**Правительство Российской Федерации**

**Федеральное государственное автономное образовательное учреждение высшего образования   
"Национальный исследовательский университет   
"Высшая школа экономики"**

Московский институт электроники и математики Национального

исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

**ОТЧЕТ**

**по Лабораторной работе №3**

**По курсу «Численные методы»**

РЕШЕНИЕ СИСТЕМ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ

|  |  |  |  |
| --- | --- | --- | --- |
| **ФИО студента** | **Номер группы** | **Вариант 15** | **Дата** |
| Пугач Виктория Павловна | БПМ-211 | 4.1.15, 4.4.6, 5.1.15, 5.2 | 20.03.2024 |

**Москва – 2024 г.**

**Оглавление**

[**Задача 4.1.15** 3](#_Toc162028303)

[**Пункт 1** 3](#_Toc162028304)

[**Пункт 2** 4](#_Toc162028305)

[**Пункт 3** 5](#_Toc162028306)

[**Пункт 4** 6](#_Toc162028307)

[**Пункт 5** 6](#_Toc162028308)

[**Задача 4.4.6** 7](#_Toc162028309)

[**Пункт 1** 7](#_Toc162028310)

[**Пункт 2** 8](#_Toc162028311)

[**Пункт 3** 9](#_Toc162028312)

[**Пункт 4** 9](#_Toc162028313)

[**Задача 5.1.15** 15](#_Toc162028314)

[**Пункт 1** 15](#_Toc162028315)

[**Пункт 2** 16](#_Toc162028316)

[**Пункт 3** 17](#_Toc162028317)

[**Пункт 4** 18](#_Toc162028318)

[**Задача 5.2** 19](#_Toc162028319)

[**Пункт 1** 19](#_Toc162028320)

# **Задача 4.1.15**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

## **Пункт 1**

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

import numpy as np

import matplotlib.pyplot as plt

x1, x2 = np.meshgrid(np.arange(-2, 2, 0.005), np.arange(-2, 2, 0.005))

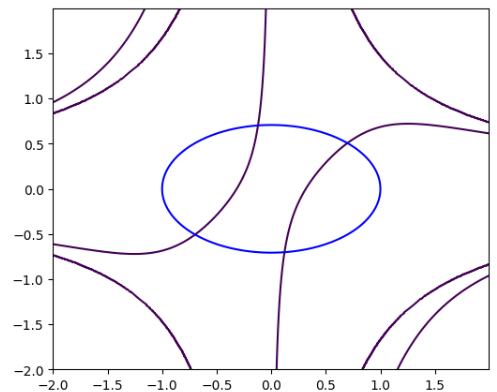
plt.figure(figsize=(6, 5))

plt.contour(x1, x2, np.tan(x1\*x2 + 0.1) - x1\*\*2, [0])

plt.contour(x1, x2, x1\*\*2 + 2\*x2\*\*2 - 1, [0], colors = "blue")

plt.show()

Вывод программы:



## **Пункт 2**

import sympy as sp

x1 = sp.Symbol('x1')

x2 = sp.Symbol('x2')

f1 = sp.tan(x1\*x2 + 0.1) - x1\*\*2

f2 = x1\*\*2 + 2\*x2\*\*2 - 1

f\_matrix\_form = sp.Matrix([f1, f2])

eps = 1e-6 # заданная точность

def root(f\_matrix, point, eps):

    iter\_count = 0

    jacobian = f\_matrix.jacobian([x1, x2]).inv() \* f\_matrix

    while True:

        jacobian\_num = jacobian.subs([(x1, point[0]), (x2, point[1])])

        point -= np.array(jacobian\_num, dtype=float).flatten()

        iter\_count += 1

        norm = np.linalg.norm(np.array(f\_matrix.subs([(x1, point[0]), (x2, point[1])]), dtype=float).flatten())

        if norm < eps:

            break

    return point, iter\_count

# локализованные приближения точек, с которых будем начинать поиск корней уравнения

# Всего у функций 4 точки пересечения

points = np.array([[1.0, 0.5],[0.3, -0.6],[-0.5, -0.3],[-0.3, 0.5]])

points\_result = []

for point in points:

  points\_tmp = root(f\_matrix\_form, point, eps)

  points\_result.append(points\_tmp)

print(points\_result)

