

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» ім. Ігоря Сікорського

Лабораторна робота №4
з дисципліни «Алгоритми та методи обчислень»

«ЧИСЕЛЬНЕ ІНТЕГРУВАННЯ»

Виконав студент групи: КВ-33
Козлов Сергій

Київ 2025

Мета роботи

Дослідити методи чисельного інтегрування за допомогою узагальнених квадратурних формул.

Загальне завдання

1. Відповідно до варіанту (таб. 4.2) за допомогою залишкового члена АНАЛІТИЧНО визначити крок інтегрування h , що забезпечує необхідну точність ε (визначається у тексті програми).

Обчислити інтеграл I_h з кроком h і визначити абсолютну похибку Δ , прийнявши за точне значення, обчислене за формулою Ньютона-Лейбніца. Результати п.1 подати у вигляді:

Задана похибка, ε	Крок інтегрування	Точне значення інтеграла	Отримана похибка, Δ

2. За допомогою методу подвійного перерахунку досягти тієї ж похибки Δ , що й у п.1. Вивести значення отриманого кроku й абсолютної похибки.

Результати п.2 подати у вигляді:

Задана похибка, Δ	Крок інтегрування	Отримана похибка

Звіт про лабораторну роботу має містити вихідний текст програми, таблицю з результатами та висновки.

Завдання за варіантом

6	$\int_1^{15} 10 \ln(x) dx$	$10x \ln(x - 1)$
---	----------------------------	------------------

Примітка. У варіантах з парним номером необхідно використовувати узагальнену формулу трапецій, у варіантах з непарним номером - узагальнену формулу Сімпсона.

Підготовчий етап

Для обчислення точного значення заданого інтегралу за формулою Ньютона-Лейбніца необхідно знайти первісну $F(x)$. А для аналітичного обчислення значення h і кількості кроків n , що забезпечують необхідну точність для методу трапецій, необхідно знайти другу похідну $f''(x)$.
Знайдемо їх:

$$f(x) = 10 \ln(x)$$

$$F(x) = 10(x \ln(x) - x)$$

$$f'(x) = \frac{10}{x}$$

$$f''(x) = -\frac{10}{x^2}$$

Зазначимо, що первісна надана в методичному посібнику не відповідає заданому інтегралу. Тому тут і далі в розрахунках використовуватиметься функція $F(x)$, обчислена вище.

