МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ЛАБОРАТОРНАЯ РАБОТА №5

‘ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА’

Вариант №22

Студенты:
Самарина Арина Анатольевна, Суржицкий Арсений Арсентьевич
Группа P3266

Преподаватель:
Машина Екатерина Александровна

Санкт-Петербург, 2024

1. Цель работы

Цель лабораторной работы: решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек. Для исследования использовать:

- многочлен Лагранжа
- многочлен Ньютона
- многочлен Гаусса

2. Порядок выполнения работы

<u>Обязательное задание</u>

- 1 часть: Вычислительная реализация задачи

X	y	X_1	X_2
2,10	3,7587	2,112	2,205
2,15	4,1861	2,355	2,254
2,20	4,9218	2,114	2,216
2,25	5,3487	2,359	2,259
2,30	5,9275	2,128	2,232
2,35	6,4193	2,352	2,284
2,40	7,0839	2,147	2,247

1. Построить таблицу конечных разностей для заданной таблицы. Таблицу отразить в отчете
2. Вычислить значения функции для аргумента $X1$, используя первую или вторую интерполяционную формулу Ньютона. Обратить внимание какой конкретно формулой необходимо воспользоваться
3. Вычислить значения функции для аргумента $X2$, используя первую или вторую интерполяционную формулу Гаусса. Обратить внимание какой конкретно формулой необходимо воспользоваться
4. Подробные вычисления привести в отчете

- 2 часть: Программная реализация

1. Исходные данные задаются тремя способами:
   a) в виде набора данных (таблицы x,y), пользователь вводит значения с клавиатуры
   b) в виде сформированных в файле данных (подготовить не менее трех тестовых вариантов)
   c) на основе выбранной функции, из тех, которые предлагает программа, например, $\sin x$. Пользователь выбирает уравнение, исследуемый интервал и количество точек на интервале (не менее двух функций).
2. Сформировать и вывести таблицу конечных разностей
3. Вычислить приближенное значение функции для заданного значения аргумента, введенного с клавиатуры, указанными методами. Сравнить полученные значения

4. Построить графики заданной функции с отмеченными узлами интерполяции и интерполяционного многочлена Ньютона/Гаусса (разными цветами)
5. Программа должна быть протестирована на различных наборах данных, в том числе и некорректных.
6. Проанализировать результаты работы программы.

3. Необязательное задание

1. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Стирлинга
2. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Бесселя.

4. Рабочие формулы

1. Интерполяционные формулы Ньютона

$$N_n(x) = f(x_0) + \sum_{k=1}^{n} f(x_0, x_1, \dots, x_k) \prod_{j=0}^{k-1} (x - x_j)$$

Для равноотстоящих узлов

$$N_n(x) = y_1 + t\Delta y_1 + \frac{t(t-1)}{2!}\Delta^2 y_1 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_1 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_1 + \frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5 y_1$$

2. Первая интерполяционная формула Гаусса

$$P_n(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta^5 y_{-2} \dots + \frac{(t+n-1)\dots(t-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-(n-1)} + \frac{(t+n-1)\dots(t-n)}{(2n)!}\Delta^{2n} y_{-n}$$

3. Вторая интерполяционная формула Гаусса

$$P_n(x) = y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \dots + \frac{(t+n-1)\dots(t-n+1)}{(2n-1)!}\Delta^{2n-1} y_{-n} + \frac{(\boldsymbol{t}+\boldsymbol{n})(t+n-1)\dots(t-n+\mathbf{1})}{(2n)!}\Delta^{2n} y_{-n}$$

4. Интерполяционные многочлены Стирлинга

$$P_n(x) = y_0 + t\frac{\Delta y_{-1} + \Delta y_0}{2!} + \frac{t^2}{2}\Delta^2 y_{-1} + \frac{t(t^2-1^2)}{3!}\cdot\frac{\Delta^3 y_{-2} + \Delta^3 y_{-1}}{2} + \frac{t^2(t^2-1^2)}{4!}\Delta^4 y_{-2} + \dots + \frac{t(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n)!}\cdot\frac{\Delta^{2n-1} y_{-n} + \Delta^{2n-1} y_{-(n-1)}}{2} + \frac{t^2(t^2-1^2)(t^2-2^2)\dots(t^2-(n-1)^2)}{(2n)!}\Delta^{2n} y_{-n}$$

