**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №6**

**по дисциплине «Вычислительная математика»**

**Тема: Метод простых итераций**

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

Санкт-Петербург

2025

## Цель работы

Изучить и реализовать метод простых итераций 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 № 6 предлагается, используя программы функции ITER и Round из файла methods.cpp (файл заголовков methods.h, директория LIBR1), найти корень уравнения f(x)=0 с заданной точностью Eps методом итераций, 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 лабораторной работы № 6 должен быть следующим.

1) Графически или аналитически отделить корень уравнения f(x)=0.

2) Преобразовать уравнение f(x)=0к виду x=ϕ(x)так, чтобы в некоторой окрестности [Left,Right] корня производная ϕ′(x)удовлетворяла условию ∣ϕ′(x)∣≤q<1. При этом следует иметь в виду, что чем меньше величина q, тем быстрее последовательные приближения сходятся к корню.

3) Выбрать начальное приближение, лежащее на отрезке [Left,Right].

4) Составить подпрограмму для вычисления значений ϕ(x)и ϕ′(x), предусмотрев округление вычисленных значений с точностью Delta.

5) Составить головную программу, вычисляющую корень уравнения и содержащую обращение к программам ϕ(x), ϕ′(x)и ITER, а также индикацию результатов.

6) Провести вычисления по программе. Исследовать скорость сходимости и обусловленность метода.

Текст программы – функции ITER, позволяющей вычислить корни уравнения x = ϕ(x) для любой функции, которая удовлетворяет достаточным условиям сходимости метода, приводится ниже

## Выполнение работы

Найдём интервал [Left, Right]

Графическое отделение:

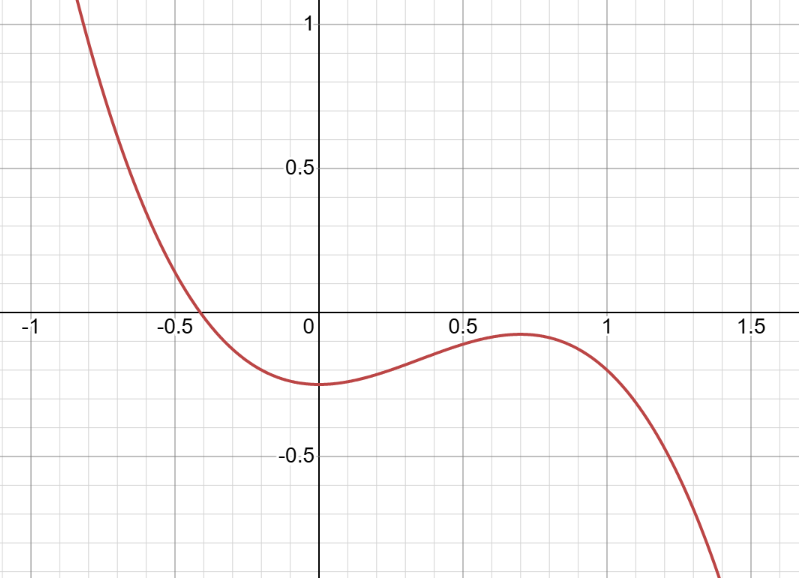


Рисунок № 1 – График функции f(x) = x^2 – x^3 - 1/(4 + x^2)

Из рисунка1 увидим, что функция положительна при f(-0.5) и отрицательна при f(-0.1), возьму интервал [-0.5, -0.1].

Аналитическое отделение:

Рассмотрим уравнение f(x) = x^2 – x^3 - 1/(4 + x^2)

Функция непрерывна. 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 [-0.5, -0.1] и вычислим значение в функции в точках:

f(-0.5) = 0.1397 => что f(-0.5) > 0

f(-0.1) = -0.242=> f(-0.1) < 0

Так как f(-0.5) > 0 и f(-0.1) < 0, выбираем отрезок [left, right] = [-0.5, -0.1]

**Преобразование уравнения к виду x = ϕ(x)**

Исходное уравнение имеет вид:

f(x) = x^2 – x^3 - 1/(4 + x^2)

Для применения метода простых итераций необходимо преобразовать уравнение к эквивалентному виду x=ϕ(x). Для этого выразим xиз исходного уравнения: x^2 – x^3 - 1/(4 + x^2)

x = x^2 – x^3 - 1/(4 + x^2) + x

Таким образом, уравнение преобразуется к виду:

x=ϕ(x) , где ϕ(x) = x = x^2 – x^3 - 1/(4 + x^2) + x

Важно отметить, что функция ϕ(x)должна удовлетворять условию сходимости метода, то есть ∣ϕ′(x)∣ < 1 в окрестности корня [left, right], чем меньше q тем быстрее метод будет сходиться.

