CHRIST

(DEEMED TO BE UNIVERSITY)

BANGALORE · INDIA

MAT 651: Complex Analysis Using Python-

E-Record

Department of Mathematics

-Arnav Sinha, 2040811

Complex Analysis using Python (MAT 651)

Lab Record

Arnav Sinha, 2040811

LAB 1: Revision

Date: 02/01/23 to 07/01/23

Aim:

The main aim of Lab 1 is to review and revise basic concepts of python. Particularly focusing on basic operations, plotting, using various libraries like pandas, cmath, etc.

1. Display the current date

In [20]:
```
from datetime import *

date.today()
```

Out[20]:
```
datetime.date(2023, 3, 28)
```

2. Write a program to accept the user's first and last name and print it with space between them.

In [1]:
```
f= str(input("Enter your first name: "))
l= str(input("Enter your last name: "))

print("Your full name is: ", f, "",l)
```

```
Enter your first name: Arnav
Enter your last name: Sinha
Your full name is:  Arnav   Sinha
```

3. Write a python program which accepts a number from the user and predicts whether it is even or odd

In [2]:

```
n=int(input("Enter a number: "))

if n%2 == 0:
    print("The number is even")
else:
    print("The number is odd")
```

```
Enter a number: 4
The number is even
```

4. Write a program to divide two numbers only if the denominator is not 0

In [3]:

```
a1=int(input("Enter a number: "))
b1=int(input("Enter a number: "))

if b1 == 0:
    print("Division is not possible")
else:
    print(a1/b1)
```

```
Enter a number: 4
Enter a number: 5
0.8
```

5. Write a program to identify the largest of the given 3 numbers.

In [4]:

```
a=int(input("Enter a number: "))
b=int(input("Enter a number: "))
c=int(input("Enter a number: "))

m=max(a,b,c)

print("The largest number is: ", m)
```

```
Enter a number: 1
Enter a number: 2
Enter a number: 3
The largest number is:  3
```

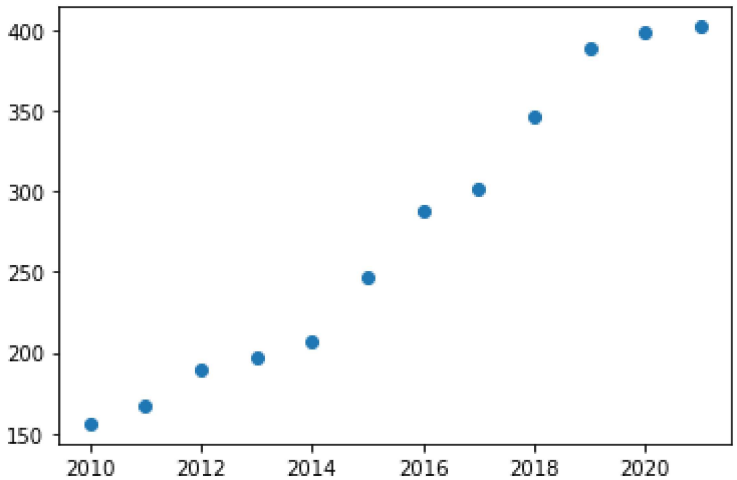
6. Plot a scatter plot (2D, 3D) for the given data.

year = 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021

price = 156, 166.9, 189, 196.9, 207.1, 247, 288, 302,346, 388, 399, 402

In [3]:

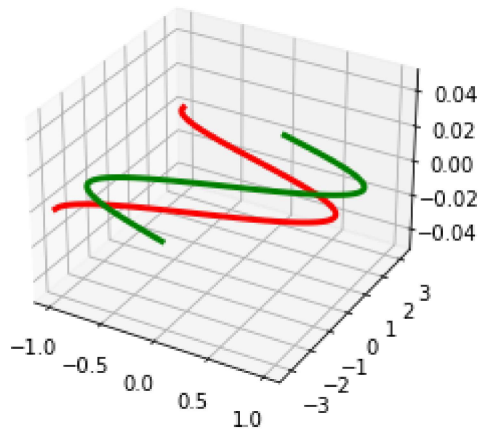
```
import matplotlib.pyplot as plt

year = [2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 20

price = [156, 166.9, 189, 196.9, 207.1, 247, 288, 302,346, 388, 399, 4

plt.scatter(year, price)

None
```



7. Create a 3D scatter plot for the cosine and sine functions

In [8]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

ax=plt.axes(projection='3d')

t=np.linspace(-np.pi, np.pi,100)
x=np.cos(t)
y=np.sin(t)

ax.plot3D(x,t,'red',lw=3)
ax.plot3D(y,t,'g', lw=3)

None
```

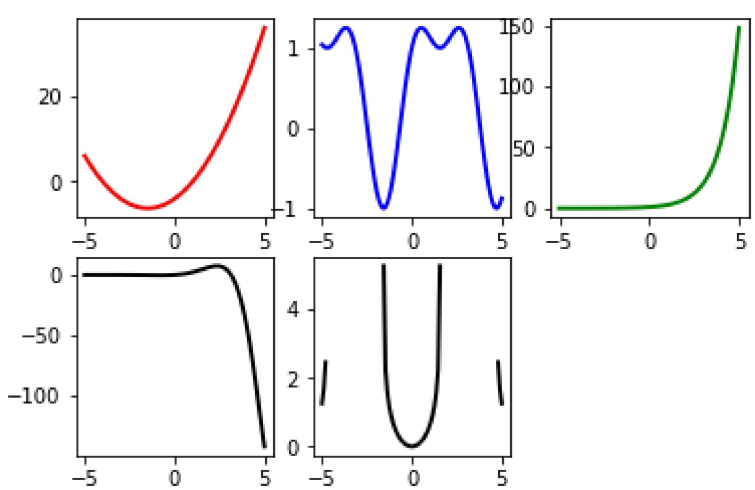


8. Create 5 subplots with the functions e^x, $e^x\ sinx$, $cosx^2 + sinx$, $log(secx)$, $x^2 + 3x - 4$

In [13]:

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

x=np.linspace(-5,5,100)
z=np.square(x) + 3*x -4
a=np.square(np.cos(x)) + np.sin(x)
y=np.exp(x)
w=np.exp(x)*np.sin(x)
b= np.log(1/np.cos(x))

plt.title("line plot")
plt.subplot(2,3,1)
plt.plot(x,z,lw=2,color='red')
plt.subplot(2,3,2)
plt.plot(x,a,lw=2,color='blue')
plt.subplot(2,3,3)
plt.plot(x,y,lw=2,color='green')
plt.subplot(2,3,4)
plt.plot(x,w,lw=2,color='black')
plt.subplot(2,3,5)
plt.plot(x,b,lw=2,color='black')

None
```

```
<ipython-input-13-eb3438e21002>:9: RuntimeWarning: invalid value encounte
red in log
  b= np.log(1/np.cos(x))
```



