НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Кафедра прикладної математики

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з кредитного модуля

"Програмування 1. Основи програмування"   
на тему:

«Програма чисельного диференціювання   
(Обчислення другої похідної)»

# Виконав Казновський Антон Томашович

Група КМ-02 факультет прикладної математики

N залікової книжки КМ-0208

Керівник Олефір О. С.

"13" грудня 2020р.

Захищена з оцінкою\_\_\_\_\_\_\_\_\_\_\_

Київ 2020

1. ПОСТАНОВКА ЗАДАЧІ

Програма для обчислення другої похідної функції складається з двох етапів.

На першому програма формує інтерполяційний поліном Ньютона та знаходить його другу похідну за допомогою функцій та методів бібліотеки SymPy. Це робиться в залежності від заданих користувачем табличних значень залежних та незалежних змінних функції, В цей процес входить і алгоритм формування різниць сусідніх значень функції, що розглядається. Для цього використовуються деякі функції бібліотеки NumPy.

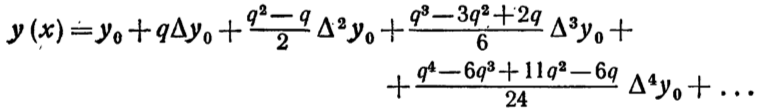
На другому етапі програма підставляє введені значення в отриману формулу та підраховує результати. Обчислюється друга похідна в заданій точці та похибка вимірювань.

Повний код програми наведено в Додатку 1.1.

2. ПРОЕКТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

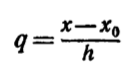
2.1. Методи вирішення задачі

Програма використовує формулу інтерполяційного поліному Ньютона, що виведена у підручнику Б. П. Демідовича та І. А. Марона «Основи обчислювальної математики» [1] (Додаток 2.1.).



Додаток 2.1. Формула інтерполяційного поліному Ньютона

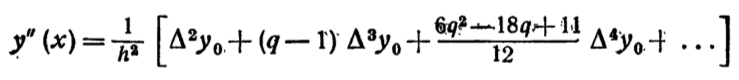
В цій та наступних формулах маємо (Додаток 2.2.).

, 

Додаток 2.2. Пояснення змінних формули

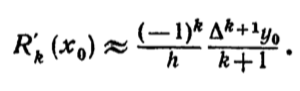
Тобто, h – різниця між кожними сусідніми аргументами функції, q – різниця між початковим та шуканим аргументом, поділена на різницю між сусідніми.

Сама ж похідна обчислюється за формулою Додатку 2.3.



Додаток 2.3. Формула другої похідної

Для обчислення приблизної похибки вимірювань використовується формула з того ж джерела:



Додаток 2.4. Формула знаходження похибки

Знайдемо другу похідну для прикладів, що розглядатимуться під час тестування програми.

1. = esinx

Знайдемо першу похідну:

Знайдемо другу похідну:

Відповідь: .

Знайдемо першу похідну:

Знайдемо другу похідну:

Відповідь: .

Знайдемо першу похідну:

Знайдемо другу похідну:

Відповідь: .

