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

ОТЧЕТ по лабораторной работе № 5-6

«Интерполяция сплайнами и Метод монотонной прогонки»

Вариант № 10

Выполнил(а): студент гр. Б9122-02.03.01сцт
Кузнецов Е. Д.

Проверил: преподаватель
Павленко Е. Р.

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

Входные данные:

1. Функция $y = x^2 - cos(\pi x)$
2. Отрезок [0.1; 0.6]

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 вычисления которых используется метод монотонной прогонки.

```
def function(x):
    return x ** 2 - np.cos(np.pi * x)


def f_derivative_function(x):
    return 2*x + np.pi * np.sin(np.pi * x)


def s_derivative_function(x):
    return 2 + (np.pi ** 2) * np.cos(np.pi * x)
```

Задаём исходную функцию, а также её первую и вторую производные.

```
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]
```

Описание алгоритма

1. Определение матриц коэффициентов
    - Матрица “c” рассчитывается как отношение длины текущего отрезка к сумме длин текущего и следующего. Начальные и конечные значения 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.

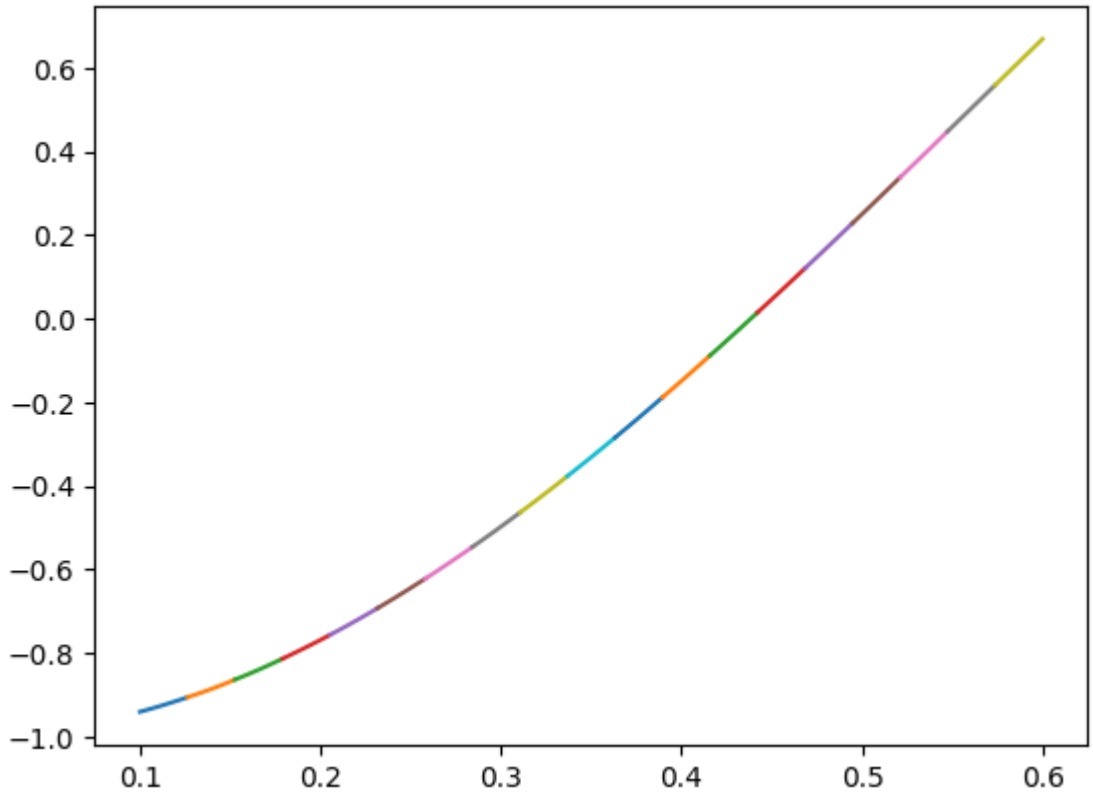
4. Вычисление моментов
   - “M“ Массив моментов инициализируется значением второй производной функции на правом конце интервала.
5. Обход в прямом порядке для вычисления α и β
   - Для каждого узла (кроме первого и последнего) массивы α и β обновляются на основе значений матриц коэффициентов a, b, c 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)
```

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 интеграла вычисляются значения сплайна и строится график на данном интервале.

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 = []

print("=" * 56)
```

