Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Вычислительной математике»

«Аппроксимация»

Выполнил:

Студент группы P32312

Лебедев В.В.

Преподаватель:

Перл О.В.

Санкт-Петербург

2023

Описание метода

Полином степени n - функция вида: $P_n(x) = c_0 + c_1 x + c_2 x^2 + \ \dots \ + c_n x^n$

Кубическая функция - полином степени 3, вида: $P_3(x) = a + bx + cx^2 + dx^3$

Задача интерполяции - нахождение промежуточных значений функции, заданной дискретным набором ее известных значений - *базовыми точками*.

Интерполяция кубическими сплайнами - один из методов интерполяции, при котором каждый отрезок, задаваемый базовыми точками, интерполируется кубической функцией. Причем так, чтобы в базовых точках значения смежных полиномов совпадали, а также совпадали их первая и вторая производная.

Численный метод интерполяции кубическими сплайнами строится на том, что известны условия на границах полиномов - в базовых точках.

Так, для произвольного полинома $p(x)$ известно, что:

1) $p(x_i) = y_i$
2) $p(x_{i+1}) = y_{i+1}$
3) $p'(x_i) = k_1$
4) $p'(x_{i+1}) = k_2$

Условия на концах интервала могут задаваться по-разному, но в случае “естественного сплайна” эти условия принимают вид:

$S''(a) = 0,\ S''(b) = 0$

Таким образом, зная граничные условия, можно восстановить коэффициенты всех полиномов внутри сплайна: $a,\ b,\ c,\ d$.

Это делается путем решения СЛАУ, где матрица коэффициентов принимает вид:

$$A = \begin{pmatrix} C_1 & B_1 & 0 & 0 & \cdots & 0 & 0 \\ A_2 & C_2 & B_2 & 0 & \cdots & 0 & 0 \\ 0 & A_3 & C_3 & B_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & B_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & A_n & C_n \end{pmatrix}$$

Где:

$A_i = h_i$;

$C_i = 2 \cdot (h_i + h_{i+1})$;

$B_i = h_{i+1}$;

$h_i = x_{i+1} - x_i$.

А столбец свободных членов: $B_i = 6(\frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}})$

После решения системы линейных уравнений мы получаем коэффициенты c_i, за счет которых можно восстановить остальные:

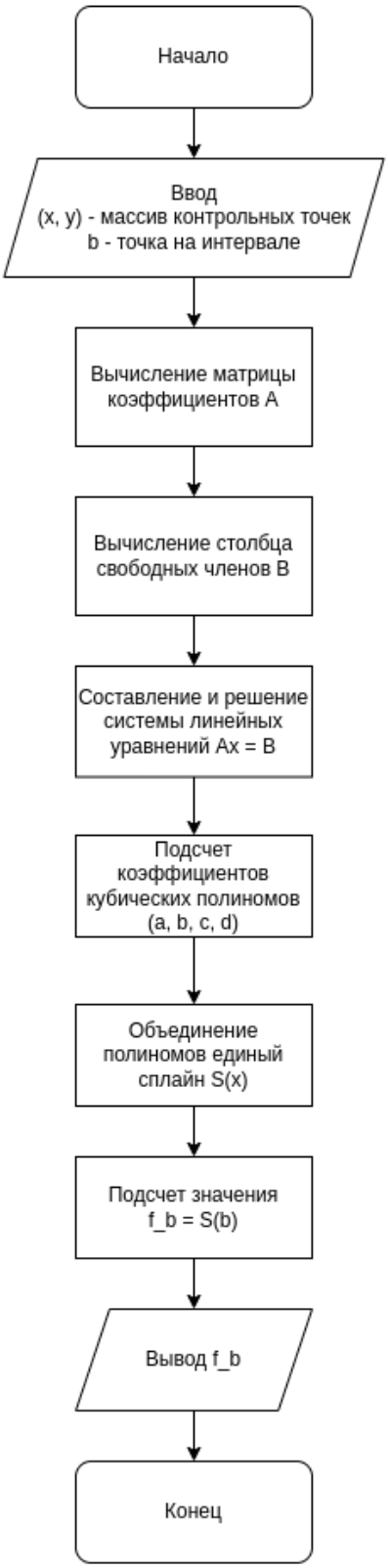
$a_i = y_i$

$d_i = \frac{c_{i+1}-c_i}{h_i}$

$b_i = \frac{c_{i+1}h_i}{2} - \frac{d_i h_i^2}{6} + \frac{y_{i+1}-y_i}{h_i}$

Зная все коэффициенты полиномов несложно собрать итоговый сплайн.

