НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

до лабораторної роботи №2

із дисципліни «Алгоритми і системи комп'ютерної математики-1. Математичні алгоритми»

на тему

«Чисельне диференціювання й інтегрування»

|  |  |
| --- | --- |
| Виконала: | Керівник: |
| студентка групи КМ-81 | Ст. викладач |
| Верзун П.В. | Ліскін В.О. |

Київ — 2021

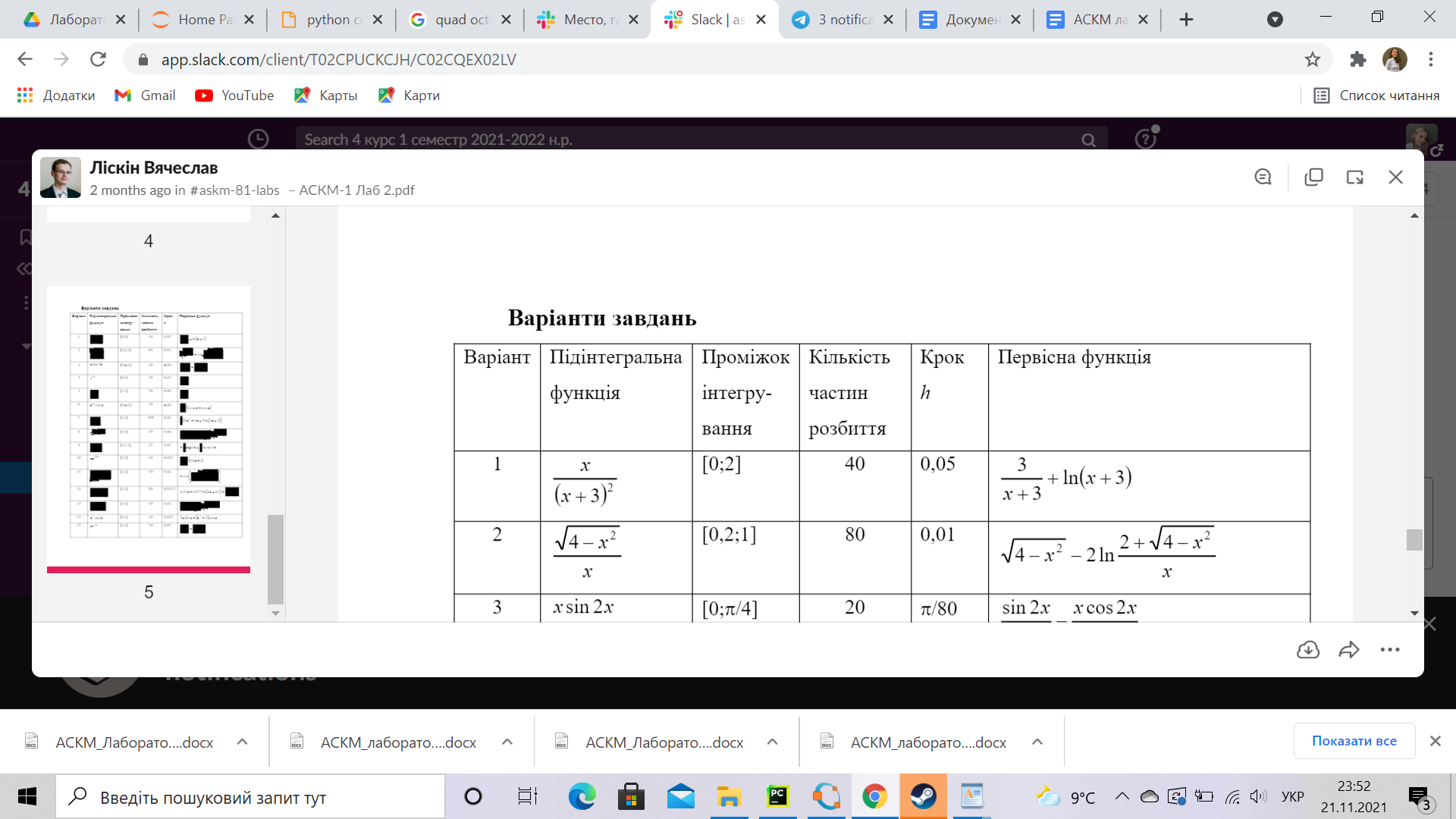
**ВСТУП**

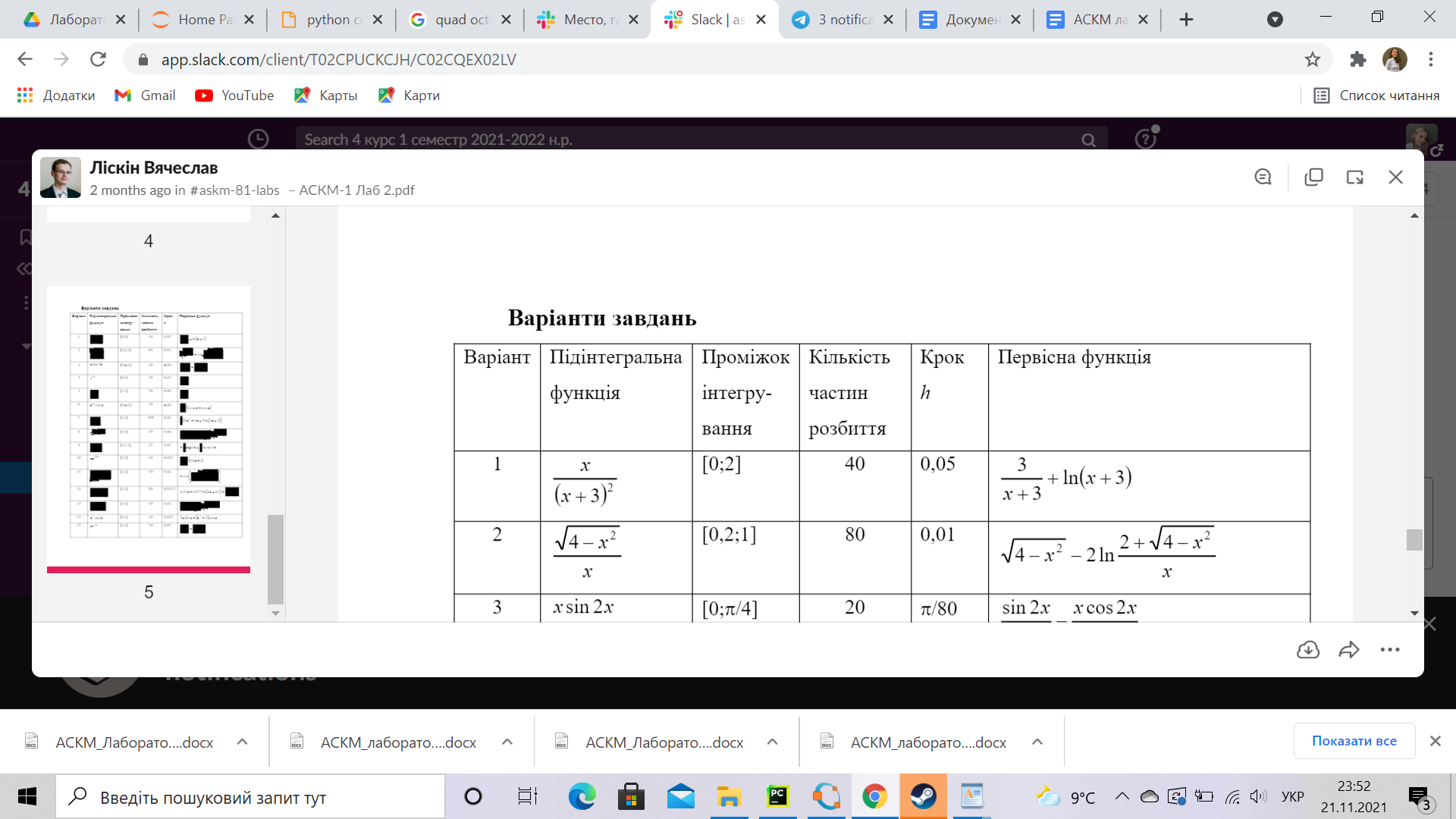
**Мета**: ознайомитися з програмними засобами чисельного диференціювання й інтегрування функцій; практичне розв’язання задач з використанням СКМ, порівняльний аналіз методів чисельного диференціювання й інтегрування.

Був реалізований метод чисельного інтегрування Сімпсона, а також метод чисельного диференціювання за допомогою інтерполяційного многочлену Лагранжа.

Код кожної програми наведено у додатку А.

