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)**

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

# Лабораторная работа №3

по дисциплине: Вычислительная математика

тема: «**Решение систем нелинейных уравнений**»

Выполнил: студент группы ПВ-233
Мороз Роман Алексеевич

Проверили:

Белгород 2025 г.

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

Цель работы обуславливает постановку и решение следующих задач:

1) Рассмотреть теоретические основы решения систем нелинейных уравнений.

2) Научиться выбирать методы и алгоритмизировать решение систем нелинейных уравнений в зависимости от численной ситуации с вниманием к проблемам разрешимости, точности, численной стабильности и эффективности.

3) Выполнить индивидуальное задание, закрепляющее на практике полученные знания и практические навыки (номер задания соответствует номеру студента по журналу; если этот номер больше, чем максимальное число заданий, тогда вариант задания вычисляется по формуле: номер по журналу % максимальный номер задания, где % — остаток от деления).

- *Первая часть данного задания предполагает построение графиков нелинейных функций из системы уравнений индивидуального задания и выбор точки начального приближения для нахождения одного корня, ближайшего к началу координат.*

- *Вторая часть задания предполагает изучение алгоритмических техник программы на языке Rust для решения системы нелинейных уравнений демонстрационного и своего индивидуального задания по методу Ньютона в интерактивном блокноте Jupyter.*

- *Третья часть задания предполагает самостоятельное написание двух программ на языке Rust: для решения той же системы нелинейных уравнений (своего индивидуального задания) по методу простой итерации и по методу градиентного спуска. Необходимо сравнить вычислительные схемы и полученные результаты для разных алгоритмов между собой.*

7) Отразить в отчете все полученные результаты, включая графики, тексты программ. Сделать выводы.