Блок схема численного метода



Листинг программы

```python
def calc_spline(points: list[tuple[float, float]]) -> Callable[[float], float]:
    x, y = zip(*points)
    n = len(points)
    h = calc_h(x)
    f = calc_constant_column(y, h)
    matrix = calc_coefficient_matrix(h)
    c = gauss_elimination(matrix, f)
    c = [0.0] + c + [0.0]
    a = y[1:]
    d = calc_d(h, c)
    b = calc_b(h, d, y, c)
    c = c[1:]
    spline_segments = [evaluate_spline_segment_function(x, i, a, b, c, d) for i in range(n - 1)]
    function = combine_spline_segments(spline_segments, x)
    return function
```

```python
def combine_spline_segments(spline_segments: list[Callable[[float], float]],
                            intervals: list[float]):
    def f(x: float):
        i = find_interval(intervals, x)
        return spline_segments[i](x)

    return f


def find_interval(interval_parts: list[float], x: float) -> int:
    for i in range(0, len(interval_parts) - 1):
        if interval_parts[i] ⩽ x ⩽ interval_parts[i + 1]:
            return i
    raise IndexError("Out of bound")
```

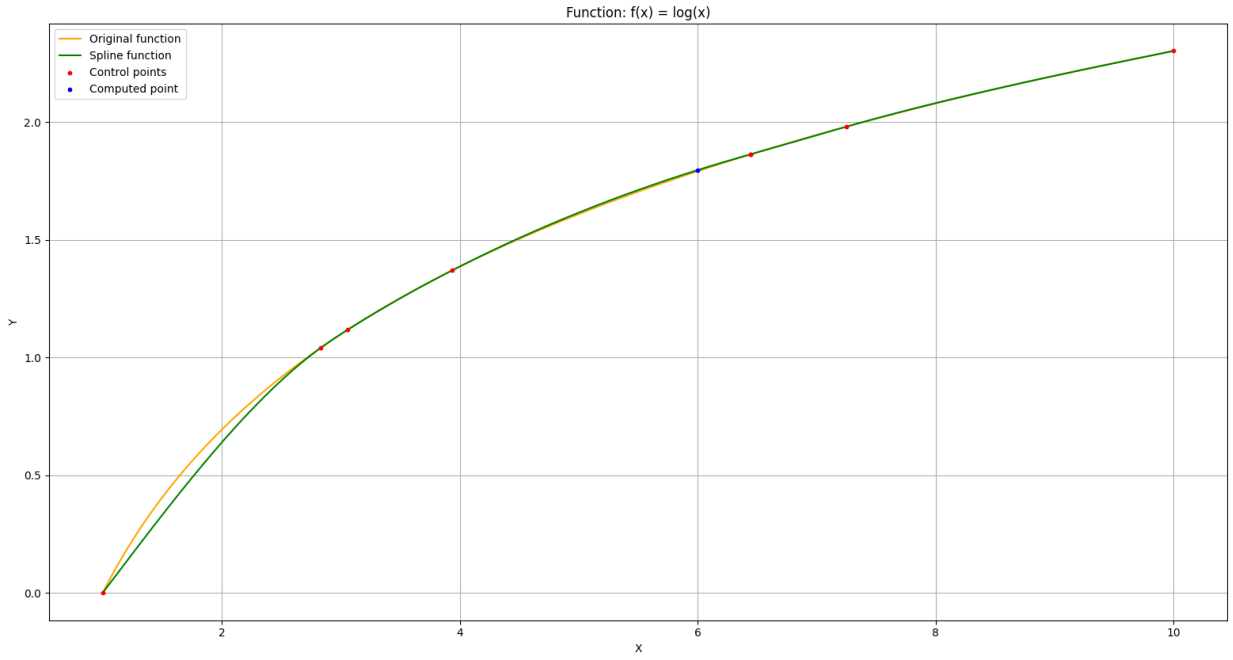
```python
def evaluate_spline_segment_function(
        intervals: list[float],
        index: int,
        a: list[float],
        b: list[float],
        c: list[float],
        d: list[float]
):
    def f(x: float) -> float:
        dx = x - intervals[index + 1]
        return a[index] + (b[index] * dx) + ((c[index] / 2) * dx * dx) + ((d[index] / 6) * dx * dx * dx)

    return f
```

```python
def calc_coefficient_matrix(h: list[float]) -> list[list[float]]:
    n = len(h) + 1
    matrix = [[0.0 for _ in range(n - 2)] for _ in range(n - 2)]
    for i in range(n - 2):
        a = h[i]
        b = h[i + 1]
        c = 2 * (h[i] + h[i + 1])
        matrix[i][i] = c
        if i < n - 3:
            matrix[i][i + 1] = b
        if i > 0:
            matrix[i][i - 1] = a
    return matrix


def calc_constant_column(f: list[float], h: list[float]) -> list[float]:
    result = []
    n = len(f)
    for i in range(1, n - 1):
        result.append(6 * (((f[i + 1] - f[i]) / h[i]) - ((f[i] - f[i - 1]) / h[i - 1])))
    return result
```

