**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский Авиационный Институт»**

**(Национальный Исследовательский Университет)**

**Институт: №8 «Информационные технологии   
и прикладная математика»   
Кафедра: 806 «Вычислительная математика   
и программирование»**

Лабораторная работа №2   
по курсу «Численные методы»

Группа: М8О-307Б-22

Студент(ка): П. В. Лебедько

Преподаватель: Д. Л. Ревизников

Оценка:

Дата: 02.04.2025

Москва, 2025

**ОГЛАВЛЕНИЕ**

[**1** **Задание 1** 3](#_Toc194493504)

[**Задание** 3](#_Toc194493505)

[**Вариант** 3](#_Toc194493506)

[**Ход лабораторной работы** 3](#_Toc194493507)

[Определение положительного корня 3](#_Toc194493508)

[Метод простой итерации 3](#_Toc194493509)

[Метод Ньютона 4](#_Toc194493510)

[Результаты 4](#_Toc194493511)

[**2** **Задание 2** 4](#_Toc194493512)

[**Задание** 4](#_Toc194493513)

[**Вариант** 5](#_Toc194493514)

[**Ход лабораторной работы** 5](#_Toc194493515)

[Определение положительного корня 5](#_Toc194493516)

[Метод простой итерации 5](#_Toc194493517)

[Метод Ньютона 6](#_Toc194493518)

[Результаты 6](#_Toc194493519)

[**3** **Выводы** 6](#_Toc194493520)

# **Задание 1**

## **Задание**

Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

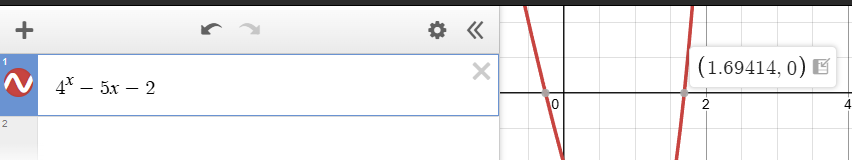
## **Вариант**

**Вариант 17**



## **Ход лабораторной работы**

### Определение положительного корня



### Метод простой итерации

Необходимо было для моей конкретной функции найти функцию phi:

def phi(x):

    return math.log(5 \* x + 2, 4)

А также ограничить производную числом q:

q = 5 / (math.log1p(3) \* 7)

Тогда реализованный алгоритм будет выглядеть так:

def simpleIterations(x0, q, eps):

    xPrev = x0

    iter = 0

    while (True):

        iter += 1

        xCur = phi(xPrev) # вычисление нового значения x

        error = q / (1 - q) \* abs(xCur - xPrev) # текущая погрешность

        if error < eps:

            break

        xPrev = xCur

    return xCur, iter

где x0 – начальная точка, посмотрев на график, взяли 1.5, eps – задаваемая точность. Условие окончания проверяется через неравенство, зависящее от числа q.

### Метод Ньютона

Необходимо в явном виде найти производную исходной функции:

def fDer(x):

    return math.log1p(3) \* 4 \*\* x - 5

Реализованный алгоритм:

def newton(x0, eps):

    xPrev = x0

    iter = 0

    while (True):

        iter += 1

        xCur = xPrev - f(xPrev) / fDer(xPrev) # вычисление нового значения x

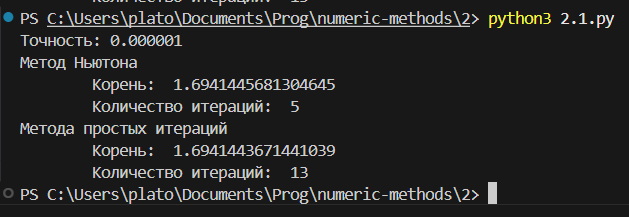
        if abs(xCur - xPrev) < eps: # погрешность – модуль разности значений

            break

        xPrev = xCur

    return xCur, iter

### Результаты



# **Задание 2**

## **Задание**

Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

## **Вариант**

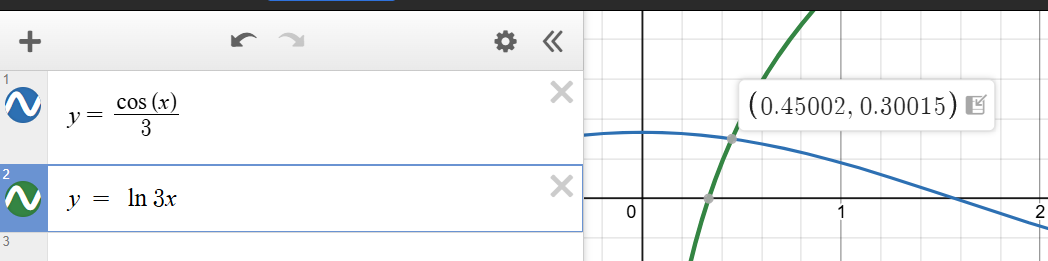
**Вариант 17**

a = 3



## **Ход лабораторной работы**

### Определение положительного корня



### Метод простой итерации

Также определим вектор-функцию phi:

def phi(x):

    return np.array([

        np.cos(x[1]) / 3,

        np.exp(x[0]) / 3

    ])

Теперь ограничить нужно норму матрицы частных производных функции phi

q = np.e / 3

Реализованный алгоритм:

def simpleIterations(x0, q, eps):

    xPrev = x0

    iter = 0

    while (True):

        iter += 1

        xCur = phi(xPrev)

        error = q / (1 - q) \* np.linalg.norm(xCur - xPrev, NORM)

        if error < eps:

            break

        xPrev = xCur

    return xCur, iter

где x0 – начальная точка, посмотрев на график, взяли точку (0, 0.3), eps – задаваемая точность. Условие окончания проверяется через неравенство, зависящее от числа q.

### Метод Ньютона

Необходимо в явном виде найти производную исходной вектор-функции:

def fDer(x):

    return np.array([

        [3, np.sin(x[1])],

        [-np.exp(x[0]), 3]

    ])

Реализованный алгоритм:

def newton(x0, eps):

    xPrev = x0

    iter = 0

    while (True):

        iter += 1

        (LU, swaps) = getLU(fDer(xPrev))

        xDelta = solverLU(LU, swaps, -f(xPrev))

        xCur = xPrev + xDelta

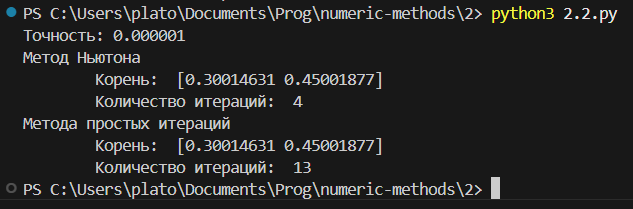
        if np.linalg.norm(xCur - xPrev, NORM) < eps:

            break

        xPrev = xCur

    return xCur, iter

### Результаты



# **Выводы**

В ходе лабораторной работы я реализовал методы простых итераций и Ньютона для решения нелинейных уравнений и систем нелинейных уравнений. Метод Ньютона сходится быстрее, но требует вычисления производных.