from_file(filename):
    """Чтение данных из файла"""
    x_data = []
    y_data = []
    with open(filename, 'r') as file:
        for line_number, content in enumerate(file, 1):
            content = content.strip()
            if not content:
                continue

            try:
                x_val, y_val = map(float, content.split())
                x_data.append(x_val)
                y_data.append(y_val)
            except ValueError:
                raise ValueError(f"Некорректные данные в строке {line_number}: '{content}'")
```

```
    if not x_data:
        raise ValueError("Файл пуст или содержит только пустые строки")

    return x_data, y_data


def generate_data_from_function_with_retry():
    """Генерация данных на основе математической функции"""
    while True:
        function_name = input("Введите имя функции (sin/cos/exp/log): ").lower()
        if function_name in ["sin", "cos", "exp", "log"]:
            break
        print("Ошибка: Несуществующая функция. Доступные: sin, cos, exp, log")

    while True:
        try:
            start_point = float(input("Введите начало отрезка: "))
            break
        except ValueError:
            print("Ошибка: Начало должно быть числом")

    while True:
        try:
            end_point = float(input("Введите конец отрезка: "))
            if end_point > start_point:
                break
            print("Ошибка: Конец должен быть больше начала")
        except ValueError:
            print("Ошибка: Конец должен быть числом")

    while True:
        try:
            point_count = int(input("Введите количество точек: "))
            if point_count > 1:
                break
            print("Ошибка: Количество точек должно быть больше 1")
        except ValueError:
            print("Ошибка: Введите целое число")

    while True:
        try:
            x_vals, y_vals = generate_data_from_function(function_name, start_point, end_point, point_count)
            return x_vals, y_vals
        except ValueError as e:
            print(f"Ошибка: {e}")
            if function_name == "log":
                print("Для функции log все x должны быть > 0")
                while True:
                    try:
                        start_point = float(input("Введите новое начало отрезка (>0): "))
                        if start_point > 0:
                            break
                        print("Ошибка: Начало должно быть > 0")
                    except ValueError:
                        print("Ошибка: Начало должно быть числом")

                while True:
                    try:
                        end_point = float(input("Введите новый конец отрезка (> начала): "))
                        if end_point > start_point:
                            break
                        print("Ошибка: Конец должен быть больше начала")
                    except ValueError:
```

```
                print("Ошибка: Конец должен быть числом")


def generate_data_from_function(func_name, start_val, end_val, point_count):
    """Генерация данных на основе выбранной функции"""
    function_map = {
        'sin': math.sin,
        'cos': math.cos,
        'exp': math.exp,
        'log': lambda x: math.log(x) if x > 0 else None,
    }

    x_vals = []
    y_vals = []
    step_size = (end_val - start_val) / (point_count - 1) if point_count > 1 else 0

    for i in range(point_count):
        x = start_val + i * step_size
        y = function_map[func_name](x)
        if y is None:
            raise ValueError(f"Недопустимое значение x={x} для функции {func_name}")
        x_vals.append(x)
        y_vals.append(y)

    return x_vals, y_vals


def main():
    print("Программа интерполяции функций")
    print("------------------------------")

    while True:
        try:
            x_data, y_data = read_input_data()
            interpolator = InterpolationSolver(x_data, y_data)

            while True:
                try:
                    target_x = float(input("\nВведите аргумент для интерполяции: "))
                    break
                except ValueError:
                    print("Ошибка: Аргумент должен быть числом")

            print("\nТаблица конечных разностей:")
            diff_table = interpolator.compute_finite_differences()
            for row in diff_table:
                print(" ".join(map(lambda val: f"{val:.4f}", row)))

            print("\nРезультаты интерполяции:")
            methods = [
                ("Лагранж", interpolator.interpolate_lagrange(target_x)),
                ("Ньютон (разделённые разности)", interpolator.interpolate_newton_divided(target_x)),
                ("Ньютон (конечные разности)", interpolator.interpolate_newton_finite(target_x)),
                ("Стирлинг", interpolator.interpolate_stirling(target_x)),
                ("Бессель", interpolator.interpolate_bessel(target_x))
            ]

            for method_name, result_value in methods:
                if result_value is not None:
                    print(f"{method_name}: {result_value:.6f}")
                else:
                    print(f"{method_name}: не применимо для данных узлов")
```

