Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №3

Тема Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

Студент Бугаенко А.П.

Группа ИУ7-45Б

Оценка (баллы) ________________

Преподаватель Градов В.М.

Москва.
2021 г

Цель работы. Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

1 Исходные данные

11. Таблица функции с количеством узлов N. Задать с помощью формулы $y = x^2$ в диапазоне [0..10] с шагом 1.

x	y
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

12. Значение аргумента x в первом интервале, например, при x=0.5 и в середине таблицы, например, при x= 5.5. Сравнить с точным значением.

2 Код программы

Листинг newton.py

```
import math as m

def Divided_diff(x, y):
    new_y = []
    n = len(x) - len(y)
    for i in range(0, len(y) - 1):
        new_y.append((y[i] - y[i + 1]) / (x[i] - x[i + n + 1]))
    return new_y

def SortTableNewton(table, x, n):
    table = sorted(table, key=lambda d: abs(d["x"] - x))
    table = sorted(table[:n+1], key=lambda t: t["x"])
    return table

def FormXYNewton(table):
    X = []
    Y = []
    for row in table:
        X.append(row["x"])
        Y.append(row["y"])
    return X, Y
```

```
def CountDiffDivNewton(X, Y):
    Y_arr = [Y]
    while len(Y_arr[-1]) != 1:
        Y = Divided_diff(X, Y)
        Y_arr.append(Y)
    return Y_arr

def CountPolynomNewton(Y_arr):
    polym = []
    for Y in Y_arr:
        polym.append(Y[0])
    return polym

def GetValApproxNewton(polym, X, x):
    x_mult = 1
    result = 0
    for i in range(0, len(polym)):
        result += x_mult * polym[i]
        x_mult = x_mult * (x - X[i])
    return result

# x - value, n - power, table - newton data table
def NewtonApprox(x, n, table):

    table = SortTableNewton(table, x, n)
    X, Y = FormXYNewton(table)

    Y_arr = CountDiffDivNewton(X, Y)
    polym = CountPolynomNewton(Y_arr)

    return GetValApproxNewton(polym, X, x)
```

Листинг main.py

```
import numpy as np
import matplotlib as mp
import math as m
import newton as nw

def func(x):
    return x*x

# Retunrs table x - first col, y - second col
def createTable(tabLen):
    returnTable = []
    x = np.arange(0, tabLen, 1)
    y = np.array([func(xi) for xi in x])
    table = np.array([x, y])
    table = np.array(table.T)
    for i in range(0, tabLen):
        returnTable.append({"x": table[i][0], "y": table[i][1]})
    table = np.array(returnTable)
    return table
```

```
def interpolateQubic(table):
    N = len(table)

    e = [0, 0]
    n = [0, 0]
    # count e, n
    for i in range(2, N):
        h_i = table[i]["x"] - table[i - 1]["x"]
        h_im = table[i - 1]["x"] - table[i - 2]["x"]

        f_i = 3 * ((table[i]["y"] - table[i - 1]["y"]) / h_i - (table[i - 1]["y"] - table[i - 2]["y"]) / h_im)

        e.append(- h_i / (h_im * e[i - 1] + 2 * (h_im + h_i)))

        n.append((f_i - h_i * n[i - 1]) / (h_im * e[i - 1] + 2 * (h_im + h_i)))

    c = [0] * (N - 1)
    c[N - 2] = n[-1]

    for i in range(N - 2, 0, -1):
        c[i - 1] = e[i] * c[i] + n[i]

    a = []
    b = []
    d = []

for i in range(1, N):
        a.append(table[i - 1]["y"])

    for i in range(1, N - 1):
        h_i = table[i]["x"] - table[i - 1]["x"]

        b.append((table[i]["y"] - table[i - 1]["y"]) / h_i - h_i * (c[i] + 2 * c[i - 1]) / 3)
        d.append((c[i] - c[i - 1]) / (3 * h_i))

    h_N = table[-1]["x"] - table[-2]["x"]
    b.append((table[-1]["y"] - table[-2]["y"]) / h_N - h_N * 2 * c[-1] / 3)
    d.append(-c[-1] / (3 * h_N))

    koefs = [a, b, c, d]

    return koefs

def countX(x, table, koefs):
    for i in range(1, len(table)):
        if x >= table[i - 1]["x"] and x <= table[i]["x"]:
            dx = x - table[i - 1]["x"]
            return koefs[0][i - 1] + koefs[1][i - 1] * dx + koefs[2][i - 1] * dx * dx + koefs[3][i - 1] * dx * dx * dx

table = createTable(11)

koefs = interpolateQubic(table)
```