5. Интерполяционные многочлены Бесселя

$$P_n(x) = \frac{y_0 + y_{-1}}{2} + \frac{1}{8}\frac{\Delta^2 y_{-1} + \Delta^2 y_0}{2} + \frac{3}{128}\frac{\Delta^4 y_{-2} + \Delta^2 y_{-1}}{2} + \dots + (-1)^n\frac{(1\cdot 3\cdot 5\dots(2n-1))^2}{2^{2n}(2n)!}\frac{\Delta^{2n} y_{-n} + \Delta^{2n} y_{-(n-1)}}{2}$$

5. Вычислительная часть

- 1 часть: Вычислительная реализация задачи

x	y	X_1	X_2
2,10	3,7587	2,112	2,205
2,15	4,1861	2,355	2,254
2,20	4,9218	2,114	2,216
2,25	5,3487	2,359	2,259
2,30	5,9275	2,128	2,232
2,35	6,4193	2,352	2,284
2,40	7,0839	2,147	2,247

1) Построить таблицу конечных разностей для заданной таблицы. Таблицу отразить в отчете

Конечные разности функций удобно располагать в таблице:

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$

2,10	3,7587	0,4274	0,3083	-0,6171	1,0778	-1,7774	2,9757
2,15	4,1861	0,7357	-0,3088	0,4607	-0,6996	1,1983	
2,20	4,9218	0,4269	0,1519	-0,2389	0,4987		
2,25	5,3487	0,5788	-0,0870	0,2598			
2,30	5,9275	0,4918	0,1728				
2,35	6,4193	0,6646					
2,40	7,0839						

2) Вычислить значения функции для аргумента $X1$, используя первую или вторую интерполяционную формулу Ньютона.

Номер	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
0	2,10	3,7587	0,4274	0,3083	-0,6171	1,0778	-1,7774	2,9757
1	2,15	4,1861	0,7357	-0,3088	0,4607	-0,6996	1,1983	
2	2,20	4,9218	0,4269	0,1519	-0,2389	0,4987		
3	2,25	5,3487	0,5788	-0,0870	0,2598			
4	2,30	5,9275	0,4918	0,1728				
5	2,35	6,4193	0,6646					
6	2,40	7,0839						

Воспользуемся формулой Ньютона для интерполирования вперед, т.к. $x = 2{,}112$, $x = 2{,}114$, $x = 2{,}128$, $x = 2{,}147$ лежат в левой половине отрезка.

$$N_6(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)(t-6)}{6!}\Delta^6 y_0$$

x	t	$N_6(x)$
2.112	0.24	3.645469
2.114	0.28	3.648865
2.128	0.56	3.785927
2.147	0.94	4.128177

Воспользуемся формулой Ньютона для интерполирования назад, т.к. $x = 2{,}352$, $x = 2{,}355$, $x = 2{,}359$ лежит в второй половине отрезка

$$N_6(x) = y_6 + t\Delta y_5 + \frac{t(t+1)}{2!}\Delta^2 y_4 + \frac{t(t+1)(t+2)}{3!}\Delta^3 y_3 + \frac{t(t+1)(t+2)(t+3)}{4!}\Delta^4 y_2 + \frac{t(t+1)(t+2)(t+3)(t+4)}{5!}\Delta^5 y_1 + \frac{t(t+1)(t+2)(t+3)(t+4)(t+6)}{6!}\Delta^6 y_0$$

x	t	$N_6(x)$
2.352	-0.96	6.432536
2.355	-0.9	6.452021
2.359	-0.82	6.477829

3) Вычислить значения функции для аргумента $X2$, используя первую или вторую интерполяционную формулу Гаусса.

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
x_{-3}=2,10	3,7587	0,4274	0,3083	-0,6171	1,0778	-1,7774	2,9757
x_{-2}=2,15	4,1861	0,7357	-0,3088	0,4607	-0,6996	1,1983	

x_{-1}=2,20	4,9218	0,4269	0,1519	-0,2389	0,4987		
x_0=2,25	5,3487	0,5788	-0,0870	0,2598			
x_1=2,30	5,9275	0,4918	0,1728				
x_2=2,35	6,4193	0,6646					
x_3=2,40	7,0839						

Воспользуемся первой интерполяционной формуле Гаусса для $x = 2{,}254$, $x = 2{,}259$, $x = 2{,}284$

$$P_6(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta^5 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)(t-3)}{6!}\Delta^6 y_{-3}$$

x	t	$N_6(x)$
2.254	0.08	5.387469
2.259	0.18	5.438215
2.284	0.68	5.726761

Воспользуемся второй интерполяционной формуле Гаусса для $x = 2{,}205$, $x = 2{,}216$, $x = 2{,}232$, $x = 2{,}247$

$$P_6(x) = y_0 + t\Delta y_{-1} + \frac{t(t+1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-2} + \frac{(t+2)(t+1)t(t-1)}{4!}\Delta^4 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)}{5!}\Delta^5 y_{-3} + \frac{(t+3)(t+2)(t+1)t(t-1)(t-2)}{6!}\Delta^6 y_{-3}$$

x	t	$N_6(x)$
2.205	-0.9	4.968647
2.216	-0.68	5.062639
2.232	-0.36	5.191328
2.247	-0.06	5.3487

