Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

Лабораторная работа № 2

«Численное решение нелинейных уравнений и систем»

Вычислительная математика

Вариант №4

Студент

Дубинин Артём Сергеевич

группа P3215

Преподаватель

Малышева Татьяна Алексеевна

Санкт - Петербург 2025 год

Цель работы:

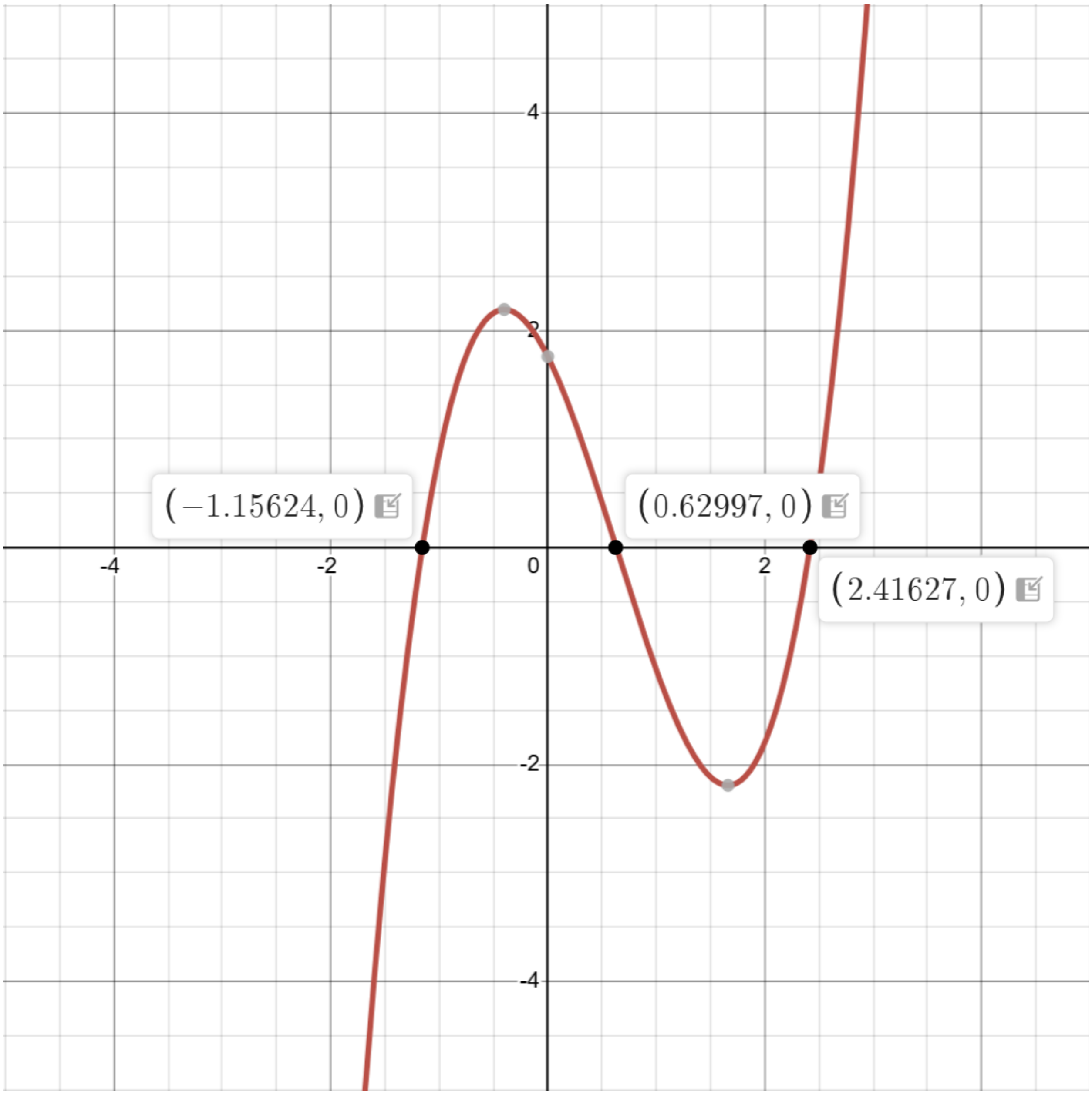
- изучить численные методы решения нелинейных уравнений и их систем, найти корни заданного нелинейного уравнения/системы нелинейных уравнений, выполнить программную реализацию методов.