9. Print the divisors of a given number

In [19]:

```
num=int(input("Enter the number:"))
print("The divisors are:")

i=1
while(i<num):
    if(num%i==0):
        print(i)
    i=i+1
```

```
Enter the number:8
The divisors are:
1
2
4
```

10. Create and import the CSV file with atleast 15 entries and obtain the graph for the data

In [3]:

```
import pandas as pd

df = pd.read_csv('Stroniumdata.csv')
print(df)
```

```
    Stronium Levels    Water Bodies
0              28.2  Grayson's Pond
1              33.2  Grayson's Pond
2              36.4  Grayson's Pond
3              34.6  Grayson's Pond
4              29.1  Grayson's Pond
5              31.0  Grayson's Pond
6              39.6     Beaver Lake
7              40.8     Beaver Lake
8              37.9     Beaver Lake
9              37.1     Beaver Lake
10             43.6     Beaver Lake
11             42.4     Beaver Lake
12             46.3   Angler's Cove
13             42.1   Angler's Cove
14             43.5   Angler's Cove
15             48.8   Angler's Cove
16             43.7   Angler's Cove
17             40.1   Angler's Cove
18             41.0  Appletree Lake
19             44.1  Appletree Lake
20             46.4  Appletree Lake
21             40.2  Appletree Lake
22             38.6  Appletree Lake
23             36.3  Appletree Lake
24             56.3      Rock River
25             54.1      Rock River
26             59.4      Rock River
27             62.7      Rock River
28             60.0      Rock River
29             57.3      Rock River
```

Homework

1. Find roots of a quadratic equation and classify them.

In [2]:

```
import math

a = float(input("Insert coefficient a: "))
b = float(input("Insert coefficient b: "))
c = float(input("Insert coefficient c: "))

discriminant = b**2 - 4 * a * c

if discriminant >= 0:
    x_1=(-b+math.sqrt(discriminant))/2*a
    x_2=(-b-math.sqrt(discriminant))/2*a
else:
    x_1= complex((-b/(2*a)),math.sqrt(-discriminant)/(2*a))
    x_2= complex((-b/(2*a)),-math.sqrt(-discriminant)/(2*a))

if discriminant > 0:
    print("The function has two distinct real roots: {} and {}".format
elif discriminant == 0:
    print("The function has one double root: ", x_1)
else:
    print("The function has two complex (conjugate) roots: {}  and {}"

```

```
Insert coefficient a: 2
Insert coefficient b: 3
Insert coefficient c: 4
The function has two complex (conjugate) roots: (-0.75+1.1989578808281798
j)  and (-0.75-1.1989578808281798j)
```

2. Print the Fibbanocci series until the given number of terms

In [15]:

```
import math

def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)


fibonacci(6)
```

Out[15]: 8

3. Check if the given number is prime or composite

In [1]:

```
num=int(input("Enter the number:"))

i=1
j=0

while(i<num):
    if(num%i==0):
        j=j+1
    i=i+1

if(j>1):
    print("The number is composite.")
else:
    print("The number is prime.")
```

```
Enter the number:5
The number is prime.
```

4. Obtain the sum of the prime factors of a given number.

In [5]:

```
num=int(input("Enter the number:"))

i=1
j=2

sum=0

while(i<num):
    if(num%i==0):
        p=1
        while(j<i/2):
            if(i%j==0):
                p=0
            j=j+1
        if(p==1):
            sum=sum+i
    i=i+1

print(sum-1)
```

```
Enter the number:59
0
```

5. Find the factorial of any given number

In [6]:

```
a = int(input("Enter a number: "))
fact = 1
while a>1:
    fact=fact*a
    a=a-1
print(fact)
```

```
Enter a number: 3
6
```

6. Solve the following

A) $\frac{dy}{dx} = -6xy, y(0) = 0$

In [4]:

```
from sympy import *

x=Symbol('x')
y=Function('y')(x)
D=Eq(6*x*y+diff(y,x,1),0)
print(D)
display(D)
dsolve(D,y)
```

```
Eq(6*x*y(x) + Derivative(y(x), x), 0)
```

$$6xy(x) + \frac{d}{dx}y(x) = 0$$

Out[4]: $y(x) = C_1 e^{-3x^2}$

In [5]:

```
print(dsolve(D,y).subs({x:0}))
```

```
Eq(y(0), C1)
```

B) $\frac{dy}{dx} = k + \frac{y}{2}, y(0) = 1$ **(solve and plot for atleast 3 values of k)**