6. Листинг программы

Main.py

```
from methods import *
from utils import *
from functions import *

import matplotlib.pyplot as plt
import numpy as np

file_name = "points1"
input_methods = ["Файл данных", "Функция", "Ручной ввод"]
input_method = choose("метод ввода", input_methods)

match input_method:
    case "Ручной ввод":
        i = 1
        points = []
        while True:
```

```
            try:
                temp = (
                    input(f"Enter pair {i} x and y. For stop
enter end:\n")
                    .strip()
                    .replace(",", ".")
                    .split(" ")
                )
                if temp == ["end"] and i >= 2:
                    i += 1
                    break
                elif temp == ["end"] and i < 2:
                    print("Minimum 2 pairs, please add more")
                    i += 1
                    continue
                x, y = map(float, temp)
                i += 1

            except ValueError:
                print("Incorrect value entered")
                continue
            pair = [x, y]
            points.append(pair)
        print(f"Entered {i-1} pairs")
        fun = None
    case "Файл данных":
        points = read_points(file_name)
        fun = None
        if len(points) < 2:
            print(f"{len(points)} точек. Должно быть минимум 2")
            exit(1)
    case "Функция":
        fun = choose("функцию", functions)
        a = read_number("Левая граница: ")
        b = read_number("Правая граница: ", lambda x: x > a)
        n = read_number("Количество точек: ", lambda x: x > 1,
integer=True)

        stp = (b - a) / (n - 1)
        points = [(a + i * stp, fun(a + i * stp)) for i in
range(n)]
    case _:
        raise ValueError("Неверный метод ввода")

print_points(points)

x = read_number("Введите X: ", lambda x: points[0][0] <= x <=
points[-1][0])

plt.scatter(
    [point[0] for point in points], [point[1] for point in
points], label="Точки"
)
```

```
x_array = np.linspace(points[0][0], points[-1][0], 1000)
if fun is not None:
    print(f"\nНастоящее значение f({x}) = {fun(x)}")
    y_array_original = [fun(x) for x in x_array]
    plt.plot(x_array, y_array_original, label=fun)
# ----------------------------------------------------------------
------------------------------------------------------------
print("\nМетод Лагранжа")
print(lagrange(points, x), end="\n\n")
plt.plot(x_array, [lagrange(points, x) for x in x_array],
label="Лагранж")
# ----------------------------------------------------------------
------------------------------------------------------------
print("Метод Ньютона")
print(newton(points, x), end="\n\n")
plt.plot(x_array, [newton(points, x) for x in x_array],
label="Ньютон")


def print_combined_answer(points, x, method):
    answer, table = method(points, x)
    print("Ответ:", answer)
    print("Таблица конечных разностей:")
    print(make_finite_difference_table(table).to_string(),
end="\n\n")


# ----------------------------------------------------------------
------------------------------------------------------------
print("Метод Ньютона с разделёнными разностями")
if not is_equidistant(points):
    print("Точки не равноотстоящие, метод не применим",
end="\n\n")
else:
    # print_combined_answer(points, x, fixed_combined_newton)

    answer, table = fixed_combined_newton(points, x)
    print("Ответ:", answer, end="\n\n")

    plt.plot(
        x_array,
        [fixed_combined_newton(points, x)[0] for x in x_array],
        label="Ньютон с разделённными разностями",
    )

# ----------------------------------------------------------------
------------------------------------------------------------
print("Метод Ньютона с конечными разностями")
if not is_equidistant(points):
    print("Точки не равноотстоящие, метод не применим",
end="\n\n")
else:
    # print_combined_answer(points, x, fixed_combined_newton)
```

```
    answer, table = fixed_combined_newton2(points, x)
    print("Ответ:", answer, end="\n\n")

    plt.plot(
        x_array,
        [fixed_combined_newton(points, x)[0] for x in x_array],
        label="Ньютон с конечными разностями",
    )

# ---------------------------------------------------------------
--------------------------------------------------------
print("Метод Стирлинга")
if not is_equidistant(points):
    print("Точки не равноотстоящие, метод не применим",
end="\n\n")
elif len(points) % 2 == 0:
    print("Четное количество точек, метод не применим",
end="\n\n")
else:
    # print_combined_answer(points, x, stirling)

    answer, table = stirling(points, x)
    print("Ответ:", answer, end="\n\n")

    plt.plot(x_array, [stirling(points, x)[0] for x in x_array],
label="Стирлинг")
# ---------------------------------------------------------------
--------------------------------------------------------
print("Метод Бесселя")
if not is_equidistant(points):
    print("Точки не равноотстоящие, метод не применим",
end="\n\n")
elif len(points) % 2 != 0:
    print("Нечетное количество точек, метод не применим",
end="\n\n")
else:
    # print_combined_answer(points, x, bessel)

    answer, table = bessel(points, x)
    print("Ответ:", answer)

    plt.plot(x_array, [bessel(points, x)[0] for x in x_array],
label="Бессель")

plt.legend()
plt.axhline(y=0, color="k")
plt.axvline(x=0, color="k")
plt.grid()
plt.show()
```

