МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет» (ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ОТЧЁТ по лабораторной работе № 5-6

«Метод монотонной прогонки и интерполяция сплайнами»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт
Ф.И.О.
Ильяхова Алиса Алексеевна

Проверил: преподаватель
Ф.И.О.
Павленко Елизавета Робертовна

Владивосток

2024

Цель работы:

1. Реализовать генерацию сплайна благодаря методу моментов.
2. Сделать таблицу ошибок.
3. Построить график зависимости абсолютной ошибки от количества узлов.
4. Написать вывод о поведении ошибки.
5. Заключение.

Основное:

1.1. Данные:

$y = x^2 - \sin(x)$

[0.5, 1.0]

1.2. Реализация сплайна:

Сплайн представляет собой условную функцию, которая на каждом интервале между узлами принимает новые коэффициенты для кубического полинома. Чтобы определить эти коэффициенты, требуется получить три массива значений. Для их вычисления применяется **метод монотонной прогонки**.

```
def build_spline():
    c_coefficient_matrix = [0] + [h_values[i] / (h_values[i] +
h_values[i + 1]) for i in range(0, n - 1)] + [0]
    a_coefficient_matrix = [0] + [h_values[i + 1] / (h_values[i] +
h_values[i + 1]) for i in range(0, n - 1)] + [0]
    b_coefficient_matrix = [1] + [2] * (n - 1) + [1]

    rhs = [s_derivative_function(interval[0])] + [
        6 / (h_values[i] + h_values[i + 1]) * ((y_values[i + 1] -
y_values[i]) / h_values[i + 1] \
                                               - (y_values[i] -
y_values[i - 1]) / h_values[i]) for i in
        range(0, n - 1)] + [s_derivative_function(interval[1])]

    alpha = [-c_coefficient_matrix[0] / b_coefficient_matrix[0]]
    betta = [rhs[0] / b_coefficient_matrix[0]]

    moments = [s_derivative_function(interval[1])]

    for i in range(0, n - 1):
        alpha.append(-c_coefficient_matrix[i] / (alpha[i] *
```

```
a_coefficient_matrix[i] + b_coefficient_matrix[i]))
        betta.append((rhs[i] - betta[i] * a_coefficient_matrix[i])
/ (
                     alpha[i] * a_coefficient_matrix[i] +
b_coefficient_matrix[i]))

    for i in range(0, n - 1):
        moments.append(alpha[n - i - 1] * moments[i] + betta[n - i
- 1])
    moments.append(s_derivative_function(interval[0]));
    moments = moments[::-1]

    a = moments[:-1:]
    b = [(moments[i + 1] - moments[i]) / h_values[i + 1] for i in
range(0, n)]
    c = [(y_values[i + 1] - y_values[i]) / h_values[i + 1] -
h_values[i + 1] / 6 * (2 * moments[i] + moments[i + 1]) for
         i in
         range(0, n)]

    return [a, b, c]
```

Алгоритм:

Определение матриц коэффициентов:

1. Матрица коэффициентов a:
   - Определяется как отношение длины следующего отрезка к сумме длин текущего и следующего отрезков.
   - Начальные и конечные значения этой матрицы также равны нулю.
2. Матрица коэффициентов b:
   - Содержит единицы на первом и последнем элементах, а все промежуточные элементы равны двум.
3. Матрица коэффициентов c:
   - Рассчитывается как отношение длины текущего отрезка к сумме длин текущего и следующего отрезков.
   - Начальные и конечные значения этой матрицы устанавливаются в ноль.

Определение правой части уравнения:

4. Правая часть уравнения вычисляется на основе значений второй производной функции на границах интервала. Для внутренних узлов она определяется как

разница между отношениями разностей значений функции и длинами отрезков, умноженная на 6.

Инициализация и вычисление массивов α и β:

5. Массив α:
   Инициализируется как отношение первого элемента матрицы c к первому элементу матрицы b с отрицательным знаком.
6. Массив β:
   Инициализируется как отношение первого элемента правой части уравнения (RHS) к первому элементу матрицы b.