```
            interpolator.plot_interpolation(target_x=target_x)

            while True:
                choice = input("\nХотите выполнить еще одну интерполяцию? (y/n): ").lower()
                if choice in ['y', 'n']:
                    break
                print("Ошибка: Введите 'y' или 'n'")

            if choice == 'n':
                print("Завершение программы.")
                break

        except KeyboardInterrupt:
            print("\nПрограмма прервана пользователем")
            break
        except Exception as e:
            print(f"\nПроизошла ошибка: {e}")
            raise e
            print("Попробуйте снова.\n")


class InterpolationSolver:
    def __init__(self, x_nodes, y_values):
        self.x_nodes = x_nodes
        self.y_values = y_values

    def interpolate_lagrange(self, x_target):
        """Интерполяция методом Лагранжа"""
        result = 0.0
        n = len(self.x_nodes)

        for i in range(n):
            term = self.y_values[i]
            for j in range(n):
                if j != i:
                    term *= (x_target - self.x_nodes[j]) / (self.x_nodes[i] - self.x_nodes[j])
            result += term

        return result

    def interpolate_newton_divided(self, x_target):
        """Интерполяция методом Ньютона с разделенными разностями"""
        n = len(self.x_nodes)
        div_diff_table = self.compute_divided_differences()

        result = div_diff_table[0][n - 1]
        for i in range(n - 2, -1, -1):
            result = result * (x_target - self.x_nodes[i]) + div_diff_table[0][i]

        return result

    def interpolate_newton_finite(self, x_target):
        """Интерполяция методом Ньютона с конечными разностями"""
        n = len(self.x_nodes)
        if n < 2:
            return None

        step = abs(self.x_nodes[1] - self.x_nodes[0])

        for i in range(1, n - 1):
            if abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) > 1e-10:
                return None
```

```
        diff_table = self.compute_finite_differences()
        result = self.y_values[0]

        for i in range(1, n):
            coefficient = diff_table[0][i] / math.factorial(i)
            t_param = (x_target - self.x_nodes[0]) / step
            term = 1
            for k in range(i):
                term *= (t_param - k)
            result += coefficient * term

        return result

    def compute_divided_differences(self):
        """Вычисление таблицы разделенных разностей"""
        n = len(self.x_nodes)
        div_diff = [[0.0] * n for _ in range(n)]

        for i in range(n):
            div_diff[i][0] = self.y_values[i]

        for j in range(1, n):
            for i in range(n - j):
                div_diff[i][j] = (div_diff[i + 1][j - 1] - div_diff[i][j - 1]) / (self.x_nodes[i + j] - self.x_nodes[i])

        return div_diff

    def compute_finite_differences(self):
        """Вычисление таблицы конечных разностей"""
        n = len(self.y_values)
        fin_diff = [[0.0] * n for _ in range(n)]

        for i in range(n):
            fin_diff[i][0] = self.y_values[i]

        for j in range(1, n):
            for i in range(n - j):
                fin_diff[i][j] = fin_diff[i + 1][j - 1] - fin_diff[i][j - 1]

        return fin_diff

    def interpolate_stirling(self, x_target):
        """Интерполяция по формуле Стирлинга"""
        n = len(self.x_nodes)
        if n % 2 == 0:  # Требуется нечетное количество точек
            return None

        # Проверка равномерности шага
        step = abs(self.x_nodes[1] - self.x_nodes[0])
        for i in range(1, n - 1):
            if abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) > 1e-10:
                return None

        center_idx = n // 2  # Центральный узел
        t_param = (x_target - self.x_nodes[center_idx]) / step
        diff_table = self.compute_finite_differences()

        result = self.y_values[center_idx]

        # Первый член: t * (Δy0 + Δy_{-1})/2
        if center_idx >= 1 and center_idx < n - 1:
            if center_idx < len(diff_table) and len(diff_table[center_idx]) > 1 and \
                    center_idx - 1 < len(diff_table) and len(diff_table[center_idx - 1]) > 1:
```

