**Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»**

**Факультет программной инженерии и компьютерной техники**

**Вычислительная математика**

**Лабораторная работа №2**

**Вариант 2**

Группа: P3267

Выполнила:

Бореева В. Ю.  
Тихонов И. В.

Проверил:

Машина Е. А.

Г. Санкт-Петербург

2024

# Цель работы

Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

# 1 Вычислительная реализация задачи:

**1 часть. Решение нелинейного уравнения**

Изображение выглядит как График, линия, текст, число

Автоматически созданное описание

1. Приближенные значения корней:

Интервалы изоляции корней:

(-4, -3,5); (-2,-1) и (1,2)

1. Центральный корень: метод простой итерации

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | xk | xk+1 | f(xk+1) | │ xk+1- xk│ |
| 1 | -2 | -1.478964401294498 | -0.242009551 | 0.5210355987 |
| 2 | -1.4789644 | -1.45285765 | 0.0076705 | 0.02610674 |
| 3 | -1.45285765 | **-1.4536851** | -0.000256 | 0.00082745 |

1. Крайний левый корень: метод хорд

Для удобства исчислений составим таблицу Exel

Изображение выглядит как текст, снимок экрана, Шрифт, число

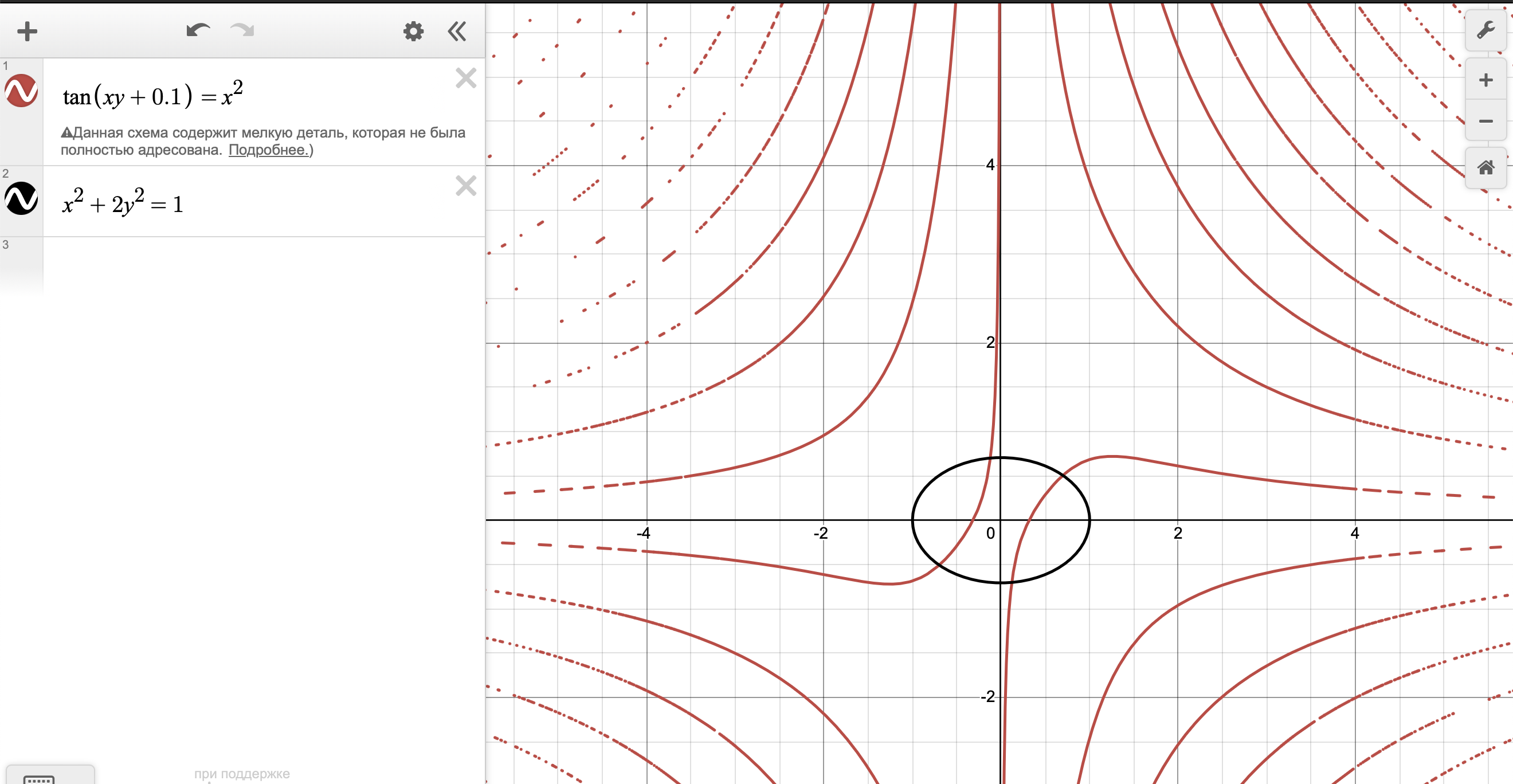
Автоматически созданное описание

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № шага |  |  |  |  |  |  |  |
| 0 | -4 | -3,5 | -3,84952 | 2,27 | -5,2725 | -0,538694 |  |
| 1 | -3,849519 | -3,5 | -3,889295 | -0,538705 | -5,2725 | 0,1562296 | 0,1504806 |
| 2 | -3,889295 | -3,5 | -3,878091 | 0,1562296 | -5,2725 | -0,042905 | 0,0397752 |
| 3 | -3,878091 | -3,5 | -3,881193 | -0,042905 | -5,2725 | 0,0119637 | 0,0112032 |
| 4 | -3,881193 | -3,5 | **-3,8803303** | 0,0119637 | -5,2725 | -3,8805701 | 0,003102 |

1. Крайний правый корень: метод половинного деления

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | a | b | x | f(a) | f(b) | f(x) | |a – b| |
| 1 | 1 | 2 | 1,5 | 6,72 | -16,63 | -2,0475 | 1 |
| 2 | 1 | 1,5 | 1,25 | 6,72 | -2,0475 | 2,99844 | 0,5 |
| 3 | 1,25 | 1,5 | 1,375 | 2,99844 | -2,0475 | 0,649102 | 0,25 |
| 4 | 1,5 | 1,375 | 1,4375 | -2,0475 | 0,649102 | -0,65478 | 0,125 |
| 5 | 1,4375 | 1,375 | 1,40625 | -0,65478 | 0,343984 | 0,00813904 | 0,0625 |
