Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
**"Национальный исследовательский университет
Высшая школа экономики"**
Департамент прикладной математики, бакалавр

Лабораторная работа:

По теме: РЕШЕНИЕ СИСТЕМ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ.

Выполнил:
Колодин Матвей Алексеевич

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2024

Содержание

1 Задача 4.1.5 **3**
1.1 Постановка задачи 3
1.2 Код 3
1.3 Результаты 5

2 Задача 4.2.2 **6**
2.1 Постановка задачи 6
2.2 Код 6
2.3 Результаты 8

1 Задача 4.1.5

1.1 Постановка задачи

Найти с точностью $\varepsilon = 10^{-6}$ все корни системы нелинейных уравнений

$$f_1(x_1, x_2) = 0,$$
$$f_2(x_1, x_2) = 0,$$

используя метод Ньютона для системы нелинейных уравнений.
ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Используя встроенные функции, локализовать корни системы уравнений графически.

2. Написать программу-функцию, вычисляющую корень системы двух нелинейных уравнений по методу Ньютона с точностью ϵ. Предусмотреть подсчет количества итераций. Для решения соответствующей системы линейных алгебраических уравнений использовать встроенную функцию.

3. Используя написанную программу, вычислить все корни заданной системы с точностью ε.

4. На основе вычисленного вектора d построить гистограмму. По гистограмме определить компоненту b_m вектора b, которая оказывает наибольшее влияние на погрешность решения.

5. Используя встроенные функции, найти все корни системы с точностью ε. Сравнить с результатами, полученнными в п. 3.

УКАЗАНИЕ. В п. 1 привести уравнения системы к виду $x_2 = g_i(x_1)$ (либо $x_1 = g_i(x_2)$), $i = 1, 2$

4.1.5	$\sin(x_1 + 1.5) - x_2 + 2.9 = 0$ $\cos(x_2 - 2) + x_1 = 0$

Рис. 1: Условия для варианта №5

1.2 Код

```
import numpy as np
import matplotlib.pyplot as plt

# Определение функций системы и их Якобиана
```

```
def system_of_equations(variables):
    x1, x2 = variables
    return np.array([
        np.sin(x1 + 1.5) - x2 + 2.9,
        np.cos(x2 - 2) + x1
    ])

def jacobian(variables):
    x1, x2 = variables
    return np.array([
        [np.cos(x1 + 1.5), -1],
        [1, -np.sin(x2 - 2)]
    ])

# Визуализация системы уравнений
def visualize_system():
    x = np.linspace(-10, 10, 400)
    y = np.linspace(-10, 10, 400)
    X, Y = np.meshgrid(x, y)
    F1 = np.sin(X + 1.5) - Y + 2.9
    F2 = np.cos(Y - 2) + X
    plt.figure(figsize=(8, 6))
    plt.contour(X, Y, F1, levels=[0], colors='r')
    plt.contour(X, Y, F2, levels=[0], colors='b')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title('Графическая локализация корней системы уравнений')
    plt.grid(True)
    plt.show()

visualize_system()

# Метод Ньютона для системы нелинейных уравнений
def newton_method(F, J, x0, eps=1e-6, max_iter=100):
    x = x0
    for i in range(max_iter):
        Fx = np.array(F(x))  # Преобразование списка в массив NumPy
        if np.linalg.norm(Fx, ord=np.inf) < eps:
            return x, i+1  # Возвращает найденное решение и количество ите
        Jx = np.array(J(x))  # Аналогично преобразование для якобиана
        Dx = np.linalg.solve(Jx, -Fx)
        x = x + Dx
    return x, max_iter
```

```
# Начальные приближения, выбранные на основе визуализации
initial_guesses = [
    np.array([-1.5, 0]),  # Первое начальное приближение около точки перес
    np.array([1.5, -1])   # Второе начальное приближение около другой точк
]

# Применение метода Ньютона к системе уравнений
roots = []
iterations = []
for initial_guess in initial_guesses:
    root, iter_count = newton_method(system_of_equations, jacobian, initia
    roots.append(root)
    iterations.append(iter_count)

# Вывод найденных корней и количества итераций
for i, (root, iter_count) in enumerate(zip(roots, iterations)):
    print(f"Корень {i+1}: {root} найден за {iter_count} итераций.")
```

1.3 Результаты

Результаты получились следующие:

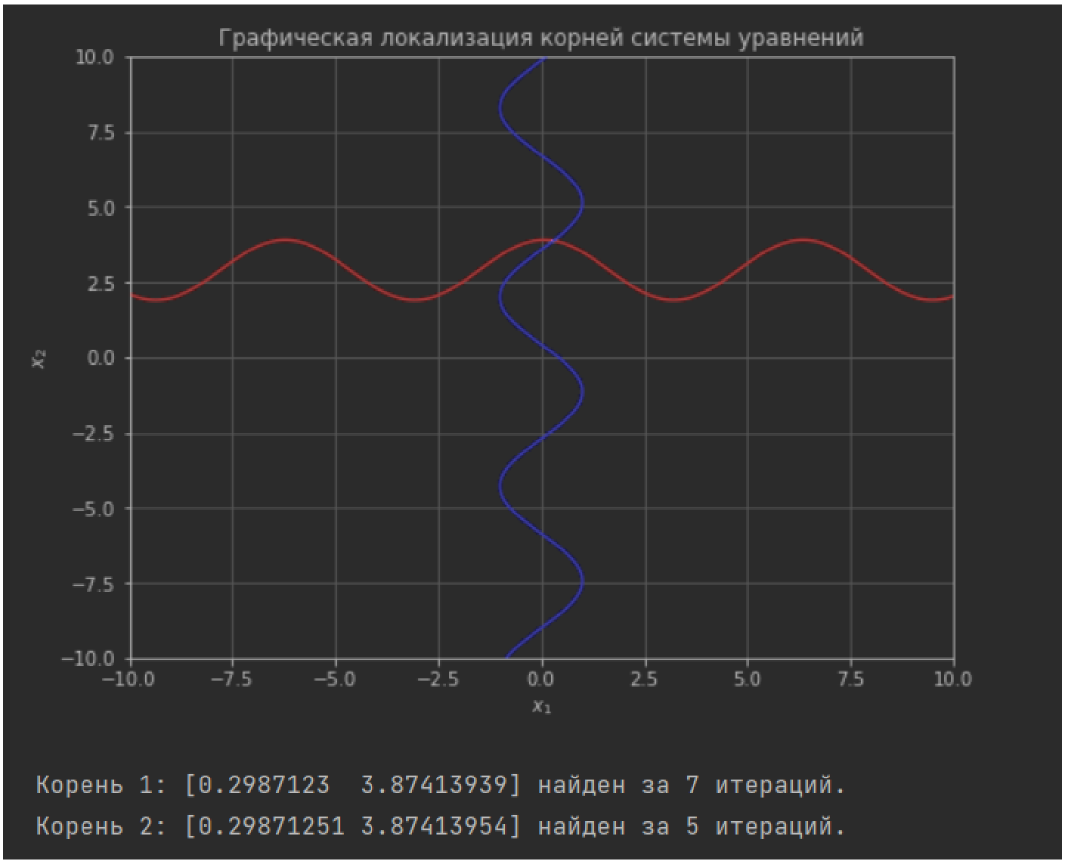

Рис. 2: Итоговые результаты по задаче

2 Задача 4.2.2

2.1 Постановка задачи

Локализовать корни системы уравнений

$$f_1(x_1, x_2, \alpha) = 0,$$
$$f_2(x_1, x_2, \alpha) = 0$$

при трех значениях параметра α. Уточнить их с точностью $\varepsilon = 10^{-5}$, используя упрощенный метод Ньютона для решения системы нелинейных уравнений.

N	$f_1(x_1,x_2,\alpha)$	$f_2(x_1,x_2,\alpha)$	α
4.2.2	$x_1^2 - x_2 + \alpha$	$-x_1 + x_2^2 + \alpha$	2, 0.25, -0.25

Рис. 3: Условия для варианта №5

2.2 Код

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

# Определение функций системы и их Якобиана
def system_of_equations_with_alpha(variables, alpha):
    x1, x2 = variables
    return np.array([
        x1**2 - x2 + alpha,
        -x1 + x2**2 + alpha
    ])

def jacobian_with_alpha(variables, alpha):
    x1, x2 = variables
    return np.array([
        [2*x1, -1],
        [-1, 2*x2]
    ])

# Визуализация системы уравнений для локализации корней
def visualize_system_with_alpha(alpha, xlim=(-3, 3), ylim=(-3, 3)):
    x = np.linspace(xlim[0], xlim[1], 400)
    y = np.linspace(ylim[0], ylim[1], 400)
    X, Y = np.meshgrid(x, y)
    F1 = X**2 - Y + alpha
```

```
    F2 = -X + Y**2 + alpha
    plt.figure(figsize=(8, 6))
    plt.contour(X, Y, F1, levels=[0], colors='r')
    plt.contour(X, Y, F2, levels=[0], colors='b')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title(f'Графическая локализация корней системы уравнений при  = {a
    plt.grid(True)
    plt.show()

# Упрощенный метод Ньютона для системы нелинейных уравнений
def simplified_newton_method(F, J, x0, alpha, eps=1e-5, max_iter=100):
    x = x0
    Jx = J(x, alpha)  # Вычисление якобиана только один раз
    for i in range(max_iter):
        Fx = F(x, alpha)
        if np.linalg.norm(Fx, ord=np.inf) < eps:
            return x, i+1  # Возвращает найденное решение и количество ите
        try:
            Dx = np.linalg.solve(Jx, -Fx)
        except np.linalg.LinAlgError as e:
            print(f"Не удалось найти корень при  = {alpha}: {e}")
            return None, i+1
        x = x + Dx
    return x, max_iter

