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

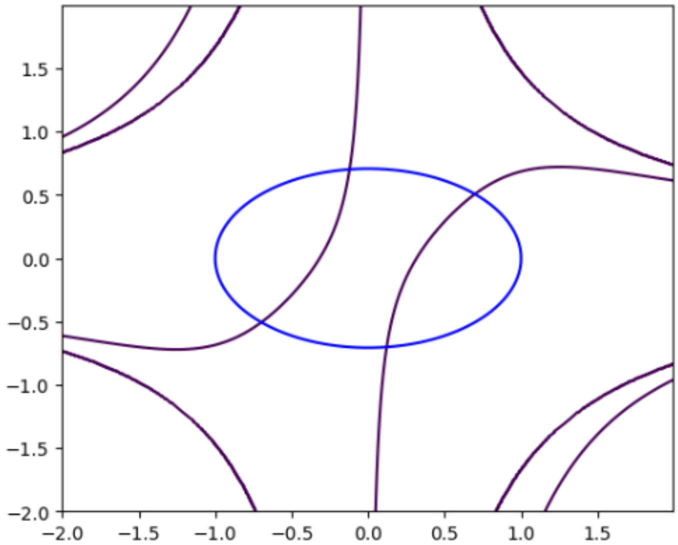
Задача 4.1. Найти с точностью $\varepsilon = 10^{-6}$ все корни системы нелинейных уравнений
$$f_1(x_1, x_2) = 0,$$
$$f_2(x_1, x_2) = 0,$$
используя метод Ньютона для системы нелинейных уравнений.
ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:
1. Используя встроенные функции, локализовать корни системы уравнений графически.
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. Используя встроенные функции, найти все корни системы с точностью ε. Сравнить с результатами, полученнными в п. 3.

Пункт 1

4.1.15	$\tan(x_1 x_2 + 0.1) - x_1^2 = 0$ $x_1^2 + 2x_2^2 - 1 = 0$

```
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

```python
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)
```

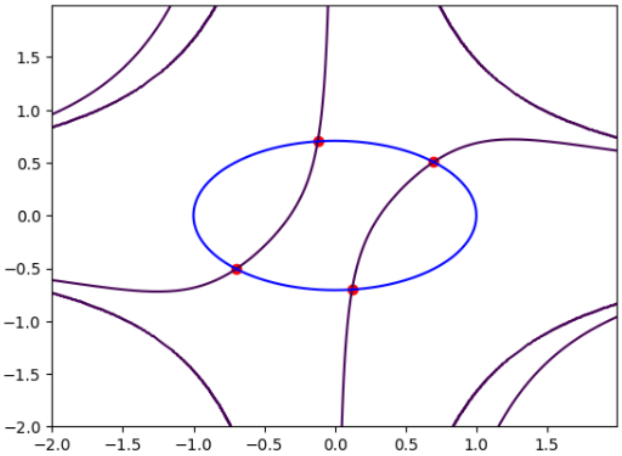
Вывод программы:

```
(array([0.6980717 , 0.50630816]), 4)
(array([ 0.12145922, -0.70187173]), 3)
(array([-0.6980717 , -0.50630816]), 4)
(array([-0.12145916,  0.70187167]), 4)
```

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()
```



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:

```
[0.6980717  0.50630816]
[ 0.12145916 -0.70187167]
[-0.6980717  -0.50630816]
[-0.12145916  0.70187167]
```

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:

```
[array([0., 0.]), array([ 6.00000000e-08, -6.00000001e-08]), array([0., 0.]), array([0., 0.])]
```

Задача 4.4.6

Задача 4.4. Плоская однородная пластина имеет форму геометрической фигуры, образованной пересечением двух кривых второго порядка. Определить площадь фигуры.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Составить уравнения заданных кривых второго порядка.
2. На одном чертеже построить графики заданных кривых. По чертежу определить форму пластины.
3. С помощью построенного чертежа локализовать координаты точек пересечения кривых.
4. Используя функцию, составленную при решении задачи 4.1, вычислить координаты точек пересечения кривых с точностью $\varepsilon = 10^{-6}$.
5. Вычислить площадь пластины.

