

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Факультет ПИ и КТ

Лабораторная работа №2

по дисциплине: «Вычислительная математика»

«Численное решение нелинейных уравнений и систем нелинейных уравнений»

Вариант 1

Выполнил:

**Болорболд Аригуун**,

группа P3211

Преподаватель:

**Малышева Татьяна Алексеевна**

Санкт-Петербург

2024



1. **Цель работы:**

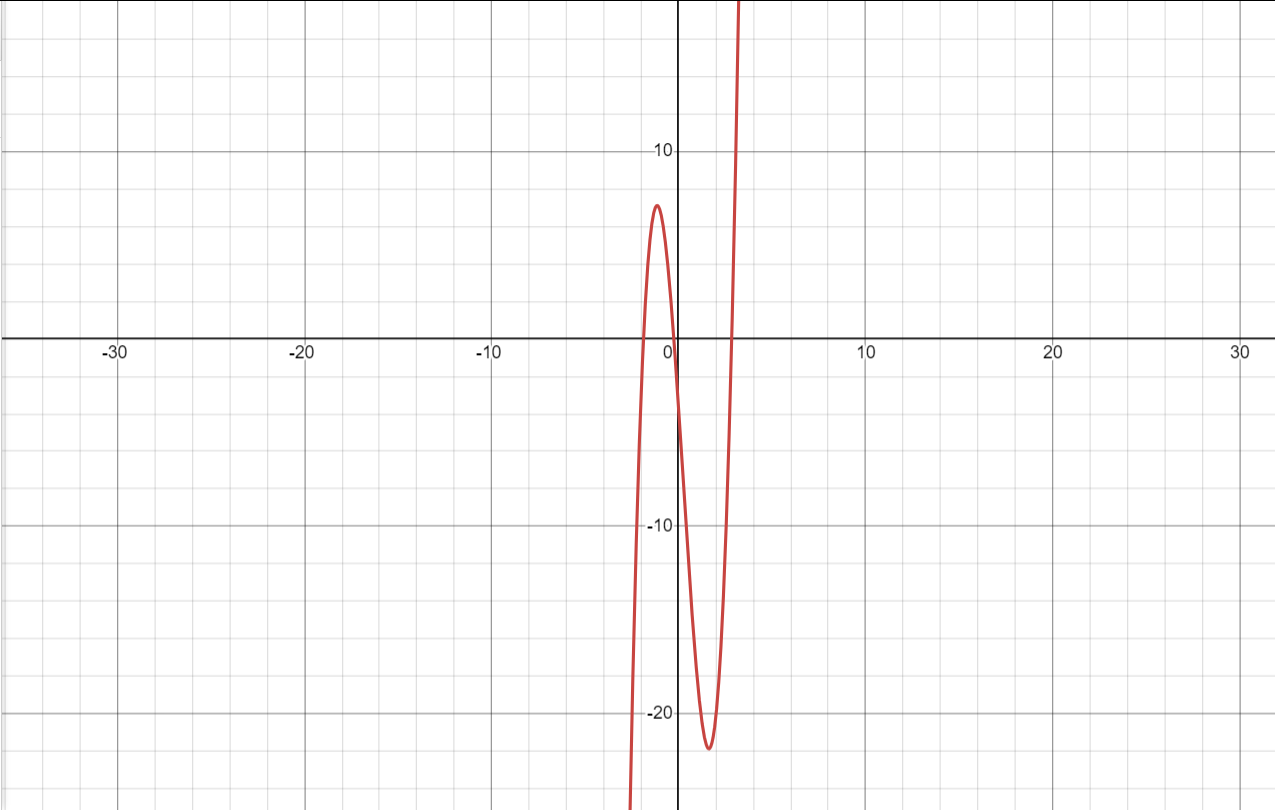
Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

1. **Задание:**
2. Отделить корни заданного нелинейного уравнения графически (вид уравнения представлен в табл. 2.6);
3. График исследуемой функции отобразить в отчете;
4. Определить интервалы изоляции корней;
5. Уточнить корни заданного нелинейного уравнения с точностью ;
6. Используемые методы для уточнения каждого из трех корней многочлена представлены в табл. 2.7;
7. Вычисления оформить в виде таблиц (табл. 2.1–2.5), в зависимости от заданного метода. Для всех значений в таблицах удержать 3 знака после запятой;
8. Для метода половинного деления заполнить таблицу 2.1;
9. Для метода хорд заполнить таблицу 2.2;
10. Для метода Ньютона заполнить таблицу 2.3;
11. Для метода секущих заполнить таблицу 2.4;
12. Для метода простой итерации заполнить таблицу 2.5;