1. Вычислительная реализация задачи:

Вид нелинейного уравнения для вычислительной реализации:

- $x^3 - 1{,}89x^2 - 2x + 1{,}76$

1)



2) Интервалы изоляции корней

a)
- **Функция:** $x^3 - 1.89x^2 - 2x + 1.76$
- **Производная:** $3x^2 - 3.78x - 2$

b) Нахождение критических точек:
Решим уравнение $3x^2 - 3.78x - 2 = 0$
$D = 38.2884$
$x_1 = 1.66, \;\; x_2 = -0.4$

c) Определение интервалов монотонности:
- $f(x)$ возрастает на $(-\infty, -0.4)$и $(1.66, +\infty)$
- $f(x)$ убывает на $(-0.4, 1.66)$

d) Поиск интервалов изоляции корней:
Вычислим значения функции в критических точках и на бесконечности:
- $f(-\infty) = -\infty$
- $f(-0.4) = 2.19$
- $f(1.66) = -2.19$
- $f(+\infty) = +\infty$

Изменения знака функции:

1) от $-\infty$ до -0.4: $f(x)$ возрастает от $-\infty$ до 2.19. Корень есть т.к f(x) пересекает ось X: $f(-2) \approx -9.8 \quad f(-1) \approx 0.87$
$x_1 \in (-2, -1)$
2) от -0.4 до 1.66: $f(x)$ убывает от 2.19 до -2.19. Корень есть т.к f(x) пересекает ось X: $f(0) \approx 1.76 \;\; f(1) \approx -1.13$
$x_2 \in (0, 1)$
3) от 1.66 до $+\infty$: $f(x)$ возрасает от -2.19 до $+\infty$. Корень есть т.к f(x) пересекает ось X: $f(2) \approx -1.8 \;\; f(3) \approx 5.75$
$x_3 \in (2, 3)$

3) Корни уравнения:
$x_1 \approx -1.16$
$x_2 \approx 0.63$
$x_3 \approx 2.42$

4)

Крайний правый корень – Метод простой итерации

Крайний левый корень – Метод половинного деления

Центральный корень – Метод секущих

Крайний правый корень (Метод простой итерации):

Проверка **условия сходимости** метода на выбранном интервале:

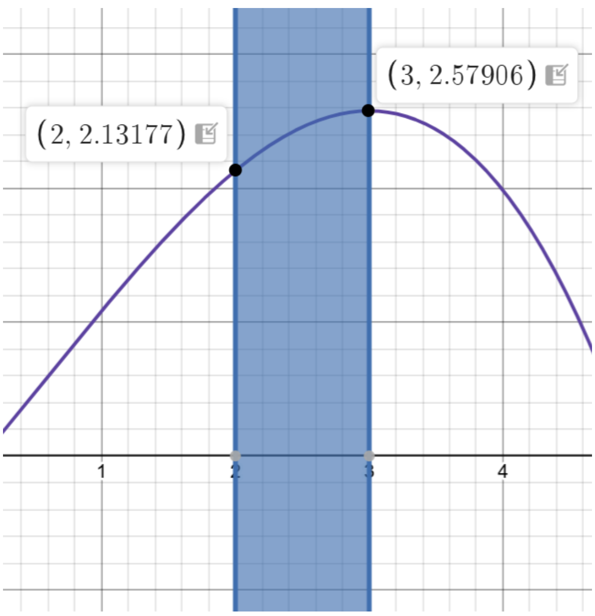
$$f(x) = x^3 - 1{,}89x^2 - 2x + 1{,}76 \quad a = 2, b = 3$$
$$f'(x) = 3x^2 - 3.78x - 2$$
$$f`(a) = 2.44 > 0 \quad f`(b) = 13.66 > 0$$
$$\max\,(|f'(a)|, |f'(b)|) = 13.66 \rightarrow \lambda = -\frac{1}{\max\,(|f'(x)|)} = -\frac{1}{13.66}$$

$$\varphi(x) = x + \lambda f(x) = x - \frac{x^3 - 1.89x^2 - 2x + 1.76}{13.66}$$

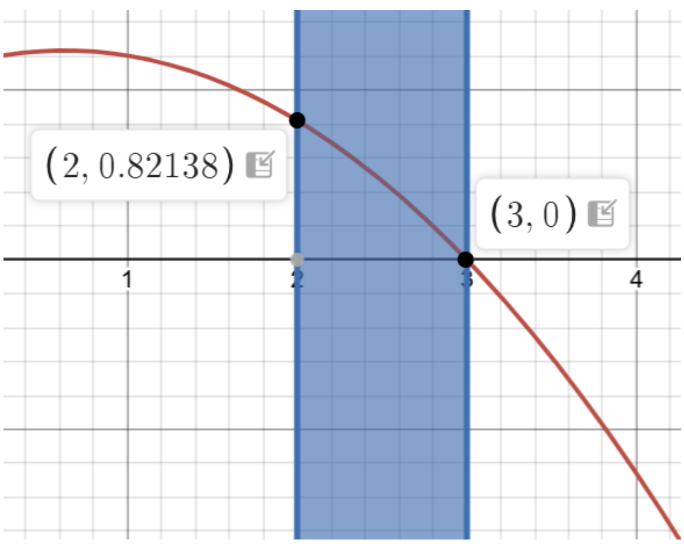
$$\varphi'^{(x)} = 1 + \lambda f'^{(x)} = 1 - \frac{3x^2 - 3.78x - 2}{13.66}$$

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

$\boxed{\varphi(x)}$



$\boxed{\varphi'(x)}$



$|\varphi'(a)| = 0.821$

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

$|\varphi'(x)| \leq q,$ где $q = 0.821$

$0 \leq q < 1 \rightarrow$ **итерационная последовательность сходится,** $\rightarrow$ критерий окончания итерационного процесса $|x_{k+1} - x_k| \leq \frac{1-q}{q}\varepsilon$, $x_0 = 3$

