Отчет по лабораторной работе № 5 и №6
Вариант № 3

Винницкая Дина Сергеевна

Группа: Б9122-02-03-01сцт

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

Входные данные:

1. Функция: $f(x) = x^2 + \ln(x) - 4$
2. Отрезок: $[1.5; 2]$

Ход работы

Реализация сплайна

Сплайн — это условная функция, которая на каждом узловом отрезке принимает новые коэффициенты для кубического полинома. Для определения этих коэффициентов необходимо получить три массива значений. Для их вычисления используется **метод монотонной прогонки**.

Инициализация функции

```
def function(x):
    return x ** 2 + np.log(x) - 4


def f_derivative_function(x):
    return 2 * x + 1 / x


def s_derivative_function(x):
    return 2 - 1 / (x * x)
```

Задаем исходную функцию, ее первую и вторую производную. Первая производная функции $f(x)$ равна:

$$f'(x) = 2x + \frac{1}{x} \tag{1}$$

Вторая производная функции $f(x)$ равна:

$$f''(x) = 2 - \frac{1}{x^2} \tag{2}$$

Инициализация функции

```
1
2  def build_spline():
3      c_coefficient_matrix = [0] + [h_values[i] /
4      (h_values[i] + h_values[i + 1]) for i in range(0, n - 1)] + [0]
5      a_coefficient_matrix = [0] + [h_values[i + 1] / (h_values[i] +
6      h_values[i + 1]) for i in range(0, n - 1)] + [0]
7      b_coefficient_matrix = [1] + [2] * (n - 1) + [1]
8
9      rhs = [s_derivative_function(interval[0])] + [
10         6 / (h_values[i] + h_values[i + 1]) *
11         ((y_values[i + 1] - y_values[i]) / h_values[i + 1] \
12         - (y_values[i] - y_values[i -1]) / h_values[i]) for i in
13         range(0, n - 1)] + [s_derivative_function(interval[1])]
14
15     alpha = [-c_coefficient_matrix[0] / b_coefficient_matrix[0]]
16     betta = [rhs[0] / b_coefficient_matrix[0]]
17
18     moments = [s_derivative_function(interval[1])]
19
20     for i in range(0, n - 1):
21         alpha.append(-c_coefficient_matrix[i] / (alpha[i] *
22         a_coefficient_matrix[i] + b_coefficient_matrix[i]))
23         betta.append((rhs[i] - betta[i] * a_coefficient_matrix[i])
24         / (alpha[i] * a_coefficient_matrix[i] + b_coefficient_matrix[i]))
25
26     for i in range(0, n - 1):
27         moments.append(alpha[n - i - 1] * moments[i] + betta[n - i - 1])
28     moments.append(s_derivative_function(interval[0]));
29     moments = moments[::-1]
30
31     a = moments[:-1:]
32     b = [(moments[i + 1] - moments[i]) / h_values[i + 1]
33     for i in range(0, n)]
34     c = [(y_values[i + 1] - y_values[i])
35     / h_values[i + 1] - h_values[i + 1] / 6 *
36     (2 * moments[i] + moments[i + 1]) for i in range(0, n)]
37
38     return [a, b, c]
```

Описание алгортима

1. **Определение матриц коэффициентов**

   - Матрица коэффициентов c рассчитывается как отношение длины текущего отрезка к сумме длин текущего и следующего отрезков. Начальные и конечные значения этой матрицы равны нулю.
   - Матрица коэффициентов a рассчитывается как отношение длины следующего отрезка к сумме длин текущего и следующего отрезков. Начальные и конечные значения этой матрицы также равны нулю.
   - Матрица коэффициентов b состоит из единицы в начале и конце, а все промежуточные элементы равны двум.

2. **Определение правой части уравнения** Правая часть уравнения рассчитывается как значения второй производной функции на концах интервала, а для внутренних узлов — как разница отношений разностей значений функции и длин отрезков, умноженная на 6.

3. **Инициализация и вычисление массивов α и β**

   - Массив α инициализируется как отношение начального элемента матрицы c к начальному элементу матрицы b с отрицательным знаком.
   - Массив β инициализируется как отношение начального элемента RHS к начальному элементу матрицы b.

4. **Вычисление моментов** M Массив моментов инициализируется значением второй производной функции на правом конце интервала.

5. **Обход в прямом порядке для вычисления α и β** Для каждого узла (кроме первого и последнего) массивы α и β обновляются на основе значений матриц коэффициентов a, b, c и RHS.