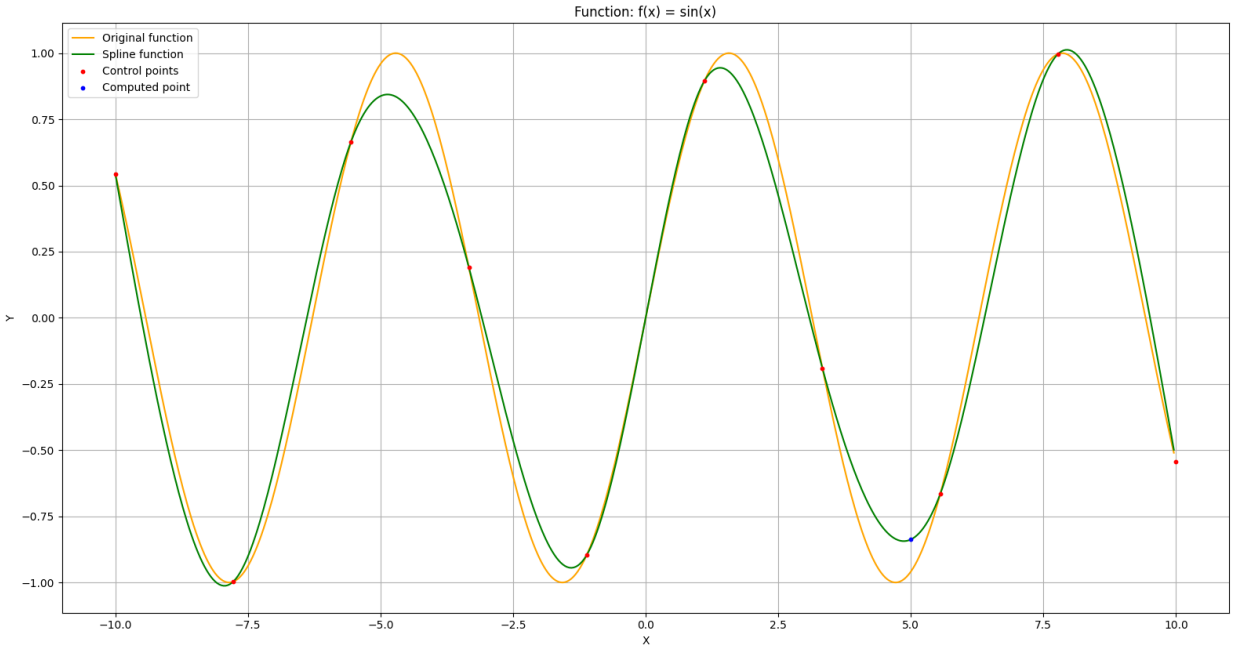
```python
def calc_h(x: list[float]) -> list[float]:
    return [x[i + 1] - x[i] for i in range(len(x) - 1)]


def calc_d(h: list[float], c: list[float]) -> list[float]:
    n = len(c)
    d = [0.0] * (n - 1)
    for i in range(n - 1):
        d[i] = (c[i + 1] - c[i]) / h[i]
    return d


def calc_b(h: list[float], d: list[float], f: list[float], c: list[float]) -> list[float]:
    n = len(f)
    b = [0.0] * (n - 1)
    for i in range(n - 1):
        b[i] = ((1 / 2) * h[i] * c[i + 1]) - ((1 / 6) * h[i] * h[i] * d[i]) + ((f[i + 1] - f[i]) / h[i])
    return b
```

Примеры

```
Functions:
1) f(x) = log(x)
2) f(x) = sin(x)
Enter function number [1 ... 2]: 1
Enter lower bound of the interval [-1000.0 ... 1000.0]: 1
Enter upper bound of the interval [1.0 ... 1000.0]: 10
Enter number of control points [2 ... 1000]: 8
Add noise? [yes, no]: yes
Enter noise deviation [0.0 ... 1000.0]: 0.0001
Enter the point you are interested in [1.0 ... 10.0]: 6
Computed value for point 6.0: 1.7965715354351928
```



```
Functions:
1) f(x) = log(x)
2) f(x) = sin(x)
Enter function number [1 ... 2]: 2
Enter lower bound of the interval [-1000.0 ... 1000.0]: -10
Enter upper bound of the interval [-10.0 ... 1000.0]: 10
Enter number of control points [2 ... 1000]: 10
Add noise? [yes, no]: no
Enter the point you are interested in [-10.0 ... 10.0]: 5
Computed value for point 5.0: -0.8375708419455715
```



Вывод

Асимптотическая сложность:

Скорость работы метода напрямую зависит от используемого метода решения СЛАУ и количества базовых точек.

В данной работе был применен *метод Гаусса с выбором главного элемента*, асимптотическая сложность которого $O(n^3)$. Поэтому итоговая сложность метода, если принимать количество точек за n будет тоже $O(n^3)$.

Если же использовать рекомендуемый при решении систем уравнений с трехдиагональными матрицами *метод Прогонки,* сложность которого $O(n)$, то можно добиться итоговой сложности $O(n)$.

Сравнение с другими методами и применимость:

Использование полиномиальных методов (Лагранжа и Ньютона) может дать более точный ответ, чем метод Сплайнов, в случае если исходная функция имеет более высокий порядок гладкости, чем кубическая функция.

Но при возрастании числа контрольных точек полиномиальные методы могут порождать осцилляции на концах интервала из-за *феномена Рунге*.

А также лучше сглаживает шумы во входных данных, чем полиномиальные методы.