**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Національний університет “Львівська політехніка”**

**Інститут ІКНІ**

**Кафедра ПЗ**



**ЗВІТ**

**До лабораторної роботи №10**

**На тему: “** **Чисельні методи інтегрування”**

**З дисципліни : “Чисельні методи”**

**Лектор:**

доц. кафедри ПЗ

Мельник Н.Б.

**Виконав:**

студ. групи ПЗ-18

Кириченко М.К.

**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:

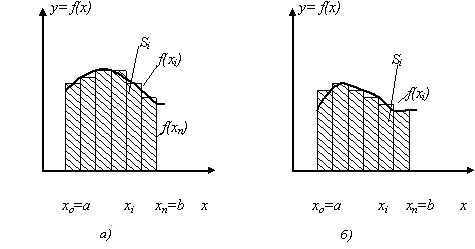
(2.1)



Лівих:

(2.2)





*а) б)*

*Рисунок 2.1 Геометрична інтерпретація методу прямокутників*

Середніх:

. (2.3)



Похибка обчислень інтегралу методом прямокутників обчислюється за формулою:

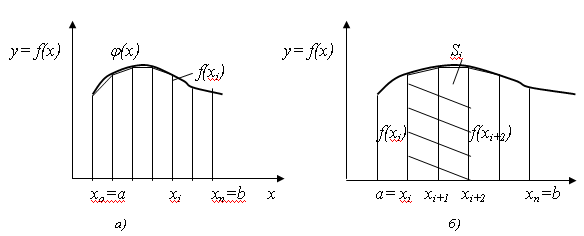
. (2.4)



**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

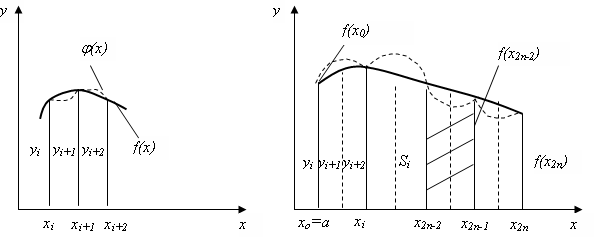
. (3.1)



**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



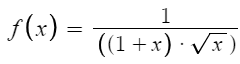
або (4.3)