| 6 | 1,4375 | 1,40625 | 1,421875 | -0,65478 | 0,00813904 | -0,32056 | 0,03125 |
| 7 | 1,421875 | 1,40625 | 1,4140625 | -0,32056 | 0,00813904 | -0,155522 | 0,015625 |
| 8 | 1,4140625 | 1,40625 | **1,41015625** | -0,155522 | 0,00813904 | -0,735199 | 0,00781 |
|  |  |  |  |  |  |  |  |

**2 часть. Решение системы нелинейных уравнений**

****

**Метод Ньютона:**

**---->**

Построим матрицу Якоби

Выбираем

*Для дальнейших исчислений составим таблицу Exel*

**Изображение выглядит как текст, Шрифт, число, снимок экрана

Автоматически созданное описание**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № интерации |  |  |  |  |  |  |
| 0 | -1 | 1 |  |  |  |  |
| 1 | -0,706691 | 0,646655 | 0,2965 | -0,291811 | -0,410191 | 0,354844 |
| 2 | -0,410191 | 0,354844 | 0,270089 | 0,252462 | -0,140102 | 0,607306 |
| 3 | -0,140102 | 0,607306 | 0,0207062 | 0,0975325 | -0,1193958 | 0,7048385 |
| 4 | **-0,1193958** | **0,7048385** | -0,0020164 | -0,0026135 |  |  |

# 2 Программная реализация задачи:

**import** math

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**def** calculate\_fk(num, k):

**if** num == 1:

**return** -1.38 \* (k \*\* 3) - 1.93 \* (k \*\* 2) - 15.28 \* k - 3.72

**elif** num == 2:

**return** k - 10 \* math.sin(k)

**elif** num == 3:

**return** 4 \* (k \*\* 3) - 7 \* k + 5

**elif** num == 4:

**return** k \*\* 3 - k + 4

**elif** num == 5:

**return** -1.38 \* (k \*\* 3) - 5.42 \* (k \*\* 2) + 2.57 \* k + 10.95

**def** derivative1(num, k):

**if** num == 1:

**return** -4.14 \* (k \*\* 2) - 3.86 \* k - 15.28

**elif** num == 2:

**return** 1 - 10 \* math.cos(k)

**elif** num == 3:

**return** 12 \* (k \*\* 2) - 7

**elif** num == 4:

**return** 3 \* (k \*\* 2) - 1

**elif** num == 5:

**return** -4.14 \* (k \*\* 2) - 10.84 \* k + 2.57

**def** derivative2(num, k):

**if** num == 1:

**return** -8.28 \* k - 3.86

**elif** num == 2:

**return** 10 \* math.sin(k)

**elif** num == 3:

**return** 24 \* k

**elif** num == 4:

**return** 6 \* k

**elif** num == 5:

**return** -8.28 \* k - 10.84

**def** graph(xmin, xmax, count, num, sys=**False**):

xlist = np.linspace(xmin, xmax, count)