**Условие задания (вариант 1): Решение нелинейного уравнения**

Вид уравнения:

1. Графическое отделение корней:



1. Интервалы изоляции корней:

Для левого корня:

Для центрального корня:

Для правого корня:

1. Уточнение корней с точностью :

Левый корень = 4 (метод секущих);

Центральный корень = 5 (метод простой итерации);

Правый корень: 3 (метод Ньютона);

1. **Описание метода, расчётные формулы:**

Уточнение правого корня (метод Ньютона):

Производные сохраняют знак на интервале изоляции, поэтому метод Ньютона эффективен.

Начальное приближение:

Знаки функции и второй производной совпадают, поэтому это подходящее начальное приближение.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |  |
| 0 | 2,9 | 2,5626 | 42,656 | 2,9 – (2,5626/42,656) =  2,84 | 0,06 |
| 1 | 2,84 | 0,0815 | 40,057 | 2,84 – (0,0815/40,057) =  **2,83796** | 0,00204 < |
| 2 | 2,83796 | -0,00016 | 39,969 | 2,83796 – (0,00016/39,969) = 2,837956 | 0,000004 |

Уточнение центрального корня (метод простой итерации):

На отрезке условие сходимости выполняется.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |
| 0 | -0,3 | -0,2597 | 0,07005 | 0,0403 |
| 1 | -0,2597 | -0,2551 | 0,00684 | 0,1473 |
| 2 | -0,2551 | 0,00611 | -3,813 | 0,02922 |
| 3 | 0,00612 | -0,00152 | -3,697 | 0,00765 |
| 4 | -0,00152 | 0.00039 | -3,7259 |  |
| 5 |  |  |  |  |

Уточнение левого корня (метод секущих):

;

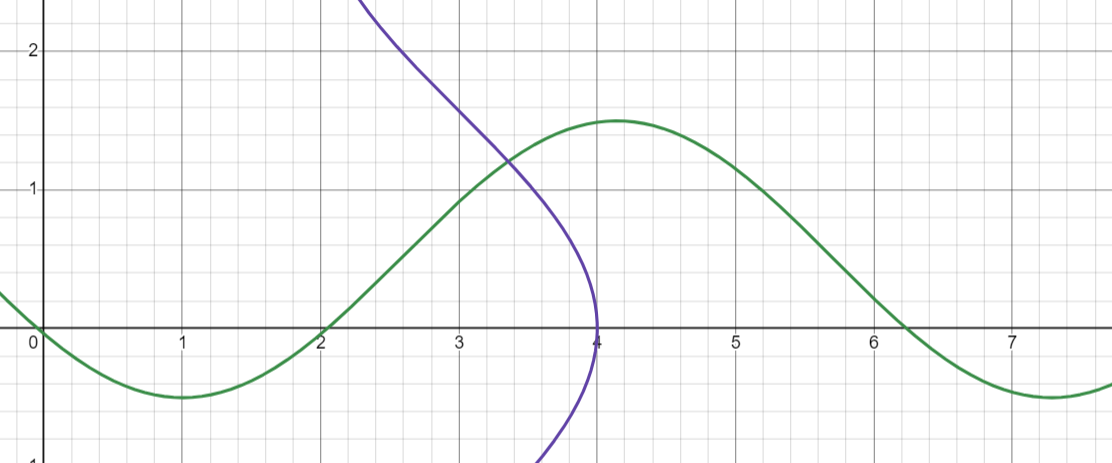
n = 3 (исходя из метода Ньютона)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № итерации |  |  |  |  |  |
| 0 | -2 | -1,8 | -1,8713 | 0,1603 | 0,0713 |
| 1 | -1,8 | -1,8713 | **-1,8795** | -0,01088 | 0,0082 < |
| 2 | -1,8713 | -1,8795 | -1,87897 | -1,4929 | 0,0482 |

**Решение системы нелинейных уравнении:**

Вид системы:

1. Графическое отделение корней:



Интервал изоляции x: [3,2; 3,6]

Интервал изоляции y: [1; 1,4]

1. Решение системы с точностью :