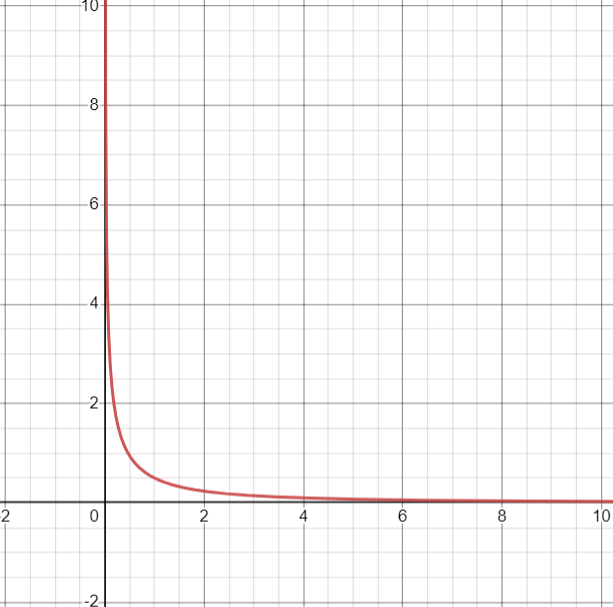
**Індивідуальне завдання**

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

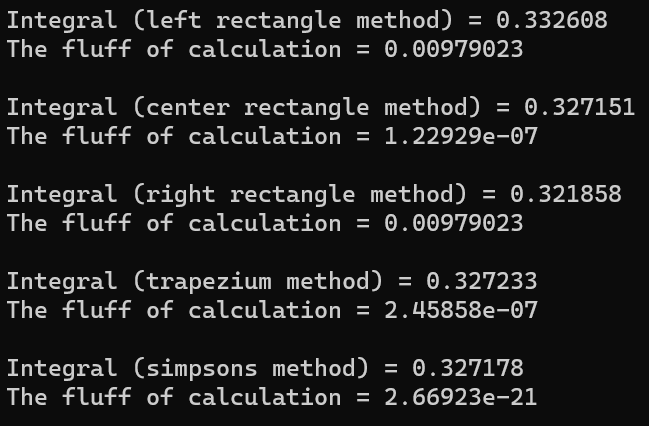
*Задана функція:*



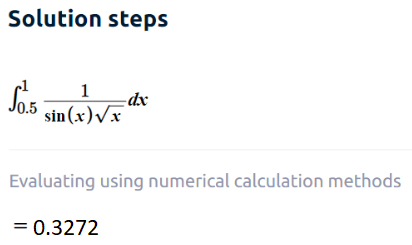
*Геометричне зображення заданої функції:*



**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 =), а метод прямокутників (лівих та правих) - найменш точним (похибка обчислення = ). Методом трапеції отримана похибка , а методом прямокутників (середніх) . Найчастіше на практиці використовують метод середніх прямокутників, але якщо це можливо, то стараються використати методи Сімпсона та трапецій.

**Додаток**

**NumericalIntegration.h**

#pragma once

#include <iostream>

#include <cmath>

struct NumericalIntegration

{

inline double f(const double x);

inline double dfdx(const double x);

inline double df2dx(const double x);

inline double df4dx(const double x);

inline double left\_rectangle\_method(std::pair<double, double>& interval, size\_t n);

inline double center\_rectangle\_method(std::pair<double, double>& interval, size\_t n);

inline double right\_rectangle\_method(std::pair<double, double>& interval, size\_t n);

inline double trapezium\_method(std::pair<double, double>& interval, size\_t n);

inline double simpsons\_method(std::pair<double, double>& interval, size\_t n);

inline double get\_max\_y(std::pair<double, double>& interval, double (NumericalIntegration::\*func)(const double x));

inline double get\_rl\_rectangle\_fluff(std::pair<double, double>& interval, size\_t n);

inline double get\_c\_rectangle\_fluff(std::pair<double, double>& interval, size\_t n);

inline double get\_trapezium\_fluff(std::pair<double, double>& interval, size\_t n);

inline double get\_simpsons\_fluff(std::pair<double, double>& interval, size\_t n);

};

inline double NumericalIntegration::f(const double x)

{

return 1/((1+x)\*sqrt(x));

}

inline double NumericalIntegration::dfdx(const double x)

{

return -1/(2\*sqrt(x\*x\*x)\* (x+1)) - 1/(sqrt(x)\* (x+1) \* (x+1));

}

inline double NumericalIntegration::df2dx(const double x)

{

return 3 / (4 \* sqrt(x \* x \* x \* x \* x) \* (x + 1))

+ 1 / (sqrt(x\*x\*x) \* (x + 1) \* (x + 1))

+ 1 / (sqrt(x) \* (x + 1) \* (x + 1) \* (x + 1));

}

inline double NumericalIntegration::df4dx(const double x)

{

return 105 / (16 \* sqrt(pow(x, 9)) \* (x + 1))

+ 15 / (2 \* sqrt(pow(x, 7)) \* (x + 1) \* (x + 1))

+ 9 / (sqrt(pow(x, 5)) \* (x + 1) \* (x + 1) \* (x + 1))

+ 12 / (sqrt(pow(x, 3)) \* pow((x + 1), 4))

+ 24 / (sqrt(x) \* pow((x + 1), 5));

}

inline double NumericalIntegration::left\_rectangle\_method(std::pair<double, double>& interval, size\_t n)

{

double dx = (interval.second - interval.first) / n;

double sum = 0;

for (size\_t i = 0; i < n-1; i++)

{

sum += f(interval.first + i \* dx) \* dx;

}

return sum;

}

inline double NumericalIntegration::center\_rectangle\_method(std::pair<double, double>& interval, size\_t n)

{

double dx = (interval.second - interval.first) / n;

double sum = 0;

for (size\_t i = 0; i < n - 1; i++)

{

sum += f(interval.first + (i + 0.5) \* dx) \* dx;

}

return sum;

}

inline double NumericalIntegration::right\_rectangle\_method(std::pair<double, double>& interval, size\_t n)