**if** sys:

x = np.linspace(-10, 10, 400)

x1, x2 = np.meshgrid(x, x)

z1 = calculate\_graph\_sys(num, x1, x2)[0]

z2 = calculate\_graph\_sys(num, x1, x2)[1]

# Построение графика с обеими функциями

plt.figure(figsize=(10, 7))

plt.contour(x1, x2, z1, levels=[0], colors='r')

plt.contour(x1, x2, z2, levels=[0], colors='b')

plt.xlabel('x1')

plt.ylabel('x2')

plt.title('Графики обеих функций на одном графике')

**else**:

ylist = [calculate\_fk(num, x) **for** x **in** xlist]

plt.plot(xlist, ylist)

ax = plt.gca()

ax.spines['left'].set\_position('center')

ax.spines['bottom'].set\_position('center')

ax.spines['top'].set\_visible(**False**)

ax.spines['right'].set\_visible(**False**)

plt.show()

**def** calculate\_fx\_sys(num, x1, x2):

**if** num == 1:

**return** [0.3 - 0.1 \* (x1 \*\* 2) - 0.2 \* (x2 \*\* 2), 0.7 - 0.2 \* (x1 \*\* 2) + 0.1 \* x1 \* x2]

**elif** num == 2:

**return** [1.5 - math.cos(x2), (1 + math.sin(x1 - 0.5)) / 2]

**def** calculate\_graph\_sys(num, x1, x2):

**if** num == 1:

**return** [0.1 \* (x1 \*\* 2) + x1 + 0.2 \* (x2 \*\* 2) - 0.3, 0.2 \* (x1 \*\* 2) + x2 + +0.1 \* x1 \* x2 - 0.7]

**elif** num == 2:

**return** [math.cos(x2) + x1 - 1.5, 2 \* x2 - math.sin(x1 - 0.5) - 1]

**def** derivative\_sys(num, x1, x2):

**if** num == 1:

**return** [abs(-0.2 \* x1) + abs(-0.4 \* x2), abs(-0.4 \* x1 - 0.1 \* x2) + abs(-0.1 \* x1)]

**elif** num == 2:

**return** [abs(math.sin(x2)), abs(math.cos((2 \* x1 - 1) / 2) / 2)]

import math  
  
from Equations import calculate\_fk, derivative1, derivative2  
  
  
def \_\_fixed\_hordes(first, second, equ, eps):  
 prev\_x = second  
 fx = calculate\_fk(equ, prev\_x)  
 fab = calculate\_fk(equ, first)  
 counter = 0  
 while True:  
 x = prev\_x - (first - prev\_x) / (fab - fx) \* fx  
 counter += 1  
 fx = calculate\_fk(equ, x)  
 if abs(x - prev\_x) < eps:  
 break  
 prev\_x = x  
 return [counter, x]  
  
  
def \_\_unfixed\_hordes(a, b, equ, eps):  
 prev\_x = math.inf  
 counter = 0  
 while True:  
 fa = calculate\_fk(equ, a)  
 fb = calculate\_fk(equ, b)  
 x = a - (b - a) / (fb - fa) \* fa  
 counter += 1  
  
 if abs(x - prev\_x) < eps:  
 break  
 if a \* x < 0:  
 b = x  
 elif b \* x < 0:  
 a = b  
 b = x  
  
 prev\_x = x  
 return [counter, x]  
  
  
def hordes(a, b, equ, eps):  
 fa = calculate\_fk(equ, a)  
 fb = calculate\_fk(equ, b)  
 x = a - (b - a) / (fb - fa) \* fa  
 der1 = derivative1(equ, x)  
 der2 = derivative2(equ, x)  
 if der1 \* der2 < 0:  
 return \_\_fixed\_hordes(a, b, equ, eps)  
 elif der1 \* der2 > 0:  
 return \_\_fixed\_hordes(b, a, equ, eps)  
 else:  
 return \_\_unfixed\_hordes(a, b, equ, eps)

import Hordes  
import Newton  
import SimpleIterations  
import SystemSimpleIterations  
from Equations import graph  
  
e = 0.01 # Точность  
  
equation\_type = int(input("Выберите задачу:\n1. Нелинейное уравнение.\n2. Система нелинейных уравнений.\n> "))  
if equation\_type == 1:  
 equation = int(  
 input(  
 "Выберите уравнение:\n1. -1,38x^3 - 1,93x^2 - 15,28x - 3,72 = 0\n2. - 10sin(x) = 0\n3. 4x^3 - "  
 "7x + 5 = 0\n4. x^3 - x + 4 = 0\n5. -1,38x^3 - 5,42x^2 + 2,57x + 10,95 = 0\n> "))  
 if equation not in [1, 2, 3, 4, 5]:  
 raise ValueError("Неизвестное уравнение.")  
  