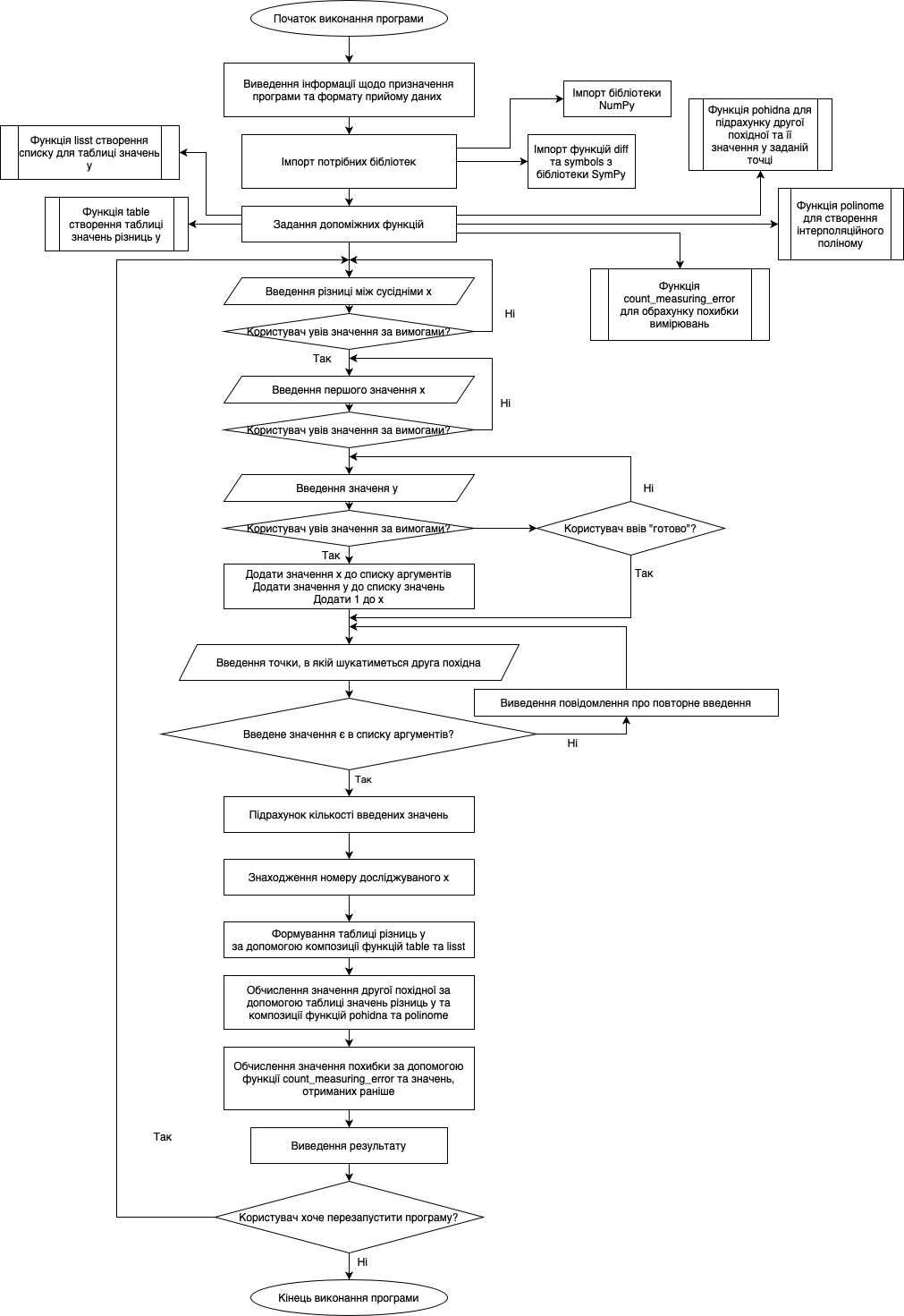
Саме за цими результатами далі перевірятимемо робочу здатність програми.

2.2. Проектування алгоритмів

Далі описані та обґрунтовані всі рішення відносно структури програми, прийняті під час виконання роботи.

2.2.1. Структура програмного забезпечення

На блок-схемі Додатку 2.2.1.1. детально зображена архітектура програми на достатньому рівні для супроводу та модифікації системи.



Додаток 2.2.1.1. Детальна блок-схема роботи програми

2.2.2. Опис розроблених алгоритмів

Програма починається з виведення інформації про призначення програми та її розробника.

Далі міститься текстовий блок «Формат даних», що коротко описує дії, виконувані програмою під час запуску, та формат значень, що має вводити користувач під час роботи.

Наступний текстовий блок «ПОРАДА» містить побажання (необов’язкове) до даних, що вводитиме користувач, спрямовані на більшу точність кінцевих результатів.

Програма імпортує модуль NumPy та функції diff та symbols модулю SymPy.

Далі іде задання 5-и функцій, що використовуватимуться для дослідження.

Функція lisst створює список, що складається з підсписків, кожен наступний з яких містить різницю між сусідніми елементами попереднього підсписку. Значення беруться зі списку значень функції list\_of\_values. Для цього використовується 3 послідовні цикли for, перший з яких формує пусті підсписки у кількості, рівній довжині списку list\_of\_values, другий заповнює перший підсписок різницями значень з list\_of\_values, а третій робить це саме, проте із значень попереднього підсписку. В кінці у отриманий список на 0 позицію додається початковий список list\_of\_values. Повертається згенерований список, в якому довжина кожного наступного підсписку на 1 менша за довжину попереднього.

