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

Факультет программной инженерии и компьютерной техники

Вычислительная математика

Лабораторная работа №2

Вариант 3

Группа: Р3269

Выполнили:

Грибкова В.Е

Долганова О.А

Проверил:

Машина Е. А.

Г. Санкт-Петербург

1. Цель лабораторной работы: изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.
2. Рабочие формулы используемых методов.

$$\varepsilon = 10^{-2}$$

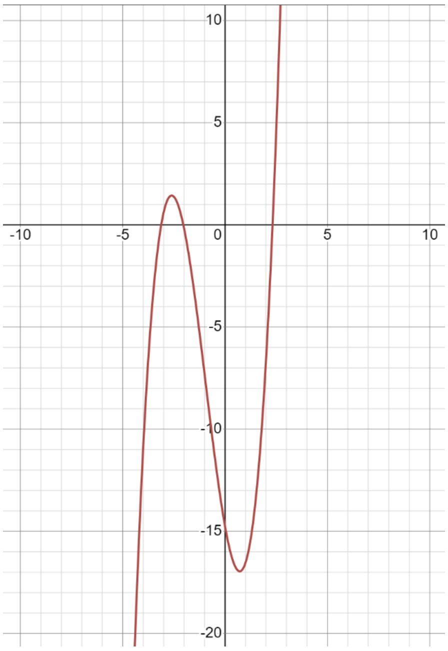
Метод половинного деления: $x_i = \frac{a_i + b_i}{2}$

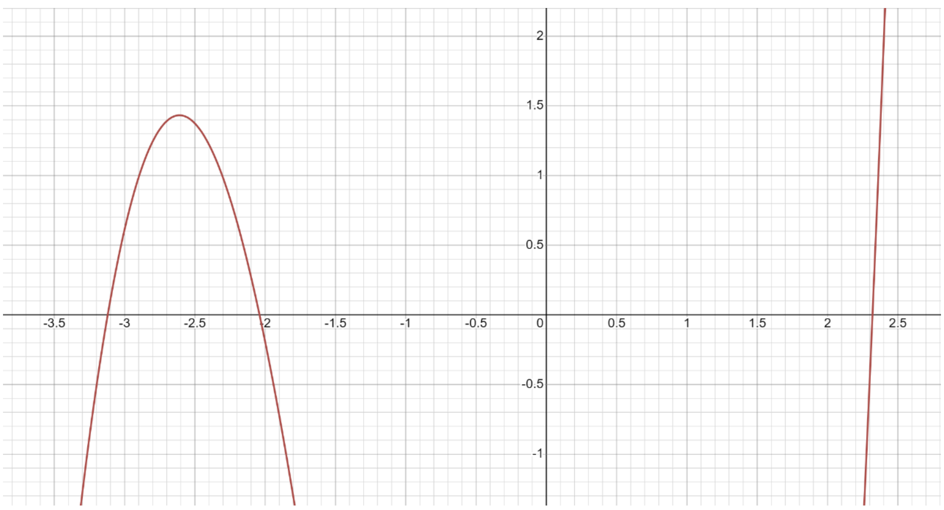
Метод Ньютона: $x_i = x_{i-1} = \frac{f(x_{i-1})}{f'(x_{i-1})}$

Метод простой итерации: $x_{i+1} = \varphi(x_i)$

3. Графики функций на исследуемом интервале.

$$x^3 + 2{,}84x^2 - 5{,}606x - 14{,}766$$





4. Заполненные таблицы вычислительной части 1 лабораторной работы

$$x^3 + 2{,}84x^2 - 5{,}606x - 14{,}766$$

Таблица 1

Уточнение корня уравнения методом половинного деления
для **крайнего правого** корня:

