Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

О т ч е т

по домашнему заданию

Вычисление и построение графика классического кубического сплайна

Дисциплина: Технологии мультимедиа

Студент гр. ИУ5-63Б _____ Терентьев В.О.
(Подпись, дата) (Фамилия И.О.)

Преподаватель _____ Афанасьев Г.И.
(Подпись, дата) (Фамилия И.О.)

Москва, 2021

1. Описание задания

Разработать на языке Python интерактивную программу вычисления и построения графика классического кубического сплайна на наборе из 10 контрольных точек с отображением уравнений сплайна.

2. Описание алгоритма работы программы

На каждом отрезке $[x_i, x_{i+1}], i = \overline{1, N-1}$, где N – количество контрольных точек, нужно определить коэффициенты полинома $S_i(x)$ третьей степени вида:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Полиномы на каждом отрезке должны проходить через две контрольные точки: x_i и $x_{i+1}, i = \overline{1, N-1}$. Отсюда получим 2 условия:

$$y_i = a_i + b_i(x_i - x_i) + c_i(x_i - x_i)^2 + d_i(x_i - x_i)^3 = a_i \quad (1)$$

$$y_{i+1} = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 \quad (2)$$

В стыках между полиномами должна обеспечиваться гладкость, соответственно в узлах должны быть одинаковые первые и вторые производные соседних многочленов. Отсюда получим еще 2 условия:

$$b_i + 2c_i(x_{i+1} - x_i) + 3d_i(x_{i+1} - x_i)^2 = b_{i+1} \quad (3)$$

$$2c_i + 6d_i(x_{i+1} - x_i) = 2c_{i+1} \quad (4)$$

Из определения классического кубического сплайна следует, что вторые производные в граничных точках равны 0. Отсюда получим 2 граничных условия:

$$c_1 = 0 \quad (5)$$

$$2c_N + 6d_N(x_N - x_{N-1}) = 0 \quad (6)$$

Обозначив $x_{i+1} - x_i = h_i$ и преобразовав систему уравнений (1)-(4), получим:

$$c_{i-1}h_{i-1} + 2c_i(h_{i-1} + h_i) + c_{i+1}h_i = 3\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right) \quad (7)$$

Это уравнение решается методом прогонки (алгоритмом Томаса) с учетом граничных условий. Уравнение (7) равносильно уравнению:

$$A_i c_{i-1} + B_i c_i + C_i c_{i+1} = F_i,$$

где $A_i = h_{i-1}$, $B_i = 2(h_{i-1} + h_i)$, $C_i = h_i$, $F_i = 3\left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right)$.

Найдем прогоночные коэффициенты по формулам:

$$\alpha_1 = \frac{-C_1}{B_1}$$

$$\beta_1 = \frac{F_1}{B_1}$$

$$\alpha_i = \frac{-C_i}{A_i\alpha_{i-1} + B_i}$$

$$\beta_i = \frac{F_i - A_i\beta_{i-1}}{A_i\alpha_{i-1} + B_i}$$

После нахождения прогоночных коэффициентов α и β, с учетом граничных условий (5) и (6), решаем уравнение (7), используя следующую формулу:

$$c_i = \alpha_i c_{i+1} + \beta_i$$

После нахождения коэффициентов c_i, используя систему уравнений (1)-(4), найдем значения оставшихся коэффициентов a_i, b_i и d_i полиномов $S_i(x)$.

После нахождения всех коэффициентов построим и выведем график классического кубического сплайна на наборе из контрольных точек, а также преобразованные уравнения полиномов на каждом отрезке.

3. Текст программы

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

class CubicSpline:

    def __init__(self, x, y):
        indices = sorted(range(len(x)), key=lambda i: x[i])
        x = [x[i] for i in indices]
        y = [y[i] for i in indices]
        self.bounds = x

        self.cnt_spls = len(self.bounds)-1 # = n

        self.pol = [[0] * 4 for i in range(self.cnt_spls)]
        self.fin_pol = [[0] * 4 for i in range(self.cnt_spls)]

        a = []
        a.append(0)
        bt = 2*(x[2]-x[0])
        a.append(-(x[2]-x[1])/bt)
        b = []
        b.append(0)
        b.append(3*((y[2]-y[1])/(x[2]-x[1])-(y[1]-y[0])/(x[1]-x[0]))/bt)
        for i in range(2, self.cnt_spls-1):
            at = x[i]-x[i-1]
            bt = 2*(x[i+1]-x[i-1])
            a.append(-(x[i+1]-x[i])/(at*a[i-1]+bt))
            b.append((3*((y[i+1]-y[i])/(x[i+1]-x[i])-(y[i]-y[i-1])/(x[i]-x[i-
1])) - at*b[i-1])/(at*a[i-1]+bt))

