МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Вычислительная математика»
Тема: Метод бисекции

Студент гр. 3342 _____ Иванов Д. М.

Преподаватель _____ Лисс А. Р.

Санкт-Петербург
2025

Задание

В лабораторной работе №3 предлагается найти корень уравнения f(x)=0 методом бисекции с заданной точностью Eps, исследовать зависимость числа итераций от точности Eps при изменении Eps от 0.1 до 0.000001, исследовать обусловленность метода (чувствительность к ошибкам в исходных данных).

Выполнение работы осуществляется по индивидуальным вариантам заданий (нелинейных уравнений), приведенным в подразделе 3.6. Номер варианта для каждого студента определяется преподавателем.

Порядок выполнения работы должен быть следующим:

1) Графически или аналитически отделить корень уравнения f(x)=0 (т.е. найти отрезки [Left, Right], на которых функция f(x) удовлетворяет условиям теоремы Коши).

2) Составить подпрограмму вычисления функции f(x).

3) Составить головную программу, содержащую обращение к подпрограмме f(x), BISECT, Round и индикацию результатов.

4) Провести вычисления по программе. Построить график зависимости числа итераций от Eps.

5) Исследовать чувствительность метода к ошибкам в исходных данных. Ошибки в исходных данных моделировать с использованием программы Round, округляющей значения функции с заданной точностью Delta.

Функция для индивидуального варианта:

$f(x) = x^4 – 13*x^2 + 36 – 1/x$

Теоретическая часть

Если найден отрезок [a,b], такой, что f(a)f(b)<0, существует точка c, в которой значение функции равно нулю, т.е. f(c)=0, c ∈ (a,b). Метод бисекции состоит в построении последовательности вложенных друг в друга отрезков, на концах которых функция имеет разные знаки. Каждый последующий отрезок получается делением пополам предыдущего. Процесс построения последовательности отрезков позволяет найти нуль функции f(x) (корень уравнения f(x)=0 с любой заданной точностью.

Рассмотрим один шаг итерационного процесса. Пусть на (n-1)-м шаге найден отрезок $[a_{n-1}, b_{n-1}] \subset [a, b]$, такой, что $f(a_{n-1})f(b_{n-1})<0$. Разделим его пополам точкой $e=(a_{n-1} + b_{n-1})/2$ и вычислим f (e). Если f(e)=0, то $e=(a_{n-1}+b_{n-1})/2$-корень уравнения. Если f (e)!=0, то из двух половин отрезка выбирается та, на концах которой функция имеет противоположные знаки, поскольку искомый корень лежит на этой половине, т.е.

$a_n=a_{n-1}$, $b_n=e$, если $f(e)f(a_{n-1}) < 0$;

an=e, $b_n= b_{n-1}$, если $f(e)f(a_{n-1}) > 0$.

Если требуется найти корень с точностью eps, то деление пополам продолжается до тех пор, пока длина отрезка не станет меньше 2*eps. Тогда координата середины отрезка есть значение корня с требуемой точностью eps.

Метод бисекции является простым и надежным методом поиска простого корня уравнения f(x)=0 (простым называется корень x=c дифференцируемой функции f(x), если f (c) и f'(c)!=0). Этот метод сходится для любых непрерывных функций f(x), в том числе недифференцируемых. Скорость его сходимости невысока. Для достижения точности eps необходимо совершить N=log2(b-a)/eps итераций. Это означает, что для получения каждых трех верных десятичных знаков необходимо совершить около 10 итераций.

Аналитический анализ функции

Для выполнения теоремы Коши нужны найти такие промежутки [a, b], в которых содержится корень уравнения f(x)=0. Преобразуем выражение.

$f(x) = (x^5 – 13*x^3 + 36*x - 1) / x$. Можем заметить, что корни числителя $g(x) = x^5 – 13*x^3 + 36*x – 1$ буду совпадать с корнями f(x) за исключением x=0, так как в этой точке функция прерывается из-за деления на 0. Также уравнение может иметь максимум 5 корней. Рассмотрим пределы функции при различных x.

$$\lim_{x \to +inf} f(x) = +inf$$

$$\lim_{x \to -inf} f(x) = +inf$$

$$\lim_{x \to 0+} f(x) = -inf$$

$$\lim_{x \to 0-} f(x) = +inf$$

Будем подставлять в f(x) различные значения и следить за тем, как изменяется знак f(x).

f(-4) = 84,25 > 0

f(-3) = 0,33 > 0

f(-2,5) = -5,7875 < 0

f(-2) = 0,5 > 0

f(-1) = 25 > 0

f(1) = 23 > 0

f(2) = -0,5 < 0

f(3) = -0,33 < 0

f(4) = 83,75 > 0

Можем уже увидеть некоторые интервалы, где знаки меняются: [-4, -2,5], [-2,5, -2], [1, 2], [3, 4].