№ шага	a	b	x	f(a)	f(b)	f(x)	|a-b|
1	2,300	2,400	2,350	-0,469	1,962	0,722	0,100
2	2,300	2,350	2,325	-0,469	0,722	0,120	0,050
3	2,300	2,325	2,313	-0,469	0,120	-0,164	0,025
4	2,313	2,325	2,319	-0,164	0,120	-0,022	0,012
5	2,319	2,325	2,322	-0,022	0,120	0,049	0,006
6	2,319	2,322	2,321	-0,022	0,049	0,025	0,003
7	2,319	2,321	2,320	-0,022	0,025	0,001	0,002
8	2,319	2,320	2,320	-0,022	0,001	0,001	0,001

Таблица 2

Уточнение корня уравнения методом Ньютона
для **центрального** корня:

№ итерации	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}	$\lvert x_{k+1} - x_k \rvert$
1	-2,050	0,046	-4,643	-2,040	0,010

Уточнение корня уравнения методом простой итерации для **крайнего левого** корня:

$$x^3 + 2{,}84x^2 - 5{,}606x - 14{,}766$$

1. преобразуем уравнение $f(x) = 0$ к равносильному (при $\lambda \neq 0$) $\lambda f(x) = 0$
2. прибавим x в обеих частях: $x = x + \lambda f(x)$
3. $\varphi(x) = x + \lambda f(x)$, $\varphi'(x) = 1 + \lambda f'(x)$
4. высокая скорость сходимости обеспечивается при $q = \max\limits_{[a,b]}|\varphi'(x)| \approx 0$. Тогда

$\boldsymbol{\lambda} = -\frac{1}{\max\limits_{[a,b]}|f'(x)|}$

$f(x) = x^3 + 2,84x^2 - 5,606x - 14,766$

$f'(x) = 3x^2 + 5,68x - 5,606$

$f'(-3,2) = 6,938$

$f'(-3,1) = 5,616$

$\lambda = -0,144$

$x = x + \lambda f(x) = x + \lambda(x^3 + 2,84x^2 - 5,606x - 14,766)$

$\phi(x) = -0,144x^3 - 0,409x^2 + 1,807x + 2,126$

$\phi'(x) = -0,432x^2 - 0,818x + 1,807$

$\phi'(-3,2) = 0,001 < 1$-условие сходимости **выполняется**

$\phi'(-3,1) = 0,191 < 1$-условие сходимости **выполняется**

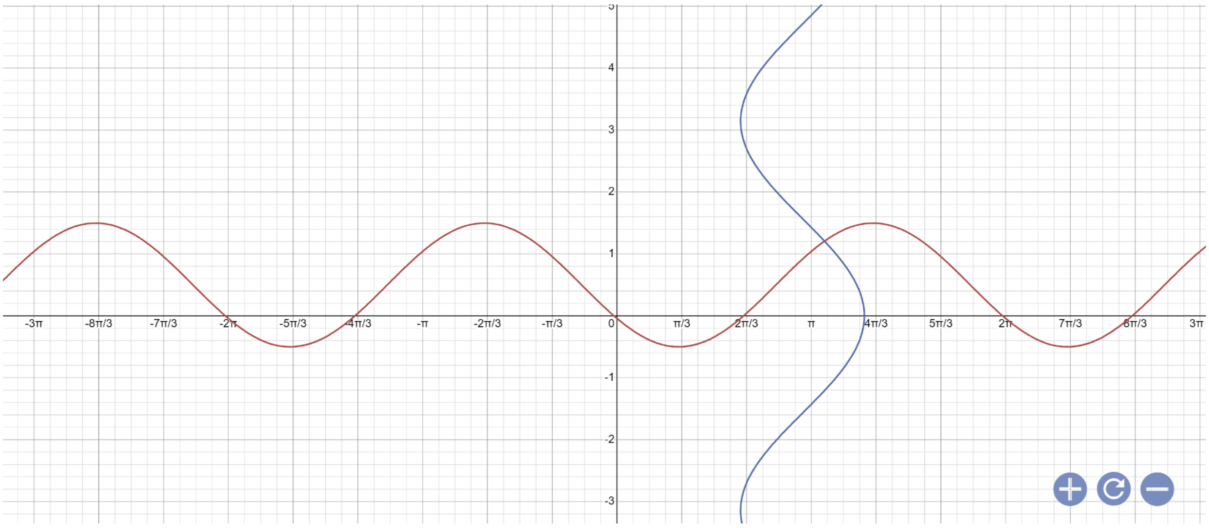
Таблица 3

№ итерации	x_k	x_{k+1}	$\lvert x_{k+1} - x_k \rvert$
1	-3,200	-3,126	0,074
2	-3,126	-3,121	0,005