```
            result += t_param * (diff_table[center_idx][1] + diff_table[center_idx - 1][1]) / 2

        # Основной цикл по i
        for i in range(1, center_idx + 1):
            # Проверка доступности индексов для нечетных разностей
            row1 = center_idx - i
            row2 = center_idx - i - 1
            order = 2 * i - 1

            if row1 < 0 or row1 >= len(diff_table) or order >= len(diff_table[row1]) or \
                    row2 < 0 or row2 >= len(diff_table) or order >= len(diff_table[row2]):
                break

            delta_val = (diff_table[row1][order] + diff_table[row2][order]) / 2
            term = delta_val

            # Вычисление произведения (t^2 - k^2)
            for k in range(1, i):
                term *= (t_param ** 2 - k ** 2)

            term *= t_param  # Дополнительный множитель t
            term /= math.factorial(2 * i - 1)
            result += term

            # Проверка доступности индексов для четных разностей
            row_even = center_idx - i - 1
            order_even = 2 * i

            if i < center_idx:  # Для четных разностей (кроме последней итерации)
                if row_even < 0 or row_even >= len(diff_table) or order_even >= len(diff_table[row_even]):
                    break

                term_even = diff_table[row_even][order_even]
                for k in range(1, i + 1):
                    term_even *= (t_param ** 2 - k ** 2)

                term_even /= math.factorial(2 * i)
                result += term_even
            else:
                break

        return result

    def find_central_node_index(self, x_target):
        """Поиск центрального узла для интерполяции"""
        n = len(self.x_nodes)
        mid_idx = n // 2
        if x_target < self.x_nodes[mid_idx]:
            while mid_idx > 0 and x_target < self.x_nodes[mid_idx - 1]:
                mid_idx -= 1
        else:
            while mid_idx < n - 1 and x_target > self.x_nodes[mid_idx + 1]:
                mid_idx += 1
        return mid_idx

    def interpolate_bessel(self, x_target):
        """Интерполяция по формуле Бесселя"""

        n = len(self.x_nodes)

        # Метод Бесселя требует чётное количество узлов
        if n % 2 != 0:
            return None
```

```
        # Проверка, что шаг равномерный
        step = self.x_nodes[1] - self.x_nodes[0]
        for i in range(1, n - 1):
            if abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) > 1e-10:
                return None

        # Найдём центральный индекс (левая из двух центральных точек)
        center_idx = self.find_central_interval_index(x_target)
        center_idx = max(1, min(center_idx, n - 3))

        # Центральная точка x
        center_x = (self.x_nodes[center_idx] + self.x_nodes[center_idx + 1]) / 2
        t_param = (x_target - center_x) / step

        # Таблица конечных разностей
        diff_table = self.compute_finite_differences()

        # Начальное приближение: среднее значение в центре
        result = (self.y_values[center_idx] + self.y_values[center_idx + 1]) / 2

        # Добавляем первую разность первого порядка (нечётная)
        result += t_param * diff_table[center_idx][1]

        max_order = len(diff_table[0]) - 1  # Максимальный порядок разности

        for i in range(1, n // 2):
            even_order = 2 * i
            odd_order = 2 * i + 1

            # Проверка на выход за границы таблицы разностей
            if center_idx - i < 0 or center_idx - i + 1 >= n:
                break
            if even_order >= max_order:
                break

            # Чётные разности: центральные
            delta_even = diff_table[center_idx - i][even_order]
            product_even = 1
            for k in range(1, i + 1):
                product_even *= (t_param ** 2 - (k - 0.5) ** 2)
            term_even = delta_even * product_even / math.factorial(even_order)
            result += term_even

            # Проверка наличия нечётной разности
            if odd_order >= max_order:
                break

            # Нечётные разности: смещены на +1
            delta_odd = diff_table[center_idx - i + 1][odd_order]
            product_odd = t_param
            for k in range(1, i + 1):
                product_odd *= (t_param ** 2 - k ** 2)
            term_odd = delta_odd * product_odd / math.factorial(odd_order)
            result += term_odd

        return result

    def find_central_interval_index(self, x_target):
        """Поиск центрального интервала"""
        for i in range(len(self.x_nodes) - 1):
            if self.x_nodes[i] <= x_target <= self.x_nodes[i + 1]:
                return i
```