In [6]:
```
k = Symbol('k')
C1= Symbol('C1')
y=Function('y')(x)
D=Eq(diff(y,x,1)-y/2-k,0)
print(D)
display(D)
dsolve(D,y)
```

```
Eq(-k - y(x)/2 + Derivative(y(x), x), 0)
```

$$-k - \frac{y(x)}{2} + \frac{d}{dx}y(x) = 0$$

Out[6]: $y(x) = C_1 e^{\frac{x}{2}} - 2k$

In [7]:
```
sol=dsolve(D,y)
display(sol)
C1= Symbol('C1')
k=Symbol('k')
constants=solve(sol.subs({x:0,y:1}),C1)
constants
```

$$y(x) = C_1 e^{\frac{x}{2}} - 2k$$

Out[7]: [2*k + 1]

In [8]:
```
constants=solve(sol.subs({x:0,y:1}),k)
constants
```

Out[8]: [C1/2 - 1/2]

Date: 09/01/23

1. Write a program to find the sum of the squares of first n natural numbers.

In [7]:
```
def sumOfSquares(n) :
    if n < 0:
        return
    sum = 0
    for i in range(n+1):
        sum += i*i
    return sum

n = int(input('Enter n : '))
sum = sumOfSquares(n)
print(f'Sum : {sum}')
```

```
Enter n : 67
Sum : 102510
```

2. Write a python program to check whether a given number is an Armstrong number or not

In numerical number theory, the Armstrong number definition is the number in any given number base, which forms the total of the same number, when each of its digits is raised to the power of the number of digits in the number.

In [8]:

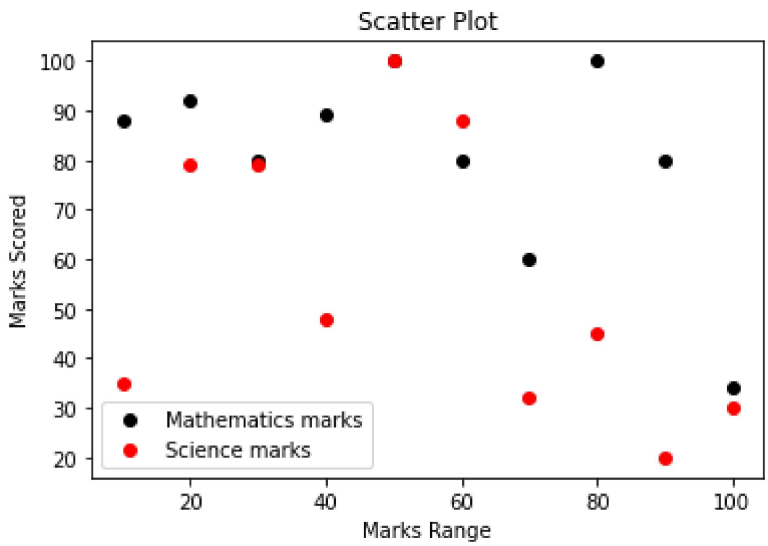
```python

num = int(input("Enter a number: "))

# initialize sum
sum = 0

# find the sum of the cube of each digit
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp //= 10

# display the result
if num == sum:
    print(num,"is an Armstrong number")
else:
    print(num,"is not an Armstrong number")
```

```
Enter a number: 2134
2134 is not an Armstrong number
```

3. Write a python program to draw scatter plot to comparing two subjects marks of Mathematics and Science

In [4]:

```
import matplotlib.pyplot as plt
import pandas as pd
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(marks_range, math_marks, label='Mathematics marks', color=
plt.scatter(marks_range, science_marks, label='Science marks', color='
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()

None
```



4. Write a python program to multiply two matrices entered by the user

In [9]:

```
import numpy as np
from sympy import *

A=np.mat(input("Enter a matrix: "))


B=np.mat(input("Enter a matrix: "))

A*B
```

```
Enter a matrix: 1,2,3;7,8,2;51,23,8
Enter a matrix: 5,6,7;5,6,7;5,6,7
```

Out[9]:

```
matrix([[ 30,  36,  42],
        [ 85, 102, 119],
        [410, 492, 574]])
```

5. Find the continuity of the following functions:

1. $f(x) = cosx$ at $x = 0$
2. $f(x) = 11x + 9$ at $x = 2$
3. $f(x) = 9x^5$ at $x = 0$
4. $f(x) = -x/2 + 9$ at $x = 8$
5. $f(x) = px + q$ at $x = -q$

In [10]:

```
## 1. cos(x) at x=0

import sympy as sp
import numpy as np
x=sp.Symbol("x")
f= sp.cos(x)
value=0

def checkifcontinuous(func, x,symbol):
    return (sp.limit(func,symbol,x).is_real)
print(checkifcontinuous(f,value,x))
```

```
True
```

In [11]:

```
## 2. 11x + 9 at x= 2

x=sp.Symbol("x")
f= 11*x + 9
value=2

def checkifcontinuous(func, x,symbol):
    return (sp.limit(func,symbol,x).is_real)
print(checkifcontinuous(f,value,x))
```

```
True
```

In [12]:

```
## 3. 9x^2 at x=2

x=sp.Symbol("x")
f= 9*x**2
value=2

def checkifcontinuous(func, x,symbol):
    return (sp.limit(func,symbol,x).is_real)
print(checkifcontinuous(f,value,x))
```

```
True
```

In [13]:

```
## 4. -x/2 + 9 at x=8

x=sp.Symbol("x")
f= -x/2 + 9
value=8

def checkifcontinuous(func, x,symbol):
    return (sp.limit(func,symbol,x).is_real)
print(checkifcontinuous(f,value,x))
```

```
True
```

6. Find the derivatives of the following functions

1. $f(x) = 2x - 3$
2. $f(x) = x^5(3 - 6x - 9x^3)$
3. $f(x) = sinxcosx$
4. $f(x) = 2tanx - 7secx$
5. $f(x) = (x^3 + x^2 + x)(x^2 + x)$