ϕ′(x) = (2\*x)/((x^2 + 4)^2) – 3\*x^2 + 2\*x + 1

На интервале [-0.5, -0.1] возрастает:

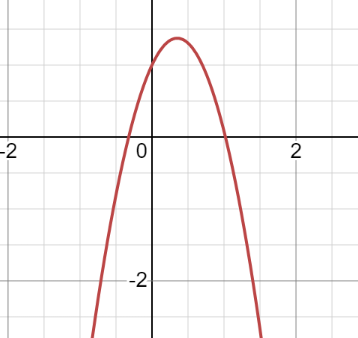


Рисунок № 2 - ϕ′(x) = (2\*x)/((x^2 + 4)^2) – 3\*x^2 + 2\*x + 1

∣ϕ′(-0.5)∣ = 0.805

∣ϕ′(-0.1)∣ = 0.757

q = max (∣ϕ′(x)∣) = 0.805 существование q гарантируется теоремой о сходимости метода простых итераций.

Начальное приближение возьмём середину интервала [-0.5, -0.1], чтобы минимизировать начальное отклонение от корня x0 = -0.3

**Реализация на Python**

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

def equal\_f(x) – Функция возвращает результат ϕ(x) = x^2 – x^3 - 1/(4 + x^2) + x

def method\_simple\_iter(x0, epsilon, delta) – функция реализует метод простых итераций

Алгоритм:

На каждом шаге вычисляется новое приближение xn+1=ϕ(xn), и процесс продолжается, пока разница между текущим и предыдущим приближениями не станет меньше заданной точности *epsilon*. Код возвращает найденное значение корня и количество итераций, затраченных на его вычисление.

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 explore\_simple\_iter (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\_simple\_iter исследует, как количество итераций метода простых итераций зависит от значения eps (точности).

Функция plot\_iterations строит график зависимости числа итераций от точности eps.

Параметры:

left, right — границы интервала для поиска корня

x0 – начальное приближение

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 и простых итераций.