```
        return len(self.x_nodes) // 2 - 1

    def plot_interpolation(self, method='all', target_x=None):
        """Визуализация результатов интерполяции"""
        plt.figure(figsize=(12, 8))  # Создание холста 12x8 дюймов
        x_plot = np.linspace(min(self.x_nodes), max(self.x_nodes), 500)  # 500 точек для плавного графика

        plt.scatter(self.x_nodes, self.y_values, color='black', s=100, label='Узлы интерполяции', zorder=5)


        if method in ['all', 'lagrange']:
            y_lagrange = [self.interpolate_lagrange(x) for x in x_plot]
            plt.plot(x_plot, y_lagrange, color='#FF4500', label='Метод Лагранжа', linewidth=2)  # Оранжево-красный

        if method in ['all', 'newtone_div']:
            y_newton_div = [self.interpolate_newton_divided(x) for x in x_plot]
            plt.plot(x_plot, y_newton_div, '--', color='#1E90FF', label='Ньютон (раздел. разности)',
                     linewidth=2)  # Ярко-синий

        if method in ['all', 'newton_fin'] and len(self.x_nodes) > 1:
            step = abs(self.x_nodes[1] - self.x_nodes[0])
            if all(abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) < 1e-10 for i in range(len(self.x_nodes) - 1)):
                y_newton_fin = [
                    self.interpolate_newton_finite(x) if self.interpolate_newton_finite(x) is not None else np.nan for x
                    in x_plot]
                plt.plot(x_plot, y_newton_fin, ':', color='#32CD32', label='Ньютон (кон. разности)',
                         linewidth=2)  # Лаймовый

        if method in ['all', 'stirling'] and len(self.x_nodes) > 1:
            step = abs(self.x_nodes[1] - self.x_nodes[0])
            if all(abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) < 1e-10 for i in
                   range(len(self.x_nodes) - 1)) and len(self.x_nodes) % 2 != 0:# Для Стирлинга (нечетное)
                y_stirling = [self.interpolate_stirling(x) if self.interpolate_stirling(x) is not None else np.nan for x
                              in x_plot]
                plt.plot(x_plot, y_stirling, '-.', color='#9370DB', label='Стирлинг', linewidth=2)  # Фиолетовый

        if method in ['all', 'bessel'] and len(self.x_nodes) > 1:
            step = abs(self.x_nodes[1] - self.x_nodes[0])
            if all(abs((self.x_nodes[i + 1] - self.x_nodes[i]) - step) < 1e-10 for i in
                   range(len(self.x_nodes) - 1)) and len(self.x_nodes) % 2 == 0: # Для Бесселя (четное)
                y_bessel = [self.interpolate_bessel(x) if self.interpolate_bessel(x) is not None else np.nan for x in
                            x_plot]
                plt.plot(x_plot, y_bessel, ':', color='#FF69B4', label='Бессель', linewidth=2)  # Розовый

        if target_x is not None:
            plt.axvline(x=target_x, color='red', linestyle=':', alpha=0.5, label=f'x = {target_x}')
            methods = [
                ('lagrange', self.interpolate_lagrange(target_x), '#FF4500'),  # Оранжево-красный
                ('newtone_div', self.interpolate_newton_divided(target_x), '#1E90FF'),  # Ярко-синий
                ('newton_fin', self.interpolate_newton_finite(target_x), '#32CD32'),  # Лаймовый
                ('stirling', self.interpolate_stirling(target_x), '#9370DB'),  # Фиолетовый
                ('bessel', self.interpolate_bessel(target_x), '#FF69B4')  # Розовый
            ]

            for method_name, value, color in methods:
                if method in ['all', method_name] and value is not None:
                    plt.scatter([target_x], [value], color=color, s=100, zorder=5, edgecolors='black')  # Отображение точек
на графике

        plt.title('Сравнение методов интерполяции')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
```

```
        plt.grid(True)
        plt.show()

if __name__ == "__main__":
    main()
```