In [4]:
```
## 1. 2x-3
from sympy import *
x=Symbol('x')
q= 2*x - 3

i=Derivative(q,x)
i.doit()
```

Out[4]: 2

In [8]:
```
## 2. x^5(3-6x - 9x^3)

from sympy import *
x=Symbol('x')
a= (x**5)*(3 - 6*x - 9*x**3)

b=Derivative(a,x)
b.doit()
```

Out[8]: $x^5\left(-27x^2-6\right)+5x^4\left(-9x^3-6x+3\right)$

In [9]:
```
## 3. sinx cosx

c= sin(x)*cos(x)

d=Derivative(c,x)
d.doit()
```

Out[9]: $-\sin^2(x)+\cos^2(x)$

In [13]:
```
## 4. 2tanx - 7secx

e= 2*tan(x) - 7*sec(x)

f=Derivative(e,x)
f.doit()
```

Out[13]: $2\tan^2(x)-7\tan(x)\sec(x)+2$

In [16]:
```
## 5. (x^3 + x^2 + x)(x^2 + x)

g= (x**3 + x**2 + x) *(x**2 + x)

h=Derivative(g,x)
h.doit()
```

Out[16]: $(2x+1)\left(x^3+x^2+x\right)+\left(x^2+x\right)\left(3x^2+2x+1\right)$

7. Define a complex number and perform the basic operations of addition, subtraction, multiplication, and division

A complex number is the sum of a real number and an imaginary number. A complex number is of the form $a + ib$ and is usually represented by z. Here both a and b are real numbers. The value 'a' is called the real part which is denoted by Re(z), and 'b' is called the imaginary part Im(z). Also, ib is called an imaginary number.

In [2]:

```
import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

```
The real part of complex number is : 5.0
The imaginary part of complex number is : 3.0
```

In [3]:

```
## Addition

a,b=1,2
c,d=4,5

m=complex(a,b)
n=complex(c,d)


m+n
```

Out[3]: (5+7j)

In [4]:

```
## Subration

m-n
```

Out[4]: (-3-3j)

In [5]:

```
## Multiplication

m*n
```

Out[5]: (-6+13j)

In [6]:
```
## Division

m/n
```

Out[6]: (0.34146341463414637+0.0731707317073171j)

8. Find the real and imaginary parts of the complex number and find its modulus and argument

In [3]:
```
# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

```
The real part of complex number is : 5.0
The imaginary part of complex number is : 3.0
```

In [20]:
```
import cmath
import math

# Initializing real numbers
x = 1.0
y = 1.0

# converting x and y into complex number
z = complex(x,y);

# converting complex number into polar using polar()
w = cmath.polar(z)

# printing modulus and argument of polar complex number
print ("The modulus and argument of polar complex number is : ",end="'
print (w)

```

```
The modulus and argument of polar complex number is : (1.414213562373095
1, 0.7853981633974483)
```

Date: 16/01/23

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Dataset

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

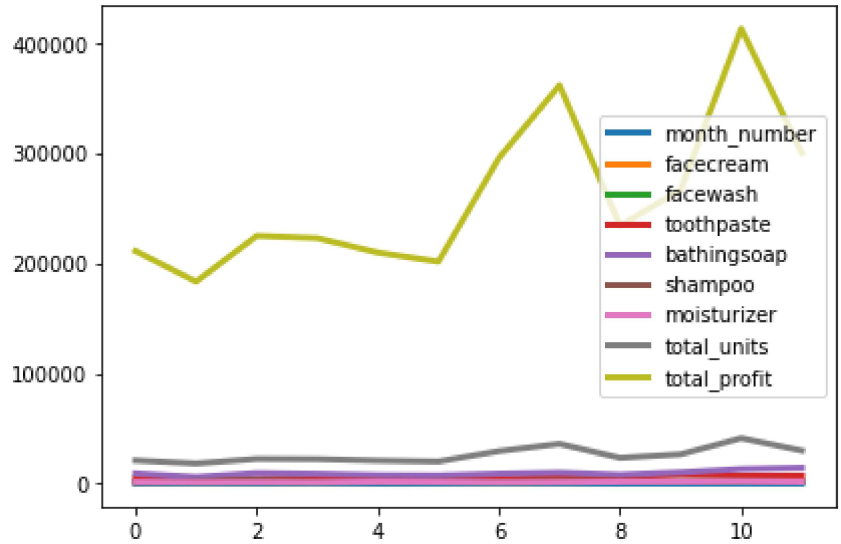
In [2]:
```
import pandas as pd

df = pd.read_csv('C:/Users/12138/Downloads/sales.csv')
print(df)
```

```
    month_number  facecream  facewash  toothpaste  bathingsoap  shampoo
\
0              1       2500      1500        5200         9200     1200
1              2       2630      1200        5100         6100     2100
2              3       2140      1340        4550         9550     3550
3              4       3400      1130        5870         8870     1870
4              5       3600      1740        4560         7760     1560
5              6       2760      1555        4890         7490     1890
6              7       2980      1120        4780         8980     1780
7              8       3700      1400        5860         9960     2860
8              9       3540      1780        6100         8100     2100
9             10       1990      1890        8300        10300     2300
10            11       2340      2100        7300        13300     2400
11            12       2900      1760        7400        14400     1800

    moisturizer  total_units  total_profit
0          1500        21100        211000
1          1200        18330        183300
2          1340        22470        224700
3          1130        22270        222700
4          1740        20960        209600
5          1555        20140        201400
6          1120        29550        295500
7          1400        36140        361400
8          1780        23400        234000
9          1890        26670        266700
10         2100        41280        412800
11         1760        30020        300200
```