# Значения параметра alpha
alphas = [2, 0.25, -0.25]

# Применение метода Ньютона для каждого значения alpha
roots_alpha = {}
for alpha in alphas:
    visualize_system_with_alpha(alpha)
    initial_guess = np.array([0, 0])  # Начальное приближение в центре
    root, iter_count = simplified_newton_method(system_of_equations_with_a
    roots_alpha[alpha] = (root, iter_count)
    if not np.any(np.isnan(root)):
        print(f"Корень для  = {alpha}: {root}, найден за {iter_count} итер
    else:
        print(f"Корень для  = {alpha} не найден.")

# Вывод результатов
roots_alpha
```

2.3 Результаты

Результаты получились следующие:

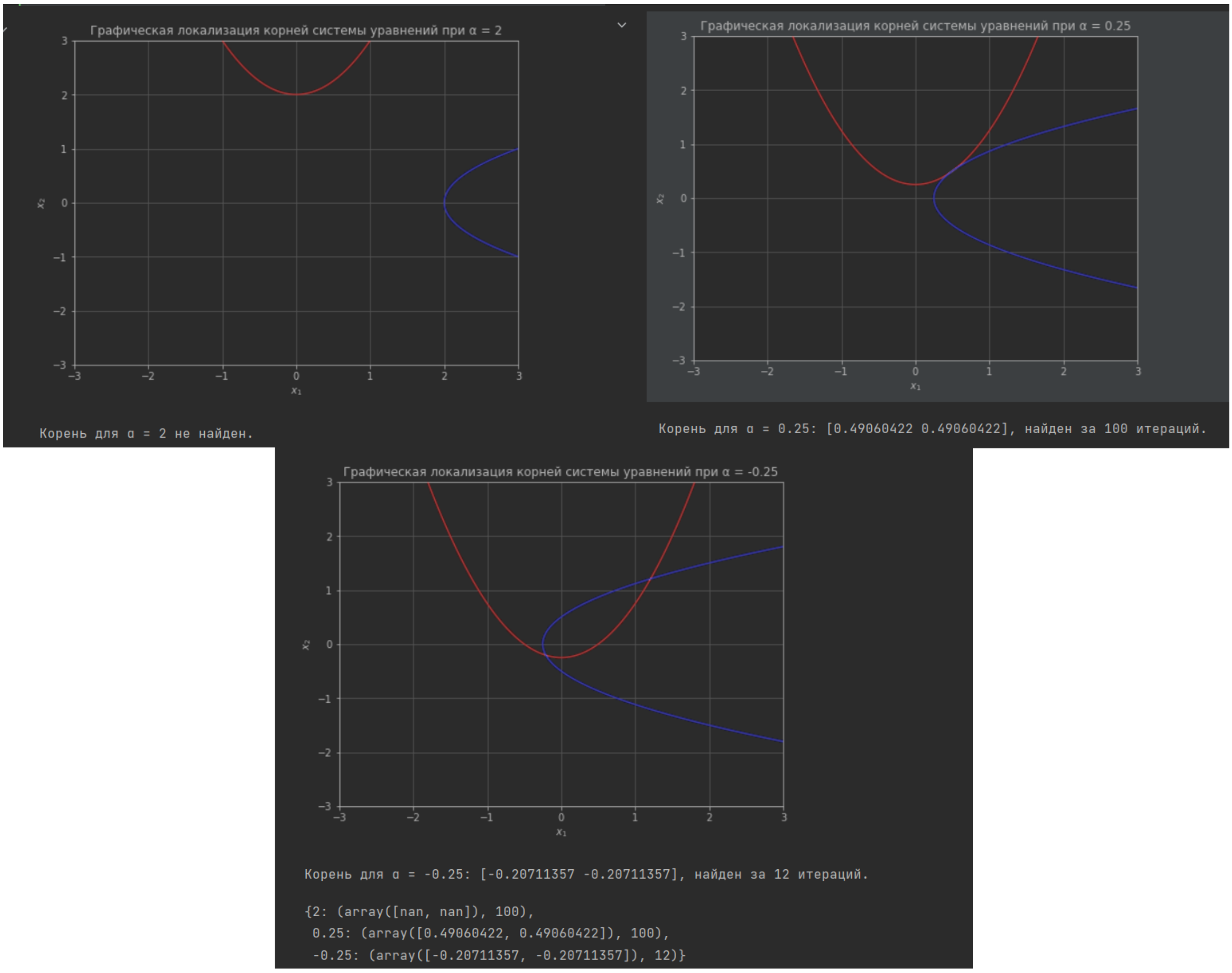

Рис. 4: Итоговые результаты по задаче