Начальные приближения x = 3,4; y = 1,2

1. **Исходный код (*Python*):**

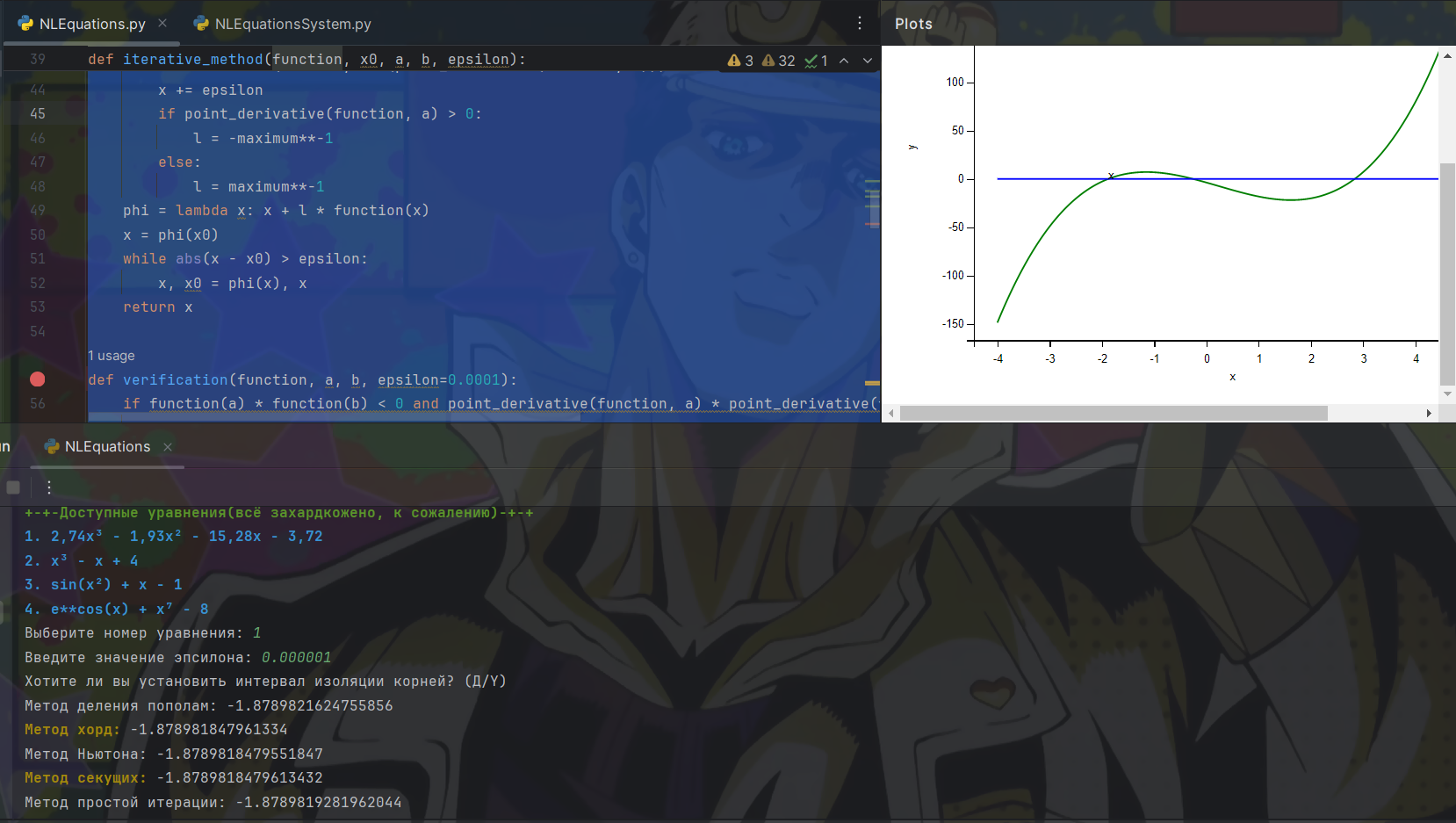
**4.1. Решение НУ:**

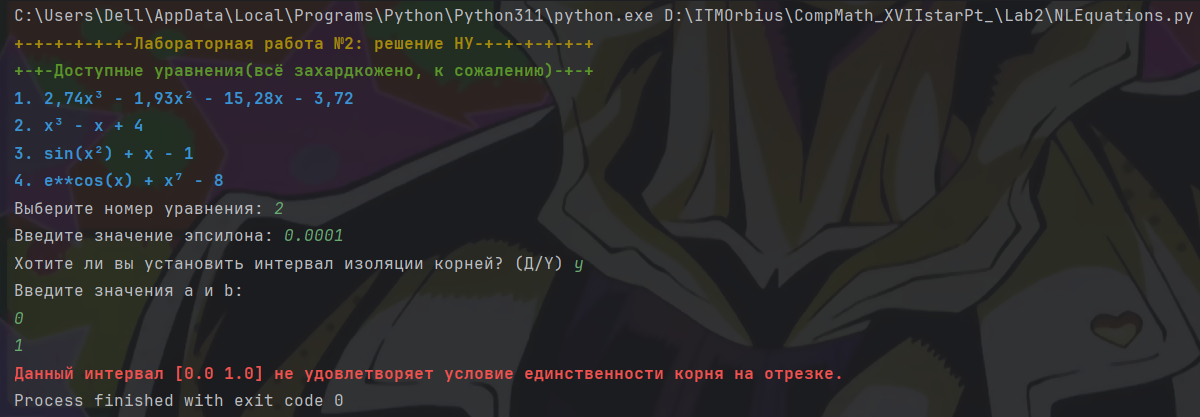
import matplotlib.pyplot as plt  
from math import e, sin, cos  
  
def bisection\_method(function, a, b, epsilon):  
 itr = 0  
 while abs(b - a) > epsilon:  
 mid = (a + b) / 2  
 if function(a) \* function(mid) > 0:  
 a = mid  
 else:  
 b = mid  
 if abs(function(mid)) < epsilon:  
 break  
 itr += 1  
 return (a + b) / 2, itr  
def chord\_method(function, a, b, epsilon):  
 iteration = 0  
 x = a - ((b - a) \* function(a)) / (function(b) - function(a))  
 while abs(function(x)) > epsilon:  
 iteration += 1  
 last\_x = x  
 if function(a) \* function(x) < 0:  
 b = x  
 else:  
 a = x  
 x = a - ((b - a) \* function(a)) / (function(b) - function(a))  
 if abs(x - last\_x) <= epsilon:  
 break  
 return x, iteration  
def newtons\_method(function, x0, epsilon):  
