**зМИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №5**

**по дисциплине «Вычислительная математика»**

**Тема: Метод Ньютона**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Лодыгин И.А. |
| Преподаватель |  | Пуеров Г.Ю. |

Санкт-Петербург

2025

## Цель работы

Изучить и реализовать метод Ньютона 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 № 5 предлагается, используя программы функции NEWTON и ROUND из файла methods.cpp (файл заголовков methods.h, директория LIBR1), найти корень уравнения f(x) = 0 с заданной точностью Eps

методом Ньютона, исследовать скорость сходимости и обусловленность метода.

Для данной работы вид функции f(x) задается индивидуально каждому студенту преподавателем из числа вариантов, приведенных в подразделе 3.6. Вариант 9:

f(x) = x^2 – x^3 - 1/(4 + x^2)

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 сходимости метода Ньютона).

2) Составить подпрограмму вычисления функции f (x), f′(x), предусмотрев округление их значений с заданной точностью delta.

3) Составить головную программу, вычисляющую корень уравнения f(x) = 0 и содержащую обращение к подпрограммам, f(x), f′(x), Round, Newton и индикацию результатов.

4) Выбрать начальное приближение корня x0 из [Left, Right], так чтобы f(x) \* f′(x) > 0

5) Провести вычисления по программе. Исследовать скорость сходимости метода и чувствительность метода к ошибкам в исходных данныхю

Для приближённого вычисления корней уравнения f(x) = 0 методом Ньютона предназначена программа – функция Newton, текст который представлен в подразделе 3.7.

## Выполнение работы

Найдём интервал [Left, Right]

f′(x) = (2\*x)/((x^2 + 4)^2) - 3\*x^2 + 2\*x

Графическое отделение:

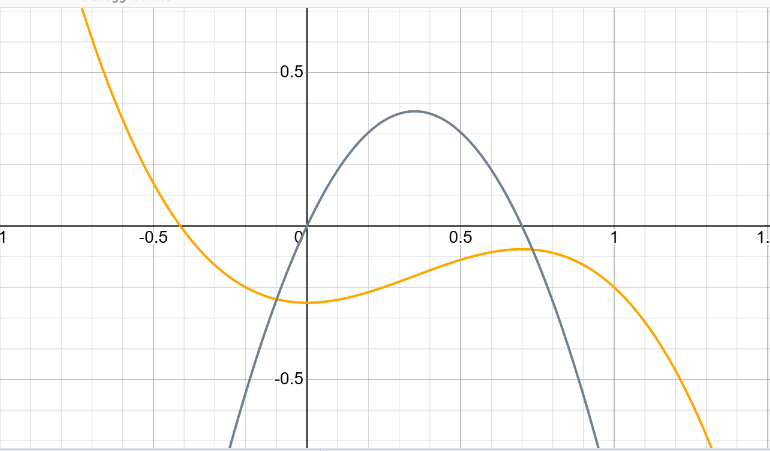


Рисунок № 1 – График f(x)(желтый) и f′(x)(голубой)

Из **Рисунка 1** видно, что функция f(x) имеет корень на интервале [-0.5, 0]. Однако для метода Ньютона важно выбрать интервал и начальное приближение так, чтобы обеспечить быструю и устойчивую сходимость.

Производная f′(x) обращается в ноль в точке x == 0, так что может привести к неустойчивости метода Ньютона, так как формула xn+1=xn−f(xn) / f′(xn) становится неопределённой или медленно сходящейся, если f′(xn) близко к нулю.

**Выбор интервала** [-0.5, -0.1]: производная f′(x)не обращается в ноль, есть корень. Для быстрой сходимости метода Ньютона начальное приближение x0 должно удовлетворять условию f(x0) \* f′(x0)> 0

В точке x0 = -0.1:

f(-0.1) = -0.238(отрицательное значение).

f′(-0.1) = -0.242(отрицательное значение).

Таким образом, f(-0.1) \* f′(-0.1) >0, что удовлетворяет условию быстрой сходимости.

**Реализация на Python**

def method\_newton(x0, epsilon, delta) – функция реализует метод ньютона для поиска корня.

Алгоритм:

На каждом шаге вычисляем значение функции f(xn)и её производной f′(xn)в текущей точке xn.

Если производная f′(xn)близка к нулю (меньше epsilon), метод останавливается, так как дальнейшие вычисления могут быть неустойчивыми.