functions.py

```
import math
from typing import Callable


class Function:
    def __init__(self, func: Callable[[float], float], text: 
str):
        self.func = func
        self.text = text

    def __call__(self, x: float):
        return self.func(x)

    def __str__(self):
        return self.text


functions = [
    Function(lambda x: -2 * x ** 3 - 5 * x ** 2 + 7 * x - 13, "-
2x^3 - 5x^2 + 7x - 13"),
    Function(lambda x: x ** 2, "x^2"),
    Function(lambda x: math.sin(x), "sin(x)"),
]
```

methods.py

```
import math
import utils


def lagrange(points, x):
    """Многочлен Лагранжа"""

    def lagrange_coefficient(points, x, index):
        n = len(points) - 1
        p_top = 1
        p_bottom = 1
        for j in range(n + 1):
            if j != index:
                p_top *= x - points[j][0]
                p_bottom *= points[index][0] - points[j][0]
        return p_top / p_bottom

    n = len(points) - 1
    summ = 0
    for i in range(n + 1):
        summ += points[i][1] * lagrange_coefficient(points, x, 
i)
    return summ


def newton(points, x, max_k=None, base_index=0):
    """Интерполяционный многочлен Ньютона с разделенными
```

```
разностями"""

    def divided_difference(points, i, k, table):
        if k == 0:
            table[i][0] = points[i][1]
            return points[i][1]
        if table[i + 1][k - 1] is None:
            table[i + 1][k - 1] = divided_difference(points, i +
1, k - 1, table)
        if table[i][k - 1] is None:
            table[i][k - 1] = divided_difference(points, i, k -
1, table)
        return (table[i + 1][k - 1] - table[i][k - 1]) / (
            points[i + k][0] - points[i][0]
        )

    divided_differences = [
        [None for _ in range(len(points))] for _ in
range(len(points))
    ]
    points = points[base_index : len(points)]
    n = len(points) - 1
    if max_k is not None and max_k < n:
        n = max_k
    summ = points[0][1]
    p = 1
    for k in range(1, n + 1):
        p *= x - points[k - 1][0]
        ds = divided_difference(points, 0, k,
divided_differences)
        summ += ds * p
    return summ


def _finite_difference(points, index, power, table):
    if power == 0:
        table[index][power] = points[index][1]
    else:
        if table[index + 1][power - 1] is None:
            table[index + 1][power - 1] = _finite_difference(
                points, index + 1, power - 1, table
            )
        if table[index][power - 1] is None:
            table[index][power - 1] = _finite_difference(
                points, index, power - 1, table
            )
        table[index][power] = table[index + 1][power - 1] -
table[index][power - 1]
    return table[index][power]


def first_newton(points, x, calc_base_index=False) -> (float,
list[list[float]]):
```

```
    """Первая интерполяционная формула Ньютона для
интерполирования вперед"""
    base_index = 0
    if calc_base_index:
        for i in range(len(points)):
            if points[i][0] > x:
                base_index = i - 1
                break
    t = (x - points[base_index][0]) / utils.step(points)
    finite_difference_table = [
        [None for _ in range(len(points))] for _ in
range(len(points))
    ]
    summ = points[base_index][1]
    top_t = 1
    for i in range(1, len(points) - base_index):  #
        top_t *= t - i + 1
        summ += (
            _finite_difference(points, base_index, i,
finite_difference_table)
            * top_t
            / math.factorial(i)
        )
    return summ, finite_difference_table


def second_newton(points, x) -> (float, list[list[float]]):
    """Вторая интерполяционная формула Ньютона для
интерполирования назад"""
    n = len(points) - 1
    t = (x - points[n][0]) / utils.step(points)
    finite_difference_table = [[None for _ in range(n + 1)] for
_ in range(n + 1)]
    summ = points[n][1]
    top_t = 1
    for i in range(1, n + 1):
        top_t *= t + i - 1
        summ += (
            _finite_difference(points, n - i, i,
finite_difference_table)
            * top_t
            / math.factorial(i)
        )
    return summ, finite_difference_table


def fixed_combined_newton(
    points, x, calc_base_index=False
) -> (float, list[list[float]]):
