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

Факультет программной инженерии и компьютерных технологий

Вычислительная математика

Лабораторная работа 2

«Численное решение нелинейных уравнений и систем»

Вариант 13

Выполнила:

Павличенко Софья Алексеевна, P3215

Проверила:
Малышева Татьяна Алексеевна

Санкт-Петербург 2025г.

Оглавление

Цель

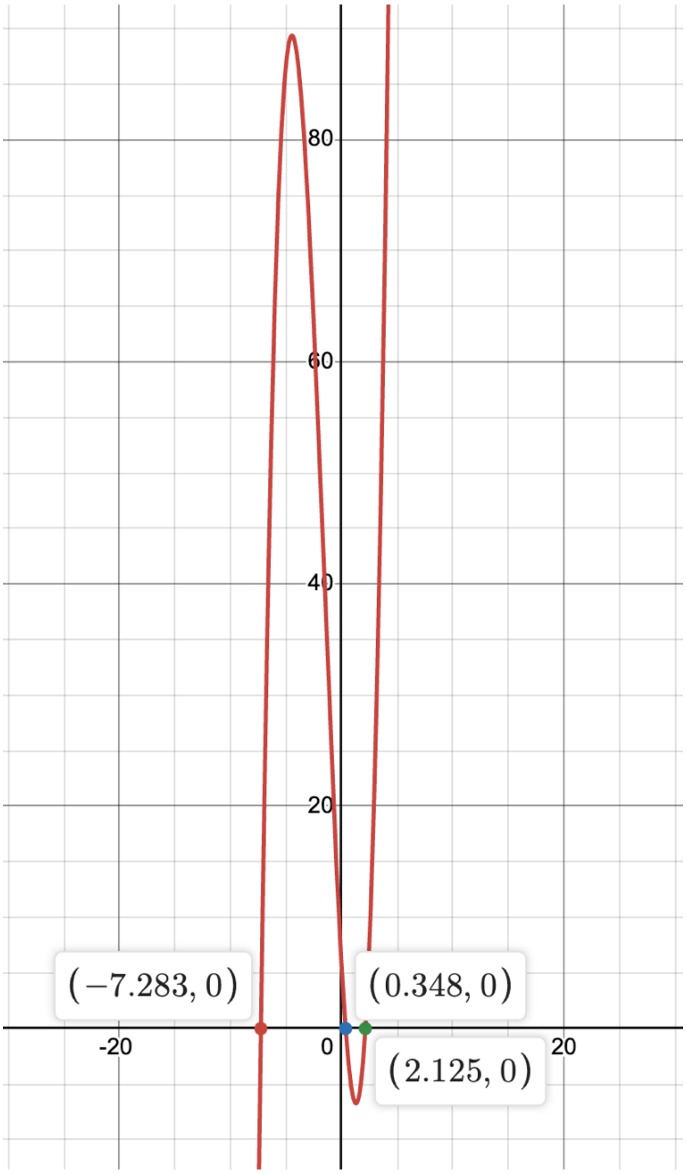
Изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

Часть 1: Вычислительная реализация задачи

Решение нелинейного уравнения

1.

$$f(x) = x^3 + 4.81x^2 - 17.27x + 5.38$$



Исходя из графика получаем три интервала изоляции корней уравнения:

$$(-8,-6), (-1, 1) \text{ и } (2, 3).$$

Уточним корни с использованием разных методов:

1. Крайний правый корень - **Метод простой итерации**

Проверка условия сходимости метода на выбранном интервале:

$f(x) = x^3 + 4.81x^2 - 17.27x + 5.38 = 0$

$f'(x) = 3x^2 + 9.62x - 17,27$

$f'(a) = 13.97 > 0, \qquad f'(b) = 38.59 > 0$

$max(|f'(a)|, |f'(b)|) = 38.59 \rightarrow \lambda = -\frac{1}{\max\limits_{[a,b]}|f'(x)|} = -\frac{1}{38.59}$

$\varphi(x) = x + \lambda f(x) = x - \frac{x^3 + 4.81x^2 - 17.27x + 5.38}{38.59}$

$\varphi'(x) = 1 + \lambda f'(x) = 1 - \frac{3x^2 + 9.62x - 17.27}{38.59}$

На отрезке начального приближения [2, 3] функция φ(x) определена, непрерывна и дифференцируема.

$$|\varphi'(a)| \approx 0.64$$

$$|\varphi'(b)| \approx 0$$

$$q = \underset{[a,b]}{max}|\varphi'(x)| \approx 0.64$$

$0 \leq q < 1 \Rightarrow$ итерационная последовательность **сходится**.