5)

Крайний правый корень (Метод простой итерации):

№	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	3.000	2.57906	1.1852	0.42094
2	2.57906	2.49187	0.513521	0.08719
3	2.49187	2.45428	0.250375	0.03759
4	2.45428	2.43595	0.120587	0.01833
5	2.43595	2.42712	0.0698864	0.00883
6	2.42712	2.422003	0.036776	0.005116
7	2.422003	2.419310	0.0194636	0.002693
8	2.419310	2.417885	0.010334	0.001424

Крайний левый корень (Метод половинного деления):

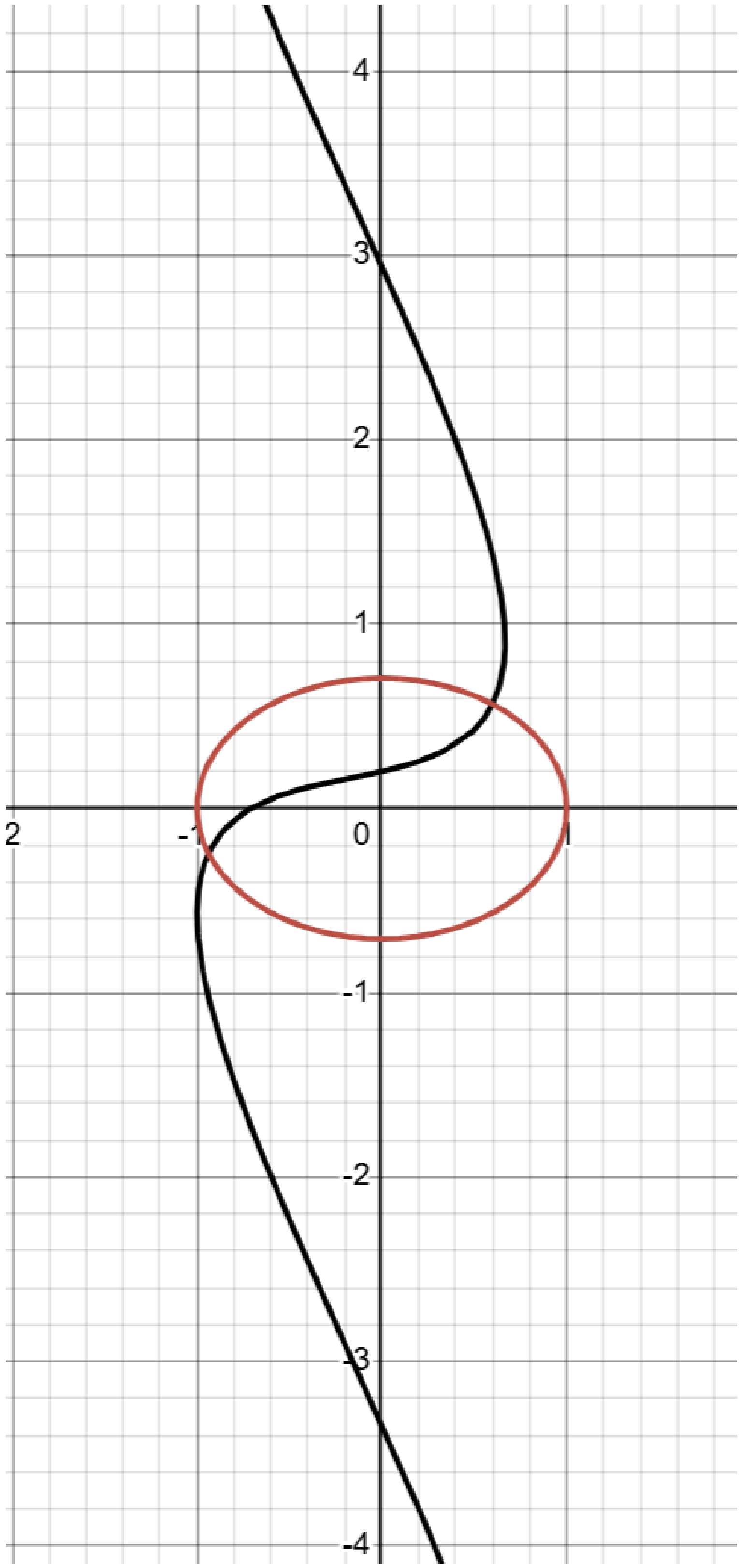
№	a	b	x	f(a)	f(b)	f(x)	|a – b|
1	-2.000	-1.000	-1.500	-9.800	0.870	-2.868	1.000
2	-1.500	-1.000	-1.25	-2.868	0.870	-0.646	0.500
3	-1.25	-1.000	-1.125	-0.646	0.870	0.194	0.25
4	-1.25	-1.125	-1.188	-0.646	0.194	-0.205	0.125
5	-1.188	-1.125	-1.156	-0.208	0.194	-0.002	0.063
6	-1.156	-1.125	-1.141	0.002	0.194	0.099	0.031
7	-1.141	-1.125	-1.133	0.096	0.194	0.145	0.016
8	-1.133	-1.125	-1.129	0.145	0.194	0.170	0.008

Центральный корень (Метод секущих):

