**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Вычислительная математика»**

**Тема: Метод бисекции**

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

Санкт-Петербург

2025

## Цель работы

Изучить и реализовать метод бисекции 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 №3 предлагается, используя программы - функции BISECT и Round из файла methods.cpp (файл заголовков metods.h, дирек14 тория LIBR1), 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.

Вариант 26:

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

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.

## Выполнение работы

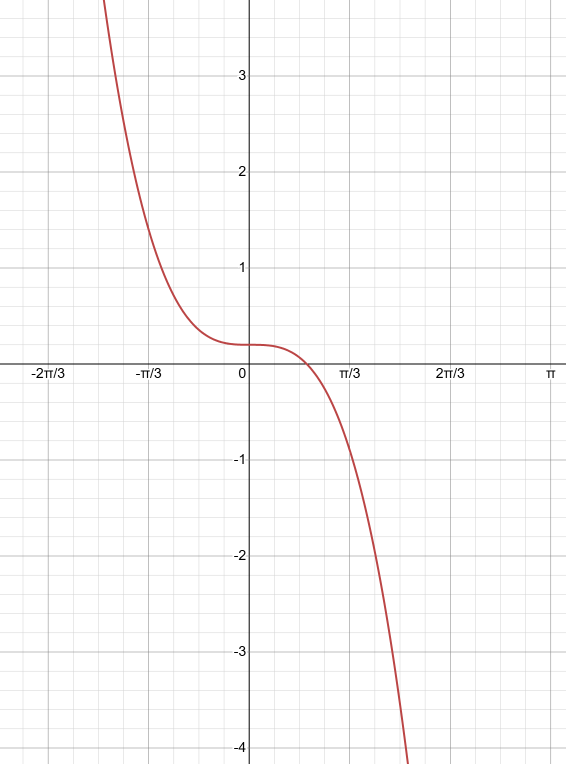


Рисунок 1

Аналитически отделим корни уравнения (Найдём интервал [left, right]):

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

Функция непрерывна (рис.1). 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:

f(0) = 0,2, что f(0) > 0

f(1) = -0,75 , f(1) < 0

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

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 Round(x, delta) – Функция принимает два аргумента x – число, которое нужно округлить, delta – величина округления, до которой будет округлено число x. Round(x, delta) выполняет округление числа x до ближайшего значения, кратного delta.

def Bisect(left, right, epsilon, delta) – функция, использующая метод бисекции для нахождения корня функции.

Метод бисекции ищет корень функции на интервале [left, right], где функция меняет знак (то есть, f(left) и f(right) имеют противоположные знаки).

Проверяем, если длина интервала right − left больше заданной точности epsilon, продолжаем работать.

На каждой итерации ищем середину отрезка middle = (left + right) / 2.

Округляем значения функции на этой средней точке и на левой границе интервала с учетом погрешности delta с помощью функции Round.

Если f(middle) = 0, то нашли корень и возвращаем его

Если произведение f(left) × f(middle) < 0, это означает, что корень находится на интервале [left, middle], и сужаем интервал, right = middle.

Если произведение f(middle) × f(right) < 0, то корень лежит на интервале [middle, right], и сужается интервал left = middle.

Дальше на каждом шаге увеличиваем счетчик итераций iter\_count для того, чтобы в итоге можно было подсчитать, сколько итераций потребовалось для достижения желаемой точности.

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

## Тестирование

Задачи:

1. Проверить правильность реализованного алгоритма с уже встроенным в модуле scipy.optimize bisect

2. Необходимо исследовать зависимость изменения epsilon от числа итераций

3. Необходимо исследовать чувствительность метода к ошибкам в исходных данных. Ошибки моделируются с использованием программы Round, округляющей значения функции с заданной точностью Delta.

**Проверка правильности реализованного алгоритма.**

Функция test\_bisect проводит несколько тестов для проверки корректности работы метода бисекции, сравнивая результаты с библиотечным методом scipy.optimize.bisect.

Тест 1: Проверка корня:

Для каждого значения точности eps из списка eps\_values функция Bisect и scipy\_bisect вычисляют корни.

Проверка, что разница между корнями не превышает eps, что подтверждает корректность работы метода с заданной точностью.