 lower\_b = -2 # Нижняя граница  
 upper\_b = -1 # Верхняя граница  
  
 datamode = int(input("Выберите режим ввода данных:\n1. С клавиатуры.\n2. Из файла.\n> "))  
 if datamode == 1:  
 lower\_b = int(input("Введите нижнюю границу интервала: "))  
 upper\_b = int(input("Введите верхнюю границу интервала: "))  
 e = float(input("Введите точность: "))  
 elif datamode != 2:  
 raise ValueError("Неверный режим ввода данных.")  
  
 method = int(input("Выберите метод:\n1. Метод хорд.\n2. Метод Ньютона.\n3. Метод простой итерации.\n> "))  
 if method == 1:  
 result = Hordes.hordes(lower\_b, upper\_b, equation, e)  
 elif method == 2:  
 result = Newton.newton(lower\_b, upper\_b, equation, e)  
 elif method == 3:  
 result = SimpleIterations.simpleIterations(lower\_b, upper\_b, equation, e)  
 else:  
 raise ValueError("Неизвестный метод.")  
 print(f"Количество итераций: {result[0]}")  
 print(round(result[1], 5))  
 graph(-20.0, 20.0, 200, equation)  
elif equation\_type == 2:  
 equation = int(  
 input(  
 "Выберите систему уравнений:\n1. 0.1x(1)^2 + x(1) + 0.2x(2)^2 - 0.3 = 0; 0.2x(1)^2 + x(2) + 0.1x(1)x(2) - "  
 "0.7 = 0\n2. cos(x(2)) + x(1) - 1,5 = 0; 2x(2) - sin(x(1) - 0,5) - 1 = 0\n> "))  
  
 datamode = int(input("Выберите режим ввода данных:\n1. С клавиатуры.\n2. Из файла.\n> "))  
 if datamode == 1:  
 e = float(input("Введите точность: "))  
 elif datamode != 2:  
 raise ValueError("Неверный режим ввода данных.")  
 prev\_x1 = int(input("Введите начальное приближение x1-0: "))  
 prev\_x2 = int(input("Введите начальное приближение x2-0: "))  
 result = SystemSimpleIterations.systemSimpleIterations(prev\_x1, prev\_x2, equation, e)  
 print(f"Количество итераций: {result[0]}\nx1: {round(result[1], 5)}\nx2: {round(result[2], 5)}")  
 graph(-20.0, 20.0, 200, equation, True)

from Equations import calculate\_fk, derivative1, derivative2  
  
  
def newton(a, b, equ, eps):  
 if calculate\_fk(equ, a) \* derivative2(equ, a) > 0:  
 prev\_x = a  
 elif calculate\_fk(equ, b) \* derivative2(equ, b) > 0:  
 prev\_x = b  
 else:  
 raise Exception("Не выполняется условие сходимости.")  
 counter = 0  
 while True:  
 x = prev\_x - calculate\_fk(equ, prev\_x) / derivative1(equ, prev\_x)  
 counter += 1  
 if abs(x - prev\_x) <= eps:  
 break  
 prev\_x = x  
 return [counter, x]

**from** Equations **import** calculate\_fk, derivative1

**def** simpleIterations(a, b, equ, eps):

lm = -1 / max(derivative1(equ, a), derivative1(equ, b))

# Условие сходимости

**if** 1 + lm \* derivative1(equ, a) > 1 **or** 1 + lm \* derivative1(equ, b) > 1:

**raise** Exception("Условие сходимости не выполняется.")

prev\_x = a

counter = 0

**while** **True**:

x = prev\_x + lm \* calculate\_fk(equ, prev\_x)

counter += 1

**if** abs(x - prev\_x) < eps:

**break**

prev\_x = x

**return** [counter, x]

**from** Equations **import** derivative\_sys, graph, calculate\_fx\_sys

**def** systemSimpleIterations(x1\_0, x2\_0, equ, eps):

# Условие сходимости

**if** **not** (round(max(derivative\_sys(equ, 1, 1)), 1) <= 0.6 < 1):

**raise** Exception("Условие сходимости не выполняется.")

prev\_x\_1, prev\_x\_2 = x1\_0, x2\_0

counter = 0

**while** **True**:

counter += 1

x1, x2 = calculate\_fx\_sys(equ, prev\_x\_1, prev\_x\_2)

**if** abs(x1 - prev\_x\_1) **and** abs(x2 - prev\_x\_2) < eps:

**break**

prev\_x\_1, prev\_x\_2 = x1, x2

**return** [counter, x1, x2]

ВЫВОД

В ходе выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений и систем нелинейных уравнений с использованием Python. Была успешно реализована программа, предусматривающая выбор уравнений, методов решения, ввод исходных данных, проверку корректности данных и сходимости методов, а также вывод результатов на экран или в файл. В результате работы были найдены корни заданных уравнений и систем с использованием различных численных методов, а также были построены графики функций для полного представления исследуемых интервалов.