Вывод программы:

Изображение выглядит как текст, Шрифт, белый, типография

Автоматически созданное описание

## **Пункт 3**

Построим полученные точки на том же графике, что и функции

a=[[0.6980717 , 0.50630816],[0.12145922, -0.70187173],[-0.6980717 , -0.50630816],[-0.12145916,  0.70187167]]

plt.plot(\*zip(\*a), marker='o', color='r', ls='')

x1, x2 = np.meshgrid(np.arange(-2, 2, 0.005), np.arange(-2, 2, 0.005))

# plt.figure(figsize=(6, 5))

plt.contour(x1, x2, np.tan(x1\*x2 + 0.1) - x1\*\*2, [0])

plt.contour(x1, x2, x1\*\*2 + 2\*x2\*\*2 - 1, [0], colors = "blue")

plt.show()

Изображение выглядит как диаграмма, линия, График, круг

Автоматически созданное описание

Вывод: Программа действительно находит решения нелинейной системы достаточно корректно

## **Пункт 4**

from scipy.optimize import fsolve

import math

eps = 1e-6 # заданная точность

def equations(vars):

    x1, x2 = vars

    eq1 = np.tan(x1 \* x2 + 0.1) - x1\*\*2

    eq2 = x1\*\*2 + 2\*x2\*\*2 - 1

    return [eq1, eq2]

points = np.array([[1.0, 0.5],[0.3, -0.6],[-0.5, -0.3],[-0.3, 0.5]])

points\_built\_in = []

for point in points:

  point\_tmp = fsolve(equations, point, xtol = eps)

  points\_built\_in.append(point\_tmp)

print(points\_built\_in)

Вывод программы:

Изображение выглядит как текст, Шрифт, белый

Автоматически созданное описание

## **Пункт 5**

a=np.array([[0.6980717 , 0.50630816],[0.12145922, -0.70187173],[-0.6980717 , -0.50630816],[-0.12145916,  0.70187167]])

b = np.array([[0.6980717 , 0.50630816],[0.12145916, -0.70187167],[-0.6980717 , -0.50630816],[-0.12145916,  0.70187167]])

delta = []

for i in range(4):

  delta.append(a[i] - b[i])

print(delta)

Вывод программы:



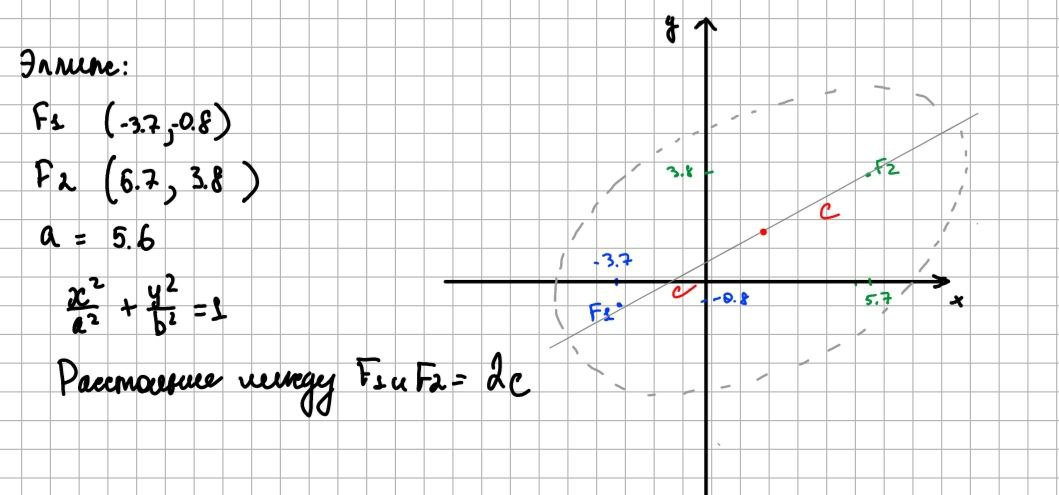
# **Задача 4.4.6**

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание



## **Пункт 1**



Изображение выглядит как Шрифт, линия, диаграмма, число

Автоматически созданное описаниеИзображение выглядит как диаграмма, линия, число, текст