 itr = 0  
 x1 = x0 - function(x0) / point\_derivative(function, x0)  
 while abs(x0 - x1) > epsilon and abs(function(x1)) > epsilon:  
 x0 = x1  
 x1 = x0 - function(x0) / point\_derivative(function, x0)  
 itr += 1  
 return x1, itr  
def secant\_method(function, x0, epsilon):  
 itr = 0  
 x1 = x0 - function(x0) / point\_derivative(function, x0)  
 while abs(x1 - x0) > epsilon:  
 x2 = x1 - (x1 - x0) \* func(x1) / (func(x1) - func(x0))  
 x0, x1 = x1, x2  
 itr += 1  
 return x1, itr  
def iterative\_method(function, x0, a, b, epsilon):  
 itr = 0  
 maximum = 0  
 x = a  
 while x < b:  
 maximum = max(maximum, abs(point\_derivative(function, x)))  
 x += epsilon  
 if point\_derivative(function, a) > 0:  
 l = -maximum\*\*-1  
 else:  
 l = maximum\*\*-1  
 phi = lambda x: x + l \* function(x)  
 x = phi(x0)  
 while abs(x - x0) > epsilon or abs(function(x)) > epsilon:  
 x, x0 = phi(x), x  
 itr += 1  
 return x, itr  
  