Решение задачи: $x_{1=} - 3,126$ $x_{1=}2,050$ $x_{3=}2,320$

5. Решение системы нелинейных уравнений (вычислительная часть 2).

$$\begin{cases} cos(x-1) + y = 0,5 \\ x - cosy = 3 \end{cases}$$



1) приведем функцию к эквивалентному виду:

$x = \phi_1 = cosy + 3$

$y = \phi_2 = 0,5 - cos(x - 1)$

Определяем, что решение системы уравнений находится в квадрате:

$3,300 < x < 3,400,\ 1,200 < y < 1,300$

Проверим условие сходимости:

$\frac{d\phi_1}{dx} = 0 \qquad \frac{d\phi_1}{dy} = - siny$

$\frac{d\phi_2}{dy} = 0 \qquad \frac{d\phi_2}{dx} = sin(y - 1)$

$|\frac{d\phi_1}{dx}| + |\frac{d\phi_1}{dy}| = 0 + | - siny| \leq 0.964$

$|\frac{d\phi_2}{dx}| + |\frac{d\phi}{dy}| = 0 + |sin(y - 1)| \leq 0.296$

$max\ [x\in G]\ \varphi\,'\,(x) \leq 0.964 < 1$-процесс сходящийся

Выберем начальное приближение: $x^{(0)} = 3,400\ \ y^{(0)} = 1,300$

1 шаг:

$x^{(1)} = cos(1,300) + 3 = 3,268 \qquad |x^{(1)} - x^{(0)}| = 0,132 > \varepsilon$

$y^{(1)} = 0,5 - cos(3,400 - 1) = 1,237 \qquad |y^{(1)} - y^{(0)}| = 0,063 > \varepsilon$

2 шаг:

$x^{(2)} = cos(1,237) + 3 = 3,328 \qquad |x^{(2)} - x^{(1)}| = 0,060 > \varepsilon$

$y^{(2)} = 0,5 - cos(3,268 - 1) = 1,142 \qquad |y^{(2)} - y^{(1)}| = 0,095 > \varepsilon$

3 шаг:

$x^{(3)} = cos(1,142) + 3 = 3,415 \qquad |x^{(3)} - x^{(2)}| = 0,087 > \varepsilon$

$y^{(3)} = 0,5 - cos(3,328 - 1) = 1,187 \qquad |y^{(3)} - y^{(2)}| = 0,045 > \varepsilon$

4 шаг:

$x^{(4)} = cos(1,187) + 3 = 3,374 \qquad |x^{(4)} - x^{(3)}| = 0,041 > \varepsilon$

$y^{(4)} = 0,5 - cos(3,415 - 1) = 1,247 \qquad |y^{(4)} - y^{(3)}| = 0,060 > \varepsilon$

5 шаг:

$x^{(5)} = cos(1,247) + 3 = 3,318 \qquad |x^{(5)} - x^{(4)}| = 0,056 > \varepsilon$

$y^{(5)} = 0,5 - cos(3,374 - 1) = 1,220$ $\quad |y^{(5)} - y^{(4)}| = 0,027 > ε$

6 шаг:

$x^{(6)} = cos(1,220) + 3 = 3,344$ $\quad |x^{(6)} - x^{(5)}| = 0,026 > ε$

$y^{(6)} = 0,5 - cos(3,318 - 1) = 1,180$ $\quad |y^{(6)} - y^{(5)}| = 0,040 > ε$

7 шаг:

$x^{(7)} = cos(1,180) + 3 = 3,381$ $\quad |x^{(7)} - x^{(6)}| = 0,037 > ε$

$y^{(7)} = 0,5 - cos(3,344 - 1) = 1,198$ $\quad |y^{(7)} - y^{(6)}| = 0,018 > ε$

8 шаг:

$x^{(8)} = cos(1,198) + 3 = 3,364$ $\quad |x^{(8)} - x^{(7)}| = 0,017 > ε$

$y^{(8)} = 0,5 - cos(3,381 - 1) = 1,224$ $\quad |y^{(8)} - y^{(7)}| = 0,026 > ε$

9 шаг:

$x^{(9)} = cos(1,224) + 3 = 3,340$ $\quad |x^{(9)} - x^{(8)}| = 0,024 > ε$

$y^{(9)} = 0,5 - cos(3,364 - 1) = 1,213$ $\quad |y^{(9)} - y^{(8)}| = 0,011 > ε$

10 шаг:

$x^{(10)} = cos(1,213) + 3 = 3,350$ $\quad |x^{(10)} - x^{(9)}| = 0,010 > ε$

$y^{(10)} = 0,5 - cos(3,340 - 1) = 1,195$ $\quad |y^{(10)} - y^{(9)}| = 0,018 > ε$

11 шаг:

$x^{(11)} = cos(1,195) + 3 = 3,367$ $\quad |x^{(11)} - x^{(10)}| = 0,017 > ε$

$y^{(11)} = 0,5 - cos(3,350 - 1) = 1,203$ $\quad |y^{(11)} - y^{(10)}| = 0,008 > ε$

12 шаг:

$x^{(12)} = cos(1,203) + 3 = 3,360$ $\quad |x^{(12)} - x^{(11)}| = 0,007 > ε$

$y^{(12)} = 0,5 - cos(3,367 - 1) = 1,215$ $\quad |y^{(12)} - y^{(11)}| = 0,012 > ε$

13 шаг:

$x^{(13)} = cos(1,215) + 3 = 3,348$ $\quad |x^{(13)} - x^{(12)}| = 0,012 > ε$

$y^{(13)} = 0,5 - cos(3,360 - 1) = 1,210$ $\quad |y^{(13)} - y^{(12)}| = 0,005 > ε$

14 шаг:

$x^{(14)} = cos(1,210) + 3 = 3,353$ $\quad |x^{(14)} - x^{(13)}| = 0,006 < ε$

$y^{(14)} = 0,5 - cos(3,348 - 1) = 1,201$ $\quad |y^{(14)} - y^{(13)}| = 0,009 < ε$

Критерий окончания итерационного процесса выполнен.

Решение задачи: $x^{(14)} = 3,353 \;\; y^{(14)} = 1,201$

6. Листинг программы, по крайней мере, коды используемых методов.