№	x_{k-1}	x_k	x_{k+1}	$f(x_{k+1})$	$\lvert x_{k+1} - x_k \rvert$
1	0.000	0.010	0.872	-0.757	0.862
2	0.010	0.872	0.61	0.062	0.262
3	0.872	0.610	**0.630**	-0.001	0.02

2. Решение системы нелинейных уравнений:

4	$\begin{cases} \sin(x+y) - 1{,}2x = 0{,}2 \\ x^2 + 2y^2 = 1 \end{cases}$	Метод Ньютона



$$\begin{cases}\sin(x+y) - 1.2x = 0.2 \\ x^2 + 2y^2 = 1\end{cases} \to \begin{cases}\mathrm{f(x,y)} = 0 \\ \mathrm{g(x,y)} = 0\end{cases} \to \begin{cases}\sin(x+y) - 1.2x - 0.2 = 0 \\ x^2 + 2y^2 - 1 = 0\end{cases}$$

Отметим, что решение системы уравнений являются точки пересечения эллипса и $\sin(x+y) - 1.2x - 0.2 = 0$, следовательно, система имеет не более двух различных решений.

Построим матрицу Якоби:

$$\frac{\partial f}{\partial x} = \cos(x+y) - 1.2, \frac{\partial f}{\partial y} = \cos\,(x+y), \frac{\partial g}{\partial x} = 2x, \frac{\partial g}{\partial y} = 4y$$

$$\begin{vmatrix}\frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y}\end{vmatrix} \begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix} = -\begin{pmatrix}f(x,y) \\ g(x,y)\end{pmatrix}$$

$$\begin{vmatrix}\cos(x+y) - 1.2 & \cos\,(x+y) \\ 2x & 4y\end{vmatrix} \begin{pmatrix}\Delta x \\ \Delta y\end{pmatrix} = \begin{pmatrix}1.2x + 0.2 - \sin\,(x+y) \\ 1 - x^2 - 2y^2\end{pmatrix}$$

$$\begin{cases}\cos(x+y)\Delta x - 1.2\Delta x + \cos\,(x+y)\Delta y = 1.2x + 0.2 - \sin\,(x+y) \\ 2x\Delta x + 4y\Delta y = 1 - x^2 - 2y^2\end{cases}$$

Корень 1: Шаг 1: Выбираем $x_0 = 0.5;\ y_0 = 0.5$

Шаг 2. Решаем полученную систему.

$$\begin{cases}-0.65\Delta x + 0.54\,\Delta y = -0.041 \\ \Delta x + 2\Delta y = 0.25\end{cases} \to \Delta x = 0.118;\ \Delta y = 0.066$$

Шаг 3. Вычисляем очередные приближения:

$$x_1 = x_0 + \Delta x = 0.5 + 0.118 = 0.618$$

$$y_1 = y_0 + \Delta y = 0.5 + 0.066 = 0.566$$

$|x_1 - x_0| \le \varepsilon$, $|y_1 - y_0| \le \varepsilon$

$|0.618 - 0.5| \le \varepsilon$, $|0.566 - 0.5| \le \varepsilon$

Повторяем действия с шага 2, только с $x_0 = 0.618;\ y_0 = 0.566$

И получаем Корень 1 = (0.6, 0.566)

Аналогично находим **другой корень**: $(-0.9381, -0.2449)$

Программная реализация задачи:

Листинг программы:

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


# ----------------------------------------------------------------------------
---
# 1. Функции-уравнения и их производные (для уравнений одной переменной)
# ----------------------------------------------------------------------------
---

def f1(x):
    return x ** 3 - x - 1


def df1(x):
    return 3 * x ** 2 - 1


def phi1(x):
    return (x + 1) ** (1 / 3)


def phi1_sym(x):
    return (x + 1) ** (sp.Rational(1, 3))


def f2(x):
    return np.sin(x) - 0.5 * x


def df2(x):
    return np.cos(x) - 0.5


def phi2(x):
    return 2.0 * np.sin(x)


def phi2_sym(x):
    return 2 * sp.sin(x)


def f3(x):
    return np.exp(x) - 3 * x


def df3(x):
    return np.exp(x) - 3


def phi3(x):
    if x <= 0:
        return 0.5
    return np.log(3.0 * x)


def phi3_sym(x):
```

```
    return sp.log(3 * x)


FUNCTIONS = {
    1: {'f': f1, 'df': df1, 'phi': phi1, 'phi_sym': phi1_sym, 'name': "x^3 -
x - 1"},
    2: {'f': f2, 'df': df2, 'phi': phi2, 'phi_sym': phi2_sym, 'name':
"sin(x) - 0.5*x"},
    3: {'f': f3, 'df': df3, 'phi': phi3, 'phi_sym': phi3_sym, 'name':
"exp(x) - 3x"}
}


# ---------------------------------------------------------------------------
---
# 3. Функции ввода-вывода
# ---------------------------------------------------------------------------
---

def write_output_to_file(filename, text):
    with open(filename, 'w', encoding='utf-8') as f:
        f.write(text)


def finalize_output(output_choice, result_text):
    text_to_write = "\n".join(result_text)
    if output_choice == 'file':
        write_output_to_file("output.txt", text_to_write)
        print("Результаты сохранены в output.txt")
    else:
        print("\n" + text_to_write)


