Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Інститут комп’ютерних наук та інформаційних технологій

Кафедра програмного забезпечення



**Звіт**

Про виконання лабораторної роботи № 10

**На тему:**

«Чисельні методи інтегрування»

**Лектор:**

доц. каф. ПЗ

Мельник Н. Б.

**Виконав:**

ст. гр. ПЗ-18

Юшкевич А.І.

**Прийняв:**

проф. каф. ПЗ

Гавриш В.І.

« … » … 2023 р.

∑ = \_\_\_\_\_\_\_\_\_\_\_\_

Львів – 2023

**Тема роботи:** Чисельні методи інтегрування.

**Мета роботи:** Ознайомлення на практиці з чисельними методами інтегрування.

**Теоретичні відомості**

У багатьох наукових і технічних задачах інтегрування функцій є важливою складовою частиною математичного моделювання площ і об’ємів, значень роботи змінної сили та інших технічних завдань. Нагадаємо, що геометричний зміст найпростішого визначеного інтеграла

, (1.1)

від *f*(*x*) ≥ 0, як відомо, полягає у тому, що значення величини *I* – це площа, обмежена кривою *y* = *f*(*x*), віссю абсцис та прямими *x* = *a*, *x* = *b.*

У випадках, коли підінтегральна функція задана в аналітичному вигляді, визначений інтеграл можна обчислити безпосередньо за допомогою невизначеного інтеграла за формулою Ньютона-Лейбніца.



Однак на практиці цією формулою часто не можна скористатися через дві основні причини:

1. вигляд функції *f(х)* не допускає безпосереднього інтегрування, тобто первісну не можна зобразити елементарними функціями;
2. значення функції *f(х)* задане тільки на фіксованій скінченній кількості точок *хi*, тобто функція задана у вигляді таблиці.

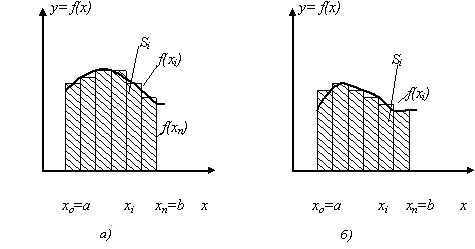
**Метод прямокутників**

Правих:

 (2.1)

Лівих:

 (2.2)



*а) б)*

*Рисунок 2.1 Геометрична інтерпретація методу прямокутників*

Середніх:

. (2.3)

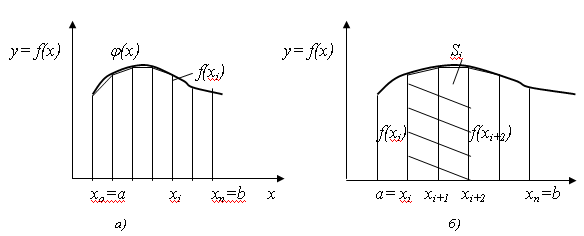
Похибка обчислень інтегралу методом прямокутників обчислюється за формулою:

. (2.4)

**Метод трапецій**

Метод трапецій полягає в тому, що відрізок інтегрування [*а, b*]розбивається на *n* рівних відрізків, всередині яких підінтегральна крива *f(x)* замінюється кусково-лінійною функцією *ϕ(х),* отриманою стягуванням ординат *n* відрізків [*xi-1, xi* ] хордами.

Інтеграл знаходиться як сума площ *si* прямокутних трапецій (pис.3.1 *а,б*).



*а) б)*

*Рисунок 3.1 Геометрична інтерпретація методу трапецій*

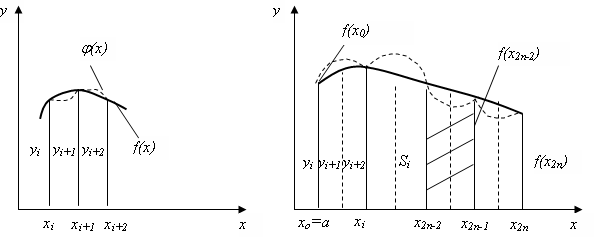
Площа кожної такої трапеції визначається як