1. Line Plot

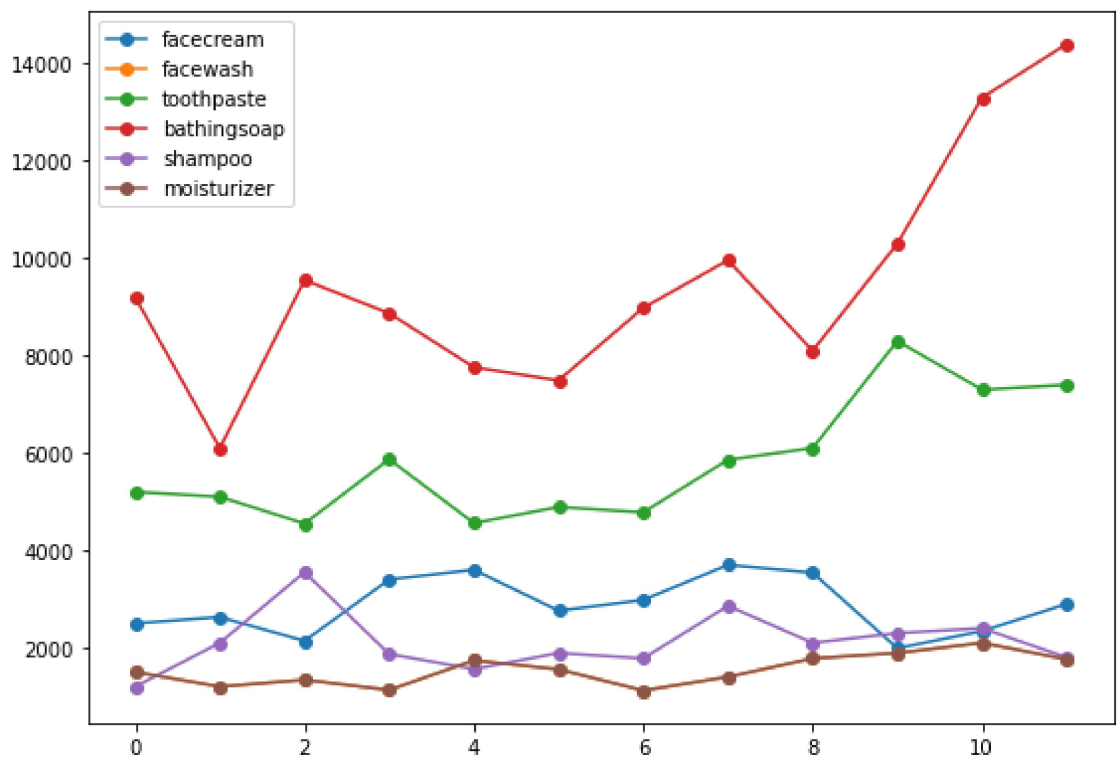
In [3]:

```
plt.rcParams["figure.figsize"] = [6.00, 4]
plt.rcParams["figure.autolayout"] = True
df.plot(linewidth=3, )
None
```



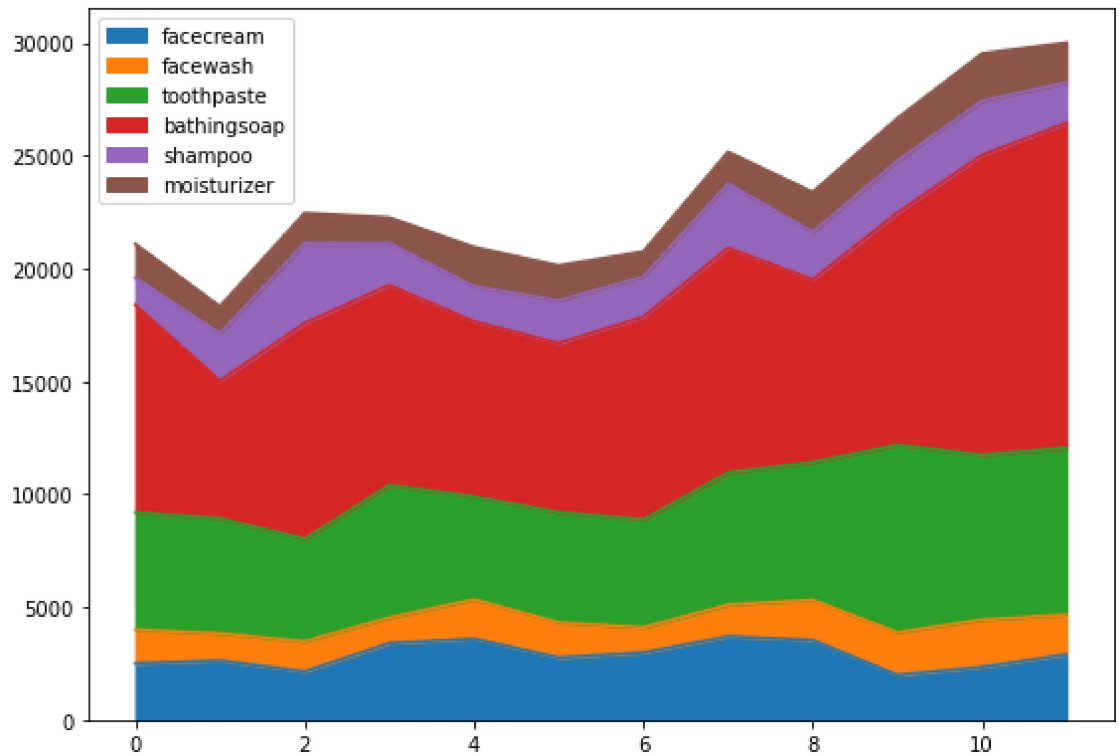
In [4]:

```
plt.rcParams["figure.figsize"] = [8.00, 5.50]
plt.rcParams["figure.autolayout"] = True

df.drop(['month_number','total_units','total_profit'], axis=1).plot(ma
None
```