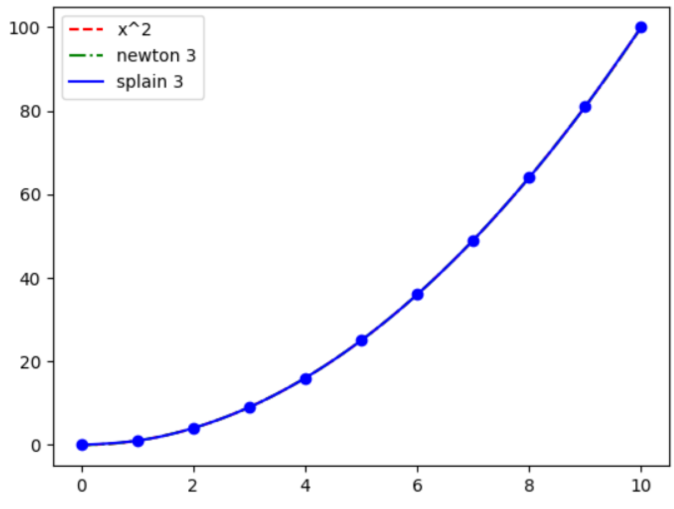
```
x_1 = 0.5
y_1 = func(x_1)
y_interp_1 = countX(x_1, table, koefs)
y_newton_1 = nw.NewtonApprox(x_1, 3, table)

x_2 = 5.5
y_2 = func(x_2)
y_interp_2 = countX(x_2, table, koefs)
y_newton_2 = nw.NewtonApprox(x_2, 3, table)

print(x_1, y_1, y_interp_1, y_newton_1)
print(x_2, y_2, y_interp_2, y_newton_2)

from numpy import linspace
from matplotlib import pyplot as plt
X = np.arange(0, 11, 1)
Y = np.array([func(xi) for xi in X])
X_ext = linspace(min(X), max(X), 100)
Y_ext = X_ext ** 2
Y_newton = [nw.NewtonApprox(x, 3, table) for x in X_ext]
Y_interp = [countX(x, table, koefs) for x in X_ext]
plt.plot(X_ext, Y_ext, 'r--', label="x^2")
plt.plot(X_ext, Y_newton, 'g-.', label="newton 3")
plt.plot(X_ext, Y_interp, 'b-', label="splain 3")
plt.plot(X, Y, 'bo')
plt.legend()
plt.show()
```

3

1. Результат работы алгоритма.

x	y исходный	y Сплайнами	y Ньютоном
0.5	0.25	0.34150	0.25
5.5	30.25	30.25034	30.25

2. Сравнение интерполяции сплайнами и интерполяции по Ньютону



Как мы можем увидеть на данном графике - интерполяция сплайнами практически совпадает с изначальной функцией, однако, как мы увидели в задании 1, имеется некоторая погрешность, в отличие от полинома Ньютона.

4 Контрольные вопросы

Как мы можем увидеть на данном графике - интерполяция сплайнами практически совпадает с изначальной функцией, однако, как мы увидели в задании 1, имеется некоторая погрешность, в отличие от полинома Ньютона.

1. Получить выражения для коэффициентов кубического сплайна, построенного на двух точках.