# ---------------------------------------------------------------------------
---
# 4. Методы решения уравнений (одной переменной)
# ---------------------------------------------------------------------------
---

def verify_interval_has_single_root(f, a, b):
    fa, fb = f(a), f(b)
    if fa == 0:
        return False, f"Внимание: f(a)=0 при a={a}."
    if fb == 0:
        return False, f"Внимание: f(b)=0 при b={b}."
    if fa * fb > 0:
        return False, "На концах интервала функция имеет одинаковый знак."
    return True, ""


def chord_method(f, a, b, tol=1e-6, max_iter=100):
    fa, fb = f(a), f(b)
    if fa * fb > 0:
        print("Предусловие метода хорд не выполнено (f(a) и f(b) одного
знака).")
    x_left, x_right = a, b
    for i in range(max_iter):
        f_left, f_right = f(x_left), f(x_right)
        c = x_right - f_right * (x_right - x_left) / (f_right - f_left)
        if abs(f(c)) < tol:
            return c, i + 1, ""
        x_left, x_right = x_right, c
    return None, max_iter, "Метод хорд: превышено число итераций."
```

```
def approx_second_derivative(f, x, h=1e-5):
    return (f(x + h) - 2 * f(x) + f(x - h)) / (h ** 2)


def newton_method(f, df, a, b, tol=1e-6, max_iter=100):
    ddf_a = approx_second_derivative(f, a)
    ddf_b = approx_second_derivative(f, b)
    debug_msg = ""
    if f(a) * ddf_a > 0:
        x = a
        debug_msg += f"Начальное приближение: a = {a} (f(a)*f''(a) > 0)"
    elif f(b) * ddf_b > 0:
        x = b
        debug_msg += f"Начальное приближение: b = {b} (f(b)*f''(b) > 0)"
    else:
        x = 0.5 * (a + b)
        debug_msg += f"Начальное приближение: центр отрезка = {x}, 
(f(a)*f''(a) < 0) и (f(b)*f''(b) < 0)"
    for i in range(max_iter):
        fx, dfx = f(x), df(x)
        if abs(dfx) < 1e-14:
            return None, i, debug_msg + " | f'(x) слишком мало."
        x_new = x - fx / dfx
        if abs(x_new - x) < tol:
            return x_new, i + 1, debug_msg
        x = x_new
    return None, max_iter, debug_msg + " | Превышено число итераций."


def iteration_method(phi_sym_func, phi_num_func, a, b, x0, tol=1e-6, 
max_iter=100):
    x_sym = sp.Symbol('x', real=True)
    phi_sym_expr = phi_sym_func(x_sym)
    dphi_sym = sp.diff(phi_sym_expr, x_sym)
    xs = np.linspace(a, b, 50)
    max_dphi = max([abs(dphi_sym.subs(x_sym, xx)) for xx in xs])
    conv_msg = ""
    if max_dphi >= 1:
        conv_msg = f"WARNING: max|phi'(x)| = {max_dphi:.3f} >= 1; метод 
может не сходиться."
    else:
        conv_msg = f"Условие сходимости выполнено: max|phi'(x)| = 
{max_dphi:.3f} < 1."
    x_cur = x0
    for i in range(max_iter):
        x_next = phi_num_func(x_cur)
        if abs(x_next - x_cur) < tol:
            return x_next, i + 1, conv_msg
        x_cur = x_next
    return None, max_iter, conv_msg


def plot_function(f, a, b):
    xs = np.linspace(a, b, 400)
    ys = [f(x) for x in xs]
    plt.figure(figsize=(6, 4))
    plt.plot(xs, ys, label='f(x)')
    plt.axhline(0, color='black', lw=0.8)
    plt.title(f"График функции на [{a}, {b}]")
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
def plot_function_with_boundaries(f, a, b):
    xs = np.linspace(a, b, 400)
    ys = [f(x) for x in xs]
    plt.figure(figsize=(6, 4))
    plt.plot(xs, ys, label='f(x)')
    plt.axvline(a, color='green', linestyle='--', label=f'a = {a}')
    plt.axvline(b, color='orange', linestyle='--', label=f'b = {b}')
    plt.axhline(0, color='black', lw=0.8)
    plt.title(f"График функции на интервале [{a}, {b}]")
    plt.legend()
    plt.grid(True)
    plt.show()


# ---------------------------------------------------------------------------
# 6. Ввод данных
# ---------------------------------------------------------------------------

def read_equation_input_file(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        lines = [line.strip() for line in f if line.strip()]
    func_choice = int(lines[0])
    a, b = map(float, lines[1].split())
    tol = float(lines[2])
    output_choice = lines[3].lower()
    return func_choice, a, b, tol, output_choice


def read_equation_input_keyboard():
    print("Сначала отображается график функции.")
    print("Выберите функцию для решения уравнения:")
    for k, v in FUNCTIONS.items():
        print(f"{k}: {v['name']}")
    func_choice = int(input("Номер функции: "))
    if func_choice not in FUNCTIONS:
        return None
    data = FUNCTIONS[func_choice]
    plot_function(data['f'], -10, 10)
    a, b = map(float, input("Введите границы интервала (a b): ").split())
    tol = float(input("Введите точность (например 1e-6): "))
    output_choice = input("Куда выводить результат? (file/console):
").strip().lower()
    return func_choice, a, b, tol, output_choice