Решение нелинейных уравнений

```
import matplotlib.pyplot as plt
from math import sin, e


# Функция для вычисления корня уравнения методом бисекции
```

```
def bisection_method(function, left_bound, right_bound,
tolerance):
   # Пока разница между границами больше допустимой погрешности
   while abs(left_bound - right_bound) > tolerance:
       # Находим середину интервала
       midpoint = (left_bound + right_bound) / 2
       # Проверка знака произведения значений функции на концах
интервала
       if function(left_bound) * function(midpoint) > 0:
           left_bound = midpoint  # Если знаки одинаковы, сдвигаем
левую границу
       else:
           right_bound = midpoint  # Если знаки разные, сдвигаем
правую границу
   return (left_bound + right_bound) / 2  # Возвращаем середину
интервала как корень


# Функция для вычисления корня уравнения методом хорд
def chord_method(function, left_bound, right_bound, tolerance):
   while abs(left_bound - right_bound) > tolerance:
       # Вычисляем точку пересечения хорды с осью x
       midpoint = (left_bound * function(right_bound) -
right_bound * function(left_bound)) / (
                  function(right_bound) - function(left_bound))
       if function(left_bound) * function(midpoint) > 0:
           left_bound = midpoint  # Если знаки одинаковы, сдвигаем
левую границу
       else:
           right_bound = midpoint  # Если знаки разные, сдвигаем
правую границу
   # Возвращаем точку пересечения хорды как корень
   return (left_bound * function(right_bound) - right_bound *
function(left_bound)) / (
               function(right_bound) - function(left_bound))


# Функция для вычисления корня уравнения методом секущих
def secant_method(function, initial_guess, tolerance):
   # Вычисляем первую приближенную точку
   next_guess = initial_guess - function(initial_guess) /
derivative_at_point(function, initial_guess)
   # Пока разница между текущим и предыдущим приближением больше
допустимой погрешности
   while abs(next_guess - initial_guess) > tolerance:
       # Вычисляем следующее приближение
       temp = next_guess - (next_guess - initial_guess) *
function(next_guess) / (
```

```
                   function(next_guess) - function(initial_guess))
       initial_guess, next_guess = next_guess, temp  # Обновляем
приближения
   return next_guess  # Возвращаем текущее приближение как корень


# Функция для вычисления корня уравнения методом простой итерации
def simple_iteration_method(function, initial_guess, left_bound,
right_bound, tolerance):
   max_derivative = 0
   current_x = left_bound
   # Поиск максимального значения производной функции на интервале
   while current_x < right_bound:
       max_derivative = max(max_derivative,
abs(derivative_at_point(function, current_x)))
       current_x += tolerance

   # Выбор шага итерации в зависимости от знака производной
   step = -1 / max_derivative if derivative_at_point(function,
left_bound) > 0 else 1 / max_derivative
   # Определение итерационной функции
   iteration_function = lambda x: x + step * function(x)

   current_x = iteration_function(initial_guess)
   # Пока разница между текущим и предыдущим приближением больше
допустимой погрешности
   while abs(current_x - initial_guess) > tolerance:
       current_x, initial_guess = iteration_function(current_x),
current_x  # Обновляем приближения
   return current_x  # Возвращаем текущее приближение как корень


# Функция для проверки существования единственного корня на
интервале
def verify(function, left_bound, right_bound, tolerance=0.00001):
   # Проверка условий для существования единственного корня на
интервале
   if function(left_bound) * function(right_bound) < 0 and
derivative_at_point(function, left_bound) *
derivative_at_point(function, right_bound) > 0:
       current_x = left_bound
       # Проверка знака производной функции на интервале
       while current_x < right_bound:
           if derivative_at_point(function, left_bound) *
derivative_at_point(function, current_x) <= 0:
               return False  # Если производная меняет знак,
возвращаем False
           current_x += tolerance
```

```
        return True  # Если производная не меняет знак, возвращаем
True
    return False  # Если начальные условия не выполнены, возвращаем
False


# Функция для вычисления производной функции в точке
def derivative_at_point(function, point, dx=0.000001):
    return (function(point + dx) - function(point)) / dx  #
Используем метод конечных разностей


# Функция для построения графика функции и её корня
def draw_plot(function, left_bound, right_bound, root, step):
    x_values = []
    y_values = []
    current_x = left_bound
    # Генерация значений функции на заданном интервале
    while current_x < right_bound:
        x_values.append(current_x)
        y_values.append(function(current_x))
        current_x += step
    plt.xlabel("X")  # Подпись оси X
    plt.ylabel("Y")  # Подпись оси Y
    plt.plot(x_values, y_values, 'g')  # Построение графика функции
    plt.annotate('x', xy=(root, function(root)))  # Аннотирование
корня на графике
    plt.plot([left_bound, right_bound], [0, 0], 'b')  # Построение
оси X
    plt.show()  # Отображение графика


# Вывод доступных функций для выбора
print('Варианты функций:')
print('1. x ** 3 + 2.84 * x ** 2 - 5.606 * x - 14.766')
print('2. x ** 3 - x + 4')
print('3. sin(x ** 2) + x + 2')
print('4. x ** 3 - 0.1 * x ** 2 + 0.5')
print('Введите номер функции (1 или 2 или 3 или 4): ')
case_number = input()
while case_number not in {'1', '2', '3', '4'}:
    print('Введите номер функции (1 или 2 или 3 или 4):')
    case_number = input()

case_number = int(case_number)
tolerance = 0.0001  # Задаем значение допустимой погрешности
if case_number == 1:
```

```
    function = lambda x: x ** 3 + 2.84 * x ** 2 - 5.606 * x -
14.766
    isolation_intervals = [(-3.2, -3), (-2.2, -2), (2.2, 2.4)]
    left_bound, right_bound = isolation_intervals[2]
elif case_number == 2:
    function = lambda x: x ** 3 - x + 4
    left_bound, right_bound = -2, -1
elif case_number == 3:
    function = lambda x: sin(x ** 2) + x + 2
    left_bound, right_bound = -2, -1.6
else:
    function = lambda x: x ** 3 - 0.1 * x ** 2 + 0.5
    left_bound, right_bound = 0.6, 1

user_input = input('Введите "Y" если хотите задать [a; b]: ')
if user_input == 'Y':
    while True:
        print('Введите a и b через пробел:')
        try:
            left_bound, right_bound = map(float, input().split())
# Ввод значений интервала
        except Exception:
            continue
        break

initial_guess = (left_bound + right_bound) / 2  # Начальное
приближение как середина интервала
print("Проверка:", verify(function, left_bound, right_bound))  #
Проверка существования единственного корня
if not verify(function, left_bound, right_bound):
    exit(0)  # Выход, если корень не единственный
print("Метод бисекции:", bisection_method(function, left_bound,
right_bound, tolerance))
print("Метод секущих:", secant_method(function, initial_guess,
tolerance))
print("Метод простой итерации:", simple_iteration_method(function,
initial_guess, left_bound, right_bound, tolerance))
root = bisection_method(function, left_bound, right_bound,
tolerance)  # Вычисление корня методом бисекции
draw_plot(function, -4, 3, root, 0.001)  # Построение графика
функции и корня
```

