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 Мицевич Максим

Группа ИУ7-41Б

Оценка (баллы) _____

Преподаватель Градов В.М.

Москва.
2021 г

Цель работы. Получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

1 Исходные данные

1. Таблица функции с количеством узлов N. Задать с помощью формулы y = x^2 в диапазоне [0..10]с шагом 1.
2. Значение аргумента xв первом интервале, например при x=0.5 и в середине таблицы, например, при x=5.5. Сравнить с точным значением.

2 Код программы

Код программы представлен на листингах 1-2.

Листинг 1. newton.py

```
from math import fabs

class Newton_polynom():
    def __init__(self, table, power, arg):
        self.power = power
        self.arg = arg
        self.table = table.copy()
        self.diff = []

    def find_nearest(self):
        self.table.sort(key = lambda x: fabs(x[0] -
self.arg))

    def find_diff(self):
        self.diff.append([x[1] for x in self.table])
        for i in range(self.power - 1, -1, -1):
            diff = []
            for j in range(0, i + 1):
```

```
                x = (self.diff[-1][j] - self.diff[-1][j
+ 1]) / (self.table[j][0] - self.table[j + self.power -
i][0])
                diff.append(x)
            self.diff.append(diff)

    def find_polynom(self):
        self.find_nearest()
        self.find_diff()

    def res(self, x):
        y = self.diff[0][0]
        for k in range(1, self.power + 1):
            xmlt = 1
            for j in range(0, k):
                xmlt *= (x - self.table[j][0])
            y += xmlt * (self.diff[k][0])

        return y
```

Листинг 2. spline.py

```
class Spline:
    def __init__(self, table, x):
        self.table = table
        self.x = x
        self.N = len(table) - 1
        self.a = [0]
        self.b = [0]
        self.d = [0]
        self.c = [0, 0]
        self.kci = [0, 0, 0]
```

```
        self.etta = [0, 0, 0]
        self.h = [0, table[1][0] - table[0][0]]


    def find_coefs(self):
        for i in range(2, self.N + 1):
            self.h.append(self.table[i][0] -
self.table[i - 1][0])
            f = 3 * ((self.table[i][1] - self.table[i -
1][1]) / self.h[i] - (self.table[i - 1][1] -
self.table[i - 2][1]) / self.h[i - 1])
            self.kci.append(-self.h[i] / (self.h[i - 1]
* self.kci[i] + 2 * (self.h[i - 1] + self.h[i])))
            self.etta.append((f - self.h[i - 1] *
self.etta[i]) / (self.h[i - 1] * self.kci[i] + 2 *
(self.h[i - 1] + self.h[i])))


        for i in range(self.N - 1, 0, -1):
            self.c.insert(0, self.kci[i + 1] * self.c[0]
+ self.etta[i + 1])
        self.c.insert(0, 0)


        for i in range(1, self.N + 1):
            self.a.append(self.table[i - 1][1])
            self.b.append((self.table[i][1] -
self.table[i - 1][1]) / self.h[i] - self.h[i] *
(self.c[i + 1] + 2 * self.c[i]) / 3)
            self.d.append((self.c[i + 1] - self.c[i]) /
(3 * self.h[i]))


    def res(self):
        pos = 0
        for i in range(0, self.N):
            if self.table[i][1] <= self.x and
self.table[i + 1][1] >= self.x:
```

```
                pos = i + 1
                break

        return (self.a[pos] + self.b[pos] * (self.x -
self.table[pos - 1][0]) + self.c[pos] * (self.x -
self.table[pos - 1][0]) ** 2 + self.d[pos] * (self.x -
self.table[pos - 1][0]) ** 3)
```

Листинг 3. main.py

```
from newton import Newton_polynom

table_xy = [[0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
table_z = [[0, 1, 4, 9, 16],
           [1, 2, 5, 10, 17],
           [4, 5, 8, 13, 20],
           [9, 10, 13, 18, 25],
           [16, 17, 20, 25, 32]]


def three_d_inter(nx, ny, x_arg, y_arg):
    y_inter_table = []
    for i in range(len(table_xy[0])):
        table = [[table_xy[0][j], table_z[i][j]] for j
in range(len(table_xy[0]))]
        np = Newton_polynom(table, nx, x_arg)
        np.find_polynom()
        y_inter_table.append(np.res(x_arg))

    table = [[table_xy[1][j], y_inter_table[j]] for j in
range(len(table_xy[0]))]
```

```
    np = Newton_polynom(table, ny, y_arg)
    np.find_polynom()

    return np.res(y_arg)


def main():
    print("x = 1.5")
    print("y = 1.5")

    print("| power | interpolation result |")

    for power in range(1, 4):
        res = three_d_inter(power, power, 1.5, 1.5)

        print(f"| {power:5} | {res:20.8} |")


if __name__ == "__main__":
    main()
```

3 Результаты работы

```
таблица значений
|  x  |  y  |
|    0|    0|
|    1|    1|
|    2|    4|
|    3|    9|
|    4|   16|
|    5|   25|
|    6|   36|
|    7|   49|
|    8|   64|
|    9|   81|
|   10|  100|
X = 0.5
Точное значение: 0.25
Значение интерполяции полиномом Ньютона 3 степени 0.25
Интерполяция сплайном - 0.3415094339622642
```

```
X = 5.5
Точное значение: 30.25
Значение интерполяции полиномом Ньютона 3 степени 30.25
Интерполяция сплайном - 30.811320754716974
```

4 Вопросы при защите лабораторной работы

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

C = 0
a = x0
b = (y1 –y0) / (x1 –x0)
d = 0

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

- Совпадение значений сплайна и интреполируемой функции во всех 3 точках
- Равенство 1 и 2 производных на соседних участках
- Равенство 2 производной нулю в точках 0 и 2

3. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо C1=C2.

Kci = 1, etta = 0 из формулы 1.22

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,mи p-заданные числа.

CN= (p–k* ηN) / (k* ξ+ m)