# ---------------------------------------------------------------------------
# Функции для решения систем методом простой итерации
# ---------------------------------------------------------------------------
def iteration_method_system(phi_sym_funcs, phi_num_funcs, x0, y0, tol=1e-6,
max_iter=100, threshold=1e6):
    """
    Фиксированная итерация для системы двух уравнений:
    x = phi1(x,y), y = phi2(x,y).
    Если значения выходят за пределы threshold, процесс останавливается как
расходящийся.
    Возвращает приближение, число итераций, вектор ошибок, константу q и
сообщение.
    """
    # Символьные переменные для производных
```

```
    x_sym, y_sym = sp.symbols('x y', real=True)
    phi1_sym, phi2_sym = phi_sym_funcs

    # Вычисляем сумму модулей частных производных в (x0,y0)
    subs = {x_sym: x0, y_sym: y0}
    d11 = abs(sp.diff(phi1_sym, x_sym).subs(subs))  # (y/3)dx = 0
    d12 = abs(sp.diff(phi1_sym, y_sym).subs(subs))  # (y/3)dy = 1/3
    d21 = abs(sp.diff(phi2_sym, x_sym).subs(subs))  # ((x^3)/2-1)dx =
(3*x^2)/2
    d22 = abs(sp.diff(phi2_sym, y_sym).subs(subs))  # ((x^3)/2-1)dy = 0

    print(d11+d12, " ", d21+d22)

    # Вывод φ-функций
    print(f"φ₁(x,y) = {phi1_sym}")
    print(f"φ₂(x,y) = {phi2_sym}")

    # производные φ-функций
    print("∂φ₁/∂x =", sp.diff(phi1_sym, x_sym))
    print("∂φ₁/∂y =", sp.diff(phi1_sym, y_sym))
    print("∂φ₂/∂x =", sp.diff(phi2_sym, x_sym))
    print("∂φ₂/∂y =", sp.diff(phi2_sym, y_sym))

    q = float(max(d11 + d12, d21 + d22))
    if q < 1:
        conv_msg = f"Условие сходимости выполнено: q = {q:.3f} < 1."
    else:
        conv_msg = f"WARNING: q = {q:.3f} >= 1; метод может расходиться."

    # Итерации
    errors = []
    x_cur, y_cur = x0, y0
    for i in range(1, max_iter + 1):
        x_next = phi_num_funcs[0](x_cur, y_cur)
        y_next = phi_num_funcs[1](x_cur, y_cur)

        # Проверка на расходящиеся значения
        if abs(x_next) > threshold or abs(y_next) > threshold:
            return (x_cur, y_cur), i - 1, errors, q, (
                f"ERROR: итерации вышли за пределы ±{threshold:.0f} на шаге
{i}. Процесс расходится."
            )

        print(x_cur, y_cur)

        errx = abs(x_next - x_cur)
        erry = abs(y_next - y_cur)
        errors.append((errx, erry))

        # Критерий останова
        if max(errx, erry) < tol:
            return (x_next, y_next), i, errors, q, conv_msg + " Процесс
сошёлся."

        x_cur, y_cur = x_next, y_next

    # Если не сошлось за max_iter
    return (x_cur, y_cur), max_iter, errors, q, (
        f"WARNING: не достигнута точность за {max_iter} итераций; "
        "возвращено последнее приближение"
    )


# ----------------------------------------------------------------------------
```

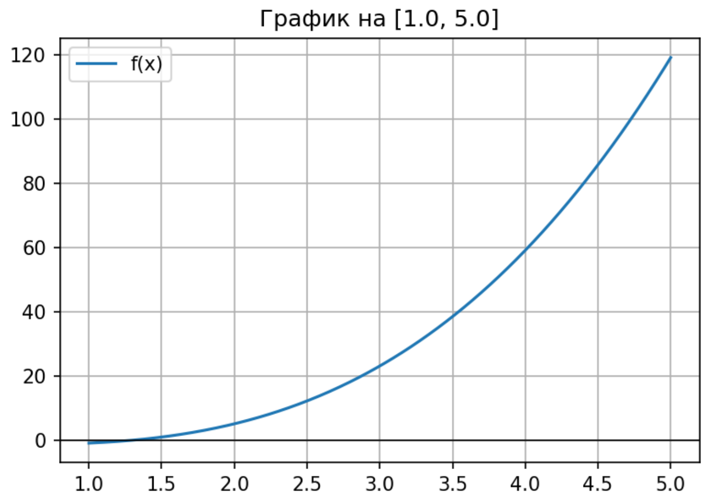
```
---
# 7. Основная логика
# -----------------------------------------------------------------------------
---

def main():
    problem_type = input("Что решаем? (equation/system): ").strip().lower()
    result_text = []

