МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №3

по дисциплине: Вычислительная математика
тема: «Решение Систем Нелинейных Уравнений»

Выполнил: ст. группы ВТ-231
Масленников Даниил

Проверили:
Островский Алексей Мичеславович

Белгород 2025 г

Лабораторная работа №3

Цель работы: Изучить методы решения нелинейных систем уравнений и особенности алгоритмизации в экосистемах языков Rust и Python

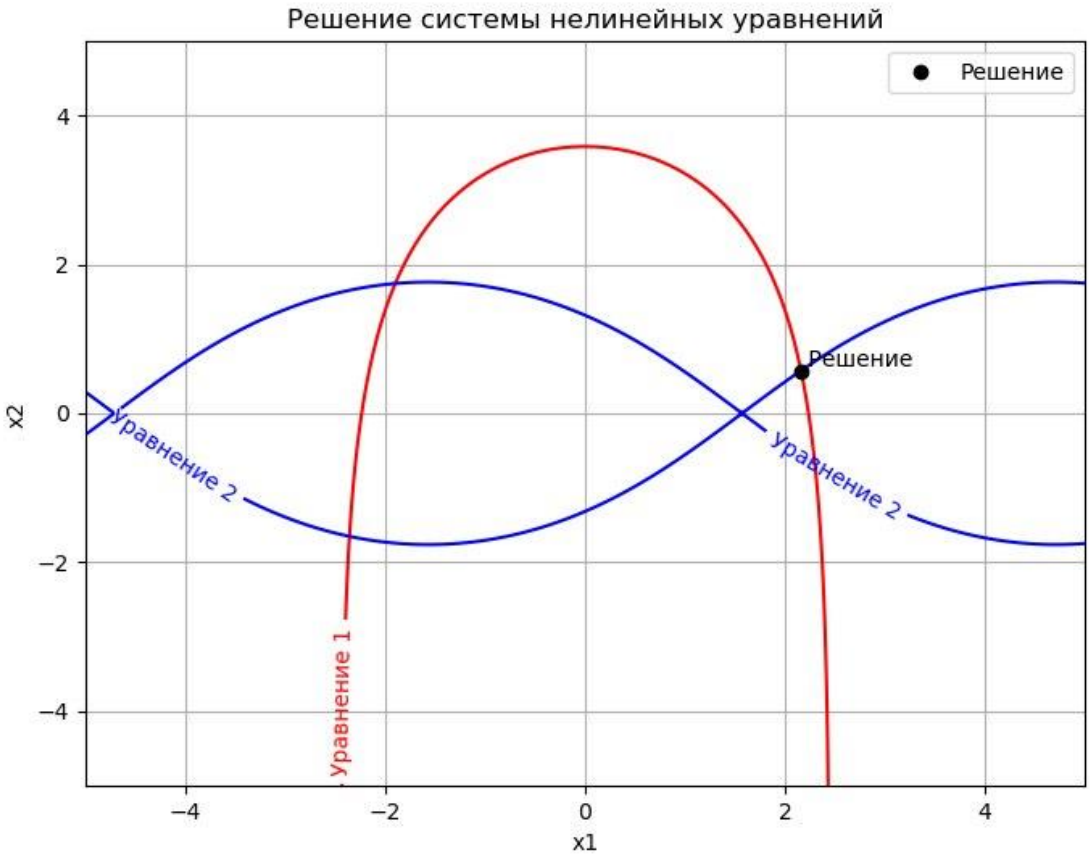
1. **Построение графиков и выбор начального приближения (Python):**
    - Построить графики нелинейных функций системы уравнений.
    - Выбрать начальное приближение для нахождения корня, ближайшего к началу координат.
    - Получить контрольное решение с использованием библиотеки scipy.optimize.fsolve.
2. **Реализация метода Ньютона (Rust):**
    - Написать программу на Rust для решения системы нелинейных уравнений методом Ньютона.
    - Использовать якобиан и обратную матрицу для обновления решения.
    - Проверить сходимость и точность.
3. **Реализация метода простой итерации (Rust):**
    - Написать программу на Rust для решения системы нелинейных уравнений методом простой итерации.
    - Преобразовать систему уравнений в итерационную форму.
    - Проверить сходимость и точность.
4. **Реализация метода градиентного спуска (Rust):**
    - Написать программу на Rust для решения системы нелинейных уравнений методом градиентного спуска.
    - Минимизировать сумму квадратов невязок системы.
    - Проверить сходимость и точность.
5. **Сравнение методов:**
    - Сравнить вычислительные схемы и результаты для методов Ньютона, простой итерации и градиентного спуска.
    - Сделать выводы о скорости сходимости, точности и применимости каждого метода.