Автоматически созданное описание

Я использовала функцию, задающую график эллипса, чей центр не лежит в центре координат, но оси симметрии параллельны осям координат.   
Изображение выглядит как Шрифт, линия, число, снимок экрана

Автоматически созданное описание

Затем я просто повернула получившийся эллипс при помощи матрицы поворота.

## **Пункт 2**

F1 = np.array([-3.7,-0.8])

F2 = np.array([5.7, 3.8])

a = 5.6

c= np.linalg.norm(F1 - F2)/2 # с = 1/2 расстояния между фокусами

center = np.array([(F1[0] + F2[0])/2,(F1[1] + F2[1])/2])

b = np.sqrt(a\*\*2 - c\*\*2)

theta = np.arctan((F1[1] - F2[1])/(F1[0] - F2[0]))

print(c, center, b, theta)

def ellipse\_equation(x, y, center, a, b):

    return ((x - center[0])\*\*2 / a\*\*2) + ((y - center[1])\*\*2 / b\*\*2) - 1

def rotate\_point(x, y, theta):

    cos\_theta = np.cos(theta)

    sin\_theta = np.sin(theta)

    x\_new = x \* cos\_theta - y \* sin\_theta

    y\_new = x \* sin\_theta + y \* cos\_theta

    return x\_new, y\_new

x, y = np.meshgrid(np.arange(-8, 8, 0.05), np.arange(-8, 8, 0.05))

# Генерация точек эллипса

x\_values = np.linspace(center[0] - a, center[0] + a, 400)

y\_values = np.linspace(center[1] - b, center[1] + b, 400)

X, Y = np.meshgrid(x\_values, y\_values)

Z = ellipse\_equation(X, Y, center, a, b)

# Поворот эллипса

X\_rotated, Y\_rotated = rotate\_point(X - center[0], Y - center[1], theta)

X\_rotated += center[0]

Y\_rotated += center[1]

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

plt.contour(x, y, x\*\*2 / 36 - y\*\*2/ 4 + 1, [0])

plt.contour(X\_rotated, Y\_rotated, Z, levels=[0], colors='r')

plt.gca().set\_aspect('equal', adjustable='box')

plt.xlabel('x')

plt.ylabel('y')

plt.title('График двух кривых второго порядка')

plt.show()

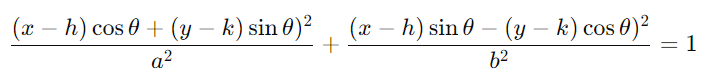
Изображение выглядит как линия, График, диаграмма, текст

Автоматически созданное описание

## **Пункт 3**

Точки пересечения кривых примерно: [-2, 2], [6, 2]

## **Пункт 4**





((x - center[0])\*np.cos(theta) + (y - center[1])\*np.sin(theta))\*\*2/a\*\*2 + ((x - center[0])\*np.sin(theta) - (y - center[1])\*np.cos(theta))\*\*2/b\*\*2 – 1 = 0

x\_approx = np.array([[-2,2],[6,2]], dtype=float)

import sympy as sp

x = sp.Symbol('x')

y = sp.Symbol('y')

f1 = x\*\*2 / 36 - y\*\*2/ 4 + 1

f2 = ((x - center[0])\*np.cos(theta) + (y - center[1])\*np.sin(theta))\*\*2/a\*\*2 + ((x - center[0])\*np.sin(theta) - (y - center[1])\*np.cos(theta))\*\*2/b\*\*2 - 1

f\_matrix\_form = sp.Matrix([f1, f2])

eps = 1e-6 # заданная точность

def root(f\_matrix, point, eps):

    iter\_count = 0

    jacobian = f\_matrix.jacobian([x, y]).inv() \* f\_matrix

    while True:

        jacobian\_num = jacobian.subs([(x, point[0]), (y, point[1])])

        point -= np.array(jacobian\_num, dtype=float).flatten()

        iter\_count += 1

        norm = np.linalg.norm(np.array(f\_matrix.subs([(x, point[0]), (y, point[1])]), dtype=float).flatten())

        if norm < eps:

            break

    return point, iter\_count

points = x\_approx

points\_result = []

for point in points:

  points\_tmp = root(f\_matrix\_form, point, eps)

  points\_result.append(points\_tmp)

print(points\_result)



