**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. 3341 |  | Лодыгин И. А. |
| Преподаватель |  | Пуеров Г. Ю. |

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.

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^2 – x^3 – 1/(4 + x^2)

**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(с)=0, с ∈ (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 [an-1, bn-1] ⊂ [a, b], такой, что f(an-1)f(bn-1)<0. Разделим его пополам точкой e=(an-1 + bn-1)/2 и вычислим f (e). Если f(e)=0, то e=(an-1+bn-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.

an=an-1, bn=e, если f (e)f(an-1) < 0 ;

an=e, bn= bn-1, если f(e)f(an-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.

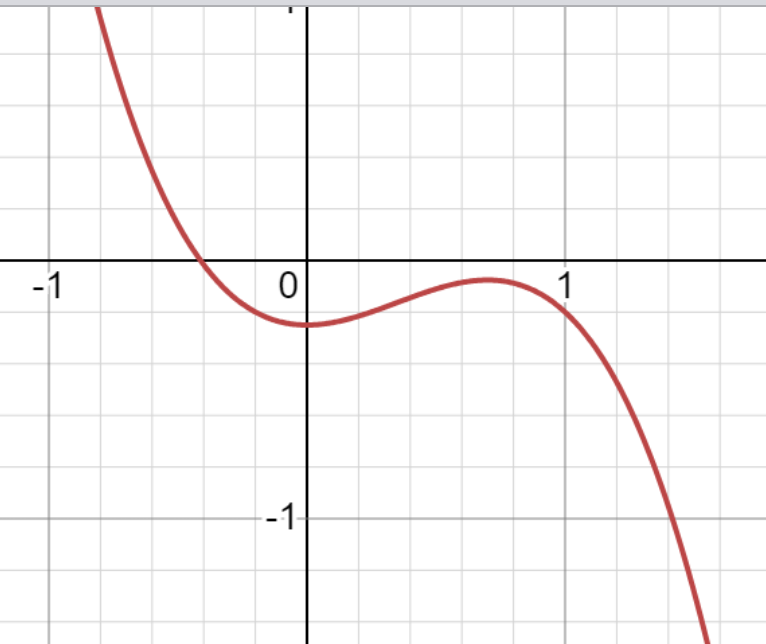
Метод бисекции является простым и надежным методом поиска простого корня уравнения f(x)=0 (простым называется корень x=c дифференцируемой функции f(x), если f (с) и f’(с)!=0). Этот метод сходится для любых непрерывных функций f(x), в том числе недифференцируемых. Скорость его сходимости невысока. Для 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 + x^4 – 4x^3 + 4x^2 - 1) / (4 + x^2). Можем заметить, что корни числителя g(x) = -x^5 + x^4 – 4x^3 + 4x^2 - 1 буду совпадать с корнями f(x) . Также уравнение может иметь максимум 5 корней. Рассмотрим пределы функции при различных x.

График функции построенный с помощью Mathway:



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

f(-5) = 150 > 0

f(-1) = 1.8 > 0

f(-0.5) = 0,1397 > 0

f(0) = -0.25 < 0

f(0.5) = -0.11 < 0

f(1) = -0.2 < 0

f(1.5) = -1.285 < 0

f(5) = -100 < 0

Можем уже увидеть некоторый интервал, где знаки меняются: [-0.5, 0].

В итоге мы получили нужный нам промежуток, в котором содержится корень 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 |
|  | bisection(-0.5, 0, 0.001, iterations) | -0.411 | f(-0.412) = 0.000144348  f(-0.410) = -0.00289647 |
|  | bisection(-0.5, 0, 0.1, iterations) | -0.4 | f(-0.3) = -0.127499  f(-0.5) = 0.139706 |
|  | bisection(-0.5, 0, 0.000001) | -0.412105 | f(-0.412106) = 2.02343e-06  f(-0.412104) = -7.38775e-07 |

**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 [-0.5, 0]. Значения Eps будем брать от 0.1 до 0.000001. Получим следующие результаты: (см. рис. 1).

