Міністерство освіти і науки України
Національний університет “Львівська політехніка”
Інститут комп’ютерних наук та інформаційних технологій
Кафедра програмного забезпечення

Звіт
Про виконання лабораторної роботи №8

На тему:

«Наближення дискретних (таблично заданих) функцій»
з дисципліни «Чисельні методи»

Лекторка:
доцент каф. ПЗ
Мельник Н. Б.
Виконав:
ст. гр. ПЗ-11
Солтисюк Д.А.
Прийняла:
доцент каф. ПЗ
Мельник Н. Б.

« __ » ________ 2022 р.

∑ = _____ _________ .

Львів – 2022

Тема: Наближення дискретних (таблично заданих) функцій.
Мета: Ознайомитися з методом інтерполяції таблично заданих функцій.

Теоретичні відомості

Інтерполяційний поліном Лагранжа – поліном, в основу якого покладено те, що в одному довільному вузлі інтерполяції поліном приймає значення одиниці, а у всіх інших – нуль. Наближена функція матиме вигляд $F(x) = \sum_{i=0}^{n} P_i(x) f(x_i)$, де

$$P_i(x) = \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)}$$

Звідки отримаємо інтерполяційний многочлен Лагранжа

$$F(x) = \sum_{i=0}^{n} \frac{(x - x_0)(x - x_1)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)(x_i - x_1)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)} f(x_i)$$

Інтерполяційний поліном Ньютона – використовує дещо інший принцип побудови:

$$P_n(x) = f(x_0) + f(x_0, x_1)(x - x_0) + f(x_0, x_1, x_2)(x - x_0)(x - x_1) + \ldots + + f(x_0, x_1, \ldots, x_n)(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$
,

Де $f(x_0, x_1, \ldots, x_n) = \frac{f(x_1, \ldots, x_n) - f(x_0, \ldots, x_{n-1})}{x_n - x_0}$ – розділена різниця n-го порядку для нерівновіддалених вузлів.

У випадку рівновіддалених вузлів використовуємо скінчені різниці

$\Delta^n f(x_i) = \Delta^{n-1} f(x_{i+1}) - \Delta^{n-1} f(x_i)$, де $x_i = x_0 + ih$

Тоді $f(x_0, x_1, \ldots, x_n) = \frac{\Delta^n f(x_0)}{n!h^n}$, звідки отримуємо інтерполяційний поліном Ньютона для рівновіддалених вузлів

$$P_n(x) = f(x_0) + \frac{\Delta f(x_0)}{1!h}(x - x_0) + \frac{\Delta^2 f(x_0)}{1!h^2}(x - x_0)(x - x_1) + \ldots +$$

$$+\frac{\Delta^n f(x_0)}{n!h^n}\left(x-x_0\right)\left(x-x_1\right)\ldots(x-x_{n-1})$$

Індивідуальне завдання

Використовуючи інтерполяційні поліноми Лагранжа та Ньютона, обчислити значення табличної заданої функції у точці x0:

9-й варіант

x	0,45	0,46	0,47	0,48	0,49	0,50	0,51	0,52	0,53	0,54
y	20,19	19,61	18,94	18,17	17,30	16,31	15,19	13,94	12,55	10,99

Код функцій

```
from common.main import InterpolationOrientedMethod, Points
from common.utils import print_header
from prettytable.prettytable import PrettyTable


class LagrangePolynomialMethod(InterpolationOrientedMethod):

    def __init__(self, points: Points, x0: float) -> None:
        super().__init__("Lagrange polynomial", points, x0)

    def execute_method(self):
        x_points = self.points[0]
        y_points = self.points[1]

        n = len(x_points)
        xp = self.x0  # x0 as a starting point
        yp = 0  # interpolated value
        t = PrettyTable(["Step", "Y"])

        for i in range(n):
            p = 1

            for j in range(n):
                if i == j:
                    continue
```

```
                p *= (xp - x_points[j]) / (x_points[i] - x_points[j])

            yp += p * y_points[i]

            t.add_row([i, yp])

        print("\n")
        print_header("Iteration process")
        print(t)

        return xp, yp


import numpy as np
from common.main import InterpolationOrientedMethod, Points
from common.utils import print_header
from prettytable.prettytable import PrettyTable


# Requirements:
# [-] Step size is constant
# [-] Degree is equal to (<number_of_points> - 1)
class NewtonPolynomialMethod(InterpolationOrientedMethod):

    def __init__(self, points: Points, x0: float) -> None:
        super().__init__("Newton's polynomial", points, x0)

    def newton_coefficient(self):
        """
        x: list or np array containing x data points
        y: list or np array containing y data points
        """

        m = len(self.points[0])

        x = np.copy(self.points[0])
        a = np.copy(self.points[1])
        for k in range(1, m):
            a[k:m] = (a[k:m] - a[k - 1]) / (x[k:m] - x[k - 1])

        return a

    # Steps:
    # [-] Calculating f(x_prev, x_curr) and f(x_prev, x_curr, x_next)
    # whereas f(x_prev, x_curr, x_next) equals to ...
    # (f(x_curr, x_next) - f(x_prev, x_curr)) / x_next - x_prev
    # [-] Building the table of values
    # rows: [x_i | f_i | f(x_i, x_ii) | f(x_i, x_ii, x_iii)]
    # [-] Using formula calculating the polynomial
    def execute_method(self):
        """
        x_data: data points at x
        y_data: data points at y
        x: evaluation point(s)
        """
        xp = self.x0
        a = self.newton_coefficient()
        n = len(self.points[0]) - 1  # Degree of polynomial
        yp = a[n]
        t = PrettyTable(["Step", "Y"])
```

```
    for k in range(1, n + 1):
        yp = a[n - k] + (xp - self.points[0][n - k]) * yp
        t.add_row([k, yp])

    print("\n")
    print_header("Iteration process")
    print(t)

    return xp, yp
```