Критерий окончания итерационного процесса:

$$|x_n - x_{n-1}| < \frac{1 - 0.64}{0.64}\varepsilon \approx 0.56\varepsilon = 0.0056$$

$$x_0 = 2$$

№ итерации	x_k	x_{k+1}	$f(x_{k+1})$	$\|x_{k+1} - x_k\|$
1	2,000	2,050	-1,198	0,050
2	2,050	2,081	-0,720	0,031
3	2,081	2,099	-0,423	0,019
4	2,099	2,110	-0,244	0,011
5	2,110	2,117	-0,140	0,006
6	2,117	**2,120**	-0,080	0,004 < **0,0056**

2. Крайний левый корень - **Метод хорд**

$a_0 = -8,\ \ b_0 = -6$

$f(a_0) = -60.62,\ \ \ f(b_0) = 66.16$

$x_0 = a_0 - \frac{b_0 - a_0}{f(b_0) - f(a_0)} f(a_0) = -7.044$

Критерий окончания итерационного процесса:

$$|x_n - x_{n-1}| \le \varepsilon = 0.01$$

№ шага	a	b	x	$f(a)$	$f(b)$	$f(x)$	$\lvert x_{k+1} - x_k \rvert$
1	-8,000	-6,000	-7,044	-60,620	66,160	16,203	-0,956
2	-8,000	-7,044	-7,245	-60,620	16,203	2,660	0,202
3	-8,000	-7,245	-7,277	-60,620	2,660	0,406	0,032
4	-8,000	-7,277	**-7,282**	-60,620	0,406	0,061	0,005 < **0,01**

3. Центральный корень - **Метод Ньютона**

Проверка условия сходимости метода на выбранном интервале:

$f'(-1) = -23.89, \qquad f'(1) = -4.65$

$f''(x) = 6x + 9.62$

$f''(-1) = 3.62, \qquad f''(1) = 15.62$

$f'(x)$ и $f''(x)$ сохраняют знак на отрезке $[a;\ b]$ $\Rightarrow$ Условия сходимости **выполняются**.

$f(-1) = 26.46, \qquad f(1) = -6.08$

$f(-1) * f''(-1) > 0 \Rightarrow x_0 = -1$

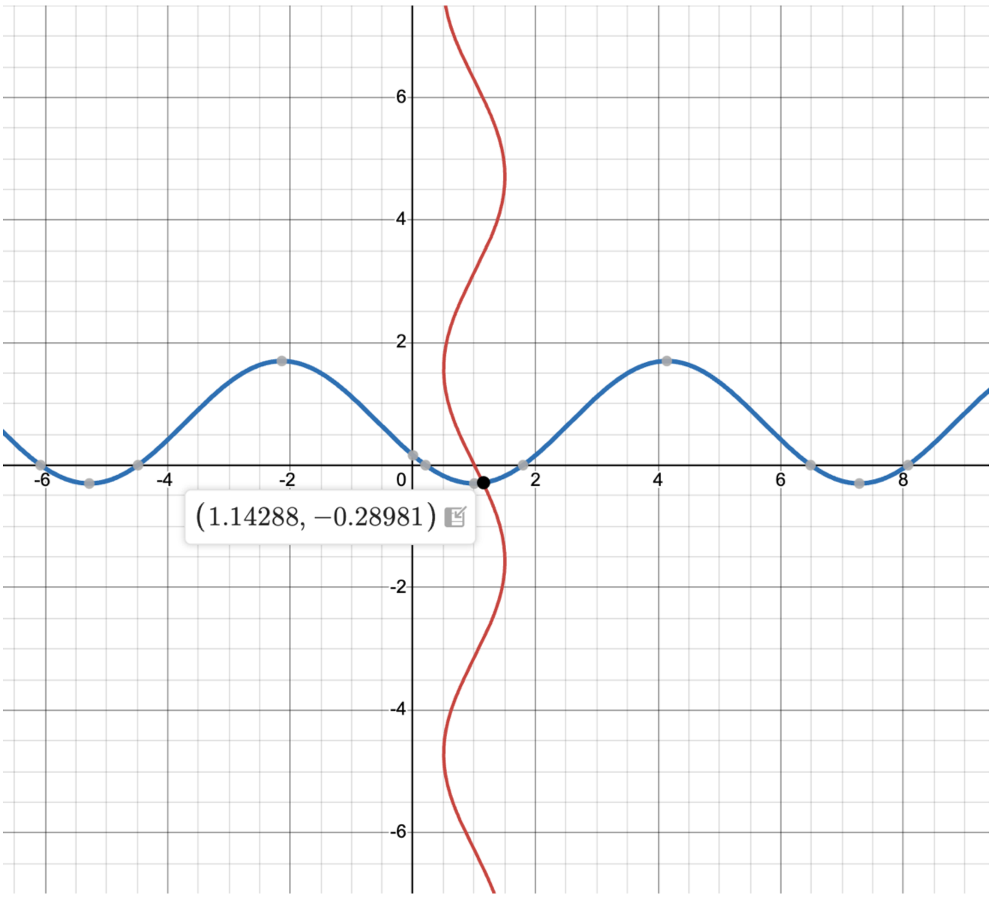
Критерий окончания итерационного процесса:

$$|x_n - x_{n-1}| \le \varepsilon = 0.01$$

№ итерации	x_k	$f(x_k)$	$f'(x_k)$	$f x_{k+1}$	$\lvert x_{k+1} - x_k \rvert$
1	-1,000	26,460	-23,890	0,108	1,108
2	0,108	3,579	-16,200	0,329	0,221
3	0,329	0,261	-13,786	0,347	0,019
4	**0,347**	0,002	-13,565	0,348	0,0002 < **0,01**

Решение системы нелинейных уравнений

$$\begin{cases} siny + 2x = 2 \\ y + cos(x-1) = 0.7 \end{cases}$$



Уточним корень с помощью **метода простой итерации:**

Решение системы находится в области G:

$$1 < x < 2, \qquad -1 < y < 0$$

$$\begin{cases} f_1(x,y) = siny + 2x - 2 = 0 \\ f_2(x,y) = y + cos(x-1) - 0{,}7 = 0 \end{cases}$$

$$\begin{cases} x = 1 - \frac{siny}{2} \\ y = 0.7 - cos(x-1) \end{cases}$$

Проверим условие сходимости:

$$\frac{\partial\varphi_1}{\partial x} = 0, \qquad \frac{\partial\varphi_1}{\partial y} = -\frac{cosy}{2}$$

$\frac{\partial\varphi_2}{\partial x} = sin(x-1), \qquad \frac{\partial\varphi_2}{\partial y} = 0$

$\left|\frac{\partial\varphi_1}{\partial x}\right| + \left|\frac{\partial\varphi_1}{\partial y}\right| = \left|-\frac{cosy}{2}\right| \leq 0.5$

$\left|\frac{\partial\varphi_2}{\partial x}\right| + \left|\frac{\partial\varphi_2}{\partial y}\right| = |sin(x-1)| \leq sin(2) \approx 0.9$

$\max\limits_{[x,y\in G]}|\varphi'(x,y)| \leq 0.9 < 1 \Rightarrow$ процесс **сходящийся**.

Критерий окончания итерационного процесса:

$$max(\left|x^{(k+1)} - x^{(k)}\right|, \left|y^{(k+1)} - y^{(k)}\right|) \leq \varepsilon = 0.01$$

Выберем начальное приближение: $x^{(0)} = 2, y^{(0)} = 0$

№ шага	$x^{(k+1)}$	$y^{(k+1)}$	$\left\|x^{(k+1)} - x^{(k)}\right\|$	$\left\|y^{(k+1)} - y^{(k)}\right\|$	$max(\left\|x^{(k+1)} - x^{(k)}\right\|, \left\|y^{(k+1)} - y^{(k)}\right\|)$
1	1,000	0,160	1,000	0,160	1,000
2	0,920	-0,300	0,080	0,460	0,460
3	1,148	-0,297	0,227	0,003	0,227
4	**1,146**	**-0,289**	0,002	0,008	0,008 < **0,01**

Часть 2: Программная реализация задачи.

