

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: ст. группы ПВ-233

Ситников Алексей Павлович

Проверил:

Горбов Даниил Игоревич

Белгород 2025 г.

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

### Вариант 13

$$\begin{cases} e^{x_1} - \log(|x_2| + 1) = 2 \\ \tan(x_1) \cdot \cos(x_2) + \sqrt{|x_1|} = 1 \end{cases}$$

Программа для построения графиков нелинейных функций:

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

def solve_nonlinear_system(initial_guess):
    solution = fsolve(nonlinear_equations, initial_guess)
    return solution

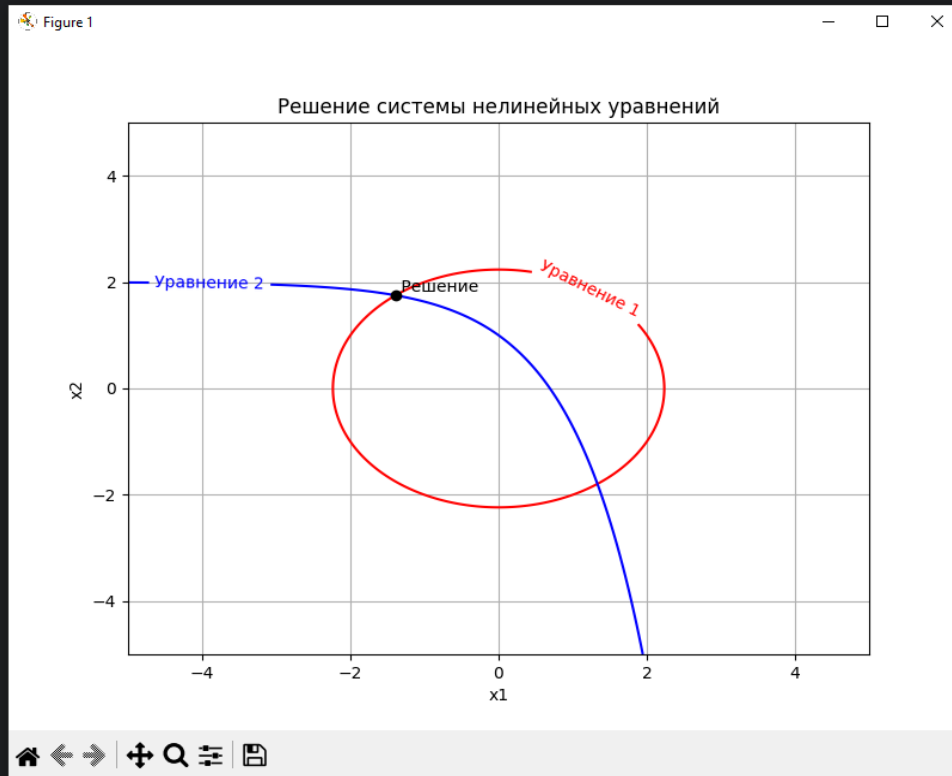
def nonlinear_equations(variables):
    x1, x2 = variables
    equation1 = x1**2 + x2**2 - 5.
    equation2 = np.exp(x1) + x2 - 2.
    return [equation1, equation2]