Варіант №2

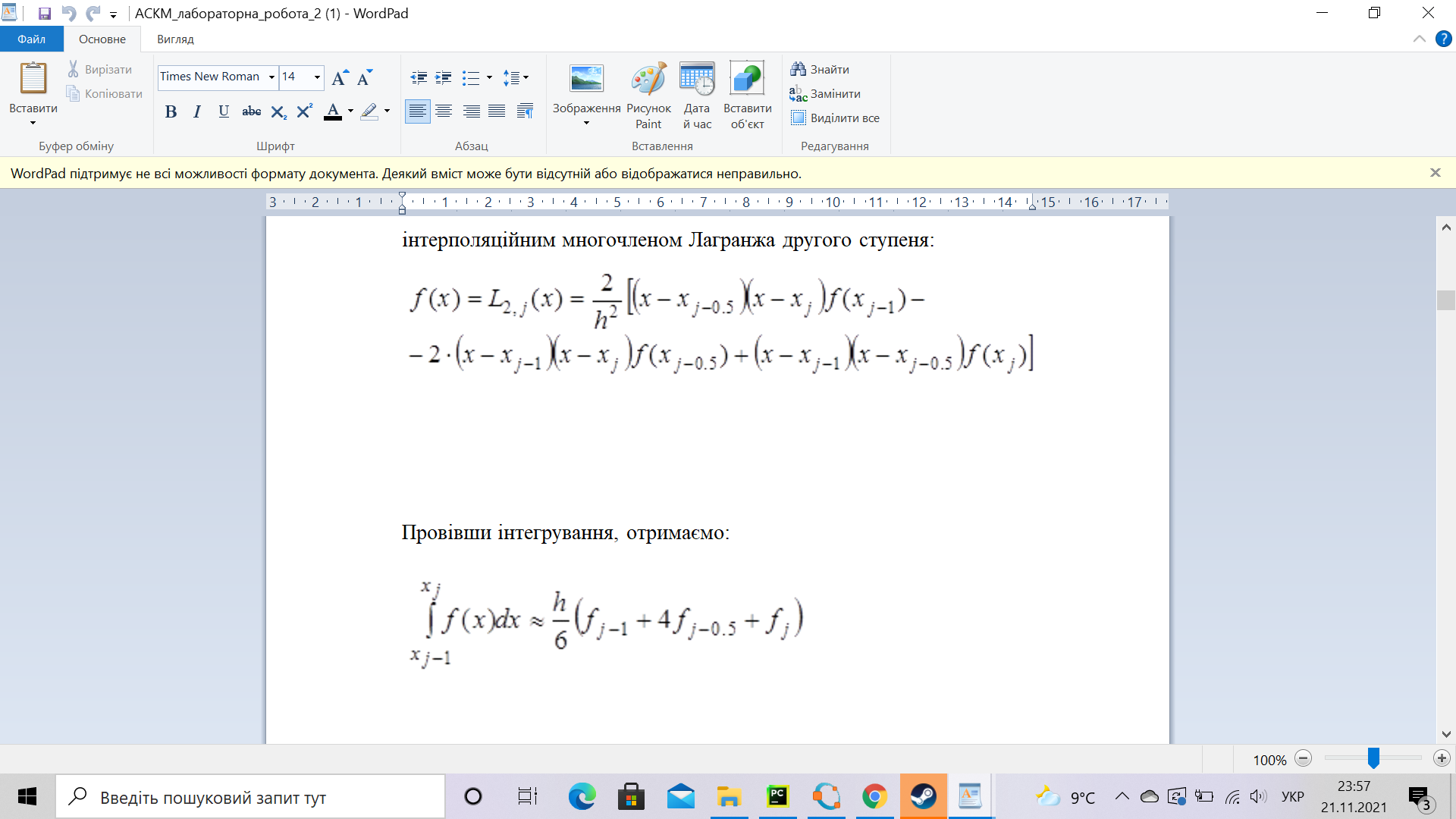




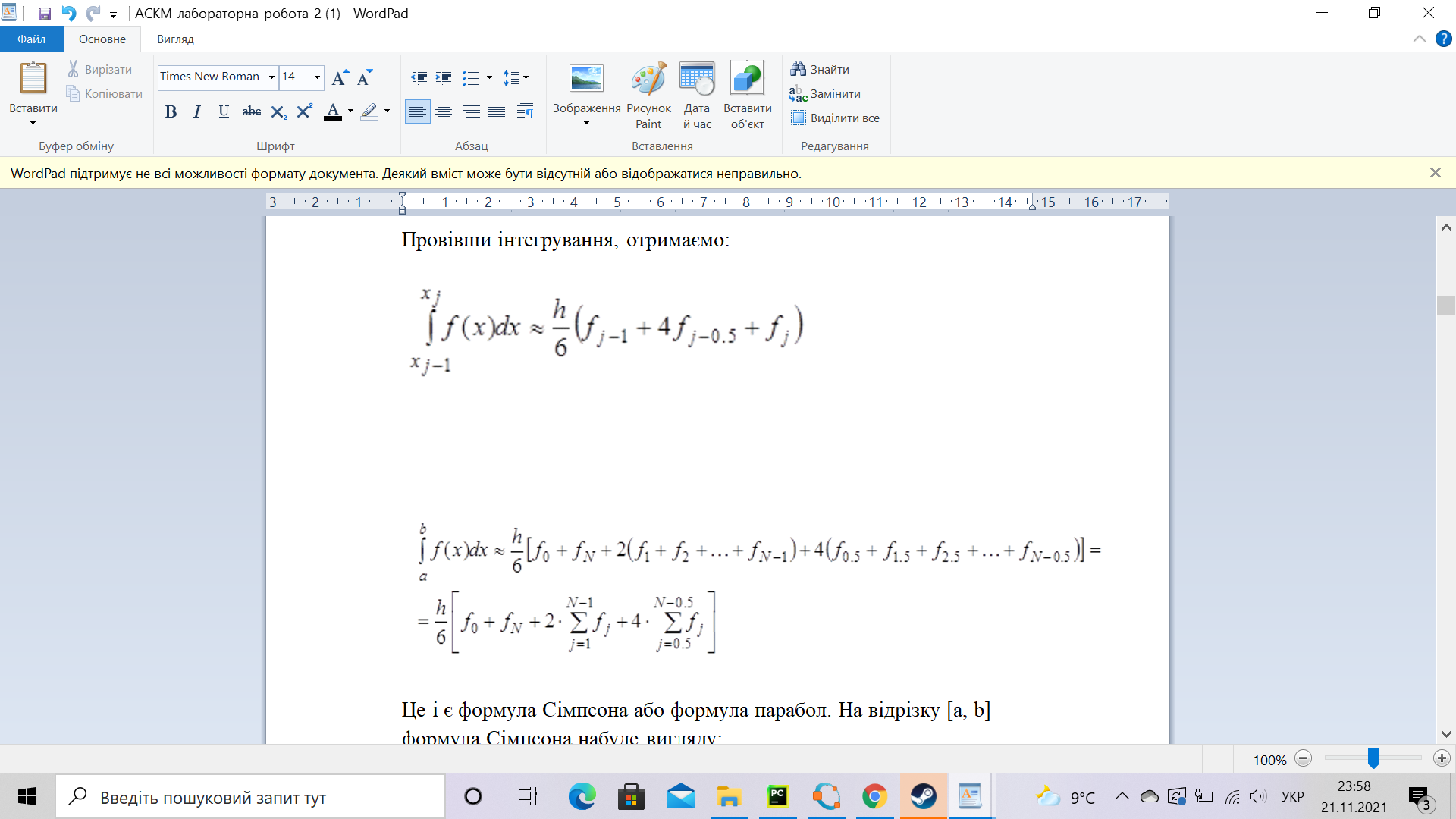
**ТЕОРЕТИЧНА ЧАСТИНА**

Метод Сімпсона

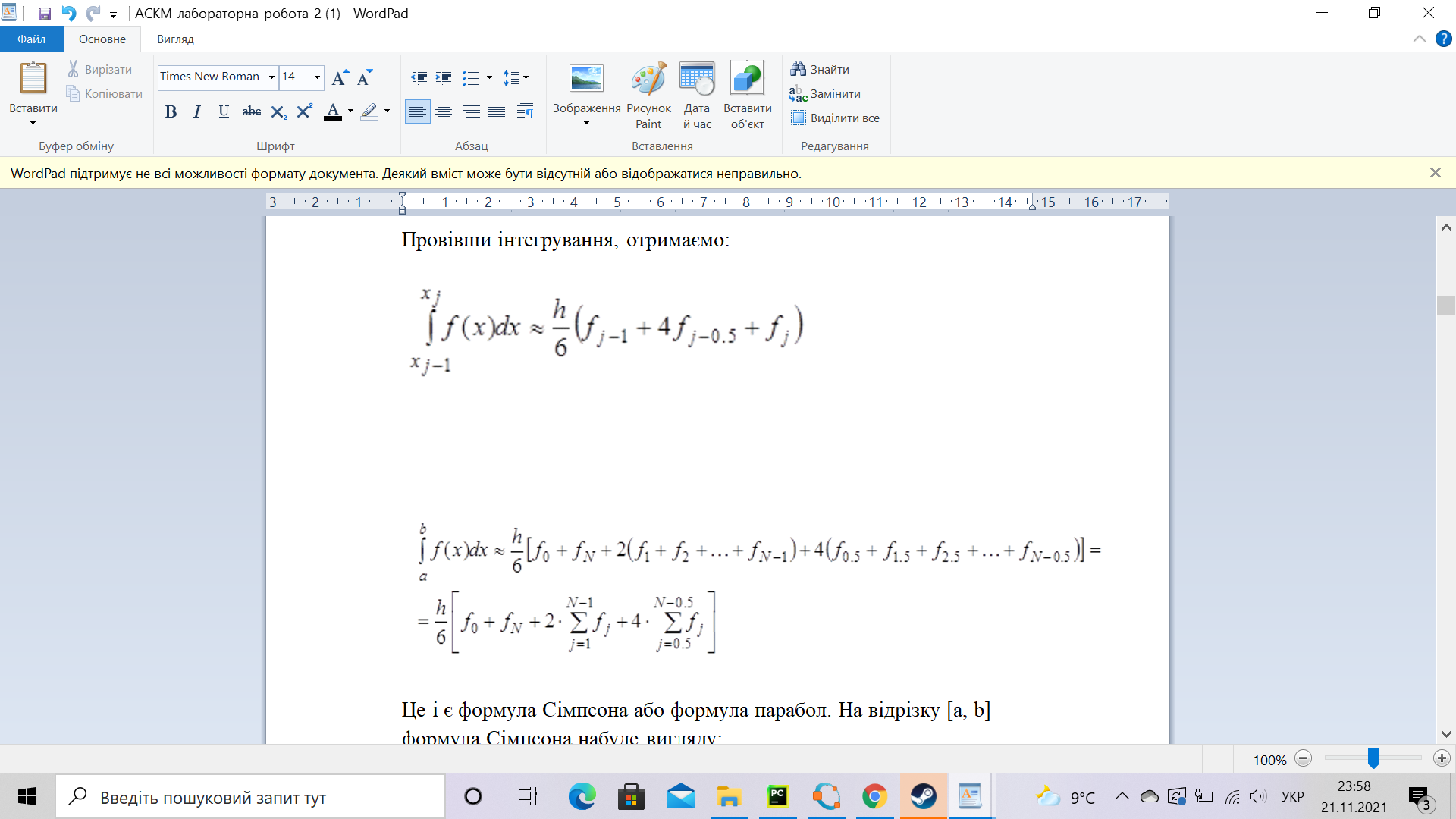
У цьому методі підінтегральна функція на частковому відрізку апроксимується параболою, що проходить через три точки тобто інтерполяційним многочленом Лагранжа другого ступеня:



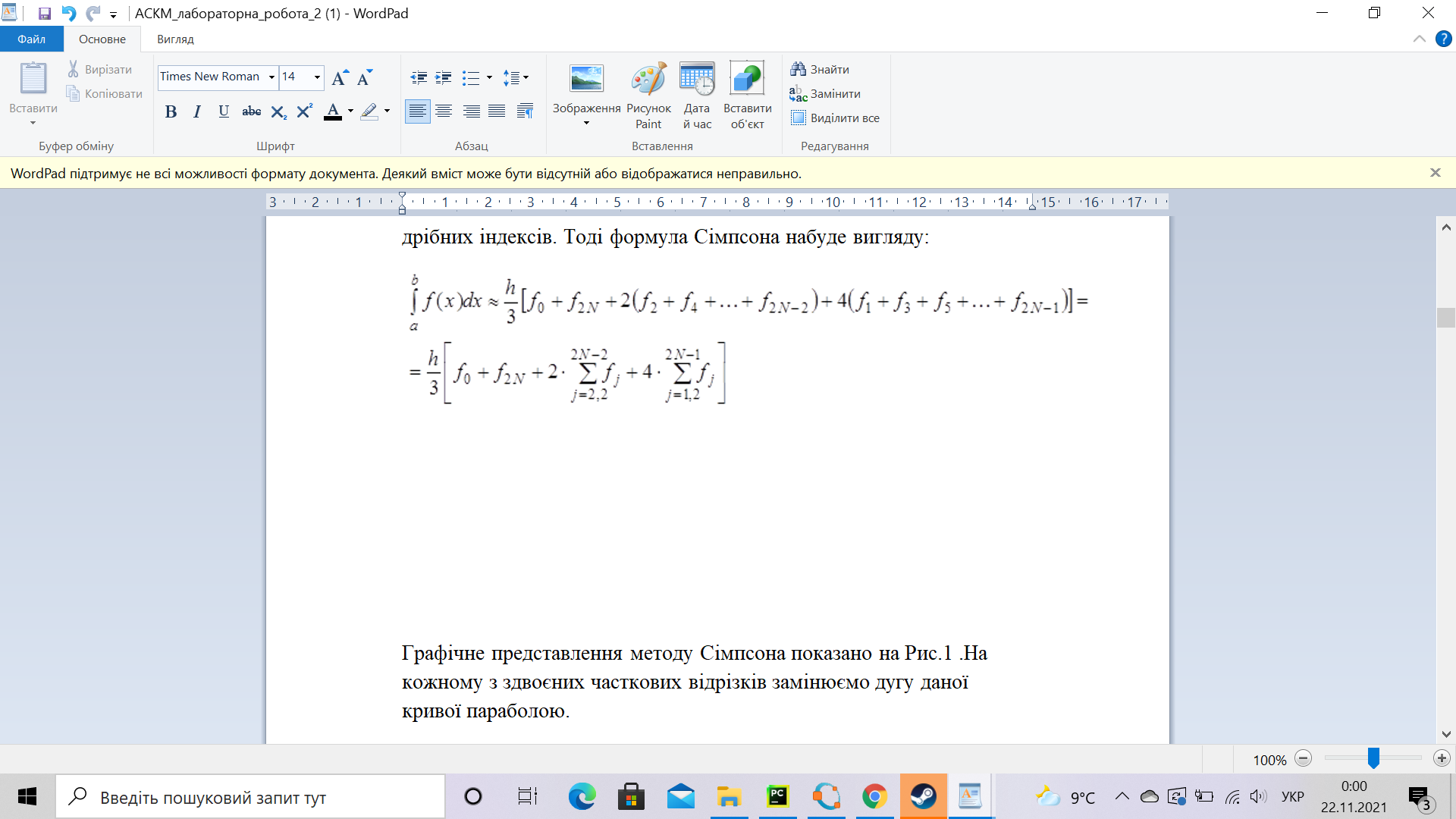
Провівши інтегрування, отримаємо:



Це і є формула Сімпсона або формула парабол. На відрізку [a, b] формула Сімпсона набуде вигляду:

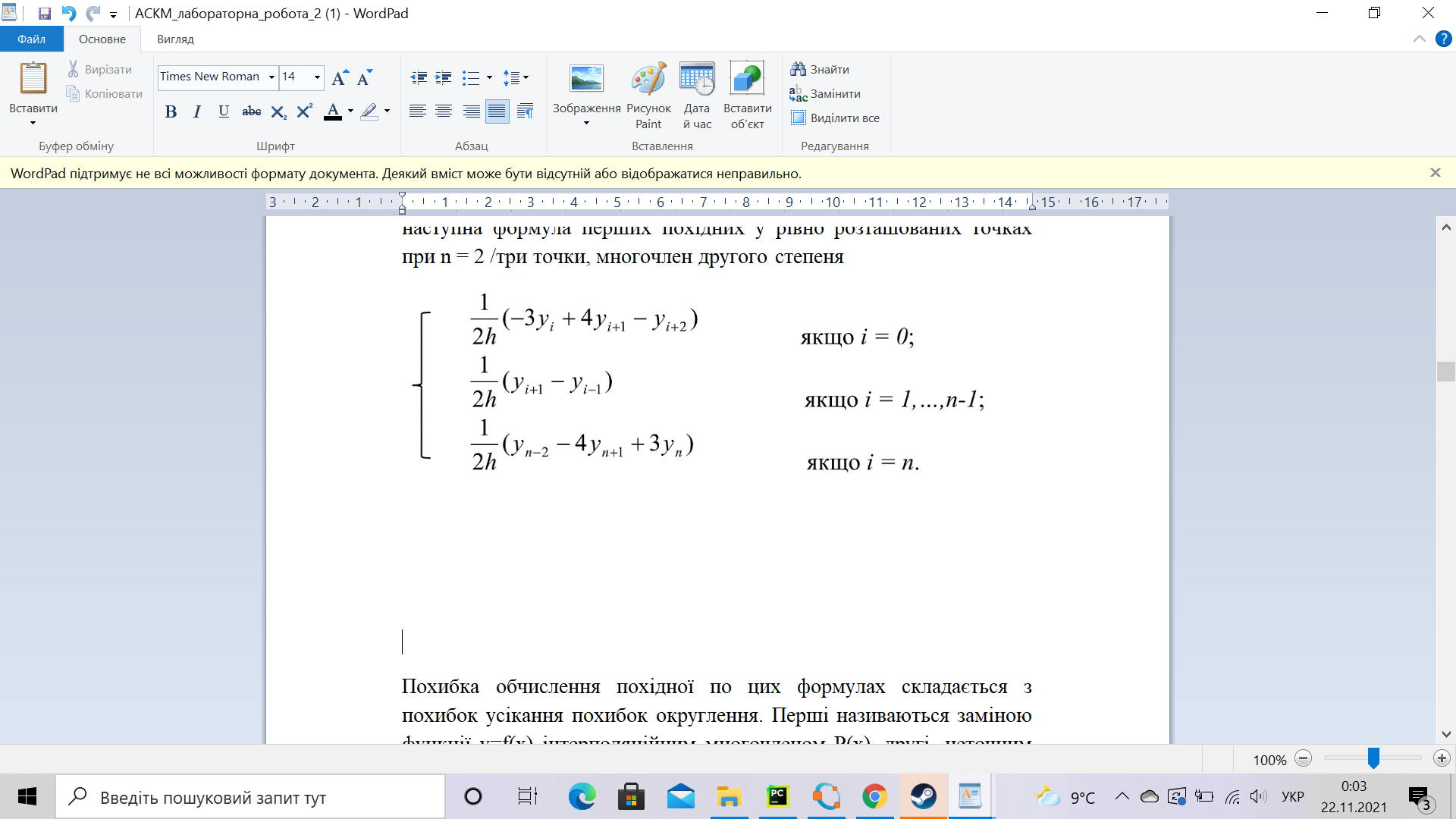


Якщо розбити відрізок інтегрування на парну кількість 2N рівних частин з кроком, то можна побудувати параболу на кожному здвоєному частковому відрізку і переписати вирази без дрібних індексів. Тоді формула Сімпсона набуде вигляду:



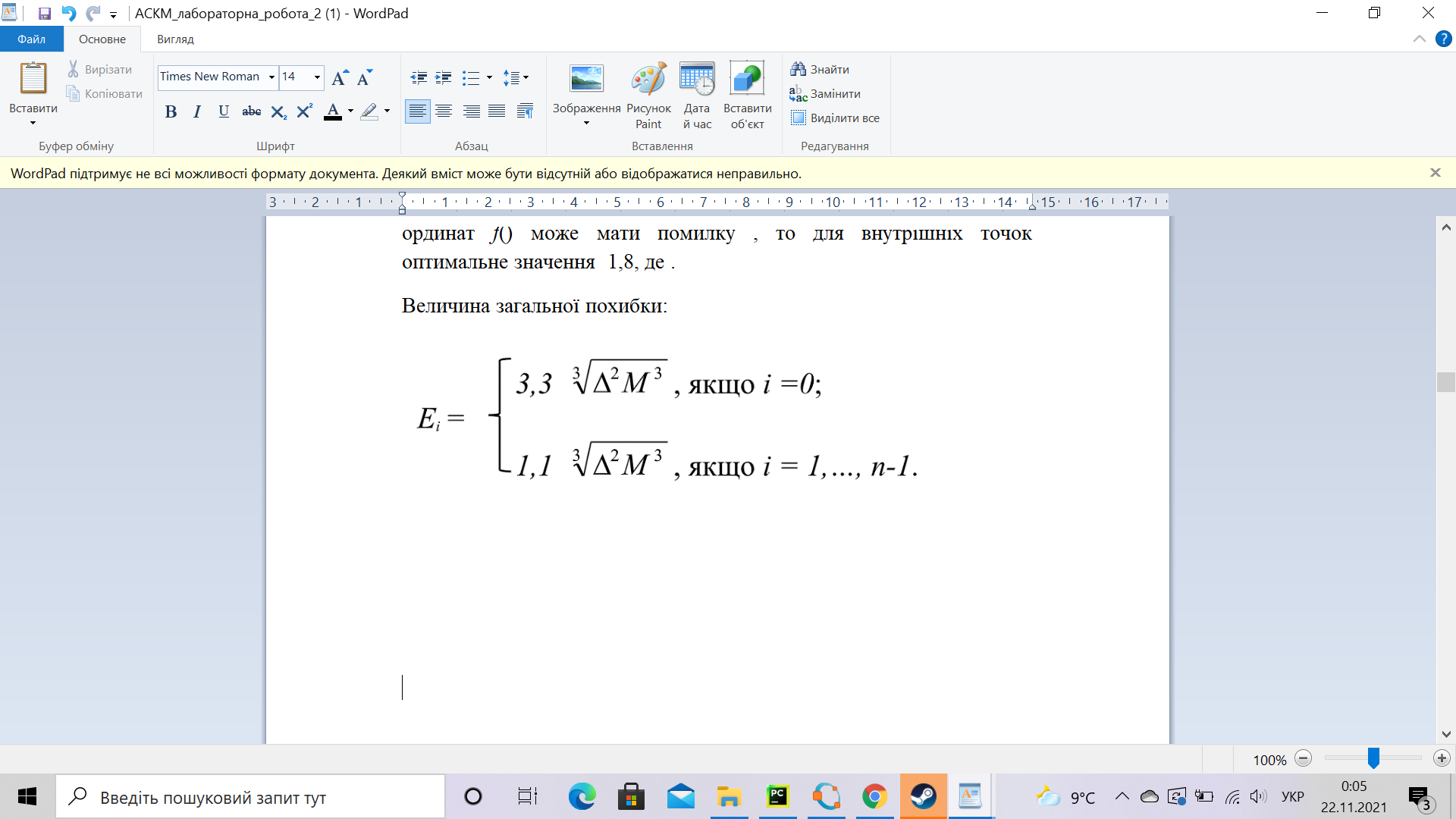
Метод диференціювання з використанням інтерполяційного многочлену Лагранжа