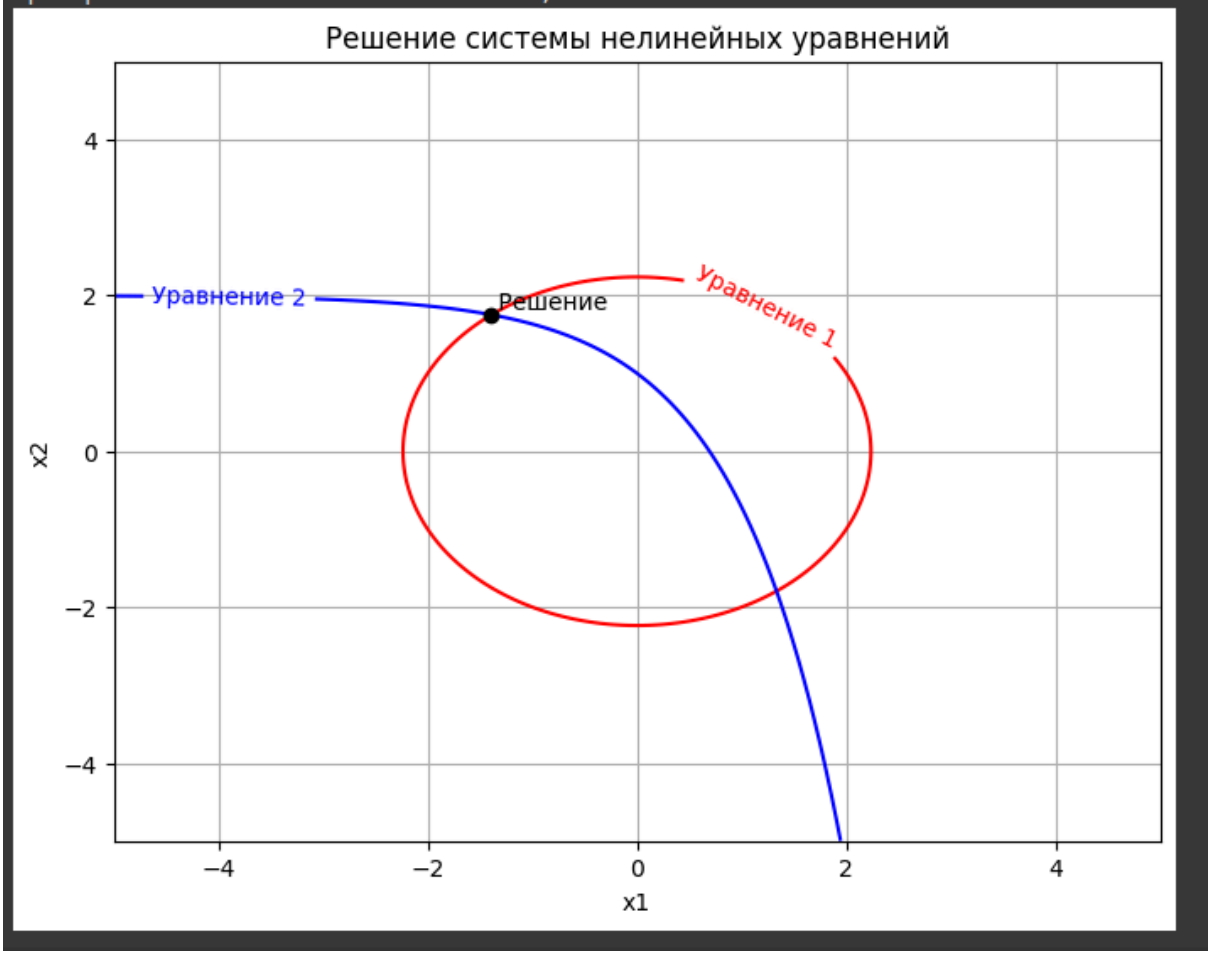
## Ход выполнения работы

### 1) Запустить демонстрационную программу на языке Python для построения графиков нелинейных функций системы уравнений:

```
def solve_nonlinear_system(initial_guess):
 """
 Решает систему нелинейных уравнений и возвращает решение.

 Параметры:
 initial_guess (list): начальное приближение решения системы уравнений.

 Возвращает:
 solution (np.ndarray): массив с найденным решением системы уравнений.
 """
 solution = fsolve(nonlinear_equations, initial_guess)
 return solution

def nonlinear_equations(variables):
 """
 Определяет систему нелинейных уравнений.

 Параметры:
 variables (list): список переменных системы уравнений (x1, x2).

 Возвращает:
 list: список, содержащий левые части системы нелинейных уравнений.
 """
 x1, x2 = variables
 equation1 = x1**2 + x2**2 - 5.
 equation2 = np.exp(x1) + x2 - 2.
 return [equation1, equation2]

def plot_solution_and_equations(solution):
 """
 Визуализирует систему нелинейных уравнений и точку решения на графике.

 Параметры:
 solution (np.ndarray): решение системы нелинейных уравнений.
```

```
 """
 x1_values = np.linspace(-5, 5, 400)
 x2_values = np.linspace(-5, 5, 400)
 X1, X2 = np.meshgrid(x1_values, x2_values)
 Z1 = X1**2 + X2**2 - 5.
 Z2 = np.exp(X1) + X2 - 2.

 plt.figure(figsize=(8, 6))

 contour1 = plt.contour(X1, X2, Z1, levels=[0], colors='r')
 contour2 = plt.contour(X1, X2, Z2, levels=[0], colors='b')

 plt.clabel(contour1, inline=1, fontsize=10, fmt='Уравнение 1')
 plt.clabel(contour2, inline=1, fontsize=10, fmt='Уравнение 2')

 # Точка решения
 plt.plot(solution[0], solution[1], 'ko') # 'ko' означает черный цвет
('k')

 # и форму точки ('o')
 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.show()


# Начальное приближение для поиска решения
initial_guess = [-1., 2.]

# Решение системы нелинейных уравнений
solution = solve_nonlinear_system(initial_guess)

# Проверка решения
solution_check = nonlinear_equations(solution)

print(f"Получено решение: x1 = {solution[0]}, x2 = {solution[1]}")
print(f"Проверка = 0: {solution_check[0]}, {solution_check[1]}")

# Построение графика системы уравнений
plot_solution_and_equations(solution)
```

```
Получено решение: x1 = -1.3905921026665142, x2 = 1.7510721298674494
Проверка = 0: -2.996713988068225e-12, -4.3010039973978564e-13
```




**2) Для своего индивидуального задания следует построить графики нелинейных функций, выбрать точку начального приближения для нахождения одного корня, ближайшего к началу координат и получить контрольное решение, которое использовать далее для проверки программ на языке Rust.**