. (3.1)



**Метод Сімпсона**

Поділимо відрізок інтегрування [*а*, *b*] на парне число *n* рівних частин з кроком . На кожному відрізку [*х*0, *х*2], [*х*2, *х*4], …, [*хi-*1, *хi*+1], …, [*хn*-2, *хn*] підінтегральну функцію *f*(*х*) замінимо інтерполяційним многочленом другого степеня (квадратичною параболою, рис. 4.1, *а,б*) та обчислення визначеного інтеграла зводиться до обчислення суми площ *n* криволінійних трапецій *Si .*



*а) б)*

*Рисунки 4.1 Геометрична інтерпретація методу Сімпсона*

Площа кожної такої криволінійної трапеції визначається за формулою Сімпсона:

. (4.1)

Отже, сума всіх криволінійних трапецій обчислюється за формулою



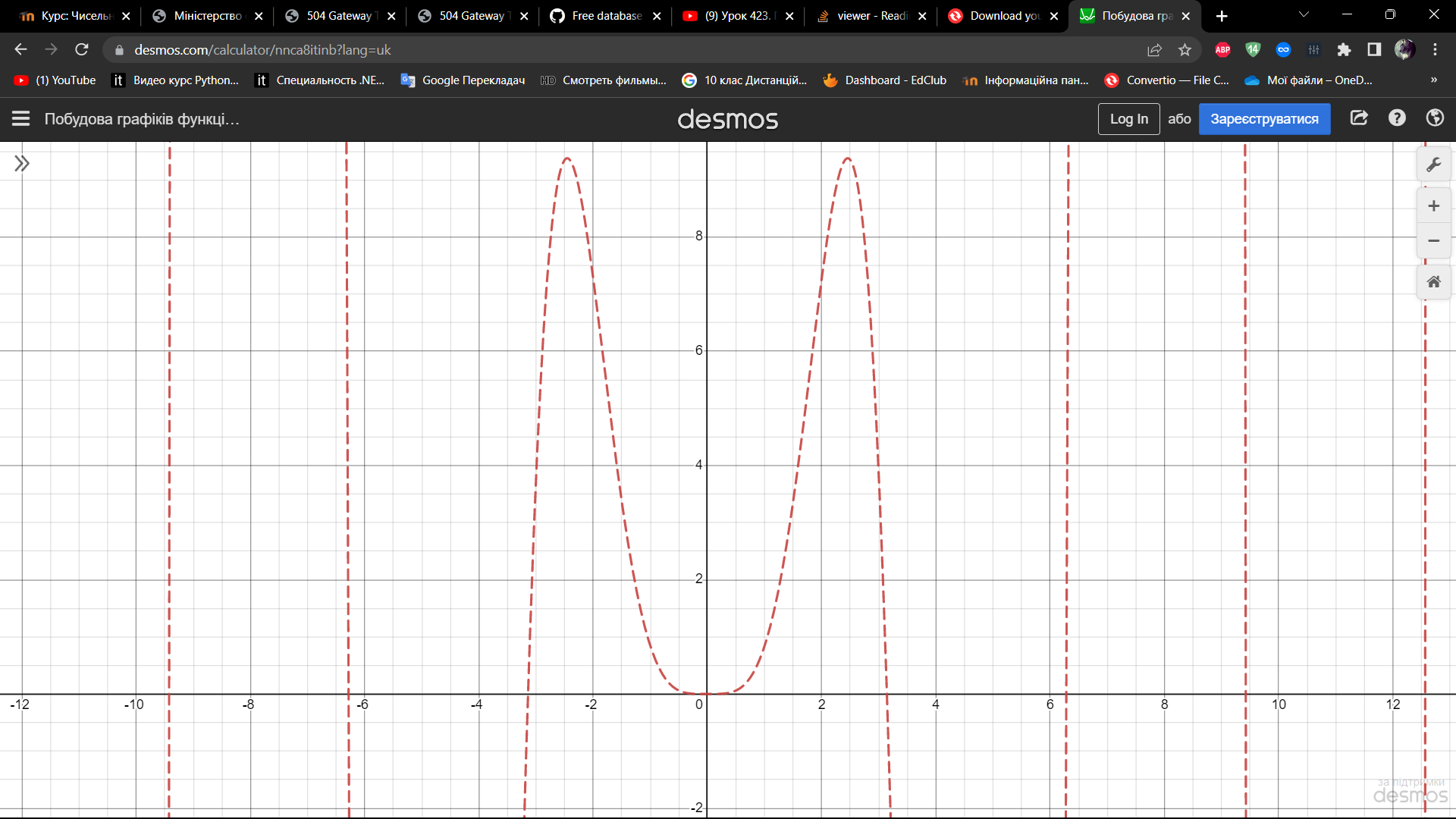
або  (4.3)