def verification(function, a, b, epsilon=0.0001):  
 if function(a) \* function(b) < 0:  
 x = a  
 while x < b:  
 x += epsilon  
 return True  
 return False  
def graph(function, a, b, root, epsilon):  
 args = []  
 vals = []  
 x = a  
 while x < b:  
 args.append(x)  
 vals.append(function(x))  
 x += epsilon  
 plt.xlabel("x")  
 plt.ylabel("y")  
 plt.plot(args, vals, 'g')  
 plt.annotate("x", xy=(root[0], function(root[0])))  
 plt.plot([a,b], [0,0], 'b')  
 plt.show()  
def point\_derivative(function, x0, dx = 0.0001):  
 return (function(x0 + dx) - function(x0)) / dx  
# *TODO: добавить выбор начального приближения ради объективной корректности!*def second\_point\_derivative(function, x0, dx = 0.0001):  
 return  
print("\033[1;33m+-+-+-+-+-+-" + "Лабораторная работа №2: решение НУ" + "-+-+-+-+-+-+\033[0m")  
print("\033[1;32m+-+-" + "Доступные уравнения(всё захардкожено, к сожалению)" + "-+-+\033[0m")  
equations = ["1. 2,74x³ - 1,93x² - 15,28x - 3,72", "2. x³ - x + 4", "3. sin(x²) + x - 1", "4. e\*\*cos(x) + x⁷ - 8"]  
for equation in equations:  
 print("\033[1;34m" + equation + "\033[0m")  
case = input("Выберите номер уравнения: ")  
if case not in ['1', '2', '3', '4']:  
 input\_equation = input("Введите уравнение: ")  
else:  
 case = int(case)  
epsilon = float(input("Введите значение эпсилона: "))  
match case:  
 case 1:  
 func = lambda x: 2.74\*x\*\*3 - 1.93\*x\*\*2 - 15.28\*x - 3.72  
 intervals = [(-2, -1.8), (-0.3, -0.1), (2.7, 2.9)]  
 a, b = intervals[0]  
 case 2:  
 func = lambda x: x\*\*3 - x + 4  
 a, b = -2, -1  
 case 3:  
 func = lambda x: sin(x\*\*2) + x - 1  
 a, b = 0.6, 1  
 case 4:  
 func = lambda x: e \*\* cos(x) + x\*\*7 - 8  
 a, b = 1, 1.4  
  
fl = input('Хотите ли вы установить интервал изоляции корней? (Д/Y) ')  
if fl.casefold() == 'y' or fl.casefold() == 'д':  
 while 1:  
 print('Введите значения a и b:')  
 try:  
 a, b = float(input()), float(input())  
 except Exception:  
 continue  
 break  
  
x0 = (a + b) / 2  
if not verification(func, a, b):  
 print("\033[1;31mДанный интервал [" + str(a) + " " + str(b) + "] не удовлетворяет условие единственности корня на отрезке.", end="")  
 exit(0)  
if epsilon <= 0.00001:  
 places = int(str(epsilon)[-1:]) + 2  
else:  
 places = str(epsilon)[::-1].find(".") + 2  
print("Метод деления пополам:", round(bisection\_method(func, a, b, epsilon)[0], places), func(bisection\_method(func, a, b, epsilon)[0]), bisection\_method(func, a, b, epsilon)[1])  
print("\033[1;33mМетод хорд:\033[0m", round(chord\_method(func, a, b, epsilon)[0], places), func(chord\_method(func, a, b, epsilon)[0]), chord\_method(func, a, b, epsilon)[1])  
print("Метод Ньютона:", round(newtons\_method(func, x0, epsilon)[0], places), func(newtons\_method(func, x0, epsilon)[0]), newtons\_method(func, x0, epsilon)[1])  
print("\033[1;33mМетод секущих:\033[0m", round(secant\_method(func, x0, epsilon)[0], places), func(secant\_method(func, x0, epsilon)[0]), secant\_method(func, x0, epsilon)[1])  
print("Метод простой итерации:", round(iterative\_method(func, x0, a, b, epsilon)[0], places), func(iterative\_method(func, x0, a, b, epsilon)[0]), iterative\_method(func, x0, a, b, epsilon)[1])  
root = bisection\_method(func, a, b, epsilon)  
graph(func, -4, 5, root, 0.0001)

