НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ ТА СПЕЦІАЛІЗОВАНИХ КОМП’ЮТЕРНИХ СИСТЕМ

**Лабораторна робота №2  
з дисципліни «Алгоритми та методи обчислення»  
на тему: «Розв’язання рівнянь з одним невідомим»**

**Варіант 21**

Виконав:  
студент 3-го курсу,  
гр. КВ-41,  
Яковенко Максим

Київ – 2016

**Завдання для лабораторної роботи**

Для заданого рівняння відповідно до варіанту (табл. 2.4) виконати відокремлення коренів та обчислити значення констант *m1* або *q* (залежно від методу уточнення коренів, що використовується). Розв‘язати задане рівняння з точністю *εi = ε (i – 1) / 10-3, і = 1, 2, ..., 4; ε0 = 0.01*. Результати подати у вигляді трьох таблиць

Для першого кореня рівняння побудувати порівняльну таблицю швидкості збігання методу ітерації та іншого методу, заданого за варіантом.

Звіт з лабораторної роботи має містити вихідний текст програми, три таблиці з результатами та висновки.

**Варіант 21:**



**Текст програми**:

**Result.h**

#pragma once

struct Result

{

double x;

double clarRoot;

int count;

Result(void);

Result(Result& source);

Result& operator=(Result& source);

~Result(void);

};

**methods.h**

#pragma once

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include "Result.h"

class Method

{

private:

public:

typedef double (\*type\_func)(double x);

Method(void);

~Method(void);

Result Iteration(double m, double M, double x, double eps, double(\*fx)(double x));

Result Bisection(double a, double b, double eps, double(\*fx)(double x));

};

**methods.cpp**

#include "methods.h"

Method::Method(void) {};

Method::~Method(void) {};

Result Method::Iteration(double m, double M, double x, double eps, double(\*fx)(double x))

{  
 double x\_n, lambda, q;

int count = 0;

Result resInf;

lambda = 1/M;

q = 1- m/M;

do {

x\_n = x;

x = x\_n - lambda\*fx(x\_n);

++count;

} while(abs(x\_n - x) > (1 - q)\*eps/q);

resInf.x = x;

resInf.clarRoot = abs(x - x\_n);

resInf.count = count;

return resInf;

}

Result Method::Bisection(double a, double b, double eps, double(\*fx)(double x))

{

double x, c;

int count = 0;

Result resInf;

while(abs(b - a) >= 2\*eps) {

c = (a + b)/2;

if((fx(a)\*fx(c)) < 0)

b = c;

if((fx(b)\*fx(c)) < 0)

a = c;

++count;

}

x = (a + b)/2;

resInf.x = x;

resInf.clarRoot = (b - a) \* 0.5;

resInf.count = count;

return resInf;

}

**Result.cpp**

#include "Result.h"

Result::Result(Result& source)

{

x = source.x;

clarRoot = source.clarRoot;

count = source.count;

}

Result& Result::operator=(Result& source)

{

if (this == &source)

return source;

x = source.x;

clarRoot = source.clarRoot;

count = source.count;

return source;

}

Result::~Result(void) {}

Result::Result(void) {}

**main.cpp**

#include <iostream>

#include <stdio.h>

#include "methods.h"

#include "math.h"

#define N 4

using namespace std;

double fx(double x)

{ return 3/(2 + cos(x)) - (x/4);

}

double minus\_fx(double x)

{ return -3/(2+cos(x))+(x/4);

}

int main()

{ double x;

double eps;

double a, b;

int i;

Method \*method = new Method();

Result result, result2;

cout<<"\t\tIteration Method"<<endl;

cout<<endl;

x = 5;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Iteration(0.7, 1.3, x, eps, minus\_fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

x = 8;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Iteration(0.2, 1.1, x, eps, fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

x = 9.5;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Iteration(0.05, 1.5, x, eps, minus\_fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

cout<<"\t\tBisection Method"<<endl;

cout<<endl;

a = 4;

b = 6;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Bisection(a, b, eps, fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

a = 7;

b = 9;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Bisection(a, b, eps, minus\_fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

a = 9;

b = 11;

cout<<"\tEps\t"<<"\t"<<" Root's value\t"<<"\tEstimation accuracy"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Bisection(a, b, eps, minus\_fx);

printf("%.14lf \t %.15lf \t %.15lf\n", eps, result.x, result.clarRoot);