4.4.5	$x^2/36 - y^2/4 = -1$	эллипс	(-2.3, 6.6)	(1.3, -0.6)	$2\sqrt{5}$
4.4.6	- " -	эллипс	(-3.7,-0.8)	(5.7, 3.8)	5.6

Пункт 1

Эллипс:

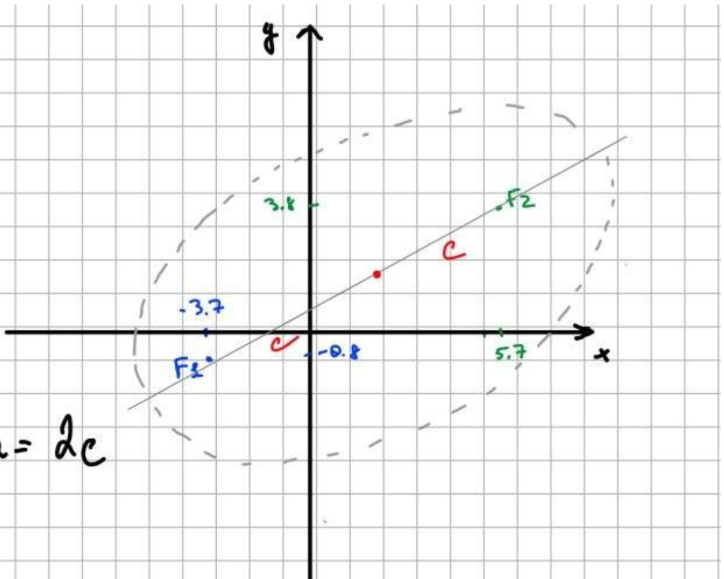
F_1 (-3.7, -0.8)

F_2 (5.7, 3.8)

$a = 5.6$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Расстояние между F_1 и $F_2 = 2c$



$$\frac{c}{a} = \sqrt{1-\left(\frac{b}{a}\right)^2}$$

$$\frac{c^2}{a^2} = \frac{a^2-b^2}{a^2}$$

$$b^2 = a^2 - c^2$$

$$b = \sqrt{a^2 - c^2}$$

$$\boxed{\varepsilon = \frac{c}{a} = \sqrt{1-\left(\frac{b}{a}\right)^2}}$$

Я использовала функцию, задающую график эллипса, чей центр не лежит в центре координат, но оси симметрии параллельны осям координат.

$$\boxed{\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1}$$

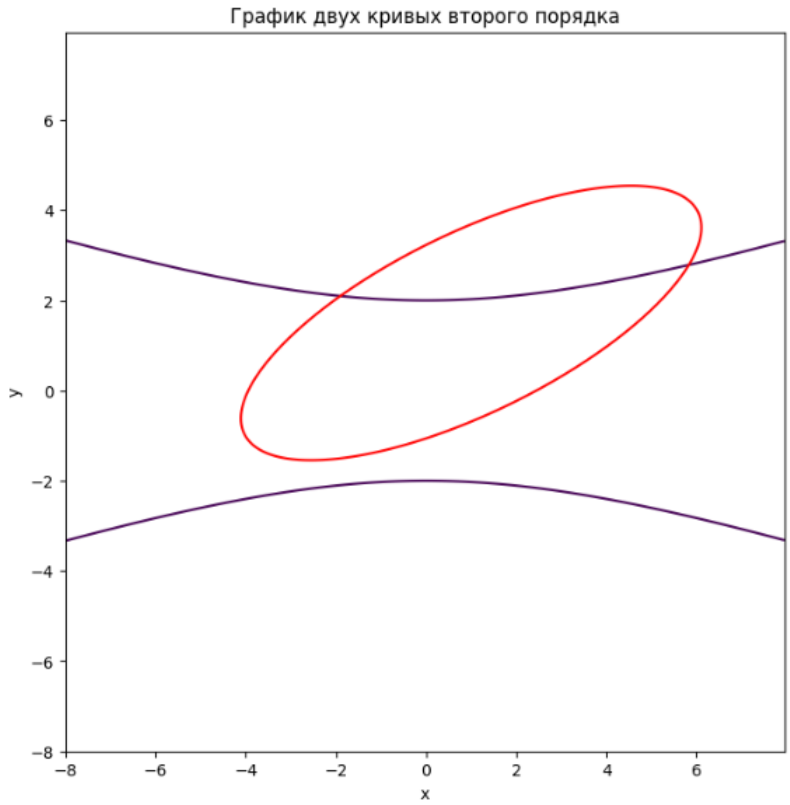
Затем я просто повернула получившийся эллипс при помощи матрицы поворота.