    if problem_type == 'equation':
        input_src = input("Считать данные из файла или клавиатуры?
(file/keyboard): ").strip().lower()
        if input_src == 'file':
            func_choice, a, b, tol, output_choice =
read_equation_input_file("input_equation.txt")
        else:
            user_input = read_equation_input_keyboard()
            if user_input is None:
                return
            func_choice, a, b, tol, output_choice = user_input

        data = FUNCTIONS.get(func_choice)
        if data is None:
            print("Некорректный номер функции.")
            return

        plot_function_with_boundaries(data['f'], a, b)
        f, df = data['f'], data['df']
        phi_num = data['phi']
        phi_sym = data['phi_sym']

        if a > b:
            a, b = b, a

        ok, msg = verify_interval_has_single_root(f, a, b)
        if not ok:
            result_text.append("Внимание: " + msg)

        root_ch, it_ch, msg_ch = chord_method(f, a, b, tol)
        if root_ch is not None:
            result_text.append(
                f"Метод хорд (a = {a}, b = {b}): корень = {root_ch:.6f},
f(root) = {f(root_ch):.6e}, итераций = {it_ch}")
        else:
            result_text.append("Метод хорд: " + msg_ch)

        root_nw, it_nw, newton_debug = newton_method(f, df, a, b, tol)
        if root_nw is not None:
            result_text.append(
                f"Метод Ньютона ({newton_debug}): корень = {root_nw:.6f},
f(root) = {f(root_nw):.6e}, итераций = {it_nw}")
        else:
            result_text.append("Метод Ньютона: не удалось найти корень. " +
newton_debug)

        x0 = 0.5 * (a + b)
        root_it, it_it, iter_debug = iteration_method(phi_sym, phi_num, a,
b, x0, tol)
        if root_it is not None:
            result_text.append(
                f"Метод итераций ({iter_debug}, начальное x0 = {x0}): корень
= {root_it:.6f}, f(root) = {f(root_it):.6e}, итераций = {it_it}")
        else:
            result_text.append("Метод итераций: не удалось найти корень. " +
```

```
iter_debug)

        finalize_output(output_choice, result_text)

    elif problem_type == 'system':
        print("Метод простой итерации для систем нелинейных уравнений")
        print("1) y = 3x;   y = x^3/2 - 1")
        print("2) x^2 + y^2 = 25;   y = x^3 - 2")
        choice = input("Выберите систему (1 или 2): ").strip()
        if choice not in ('1', '2'):
            print("Неверный выбор системы.")
            return
        tol = float(input("Введите точность (например 1e-6): "))

        #Настройка phi-функций и графика
        if choice == '1':
            x_s, y_s = sp.symbols('x y', real=True)

            # φ-функции (символьные)
            phi_sym = (y_s / 3,
                       x_s ** 3 / 2 - 1)

            # φ-функции (числовые)
            phi_num = (lambda x, y: y / 3,
                       lambda x, y: (x ** 3) / 2 - 1)

            xs = np.linspace(-3, 3, 400)
            plt.plot(xs, [3 * x for x in xs], label='y=3x')
            plt.plot(xs, [(x ** 3) / 2 - 1 for x in xs], label='y=x³/2 -1')
            title = 'Система 1: y=3x и y=x³/2 -1'
        else:  # система 2
            x_s, y_s = sp.symbols('x y', real=True)

            # Спрашиваем у пользователя, какую ветвь корня он хочет
            branch = input("Выберите ветвь для y = ±√(25 - x²) (введите '+'
или '-'): ").strip()
            if branch == '+':
                phi2_sym = sp.sqrt(25 - x_s ** 2)
                phi2_num = lambda x, y: np.sqrt(max(0, 25 - x ** 2))
            else:
                phi2_sym = -sp.sqrt(25 - x_s ** 2)
                phi2_num = lambda x, y: -np.sqrt(max(0, 25 - x ** 2))

            # φ₁ остаётся без изменений
            phi1_sym = sp.root(y_s + 2, 3)
            phi1_num = lambda x, y: np.cbrt(y + 2)

            phi_sym = (phi1_sym, phi2_sym)
            phi_num = (phi1_num, phi2_num)

            # Рисуем оба варианта просто для наглядности
            xs = np.linspace(-5, 5, 400)
            plt.plot(xs, [np.sqrt(max(0, 25 - x ** 2)) for x in xs],
label='+sqrt(25 - x²)')
            plt.plot(xs, [-np.sqrt(max(0, 25 - x ** 2)) for x in xs],
label='-sqrt(25 - x²)')
            plt.plot(xs, [x ** 3 - 2 for x in xs], label='y = x³ - 2')
            title = 'Система 2: x² + y² = 25 и y = x³ - 2'

        plt.title(title)
        plt.axhline(0, color='black', linewidth=0.5)  # Ось X
        plt.axvline(0, color='black', linewidth=0.5)  # Ось Y
        plt.legend()
        plt.grid(True)
```