2. Stack Plot

In [5]:
```
df.drop(['month_number','total_units','total_profit'], axis=1).plot.ar
None
```



3. Finding the mean, median and mode of the data

1. Mean

In [7]:
```
def mean(x):
    summ=sum(x)
    len1=len(x)
    m=summ/len1
    return m

mean(df.toothpaste)
```

Out[7]: 5825.833333333333

In [8]:
```
mean(df.facecream)
```

Out[8]: 2873.3333333333335

In [9]:
```
np.mean(df)
```

Out[9]:
```
month_number         6.500000
facecream         2873.333333
facewash          1542.916667
toothpaste        5825.833333
bathingsoap       9500.833333
shampoo           2117.500000
moisturizer       1542.916667
total_units      26027.500000
total_profit    260275.000000
dtype: float64
```

2. Median

In [10]:
```
def median(num):
    num.sort()
    n=len(num)
    if n % 2 == 0:
        median1 = num[n//2]
        median2 = num[n//2 - 1]
        median = (median1 + median2)/2
    else:
        median = num[n//2]
    print("Median is: " + str(median))

```

In [11]:
```
df.median()
```

Out[11]:
```
month_number         6.5
facecream         2830.0
facewash          1527.5
toothpaste        5530.0
bathingsoap       9090.0
shampoo           1995.0
moisturizer       1527.5
total_units      22935.0
total_profit    229350.0
dtype: float64
```

3. Mode

In [12]:
```
from scipy import stats

a,b=stats.mode(df)
a
```

Out[12]:
```
array([[     1,   1990,   1120,   4550,   6100,   2100,   1120,  18330,
        183300]], dtype=int64)
```

4. Range

In [13]:
```
def range(numb):
    minimum = min(numb)
    maximum = max(numb)
    range = maximum - minimum
    return range

print(range(df.facecream))
```

```
1710
```

In [14]:
```
range(df.toothpaste)
```

Out[14]:
```
3750
```

Conclusion:

This Lab practical offers a thorough examination of several Python ideas and libraries, encompassing the application of pandas for data analysis, plotting through Matplotlib, and carrying out rudimentary statistical analysis. Through this practical, we have obtained a more profound comprehension of these subjects.

Lab 2

Cmath functions for complex number

Date: 09/01/23 to 14/01/23

Aim:

The objective of this lab practical is to comprehend the idea of complex numbers, learn how to execute operations on them, obtain various forms of complex numbers, such as rectangular form, polar form, and exponential form, calculate conjugates, modulus, and argument of complex numbers, plot them on the complex plane, and test the Commutative law, Associative law, and Distributive law.

In [15]:

```
## 1. Real and Imaginary Part of a Complex number

import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)
```

```
The real part of complex number is : 5.0
The imaginary part of complex number is : 3.0
```

In [16]:

```
## 2. Conjugate

import numpy as np

np.conj(z)

```

Out[16]:

```
(5-3j)
```

In [22]:

```
## 3. Converting rectangular coordinates to polar coordinates

cmath.polar(z)

```

Out[22]:

```
(5.830951894845301, 0.5404195002705842)
```

In [25]:

```
## 4. Argument or Theta value

theta=cmath.phase(z)
theta
```

Out[25]:

```
0.5404195002705842
```

In [26]:

```
## 5. Modulus

mod=abs(z)
mod
```

Out[26]:

```
5.830951894845301
```

In [30]:

```
## 6. Converting polar coordinates into rectangular coordinates

p=cmath.rect(mod, theta)
p
```

Out[30]: (5+3.0000000000000004j)

In [14]:

```
## Plotting of complex numbers

import matplotlib.pyplot as plt

list=[]
i=0

while i < 5:
    i=i+1
    a=int(input("Enter the value of real part: "))
    b=int(input("Enter thr value of the imaginary part: "))
    ab=complex(a,b)
    list.append(ab)


print(list)
```

```
Enter the value of real part: 3
Enter thr value of the imaginary part: 7
Enter the value of real part: 7
Enter thr value of the imaginary part: 7
Enter the value of real part: 7
Enter thr value of the imaginary part: 7
Enter the value of real part: 7
Enter thr value of the imaginary part: 7
Enter the value of real part: 7
Enter thr value of the imaginary part: 7
[(3+7j), (7+7j), (7+7j), (7+7j), (7+7j)]
```

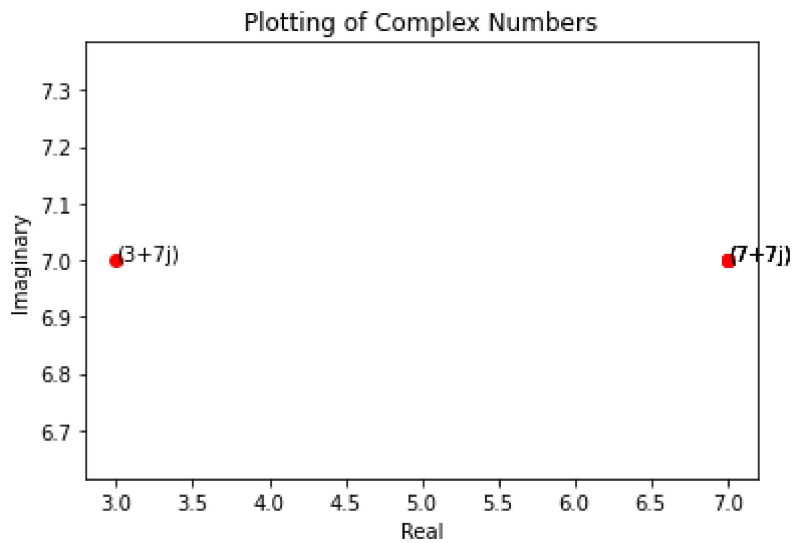
In [15]:

```
l=np.array(list)
l
```

Out[15]: array([3.+7.j, 7.+7.j, 7.+7.j, 7.+7.j, 7.+7.j])