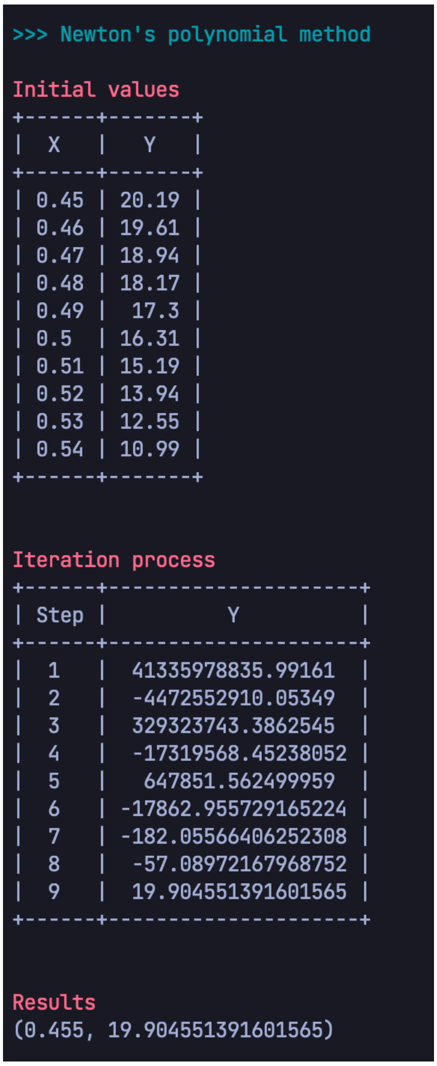
Протокол роботи

```
>>> Lagrange polynomial method

Initial values
+------+-------+
|  X   |   Y   |
+------+-------+
| 0.45 | 20.19 |
| 0.46 | 19.61 |
| 0.47 | 18.94 |
| 0.48 | 18.17 |
| 0.49 |  17.3 |
| 0.5  | 16.31 |
| 0.51 | 15.19 |
| 0.52 | 13.94 |
| 0.53 | 12.55 |
| 0.54 | 10.99 |
+------+-------+


Iteration process
+------+--------------------+
| Step |         Y          |
+------+--------------------+
|  0   | 3.7446510314941346 |
|  1   | 36.478353881836036 |
|  2   | -5.675399780273359 |
|  3   | 50.940607910156054 |
|  4   | -6.814931030273236 |
|  5   | 35.53542144775377  |
|  6   | 14.021508483886773 |
|  7   | 21.18124359130858  |
|  8   | 19.784650115966794 |
|  9   | 19.904551391601558 |
+------+--------------------+


Results
(0.455, 19.904551391601558)
```



```
>>> Newton's polynomial method

Initial values
+------+-------+
|  X   |   Y   |
+------+-------+
| 0.45 | 20.19 |
| 0.46 | 19.61 |
| 0.47 | 18.94 |
| 0.48 | 18.17 |
| 0.49 |  17.3 |
| 0.5  | 16.31 |
| 0.51 | 15.19 |
| 0.52 | 13.94 |
| 0.53 | 12.55 |
| 0.54 | 10.99 |
+------+-------+


Iteration process
+------+---------------------+
| Step |          Y          |
+------+---------------------+
|  1   |  41335978835.99161  |
|  2   |  -4472552910.05349  |
|  3   |  329323743.3862545  |
|  4   |  -17319568.45238052 |
|  5   |   647851.562499959  |
|  6   | -17862.955729165224 |
|  7   | -182.05566406252308 |
|  8   |  -57.08972167968752 |
|  9   |  19.904551391601565 |
+------+---------------------+


Results
(0.455, 19.904551391601565)
```

Рис.1. Робота програми

Висновки

Виконуючи лабораторну роботу №8, я ознайомився з методом інтерполяції таблично заданих функцій, та склав програму для інтерполяції методом Лагранжа та методом Ньютона (універсальні).