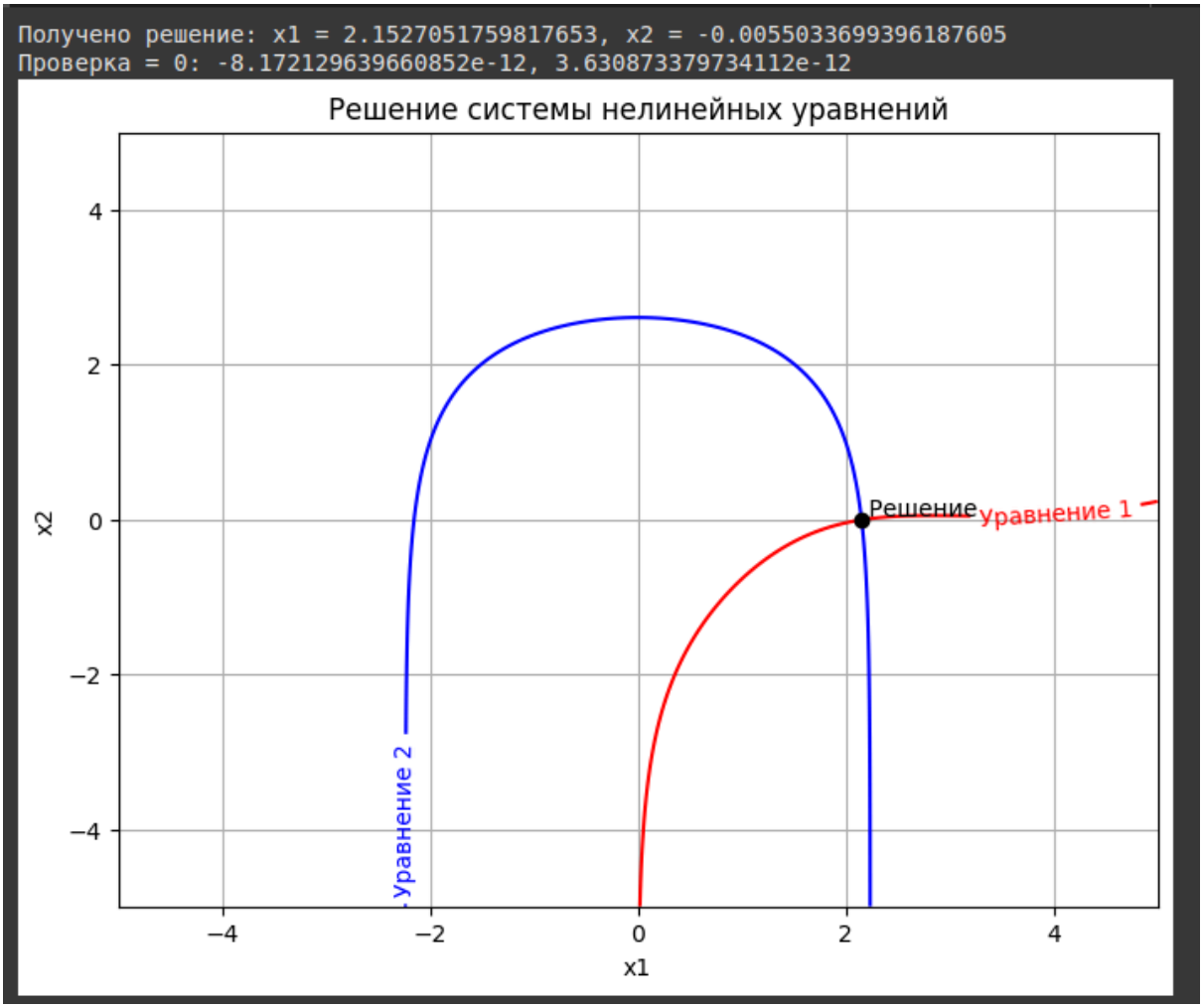
9. $$\begin{cases} x_1 \cdot \exp(-x_2) + \sin(x_1) = 3 \\ x_1^2 + \exp(x_2 - 1) = 5 \end{cases}$$

```
def solve_nonlinear_system(initial_guess):
 """
 Решает систему нелинейных уравнений и возвращает решение.

 Параметры:
 initial_guess (list): начальное приближение решения системы уравнений.

 Возвращает:
 solution (np.ndarray): массив с найденным решением системы уравнений.
```

```
    """
    solution = fsolve(nonlinear_equations, initial_guess)
    return solution

def nonlinear_equations(variables):
    """
    Определяет систему нелинейных уравнений.

    Параметры:
    variables (list): список переменных системы уравнений (x1, x2).

    Возвращает:
    list: список, содержащий левые части системы нелинейных уравнений.
    """
    x1, x2 = variables
    equation1 = x1*np.exp(-x2) + np.sin(x1) - 3
    equation2 = x1**2 + np.exp(x2 - 1) - 5
    return [equation1, equation2]

def plot_solution_and_equations(solution):
    """
    Визуализирует систему нелинейных уравнений и точку решения на графике.

    Параметры:
    solution (np.ndarray): решение системы нелинейных уравнений.
    """
    x1_values = np.linspace(-5, 5, 400)
    x2_values = np.linspace(-5, 5, 400)
    X1, X2 = np.meshgrid(x1_values, x2_values)
    Z1 = X1*np.exp(-X2) + np.sin(X1) - 3
    Z2 = X1**2 + np.exp(X2 - 1) - 5

    plt.figure(figsize=(8, 6))

    contour1 = plt.contour(X1, X2, Z1, levels=[0], colors='r')
    contour2 = plt.contour(X1, X2, Z2, levels=[0], colors='b')

    plt.clabel(contour1, inline=1, fontsize=10, fmt='Уравнение 1')
    plt.clabel(contour2, inline=1, fontsize=10, fmt='Уравнение 2')

    # Точка решения
    plt.plot(solution[0], solution[1], 'ko') # 'ko' означает черный цвет
('k')
```

```
 # и форму точки ('o')
 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.show()


# Начальное приближение для поиска решения
initial_guess = [-1., 2.]

# Решение системы нелинейных уравнений
solution = solve_nonlinear_system(initial_guess)

# Проверка решения
solution_check = nonlinear_equations(solution)

print(f"Получено решение: x1 = {solution[0]}, x2 = {solution[1]}")
print(f"Проверка = 0: {solution_check[0]}, {solution_check[1]}")

# Построение графика системы уравнений
plot_solution_and_equations(solution)
```

```
Получено решение: x1 = 2.1527051759817653, x2 = -0.0055033699396187605
Проверка = 0: -8.172129639660852e-12, 3.630873379734112e-12
```