Точки пересечения кривых примерно: [-2, 2], [6, 2]

Результаты совпадают в пределах погрешности

## **Пункт 5**

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, График, линия

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, линия, число

Автоматически созданное описание

Далее я перевела это в формат, приемлемый для питона.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

point1 = np.array([-1.90061706,  2.09794463])

point2 = np.array([5.82675909, 2.7878929 ])

x\_between =  np.arange(point1[0], point2[0], 0.0005)

x0, y0 = center[0], center[1]

y\_curve = []

y\_ellips = []

y2\_coef = 0.0318878 \* math.sin(theta)\*\*2 + 0.251256 \* math.cos(theta)\*\*2

def y1\_coef(x):

  return   - 0.0956633 \* math.sin(theta)\*\*2 \

          - 0.438737 \* x \* math.cos(theta) \* math.sin(theta) \

          + 0.438737\* math.cos(theta) \* math.sin(theta) \

          - 0.753769 \* math.cos(theta)\*\*2

def const\_coef(x):

  return -1 \

+ 0.597214 \* math.cos(theta)\*\*2 \

- 0.0637755 \* x \* math.cos(theta)\*\*2 \

+ 0.0318878 \* x\*\*2 \* math.cos(theta)\*\*2 \

    - 0.658106 \* math.cos(theta) \* math.sin(theta) \

          + 0.658106 \* x \* math.cos(theta) \* math.sin(theta) \

          + 0.323004 \* math.sin(theta)\*\*2 \

          - 0.502513 \* x \* math.sin(theta)\*\*2 \

          + 0.251256 \* x\*\*2 \* math.sin(theta)\*\*2

for x\_ in x\_between:

  y\_curve.append(np.max(np.roots([0.25, 0, -x\_ \*\* 2 / 36 - 1])))

  # коэф-ты перед y\*\*2, y, const

  y\_ellips.append(np.max(np.roots([ y2\_coef, y1\_coef(x\_), const\_coef(x\_)])))

plt.plot(x\_between, y\_curve)

plt.plot(x\_between, y\_ellips, color='red')

plt.gca().set\_aspect('equal')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

area = 0

for i in range(1, len(x\_between)):

  area += (x\_between[i] - x\_between[i - 1]) \* (y\_ellips[i] - y\_curve[i])

print(f'Площадь фигуры равна {area = }')

Изображение выглядит как линия, График, диаграмма, скат

Автоматически созданное описание

Найдем площадь оставшейся фигуры:

x\_between =  np.arange(point2[0], 6.2, 0.0005)

x0, y0 = center[0], center[1]

y\_ellips\_high = []

y\_ellips\_low = []

y2\_coef = 0.0318878 \* math.sin(theta)\*\*2 + 0.251256 \* math.cos(theta)\*\*2

def y1\_coef(x):

  return   - 0.0956633 \* math.sin(theta)\*\*2 \

          - 0.438737 \* x \* math.cos(theta) \* math.sin(theta) \

          + 0.438737\* math.cos(theta) \* math.sin(theta) \

          - 0.753769 \* math.cos(theta)\*\*2

def const\_coef(x):

  return -1 \

+ 0.597214 \* math.cos(theta)\*\*2 \

- 0.0637755 \* x \* math.cos(theta)\*\*2 \

+ 0.0318878 \* x\*\*2 \* math.cos(theta)\*\*2 \

    - 0.658106 \* math.cos(theta) \* math.sin(theta) \

          + 0.658106 \* x \* math.cos(theta) \* math.sin(theta) \

          + 0.323004 \* math.sin(theta)\*\*2 \

          - 0.502513 \* x \* math.sin(theta)\*\*2 \

          + 0.251256 \* x\*\*2 \* math.sin(theta)\*\*2

for x\_ in x\_between:

  y\_ellips\_high.append(np.max(np.roots([ y2\_coef, y1\_coef(x\_), const\_coef(x\_)])))