Для одержання формул чисельного диференціювання часто застосовують інтерполяційний многочлен Лагранжа. Істотні зручності для обчислень забезпечують таблиці значень функцій у точках, розташованих з постійним кроком h = x i – x i-1 , i = i,…, n. З урахуванням цього :



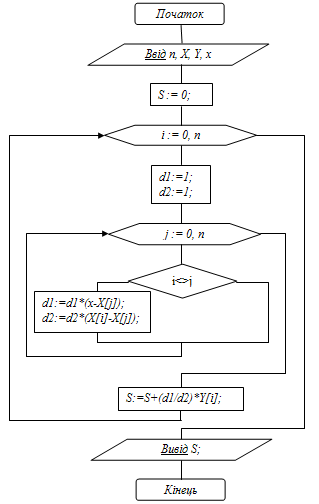
Похибка обчислення похідної по цих формулах складається з похибок усікання похибок округлення. Перші називаються заміною функції y=f(x) інтерполяційним многочленом P(x), другі- неточним значенням . Зі зменшенням кроку h похибка усікання зменшується, а похибка округлення збільшується.

Величина загальної похибки:

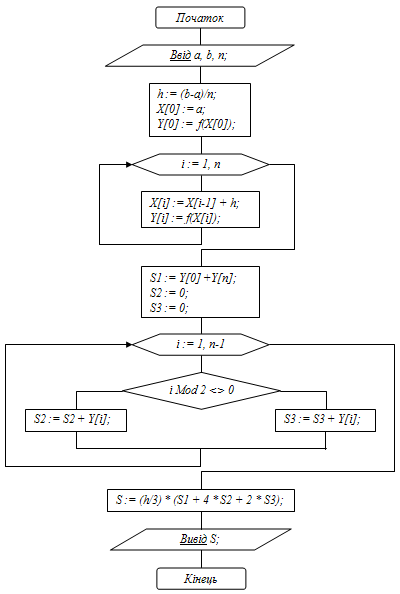


БЛОК – СХЕМИ АЛГОРИТМІВ

Метод Лагранжа

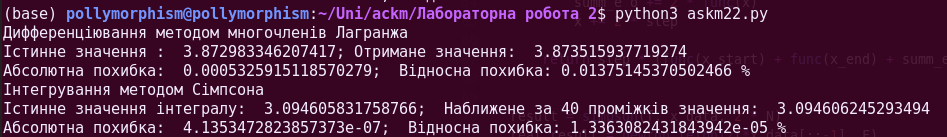


Метод Сімпсона

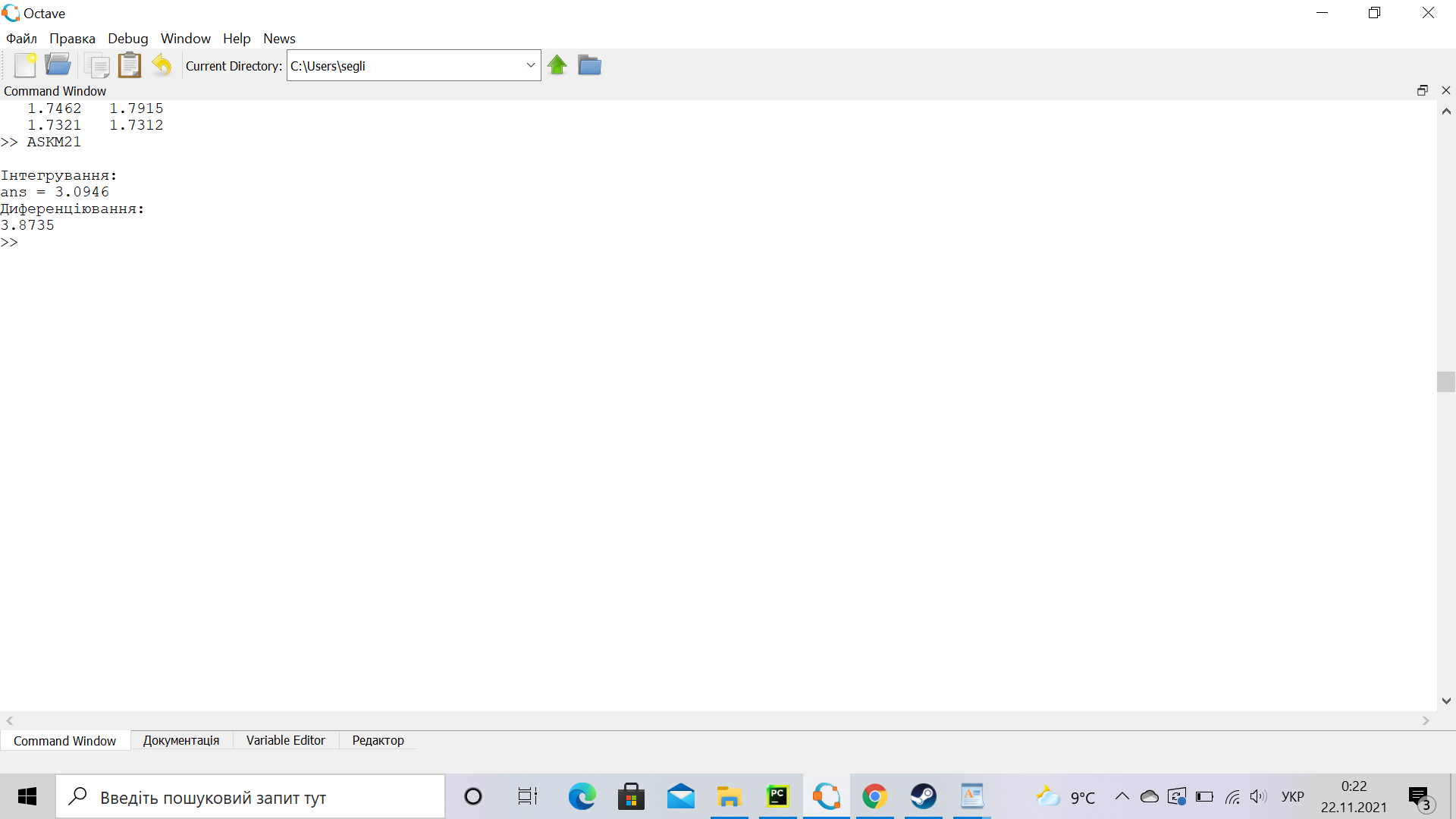


**РЕЗУЛЬТАТИ ВИКОНАННЯ РОБОТИ**

Реалізація на python



Реалізація на Octave:



ДОДАТОК 1

Python

import math

N = 40

h = 0.01

x\_data = [0.2, 1]

f = lambda x: math.sqrt(4 - x \*\* 2) / x

F = lambda x: math.sqrt(4 - x \*\* 2) - 2 \* math.log((2 + math.sqrt(4 - x \*\* 2)) / x)

newton\_leibniz = lambda B, A, func: func(B) - func(A)

def f\_by\_step(x\_start, func, step, n):

return [func(x\_start + idx \* step) for idx in range(n + 1)]

print("Дифференціювання методом многочленів Лагранжа")

def lagrange\_polinom(f\_arr, i, step, n):

if i == 0:

return (-3 \* f\_arr[i] + 4 \* f\_arr[i + 1] - f\_arr[i + 2]) / (2 \* step)

elif i == n:

return (f\_arr[i - 2] - 4 \* f\_arr[i - 1] + 3 \* f\_arr[i]) / (2 \* step)

else:

return (f\_arr[i + 1] - f\_arr[i - 1]) / (2 \* step)

def lagrange(func, interval, step, x):

x\_start, x\_end = interval

n = int((x\_end - x\_start) / step)

f\_arr = f\_by\_step(x\_start, func, step, n)

for idx in range(n):

if x\_start + idx \* step <= x and x < x\_start + (idx + 1) \* step:

return lagrange\_polinom(f\_arr, idx, step, n)

x\_ = 0.5

result = lagrange(F, x\_data, h, x\_)