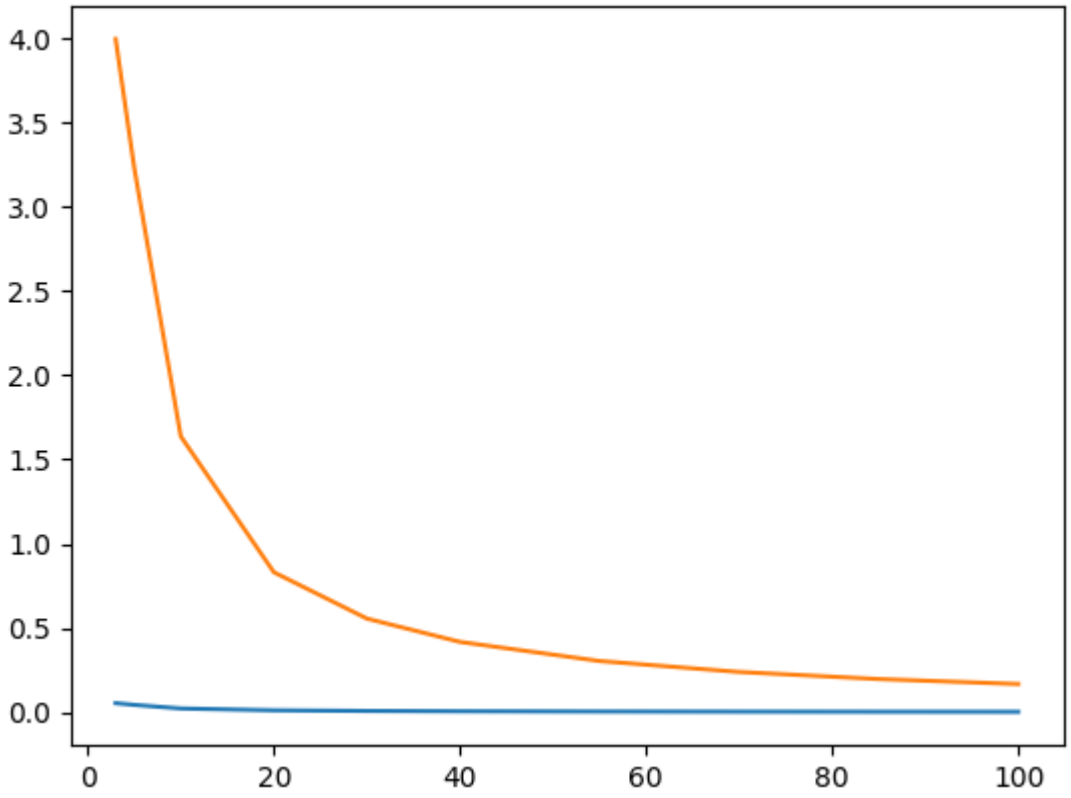
```
for num_intervals in ns:

    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')
```

1. `norm(lst)` возвращает максимальное абсолютное значение списка.
2. Задаются числа интервалов (ns) и списки для отклонений.
3. 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	0.054691487794947036	5.811711289164865
5	0.03937278720825485	4.183891883907422
10	0.020980495983169645	2.229461846326487
20	0.010538349114151302	1.1198423188906592
30	0.007017645264170036	0.7457198523844043
40	0.005258071090356475	0.5587412657272681
55	0.003820246473621336	0.4059529271059371
70	0.002999691384687475	0.31875783576707634
85	0.0024692364473677397	0.26238981449157583
100	0.0020981711077577847	0.2229590966563918

График зависимости абсолютной ошибки от кол-ва узлов



Вывод о поведении ошибок

На основе всех приведённых выше данных, можно сделать такие выводы:

- Уменьшение абсолютного отклонения (Δ). С увеличением количества интервалов n, абсолютное отклонение (Δ) последовательно уменьшается. Это свидетельствует о том, что при увеличении числа интервалов аппроксимация сплайном становится точнее.
- 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. Судя по данным, 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.

Результаты подтверждают, что увеличение числа интервалов *n* улучшает точность аппроксимации сплайном как в абсолютном, так и в относительном выражении.

Заключение

В данной 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 проведённых 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.