Блок-схеми алгоритмів

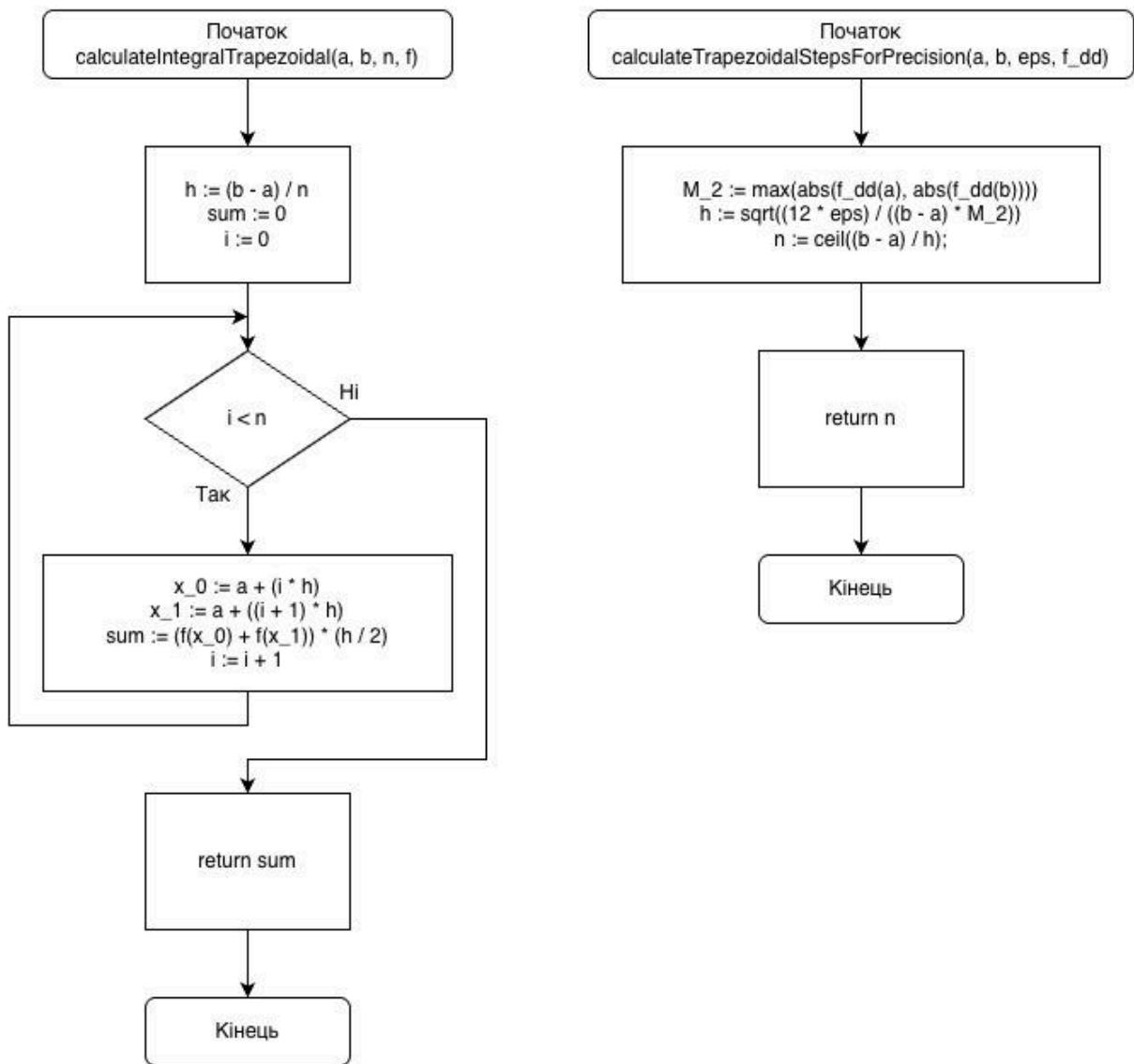


Рис. 1 - блок-схема алгоритму з аналітичним обчисленням кроку

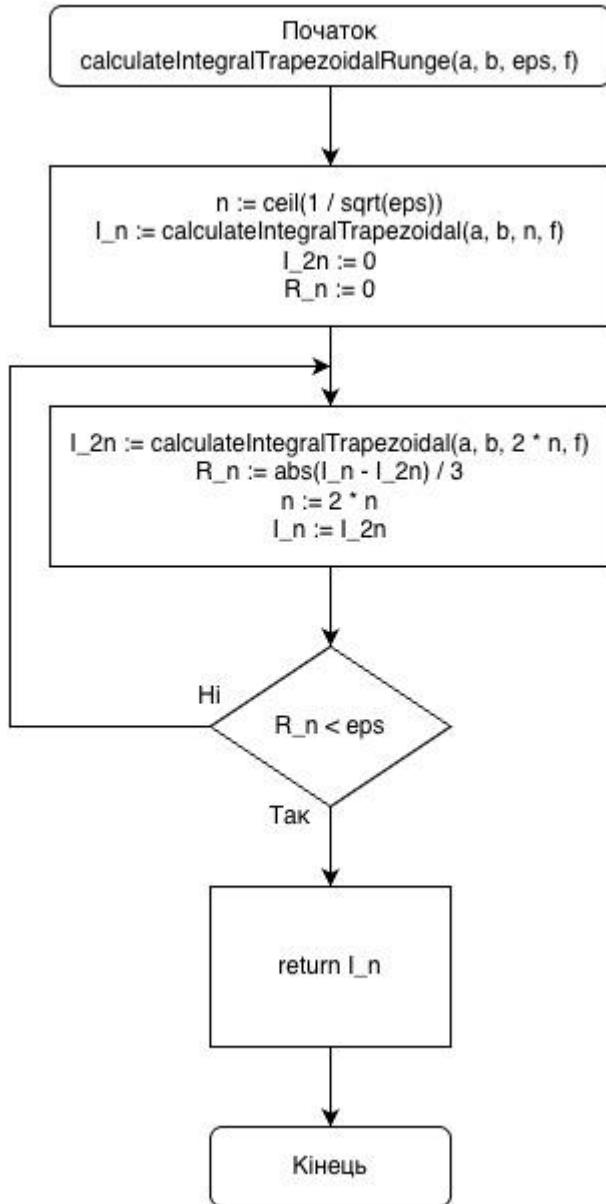


Рис. 2 - блок-схема алгоритму з застосуванням принципу Рунге

Результати

Табл. 1 - Результати алгоритму з аналітичним обчисленням кроку

Задана похивка, ϵ	Крок інтегрування	Точне значення інтеграла	Отримана похибка, Δ
10^{-2}	0.029227557411273	266.207530165331491	0.000664396494301
10^{-3}	0.009253139458030	266.207530165331491	0.000066593588485
10^{-4}	0.002927645336679	266.207530165331491	0.00006666414095
10^{-5}	0.000925803465150	266.207530165331491	0.00000666643018
10^{-6}	0.000292764533668	266.207530165331491	0.00000066665052
10^{-7}	0.000092581570977	266.207530165331491	0.00000006669268
10^{-8}	0.000029276943153	266.207530165331491	0.00000000657167