**Індивідуальне завдання**

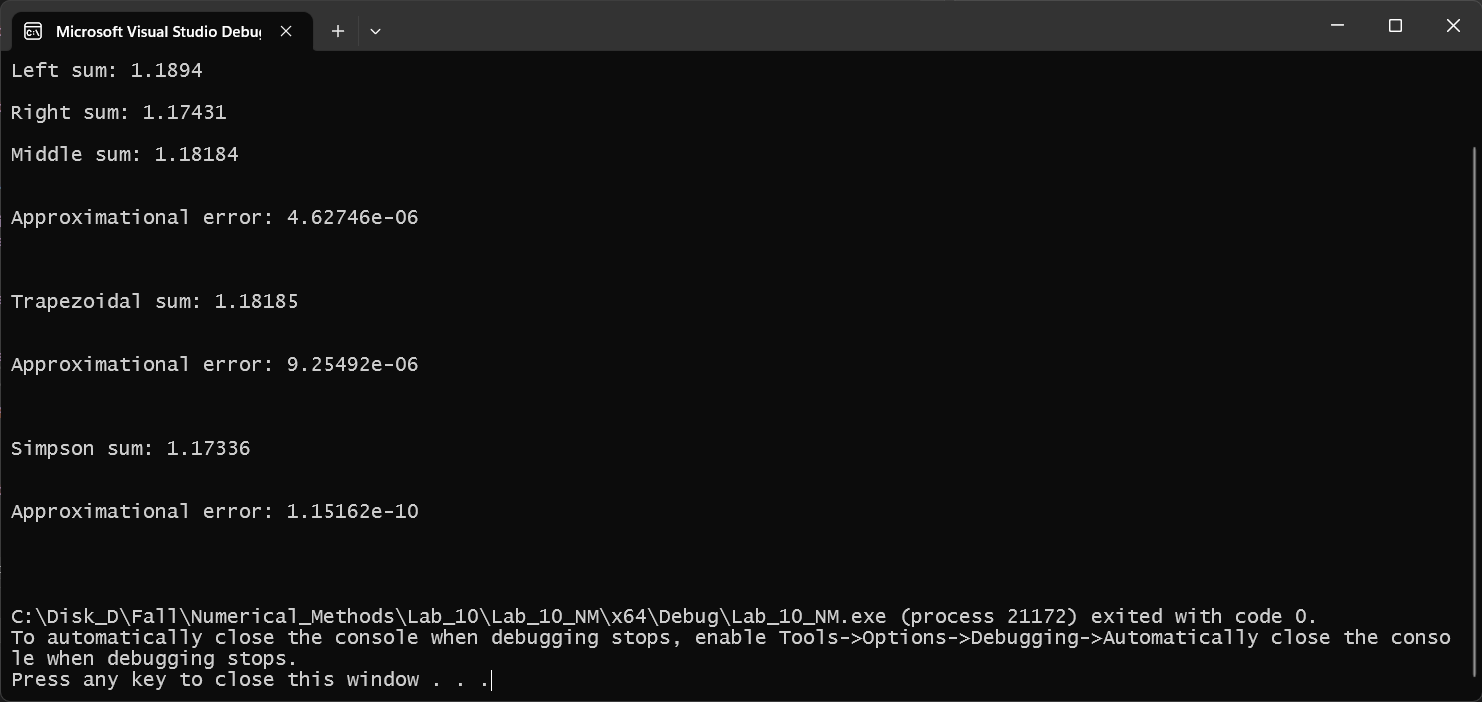
Методом найменших квадратів побудувати лінійний, квадратичний і кубічний апроксимаційні поліноми для таблично заданої функції.

