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 , де

Звідки отримаємо інтерполяційний многочлен Лагранжа

**Інтерполяційний поліном Ньютона** – використовує дещо інший принцип побудови:

,

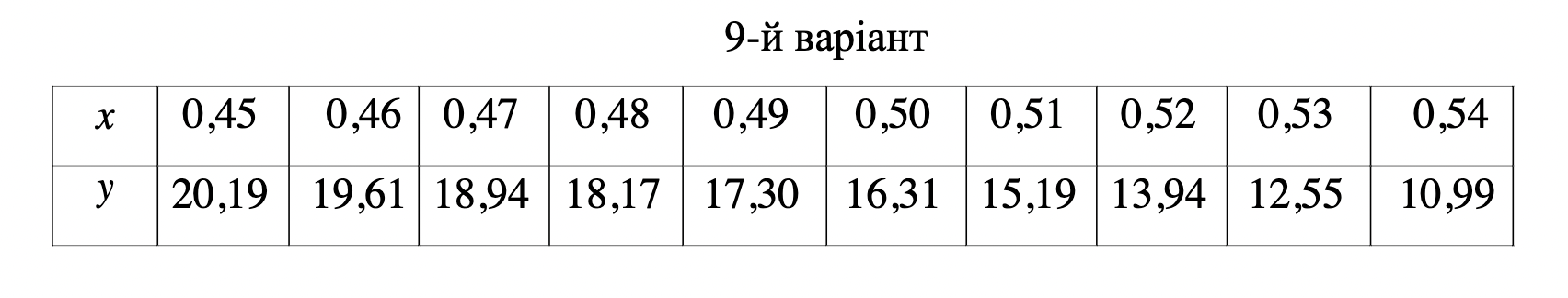
Де – розділена різниця n-го порядку для нерівновіддалених вузлів.

У випадку рівновіддалених вузлів використовуємо скінчені різниці

, де

Тоді , звідки отримуємо інтерполяційний поліном Ньютона для рівновіддалених вузлів

**Індивідуальне завдання**

Використовуючи інтерполяційні поліноми Лагранжа та Ньютона, обчислити значення табличної заданої функції у точці х0:

**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 contanining x data points

y: list or np array contanining 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**

****

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).