Тест 2: Проверка числа итераций:

Для каждого значения eps проверяется, что количество итераций метода бисекции не уменьшается при уменьшении точности. Это гарантирует, что метод выполняет больше шагов при более высокой точности, как ожидается от численных методов.

**Исследование зависимости изменения epsilon от числа итераций**

Функция explore\_iterations\_by\_eps исследует, как количество итераций метода бисекции 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.

Параметры:

left, right — границы интервала для поиска корня (те же, что в предыдущей функции),

eps\_values — список значений точности,

Алгоритм:

Вызывает функцию explore\_iterations\_by\_eps\_and\_delta, чтобы получить данные о числе итераций для различных значений eps.

Функция строит график с помощью библиотеки matplotlib, где по оси X откладываются значения eps (с логарифмической шкалой), а по оси Y — количество итераций.

Результаты тестирования представлены в таблице № 1 и результаты исследований представлены на рисунке № 1.

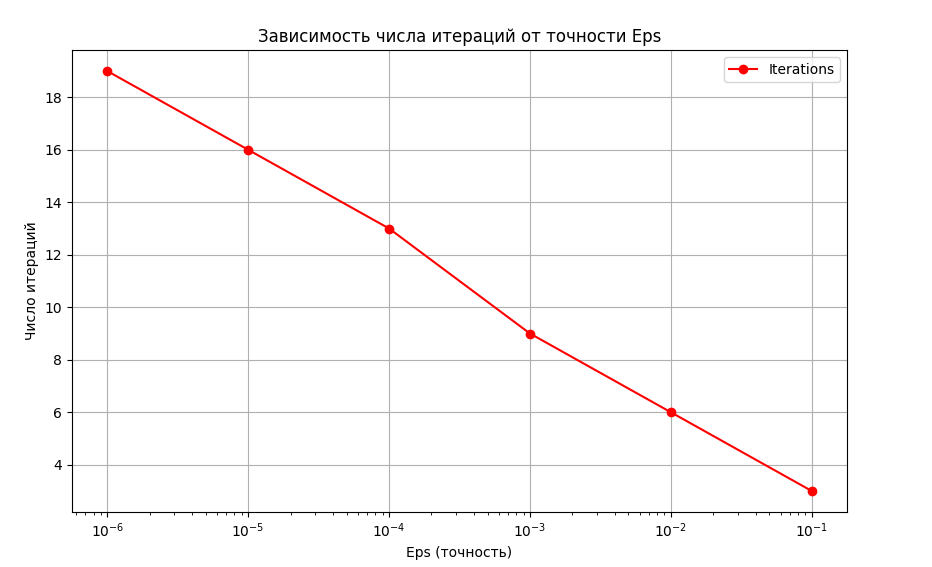


Рисунок № 1 – Зависимость числа итераций от точности Eps.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| *epsilon* | Полученный результат  (Корень) | Ожидаемый результат | Комментарии |
| 1e-1 | 0.5625 | 0.5625 | Верно |
| 1e-2 | 0.601562 | 0.601562 | Верно |
| 1e-3 | 0.598633 | 0.598633 | Верно |
| 1e-4 | 0.599060 | 0.599060 | Верно |
| 1e-5 | 0.599052 | 0.599052 | Верно |
| 1e-6 | 0.599048 | 0.599048 | Верно |

**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 test\_bisect\_with\_error тестирует работу метода бисекции с учетом погрешности и сравнивает его результаты с эталонным решением, найденным с помощью функции scipy\_bisect из библиотеки SciPy.

Описание параметров:

left – левая граница интервала, на котором ищется корень.

right – правая граница интервала.

