**зМИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Вычислительная математика»**

**Тема: Метод хорд**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Ягудин Д.Р. |
| Преподаватель |  | Пуеров Г.Ю. |

Санкт-Петербург

2025

## Цель работы

Изучить и реализовать метод хорд для численного решения уравнения f(x)=0, а также исследовать зависимость числа итераций от заданной точности Epsilon и оценить чувствительность метода к ошибкам в исходных данных.

## Задание

В лабораторной работе №4 предлагается, используя программы функции HORDA и Round из файла methods.cpp (файл заголовков metods.h, директория LIBR1), найти корень уравнения f(x) = 0 с заданной точностью Eps методом хорд, исследовать скорость сходимости и обусловленности метода.

Для данной работы, как и для лабораторной работы №3 задаются индивидуальные варианты нелинейных уравнений (см. подраздел 3.6).

Вариант 26:

f(x) = 1/(3+ 2cos(x)) - x3

**Порядок выполнения лабораторной работы №4:**

1. Графически или аналитически отделить корень уравненияf(x)=0 (т.е. найти отрезки [Left, Right], на которых функцияf(x)удовлетворяет условиям применимости метода).

2. Составить подпрограмму - функцию вычисления функцииf(x*),* предусмотрев округление значений функции с заданной точностьюDeltaс использованием программы Round.

3. Составить головную программу, вычисляющую корень уравненияf(x)=0 и содержащую обращение к подпрограмме f(x), HORDA, Round и индикацию результатов.

4. Провести вычисления по программе. Теоретически и экспериментально исследовать скорость сходимости и обусловленность метода.

В подразделе 3.7 приводится текст программы - функции HORDA,предназначенной для решения уравненияf(x)=0методом хорд.

## Выполнение работы

Найдём интервал [Left, Right]

Графическое отделение:

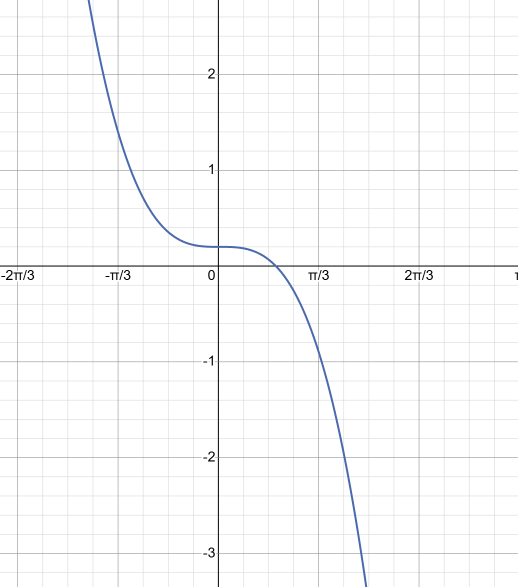


Рисунок № 1 – График функции 1/(3+ 2cos(x)) - x3

Из рисунка1 увидим, что функция положительна при f(0) и отрицательна при f(1), возьму интервал [0, 1].

Аналитическое отделение:

Рассмотрим уравнение f(x) = 1/(3+ 2cos(x)) - x3

Функция непрерывна (состоит из экспоненты и многочлена). По теореме Коши о промежуточных значения, если f(x) непрерывна на отрезке [a, b] и принимает значения разных знаков на концах отрезка, то внутри этого отрезка существует хотя бы одна точка c, в которой f(c) = 0.

Возьмём отрезок [0, 1] и вычислим значение в функции в точках:

f(0) = 1/(3+ 2cos(x)) - x3 = 0.2, следовательно f(0) > 0

f(1) = 1/(3+ 2cos(x)) - x3 = -0.75, следовательно f(1) < 0

Так как f(0) > 0 и f(1) < 0, выбираем отрезок [left, right] = [0, 1]

**Описание метода хорд**

**Исходные точки:**

**На графике функции f(x)f выбираются две точки: A и B, которые лежат на интервале [a,b], где функция меняет знак. Точка A имеет координаты (a,f(a)), а точка B — (b,f(b)).**

**Построение хорды**:

Межу точка A и B проводится прямая линия(хорда), пересекающая ось абцисс(ось X) в точке C, которая является новым приближением корня. Из точки C проводится перпендикуляр до пересечения с графиком функции f(x). Точка пересечения обозначается как D. После между точкой D и одной из предыдущих точек (A или B), выбирается точка, сохраняющая переменность знака и тогда интервал становиться [a, c] или [c,b] в нашем случае [c, b]. Аналогично из данной точки строится хорда и идёт поиск следующего приближения корня, пока расстояние между последовательными приближениями не станет меньше заднной точности epsilon.

**Формула для нахождения точки C:**

Точка C находится как пересечение хорды с осью x. Её координата вычисля-ется по формуле:

c= a− f(a)⋅(b−a) / f(b)−f(a)

**Условие остановки:**

Метод останавливается, когда ∣f(c)∣ < eps или ∣b−a∣ < eps.