Для систем линейных уравнений методом Ньютона

```
import sympy
from sympy import diff
import numpy
import matplotlib.pyplot as plt
```

```
while True:
   try:
       print("Выберите систему уравнений (1 или 2): ")
       print("1. 2x^2-y=4")
       print("    4x-y=1")
       print("2. x^3-1y=5")
       print("    2x+5y=1")
       num = int(input())
   except ValueError:
       print("Пожалуйста, введите число")
       continue
   if num != 1 and num != 2:
       print('Пожалуйста, введите 1 или 2')
       continue
   else:
       break

print("Введите начальное приближение для x и y: ")
pr = input().split(" ")
if len(pr) != 2:
   print("Вы ввели не два числа")
   exit(0)
x_0 = pr[0]
y_0 = pr[1]

def fun(x, y, num):
   if num == 1:
       f1 = 5 * x ** 2 - 3 * y - 4
       f2 = 7 * x - y - 1
   else:
       f1 = x ** 3 - 3 * y - 5
       f2 = 2 * x + y - 1
   return [f1, f2]


def Newton(x_0, y_0, e):
   x_0 = float(x_0)
   y_0 = float(y_0)
   x = sympy.symbols('x')
   y = sympy.symbols('y')
   m = fun(x, y, num)
   f_x = diff(m[0], x).subs(x, x_0).subs(y, y_0)
   f_y = diff(m[0], y).subs(x, x_0).subs(y, y_0)
   g_x = diff(m[1], x).subs(x, x_0).subs(y, y_0)
   g_y = diff(m[1], y).subs(x, x_0).subs(y, y_0)
   j = f_x * g_y - f_y * g_x
   f = m[0].subs(x, x_0).subs(y, y_0)
```