6. **Обход в обратном порядке для вычисления моментов** M Для каждого узла (кроме первого и последнего) массив моментов M обновляется на основе значений массивов α и β.

7. **Вычисление коэффициентов a, b и c**

   - Коэффициенты a равны всем элементам массива моментов M, кроме последнего.
   - Коэффициенты b рассчитываются как разница соседних элементов массива моментов M, деленная на длину соответствующих отрезков.
   - Коэффициенты c рассчитываются как разница значений функции в соседних узлах, деленная на длину соответствующих отрезков, с корректировкой на основе длин отрезков и значений моментов.

8. **Возврат коэффициентов** Функция возвращает массивы коэффициентов a, b и c, которые используются для вычисления непосредственного сплайна.

Реализация 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: Генерируется массив из 20 равномерно распределенных значений x на заданном интервале.

2. **Вычисление длин отрезков** h: Вычисляются длины каждого отрезка между узлами.

3. **Вычисление значений функции** y: Вычисляются значения функции $f(x)$ в каждом узле.

4. **Построение коэффициентов сплайна**: Вызывается функция `build_spline` для вычисления коэффициентов сплайна.

5. **Визуализация сплайна**: Для каждого интервала вычисляются значения сплайна и строится график на данном интервале.

График

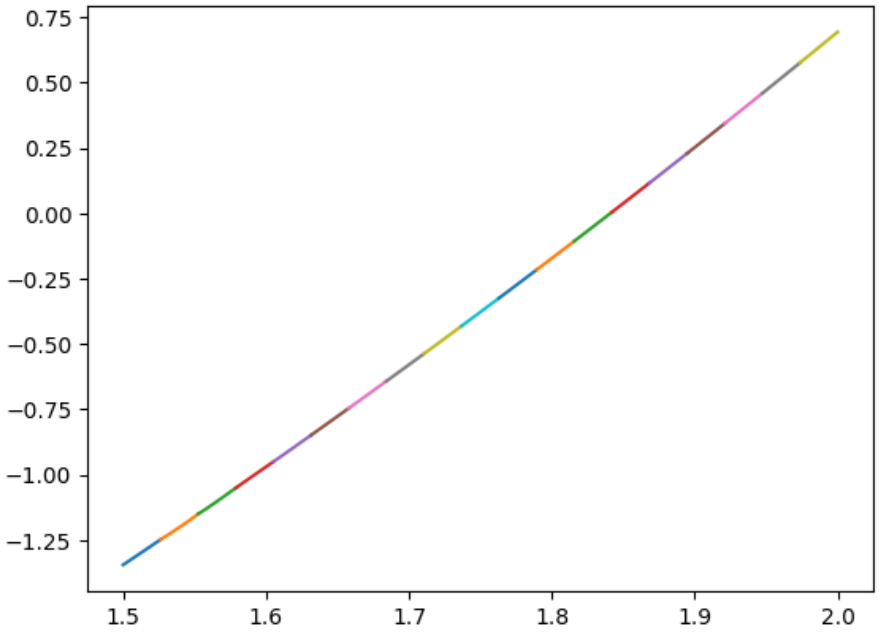

Рис. 1: График сплайна

Ошибки

Реализация

```
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 = []

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')
```

Алгоритм

`norm(lst)` возвращает максимальное абсолютное значение списка.
Задаются числа интервалов (`ns`) и списки для отклонений.
Цикл по числу интервалов

- Для каждого значения `num_intervals`:
  - Генерация сетки x, вычисление длин отрезков h и значений функции y.
  - Построение коэффициентов сплайна.
  - Вычисление значений сплайна и добавление их в список.
  - Вычисление значений оригинальной функции на плотной сетке.
  - Вычисление максимального и относительного отклонений.
  - Добавление отклонений в списки.