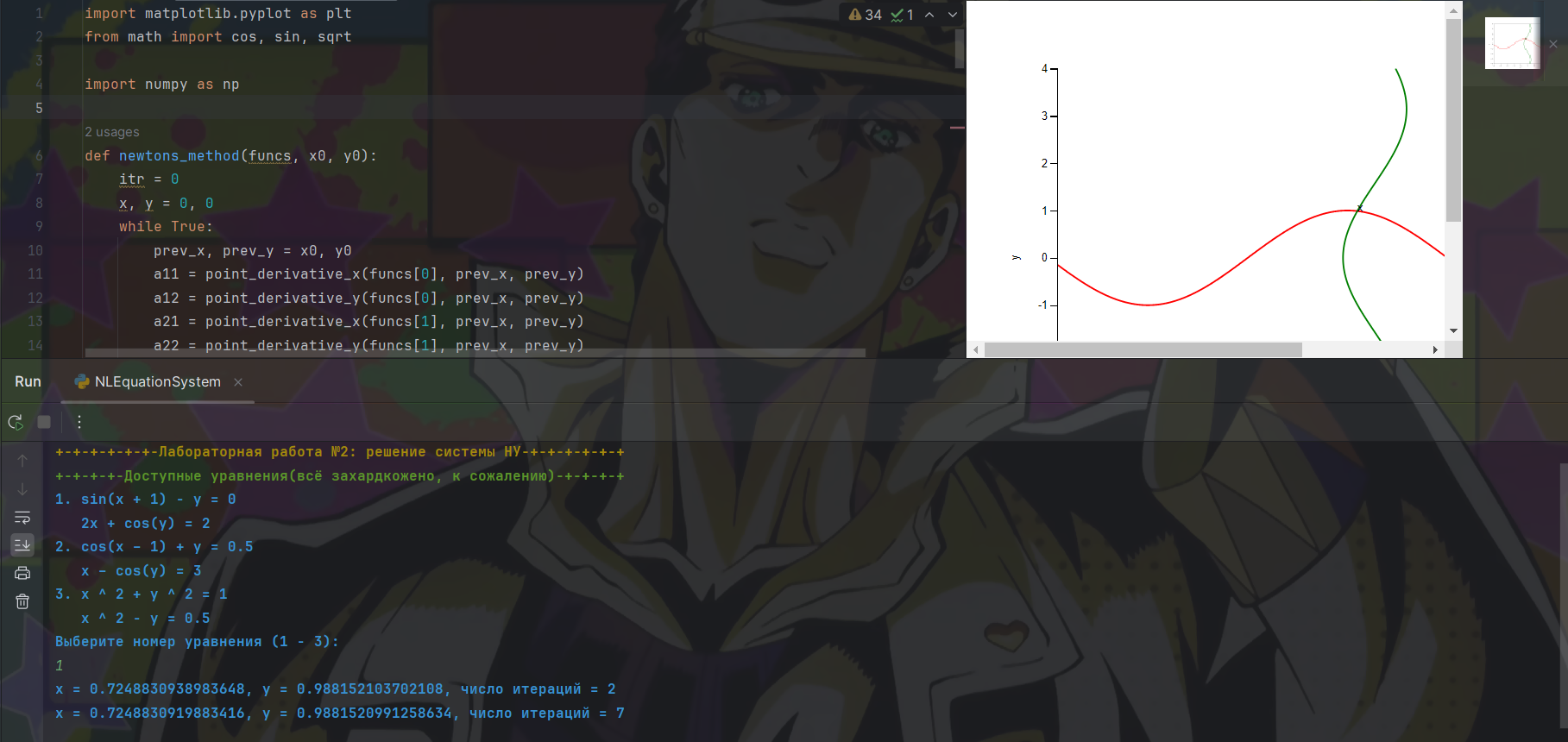
**4.2. Решение системы НУ:**

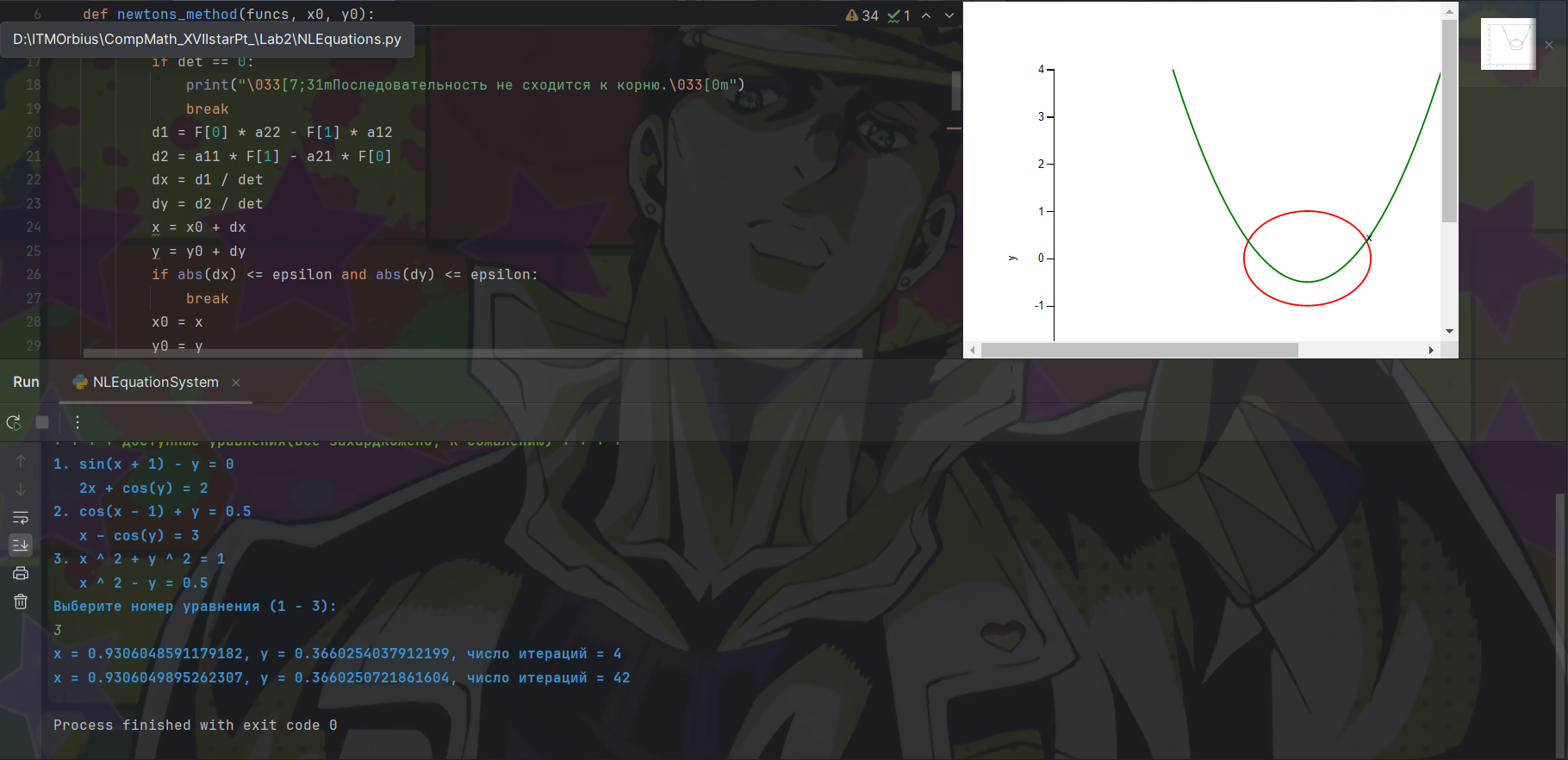
import matplotlib.pyplot as plt  
from math import cos, sin, sqrt  
  
import numpy as np  
  
def newtons\_method(funcs, x0, y0):  
 itr = 0  
 x, y = 0, 0  
 while True:  
 prev\_x, prev\_y = x0, y0  
 a11 = point\_derivative\_x(funcs[0], prev\_x, prev\_y)  
 a12 = point\_derivative\_y(funcs[0], prev\_x, prev\_y)  
 a21 = point\_derivative\_x(funcs[1], prev\_x, prev\_y)  
 a22 = point\_derivative\_y(funcs[1], prev\_x, prev\_y)  
 F = [-funcs[0](prev\_x, prev\_y), -funcs[1](prev\_x, prev\_y)]  
 det = a11 \* a22 - a12 \* a21  
 if det == 0:  
 print("\033[7;31mПоследовательность не сходится к корню.\033[0m")  
 break  
 d1 = F[0] \* a22 - F[1] \* a12  
 d2 = a11 \* F[1] - a21 \* F[0]  
 dx = d1 / det  
 dy = d2 / det  
 x = x0 + dx  
 y = y0 + dy  
 if abs(dx) <= epsilon and abs(dy) <= epsilon:  
 break  
 x0 = x  
 y0 = y  