Вычисляем новое приближение xn+1 по формуле:

xn+1=xn−f(xn) / f′(xn).

xn+1 становится текущим приближением для следующей итерации.

Условиевыхода

Если разница между текущим и предыдущим приближениями ∣xn+1−xn∣ становится меньше заданной точности epsilon, метод завершается, и xn+1считается найденным корнем.

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

def derivative\_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 explore\_iterations\_newton(x0, 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, также вызывается функция plot\_iterations, которая содержит в себе исследование зависимости изменения epsilon от числа итераций.

**Исследование зависимости изменения epsilon от числа итераций**

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

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:

Функция plot\_iterations строит график зависимости числа итераций от точности eps, для исследования были построены графики метод бисекции, хорд, Ньютона.

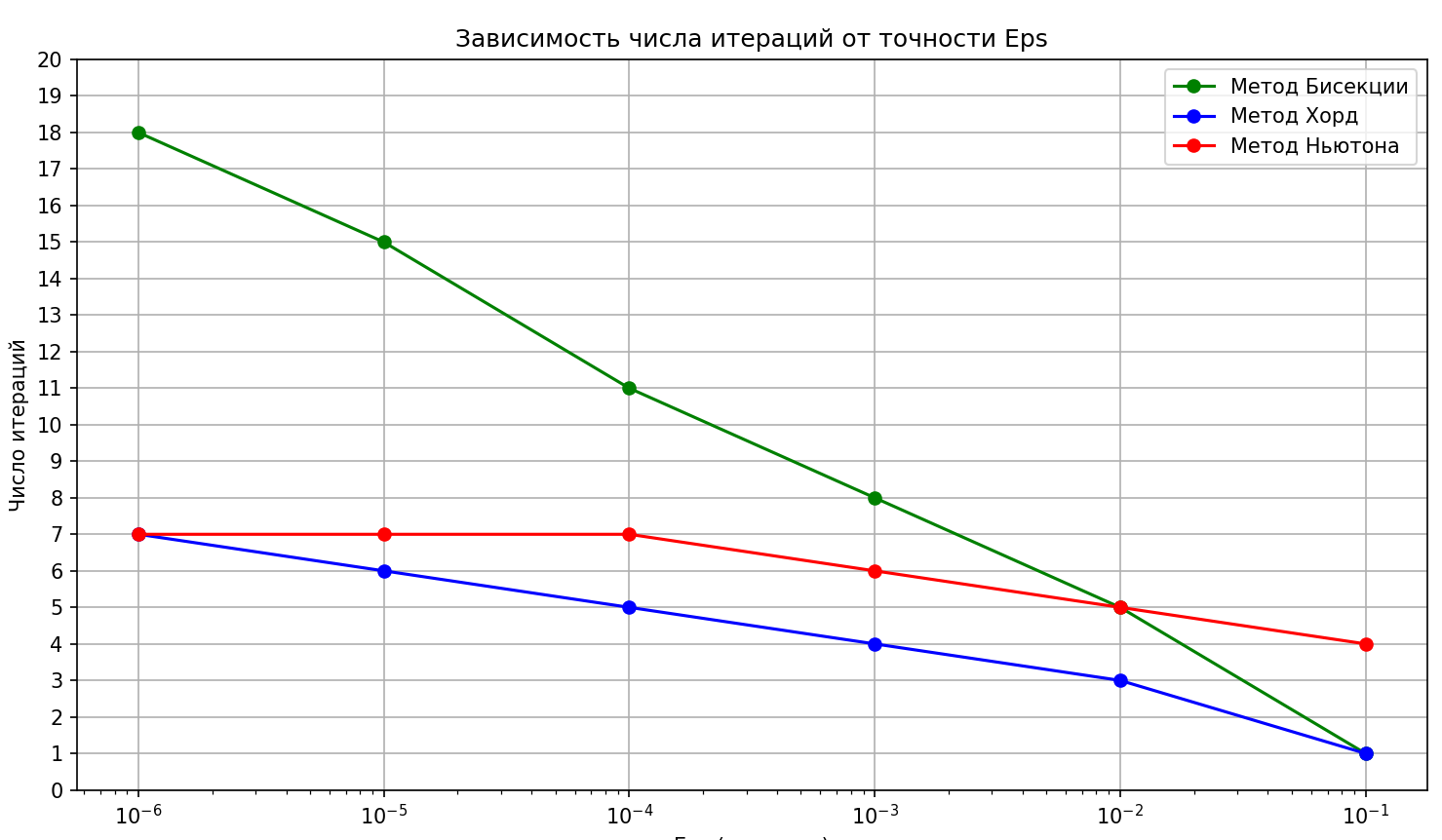


Рисунок № 2 – Зависимость числа итераций от точности Eps методов.