**3) В интерактивном блокноте Jupyter инсталлировать Rust, используя следующую инструкцию:**

```
# Установка Rust
!curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s --
-y
import os
os.environ['PATH'] += ":/root/.cargo/bin"
```

```
info: downloading installer
info: profile set to 'default'
info: default host triple is x86_64-unknown-linux-gnu
info: syncing channel updates for 'stable-x86_64-unknown-linux-gnu'
info: latest update on 2025-02-20, rust version 1.85.0 (4d91de4e4 2025-02-17)
info: downloading component 'cargo'
info: downloading component 'clippy'
info: downloading component 'rust-docs'
info: downloading component 'rust-std'
info: downloading component 'rustc'
info: downloading component 'rustfmt'
info: installing component 'cargo'
  8.7 MiB /   8.7 MiB (100 %)   7.0 MiB/s in  1s
info: installing component 'clippy'
info: installing component 'rust-docs'
 18.2 MiB /  18.2 MiB (100 %)   1.7 MiB/s in  7s
info: installing component 'rust-std'
 26.7 MiB /  26.7 MiB (100 %)   8.2 MiB/s in  3s
info: installing component 'rustc'
 69.5 MiB /  69.5 MiB (100 %)   9.3 MiB/s in  8s
info: installing component 'rustfmt'
info: default toolchain set to 'stable-x86_64-unknown-linux-gnu'

  stable-x86_64-unknown-linux-gnu installed - rustc 1.85.0 (4d91de4e4 2025-02-17)


Rust is installed now. Great!

To get started you may need to restart your current shell.
This would reload your PATH environment variable to include
Cargo's bin directory ($HOME/.cargo/bin).

To configure your current shell, you need to source
the corresponding env file under $HOME/.cargo.

This is usually done by running one of the following (note the leading DOT):
. "$HOME/.cargo/env"            # For sh/bash/zsh/ash/dash/pdksh
source "$HOME/.cargo/env.fish"  # For fish
source "$HOME/.cargo/env.nu"    # For nushell
```

```
# Проверка установки Rust
!rustc --version
```

```
rustc 1.85.0 (4d91de4e4 2025-02-17)
```