real\_result = f(x\_)

print(f"Iстинне значення : {real\_result}; Отримане значення: {result}")

print(

f"Абсолютна похибка: {abs(result-real\_result)}; Вiдносна похибка: {abs(result-real\_result) / real\_result \* 100} %"

)

print("Інтегрування методом Сімпсона")

def simpson(func, interval, n):

x\_start, x\_end = interval

step = (x\_end - x\_start) / n

# Odd

summ\_e\_o = 0

x = x\_start + step

for i in range(n // 2):

summ\_e\_o += 4 \* func(x)

x += 2 \* step

# Even

x = x\_start + 2 \* step

for j in range(n // 2 - 1):

summ\_e\_o += 2 \* func(x)

x += 2 \* step

return step \* (func(x\_start) + func(x\_end) + summ\_e\_o) / 3

result = simpson(f, x\_data, 2 \* N)

real\_result = newton\_leibniz(\*x\_data[::-1], F)

print(

f"Iстинне значення iнтегралу: {real\_result}; Наближене за {N} промiжкiв значення: {result}"

)

print(

f"Абсолютна похибка: {abs(result-real\_result)}; Вiдносна похибка: {abs(result-real\_result) / real\_result \* 100} %"

)

Octave

disp("Інтегрування:")

quad('sqrt(4 - x\*\*2)/x',0.2,1,0.01)

disp("Диференціювання:")

x\_start = 0.2;

x\_end = 1;

h\_dif = 0.01;

N = 40;

f = @(x) sqrt(4 - x.^2);

F = @(x) sqrt(4 - x.^2) - 2\*log((2 + sqrt(4 - x.^2)) / x);

x = linspace(x\_start, x\_end, N);

h = (x\_end - x\_start) / 2 \* (N - 1);

xi = x\_start:h:x\_end;

real = h / 3 \* (f(xi(1)) + 2 \* sum(f(xi(3: 2: end - 2))) + 4 \* sum(f(xi(2: 2: end))) + f(xi(end)));

real\_result = F(x\_end) - F(x\_start);

real;

x = 0.5;

x0 = x\_start:h\_dif:x\_end;

n = size(x0, 2) - 1;

new\_real = 0;

for idx=1:n

if ((x < x0(idx+1)) && (x >= x0(idx)))

if (idx == 0)

new\_real += (-3 \* F(x0(idx)) + 4 \* F(x0(idx + 1)) - F(x0(idx + 2))) / (2 \* h\_dif);

break;

elseif (idx == n)

new\_real += (3 \* F(x0(idx)) - 4 \* F(x0(idx - 1)) + F(x0(idx - 2))) / (2 \* h\_dif);

break;

else

new\_real += (F(x0(idx + 1)) - F(x0(idx - 1))) / (2 \* h\_dif);

break;

endif;

endif;

endfor;

disp(new\_real);