**Реализация алгоритма на Python**

Функции использованные для решения задачи:

def f(x) – функция принимает один аргумент x, являющийся точкой, в которой нужно вычислить значение f(x)

def add\_error(value, delta) – Функция для добавления случайной ошибки.

def method\_chord(a, b, epsilon, delta) – Функция реализует метод хорд для нахождения корня функции.

def explore\_iterations\_chord(left, right, eps\_values) – Функция анализирует сходимость метода в зависимости от заданной точности epsilon

def main() – головная процедура, внутри задаётся интервал [left, right], eps\_values — список значений точности epsilon, с которыми будет производиться вычисление корня, вызывается функция plot\_iterations, которая содержит в себе исследование зависимости изменения epsilon от числа итераций, исследование чувствительности метода к ошибкам в исходных

## Исследование зависимости изменения epsilon от числа итераций

Функция explore\_iterations\_chord исследует, как количество итераций метода хорд зависит от значения eps (точности).

Параметры:

left, right — границы интервала для поиска корня,

eps\_values — список значений точности, для которых будет исследована зависимость,

Функция plot\_iterations строит график зависимости числа итераций от точности eps, для исследования были взяты данные из прошлой лабораторной работе по методу бисекции, а также построены графики для сравнения сходимости метода хорд и метода бисекции

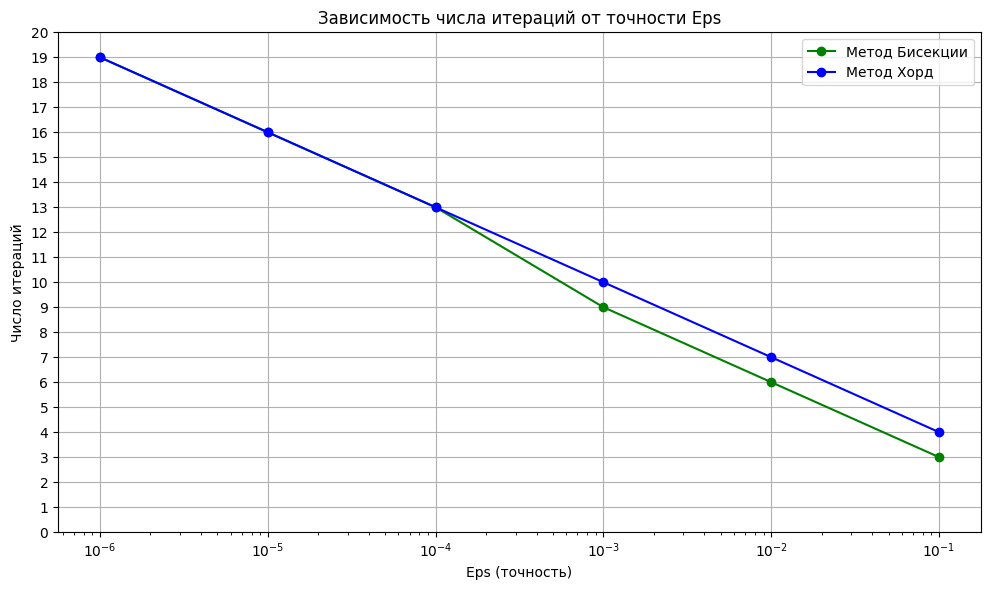


Рисунок № 3 – Сходимость метода хорд и бисекции

Преимущества метода хорд:

Сходится быстрее чем метод бисекции благодаря использованию информации о наклоне функции.

Не требует вычисления производной, в отличие от метода Ньютона.

Недостатки:

Метод может сходиться медленно на функциях со сложной формой.

**Исследование чувствительности метода к ошибкам в исходных данных**

Для данного исследования 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.

Функция explore\_error тестирует работу метода хорды с учетом погрешности.

Таблица 2 – Результаты перебора eps и delta для промежутка [0, 1]

|  |  |  |
| --- | --- | --- |
| delta | eps | Значение корня |
| 0.1 | 0.1 | 0.6 |
| 0.01 | 0.1 | 0.50 |
| 0.01 | 0.01 | 0.59 |
| 0.001 | 0.1 | 0.500 |
| 0.001 | 0.01 | 0.590 |
| 0.001 | 0.001 | 0.598 |
| 0.0001 | 0.1 | 0.5000 |
| 0.0001 | 0.01 | 0.5900 |
| 0.0001 | 0.001 | 0.5980 |
| 0.0001 | 0.0001 | 0.5990 |
| 0.00001 | 0.1 | 0.50000 |
| 0.00001 | 0.01 | 0.59000 |
| 0.00001 | 0.001 | 0.59800 |
| 0.00001 | 0.0001 | 0.59900 |
| 0.00001 | 0.00001 | 0.59904 |
| 0.000001 | 0.1 | 0.500000 |
| 0.000001 | 0.01 | 0.590000 |
| 0.000001 | 0.001 | 0.598000 |
| 0.000001 | 0.0001 | 0.599000 |
| 0.000001 | 0.00001 | 0.599040 |
| 0.000001 | 0.000001 | 0.599047 |

**Корень = 0.599047**

**Плохая обусловленность**:

Наблюдается при больших delta (например, delta = 0.1).