Табл. 2 - Результати алгоритму з застосуванням принципу Рунге

Задана похивка, ϵ	Крок інтегрування	Отримана похибка, Δ
10^{-2}	0.087500000000000	0.005953236852861
10^{-3}	0.027343750000000	0.000581513884924
10^{-4}	0.008750000000000	0.000059548448235
10^{-5}	0.002760252365931	0.000005925882306
10^{-6}	0.000875000000000	0.000000595485517
10^{-7}	0.000276636104964	0.000000059521085
10^{-8}	0.000087500000000	0.000000005955712

Висновок

Під час виконання лабораторної роботи були розглянуто сімейство алгоритмів чисельного інтегрування за допомогою квадратурних формул. Дослідження було сфокусоване на методі трапецій, але добуті висновки стосуються і інших форм методів Ньютона-Котеса.

Першочерговим етапом для обчислення інтегралу за квадратурою є вибір кроку h для досягнення бажаної точності.

Якщо підінтегральна функція задана алгебраїчно то h можна знайти аналітично, вивівши верхню границю h з формули залишкового члена. Однак, з формули Ньютона-Котеса випливає, що квадратурний метод N -го порядку вимагає обчислення локального максимуму похідної $(N+1)$ -го для отримання значення h . Отже, такий спосіб слід використовувати якщо підінтегральна функція відносно проста, або значення відповідної похідної відомі з інших джерел.

Альтернативно, можна скористатись принципом Рунге, і поступово подвоювати кількість кроків, поки бажана точність не буде досягнута. В такий спосіб можна розрахувати наближене значення для функції заданої таблицею значень в окремих точках. Це вимагає додаткових обчислень підінтегральної функції порівняно з пошуком необхідного h аналітично.

Для вивчення і перевірки роботи алгоритму трапецій в рамках виконання роботи була створена програма мовою С. Вихідний код і результати виконання представлені у звіті.

Вихідний код програми

```
// main.c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "integral/integral.h"
#include "log/log.h"

const double A = 1.0;
const double B = 15.0;

const double MIN_EPS_POW = -2.0;
const double MAX_EPS_POW = -8.0;
const double EPS_POW_STEP = 1.0;

double F(double x) {
    return 10.0 * (x * log(x) - x);
}

double f(double x) {
    return 10.0 * log(x);
}

double f_d(double x) {
    return 10.0 / x;
}

double f_dd(double x) {
    return -10.0 / (x * x);
}

void calculateIntegralAnalytical() {
    FILE *file = fopen("../out/1.csv", "w");

    if (file == NULL) {
        fprintf(stderr, "Error: Could not open ../out/1.csv for writing\n");
        return;
    }

    double I_precise = calculateIntegralNewtonLeibniz(A, B, F);

    writeTableRowToConsole(4, "Epsilon", "Step", "Exact Value", "Obtained Error");
    writeTableRowToFile(file, 4, "Epsilon", "Step", "Exact Value", "Obtained Error");

    for (int eps_pow = MIN_EPS_POW; eps_pow >= MAX_EPS_POW; eps_pow -=
EPS_POW_STEP) {
        double eps = pow(10, eps_pow);

        int n = calculateTrapezoidalStepsForPrecision(A, B, eps, f_dd);

        IntegralResult result = calculateIntegralTrapezoidal(A, B, n, f);
        double I = result.I;
        double h = result.h;

        char *eps_str = exponentToString(10, eps_pow);
        char *h_str = doubleToString(h, "%.15f");
        char *I_precise_str = doubleToString(I_precise, "%.15f");
        char *error_str = doubleToString(fabs(I_precise - I), "%.15f");

        writeTableRowToConsole(4, eps_str, h_str, I_precise_str, error_str);
    }
}
```

```

writeTableRowToFile(file, 4, eps_str, h_str, I_precise_str, error_str);

free(eps_str);
free(h_str);
free(I_precise_str);
free(error_str);
}

fclose(file);
}

void calculateIntegralRunge() {
FILE *file = fopen("../out/2.csv", "w");

if (file == NULL) {
    fprintf(stderr, "Error: Could not open ../out/2.csv for writing\n");
    return;
}

double I_precise = calculateIntegralNewtonLeibniz(A, B, F);

writeTableRowToConsole(3, "Epsilon", "Step", "Obtained Error");
writeTableRowToFile(file, 3, "Epsilon", "Step", "Obtained Error");

for (int eps_pow = MIN_EPS_POW; eps_pow >= MAX_EPS_POW; eps_pow -=
EPS_POW_STEP) {
    double eps = pow(10, eps_pow);

    IntegralResult result = calculateIntegralTrapezoidalRunge(A, B, eps, f);
    double I = result.I;
    double h = result.h;

    char *eps_str = exponentToString(10, eps_pow);
    char *h_str = doubleToString(h, "%.15f");
    char *error_str = doubleToString(fabs(I_precise - I), "%.15f");

    writeTableRowToConsole(3, eps_str, h_str, error_str);
    writeTableRowToFile(file, 3, eps_str, h_str, error_str);

    free(eps_str);
    free(h_str);
    free(error_str);
}

fclose(file);
}

int main() {
printf("\n");

calculateIntegralAnalytical();

printf("\n");

calculateIntegralRunge();

printf("\n");

return 0;
}

