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

$$I = \int_a^b f(x)dx \quad , \qquad (1.1)$$

від $f(x) \geq 0$, 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.

$$\int_a^b f(x)dx = F(x)\Big|_a^b = F(b) - F(a).$$

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

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

Метод прямокутників

Правих:

$$I_{пр} = \int_a^b f(x)dx \approx h \cdot [f(x_n) + f(x_{n-1}) + ... + f(x_1)] = \frac{b-a}{n}\sum_{i=1}^{n} f(x_i) \qquad (2.1)$$

Лівих:

$$I_l = \int_a^b f(x)dx \approx h \cdot [f(x_{n-1}) + f(x_{n-2}) + ... + f(x_0)] = \frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i) \qquad (2.2)$$

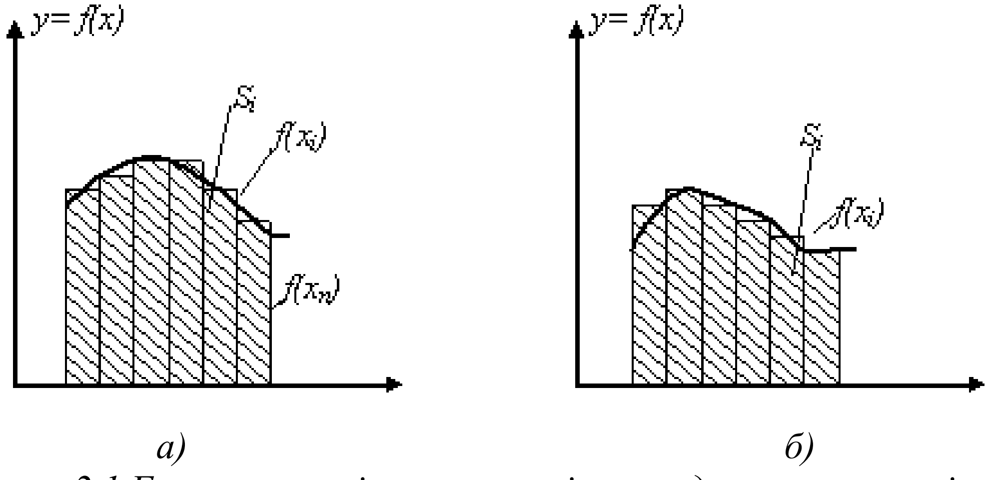


Рисунок 2.1 Геометрична інтерпретація методу прямокутників

Середніх:

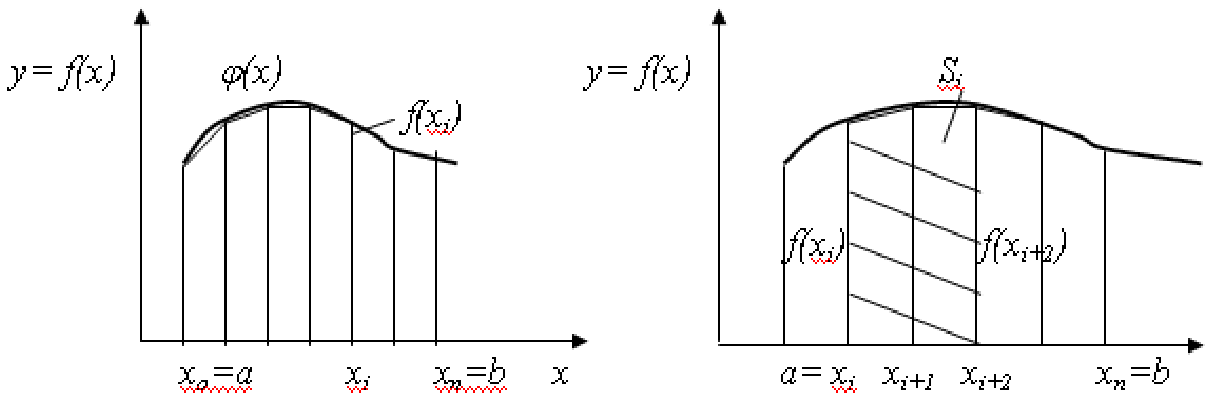
$$I_l = \int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i + \frac{b-a}{2n}) . \qquad (2.3)$$

Похибка обчислень інтегралу методом прямокутників обчислюється за формулою:

$$R(f) = \frac{f''(\xi)}{24}(b-a)h^2 . \qquad (2.4)$$

Метод трапецій

Метод трапецій полягає в тому, що відрізок інтегрування [*a, b*] розбивається на *n* рівних відрізків, всередині яких підінтегральна крива *f(x)* замінюється кусково-лінійною функцією *П(x),* отриманою стягуванням ординат *n* відрізків [x_{i-1}, x_i] хордами. Інтеграл знаходиться як сума площ s_i прямокутних трапецій (рис.3.1 *а,б*).