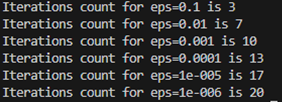


Рисунок 1 – результаты зависимости количества итераций от Eps

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

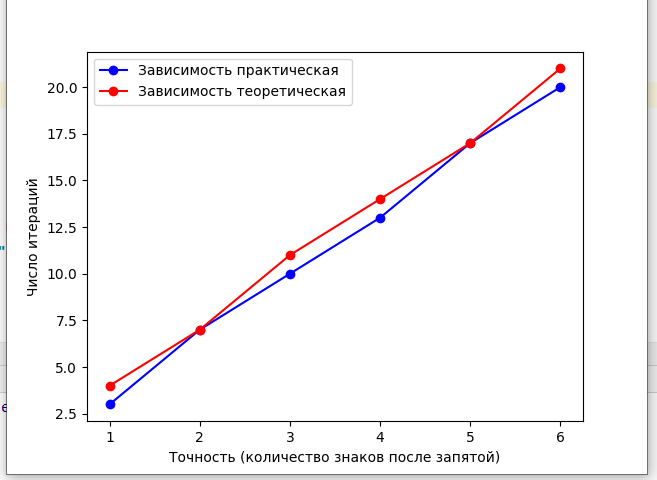


Рисунок 2 – графики практической и теоретической зависимостей

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

**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 промежуток [-0.5, 0] и будем перебирать для него delta и eps. Будем сравнивать наши результаты с обычным вызовом для этого промежутка с точностью 10^-6. bisection(-0.5, 0, 0.000001, iterations) -> -0.412105

Результаты исследования см. в таблице 2.

Таблица 2 – Результаты перебора eps и delta для промежутка [-0.5, 0]

|  |  |  |
| --- | --- | --- |
| delta | eps | Значение корня |
| 0.1 | 0.1 | -0.4 |
| 0.1 | 0.01 | -0.45 |
| 0.1 | 0.001 | -0.45 |
| 0.1 | 0.0001 | -0.45 |
| 0.1 | 0.00001 | -0.45 |
| 0.1 | 0.000001 | -0.45 |
| 0.01 | 0.1 | -0.4 |
| 0.01 | 0.01 | -0.41 |
| 0.01 | 0.001 | -0.415 |
| 0.01 | 0.0001 | -0.415 |
| 0.01 | 0.00001 | -0.415 |
| 0.01 | 0.000001 | -0.415 |
| 0.001 | 0.1 | -0.4 |
| 0.001 | 0.01 | -0.41 |
| 0.001 | 0.001 | -0.413 |
| 0.001 | 0.0001 | -0.4125 |
| 0.001 | 0.00001 | -0.4125 |
| 0.001 | 0.000001 | -0.412499 |
| 0.0001 | 0.1 | -0.4 |
| 0.0001 | 0.01 | -0.41 |
| 0.0001 | 0.001 | -0.413 |
| 0.0001 | 0.0001 | -0.4122 |
| 0.0001 | 0.00001 | -0.41215 |
| 0.0001 | 0.000001 | -0.41215 |
| 0.00001 | 0.1 | -0.4 |
| 0.00001 | 0.01 | -0.41 |
| 0.00001 | 0.001 | -0.411 |
| 0.00001 | 0.0001 | -0.412 |
| 0.00001 | 0.00001 | -0.4121 |
| 0.00001 | 0.000001 | -0.412105 |
| 0.000001 | 0.1 | -0.4 |
| 0.000001 | 0.01 | -0.41 |
| 0.000001 | 0.001 | -0.411 |
| 0.000001 | 0.0001 | -0.412 |
| 0.000001 | 0.00001 | -0.4121 |
| 0.000001 | 0.000001 | -0.412105 |

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 2-ем разряде после запятой, delta=0.01 – ошибка на 3-ом разряде после запятой, delta=0.001 – ошибка на 4-ом разряде после запятой, delta=0.0001 – ошибка на 5-ом разряде после запятой. Для менее существенных ошибок 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 2-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: main.cpp

#include <iostream>

#include <cmath>

double func(double x){

    return pow(x, 2) - pow(x, 3) - 1/(4 + pow(x, 2));

}

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);

    }

}

double errorFunc(double x, double delta) {

    return func(roundValue(x, delta));

}

double errorCheckInBisection(double a, double b, double eps, int& iterationsCount, double delta) {

    double e = (a + b) / 2;

    if (errorFunc(a, delta) \* errorFunc(b, delta) < 0) {

        if (errorFunc(e, delta) == 0 || (b - a) < 2 \* eps) {

            return roundValue(e, eps);

        }

        else if (errorFunc(e, delta) \* errorFunc(a, delta) < 0) {

            return errorCheckInBisection(a, e, eps, ++iterationsCount, delta);

        }

        else {

            return errorCheckInBisection(e, b, eps, ++iterationsCount, delta);

        }

    }

    else {

        std::cout << "Неверно заданный интервал!" << std::endl;

        exit(1);

    }

}