*Задана функція:*

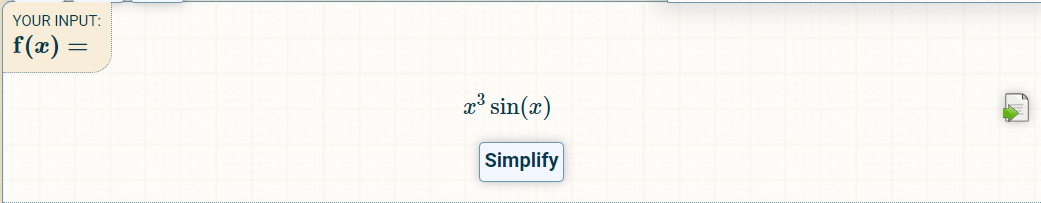
*Геометричне зображення заданої функції:*

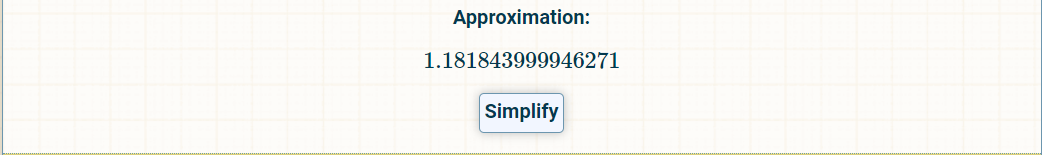


**Результат виконання програми**

****

**Аналітичний розв’язок визначеного інтегралу**

****

****

**Висновки**

У результаті виконання лабораторної роботи використано на практиці чисельні методи інтегрування, а саме: метод прямокутників (лівих, середніх та правих), метод трапецій та метод парабол (Сімпсона). Результат отриманий з використанням методу Сімпсона є найбільш точним, оскільки похибка обчислення є найменшою (похибка обчислення =), метод прямокутників має похибку обчислення . Методом трапеції отримана похибка . Найчастіше на практиці використовують метод середніх прямокутників, але якщо це можливо, то намагаютья використати методи Сімпсона та трапецій.

**Додаток**

Integration.h:

#pragma once

#include <iostream>

#include <vector>

#include <cmath>

#include <numbers>

#include <algorithm>

#include <numeric>

using namespace std;

enum ETypes {

e\_riemann,

e\_trapezoidal,

e\_simpson

};

class Integration

{

public:

Integration() = delete;

Integration(double left\_border, double right\_border, unsigned int n, double (\*func)(double x));

double RiemannRightSum();

double RiemannLeftSum();

double RiemannMiddleSum();

double Trapezoidal();

double Simpson();

double GetError(ETypes type);

double FindStep(double epsilon);

private:

vector<double> m\_x, m\_y;

double m\_left\_border, m\_right\_border, m\_step, m\_num\_of\_steps;

double (\*m\_func)(double x);

double GetSecondDerivative(double x);

double GetFourthDerivative(double x);

double GetM();

};

Integration.cpp:

#include "Integration.h"

Integration::Integration(double left\_border, double right\_border, unsigned int n, double (\*func)(double x)) {

m\_left\_border = left\_border;

m\_right\_border = right\_border;

m\_func = func;

m\_num\_of\_steps = n;

double mid = fabs(right\_border - left\_border) / n;

m\_step = mid;

double x{ left\_border };

for (int i = 0; i <= n; i++) {

m\_x.push\_back(x);

m\_y.push\_back(func(x));

x += mid;

}

}

double Integration::RiemannRightSum() {

return m\_step \* accumulate(m\_y.begin(), m\_y.end() - 1, 0.0);

}

double Integration::RiemannLeftSum() {

return m\_step \* accumulate(m\_y.begin() + 1, m\_y.end(), 0.0);