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 # c = 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()
```



Пункт 3

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

Пункт 4

$$\frac{((x-h)\cos\theta + (y-k)\sin\theta)^2}{a^2} + \frac{((x-h)\sin\theta - (y-k)\cos\theta)^2}{b^2} = 1$$

```
((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)
```

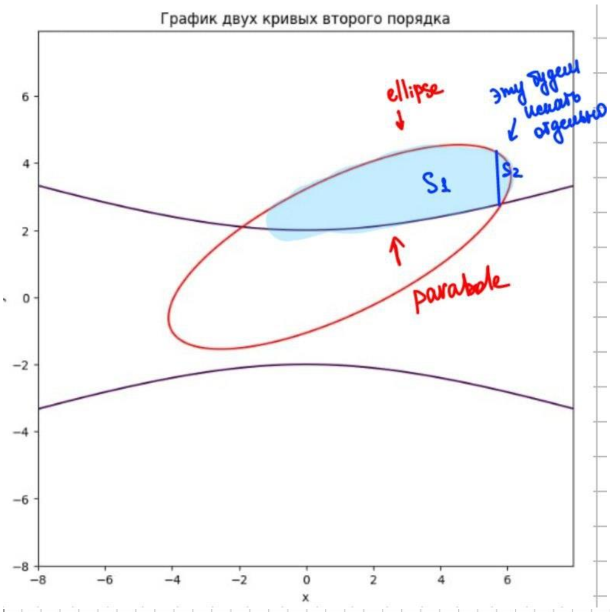
```python
import sympy as sp