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 = 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)
```

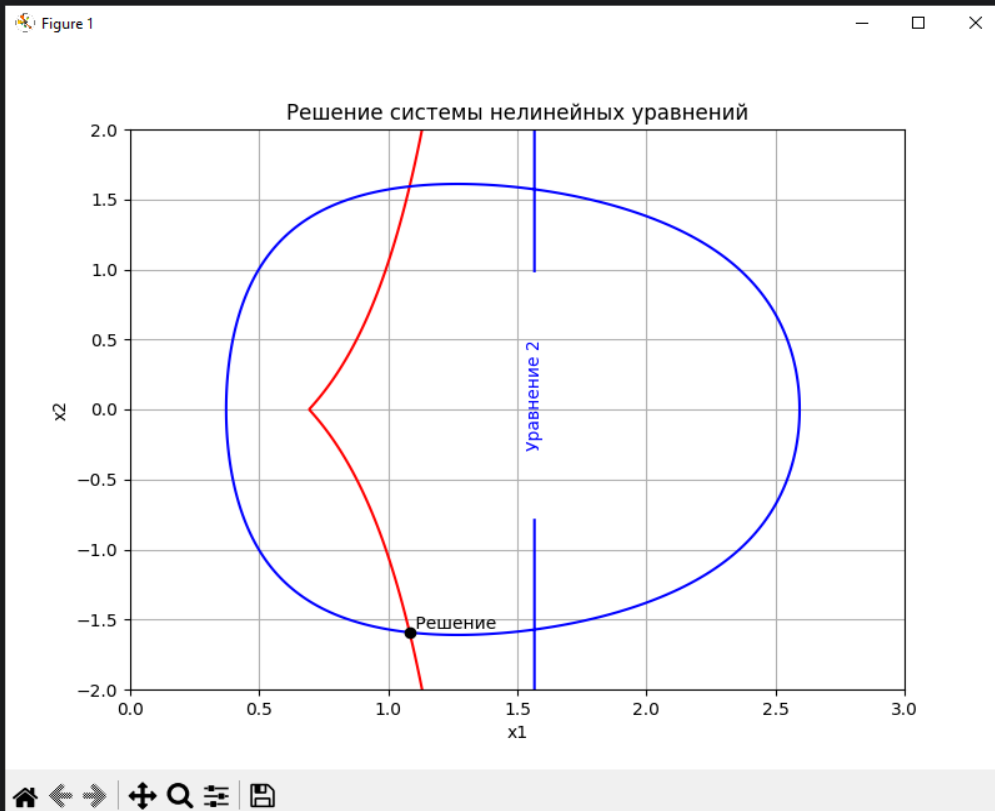
## Вывод программы с тестовыми значениями:

```
C:\Users\admin\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\admin\PycharmProjects\pythonProject\main.py
Получено решение: x1 = -1.3905921026665145, x2 = 1.7510721298674494
Проверка = 0: -2.9967139880682225e-12, -4.3010039973978564e-13
```



## Вывод программы с индивидуальными значениями:

```
C:\Users\admin\PycharmProjects\pythonProject2\.venv\Scripts\python.exe C:\Users\admin\PycharmProjects\pythonProject\main.py
Получено решение: x1 = 1.0826690194410598, x2 = -1.5923102222217416
Проверка = 0: -4.884981308350689e-15, -4.440892098500626e-15
```



## Решения системы нелинейных уравнений по методу Ньютона на языке Rust:

```
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] {
    [f64::exp(x[0]) - (x[1].abs() + 1.).log(2.) - 2., x[0].tan()*x[1].cos() + x[0].abs().sqrt() - 1.]
}

// Функция для задания Якобиана
fn jacobian(x: [f64; 2]) -> [[f64; 2]; 2] {
    [[f64::exp(x[0]) , x[1]/2.0_f64.ln()*x[1].abs() + 2.0_f64.ln()*x[1].powi(2)],
     [(x[1].cos()/x[0].cos().powi(2)) + (x[0])/(2.*x[0].abs().powf(1.5)),
      x[1].sin()*x[0].tan()*(-1.)]]
}

// Бесконечная норма
fn norm(vector: [f64; 2]) -> f64 {
    vector.iter().map(|&v| v.abs()).fold(0., f64::max)
}

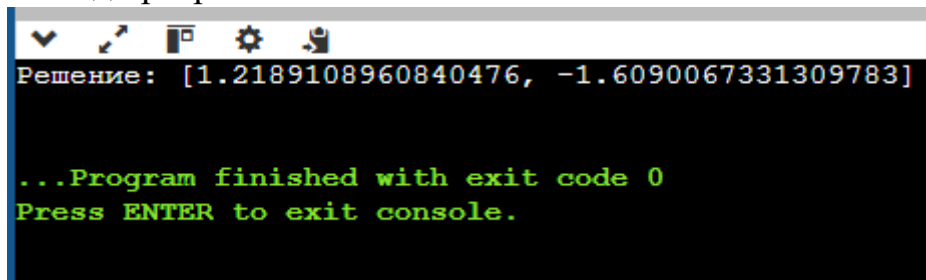
// Метод Ньютона
fn newton_method(f: fn([f64; 2]) -> [f64; 2], jacobian: fn([f64; 2]) -> [[f64; 2]; 2], initial_guess: [f64; 2], epsilon: f64, max_iterations: usize) -> Result<[f64; 2], &'static str> {
    let mut x = initial_guess;
    for _ in 0..max_iterations {
```