Функція table формує матрицю, а саме двовимірний масив типу numpy.array зі списку, отриманого з функції lisst. Для цього у кожен підсписок додається стільки нулів, скільки не вистачає до того, аби довжина цього підсписку дорівнювала довжині першого підсписку. Цей алгоритм реалізовано за допомогою двох вкладених циклів for. Повертається масив зі значеннями.

Функція polinome реалізує створення поліному в аналітичному записі. Для цього використовуються елементи бібліотеки SymPy. Змінні a та q переводяться у тип символів, задається початкове значення поліному 0 (локальна змінна summa) та перший одночлен a=1. Далі в циклі for (з j від 0 до значення довжини списку аргументів) до summa додаються одночлени, що обчислюються як попередній, помножений на різницю q та j та поділений на j+1. Повертається отриманий поліном зі змінною q.

Функція pohidna обчислює змінну h, як обернене число різниці сусідніх аргументів, та змінну param, як добутку h та різниці між досліджуваним х та та початковим. Далі функція рахує другу похідну поліному за допомогою функції diff модулю SymPy та множить отриманий поліном на h2. В отримане значення підставляється раніше отримане значення param за допомогою методу evalf. Повертається результат – друга похідна у досліджуваній точці.

Функція count\_measuring\_error підраховує та повертає значення похибки за формулою.

Наступною частиною програми є введення даних. Це реалізовано за допомогою циклів while. Спочатку вводиться різниця між *х*-ами, потім перше значення х. Далі значення *у* для введеного *х*, наступне значення для *х*, що дорівнює сумі попереднього *х* та різниці між *х* і т.д. Коли користувач вводить «готово», введення припиняється. Усі введені значення *у* формують список list\_of\_values, а *х*, для яких ці значення вводились, список list\_of\_arguments. Далі необхідно ввести значення *х*, для якого шукатиметься друга похідна. До того ж, це значення має бути серед тих, для якого ввели значення звичайної функції. В усіх уведеннях наявна перевірка на правильність формату введення. В разі некоректності з’являється інформативне повідомлення та запитується повторне введення.

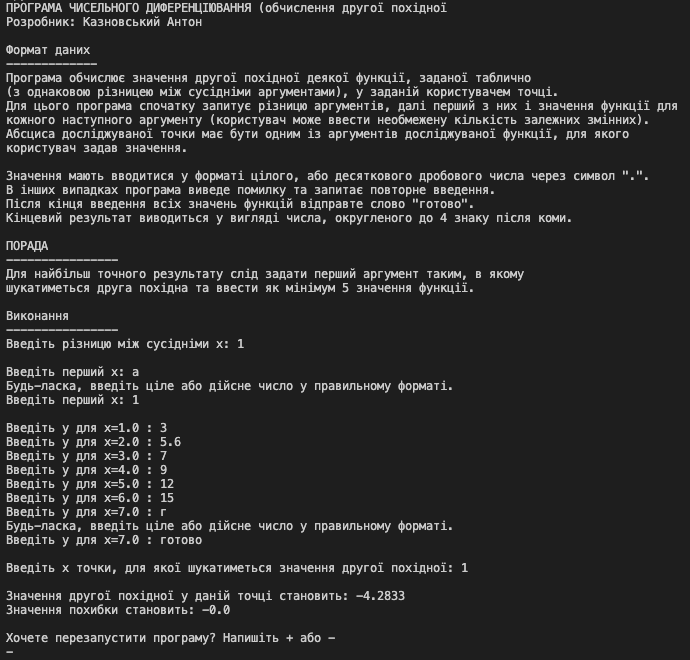
Далі програма обраховує певні значення та підставляє їх у попередні функції. Насамкінець, виводиться результат. Частина коду після задання функцій поміщена у цикл while, що дозволяє спробувати скористатися програмою повторно без її перезапуску.

Весь код написаний відповідно до стандартів написання РЕР 8.

2.2.3. Засоби керування програми

Впродовж користування програмним забезпеченням користувача супроводжують відповідні інформаційні виведення. На початку міститься інформація про формат введення. При відправці даних у некоректному вигляді користувача повідомляють про це та запитують ввести дані повторно.

Приклад реалізації програми наведено у Додатку 2.2.3.1.



Додаток 2.2.3.1. Приклад реалізації програми

3. РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1. Опис вхідних даних

Під час введення значень функції для аргументів, користувач бачить, для якого конкретного аргументу він пише це значення, що допомагає легше зорієнтуватися.