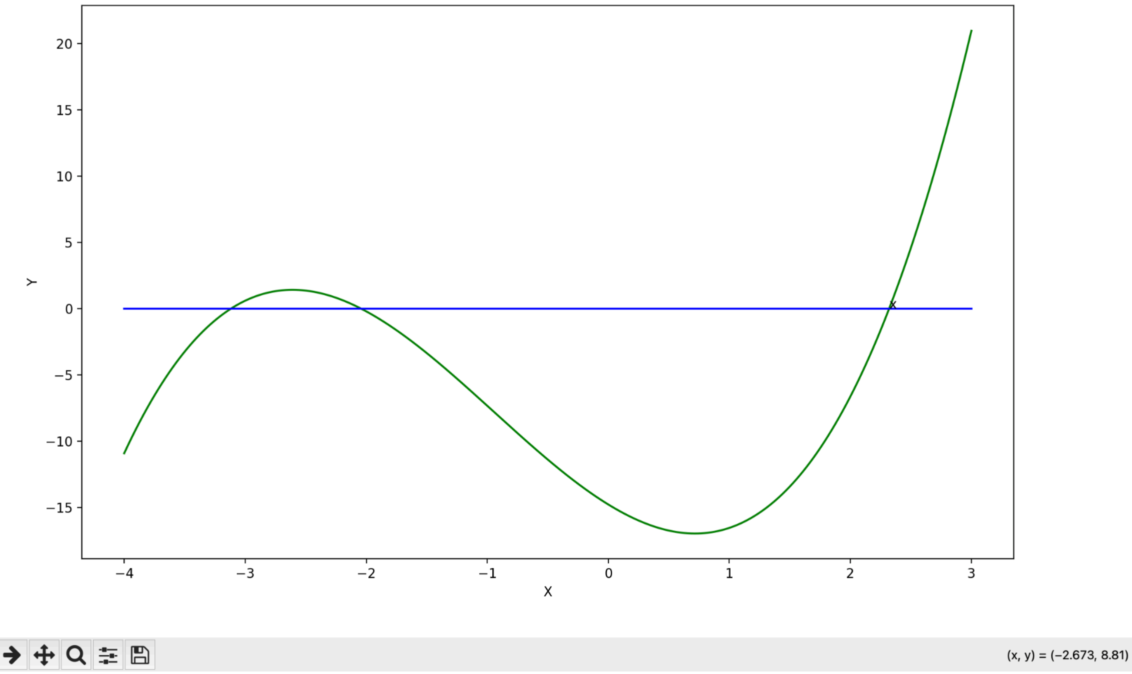
```
    g = m[1].subs(x, x_0).subs(y, y_0)
    x = float(x_0 - (f * g_y - f_y * g) / j)
    y = float(y_0 - (f_x * g - f * g_x) / j)
    it = 1
    while abs(x - x_0) > e or abs(y - y_0) > e:
        x_0 = x
        y_0 = y
        x = sympy.symbols('x')
        y = sympy.symbols('y')
        f_x = diff(m[0], x).subs(x, x_0).subs(y, y_0)
        f_y = diff(m[0], y).subs(x, x_0).subs(y, y_0)
        g_x = diff(m[1], x).subs(x, x_0).subs(y, y_0)
        g_y = diff(m[1], y).subs(x, x_0).subs(y, y_0)
        j = f_x * g_y - f_y * g_x
        f = m[0].subs(x, x_0).subs(y, y_0)
        g = m[1].subs(x, x_0).subs(y, y_0)
        x = x_0 - float(f * g_y - f_y * g) / j
        y = y_0 - float(f_x * g - f * g_x) / j
        it += 1
    print("Метод Ньютона дал следующий результат: x=", x, "y=", y)
    print("Количество итераций равно ", it)
    print("Погрешность равна ", x - x_0)
    return 1


Newton(x_0, y_0, 0.01)

x = numpy.arange(-10, 10, 0.01)
y = numpy.arange(-10, 10, 0.01)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.xlim(-5, 5)
plt.ylim(-10, 35)
if num == 1:
    plt.title('$5x^2-3y=4, 7x-y=1$')
    plt.plot(x, 5 * x ** 2 / 3 - 4 / 3, x, 7 * x - 1, [-0.047,
4.247], [-1.33, 28.73], 'b')
else:
    plt.title('$x^3-3y=5, 2x+y=1$')
    plt.plot(x, x ** 3 / 3 - 5 / 3, x, 1 - 2 * x, 1.107, -1.214,
'b')
plt.show()
```

7. Результаты выполнения программы при различных исходных данных.