а) *б)*

Рисунок 3.1 Геометрична інтерпретація методу трапецій

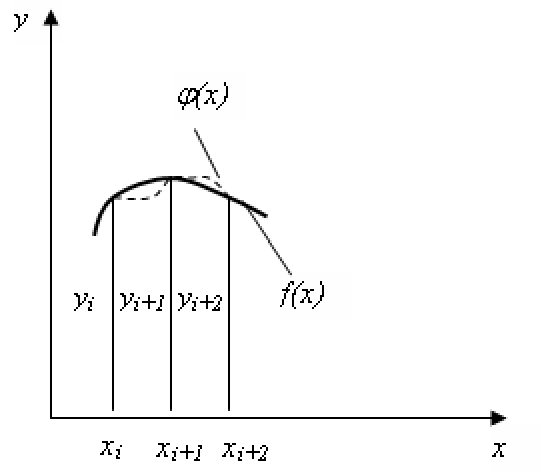
Площа кожної такої трапеції визначається як

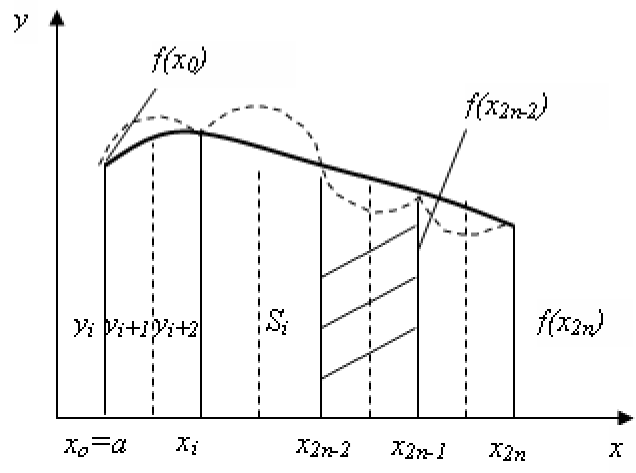
$$S_i = h\frac{f(x_i) + f(x_{i+1})}{2} . \qquad (3.1)$$

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n-1} S_i = \frac{b-a}{n}(\frac{y_0}{2} + y_1 + y_2 + \ldots + y_{n-1} + \frac{y_n}{2}) = \frac{b-a}{n}\left[\sum_{i=0}^{n-1} y_i + \frac{y_0 + y_n}{2}\right]$$

Метод Сімпсона

Поділимо відрізок інтегрування [*a*, *b*] на парне число *n* рівних частин з кроком $h = \frac{b-a}{2n}$. На кожному відрізку [x_0, x_2], [x_2, x_4], …, [x_{i-1}, x_{i+1}], …, [x_{n-2}, x_n] підінтегральну функцію *f*(*x*) замінимо інтерполяційним многочленом другого степеня (квадратичною параболою, рис. 4.1, *а,б*) та обчислення визначеного інтеграла зводиться до обчислення суми площ *n* криволінійних трапецій S_i.





а) *б)*

Рисунки 4.1 Геометрична інтерпретація методу Сімпсона

Площа кожної такої криволінійної трапеції визначається за формулою Сімпсона:

$$S_i = \frac{h}{3}[f(x_i) + 4f(x_{i+1}) + f(x_{i+2})] \quad . \tag{4.1}$$

Отже, сума всіх криволінійних трапецій обчислюється за формулою

$$\sum_{i=1}^{n} S_i = \frac{h}{3}[y_0 + y_{2n} + 4(y_1 + ... + y_{2n-1}) + 2(y_2 + ... + y_{2n-2})]$$

або

$$\int_a^b f(x)dx \approx \frac{h}{3}[y_0 + y_{2n} + 4\sum_{i=1}^{n} y_{2i-1} + 2\sum_{i=1}^{n-1} y_{2i}] \tag{4.3}$$

Індивідуальне завдання

Скласти програму чисельного інтегрування відповідно до варіанта:

1) за методом лівих, правих та середніх прямокутників;