В кінці в користувача також запитується повторний запуск програми, на що він має відповісти позитивно чи негативно («+» або «-«), в залежності від відповіді програма запуститься ще раз.

Усі введення перевіряються на відповідність умовам введення та в разі невиконання про це виводиться повідомлення та запитується повторне введення.

3.2. Опис результатів

Результати обчислень підписані відповідно до їх сутності та округлені до 4 знаку після коми для компактності. Виводиться обчислене значення другої похідної функції у заданій точці та похибка обчислення.

3.3. Контрольні приклади

1. = esinx

Отримаємо значення для введення підставивши, наприклад, х від 0,2 до 1,6 з кроком 0,2:  
= esin0,2 ≈ 1,2197  
= esin0,4 ≈ 1,4761  
= esin0,6 ≈ 1,7589  
= esin0,8 ≈ 2,049  
= esin1 ≈ 2,32  
= esin1,2 ≈ 2,54

= esin1,4 ≈ 2,6743

= esin1,6 ≈ 2,717

Тепер знайдемо значення другої похідної у точці х = 2: ≈ 0,9293

2.

Отримаємо значення для введення підставивши, наприклад, х від 2,5 до 5 з кроком 0.5:

≈ 4,8333  
 = 5  
 = 5,3  
 ≈ 5,6667  
 ≈ 6,0716  
 = 6,5

Тепер знайдемо значення другої похідної в точці х = 2,5: ≈ 1,1851

3.

Отримаємо значення для введення підставивши, наприклад, х від 2 до 7:

= = 80  
= = 405  
= = 1280  
= = 3125  
= = 6480  
= = 12005

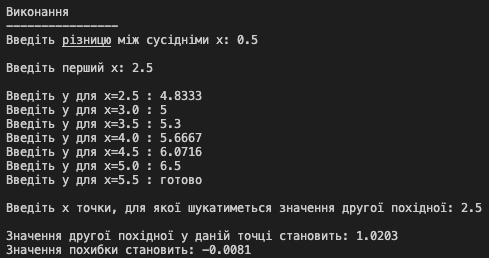
Тепер знайдемо значення другої похідної у точці х = 2: = 240

3.4. Експериментальні розрахунки

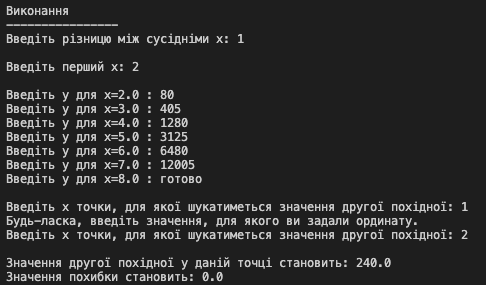
Перевіримо робочу здатність програми, а тобто відповідність попередніх розрахунків з обчисленнями, що веде програма.



Додаток 3.4.1. Тестування програми за першим контрольним прикладом



Додаток 3.4.2. Тестування програми за другим контрольним прикладом



Додаток 3.4.3. Тестування програми за третім контрольним прикладом

Можна помітити, що програма має деякі неточності.

Результат залежить, насамперед, від функції, що тестується, кількості введених даних та обраної точки для розрахунку другої похідної. Для найпростіших функцій (де аргумент просто має певний степінь та не міститься у жодній іншій функції) результат є найточнішим.

ВИСНОВКИ

Під час виконання розрахунково-графічної роботи було написано програму для обчислення другої похідної у певній точці функції, заданої таблично.

Для роботи ПЗ користувачу необхідно ввести різницю між сусідніми аргументами, перший аргумент, значення функції для аргументів та досліджувану точку. Програма виведе результат – другу похідну у введеній точці та похибку обчислень.

Реалізація програми складається з двох основних частин: створення поліному Ньютона та підстановки у нього значень, одержаних від користувача. Було проведено декілька тестувань для трьох різних функцій, результати в більшості з них майже співпали з результатами обчислень поза комп’ютером.

На точність результатів виконання програми впливає кількість заданих значень, характер функції, обрана точка, для якої обчислюється друга похідна.

В свою чергу я поглибив свої знання з математики та програмування, навчився інтерпретувати математичні формули та закони мовою програмування Python.

ЛІТЕРАТУРА

1. Демидович Б.П., Марон И.А. Основы вычислительной математики / Б.П. Демидович, И.А. Марон - М. : Наука, 1970. - 660 с.