```
        plt.axis('equal')  # Сохраняем пропорции

        # Устанавливаем желаемые границы осей
        plt.xlim(-7, 7)  # От -10 до 10 по X
        plt.ylim(-7, 7)  # От -10 до 10 по Y

        plt.show()

        x0 = float(input("Введите начальное приближение x0: "))
        y0 = float(input("Введите начальное приближение y0: "))

        sol, iters, errors, q, message = iteration_method_system(
            phi_sym, phi_num, x0, y0, tol, max_iter=1000, threshold=1e6
        )

        print(f"q = {q:.3f}. {message}")
        x_sol, y_sol = sol
        print(f"Приближение: x = {x_sol:.6f}, y = {y_sol:.6f}")
        print(f"Итераций: {iters}")
        # print("Погрешности по шагам:")
        # for idx, (dx, dy) in enumerate(errors, 1):
        #     print(f"{idx}: |Δx|={dx:.2e}, |Δy|={dy:.2e}")

        # Вычисление невязок в системе
        try:
            if choice == '1':
                r1 = y_sol - 3 * x_sol
                r2 = y_sol - (x_sol ** 3) / 2 + 1
            else:
                r1 = x_sol ** 2 + y_sol ** 2 - 25
                r2 = y_sol - x_sol ** 3 + 2
            print(f"Невязки: f1={r1:.2e}, f2={r2:.2e}")
        except Exception:
            print("Невязки не удалось вычислить из-за переполнения.")


if __name__ == "__main__":
    main()
```

Результаты выполнения программы при различных исходных данных:

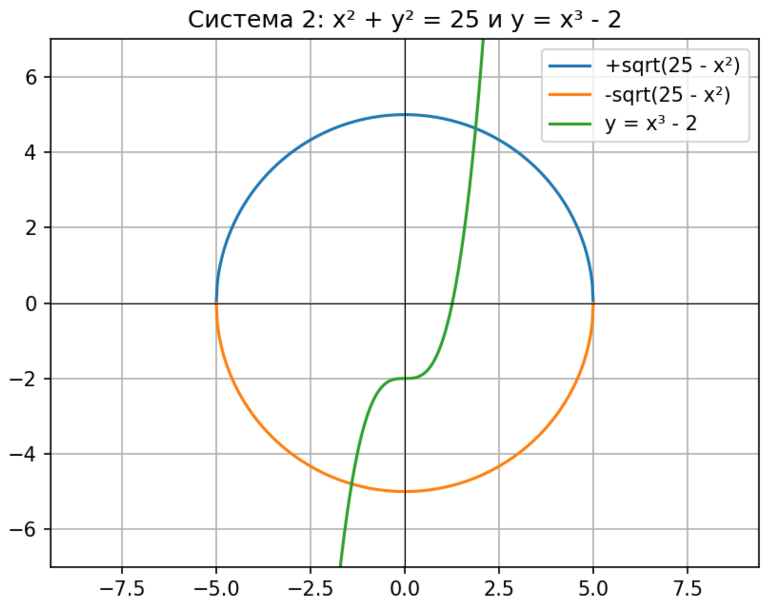
Что решаем? (equation/system): equation
Считать данные из файла или клавиатуры? (file/keyboard): keyboard
Выберите функцию для решения уравнения:
1: x^3 - x - 1
2: sin(x) - 0.5*x
3: exp(x) - 3x
Номер функции: 1
Введите границы интервала (a b): 1 5
Введите точность (например 1e-6): 0.001
Куда выводить результат? (file/console): console



Convergence condition: max|phi'(x)| = 0.210 < 1.

Метод хорд: корень = 1.324805, f(root) = 3.698650e-04, итераций = 6
Метод Ньютона: корень = 1.324718, f(root) = 1.166547e-10, итераций = 6
Метод итераций: корень = 1.324780, f(root) = 2.649422e-04, итераций = 6

Что решаем? (equation/system): system
Метод простой итерации для систем нелинейных уравнений
1) y = 3x;   y = x^3/2 - 1
2) x^2 + y^2 = 25;   y = x^3 - 2
Выберите систему (1 или 2): 2
Введите точность (например 1e-6): 1e-6
Выберите ветвь для y = ±√(25 - x²) (введите '+' или '-'): +
Введите начальное приближение x0: 4
Введите начальное приближение y0: 4



0.100951144046230   1.33333333333333
φ₁(x,y) = (y + 2)**(1/3)
φ₂(x,y) = sqrt(25 - x**2)
∂φ₁/∂x = 0
∂φ₁/∂y = 1/(3*(y + 2)**(2/3))
∂φ₂/∂x = -x/sqrt(25 - x**2)
∂φ₂/∂y = 0
4.0 4.0
1.8171205928321394 3.0
1.709975946676697 4.658119014270178
1.8812673488586602 4.698508514601961
1.8850637323963553 4.632583853760049
1.8788592609772667 4.631040350159338
1.878713503251082 4.633561036334793
1.8789515276017736 4.633620136858658

```
1.878957107645896 4.633523622139307
1.8789479950802244 4.633521359358017
1.8789477814356572 4.6335250546192155
q = 1.333. WARNING: q = 1.333 >= 1; метод может расходиться.
Процесс сошёлся.
Приближение: x = 1.878948, y = 4.633525
Итераций: 11
Невязки: f1=1.31e-06, f2=8.66e-08
```

Вывод:

В ходе выполнения лабораторной работы были изучены численные методы решения нелинейных уравнений и систем нелинейных уравнений с использованием Python. 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.