Для нелинейных уравнений

Листинг программы

```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt

MAX_ITERATIONS = 500


def print_results(result):
    """Вывод результатов в консоль или файл"""
    if output_number == '2':
        with open(output_filename, 'a') as file:
            file.write(result + '\n')
    else:
        print(result)


def build_graph():
    """Построение графика функции"""
    x_values = np.linspace(a, b, 400)
    y_values = f(x_values)

    plt.figure(figsize=(12, 10))

    plt.plot(x_values, y_values, label=f'f(x)')

    plt.xticks(np.arange(a, b, 0.25))

    plt.axhline(0, color='black', linewidth=1)
    plt.axvline(0, color='black', linewidth=1)
    plt.axvline(a, color='green', linestyle='--', label=f'a = {a:.3f}')
    plt.axvline(b, color='blue', linestyle='--', label=f'b = {b:.3f}')
    plt.title(f'График функции f(x) на интервале [{a}, {b}]')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.ylim(max(f(a), -25), min(f(b), 25))
    plt.legend()
    plt.grid(True)
    plt.show()


def is_monotonic_derivative():
    x = sp.symbols('x')

    test_points = [a + (b - a) * i / 10 for i in range(11)]
    derivative_values = [derivatives[equation_number].subs(x, point).evalf() for
point in test_points]

    sign_changes = 0
    for i in range(1, len(derivative_values)):
        if derivative_values[i - 1] * derivative_values[i] < 0:
            sign_changes += 1
```