  # коэф-ты перед y\*\*2, y, const

  y\_ellips\_low.append(np.min(np.roots([ y2\_coef, y1\_coef(x\_), const\_coef(x\_)])))

plt.plot(x\_between, y\_ellips\_low, color='red')

plt.plot(x\_between, y\_ellips\_high)

plt.gca().set\_aspect('equal')

plt.xlabel('x')

plt.ylabel('y')

plt.show()

area2 = 0

for i in range(1, len(x\_between)):

  area2 += (x\_between[i] - x\_between[i - 1]) \* (y\_ellips[i] - y\_curve[i])

print('Площадь первой части фигуры равна =', area2)

Изображение выглядит как текст, снимок экрана, диаграмма, График

Автоматически созданное описание

Площадь в месте «палочки» все-равно равна 0.

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

# **Задача 5.1.15**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

## **Пункт 1**

import numpy as np

eps = 1e-5

A = np.array([[0.33, 0.1, 0.1, 0, 0.02, 0.1],

              [0.99, 4.9, 0.4, 2.97, 0.21, -0.3],

              [1.32, -1.6, 6.6, 3.3, 0.24, 0.1],

              [1.98, 1.2, 1.1, 6.93, 0.81, -1.2],

              [1.98, -1.5, 0.4, -1.98, 6.1, 0],

              [0.99, 0.4, 0.3, 1.65, 0.9, 4.3]])

b = np.array([1.620,

              23.365,

              -14.010,

              18.955,

              24.880,

              -1.500])

def gauss\_func(A1, b1):

  A = A1.copy()

  b = b1.copy()

  n = len(b)

  for i in range(n):

    max\_el = A[i,i] # Мы проверяем элементы ниже главной диагонали

                    # Ищем среди них максимальный элемент в каждом столбце (среди тех, что ниже a\_ii)

    max\_index = i

    for j in range(i + 1, n):

      if abs(A[j, i]) > abs(max\_el):

        max\_index = j

        max\_el = A[j, i]  # Ищем максмальный (по модулю) элемент

    if max\_index != i:  # Перестановка строк в случае, если максимальный по модулю

                            # элемент не устоит УЖЕ на диагонали. В таком случае ничего не меняем

      A[i, :], A[max\_index, :] = A[max\_index, :].copy(), A[i, :].copy()

      b[i], b[max\_index] = b[max\_index].copy(), b[i].copy()

    for j in range(i+1, n):    #Прямой ход

      factor = A[j, i] / A[i, i] # делаем так, чтобы под главной диагональю были 0

      A[j,:] -= factor \* A[i, :]

      b[j] -= factor \* b[i]

  x = np.zeros(n)   # Вектор решений

  for i in range(n-1, -1, -1):   # Обратный ход

    x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]

  return x

x = gauss\_func(A, b)

print("\nРешение системы уравнений Ax=b:")

print(x)

print("\nРешение системы уравнений Ax=b при помощи встроенной функции:")

print(np.linalg.solve(A,b))

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

## **Пункт 2**

Проверка достаточного условия сходимости итерационных методов



B = np.empty(A.shape, dtype=float)

for i in range(A.shape[0]):

  for j in range(A.shape[1]):

    B[i, j] = - A[i, j] / A[i, i] if i != j else 0

print("Бесконечная норма матрицы B = ", np.linalg.norm(B, ord=np.inf))



## **Пункт 3**

def zeid(A, b, x0, n):

    B = np.empty(A.shape, dtype=float)

    for i in range(A.shape[0]):

        for j in range(A.shape[1]):