Решение нелинейных уравнений

```
Варианты функций:
```

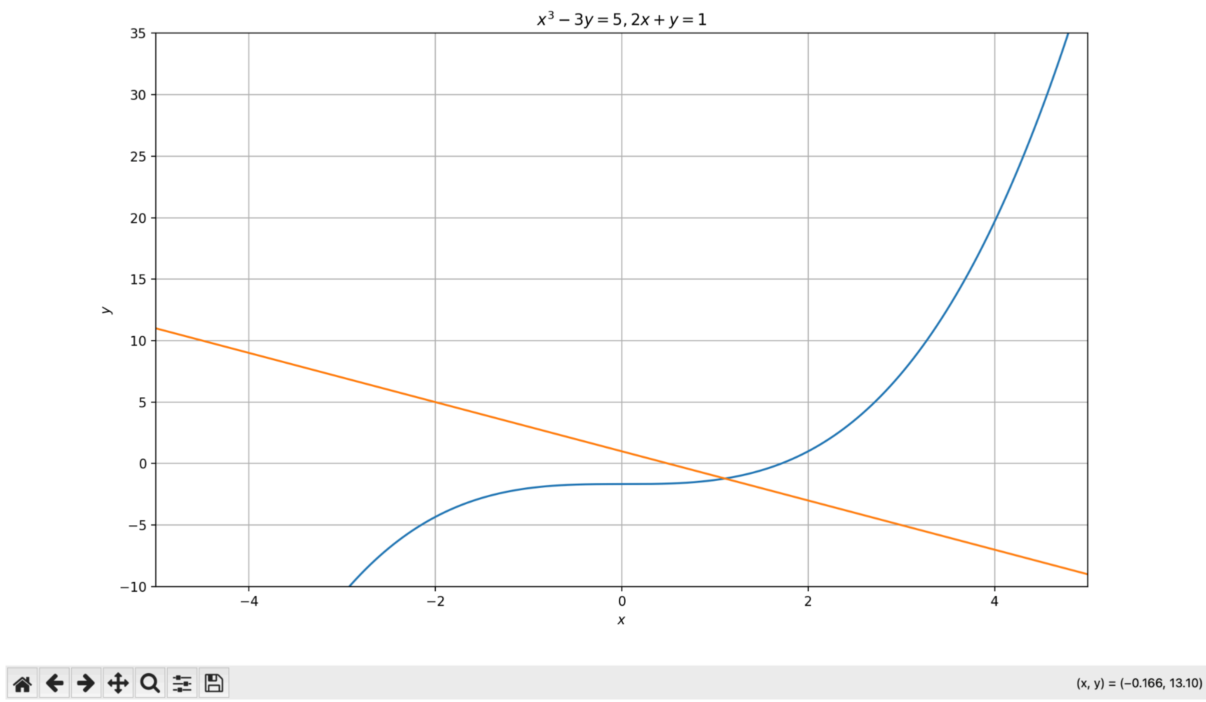
```
1. x ** 3 + 2.84 * x ** 2 - 5.606 * x - 14.766
2. x ** 3 - x + 4
3. sin(x ** 2) + x + 2
4. x ** 3 - 0.1 * x ** 2 + 0.5
Введите номер функции (1 или 2 или 3 или 4):
1
Введите "Y" если хотите задать [a; b]:
Проверка: True
Метод бисекции: 2.3199707031249996
Метод секущих: 2.3199467076713804
Метод простой итерации: 2.3199411387737556
```



Для систем линейных уравнений методом Ньютона

```
Выберите систему уравнений (1 или 2):
1. 2x^2-y=4
  4x-y=1
2. x^3-1y=5
  2x+5y=1
2
Введите начальное приближение для x и y:
2 3
Метод Ньютона дал следующий результат: x= 1.10714756783012 y=
-1.21429513566025
Количество итераций равно  4
```

```
Погрешность равна  -9.94507065588124e-5
```



8. Вывод: изучили численные методы решения нелинейных уравнений и их систем, нашли корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнили программную реализацию методов.