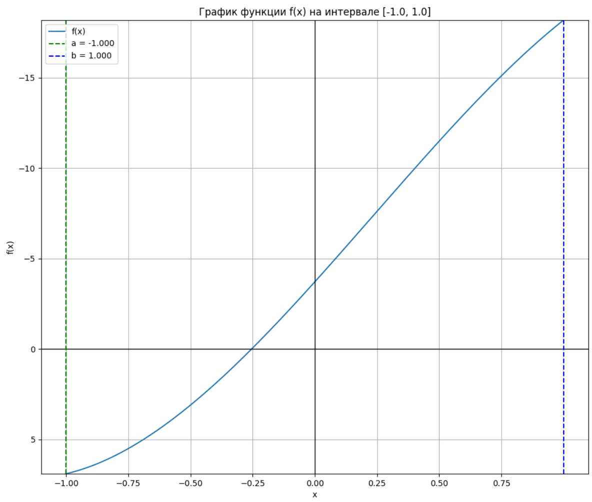
```
    return sign_changes == 0


def half_division_method(a, b, eps, n=0):
    """Метод половинного деления"""
    if n > MAX_ITERATIONS:
        raise ValueError("Метод не сошёлся за максимальное число итераций.")
    x = (a + b) / 2
    print_results(f"Шаг {n}: x = {x:.6f}, f(x) = {f(x):.6f}, [a, b] = [{a:.6f},
{b:.6f}], |b - a| = {abs(b - a):.6f}")
    if abs(b - a) < eps and abs(f(x)) < eps:
        return (a + b) / 2, f((a + b) / 2), n
    if f(a) * f(x) > 0:
        return half_division_method(x, b, eps, n + 1)
    else:
        return half_division_method(a, x, eps, n + 1)


def secant_method(x0, x1, eps, n = 0):
    """Метод секущих"""
    if n > MAX_ITERATIONS:
        raise ValueError("Метод не сошёлся за максимальное число итераций.")
    x = x1 - (x1-x0)/(f(x1)-f(x0)) * f(x1)   # используем разностное приближение
    print_results(f"Шаг {n}: x_(i-1) = {x0:.6f}, x_i = {x1:.6f}, x_(i+1) =
{x:.6f}, f(x_(i+1)) = {f(x):.6f}, |x_(i+1) - x_i| = {abs(x1 - x0):.6f}")
    if abs(x - x1) <= eps or abs(f(x)) <= eps:
        return x, f(x), n
    return secant_method(x1, x, eps, n + 1)


def simple_iteration_method(x0, phi, eps, n = 0):
    """Метод простой итерации"""
    if n > MAX_ITERATIONS:
        raise ValueError("Метод не сошёлся за максимальное число итераций.")
    x = phi(x0)
    print_results(f"Шаг {n}: x_i = {x0:.6f}, x_(i+1) = {x:.6f}, φ(x_(i+1)) =
{phi(x):.6f}, f(x_(i+1)) = {f(x):.6f}, |x_(i+1) - x_i| = {abs(x - x0):.6f}")
    if abs(f(x)) < eps:
        return x, f(x), n
    return simple_iteration_method(x, phi, eps, n + 1)


print('Вычислительная математика. Лабораторная работа 2_1: "Численное решение
нелинейных уравнений". Вариант 13\n')


# --- Исходные данные ---

a = 0
b = 0
eps = 0


# --- Уравнения и их производные ---

x = sp.symbols('x')
equations = {
    '1': sp.sympify(2.74 * x**3 - 1.93 * x**2 - 15.28 * x - 3.72),
    '2': sp.sympify(3.12*sp.exp(0.8*x) - 2.45*sp.sin(1.3*x) + 4.67*x - 7.89),
    '3': sp.sympify(-0.38*x**3 - 3.42*x**2 + 2.51*x + 8.75),
    '4': sp.sympify(-2.71*sp.log(1 + 0.9*x) - 5.32*x + 4.12)
```

```
}

derivatives = {key:  sp.diff(equations[key]) for key in equations.keys()}
second_derivatives = {key: sp.lambdify(x, sp.diff(derivatives[key]), "numpy") for
key in equations.keys()}


# --- Ввод данных ---

print('1: 2.74x^3 - 1.93x^2 - 15.28x - 3.72\n2: 3.12e^(0.8x) - 2.45sin(1.3x) +
4.67x - 7.89')
print('3: -0.38x^3 - 3.42x^2 + 2.51x + 8.75\n4: -2.71ln(1 + 0.9x) - 5.32x +
4.12')
equation_number = input('Выберите уравнение: ')

while equation_number not in {'1', '2', '3', '4'}:
    equation_number = input('Выберите первое (1), второе (2), третье (3) или
четвёртое (4) уравнение: ')

f = sp.lambdify(x, equations[equation_number], "numpy")
df = sp.lambdify(x, derivatives[equation_number], "numpy")


print('\n1: Метод половинного деления\n2: Метод секущих\n3: Метод простой
итерации')
method_number = input('Выберите метод решения: ')
while method_number not in {'1', '2', '3'}:
    method_number = input('Выберите метод половинного деления (1), метод секущих
(2) или метод простой итерации (3): ')


print('\nКак вы хотите ввести исходные данные(границы интервала, начальное
приближение к корню и погрешность вычисления)?')
print("1: с консоли\n2: с файла")
input_number = input()
while input_number not in {'1', '2'}:
    input_number = input('Выберите первый (1) или второй (2) вариант: ')

if input_number == '1':
    print("\nВведите границы интервала:")
    a = float(input('a = '))
    if equation_number == '4':
        while a <= -1/0.9:
            print('Выходит за области определения (x > -1/0.9). Пожалуйста,
повторите ввод.')
            a = float(input('a = '))
    b = float(input('b = '))
    while a >= b:
        print('Правая границы должна быть больше левой. Пожалуйста, повторите
ввод.')
        b = float(input('b = '))
    print("Введите погрешность вычисления:")
    eps = float(input('ε = '))
else:
    input_filename = input("Введите название файла: ").strip()
    with open(input_filename, 'r') as file:
        lines = file.readlines()
        a, b, eps = float(lines[0]), float(lines[1]), float(lines[2])
        if a >= b:
            raise ValueError('Правая границы должна быть больше левой.')
        if equation_number == '4':
```

```
            if a <= -1/0.9:
                raise ValueError('Левая граница выходит за области определения (x
> -1/0.9).')

print('\nГде вы хотите вывести результаты (найденный корень уравнения, значение
функции в корне, число итераций)?')
print("1: в консоли\n2: в файле")
output_number = input()
while output_number not in {'1', '2'}:
    output_number = input('Выберите первый (1) или второй (2) вариант: ')

if output_number == '2':
    output_filename = input("Введите название файла: ").strip()

# --- Построение графика функции ---

build_graph()


# --- Проверка существования единственного корня на заданном интервале ---

if f(a) * f(b) >= 0:
    raise ValueError("На заданном интервале отсутствуют корни!")
elif not is_monotonic_derivative():
    raise ValueError("На заданном интервале несколько корней!")


# --- Вычисление корней ---

print()

if method_number == '1':
    x, fx, n = half_division_method(a, b, eps)
    print_results(f"\nИтог: x = {x:.6f}, f(x) = {fx:.6f}, шагов: {n}")

elif method_number == '2':
    x0 = b
    if f(a) * second_derivatives[equation_number](a) > 0:
        x0 = a
    x, fx, n = secant_method(x0, x0 + eps, eps)
    print_results(f"\nИтог: x = {x:.6f}, f(x) = {fx:.6f}, шагов: {n}")

else:
    lamb = 1 / max(df(a), df(b))
    if df(a) > 0 and df(b) > 0:
        lamb = -lamb
    phi = lambda x: x + lamb * f(x)
    dphi = lambda x: 1 + lamb * df(x)
    q = max(abs(phi(a)), abs(phi(b)))
    print_results(f'|φ(a)| = {abs(phi(a))}, |φ(b)| = {abs(phi(b))}')
    if q < 1:
        print_results("Итерационная последовательность сходится!\n")
        if q <= 0.5:
            x, fx, n = simple_iteration_method(a, phi, eps)
        else:
            x, fx, n = simple_iteration_method(a, phi, (1 - q) / q * eps)
        print_results(f"\nИтог: x = {x:.6f}, f(x) = {fx:.6f}, шагов: {n}")
    else:
        print_results("Условие сходимости не выполняется!")
```