Вариант 14

14. $$\begin{cases} x_1^2 + \sqrt{e^{x_2}} = 6 \\ \sin(x_1) + \cosh(x_2) = 2 \end{cases}$$

Задание 1: Построение графиков нелинейных функций на Python

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

def nonlinear_equations(variables):
    x1, x2 = variables
    equation1 = np.sqrt(np.exp(x2)) + x1**2 - 6
    equation2 = np.sin(x1) + np.cosh(x2) - 2
    return [equation1, equation2]

def plot_solution_and_equations(solution):
    x1_values = np.linspace(-5, 5, 400)
    x2_values = np.linspace(-5, 5, 400)
    X1, X2 = np.meshgrid(x1_values, x2_values)
    Z1 = np.sqrt(np.exp(X2)) + X1**2 - 6
    Z2 = np.sin(X1) + np.cosh(X2) - 2

    plt.figure(figsize=(8, 6))
    contour1 = plt.contour(X1, X2, Z1, levels=[0], colors='r')
    plt.clabel(contour1, inline=True, fontsize=10, fmt='Уравнение 1')
    contour2 = plt.contour(X1, X2, Z2, levels=[0], colors='b')
    plt.clabel(contour2, inline=True, fontsize=10, fmt='Уравнение 2')
    plt.plot(solution[0], solution[1], 'ko', label='Решение')
    plt.text(solution[0], solution[1], ' Решение', verticalalignment='bottom')
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Решение системы нелинейных уравнений')
    plt.grid(True)
    plt.legend()
    plt.show()

initial_guess = [1.0, 1.0]

solution = fsolve(nonlinear_equations, initial_guess)
print(f"Решение: x1 = {solution[0]}, x2 = {solution[1]}")

plot_solution_and_equations(solution)
```




```
❯ python3 main.py
Решение: x1 = 2.1605554830913785, x2 = 0.5733631509687981
```

Задание 2: Реализация следующих алгоритмов на Rust: метод Ньютона, простой итерации и градиентного спуска

```
use std::f64;

fn is_zero(n: f64, eps: f64) -> bool {
    n.abs() < eps
}

fn inverse_matrix_2x2(matrix: [[f64; 2]; 2], epsilon: f64) -> Result<[[f64; 2]; 2], &'static str> {
    let det = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
    if is_zero(det, epsilon) {
        return Err("Матрица вырождена");
    }
    Ok([
        [matrix[1][1] / det, -matrix[0][1] / det],
        [-matrix[1][0] / det, matrix[0][0] / det],
    ])
}
```

```rust
fn matrix_vector_multiply(matrix: [[f64; 2]; 2], vector: [f64; 2]) -> [f64; 2] {
    [
        matrix[0][0] * vector[0] + matrix[0][1] * vector[1],
        matrix[1][0] * vector[0] + matrix[1][1] * vector[1],
    ]
}

fn f(x: [f64; 2]) -> [f64; 2] {
    [
        x[0].powi(2) + f64::sqrt(f64::exp(x[1])) - 6.0,
        f64::sin(x[0]) + f64::cosh(x[1]) - 2.0,
    ]
}

fn jacobian(x: [f64; 2]) -> [[f64; 2]; 2] {
    [
        [2.0 * x[0], 0.5 * f64::exp(x[1]) / f64::sqrt(f64::exp(x[1]))],
        [f64::cos(x[0]), f64::sinh(x[1])],
    ]
}

fn newton_method(
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    for iteration in 0..max_iterations {
        let j = jacobian(x);
        let inv_j = inverse_matrix_2x2(j, epsilon)?;
        let fx = f(x);
        let delta = matrix_vector_multiply(inv_j, [-fx[0], -fx[1]]);
        x = [x[0] + delta[0], x[1] + delta[1]];

        if delta[0].abs() < epsilon && delta[1].abs() < epsilon {
            println!("Метод Ньютона сошелся за {} итераций", iteration);
            return Ok(x);
        }
    }
    Err("Метод Ньютона не сошелся")
}