}

double Integration::RiemannMiddleSum() {

vector<double> special\_y;

for (const auto& element : m\_x) {

special\_y.push\_back(m\_func(element + m\_step / 2.0));

}

return m\_step \* accumulate(special\_y.begin(), special\_y.end() - 1, 0.0);

}

double Integration::GetSecondDerivative(double x) {

return -pow(x, 3) \* sin(x) + 6 \* pow(x, 2) \* cos(x) + 6 \* x \* sin(x);

}

double Integration::GetFourthDerivative(double x) {

return pow(x, 3) \* sin(x) - 12 \* pow(x, 2) \* cos(x) - 36 \* x \* sin(x) + 24 \* cos(x);

}

double Integration::GetError(ETypes type) {

double result{ 0 };

switch (type) {

case e\_riemann:

result = fabs(GetSecondDerivative(m\_left\_border) \* (m\_right\_border - m\_left\_border) \* pow(m\_step, 2) / 24);

break;

case e\_trapezoidal:

result = fabs(GetSecondDerivative(m\_left\_border) \* (m\_right\_border - m\_left\_border) \* pow(m\_step, 2) / 12);

break;

case e\_simpson:

result = fabs(GetM() \* pow(m\_right\_border - m\_left\_border, 5) / (180 \* pow(m\_num\_of\_steps, 4)));

break;

default:

break;

}

return result;

}

double Integration::Trapezoidal() {

double accum{ accumulate(m\_y.begin() + 1, m\_y.end() - 1, 0.0) };

double first{ \*m\_y.begin() };

double end{ \*(m\_y.end() - 1) };

double funcs{ (first + end) / 2.0 };

return m\_step \* (funcs + accum);

}

double Integration::Simpson() {

vector<double> \_2i\_minus\_1;

vector<double> \_2i;

for (int i = 1; i < m\_num\_of\_steps; i++) {

if (!(i % 2))

\_2i\_minus\_1.push\_back(m\_y[i]);

else

\_2i.push\_back(m\_y[i]);

}

return (m\_step / 3.0) \* (\*m\_y.begin() + \*(m\_y.end() - 1.0) + 4.0 \* accumulate(\_2i\_minus\_1.begin(), \_2i\_minus\_1.end(), 0.0) + 2.0 \* accumulate(\_2i.begin(), \_2i.end(), 0.0));

}

double Integration::GetM() {

double max{ fabs(GetFourthDerivative(m\_left\_border)) };

for (const double& element : m\_x) {

if (fabs(GetFourthDerivative(element)) > max)

max = fabs(GetFourthDerivative(element));

}

return max;

}

double Integration::FindStep(double epsilon) {

double max\_derivative{ GetM() };

return pow(0.5 \* epsilon \* 180.0 / ((m\_right\_border - m\_left\_border) \* max\_derivative), 1.0 / 4.0);

}

Lab\_10\_NM:

#include <iostream>

#include "Integration.h"

double f(double x) {

return pow(x, 3) \* sin(x);

}

int main()

{

double epsilon = 0.001;

double left\_border = numbers::pi / 3.0;

double right\_border = numbers::pi / 2.0;

unsigned int division = 100;

Integration integral(left\_border, right\_border, division, f);

cout << "Minimal step for epsilon equal " << epsilon << " is: " << integral.FindStep(epsilon) << "\n\n\n\n";

cout << "Left sum: " << integral.RiemannLeftSum() << "\n\n";

cout << "Right sum: " << integral.RiemannRightSum() << "\n\n";

cout << "Middle sum: " << integral.RiemannMiddleSum() << "\n\n\n";

cout << "Approximational error: " << integral.GetError(e\_riemann) << "\n\n\n\n";

cout << "Trapezoidal sum: " << integral.Trapezoidal() << "\n\n\n";

cout << "Approximational error: " << integral.GetError(e\_trapezoidal) << "\n\n\n\n";

cout << "Simpson sum: " << integral.Simpson() << "\n\n\n";

cout << "Approximational error: " << integral.GetError(e\_simpson) << "\n\n\n\n";

}