Метод Ньютона является быстрым, но требует аккуратного выбора начального приближения. Число итераций растёт очень медленно для 0.1(4 итерации, для 1e-6(7 итераций).

Метод хорд является компромиссом между скоростью и устойчивостью.

Метод бисекции — самый медленный, но наиболее устойчивый.

**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(x0, eps\_values): тестирует работу метод Ньютона с учетом погрешности.

Таблица 1 – Результаты перебора eps и delta для промежутка [-0.5, -0.1]

|  |  |  |
| --- | --- | --- |
| delta | eps | Значение корня |
| 0.1 | 0.1 | -0.4 |
| 0.01 | 0.1 | -0.4 |
| 0.01 | 0.01 | -0.41 |
| 0.001 | 0.1 | -0.4 |
| 0.001 | 0.01 | -0.41 |
| 0.001 | 0.001 | -0.412 |
| 0.0001 | 0.1 | -0.4 |
| 0.0001 | 0.01 | -0.41 |
| 0.0001 | 0.001 | -0.412 |
| 0.0001 | 0.0001 | -0.4121 |
| 0.00001 | 0.1 | -0.4 |
| 0.00001 | 0.01 | -0.41 |
| 0.00001 | 0.001 | -0.412 |
| 0.00001 | 0.0001 | -0.4121 |
| 0.00001 | 0.00001 | -0.4121 |
| 0.000001 | 0.1 | -0.4 |
| 0.000001 | 0.01 | -0.41 |
| 0.000001 | 0.001 | -0.412 |
| 0.000001 | 0.0001 | -0.4121 |
| 0.000001 | 0.00001 | -0.4121 |
| 0.000001 | 0.000001 | -0.412105 |

**Корень =** -0.412105

**Хорошая обусловленность**:

Наблюдается при малых delta (например, delta =0.000001) и epsilon, f′(x) не близка к нулю.

В нашем случае при delta=0.000001метод Ньютона демонстрирует хорошую обусловленность.

**Плохая обусловленность**:

Наблюдается при больших delta и epsilon, а также когда f′(x)близка к нулю.

В нашем случае delta = 0.1 ошибка в результате значительна

**Выводы**

В ходе выполнения задания, направленного на исследование метода Ньютона для нахождения корней уравнения f(x) = x^2 – x^3 - 1/(4 + x^2), можно подвести следующие итоги:

1. Метод Ньютона эффективно находит корень уравнения f(x) на интервале [-0.5, -0.1].
2. Метод чувствителен к ошибкам округления, особенно при больших delta. Для повышения точности необходимо минимизировать delta.
3. Увеличение точности epsilon требует большего количества итераций, что соответствует теоретическим ожиданиям и рост медленный для метода Ньютона.

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

# Производная функции f(x)

def derivative\_f(x):

    return (2\*x)/pow((pow(x, 2) + 4), 2) - 3\*pow(x, 2) + 2\*x

# 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

# Метод Ньютона с добавлением ошибки

def method\_newton(x0, epsilon, delta):

    iteration = 0

    xn = x0

    while True:

        iteration += 1

        # Добавляем ошибку к значениям функции и производной

        fxn = add\_error(f(xn), delta)

        derivative\_fxn = add\_error(derivative\_f(xn), delta)

        # Проверка, что производная не близка к нулю

        if abs(derivative\_fxn) < epsilon:

            return -1, -1

        # Вычисление следующего приближения

        xn1 = xn - fxn / derivative\_fxn

        # Проверка условия выхода

        if abs(xn1 - xn) < epsilon:

            return xn1, iteration

        xn = xn1

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\_iterations\_newton(x0, eps\_values):

    iterations = []

    for eps in eps\_values:

        \_, iters = method\_newton(x0, 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\_newton(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)

    iterations\_newton = explore\_iterations\_newton(x0, 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')

    # График для метода Ньютона

    plt.plot(eps\_values\_reversed, iterations\_newton, marker='o', color='red', label='Метод Ньютона')

   # 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()

def main():

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

    plot\_iterations(-0.5, -0.1, -0.1, epsilons)

    explore\_error(-0.5, -0.1, epsilons)

main()