2. Прохоренок Н.А., Дронов В.А. Python 3. Самое необходимое / Н. Прохоренок, В. Дронов – Санкт-Петербург: БХВ Петербург, 2016. — 460 с.

3. Громова В.В. Програмування на мові Python / Громова В.В. 2018 – 198 с.

ДОДАТКИ

# Виведення початкової інформації

print("ПРОГРАМА ЧИСЕЛЬНОГО ДИФЕРЕНЦІЮВАННЯ (обчислення другої похідної \nРозробник: Казновський Антон\n")

print("Формат даних\n"+"-"\*13,

"\nПрограма обчислює значення другої похідної деякої функції, заданої таблично\n" +

"(з однаковою різницею між сусідніми аргументами), у заданій користувачем точці.\n" +

"Для цього програма спочатку запитує різницю аргументів, далі перший з них і значення функції для\n" +

"кожного наступного аргументу (користувач може ввести необмежену кількість залежних змінних).\n" +

"Абсциса досліджуваної точки має бути одним із аргументів досліджуваної функції, для якого\n" +

"користувач задав значення.\n")

print('Значення мають вводитися у форматі цілого, або десяткового дробового числа через символ ".".\n' +

'В інших випадках програма виведе помилку та запитає повторне введення.\n' +

'Після кінця введення всіх значень функцій відправте слово "готово".\n' +

'Кінцевий результат виводиться у вигляді числа, округленого до 4 знаку після коми.\n')

print("ПОРАДА\n" + "-"\*16 + "\nДля найбільш точного результату слід задати перший аргумент таким, в якому\n"+

"шукатиметься друга похідна та ввести як мінімум 5 значення функції.\n")

print("Виконання\n" + "-"\*16)

# Імпорт додаткових бібліотек

import numpy as np

from sympy import diff, symbols

# Задання допоміжних функцій

def table(lst):

'''Функція створення матриці значень дельта у'''

k = len(lst[0])

for i in lst:

if len(i) != k:

for d in range((k - len(i))):

i.append(float(0))

delta\_y\_table = np.array(lst).transpose()

return delta\_y\_table

def lisst(list\_of\_values, length):

'''Функція створення списку для таблиці значень y'''

delta\_y\_list = []

for d in range(length-1):

delta\_y\_list.append([])

for k in range(1, length):

delta\_y\_list[0].append(list\_of\_values[k] - list\_of\_values[k-1])

for i in range(1, length):

for k in range(len(delta\_y\_list[i-1]) - 1):

delta\_y\_list[i].append(delta\_y\_list[i-1][k+1] - delta\_y\_list[i-1][k])

delta\_y\_list.insert(0, list\_of\_values)

return delta\_y\_list

def polinome(delta\_y\_list, i):

'''Функція для створення поліному'''

a, q = symbols('a, q')

summa = 0

a=1

for j in range(len(list\_of\_arguments)):

summa += a\*(deltas\_ys[i, j])

a \*= q - j

a /= 1 + j

return summa

def pohidna(summa):

'''Функція, що рахує другу похідну у даній точці для отриманого поліному'''

q = symbols('q')

h = (delta)\*\*(-1)

second\_pohidna = (h\*\*2)\*diff(summa, q, 2)

param = h\*(find\_x-list\_of\_arguments[0])

result = second\_pohidna.evalf(subs={q:param})

return result

def count\_measuring\_error(length, deltas\_ys, i):

'''Функція для розрахунку похибки обчислень'''

h = (delta)\*\*(-1)

return (-1)\*\*(length-i-2)\*h/(length-i)\*deltas\_ys[i, length-i-1]

#==========================

# Реалізація програми

while True:

#Введення різниці між х

while True:

try:

delta = float(input("Введіть різницю між сусідніми х: "))

break

except ValueError:

print("Будь-ласка, введіть ціле або дійсне число у правильному форматі.")

print()

#Введення першого значення х

while True:

try:

x = float(input("Введіть перший х: "))

break

except ValueError:

print("Будь-ласка, введіть ціле або дійсне число у правильному форматі.")

print()

#Введення та формування списку значень

list\_of\_arguments = []

list\_of\_values = []

while True:

n = input("Введіть у для х="+str(x)+ " : ")

if n == "готово":

if len(list\_of\_values) != 0:

break

else:

print("Введіть хочаб одне значення у")

else:

try:

list\_of\_values.append(float(n))

list\_of\_arguments.append(x)

x += delta

except ValueError:

print("Будь-ласка, введіть ціле або дійсне число у правильному форматі.")