Вычисление моментов M:

7. Массив моментов инициализируется значением второй производной функции на правом конце интервала.

Прямой обход для вычисления α и β:

8. Для каждого узла (кроме первого и последнего) массивы α и β обновляются с использованием значений матриц коэффициентов a, b, c и RHS.

Обратный обход для вычисления моментов M:

9. Для каждого узла (кроме первого и последнего) массив моментов M обновляется на основе значений массивов α и β.

Вычисление коэффициентов a, b и c:

10. Коэффициенты a представляют собой все элементы массива моментов M, за исключением последнего.
11. Коэффициенты b вычисляются как разница между соседними элементами массива моментов M, деленная на длину соответствующих отрезков.
12. Коэффициенты c определяются как разница значений функции в соседних узлах, деленная на длину соответствующих отрезков, с корректировкой по длинам отрезков и значениям моментов.

Возврат коэффициентов:

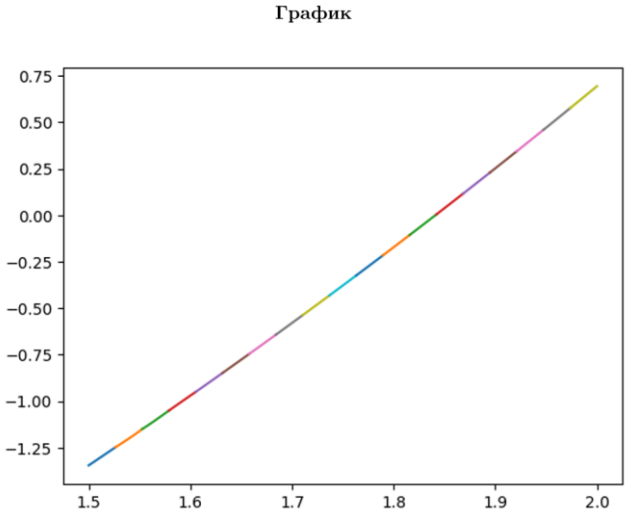
13. Функция возвращает массивы коэффициентов a, b и c, которые используются для расчета сплайна.

```
def evaluate_spline(x, i):
    return y_values[i] + spline_coefficients[2][i] * (x -
x_values[i]) + spline_coefficients[0][i] * (
            x - x_values[i]) ** 2 / 2 + spline_coefficients[1][i] *
```

```
(x - x_values[i]) ** 3 / 6

x_values = np.linspace(*interval, 20)
n = len(x_values) - 1
h_values = [abs(x_values[_] - x_values[_ - 1]) for _ in range(0, n
+ 1)]
y_values = [function(_) for _ in x_values]

spline_coefficients = build_spline()

for i in range(n):
    xl = np.linspace(x_values[i], x_values[i + 1], 10)
    yl = evaluate_spline(xl, i)

    plt.plot(xl, yl)
```

Алгоритм:

1. Создание сетки значений x:
   Формируется массив из 20 равномерно распределенных значений x на заданном диапазоне.
2. Вычисление длин отрезков h:
   Определяются длины каждого отрезка между узловыми точками.
3. Вычисление значений функции y:
   Рассчитываются значения функции f(x) для каждого узла.
4. Построение коэффициентов сплайна:
   Вызывается функция **build_spline** для вычисления коэффициентов сплайна.
5. Визуализация сплайна:
   Для каждого интервала рассчитываются значения сплайна и строится график на этом интервале.



1.3. Ошибки:

```
def norm(lst):
    return max(list(map(np.fabs, lst)))

ns = [3, 5, 10, 20, 30, 40, 55, 70, 85, 100]
max_deviations = []
relative_deviations = []

print("=" * 56)

for num_intervals in ns:

    spline_y_values = []
    x_values = np.linspace(*interval, num_intervals)
    n = num_intervals - 1
    h_values = [abs(x_values[_] - x_values[_ - 1]) for _ in
range(0, n + 1)]
    y_values = [function(_) for _ in x_values]

    spline_coefficients = build_spline()

    for i in range(n):
        xl = np.linspace(x_values[i], x_values[i + 1], 10)
        yl = evaluate_spline(xl, i)

        spline_y_values += [*yl]

    original_y_values = function(np.linspace(*interval, n * 10))

    spline_norm = norm(np.array(spline_y_values) -
original_y_values)
    function_norm = norm(original_y_values)

    max_deviations.append(spline_norm)
    relative_deviations.append(spline_norm / function_norm * 100)

    print(num_intervals, spline_norm, spline_norm / function_norm *
100, sep='\t')
```