```

```
// integral.h
#ifndef INTEGRAL_H
#define INTEGRAL_H

typedef struct {
    double I;
    double h;
} IntegralResult;

double calculateIntegralNewtonLeibniz(double a, double b, double (*F)(double x));

int calculateTrapezoidalStepsForPrecision(double a, double b, double eps,
double (*f_dd)(double x));

IntegralResult calculateIntegralTrapezoidal(double a, double b, int n, double
(*f)(double x));

IntegralResult calculateIntegralTrapezoidalRunge(double a, double b, double
eps, double (*f)(double x));

#endif
```

```

// integral.c
#include "integral.h"

#include <math.h>
#include <stdio.h>

double calculateIntegralNewtonLeibniz(double a, double b, double (*F)(double x)) {
    return F(b) - F(a);
}

int calculateTrapezoidalStepsForPrecision(double a, double b, double eps,
double (*f_dd)(double x)) {
    double M_2 = fmax(fabs(f_dd(a)), fabs(f_dd(b)));
    double h = sqrt((12.0 * eps) / ((b - a) * M_2));
    int n = ceil((b - a) / h);
    return n;
}

IntegralResult calculateIntegralTrapezoidal(double a, double b, int n, double (*f)(double x)) {
    double h = (b - a) / n;
    double sum = 0.0;

    for (int i = 0; i < n; i++) {
        double x_0 = a + (i * h);
        double x_1 = a + ((i + 1) * h);

        sum += (f(x_0) + f(x_1)) * h / 2.0;
    }

    IntegralResult result;
    result.I = sum;
    result.h = h;

    return result;
}

IntegralResult calculateIntegralTrapezoidalRunge(double a, double b, double eps,
double (*f)(double x)) {
    int n = ceil(1.0 / sqrt(eps));

    double I_n = calculateIntegralTrapezoidal(a, b, n, f).I;
    double I_2n = 0.0;
    double R_n = 0.0;

    do {
        I_2n = calculateIntegralTrapezoidal(a, b, 2 * n, f).I;
        R_n = fabs(I_n - I_2n) / 3.0;

        n *= 2;
        I_n = I_2n;
    } while (R_n > eps);

    IntegralResult result;
    result.I = I_n;
    result.h = (b - a) / n;

    return result;
}

```

```
// log.h
#ifndef LOG_H
#define LOG_H

#include <stdio.h>

extern const int MAX_STRING_SIZE;

char* exponentToString(int base, int exponent);
char* doubleToString(double value, char* format);
char* intToString(int value);
void writeTableRowToConsole(int columns, ...);
void writeTableRowToFile(FILE* file, int columns, ...);

#endif
```

```

// log.c
#include "log.h"

#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

const int MAX_STRING_SIZE = 256;

char *exponentToString(int base, int exponent) {
    char *string = (char *) malloc(MAX_STRING_SIZE * sizeof(char));

    sprintf(string, MAX_STRING_SIZE, "%d^%d", base, exponent);

    return string;
}

char *doubleToString(double value, char *format) {
    char *string = (char *) malloc(MAX_STRING_SIZE * sizeof(char));

    sprintf(string, MAX_STRING_SIZE, format, value);

    return string;
}

char *intToString(int value) {
    char *string = (char *) malloc(MAX_STRING_SIZE * sizeof(char));

    sprintf(string, MAX_STRING_SIZE, "%d", value);

    return string;
}

void writeTableRowToConsole(int columns, ...) {
    va_list args;
    va_start(args, columns);

    for (int i = 0; i < columns; i++) {
        const char *column_value = va_arg(args, const char *);

        printf("%-20s", column_value);
    }

    printf("\n");
    va_end(args);
}

void writeTableRowToFile(FILE *file, int columns, ...) {
    va_list args;
    va_start(args, columns);

    for (int i = 0; i < columns; i++) {
        const char *column_value = va_arg(args, const char *);

        fprintf(file, "%-20s", column_value);

        if (i < columns - 1) {
            fprintf(file, ",");
        }
    }

    fprintf(file, "\n");
}

```

```
    va_end(args);  
}
```