Пусть у нас есть две точки (x_1, y_1) и (x_2, y_2). Нам необходимо найти коэффициенты для выражения следующего вида:

$$\psi(x) = a + b(x - x_1) + c(x - x_1)^2 + d(x - x_1)^3$$

$$\psi(x_1) = y_1, \psi(x_2) = y_2$$

$$a = y_1 \ (1)$$

$$y_2 = a + b(x_2 - x_1) + c(x_2 - x_1)^2 + d(x_2 - x_1)^3 \ (2)$$

Возьмём вторые производные на концах участка равными 0, тогда:

$$\psi(x_1)'' = 0 \Rightarrow c = 0 \ (3)$$

$$\psi(x_2)'' = 0 \Rightarrow 2c + 6d(x_2 - x_1) = 0 \ (4)$$

Теперь число уравнений позволяет найти все коэффициенты, из (3) и (4) очевидно, что c = 0 и d = 0. Тогда подставим c и d в (2) для того, чтобы найти b:

$$y_2 = y_1 + b(x_2 - x_1) + 0(x_2 - x_1)^2 + 0(x_2 - x_1)^3$$

$$y_2 = y_1 + b(x_2 - x_1) \Rightarrow b = \frac{y_2 - y_1}{x_2 - x_1}$$

В итоге получаем коэффициенты:

$$a = y_1, b = \frac{y_2 - y_1}{x_2 - x_1}, c = 0, d = 0$$

Тогда у нас получится уравнение вида:

$$\psi(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

Таким образом, функция выродилась в прямую, соединяющую две точки.

2. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках

Так как точек три, будет два отрезка, для которых нужен будет отдельный набор коэффициентов. Поэтому у нас будет два полинома с общим количеством коэффициентов равным 8.

Значения в узлах: $\psi(x_i) = y_i, \psi(x_{i+1}) = y_{i+1}, i = \overline{1,2}$

$a_i = y_i$, для $i = \overline{1,2}$ $(1,2)$

$$y_{i+1} = a_i + b_i(x_{i+1} - x_i) + c_i(x_{i+1} - x_i)^2 + d_i(x_{i+1} - x_i)^3 \ (3, 4)$$

Условие для первых производных:

$$\psi_1(x_2)' = \psi_2(x_2)'$$

$$b_1 + 2c_1(x_2 - x_1) + 3d_1(x_2 - x_1)^2 = b_2 \ (5)$$

Условие для вторых производных:

$$\psi_1(x_2)'' = \psi_2(x_2)''$$

$$c_1 + 3d_1(x_2 - x_1) = c_2 \ (6)$$

Условия на концах области:

$$\psi(x_1)'' = 0 \Rightarrow c_1 = 0 \ \ (7)$$

$$\psi(x_2)'' = 0 \Rightarrow c_2 + 3d_2(x_2 - x_1) = 0 \ (8)$$

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо $C_1=C_2$.

Для прогоночных коэффициентов сплайна будет справедливо следующее выражение:

$$c_i = \xi_{i+1} c_{i+1} + \eta_{i+1}$$

Поэтому если $c_1=c_2$, то начальные значения прогоночных коэффициентов будут равны:

$$\xi_2 = 1, \eta_2 = 0$$

4. Написать формулу для определения последнего коэффициента сплайна CN , чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано kCN-1+mCN=p, где k,m и p - заданные числа.

$$c_i = \xi_{i+1} c_{i+1} + \eta_{i+1}$$

Следовательно, имеем:

$$c_N = \xi_{N+1} c_{N+1} + \eta_{N+1}$$

Теперь рассмотрим граничное условие:

$$kc_{N-1} + mc_N = p$$

Из чего мы можем вывести следующее выражение:

$$k(\xi_N c_N + \eta_N) + mc_N = p$$

Тогда c_N выражается, как:

$$c_N = \frac{p - k\eta_N}{k\xi_N + m}$$