}

cout<<endl;

cout<<"\tComparison Table "<<endl;

cout<<endl;

x = 5;

a = 4;

b = 6;

cout<<"\tEps\t"<<"\t"<<"Iteration method\t"<<"Bisection method"<<endl;

for(i = 0, eps = 1e-2; i <= N; ++i, eps /= 1e3) {

result = method->Iteration(0.7, 1.3, x, eps, minus\_fx);

result2 = method->Bisection(a, b, eps, fx);

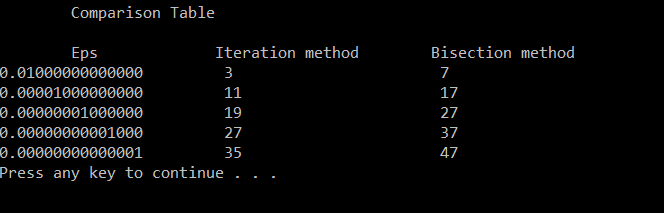
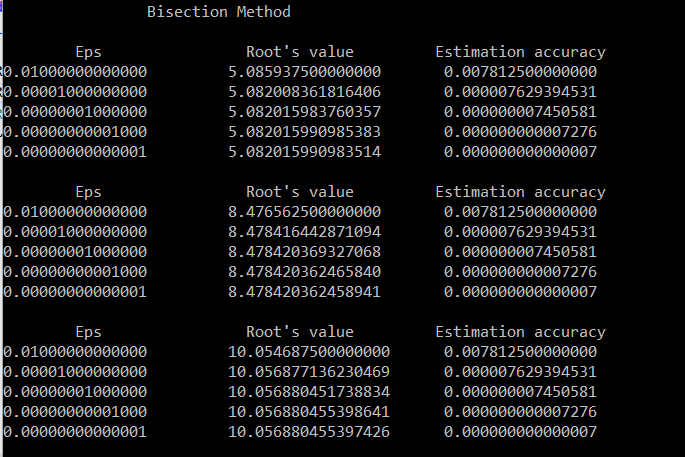
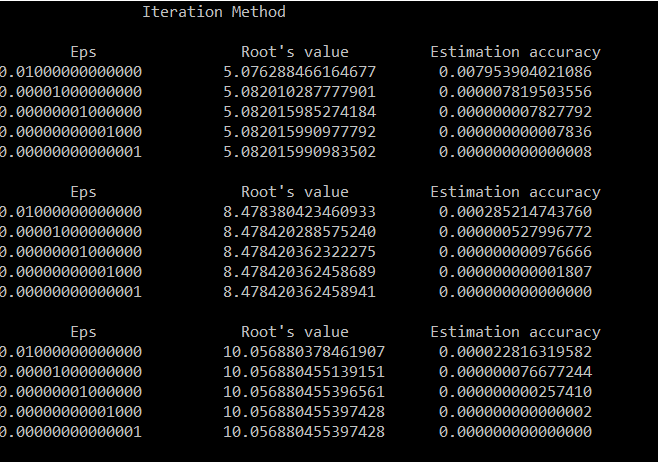
printf("%.14lf \t %d \t\t\t %d\n", eps, result.count, result2.count);

}

return 0;

}

**Таблиці результатів:**



**Висновки:**

В ході виконання лабораторної роботи ми досліджували методи розв’язання рівнянь з одним невідомим.

Для знаходження коренів рівняння необхідно спочатку відокремити, тобто визначити певний відрізок, що містить в собі лише один корінь рівняння. Потім відокремленні корені уточняються відповідними способами.

В даному варіанті методами уточнення кореня були: ітераційний метод та метод пропорційного поділу відрізку(метод бісекції).

Обидва методи уточнення коренів є досить точними, тому ми порівнювали їх за кількістю ітерацій необхідних для знаходження кореня заданої точності. Метод ітерації виявився швидшим від методу бісекції, що видно з результатів поданих у таблиці 3. Проте метод ітерації можна використовувати лише для рівнянь, придатних до ітерації, і, часто, необхідно знаходити чи певним чином зводити рівняння до придатного до ітерації, що ускладнює процес уточнення кореня.