            B[i, j] = - A[i, j] / A[i, i] if i != j else 0

    c = np.empty(b.shape, dtype=float)

    for i in range(c.shape[0]):

        c[i] = b[i] / A[i, i]

    # B1 - нижнетреугольная матрица

    # B2 - верхнетреугольная матрица

    x = x0

    for \_ in range(n):

        x\_new = np.zeros(x.shape)

        for i in range(B.shape[0]):

            x\_new[i] = np.sum(B[i][:i] \* x\_new[:i]) + np.sum(B[i][i:] \* x[i:]) + c[i] # Мы не создаем отдельно B1, B2, а просто пользуемся срезами

        x = x\_new

    return x

# Решение, полученное в предыдущем пункте, называлось x\_gauss

x\_zeid = zeid(A, b, np.zeros(6), 10)

print('Решение, полученное методом Гаусса: ', x\_gauss)

print('Решение, полученное методом Зейделя: ', x\_zeid)



delta = np.linalg.norm(x\_zeid - x\_gauss, ord=np.inf)

print(delta)



## **Пункт 4**

x\_zeid2 = zeid(A, b, np.array([1,2,3,4,5,1]), 10)

x\_zeid3 = zeid(A, b, np.array([0,1,0,2,0,1]), 10)

x\_zeid4= zeid(A, b, np.array([1,0,0,1,0,0]), 10)

x\_zeid5= zeid(A, b, np.array([6,3,2,0,3,2]), 10)

delta5 = np.linalg.norm(x\_zeid5 - x\_gauss, ord=np.inf)

delta2 = np.linalg.norm(x\_zeid2 - x\_gauss, ord=np.inf)

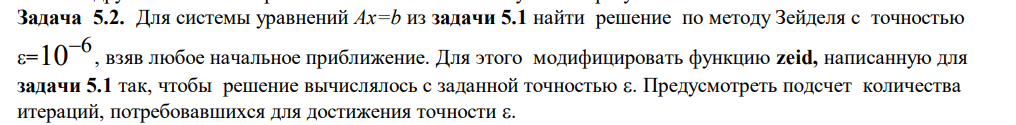
delta3 = np.linalg.norm(x\_zeid3 - x\_gauss, ord=np.inf)

delta4 = np.linalg.norm(x\_zeid4 - x\_gauss, ord=np.inf)

print(delta2, delta3, delta4, delta5)



# **Задача 5.2**



Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание



## **Пункт 1**

eps2 = 1e-6

def zeid\_error(A, b, x0, eps):

  B = np.empty(A.shape, dtype=float)

  for i in range(A.shape[0]):

    for j in range(A.shape[1]):

      B[i, j] = - A[i, j] / A[i, i] if i != j else 0

  c = np.empty(b.shape, dtype=float)

  for i in range(c.shape[0]):

    c[i] = b[i] / A[i, i]

  B1 = np.zeros(B.shape)

  B2 = np.zeros(B.shape)

  # в явном виде B1, B2

  for i in range(A.shape[0]):

    for j in range(A.shape[1]):

      if j < i:

        B1[i, j] = B[i, j]

      if j > i:

        B2[i, j] = B[i, j]

  iterations\_number = 0

  answer = x0

  error = 1

  while error > eps:

    x\_new = np.zeros(6)

    for i in range(B.shape[0]):

      x\_new[i] = np.sum(B[i][:i] \* x\_new[:i]) + np.sum(B[i][i:] \* answer[i:]) + c[i]

    error = np.linalg.norm(answer - x\_new, ord=np.inf)\*np.linalg.norm(B2, ord=np.inf) / (1 - np.linalg.norm(B, ord=np.inf))

    answer = x\_new

    iterations\_number += 1

  return answer, iterations\_number

x\_zeid\_err, it = zeid\_error(A, b, np.zeros(6), eps2)

print('Решение, полученное методом Гаусса: ', x\_gauss)

print('Решение, полученное методом Зейделя: ', x\_zeid)

print('Решение, полученное методом Зейделя с погрешностью: ', x\_zeid\_err, ", кол-во итераций: ", it)