{

double dx = (interval.second - interval.first) / n;

double sum = 0;

for (size\_t i = 1; i < n; i++)

{

sum += f(interval.first + i \* dx) \* dx;

}

return sum;

}

inline double NumericalIntegration::get\_max\_y(std::pair<double, double>& interval, double (NumericalIntegration::\*func)(const double x))

{

double max = (this->\*func)(interval.first);

for (double i = interval.first; i <= interval.second; i = i + 0.001)

{

if (max < (this->\*func)(i))

max = i;

}

return (this->\*func)(max);

}

inline double NumericalIntegration::get\_rl\_rectangle\_fluff(std::pair<double, double>& interval, size\_t n)

{

return abs(get\_max\_y(interval, &NumericalIntegration::dfdx) \* (interval.second - interval.first)

\* (interval.second - interval.first) / (2.0 \* n));

}

inline double NumericalIntegration::get\_c\_rectangle\_fluff(std::pair<double, double>& interval, size\_t n)

{

return abs(get\_max\_y(interval, &NumericalIntegration::df2dx) \* (interval.second - interval.first)

\* (interval.second - interval.first) \* (interval.second - interval.first)

/ (24.0 \* n \* n));

}

inline double NumericalIntegration::get\_trapezium\_fluff(std::pair<double, double>& interval, size\_t n)

{

return abs(get\_max\_y(interval, &NumericalIntegration::df2dx) \* (interval.second - interval.first)

\* (interval.second - interval.first) \* (interval.second - interval.first) / (12.0 \*n\*n));

}

inline double NumericalIntegration::get\_simpsons\_fluff(std::pair<double, double>& interval, size\_t n)

{

return abs(get\_max\_y(interval, &NumericalIntegration::df4dx) \* (interval.second - interval.first)

\* (interval.second - interval.first) \* (interval.second - interval.first)

\* (interval.second - interval.first) \* (interval.second - interval.first)

/ (2880.0 \* n \* n \* n \* n));

}

inline double NumericalIntegration::trapezium\_method(std::pair<double, double>& interval, size\_t n)

{

double dx = (interval.second - interval.first) / n;

double sum = 0;

for (size\_t i = 0; i < n-1; i++)

{

double medianLine = (f(interval.first + i \* dx) + f(interval.first + (i + 1) \* dx)) / 2;

sum += medianLine \* dx;

}

return sum;

}

inline double NumericalIntegration::simpsons\_method(std::pair<double, double>& interval, size\_t n)

{

double dx = (interval.second - interval.first) / n;

double sum = 0;

for (size\_t i = 0; i < n-1; i++)

{

sum += (f(interval.first + i \* dx) + 4 \* f(interval.first + (i + 0.5) \* dx)

+ f(interval.first + (i + 1) \* dx)) \* (dx / 6.0);

}

return sum;

}

**Main.cpp**

#include "NumericalIntegration.h"

int main(void)

{

std::pair<double, double> interval(0.5, 1.0);

size\_t n = 20;

NumericalIntegration obj;

std::cout << "Integral (left rectangle method) = " <<obj.left\_rectangle\_method(interval, n) << std::endl;

std::cout << "The fluff of calculation = " << obj.get\_rl\_rectangle\_fluff(interval, n) << "\n\n";

std::cout << "Integral (center rectangle method) = " << obj.center\_rectangle\_method(interval, n) << std::endl;

std::cout << "The fluff of calculation = " << obj.get\_c\_rectangle\_fluff(interval, n) << "\n\n";

std::cout << "Integral (right rectangle method) = " << obj.right\_rectangle\_method(interval, n) << std::endl;

std::cout << "The fluff of calculation = " << obj.get\_rl\_rectangle\_fluff(interval, n) << "\n\n";

std::cout << "Integral (trapezium method) = " << obj.trapezium\_method(interval, n) << std::endl;

std::cout << "The fluff of calculation = " << obj.get\_trapezium\_fluff(interval, n) << "\n\n";

std::cout << "Integral (simpsons method) = " << obj.simpsons\_method(interval, n) << std::endl;

std::cout << "The fluff of calculation = " << obj.get\_simpsons\_fluff(interval, n) << "\n\n";

}