// простая итерация
fn simple_iteration_method(
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    for iteration in 0..max_iterations {
        let x_new = [
```

```
            f64::sqrt(6.0 - f64::sqrt(f64::exp(x[1]))),
            f64::acosh(2.0 - f64::sin(x[0])),
        ];

        if (x_new[0] - x[0]).abs() < epsilon && (x_new[1] - x[1]).abs() <
epsilon {
            println!("Метод простой итерации сошелся за {} итераций",
iteration);
            return Ok(x_new);
        }
        x = x_new;
    }
    Err("Метод простой итерации не сошелся")
}

// градиентный спуск
fn gradient_descent(
    initial_guess: [f64; 2],
    learning_rate: f64,
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    for iteration in 0..max_iterations {
        let grad = gradient(x);
        x = [x[0] - learning_rate * grad[0], x[1] - learning_rate * grad[1]];

        if grad[0].abs() < epsilon && grad[1].abs() < epsilon {
            println!("Метод градиентного спуска сошелся за {} итераций",
iteration);
            return Ok(x);
        }
    }
    Err("Метод градиентного спуска не сошелся")
}

fn gradient(x: [f64; 2]) -> [f64; 2] {
    let h = 1e-6;
    let mut grad = [0.0; 2];

    for i in 0..2 {
        let mut x_plus_h = x;
        let mut x_minus_h = x;
        x_plus_h[i] += h;
        x_minus_h[i] -= h;

        grad[i] = (objective_function(x_plus_h) - objective_function(x_minus_h))
/ (2.0 * h);
    }
    grad
}
```

```rust
fn objective_function(x: [f64; 2]) -> f64 {
    let eq1 = x[0].powi(2) + f64::sqrt(f64::exp(x[1])) - 6.0;
    let eq2 = f64::sin(x[0]) + f64::cosh(x[1]) - 2.0;
    eq1.powi(2) + eq2.powi(2)
}

fn main() {
    let initial_guess = [1.0, 1.0];

    let epsilon = 1e-6;       // точность
    let max_iterations = 10000; // макс итераций
    let learning_rate = 0.01; // шаг

    match newton_method(initial_guess, epsilon, max_iterations) {
        Ok(solution) => println!("Метод Ньютона: x1 = {}, x2 = {}", solution[0],
solution[1]),
        Err(e) => println!("Метод Ньютона: {}", e),
    }

    match simple_iteration_method(initial_guess, epsilon, max_iterations) {
        Ok(solution) => println!("Метод простой итерации: x1 = {}, x2 = {}",
solution[0], solution[1]),
        Err(e) => println!("Метод простой итерации: {}", e),
    }

    match gradient_descent(initial_guess, learning_rate, epsilon,
max_iterations) {
        Ok(solution) => println!("Метод градиентного спуска: x1 = {}, x2 = {}",
solution[0], solution[1]),
        Err(e) => println!("Метод градиентного спуска: {}", e),
    }
}
```

```
❯ ./main
Метод Ньютона сошелся за 6 итераций
Метод Ньютона: x1 = 2.3011431611753337, x2 = -0.6998522463059262
Метод простой итерации сошелся за 13 итераций
Метод простой итерации: x1 = 2.160555504707232, x2 = 0.5733626028963114
Метод градиентного спуска сошелся за 1323 итераций
Метод градиентного спуска: x1 = 2.1605553364973216, x2 = 0.5733642193256366
```

Вывод: сравнив все методы пришел к выводу, что метод градиентного спуска наиболее универсален, ведь метод Ньютона требует больше ресурсов на каждой итерации, методы Ньютона и простой итерации чувствительны к выбору начального приближения. Градиентный спуск же чувствителен только к шагу, позволяя его применять наиболее часто.