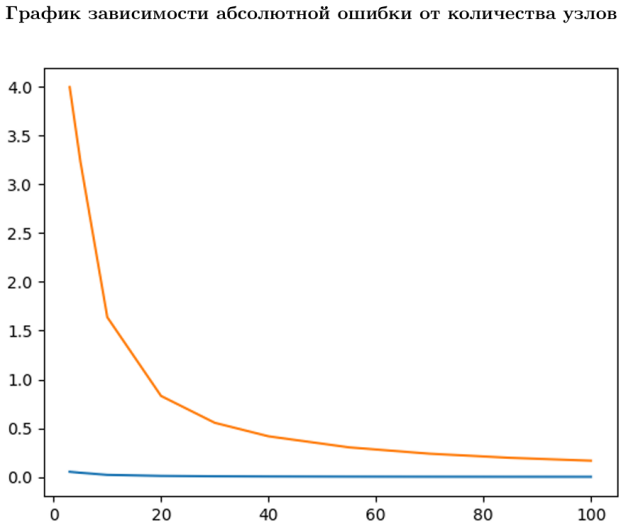
```
3   0.04865621454596836 1.5742768316912423
5   0.039074338372750095    1.264254241475544
10  0.021279500533230244    0.6885004308704817
20  0.010884971692150636    0.35218437990851975
30  0.007297107136738035    0.23609865278118666
40  0.005486266276262253    0.1775087102809625
55  0.003997413163833841    0.12933671452339984
70  0.003143965118726832 5  0.10172331514571493
85  0.002590764235711695 6  0.08382444361350633
100 0.002203089073864017 4  0.0712811738336027
```

(абсолютная и относительная отклонения)

Функция **norm(lst)** возвращает максимальное абсолютное значение из списка. Задаются количество интервалов **(ns)** и списки для отклонений. Далее выполняется цикл по количеству интервалов:

Для каждого значения **num_intervals**:

- Генерируется сетка x, вычисляются длины отрезков h и значения функции y.
- Строятся коэффициенты сплайна.
- Вычисляются значения сплайна и добавляются в список.
- Вычисляются значения оригинальной функции на более плотной сетке.
- Определяются максимальные и относительные отклонения.
- Отклонения добавляются в соответствующие списки.



Используемые библиотеки:

В ходе работы мне потребовалось использовать следующие библиотеки: numpy, pandas, matplotlib.

Библиотека **numpy** предоставляет поддержку для работы с многомерными массивами и матрицами, а также большое количество математических функций для выполнения операций над этими массивами.

Библиотека **pandas** нужна для обработки и анализа данных, предоставляющая удобные структуры данных и операции для их анализа.

Библиотека **matplotlib** нужна для создания статических, анимационных и интерактивных визуализаций в Python.

Вывод:
В заключение, проведенное исследование по аппроксимации функции с использованием кубических сплайнов подтвердило высокую эффективность данного метода. В процессе работы были выполнены ключевые этапы, включая разбиение интервала на различные числа интервалов, вычисление значений функции и её производных в узловых точках, построение коэффициентов сплайнов и оценку погрешности аппроксимации. Основные выводы подчеркивают, что с увеличением числа интервалов n наблюдается последовательное уменьшение как абсолютного (Δ), так и относительного (δ) отклонений, что свидетельствует о повышении точности аппроксимации. Наиболее значительное снижение отклонений фиксируется при малых значения n, тогда как при больших значениях эффект от дальнейшего увеличения числа интервалов становится менее выраженным. Эти результаты демонстрируют, что кубические сплайны являются надежным инструментом для точного представления функций, что делает их полезными в различных прикладных задачах, требующих высокой точности аппроксимации.