print()

#Введення значення х, в якому шукатиметься друга похідна

while True:

try:

find\_x = float(input("Введіть х точки, для якої шукатиметься значення другої похідної: "))

if find\_x in list\_of\_arguments:

break

else:

print("Будь-ласка, введіть значення, для якого ви задали ординату.")

except ValueError:

print("Будь-ласка, введіть ціле або дійсне число у правильному форматі.")

print()

#Опрацювання даних

length = len(list\_of\_values)

i = list\_of\_arguments.index(find\_x)

deltas\_ys = table(lisst(list\_of\_values, length))

znachennya = pohidna(polinome(deltas\_ys, i))

znachennya = round(float(znachennya), 4)

pohibka = round(count\_measuring\_error(length, deltas\_ys, i), 4)

#Виведення результату

print("Значення другої похідної у даній точці становить:", znachennya)

print("Значення похибки становить:", pohibka)

#Запит на повторний запуск програми

cont = input("\nХочете перезапустити програму? Напишіть + або -\n")

while True:

if (cont != "-") and (cont != "+"):

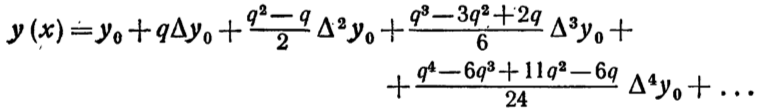
cont = input("Неправильний формат введення. Напишіть + або -\n")

else: break

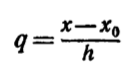
if cont == "-":

break

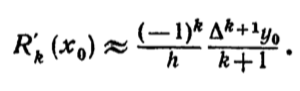
Додаток 1.1. Код програми чисельного диференціювання



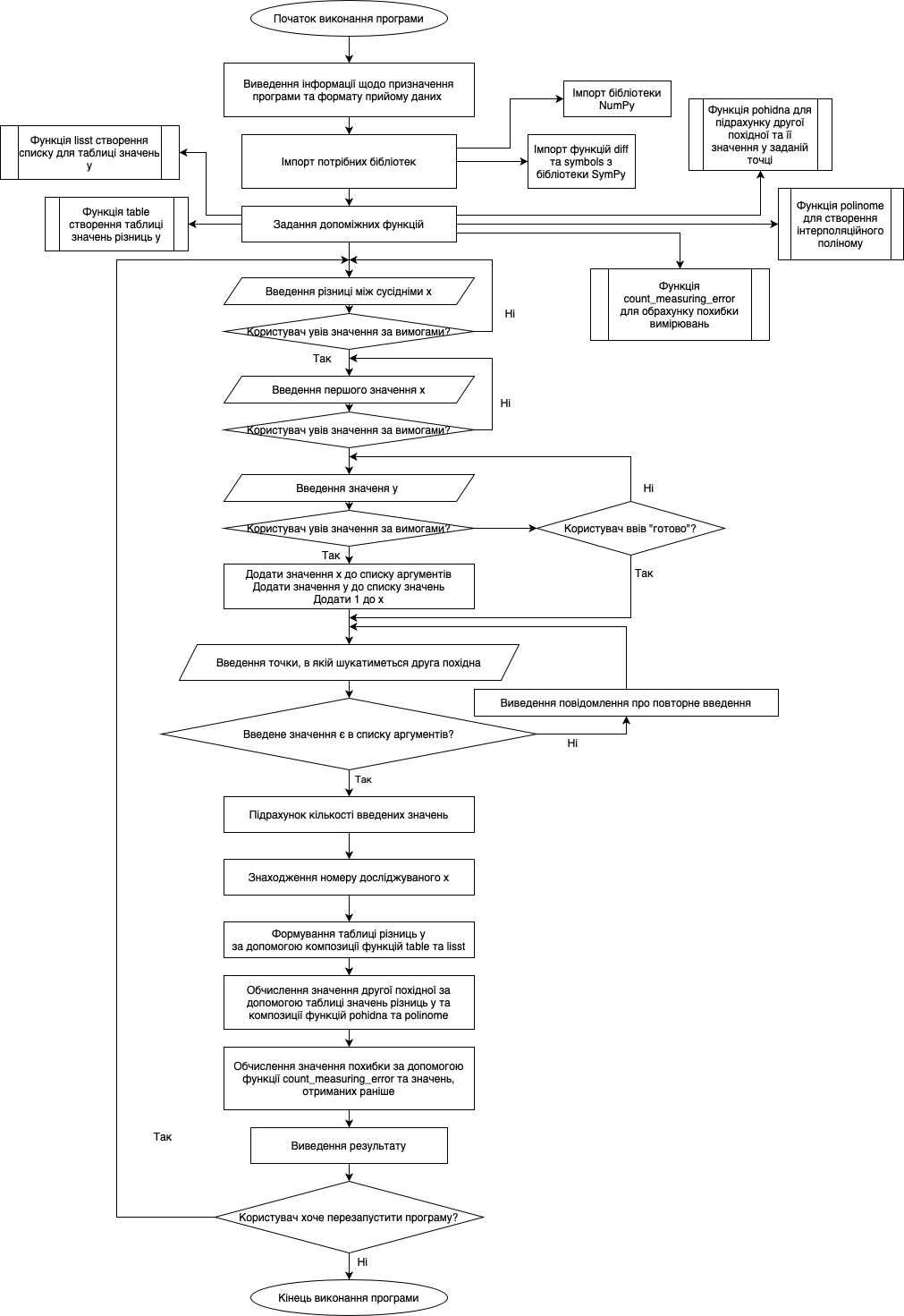
Додаток 2.1. Формула інтерполяційного поліному Ньютона