 itr += 1  
 if itr > 300:  
 print("\033[7;31mНе удалось добиться нужной точности за вменяемое количество итераций.\033[0m")  
 break  
 return x, y, itr  
# *TODO: проверка частных производных*# *TODO: анализ функции*def iterative\_method(func1, func2, x01, x02, eps, max\_itr = 1000):  
 x1 = func1(x01)  
 x2 = func2(x02)  
 itr = 0  
 while abs(x1 - x01) > eps or abs(x2 - x02) > eps:  
 x2, x02 = func1(x1), x2  
 x1, x01 = func2(x2), x1  
 itr += 1  
 if itr >= max\_itr:  
 print("\033[7;31mНе удалось вычислить корень системы в пределе 1000 итерации.\033[0m")  
 break  
 return x1, x2, itr  
  
def point\_derivative\_x(func, x0, y0, dx=0.001):  
 return (func(x0 + dx, y0) - func(x0, y0)) / dx  
  
def point\_derivative\_y(func, x0, y0, dy=0.001):  
 return (func(x0, y0 + dy) - func(x0, y0)) / dy  
  
def plot\_system(funcs, x1, y1):  
 x = np.linspace(-4, 4, 1600)  
 y = np.linspace(-4, 4, 1600)  
 X, Y = np.meshgrid(x, y)  
  