Результат работы программы

```
Вычислительная математика. Лабораторная работа 2_1: "Численное решение нелинейных уравнений". Вариант 13

1: 2.74x^3 - 1.93x^2 - 15.28x - 3.72
2: 3.12e^(0.8x) - 2.45sin(1.3x) + 4.67x - 7.89
3: -0.38x^3 - 3.42x^2 + 2.51x + 8.75
4: -2.71ln(1 + 0.9x) - 5.32x + 4.12
Выберите уравнение: 1

1: Метод половинного деления
2: Метод секущих
3: Метод простой итерации
Выберите метод решения: 1

Как вы хотите ввести исходные данные(границы интервала, начальное приближение к корню и погрешность вычисления)?
1: с консоли
2: с файла
1

Введите границы интервала:
a = -0.8
b = -0.2
Введите погрешность вычисления:
ε = 0.01

Где вы хотите вывести результаты (найденный корень уравнения, значение функции в корне, число итераций)?
1: в консоли
2: в файле
1

Шаг 0: x = -0.500000, f(x) = 3.095000, [a, b] = [-0.800000, -0.200000], |b - a| = 0.600000
Шаг 1: x = -0.350000, f(x) = 1.274097, [a, b] = [-0.500000, -0.200000], |b - a| = 0.300000
Шаг 2: x = -0.275000, f(x) = 0.279060, [a, b] = [-0.350000, -0.200000], |b - a| = 0.150000
Шаг 3: x = -0.237500, f(x) = -0.236570, [a, b] = [-0.275000, -0.200000], |b - a| = 0.075000
Шаг 4: x = -0.256250, f(x) = 0.022664, [a, b] = [-0.275000, -0.237500], |b - a| = 0.037500
Шаг 5: x = -0.246875, f(x) = -0.106605, [a, b] = [-0.256250, -0.237500], |b - a| = 0.018750
Шаг 6: x = -0.251563, f(x) = -0.041883, [a, b] = [-0.256250, -0.246875], |b - a| = 0.009375
Шаг 7: x = -0.253906, f(x) = -0.009587, [a, b] = [-0.256250, -0.251563], |b - a| = 0.004688

Итог: x = -0.253906, f(x) = -0.009587, шагов: 7
```



Для систем нелинейных уравнений

Листинг программы

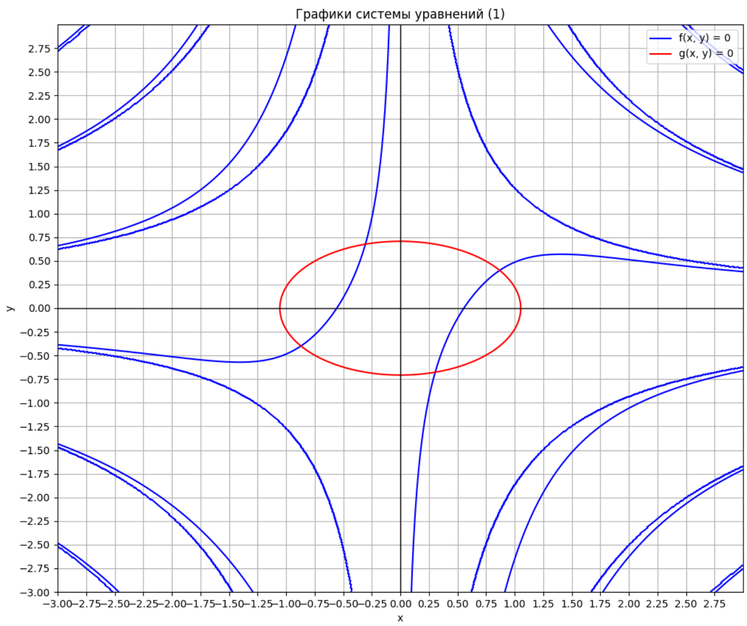
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import sympy as sp