Также рассмотрим предел функции при x->0+ (см. выше) и f(1). Можем заметить, что начиная с x=1 и идя к x=0 значение функции идет бесконечно вниз и, таким образом, пересечет прямую Ox в промежутке [0, 1].

В итоге мы получили нужные нам промежутки, в которых содержатся корни уравнения f(x)=0.

Написание программы для реализации алгоритма бисекции

Реализуем алгоритм на языке C++. Для этого напишем некоторые функции.

double func(double x) – функция для вывола значения функции f(x) по переданному x.

double roundValue(double x, double delta) – функция для округления переданного числа на заданную точность. Сначала идет деление на эту точность, после этого округление до целого с помощью команды round и затем умножение на переданную точность.

double bisection(double a, double b, double eps, int iterationsCount) – сам алгоритм бисекции. Передаются границы отрезка [a, b], точность Eps и счетчик количества итераций (для анализа). Берется на каждой итерации середина отрезка e. Если значение функции в этой точке равно нулю или длина отрезка меньше, чем 2*eps, возвращается найденная середина, округленная до нужной точности. В противном случае продолжается деление дальше через рекурсию. И выбирается та половина отрезка, на концах которой значения функции имеет противоположные знаки. То есть идет вызов либо bisection(a, e, eps, ++iterationsCount), либо bisection(e, b, eps, ++iterationsCount).

Тестирование программы приведено в таблице 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	bisection(-4, -2.5, 0.001, iterations)	-2.989	f(-2.988)= -0.01944 f(-2.990)= 0.03853
2.	bisection(-2.5, -2, 0.1, iterations)	-2.1	f(-2.0)= 0.50000 f(-2.2)= -3.03985
3.	bisection(0, 1, 0.01, iterations)	0.02	f(0.01)= -64.00129 f(0.03)= 2.65496
4.	bisection(3, 4, 0.000001, iterations)	3.010907	f(3.010906)= -0.0005 f(3.010908)= 0.00001

Исследование зависимости числа итераций от Eps

Для исследование данной зависимости напишем программу, которая будет перебирать различные eps и высчитывать количество итераций (количество вызовов функции), которое потребовалось данной программе. Будем перебирать, к примеру, для отрезка [-4, -2,5]. Значения Eps будем брать от 0.1 до 0.000001. Получим следующие результаты: (см. рис. 1).

```
PS C:\Users\user\Desktop\ВычМат\lab1> g++ main.cpp
PS C:\Users\user\Desktop\ВычМат\lab1> ./a.exe
Iterations count for eps=0.1 is 3
Iterations count for eps=0.01 is 7
Iterations count for eps=0.001 is 10
Iterations count for eps=0.0001 is 13
Iterations count for eps=1e-005 is 17
Iterations count for eps=1e-006 is 20
PS C:\Users\user\Desktop\ВычМат\lab1>
```

Рисунок 1 – результаты зависимости количества итераций от Eps

Построим график для данных полученных значений. Также сравним его теоретической зависимостью N=log2((b - a) / eps). Получили следующую картину (см. рис. 2).

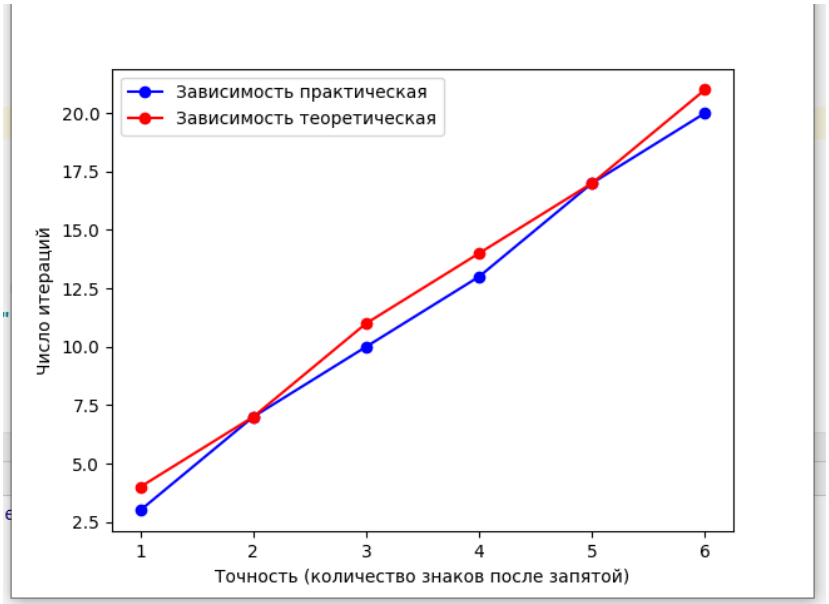

Рисунок 2 – графики практической и теоретической зависимостей

По данным графикам видно, что наша практическая зависиомсть почти совпала с теоретической, что подвтерждает изложенный выше теоретический анализ алгоритма.