**4) Запустить демонстрационную программу для решения системы нелинейных уравнений по методу Ньютона в интерактивном блокноте Jupyter.**

```
%%writefile lab3_newton.rs

use std::f64;
```

```rust
// Функция для проверки равенства нулю с учетом погрешности
fn is_zero(n: f64, eps: f64) -> bool {
    n.abs() < eps
}

// Функция для вычисления обратной матрицы 2x2
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 является вырожденной и не имеет обратной.");
    }

    let inv_matrix = [
        [matrix[1][1] / det, -matrix[0][1] / det],
        [-matrix[1][0] / det, matrix[0][0] / det],
    ];

    Ok(inv_matrix)
}

// Функция умножения матрицы на вектор
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],
    ]
}

// Функция для задания системы уравнений
fn f(x: [f64; 2]) -> [f64; 2] {
    [
        x[0] * f64::exp(-x[1]) + f64::sin(x[0]) - 5.0,
        f64::exp(x[0]) + x[1] - 2.0
    ]
}

// Функция для задания Якобиана
fn jacobian(x: [f64; 2]) -> [[f64; 2]; 2] {
    [
        [
            f64::exp(-x[1]) + f64::cos(x[0]),
```

```
            -x[0] * f64::exp(-x[1])
        ],
        [
            f64::exp(x[0]),
            1.0
        ]
    ]
}

// Бесконечная норма
fn norm(vector: [f64; 2]) -> f64 {
    vector.iter().map(|&v| v.abs()).fold(0.0, f64::max)
}

// Метод Ньютона
fn newton_method(
    f: fn([f64; 2]) -> [f64; 2],
    jacobian: fn([f64; 2]) -> [[f64; 2]; 2],
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;

    for _ 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 norm(delta) < epsilon {
            return Ok(x);
        }
    }

    Err("Алгоритм не сошелся")
}

fn main() {
    let epsilon = 1e-8; // Задаем значение epsilon
    let initial_guess = [-1.0, 2.0];
```

```
    match newton_method(f, jacobian, initial_guess, epsilon, 100) {
        Ok(solution) => println!("Решение: {:?}", solution),
        Err(e) => println!("{}", e),
    }
}
```

```
# Компиляция
!rustc lab3_newton.rs
```

```
# Запуск скомпилированной программы
!./lab3_newton
```

```
Решение: [1.176439856868779, -1.242808764448568]
```

**5) Модифицировать и отладить программу для выполнения своего индивидуального**

```
%%writefile lab5_newton.rs

use std::f64;

// Функция для проверки равенства нулю с учетом погрешности
fn is_zero(n: f64, eps: f64) -> bool {
    n.abs() < eps
}

// Функция для вычисления обратной матрицы 2x2
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 является вырожденной и не имеет обратной.");
    }

    let inv_matrix = [
        [matrix[1][1] / det, -matrix[0][1] / det],
```

```
        [-matrix[1][0] / det, matrix[0][0] / det],
    ];

    Ok(inv_matrix)
}

// Функция умножения матрицы на вектор
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],
    ]
}

// Функция для задания системы уравнений
fn f(x: [f64; 2]) -> [f64; 2] {
    [
        x[0] * f64::exp(-x[1]) + f64::sin(x[0]) - 3.0,
        x[0].powi(2) + f64::exp(x[1] - 1.0) - 5.0
    ]
}

// Функция для задания Якобиана
fn jacobian(x: [f64; 2]) -> [[f64; 2]; 2] {
    [
        [
            f64::exp(-x[1]) + f64::cos(x[0]),
            -x[0] * f64::exp(-x[1])
        ],
        [
            2.0 * x[0],
            f64::exp(x[1] - 1.0)
        ]
    ]
}

// Бесконечная норма
fn norm(vector: [f64; 2]) -> f64 {
    vector.iter().map(|&v| v.abs()).fold(0.0, f64::max)
}

// Метод Ньютона
```

```rust
fn newton_method(
    f: fn([f64; 2]) -> [f64; 2],
    jacobian: fn([f64; 2]) -> [[f64; 2]; 2],
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;

    for _ 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 norm(delta) < epsilon {
            return Ok(x);
        }
    }

    Err("Алгоритм не сошелся")
}

fn main() {
    let epsilon = 1e-8; // Задаем значение epsilon
    let initial_guess = [1.0, 1.0];

    match newton_method(f, jacobian, initial_guess, epsilon, 100) {
        Ok(solution) => println!("Решение: {:?}", solution),
        Err(e) => println!("{}", e),
    }
}
```

```
# Компиляция
!rustc lab5_newton.rs
```

```
# Запуск скомпилированной программы
!./lab5_newton
```

```
!rustc lab5_newton.rs

# Запуск скомпилированной программы
!./lab5_newton
```

```
Решение: [2.152705175981252, -0.005503369943502421]
```