Полный код:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

interval = [1.5, 2.0]

def function(x):
    return x ** 2 - np.sin(x)

def f_derivative_function(x):
    return 2 * x - np.cos(x)

def s_derivative_function(x):
    return 2 + np.sin(x)
```

```
def build_spline():
    c_coefficient_matrix = [0] + [h_values[i] / (h_values[i] +
h_values[i + 1]) for i in range(0, n - 1)] + [0]
    a_coefficient_matrix = [0] + [h_values[i + 1] / (h_values[i] +
h_values[i + 1]) for i in range(0, n - 1)] + [0]
    b_coefficient_matrix = [1] + [2] * (n - 1) + [1]

    rhs = [s_derivative_function(interval[0])] + [
        6 / (h_values[i] + h_values[i + 1]) * ((y_values[i + 1] -
y_values[i]) / h_values[i + 1] \
                                               - (y_values[i] -
y_values[i - 1]) / h_values[i]) for i in
        range(0, n - 1)] + [s_derivative_function(interval[1])]

    alpha = [-c_coefficient_matrix[0] / b_coefficient_matrix[0]]
    betta = [rhs[0] / b_coefficient_matrix[0]]

    moments = [s_derivative_function(interval[1])]

    for i in range(0, n - 1):
        alpha.append(-c_coefficient_matrix[i] / (alpha[i] *
a_coefficient_matrix[i] + b_coefficient_matrix[i]))
        betta.append((rhs[i] - betta[i] * a_coefficient_matrix[i])
/ (
                     alpha[i] * a_coefficient_matrix[i] +
b_coefficient_matrix[i]))

    for i in range(0, n - 1):
        moments.append(alpha[n - i - 1] * moments[i] + betta[n - i
- 1])
    moments.append(s_derivative_function(interval[0]));
    moments = moments[::-1]

    a = moments[:-1:]
    b = [(moments[i + 1] - moments[i]) / h_values[i + 1] for i in
range(0, n)]
    c = [(y_values[i + 1] - y_values[i]) / h_values[i + 1] -
h_values[i + 1] / 6 * (2 * moments[i] + moments[i + 1]) for
         i in
         range(0, n)]
```

```
    return [a, b, c]

def evaluate_spline(x, i):
    return y_values[i] + spline_coefficients[2][i] * (x -
x_values[i]) + spline_coefficients[0][i] * (
            x - x_values[i]) ** 2 / 2 + spline_coefficients[1][i] *
(x - x_values[i]) ** 3 / 6

x_values = np.linspace(*interval, 20)
n = len(x_values) - 1
h_values = [abs(x_values[_] - x_values[_ - 1]) for _ in range(0, n
+ 1)]
y_values = [function(_) for _ in x_values]

spline_coefficients = build_spline()

for i in range(n):
    xl = np.linspace(x_values[i], x_values[i + 1], 10)
    yl = evaluate_spline(xl, i)

    plt.plot(xl, yl)

def norm(lst):
    return max(list(map(np.fabs, lst)))

ns = [3, 5, 10, 20, 30, 40, 55, 70, 85, 100]
max_deviations = []
relative_deviations = []

print("=" * 56)

for num_intervals in ns:

    spline_y_values = []
    x_values = np.linspace(*interval, num_intervals)
    n = num_intervals - 1
    h_values = [abs(x_values[_] - x_values[_ - 1]) for _ in
range(0, n + 1)]
    y_values = [function(_) for _ in x_values]

    spline_coefficients = build_spline()
```

```
    for i in range(n):
        xl = np.linspace(x_values[i], x_values[i + 1], 10)
        yl = evaluate_spline(xl, i)

        spline_y_values += [*yl]

    original_y_values = function(np.linspace(*interval, n * 10))

    spline_norm = norm(np.array(spline_y_values) -
original_y_values)
    function_norm = norm(original_y_values)

    max_deviations.append(spline_norm)
    relative_deviations.append(spline_norm / function_norm * 100)

    print(num_intervals, spline_norm, spline_norm / function_norm *
100, sep='\t')

plt.plot(ns, max_deviations)
plt.plot(ns, relative_deviations)
```