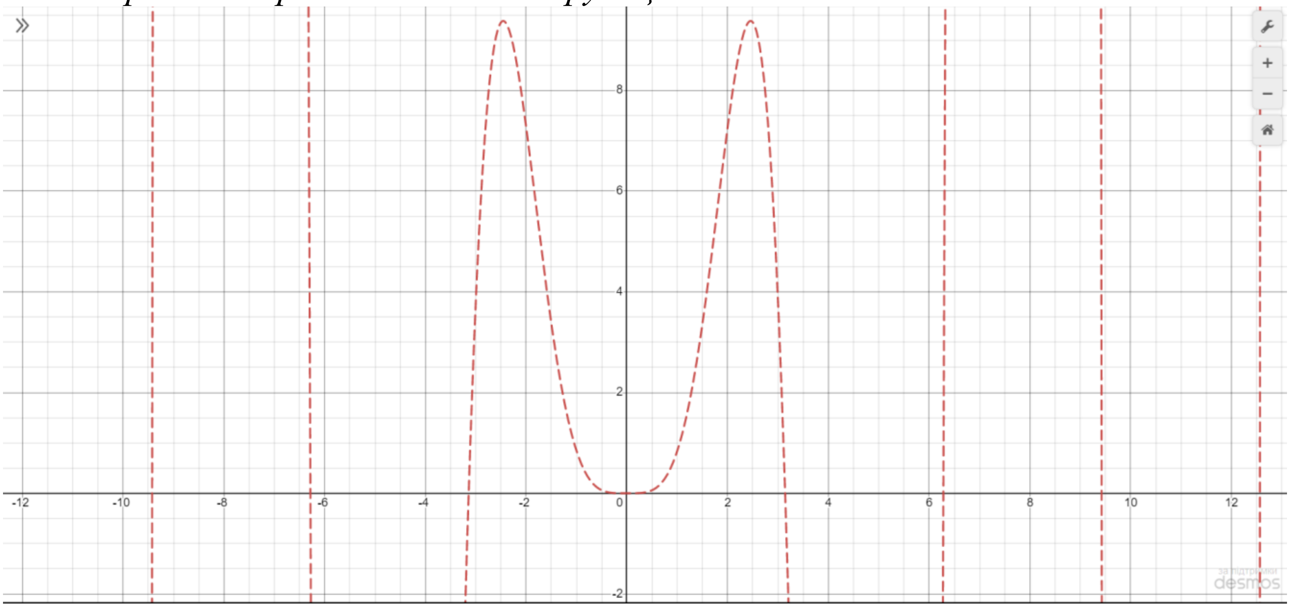
2) за методом трапецій;

3) за методом Сімпсона.

Інтеграл:

$$\int_{\frac{\pi}{3}}^{\frac{\pi}{2}} x^3 sinx$$

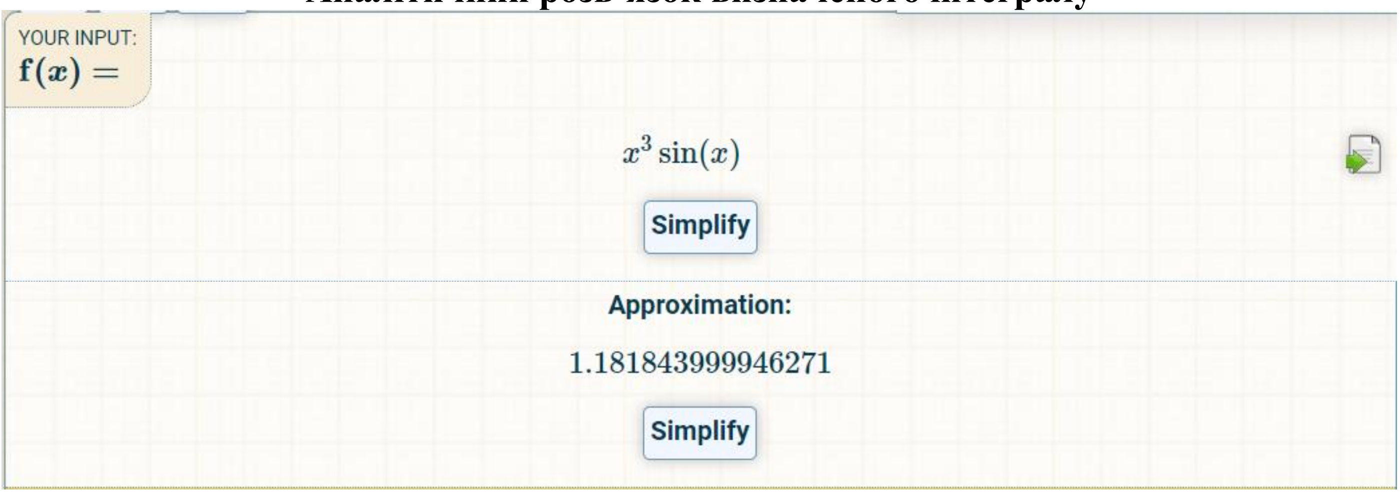
Геометричне зображення заданої функції:



Результат виконання програми

```
Left sum: 1.1894
Right sum: 1.17431
Middle sum: 1.18184

Approximational error: 4.62746e-06


Trapezoidal sum: 1.18185

Approximational error: 9.25492e-06


Simpson sum: 1.17336

Approximational error: 1.15162e-10
```

Аналітичний розв'язок визначеного інтегралу



Висновки

У результаті виконання лабораторної роботи, розроблено обчислюваний алгоритм, реалізований мовою програмування С чисельного інтегрування, а саме за методом лівих, правих та середніх прямокутників, за методом трапецій та за методом Сімпсона. Визначено похибки обчислення інтеграла за методом прямокутників – $4{,}62746 * 10^{-6}$, за методом трапецій – $9{,}25492 * 10^{-6}$, а за методом Сімпсона – $1{,}15162 * 10^{-10}$. Програмний продукт розроблений у середовищі Microsoft Visual Studio мовою програмування С++.

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";
}
```