Таблица ошибок

n	Абсолютное отклонение	Относительное отклонение
3	0.05371643350201083	3.9951684278292565
5	0.04343490397994143	3.230478007069501
10	0.022010032911421007	1.6369997568788763
20	0.011168471008359937	0.8306568372238579
30	0.00746909969040388	0.555515497250833
40	0.0056090498036676095	0.417173986148873
55	0.004083074671390463	0.3036793389307547
70	0.00320965733824341	0.23871878354359727
85	0.002644010127457186	0.1966486807744287
100	0.002247842609951256	0.16718365759838455

Таблица 1: Таблица абсолютных и относительных отклонений

График зависимости абсолютной ошибки от количества узлов

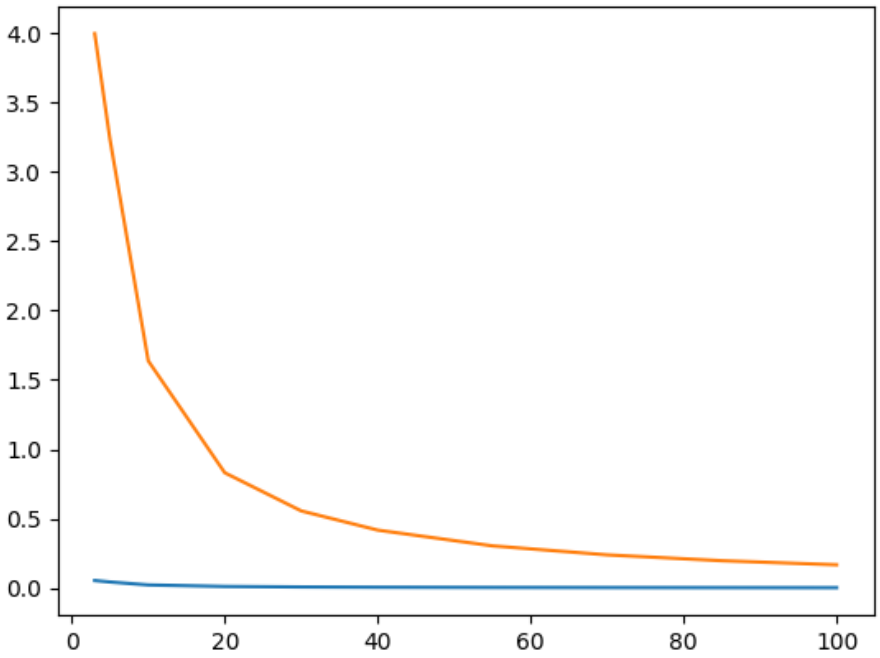

Рис. 2: График сплайна

Вывод о поведении ошибок

На основе приведенных данных можно сделать следующие выводы:

- **Уменьшение абсолютного отклонения (Δ)**: С увеличением количества интервалов n, абсолютное отклонение (Δ) последовательно уменьшается. Это свидетельствует о том, что аппроксимация сплайном становится точнее при увеличении числа интервалов.
- **Уменьшение относительного отклонения (δ)**: Аналогично, относительное отклонение (δ) также уменьшается с увеличением числа интервалов. Это показывает, что относительная погрешность аппроксимации снижается, делая аппроксимацию более точной по сравнению с исходной функцией.
- **Скорость уменьшения отклонений**: Как видно из данных, наиболее значительное снижение отклонений наблюдается при малых значениях n. При больших значениях n уменьшение отклонений становится менее выраженным. Это может свидетельствовать о том, что дальнейшее увеличение числа интервалов приводит к менее значимому улучшению точности аппроксимации.

В целом, результаты подтверждают, что увеличение числа интервалов n улучшает точность аппроксимации сплайном как в абсолютном, так и в относительном выражении.

Заключение

В данной лабораторной работе была проведена аппроксимация функции с использованием кубических сплайнов. Основные этапы включали:

1. Разбиение заданного интервала на различные числа интервалов.
2. Вычисление значений функции и её производных в узловых точках.
3. Построение коэффициентов кубических сплайнов.
4. Оценка погрешности аппроксимации путем вычисления максимальных и относительных отклонений.

На основе проведенных экспериментов и анализа полученных данных были сделаны следующие выводы:

1. **Точность аппроксимации**: С увеличением числа интервалов n максимальное отклонение (Δ) и относительное отклонение (δ) последовательно уменьшаются. Это свидетельствует о повышении точности аппроксимации кубическим сплайном при увеличении числа интервалов.
2. **Скорость сходимости**: Наиболее значительное снижение отклонений наблюдается при малых значениях n. При больших значениях n уменьшение отклонений становится менее выраженным, что указывает на уменьшение эффекта от дальнейшего увеличения числа интервалов.
3. **Практическое применение**: Аппроксимация кубическими сплайнами показала свою эффективность для точного представления функций. Метод может быть полезен в различных прикладных задачах, требующих высокой точности аппроксимации.