Ошибка в результате значительна из-за большого числа обусловленности.

**Хорошая обусловленность**:

Наблюдается при малых delta (например, delta =0.000001).

Ошибка в результате минимальна, так как число обусловленности мало.

**Умеренная обусловленность**:

Наблюдается при средних значениях delta (например, delta=0.01).

Ошибка в результате небольшая, но присутствует.

**Выводы**

В ходе выполнения задания, направленного на исследование метода хорд для нахождения корней уравнения f(x) = 1/(3+ 2cos(x)) - x3 = 0, можно подвести следующие итоги:

1. Реализованный метод хорд корректно находит корень функции на заданном интервале с необходимой точностью.

2. График зависимости числа итераций от точности подтвердил, что метод хорд быстрее сходится чем метод бисекции.

3. Метод хорд требует правильного выбора интервала и учёта ошибок вычислений возможных из-за окргуления

# Приложение А Исходный код программы

Название файла: main.py

import math

import numpy as np

import random

import matplotlib.pyplot as plt

# Функция f(x)

def f(x):

    return 1 / (3 + 2 \* math.cos(x)) - x\*\*3

# Функция для добавления случайной ошибки

def add\_error(x, delta):

    if delta == 0.0:

        return x

    error = random.uniform(-delta / 2, delta / 2)

    return x + error

def method\_bisection(a, b, epsilon, delta):

    iter\_count = 0

    while (b - a) / 2 > epsilon:

        iter\_count += 1

        middle = (a + b) / 2

        f\_middle = add\_error(middle, delta)

        f\_left = add\_error(a, delta)

        if f\_middle == 0:

            return middle, iter\_count

        if f\_left \* f\_middle < 0:

            b = middle

        else:

            a = middle

    return (a + b) / 2, iter\_count

# Метод хорд с добавлением ошибки

def method\_chord(a, b, epsilon, delta):

    if f(a) \* f(b) >= 0:

        return -1, -1

    iteration = 0

    while True:

        iteration += 1

        a\_value = add\_error(f(a), delta)

        b\_value = add\_error(f(b), delta)

        # Вычисляем новое приближение

        c2 = a - (a\_value \* (b - a)) / (b\_value - a\_value)

        # Проверяем условие выхода

        if abs(f(c2)) < epsilon:

            return c2, iteration

        # Обновляем интервал

        if f(a) \* f(c2) < 0:

            b = c2

        else:

            a = c2

def explore\_iterations\_bisection(left, right, eps\_values):

    iterations = []

    for eps in eps\_values:

        \_, iter\_count = method\_bisection(left, right, eps, 0)

        iterations.append(iter\_count)

    return iterations

def explore\_iterations\_chord(left, right, eps\_values):

    iterations = []

    for eps in eps\_values:

        \_, iters = method\_chord(left, right, eps, 0)

        iterations.append(iters)

    return iterations

def explore\_error(left, right, eps\_values):

    data = []

    for delta in eps\_values:

        for eps in eps\_values:

            if eps >= delta:

                root, iterations = method\_chord(left, right, eps, delta)

                rounded\_root = round(root, int(-np.log10(eps)))

                data.append([delta, eps, rounded\_root, iterations])

    print("{:<10} {:<10} {:<15} {:<15}".format(

        "delta", "epsilon", "Значение корня", "Кол-во итераций"

    ))

    for row in data:

        delta, eps, root, iterations = row

        # Форматируем вывод с добавлением незначащих нулей

        print("{:<10} {:<10} {:<15} {:<15}".format(

            f"{delta:.6f}".rstrip('0').rstrip('.') if '.' in f"{delta:.6f}" else f"{delta:.6f}",

            f"{eps:.6f}".rstrip('0').rstrip('.') if '.' in f"{eps:.6f}" else f"{eps:.6f}",

            f"{root:.6f}".rstrip('0').rstrip('.') if '.' in f"{root:.6f}" else f"{root:.6f}",

            iterations

        ))

def plot\_iterations(left, right, x0, eps\_values):

    eps\_values\_reversed = sorted(eps\_values, reverse=True)

    iterations\_bisection = explore\_iterations\_bisection(left, right, eps\_values\_reversed)

    iterations\_chord = explore\_iterations\_chord(left, right, eps\_values\_reversed)

    plt.figure(figsize=(10, 6))

    # График для метода Бисекции

    plt.plot(eps\_values\_reversed, iterations\_bisection, marker="o", color="green", label="Метод Бисекции")

    # График для метода Хорд

    plt.plot(eps\_values\_reversed, iterations\_chord, marker='o', color='blue', label='Метод Хорд')

   # Настройки графика

    plt.xscale('log')

    plt.yscale('linear')

    plt.xlabel('Eps (точность)')

    plt.ylabel('Число итераций')

    plt.title('Зависимость числа итераций от точности Eps')

    plt.yticks(range(0, 21, 1))

    plt.legend()

    plt.grid(True)

    plt.tight\_layout()

    plt.show()

epsilons = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]

plot\_iterations(0, 1, 1, epsilons)

explore\_error(0, 1, epsilons)