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)
```

```
[(array([-1.90061706,  2.09794463]), 3), (array([5.82675909, 2.7878929 ]), 4)]
```

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

Результаты совпадают в пределах погрешности

Пункт 5



$$\cos^2\theta\,(y^2-2yy_0+y_0^2)$$

$$\frac{(x-x_0)^2\cos^2\theta+2(x-x_0)\cdot\cos\theta\cdot(y-y_0)\cdot\sin\theta+(y-y_0)^2\sin^2\theta}{a^2}+\frac{(x-x_0)^2\sin^2\theta-2(x-x_0)(y-y_0)\sin\theta\cdot\cos\theta+(y-y_0)^2\cos^2\theta}{b^2}=$$

$$+2\sin2\theta\,(xy-xy_0-x_0y+x_0y_0)\qquad \sin^2\theta\,(y^2-2yy_0+y_0^2)\qquad -2\sin2\theta\,(xy-xy_0-x_0y+x_0y_0)$$

y^2: $\dfrac{\sin^2\theta}{a^2}+\dfrac{\cos^2\theta}{b^2}$

y: $\dfrac{2\sin2\theta\,(x-x_0)-2\sin^2\theta\cdot y_0}{a^2}-\dfrac{2\sin2\theta\,(x-x_0)-2\cos^2\theta\, y_0}{b^2}$

const: $\dfrac{(x-x_0)^2\cos^2\theta+2\sin2\theta\, y_0(x_0-x)+\sin^2\theta\, y_0^2}{a^2}+\dfrac{(x-x_0)^2\sin^2\theta-2\sin2\theta\cdot y_0(x_0-x)-\cos^2\theta\cdot y_0^2}{b^2}-1$

In[4]:= $\dfrac{((x-x0)\,\mathrm{Cos}[\theta]+(y-y0)\,\mathrm{Sin}[\theta])^2}{a^2}+\dfrac{((x-x0)\,\mathrm{Sin}[\theta]-(y-y0)\,\mathrm{Cos}[\theta])^2}{b^2}-1$ // Expand
раскрыть скобки

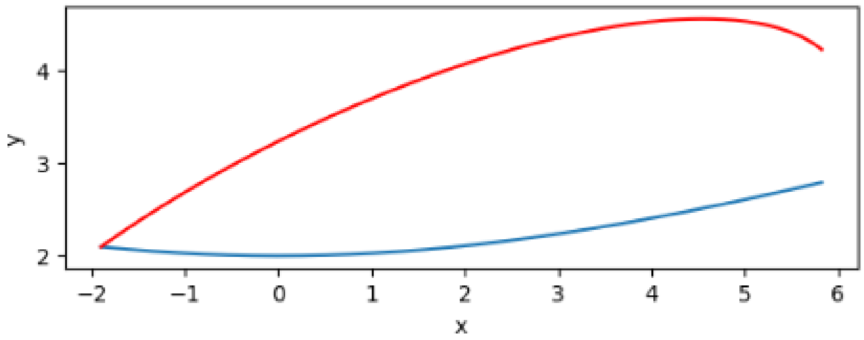
Out[4]= $-1+\frac{x^2\,\mathrm{Cos}[\theta]^2}{a^2}-\frac{2\,x\,x0\,\mathrm{Cos}[\theta]^2}{a^2}+\frac{x0^2\,\mathrm{Cos}[\theta]^2}{a^2}+\frac{y^2\,\mathrm{Cos}[\theta]^2}{b^2}-\frac{2\,y\,y0\,\mathrm{Cos}[\theta]^2}{b^2}+\frac{y0^2\,\mathrm{Cos}[\theta]^2}{b^2}+$
$\frac{2\,x\,y\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{a^2}-\frac{2\,x\,y\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{b^2}-\frac{2\,x0\,y\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{a^2}+\frac{2\,x0\,y\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{b^2}-$
$\frac{2\,x\,y0\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{a^2}+\frac{2\,x\,y0\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{b^2}+\frac{2\,x0\,y0\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{a^2}-\frac{2\,x0\,y0\,\mathrm{Cos}[\theta]\,\mathrm{Sin}[\theta]}{b^2}+$
$\frac{x^2\,\mathrm{Sin}[\theta]^2}{b^2}-\frac{2\,x\,x0\,\mathrm{Sin}[\theta]^2}{b^2}+\frac{x0^2\,\mathrm{Sin}[\theta]^2}{b^2}+\frac{y^2\,\mathrm{Sin}[\theta]^2}{a^2}-\frac{2\,y\,y0\,\mathrm{Sin}[\theta]^2}{a^2}+\frac{y0^2\,\mathrm{Sin}[\theta]^2}{a^2}$

Далее я перевела это в формат, приемлемый для питона.

```
'''
(-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

          - 0.0956633 * y * math.sin(theta)**2
          - 0.438737 * x * y * math.cos(theta) * math.sin(theta)
          + 0.438737 * y * math.cos(theta) * math.sin(theta)
          - 0.753769 * y * math.cos(theta)**2

          + 0.0318878 * y**2 * math.sin(theta)**2)
         + 0.251256 * y**2 * math.cos(theta)**2


'''
```

```
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 = }')
```



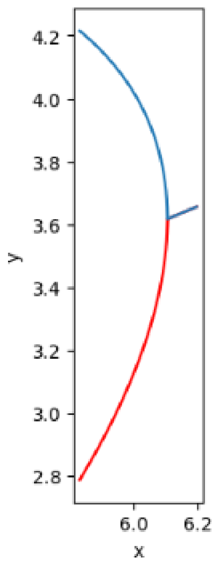

```
Площадь фигуры равна area = 12.153318891037872
```

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)
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py
  return np.asarray(x, float)
```




```
Площадь первой части фигуры равна = 0.05483032914880121
```

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

```
[42] area_all = area1 + area2

     print('Площадь фигуры: ', area_all)
```

```
Площадь фигуры:  12.208149220186673
```