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

|  |  |  |
| --- | --- | --- |
| delta | eps | Значение корня |
| 0.1 | 0.1 | 0.5625 |
| 0.1 | 0.01 | 0.742188 |
| 0.1 | 0.001 | 0.749023 |
| 0.1 | 0.0001 | 0.687439 |
| 0.1 | 0.00001 | 0.608299 |
| 0.1 | 0.000001 | 0.621444 |
| 0.01 | 0.1 | 0.5625 |
| 0.01 | 0.01 | 0.601562 |
| 0.01 | 0.001 | 0.600586 |
| 0.01 | 0.0001 | 0.596619 |
| 0.01 | 0.00001 | 0.597603 |
| 0.01 | 0.000001 | 0.609374 |
| 0.001 | 0.1 | 0.5625 |
| 0.001 | 0.01 | 0.601562 |
| 0.001 | 0.001 | 0.598633 |
| 0.001 | 0.0001 | 0.598938 |
| 0.001 | 0.00001 | 0.599602 |
| 0.001 | 0.000001 | 0.599006 |
| 0.0001 | 0.1 | 0.5625 |
| 0.0001 | 0.01 | 0.601562 |
| 0.0001 | 0.001 | 0.598633 |
| 0.0001 | 0.0001 | 0.599060 |
| 0.0001 | 0.00001 | 0.599052 |
| 0.0001 | 0.000001 | 0.599120 |
| 0.00001 | 0.1 | 0.5625 |
| 0.00001 | 0.01 | 0.601562 |
| 0.00001 | 0.001 | 0.598633 |
| 0.00001 | 0.0001 | 0.599060 |
| 0.00001 | 0.00001 | 0.599052 |
| 0.00001 | 0.000001 | 0.599059 |
| 0.000001 | 0.1 | 0.5625 |
| 0.000001 | 0.01 | 0.601562 |
| 0.000001 | 0.001 | 0.598633 |
| 0.000001 | 0.0001 | 0.599060 |
| 0.000001 | 0.00001 | 0.599052 |
| 0.000001 | 0.000001 | 0.599048 |

**Выводы**

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

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

2. Метод является чувствительным к ошибкам округления, и погрешности могут влиять на конечный результат, особенно при больших значениях погрешности округления.

3. Для более точных расчетов важно учитывать влияние округлений и выбирать подходящее значение δ, чтобы минимизировать ошибки.

4. График зависимости числа итераций от точности подтвердил, что увеличение точности требует большего количества шагов, что соответствует теоретическим ожиданиям.

# Приложение А Исходный код программы

Название файла: main.py

import math

import numpy as np

from scipy.optimize import bisect as scipy\_bisect

import matplotlib.pyplot as plt

import random

def f(x):

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

def Round(x, delta):

    if delta == 0.0:

        return x  # Без округления

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

    return x + error

def error\_f(x, delta):

    return Round(f(x), delta)

def Bisect\_with\_error(left, right, epsilon, delta):

    iter\_count = 0

    while (right - left) / 2 > epsilon:

        middle = (left + right) / 2

        f\_middle = error\_f(middle, delta)

        f\_left = error\_f(left, delta)

        if f\_middle == 0:

            return middle, iter\_count

        if f\_left \* f\_middle < 0:

            right = middle

        else:

            left = middle

        iter\_count += 1

    return (left + right) / 2, iter\_count

def explore\_iterations\_by\_eps(left, right, eps\_values, delta):

    iterations = []

    for eps in eps\_values:

        \_, iter\_count = Bisect\_with\_error(left, right, eps, delta)

        iterations.append(iter\_count)

    return iterations

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

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

    iterations = explore\_iterations\_by\_eps(left, right, eps\_values\_reversed, 0)

    plt.figure(figsize=(10, 6))

    plt.plot(eps\_values\_reversed, iterations, marker='o', label=f'Iterations', color="red")

    plt.xscale('log')

    plt.yscale('linear')

    plt.xlabel('Eps (точность)')

    plt.ylabel('Число итераций')

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

    plt.legend()

    plt.grid(True)

    plt.show()

def test\_bisect\_with\_error(left, right):

    eps\_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]

    delta\_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]

    print("Таблица: Влияние delta и eps на значение корня")

    print(f"{'delta':<10} {'eps':<10} {'Root':<15} {'Scipy':<15} {'Error':<10}")

    for delta in delta\_values:

        for eps in eps\_values:

            root, \_ = Bisect\_with\_error(left, right, eps, delta)

            exact\_root = scipy\_bisect(f, left, right, xtol=delta)

            error = abs(root - exact\_root)

            print(f"{delta:<10} {eps:<10} {root:<15.6f} {exact\_root:<15.6f} {error:<10.6f}")

def main():

    left, right = map(float, input("Введите границы [left right]: ").split())

    test\_bisect\_with\_error(left, right)

    eps\_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6]

    plot\_iterations(left, right, eps\_values)

main()