 Z1 = np.array([funcs[0](x\_, y\_) for x\_, y\_ in zip(np.ravel(X), np.ravel(Y))]).reshape(X.shape)  
 Z2 = np.array([funcs[1](x\_, y\_) for x\_, y\_ in zip(np.ravel(X), np.ravel(Y))]).reshape(X.shape)  
 plt.annotate('x', xy=(x1, y1))  
 plt.contour(X, Y, Z1, levels=[0], colors='r')  
 plt.contour(X, Y, Z2, levels=[0], colors='g')  
 plt.xlabel('x')  
 plt.ylabel('y')  
 plt.show()  
print("\033[1;33m+-+-+-+-+-+-" + "Лабораторная работа №2: решение системы НУ" + "-+-+-+-+-+-+\033[0m")  
print("\033[1;32m+-+-+-+-" + "Доступные уравнения(всё захардкожено, к сожалению)" + "-+-+-+-+\033[0m")  
print("\033[3;34m1. sin(x + 1) - y = 0\n 2x + cos(y) = 2\033[0m")  
print("\033[3;34m2. cos(x – 1) + y = 0.5\n x – cos(y) = 3\033[0m")  
print("\033[3;34m3. x ^ 2 + y ^ 2 = 1\n x ^ 2 - y = 0.5\033[0m")  
print("Выберите номер уравнения (1 - 3): ")  
case\_number = input()  
while case\_number not in {'1', '2', '3'}:  
 print("\033[1;32mВыберите номер уравнения (1 - 3):\033[0m")  
 case\_number = input()  
  
case\_number = int(case\_number)  
if case\_number == 1:  
 func1 = lambda x, y: sin(x + 1) - y  
 func2 = lambda x, y: 2 \* x + cos(y) - 2  
 f1 = lambda x: sin(x + 1)  
 f2 = lambda y: 1 - cos(y) / 2  
 a1, b1 = 0.6, 0.8  
 a2, b2 = 0.8, 1  
elif case\_number == 2:  
 func1 = lambda x, y: cos(x - 1) + y - 0.5  
 func2 = lambda x, y: x - cos(y) - 3  
 f1 = lambda x: 0.5 - cos(x - 1)  
 f2 = lambda y: cos(y) + 3  
 a1, b1 = 10, 10  
 a2, b2 = 15, 15  
else:  
 func1 = lambda x, y: x \*\* 2 + y \*\* 2 - 1  
 func2 = lambda x, y: x \*\* 2 - y - 0.5  
 f2 = lambda y: sqrt(1 - y \*\* 2)  
 f1 = lambda x: x \*\* 2 - 0.5  
 a1, b1 = 5, 15  
 a2, b2 = 100, 100  
funcs = [func1, func2]  
x01 = (a1 + b1) / 2  
x02 = (a2 + b2) / 2  
epsilon = 0.000001  
if epsilon <= 0.00001:  
 places = int(str(epsilon)[-1:]) + 4  
else:  
 places = str(epsilon)[::-1].find(".") + 4  
x, y, itr = newtons\_method(funcs, x01, x02)  
print("Метод Ньютона: ")  
print(f"x = {round(x, places)}, y = {round(y, places)}, число итераций = {itr}")  
print(func1(x, y), func2(x, y))  
root = newtons\_method(funcs, x01, x02)[0]  
x, y, itr = iterative\_method(f1, f2, x01, x02, epsilon)  
print("\033[1;33mМетод простых итераций:\033[0m")  
print(f"x = {round(x, places)}, y = {round(y, places)}, число итераций = {itr}")  
print(func1(x, y), func2(x, y))  
plot\_system(funcs, x, f1(x))

1. **Примеры работы программ:**









1. **Вывод:**

В рамках этой работы по вычислительной математике я научился решать трансцендентные уравнения и системы из них, имплементировав их в программной среде. Несмотря на то, что мне надо было реализовать конкретные методы, я реализовал их всех чтобы окончательно подкрепить свой уровень знания. Самая болезненная работа, но этого я всё-таки смог преодолеть.