```

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-6; // Задаем значение epsilon
  let initial_guess = [1., -1.5];
  match newton_method(f, jacobian, initial_guess, epsilon, 100) {
    Ok(solution) => println!("Решение: {:?}", solution),
    Err(e) => println!("{}", e),
  }
}

```

Вывод программы:



```

Решение: [1.2189108960840476, -1.6090067331309783]

...Program finished with exit code 0
Press ENTER to exit console.

```

Совпало с вычисленным на питоне с использованием «fsolve»

Метод простых итераций:

```

use std::f64;

fn simpleIteration(mut arr: [f64; 2]){
  for _ in 0..1000{

    let x1 = ((arr[1].abs() + 1.).log(2.) + 2.).ln();
    let x2 = ((1. - arr[0].abs().sqrt())/arr[0].tan()).acos();
    if (x1 - arr[0]).abs() < 1e-6 && (x2 - arr[1]).abs() < 1e-6{
      println!("Решение: [{}, {}]", x1, x2);
      return;
    }

    arr[0] = x1;
    arr[1] = x2;
  }
}

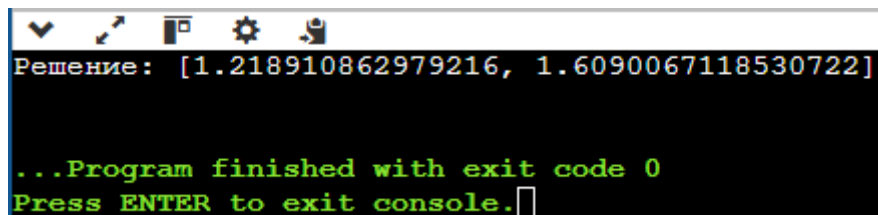
```

```

    }
    println!("Алгоритм не сошелся");
}

fn main() {
    let arr = [1., -1.5];
    simpleIteration(arr);
}

```



```

Решение: [1.218910862979216, 1.6090067118530722]

...Program finished with exit code 0
Press ENTER to exit console.

```

Найдено верхнее пересечение, которое симметрично относительно абсциссы к предыдущим решениям.

Метод градиентного спуска:

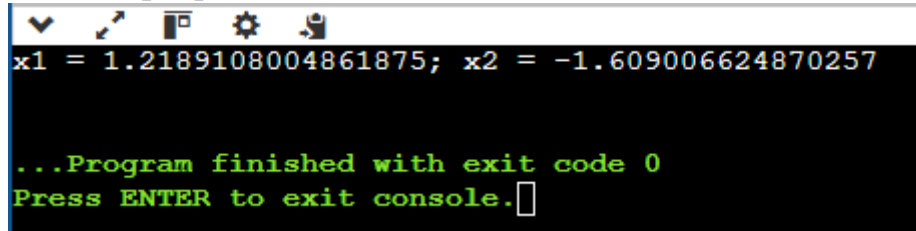
```

use std::f64;
fn main() {
    let epsilon = 1e-6;
    let mut x1: f64 = 1.0;
    let mut x2: f64 = -1.5;
    let a = 0.01;

    while true{
        let E1 = (f64::exp(x1) - (x2.abs() + 1.).log(2.) - 2.)*f64::exp(x1) +
(x1.tan()*x2.cos() + (x1.abs()).sqrt() - 1.) * ((x2.cos()/x1.cos().powi(2)) +
(x1)/(2.*x1.abs().powf(1.5)));
        let E2 = (f64::exp(x1) - (x2.abs() + 1.).log(2.) -
2.)*x2/(2.0_f64.ln()*x2.abs() + 2.0_f64.ln()*x2.powi(2)) + (x1.tan()*x2.cos() +
(x1.abs()).sqrt() - 1.)*x2.sin()*x1.tan()*(-1.);
        x1 = x1 - a*E1;
        x2 = x2 - a*E2;
        if E1 < epsilon && E2 < epsilon{
            break;
        }
    }
    println!("x1 = {}; x2 = {}", x1, x2);
}

```

Вывод программы:

A screenshot of a Windows command prompt window. The title bar shows standard Windows icons. The console output is as follows:

```
x1 = 1.2189108004861875; x2 = -1.609006624870257

...Program finished with exit code 0
Press ENTER to exit console.
```

Ответ сошёлся.

**Вывод:** в ходе проделанной работы, я изучил способы решения нелинейных уравнений.