    """Интерполяция Ньютона для равноотстоящих узлов"""
    if x <= (points[0][0] + points[-1][0]) / 2:
        return first_newton(points, x, calc_base_index)
    else:
```

```
        return second_newton(points, x)


def finite_difference(points):
    """Вычисляет таблицу конечных разностей для заданных
точек."""
    n = len(points)
    table = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        table[i][0] = points[i][1]

    for j in range(1, n):
        for i in range(n - j):
            table[i][j] = table[i + 1][j - 1] - table[i][j - 1]

    return table


def forward_newton_interpolation_with_table(points, x):
    """Вычисляет значение интерполяционного многочлена Ньютона
(интерполяция вперед) в точке x с таблицей конечных
разностей."""
    table = finite_difference(points)
    n = len(points)
    h = points[1][0] - points[0][0]
    t = (x - points[0][0]) / h
    result = table[0][0]

    product = 1
    for i in range(1, n):
        product *= t - (i - 1)
        result += (product / math.factorial(i)) * table[0][i]

    return result, table


def backward_newton_interpolation_with_table(points, x):
    """Вычисляет значение интерполяционного многочлена Ньютона
(интерполяция назад) в точке x с таблицей конечных разностей."""
    table = finite_difference(points)
    n = len(points)
    h = points[1][0] - points[0][0]
    t = (x - points[n - 1][0]) / h
    result = table[n - 1][0]

    product = 1
    for i in range(1, n):
        product *= t + (i - 1)
        result += (product / math.factorial(i)) * table[n - i -
1][i]

    return result, table
```

```
def fixed_combined_newton2(points, x) -> (float,
list[list[float]]):
    """Интерполяция Ньютона для равноотстоящих узлов"""
    if x <= (points[0][0] + points[-1][0]) / 2:
        return forward_newton_interpolation_with_table(points,
x)
    else:
        return backward_newton_interpolation_with_table(points,
x)


def stirling(points, x):
    """Интерполяционный многочлен Стирлинга"""
    if len(points) % 2 == 0:
        raise Exception("Stirling interpolation requires odd
number of points")
    _2n = len(points) - 1
    index = len(points) // 2
    t = (x - points[index][0]) / utils.step(points)
    finite_difference_table = [
        [None for _ in range(len(points))] for _ in
range(len(points))
    ]
    summ = points[index][1]
    top_t = 1
    for i in range(1, _2n + 1, 2):
        first_fd = _finite_difference(points, index, i,
finite_difference_table)
        second_fd = _finite_difference(points, index - 1, i,
finite_difference_table)
        third_fd = _finite_difference(points, index - 1, i + 1,
finite_difference_table)
        top_t *= t**2 - (i // 2) ** 2
        first_summand = (
            top_t
            / (t if t != 0 else 1)
            / math.factorial(i)
            * (first_fd + second_fd)
            / 2
        )
        second_summand = top_t / math.factorial(i + 1) *
third_fd
        summ += first_summand + second_summand
        index -= 1
    return summ, finite_difference_table


def bessel(points, x):
    """Интерполяционный многочлен Бесселя"""
    if len(points) % 2 != 0:
        raise Exception("Number of points must be even")
```

```
    _2n = len(points) - 1
    index = len(points) // 2 - 1
    t = (x - points[index][0]) / utils.step(points)
    finite_difference_table = [
        [None for _ in range(len(points))] for _ in
range(len(points))
    ]
    th = t - 0.5
    summ = (points[index][1] + points[index + 1][1]) / 2 + th *
_finite_difference(
        points, index, 1, finite_difference_table
    )
    top_t = 1
    for i in range(2, _2n + 1, 2):
        top_t *= (t - i // 2) * (t + i // 2 - 1)
        first_summand = (
            top_t
            / math.factorial(i)
            * (
                _finite_difference(points, index, i,
finite_difference_table)
                + _finite_difference(points, index - 1, i,
finite_difference_table)
            )
            / 2
        )
        second_summand = (
            th
            * top_t
            / math.factorial(i + 1)
            * _finite_difference(points, index - 1, i + 1,
finite_difference_table)
        )
        summ += first_summand + second_summand
        index -= 1
    return summ, finite_difference_table
```