, 

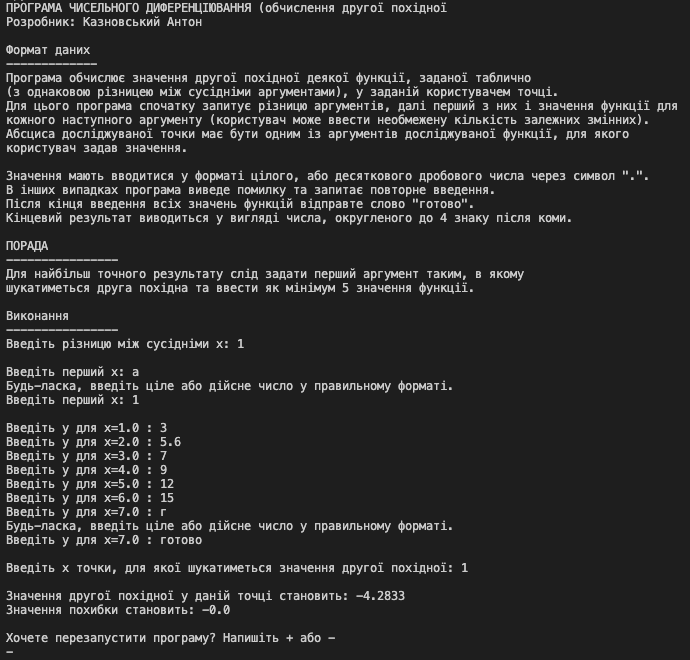
Додаток 2.2. Пояснення змінних формули



Додаток 2.4. Формула знаходження похибки



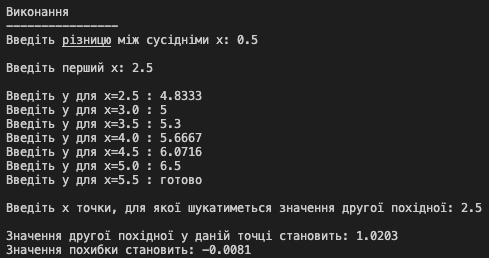
Додаток 2.2.1.1. Детальна блок-схема роботи програми



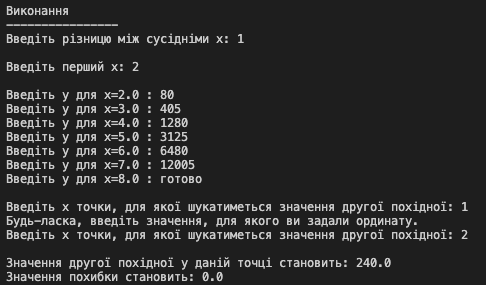
Додаток 2.2.3.1. Приклад реалізації програми



Додаток 3.4.1. Тестування програми за першим контрольним прикладом



Додаток 3.4.2. Тестування програми за другим контрольним прикладом



Додаток 3.4.3. Тестування програми за третім контрольним прикладом