Исследование чувствительности метода к ошибкам в исходных данных

Для выполнения данного исследование необходимо изменять точность выходного значения функции, округляя на некоторое количество знаков после запятой. Тем самым будут возникать ошибки в наших исходных данных, и мы проанализируем, насколько точно алгоритм будет работать.

Напишем дополнительные функции

double errorFunc(double x, double delta) – функция, вызывающая f(x), но с округлением delta через roundValue.

double errorCheckInBisection(double a, double b, double eps, int& iterationsCount, double delta) – функция, содержащая тот же самый алгоритм бисекции. Помимо всего, она содержит дополнительный аргумент delta, который показывается точность значения f(x) через errorFunc.

Мы возьмем некоторый промежуток (к примеру [-4, -2,5]) и будем перебирать для него delta и eps. Будем сравнивать наши результаты с обычным вызовом для этого промежутка с точность 10^-6. bisection(-4, -2.5, 0.000001, iterations) -> -2.988672

Результаты исследования см. в таблице 2.

Таблица 2 – Результаты перебора eps и delta для промежутка [-4, -2,5]

delta	eps	Значение корня
0.1	0.1	-3.0
0.1	0.01	-2.99
0.1	0.001	-2.989
0.1	0.0001	-2.9893
0.1	0.00001	-2.98926
0.1	0.000001	-2.989258
0.01	0.1	-3.0
0.01	0.01	-2.99
0.01	0.001	-2.989
0.01	0.0001	-2.9885
0.01	0.00001	-2.98853
0.01	0.000001	-2.988525
0.001	0.1	-3.0
0.001	0.01	-2.99
0.001	0.001	-2.989

0.001	0.0001	-2.9886
0.001	0.00001	-2.98866
0.001	0.000001	-2.988663
0.0001	0.1	-3.0
0.0001	0.01	-2.99
0.0001	0.001	-2.989
0.0001	0.0001	-2.9886
0.0001	0.00001	-2.98867
0.0001	0.000001	-2.988671
0.00001	0.1	-3.0
0.00001	0.01	-2.99
0.00001	0.001	-2.989
0.00001	0.0001	-2.9886
0.00001	0.00001	-2.98867
0.00001	0.000001	-2.988672
0.000001	0.1	-3.0
0.000001	0.01	-2.99
0.000001	0.001	-2.989
0.000001	0.0001	-2.9886
0.000001	0.00001	-2.98867
0.000001	0.000001	-2.988672

По данным результатам можем заметить следующее. Для больших округлений функции (delta от 0.1 до 0.0001) могут возникать неточности при вычислении корня: delta=0.1 – ошибка на 3-ем разряде после запятой, delta=0.01 – ошибка на 4-ом разряде после запятой, delta=0.001 – ошибка на 5-ом разряде после запятой, delta=0.0001 – ошибка на 6-ом разряде после запятой. Для менне существенных ошибко наш итоговый результат не поменяется.

В итоге можно сказать, что алгоритм хорошо устойчив к ошибкам в исходных данных. При небольших изменениях выходное значение будет таким же. А при серьезных округлениях ошибки возникнут только на 3-ем разряде после запятой, что является неплохим результатом.

Вывод

В ходе лабораторной работы была написана программа на языке C++, выполняющая поиск корней некоторой функции f(x). Данный алгоритм был протестирован на некоторых входных данных. Также провелись некоторые исследования. По их результатам мы получили зависимость количества итераций от точности корня и сравнили ее с теоретическими вычислениями.

Также было проведено исследование на чувтсвительность метода к ошибкам. Мы убедились, что алгоритм бисекции достаточно устойчив к небольшим ошибкам в исходных данных, но при значительном увеличении Delta точность результата может снизиться на небольшое значение.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: bisection.cpp

```
#include <iostream>
#include <cmath>

double func(double x){
    return pow(x, 4) - 13 * pow(x, 2) + 36 - 1 / x;
}

double roundValue(double x, double delta){
    return round(x / delta) * delta;
}

double bisection(double a, double b, double eps, int& iterationsCount){
    double e = (a + b) / 2;
    if (func(a) * func(b) < 0){
        if (func(e) == 0 || (b - a) < 2 * eps){
            return roundValue(e, eps);
        }
        else if (func(e) * func(a) < 0){
            return bisection(a, e, eps, ++iterationsCount);
        }
        else{
            return bisection(e, b, eps, ++iterationsCount);
        }

    }
    else{
        std::cout << "Неверно заданный интервал!" << std::endl;
        exit(1);
    }
}
```