MAX_ITERATIONS = 150


def print_results(result):
    """Вывод результатов в консоль или файл"""
    if output_number == '2':
        with open(output_filename, 'a') as file:
            file.write(result + '\n')
    else:
        print(result)


def build_graph():
    """Построение графика функции"""
    x = np.linspace(-3, 3, 1000)
    y = np.linspace(-3, 3, 1000)
    X, Y = np.meshgrid(x, y)

    F = f(X, Y)
    G = g(X, Y)

    plt.figure(figsize=(12, 10))

    plt.contour(X, Y, F, levels=[0], colors='blue')
    plt.contour(X, Y, G, levels=[0], colors='red')

    plt.xticks(np.arange(-3, 3, 0.25))
    plt.yticks(np.arange(-3, 3, 0.25))

    proxy_f = Line2D([0], [0], color='blue', label='f(x, y) = 0')
    proxy_g = Line2D([0], [0], color='red', label='g(x, y) = 0')

    plt.axhline(0, color='black', linewidth=1)
    plt.axvline(0, color='black', linewidth=1)
    plt.title(f'Графики системы уравнений ({system_number})')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend(handles=[proxy_f, proxy_g])
    plt.grid(True)
    plt.show()


def newtons_method(x0, y0, eps, n = 0):
    """Метод Ньютона"""
    if n > MAX_ITERATIONS:
        raise ValueError("Метод не сошёлся за максимальное число итераций.")

    f_val = -f(x0, y0)
    g_val = -g(x0, y0)

    df_dx = sp.lambdify((x, y), sp.diff(systems[system_number]['f'], x), 'numpy')
    df_dy = sp.lambdify((x, y), sp.diff(systems[system_number]['f'], y), 'numpy')
```

```
    dg_dx = sp.lambdify((x, y), sp.diff(systems[system_number]['g'], x), 'numpy')
    dg_dy = sp.lambdify((x, y), sp.diff(systems[system_number]['g'], y), 'numpy')

    J = np.array([
        [df_dx(x0, y0), df_dy(x0, y0)],
        [dg_dx(x0, y0), dg_dy(x0, y0)]
    ])
    F = np.array([f_val, g_val])
    dx, dy = np.linalg.solve(J, F)

    X = x0 + dx
    Y = y0 + dy

    print_results(f"Шаг {n}: x_i = {X:.6f}, y_i = {Y:.6f}, f(x_i, y_i) =
{f_val:.6f}, g(x_i, y_i) = {g_val:.6f}, "
            f"|dx| = {abs(dx):.6f}, |dy| = {abs(dy):.6f}, x_(i+1) = {X:.6f},
y_(i+1) = {Y:.6f}")

    if abs(X - x0) <= eps and abs(Y - y0) <= eps:
        return X, x0, Y, y0, n
    return newtons_method(X, Y, eps, n + 1)


print('Вычислительная математика. Лабораторная работа 2_2: "Численное решение
системы нелинейных уравнений". '
      'Вариант 13: Метод Ньютона\n')


# --- Исходные данные ---

x0 = 0
y0 = 0
eps = 0


# --- Системы уравнений и их производные ---

x, y = sp.symbols('x y')

systems = {
    '1': {
        'f': sp.sympify(sp.tan(x * y + 0.3) - x**2),
        'g': sp.sympify(0.9 * x**2 + 2 * y**2 - 1),
    },
    '2': {
        'f': sp.sympify(x + sp.sin(y) + 0.4),
        'g': sp.sympify(2 * y - sp.cos(x + 1)),
    }
}


# --- Ввод данных ---

print('1: { tg(xy + 0.3) = x^2; 0.9x^2 + 2y^2 = 1 }\n2: { x + siny = -0.4; 2y -
cos(x + 1) = 0 }')
system_number = input('Выберите систему уравнений: ')

while system_number not in {'1', '2', '3', '4'}:
    system_number = input('Выберите первую (1) или вторую (2) систему уравнений:
')
```