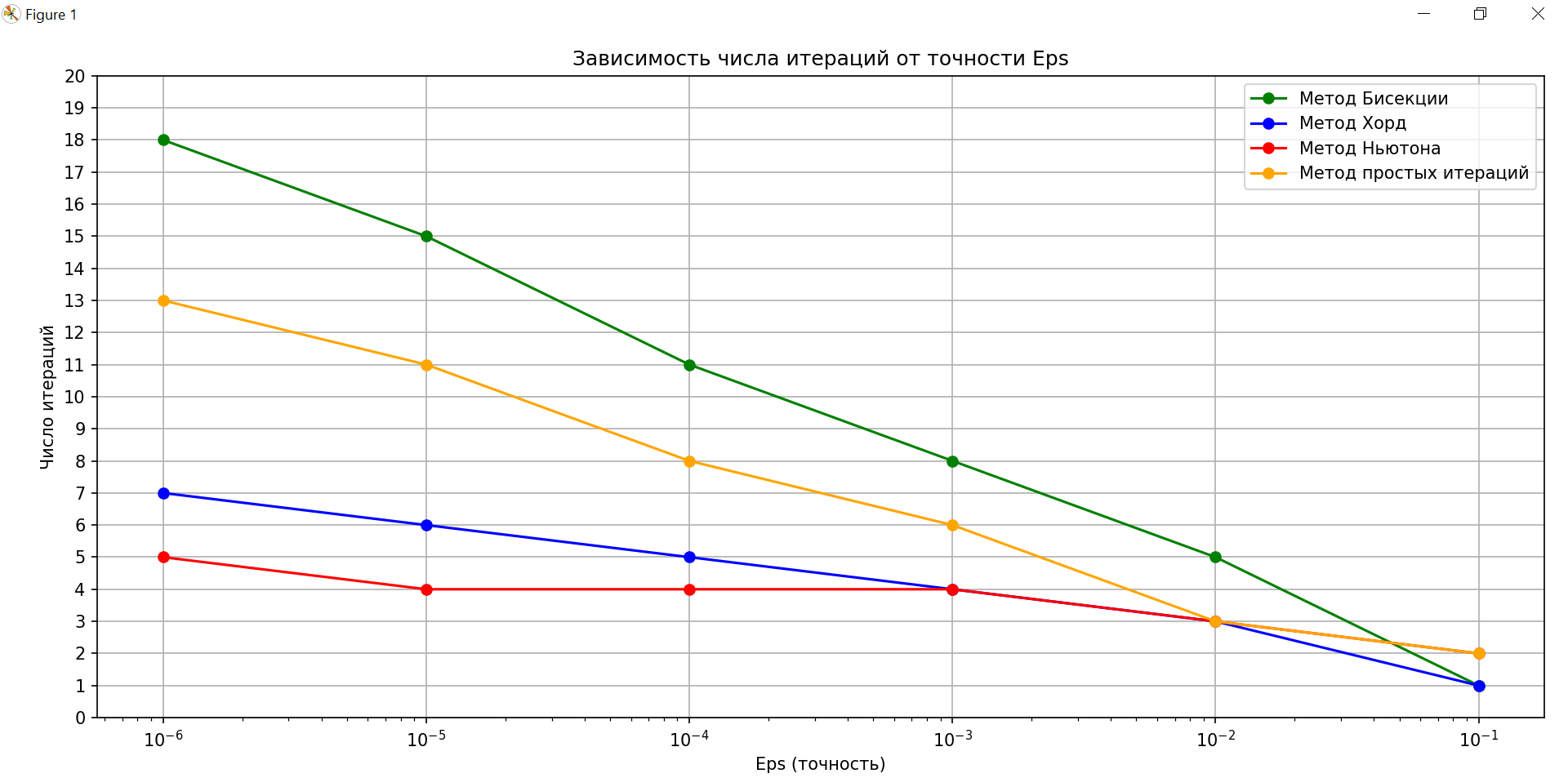


Рисунок № 3 – Графики методов зависимость итераций от Eps

**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 метода простых итераций с учетом погрешности.

Таблица 1 – Результаты перебора eps и delta для промежутка [-0.5, -0.1] и начальное приближение x0 = -0.3

|  |  |  |
| --- | --- | --- |
| 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 | -0.42 |
| 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 | -0.41211 |
| 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**

Обусловленность метода зависит от выбора функции ϕ(x). Если ∣ϕ′(x)∣ близко к 1, то метод может сходиться медленно или даже расходиться.

Чем меньше q = max∣ϕ′(x)∣, тем лучше обусловленность метода.

**Выводы**

В ходе выполнения задания, направленного на исследование метода простых итераций для нахождения корней уравнения f(x) = 0, можно подвести следующие итоги:

1. Метод прост в реализации, так как требует только вычисления функции ϕ(x) на каждом шаге и не требует знания производной f′(x)*,* в отличие от метода Ньютона.
2. Скорость сходимости метода зависит от величины q = max (∣ϕ′(x)∣)
3. Было выявлено, что для решений уравнения f(x) наиболее быстрым и менее затратным по итерациям будет метод Ньютона.

# Приложение А Исходный код программы

Название файла: main.py

import math

import numpy as np

import random

import matplotlib.pyplot as plt

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

# Эквивалентное f(x): ϕ(x)

def equal\_f(x):

    return pow(x, 2) - pow(x, 3) - 1/(4 + pow(x, 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 method\_simple\_iter(x0, epsilon, delta):

    iteration = 0

    xn = x0

    while True:

        iteration += 1

        xn1 = add\_error(equal\_f(xn), delta)

        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\_simple\_iter(x0, eps\_values):

    iterations = []

    for eps in eps\_values:

        \_, iters = method\_simple\_iter(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\_simple\_iter(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)

    iterations\_simple\_iter = explore\_simple\_iter(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')

    # График для метода простых итераций

    plt.plot(eps\_values\_reversed, iterations\_simple\_iter, marker='o', color='orange', label="Метод простых итераций")

   # 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, -0.3, epsilons)

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

main()