**6) Написать и отладить две программы на языке Rust для решения той же системы нелинейных уравнений (своего индивидуального задания) по методу простой итерации и по методу градиентного спуска.**

```rust
%%writefile lab6_iter.rs

use std::f64;

fn simple_iteration(
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    let relaxation = 0.9;

    for _ in 0..max_iterations {
        let fx = [
            x[0] * f64::exp(-x[1]) + f64::sin(x[0]) - 3.0,
            x[0].powi(2) + f64::exp(x[1] - 1.0) - 5.0
        ];

        let denom1 = f64::exp(-x[1]) + f64::cos(x[0]);
        let denom2 = f64::exp(x[1] - 1.0);

        let delta = [
            -relaxation * fx[0] / denom1,
            -relaxation * fx[1] / denom2
        ];

        x[0] += delta[0];
        x[1] += delta[1];

        if delta[0].abs().max(delta[1].abs()) < epsilon {
```

```
            return Ok(x);
        }
    }

    Err("Метод не сошелся")
}

fn main() {
    let epsilon = 1e-8;
    let initial_guess = [1.5, 0.5];

    match simple_iteration(initial_guess, epsilon, 1000) {
        Ok([x1, x2]) => {
            let res = [
                x1 * f64::exp(-x2) + f64::sin(x1) - 3.0,
                x1.powi(2) + f64::exp(x2 - 1.0) - 5.0
            ];
            println!("Решение: x1 = {:.5}, x2 = {:.5}", x1, x2);
            println!("Невязки: {:.?}, {:.?}", res[0], res[1]);
        },
        Err(e) => println!("Ошибка: {}", e),
    }
}
```

```
38] # Запуск скомпилированной программы
    !./lab6_iter

    Решение: x1 = 2.152705174941164, x2 = -0.005503366567627396
    Проверка: f1 = -7.781542521456686e-9, f2 = -3.242906387868061e-9
```

```
%%writefile lab6_gd.rs

use std::f64;

fn f(x: [f64; 2]) -> [f64; 2] {
    [
        x[0] * f64::exp(-x[1]) + f64::sin(x[0]) - 3.0,
        x[0].powi(2) + f64::exp(x[1] - 1.0) - 5.0
    ]
}

fn jacobian(x: [f64; 2]) -> [[f64; 2]; 2] {
```

```
    [
        [
            f64::exp(-x[1]) + f64::cos(x[0]),
            -x[0] * f64::exp(-x[1])
        ],
        [
            2.0 * x[0],
            f64::exp(x[1] - 1.0)
        ]
    ]
}

fn gradient_descent(
    initial_guess: [f64; 2],
    epsilon: f64,
    max_iterations: usize,
) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    let mut alpha = 0.1;
    let mut prev_loss = f64::INFINITY;

    for _ in 0..max_iterations {
        let fx = f(x);
        let j = jacobian(x);

        let grad = [
            2.0 * (fx[0]*j[0][0] + fx[1]*j[1][0]),
            2.0 * (fx[0]*j[0][1] + fx[1]*j[1][1])
        ];

        let x_new = [x[0] - alpha * grad[0], x[1] - alpha * grad[1]];

        let fx_new = f(x_new);
        let new_loss = fx_new[0].powi(2) + fx_new[1].powi(2);

        if new_loss < prev_loss {
            alpha *= 1.2;
            x = x_new;
            prev_loss = new_loss;
        } else {
            alpha *= 0.5;
        }
```

```
        if grad[0].hypot(grad[1]) < epsilon || new_loss <
epsilon.powi(2) {
            return Ok(x);
        }
    }

    Err("Метод не сошелся за указанное число итераций")
}

fn main() {
    let epsilon = 1e-8;
    let initial_guess = [1.5, 0.5];

    match gradient_descent(initial_guess, epsilon, 10000) {
        Ok([x1, x2]) => {
            let res = f([x1, x2]);
            println!("Решение: x1 = {:.?}, x2 = {:.?}", x1, x2);
            println!("Проверка: f1 = {:.?}, f2 = {:.?}", res[0],
res[1]);
        },
        Err(e) => println!("Ошибка: {}", e),
    }
}
```

```
# Запуск скомпилированной программы
!./lab6_gd
```

```
Решение: x1 = 2.152705174941164, x2 = -0.005503366567627396
Проверка: f1 = -7.781542521456686e-9, f2 = -3.242906387868061e-9
```

**Вывод: изучили методы решения систем нелинейных уравнений и особенности их алгоритмизации в экосистемах языков Python и Rust.**