        self.pol[self.cnt_spls-1][2] = (3*((y[self.cnt_spls]-y[self.cnt_spls-
1])/(x[self.cnt_spls]-x[self.cnt_spls-1])-(y[self.cnt_spls-1]-y[self.cnt_spls-
2])/(x[self.cnt_spls-1]-x[self.cnt_spls-2])) - (x[self.cnt_spls-1]-
x[self.cnt_spls-2])*b[self.cnt_spls-2])/((x[self.cnt_spls-1]-x[self.cnt_spls-
2])*a[self.cnt_spls-2]+(2*(x[self.cnt_spls]-x[self.cnt_spls-2])))
        self.pol[self.cnt_spls-1][0] = y[self.cnt_spls-1]
        self.pol[self.cnt_spls-1][3] = -(self.pol[self.cnt_spls-
1][2])/(3*(x[self.cnt_spls]-x[self.cnt_spls-1]))
        self.pol[self.cnt_spls-1][1] = (y[self.cnt_spls]-y[self.cnt_spls-
1])/(x[self.cnt_spls]-x[self.cnt_spls-1]) - self.pol[self.cnt_spls-
1][2]*(x[self.cnt_spls]-x[self.cnt_spls-1]) - self.pol[self.cnt_spls-
1][3]*(x[self.cnt_spls]-x[self.cnt_spls-1])*(x[self.cnt_spls]-x[self.cnt_spls-1])
```

```
        self.fin_pol[self.cnt_spls-1][0] = self.pol[self.cnt_spls-1][0] -
 self.pol[self.cnt_spls-1][1]*x[self.cnt_spls-1] + self.pol[self.cnt_spls-
1][2]*x[self.cnt_spls-1]*x[self.cnt_spls-1] - self.pol[self.cnt_spls-
1][3]*x[self.cnt_spls-1]*x[self.cnt_spls-1]*x[self.cnt_spls-1]
        self.fin_pol[self.cnt_spls-1][1] = self.pol[self.cnt_spls-1][1] -
 self.pol[self.cnt_spls-1][2]*2*x[self.cnt_spls-1] + self.pol[self.cnt_spls-
1][3]*3*x[self.cnt_spls-1]*x[self.cnt_spls-1]
        self.fin_pol[self.cnt_spls-1][2] = self.pol[self.cnt_spls-1][2] -
 self.pol[self.cnt_spls-1][3]*3*x[self.cnt_spls-1]
        self.fin_pol[self.cnt_spls-1][3] = self.pol[self.cnt_spls-1][3]

        for i in range(self.cnt_spls-2, 0, -1):
            self.pol[i][2] = a[i]*self.pol[i+1][2]+b[i]

        self.pol[0][2] = 0

        for i in range(0, self.cnt_spls-1):
            self.pol[i][0] = y[i]
            self.pol[i][1] = (y[i+1]-y[i])/(x[i+1]-x[i]) -
 (2*self.pol[i][2]+self.pol[i+1][2])*(x[i+1]-x[i])/3
            self.pol[i][3] = (self.pol[i+1][2]-self.pol[i][2])/(3*(x[i+1]-x[i]))

            self.fin_pol[i][0] = self.pol[i][0] -
 self.pol[i][1]*x[i] + self.pol[i][2]*x[i]*x[i] - self.pol[i][3]*x[i]*x[i]*x[i]
            self.fin_pol[i][1] = self.pol[i][1] -
 self.pol[i][2]*2*x[i] + self.pol[i][3]*3*x[i]*x[i]
            self.fin_pol[i][2] = self.pol[i][2] - self.pol[i][3]*3*x[i]
            self.fin_pol[i][3] = self.pol[i][3]


    def __call__(self, xa):
        y = []
        for x in xa:
            flg = True
            if x <= self.bounds[0]:
                y.append(self.fin_pol[0][3]*x*x*x + self.fin_pol[0][2]*x*x + self
.fin_pol[0][1]*x + self.fin_pol[0][0])
                continue
            for i in range(1, self.cnt_spls+1):
                if x <= self.bounds[i]:
                    y.append(self.fin_pol[i-1][3]*x*x*x + self.fin_pol[i-
1][2]*x*x + self.fin_pol[i-1][1]*x + self.fin_pol[i-1][0])
                    flg = False
                    break
            if flg:
                y.append(self.fin_pol[self.cnt_spls-
1][3]*x*x*x + self.fin_pol[self.cnt_spls-1][2]*x*x + self.fin_pol[self.cnt_spls-
1][1]*x + self.fin_pol[self.cnt_spls-1][0])
        return y
```

```
x = [int(x) for x in input('Введите значения по x: ').split()]
y = [int(x) for x in input('Введите значения по y: ').split()]

indices = sorted(range(len(x)), key=lambda i: x[i])
x = [x[i] for i in indices]
y = [y[i] for i in indices]

cs = CubicSpline(x, y)

fig, ax = plt.subplots()

xs = np.arange(x[0], x[1], 0.001)
ax.plot(xs, cs(xs), label='y={:.3f}x\u00B3{:+.3f}x\u00B2{:+.3f}x{:+.3f}, x\u2208[
{}, {}]'.format(cs.fin_pol[0][3], cs.fin_pol[0][2], cs.fin_pol[0][1], cs.fin_pol[
0][0], x[0], x[1]))

for i in range(1, len(x)-1):
    xs = np.arange(x[i], x[i+1], 0.001)
    ax.plot(xs, cs(xs), label='y={:.3f}x\u00B3{:+.3f}x\u00B2{:+.3f}x{:+.3f}, x\u2
208({}, {}]'.format(cs.fin_pol[i][3], cs.fin_pol[i][2], cs.fin_pol[i][1], cs.fin_
pol[i][0], x[i], x[i+1]))

plt.plot(x, y, 'o', label='Контрольные точки')

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

4. Примеры выполнения программы