```
f = sp.lambdify((x, y), systems[system_number]['f'], "numpy")
g = sp.lambdify((x, y), systems[system_number]['g'], "numpy")

build_graph()

print('\nКак вы хотите ввести исходные данные(начальные приближение к корню и
погрешность вычисления)?')
print("1: с консоли\n2: с файла")
input_number = input()
while input_number not in {'1', '2'}:
    input_number = input('Выберите первый (1) или второй (2) вариант: ')

if input_number == '1':
    print("\nВведите начальные приближения:")
    x0 = float(input('x0 = '))
    y0 = float(input('y0 = '))
    print("Введите погрешность вычисления:")
    eps = float(input('ε = '))
else:
    input_filename = input("Введите название файла: ").strip()
    with open(input_filename, 'r') as file:
        lines = file.readlines()
        x0, y0, eps = float(lines[0]), float(lines[1]), float(lines[2])


print('\nГде вы хотите вывести результаты (найденный корень уравнения, значение
функции в корне, число итераций)?')
print("1: в консоли\n2: в файле")
output_number = input()
while output_number not in {'1', '2'}:
    output_number = input('Выберите первый (1) или второй (2) вариант: ')

if output_number == '2':
    output_filename = input("Введите название файла: ").strip()


# --- Вычисление корней ---

X, x, Y, y, n = newtons_method(x0, y0, eps)
print_results(f"\nИтог: x = {X:.6f}, y = {Y:.6f}, шагов: {n}, вектор
погрешностей: [{abs(X - x):.6f}, {abs(Y - y):.6f}]")
```

Результат работы программы

```
Вычислительная математика. Лабораторная работа 2_2: "Численное решение системы нелинейных уравнений". Вариант 13: Метод Ньютона

1: { tg(xy + 0.3) = x^2; 0.9x^2 + 2y^2 = 1 }
2: { x + siny = -0.4; 2y - cos(x + 1) = 0 }
Выберите систему уравнений: 1

Как вы хотите ввести исходные данные(начальные приближение к корню и погрешность вычисления)?
1: с консоли
2: с файла
1

Введите начальные приближения:
x0 = 1
y0 = 1
Введите погрешность вычисления:
ε = 0.0001

Где вы хотите вывести результаты (найденный корень уравнения, значение функции в корне, число итераций)?
1: в консоли
2: в файле
1
Шаг 0: x_i = 1.709788, y_i = 0.205595, f(x_i, y_i) = -2.602102, g(x_i, y_i) = -1.900000, |dx| = 0.709788, |dy| = 0.794405, x_(i+1) = 1.709788, y_(i+1) = 0.205595
Шаг 1: x_i = 1.119370, y_i = 0.329022, f(x_i, y_i) = 2.160764, g(x_i, y_i) = -1.715578, |dx| = 0.590418, |dy| = 0.123427, x_(i+1) = 1.119370, y_(i+1) = 0.329022
Шаг 2: x_i = 0.910135, y_i = 0.387816, f(x_i, y_i) = 0.463503, g(x_i, y_i) = -0.344203, |dx| = 0.209235, |dy| = 0.058794, x_(i+1) = 0.910135, y_(i+1) = 0.387816
Шаг 3: x_i = 0.871102, y_i = 0.399182, f(x_i, y_i) = 0.063453, g(x_i, y_i) = -0.046315, |dx| = 0.039033, |dy| = 0.011366, x_(i+1) = 0.871102, y_(i+1) = 0.399182
Шаг 4: x_i = 0.869642, y_i = 0.399595, f(x_i, y_i) = 0.002194, g(x_i, y_i) = -0.001630, |dx| = 0.001460, |dy| = 0.000413, x_(i+1) = 0.869642, y_(i+1) = 0.399595
Шаг 5: x_i = 0.869640, y_i = 0.399596, f(x_i, y_i) = 0.000003, g(x_i, y_i) = -0.000002, |dx| = 0.000002, |dy| = 0.000001, x_(i+1) = 0.869640, y_(i+1) = 0.399596

Итог: x = 0.869640, y = 0.399596, шагов: 5, вектор погрешностей: [0.000002, 0.000001]
```



Выводы

В ходе работы я изучила численные методы решения нелинейных уравнений и систем. Вручную использовала метод простой итерации, метод хорд и метод Ньютона для решения нелинейных уравнений и метод простой итерации для системы нелинейных уравнений. На Python программно реализовала метод половинного деления, метод секущих и метод простой итерации для нелинейных уравнений, а также метод Ньютона для систем нелинейных уравнений.

Результаты показали, что методы эффективны при правильном выборе начальных приближений и проверке условий сходимости. Визуализация графиков помогла уточнить корни и повысить точность вычислений.