Задача 5.1.15

Задача 5.1. Дана система уравнений $Ax=b$. Найти решение системы с помощью метода Гаусса. Выполнить 10 итераций по методу Зейделя. Принимая решение, полученное с помощью метода Гаусса за точное, найти величину абсолютной погрешности итерационного решения.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Задать матрицу системы A и вектор правой части b. Найти решение системы $Ax=b$ с помощью метода Гаусса.
2. Преобразовать систему $Ax=b$ к виду $x=Bx+c$, удобному для итераций. Проверить выполнение достаточного условия сходимости итерационных методов $\|B\|_\infty < 1$.
3. Написать программу-функцию **zeid**, решающую систему уравнений с помощью метода Зейделя, выполнить 10 итераций по методу Зейделя; взять любое начальное приближение. Принимая решение, полученное в п. 1 за точное, найти величину абсолютной погрешности итерационного решения (использовать норму $\|\cdot\|_\infty$).
4. Взять другое начальное приближение. Объяснить полученные результаты.

5.1.15	0.33	0.1	0.1	0	0.02	0.1	1.620
	0.99	4.9	0.4	2.97	0.21	-0.3	23.365
	1.32	-1.6	6.6	3.3	0.24	0.1	-14.010
	1.98	1.2	1.1	6.93	0.81	-1.2	18.955
	1.98	-1.5	0.4	-1.98	6.1	0	24.880
	0.99	0.4	0.3	1.65	0.9	4.3	-1.500

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))
```

```
Решение системы уравнений Ax=b:
[ 5.15151515  3.5        -2.5         0.21212121  3.5        -2.5       ]

Решение системы уравнений Ax=b при помощи встроенной функции:
[ 5.15151515  3.5        -2.5         0.21212121  3.5        -2.5       ]
```

Пункт 2

Проверка достаточного условия сходимости итерационных методов $\|B\|_\infty < 1$.

```
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))
```

```
Бесконечная норма матрицы B =  0.9939393939393939394
```

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)
```

```
Решение, полученное методом Гаусса:  [ 5.15151515  3.5        -2.5         0.21212121  3.5        -2.5       ]
Решение, полученное методом Зейделя:  [ 5.15151884  3.50001504 -2.49998354  0.21211819  3.50000044 -2.50000233]
```

```
delta = np.linalg.norm(x_zeid - x_gauss, ord=np.inf)
print(delta)
```

```
1.6457750101395163e-05
```

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)
```

```
4.559414420590713e-05 3.5334291697441955e-05 2.3135039216359843e-05 2.6405871861090446e-05
```

Задача 5.2

Задача 5.2. Для системы уравнений $Ax=b$ из **задачи 5.1** найти решение по методу Зейделя с точностью $\varepsilon=10^{-6}$, взяв любое начальное приближение. Для этого модифицировать функцию **zeid,** написанную для **задачи 5.1** так, чтобы решение вычислялось с заданной точностью ε. Предусмотреть подсчет количества итераций, потребовавшихся для достижения точности ε.

5.1.15	0.33	0.1	0.1	0	0.02	0.1	1.620
	0.99	4.9	0.4	2.97	0.21	-0.3	23.365
	1.32	-1.6	6.6	3.3	0.24	0.1	-14.010
	1.98	1.2	1.1	6.93	0.81	-1.2	18.955
	1.98	-1.5	0.4	-1.98	6.1	0	24.880
	0.99	0.4	0.3	1.65	0.9	4.3	-1.500

$$\|x^{(n)} - x^{(n-1)}\|\|B_2\|/(1 - \|B\|) < \varepsilon$$

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)
```

```
Решение, полученное методом Гаусса:  [ 5.15151515  3.5        -2.5         0.21212121  3.5        -2.5       ]
Решение, полученное методом Зейделя:  [ 5.15151884  3.50001504 -2.49998354  0.21211819  3.50000044 -2.50000233]
Решение, полученное методом Зейделя с погрешностью:  [ 5.15151515  3.5        -2.5         0.21212121  3.5        -2.5       ] , кол-во итераций:  18
```