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

Изображение выглядит как текст, Шрифт, белый, каллиграфия

Контент, сгенерированный ИИ, может содержать ошибки.

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:  
Изображение выглядит как текст, снимок экрана, диаграмма, График

Контент, сгенерированный ИИ, может содержать ошибки.

Вывод программы с индивидуальными значениями:

Изображение выглядит как текст, диаграмма, График, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

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:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

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);

}

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

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:  
Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Ответ сошёлся.

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