In [16]:

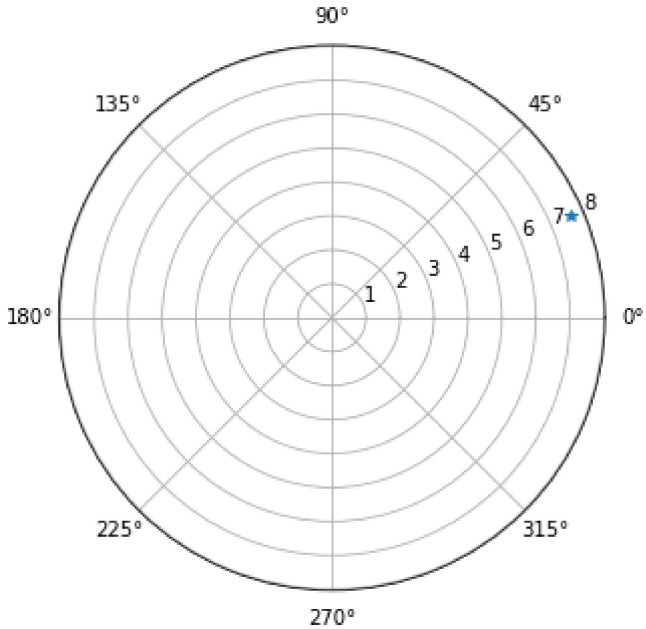
```

a1=[n.real for n in list]

b1=[n.imag for n in list]

plt.scatter(a1,b1,color="Red")

for j,n in enumerate(list):
    plt.text(a1[j],b1[j],str(n))

plt.xlabel("Real")
plt.ylabel("Imaginary")
plt.title("Plotting of Complex Numbers")
plt.show()
```



Plotting Complex number on a complex plane

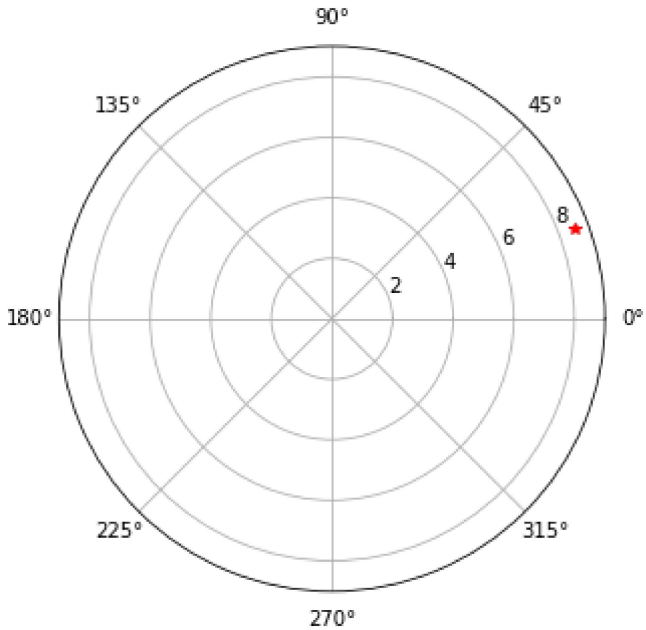
In [17]:

```
## 1. complex(4,3)

from matplotlib.pylab import *
figure(figsize=(6,5))
b=complex(7,3)

polar(angle(b),abs(b), marker='*')
show()
```



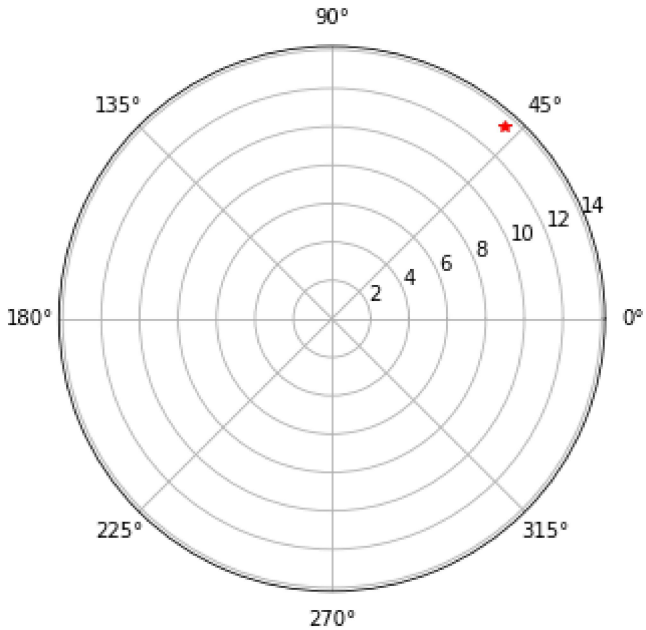
In [5]:

```
## 2. complex(8,3)

from matplotlib.pylab import *
figure(figsize=(6,5))
c=complex(8,3)

polar(angle(c),abs(c), marker='*', color="red")
show()
```



In [18]:

```
## 3. complex(9,5)

from matplotlib.pylab import *
figure(figsize=(6,5))
c=complex(9,10)

polar(angle(c),abs(c), marker='*', color="red")
show()
```



Prove the following:

For complex nos, z1, z2, z3. Prove the following:

a. Commutative, Associative Laws for multiplication

b. |z1+z2| ≤ |z1| + |z2|

c. Distributive laws of addition over multiplication

d. z1 +- z1 bar belongs to R

e. z1.z2 belongs to R

In [19]:

```
# 7. For complex nos, z1, z2, z3. Prove the following:
```

In [23]:

```
# a. Commutative, Associative Laws for multiplication Cube roots of Ur

import cmath

x1= 5
y1 = 3

z1 = complex(x1,y1)

x2=4
y2=2

z2=complex(x2,y2)

x3=1
y3=2

z3=complex(x3,y3)
```

Commutative Law

In [29]:

```

if z1*z2== z2*z1:
    print("Commutative Law is satisfied")
else:
    print("Commutative law is not satisfied")

```

```
Commutative Law is satisfied
```

Associative Law

In [30]:

```
if (z1*z2)*z3==z1*(z2*z3):
    print("Associative Law is satisfied")
else:
    print("Associative law is not satisfied")
```

```
Associative Law is satisfied
```

In [31]:

```
## c. Distributive laws of addition over multiplication

if z1*(z2+z3)== z1*z2+z1*z3:
    print("Distributive property is satisfied")
else:
    print("Not satisfied")

```

```
Distributive property is satisfied
```

In [13]:
```
## d. z1 +- z1 bar belongs to R

a=complex(3,6)
a_c=a.conjugate()
a+a_c

```

Out[13]: (6+0j)

In [16]:
```
## e.

a*a_c
```

Out[16]: (45+0j)

Conclusion:

The lab practical enhanced the understanding of complex numbers by teaching how to perform basic operations, express them in different forms, and plot them on the complex plane. The application of laws like Commutative, Associative, and Distributive was also shown to hold true for complex numbers.

LAB 3:

Limit of a complex sequence

Date: 16/01/23 to 28/01/23

Aim:

The concept of limits of complex fuctions, plotting the same and customizing them is the main objective of the lab.

In [1]:
```
from sympy import *
from numpy import *
```

In [2]:

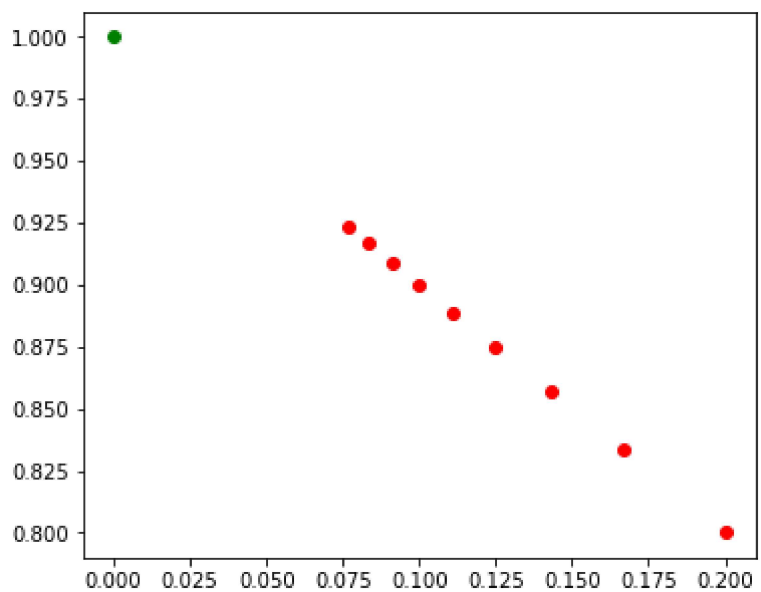
```
z = Symbol('z')
real_z=eval(input('enter real part of function in terms of z= '))
img_z=eval(input('enter imaginary part of function in terms of z= '))

limit_real_z=limit(real_z,z,inf)
limit_img_z=limit(img_z,z,inf)

limit_z=complex(limit_real_z,limit_img_z)
print('Limit of ',real_z + 1j*img_z,'is',limit_z)
```

```
enter real part of function in terms of z= z
enter imaginary part of function in terms of z= 1/z
Limit of  z + 1.0*I/z is (inf+0j)
```

In [3]:

```
import matplotlib.pyplot as plt
z=Symbol('z')
a=complex(limit((1)/z,z,inf),limit((z-1)/z,z,inf))
print(a)
x=[]
y=[]
l=int(input("Enter the lower limit of range: "))
u=int(input("Enter the lower limit of range: "))
b=l
plt.figure(figsize=(6,5))
for i in range(0,u-1):
    re = 1/b
    im = (b-1)/b
    x.append(re)
    y.append(im)
    b+=1
plt.scatter(x,y,color='r')
plt.scatter(a.real,a.imag,color='g')
plt.show()
```

```
1j
Enter the lower limit of range: 5
Enter the lower limit of range: 10
```



Find the limit of the following the complex sequences

In [2]:

```
## 1. i(n^2/(n^1+1))

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: 0
Enter imaginary part of function in terms of n: n**2/(n**1 + 1)
Limit of  1.0*I*n**2/(n + 1) is  infj
```

In [4]:

```
## 2. (1+(4/n))^n + i(7n/(n+4))

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: (1+(4/n))**n
Enter imaginary part of function in terms of n: (7*n/(n+4))
Limit of  7.0*I*n/(n + 4) + (1 + 4/n)**n is  (54.598150033144236+7j)
```

In [5]:

```
## 3. sqrt(n+1) - i(sqrt(n) + 3)

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: (n+1)**(1/2)
Enter imaginary part of function in terms of n: -((n)**(1/2) + 3)
Limit of  1.0*I*(-n**0.5 - 3) + (n + 1)**0.5 is  (inf-infj)
```

In [2]:

```
## 4. 1/2^n + n((1-i)/2)

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: 1/2**n + n/2
Enter imaginary part of function in terms of n: -n/2
Limit of  n/2 - 0.5*I*n + 2**(-n) is  (inf-infj)
```

In [3]:

```
## 5. n((3+4i)^2 / 5)

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: -(7*n/5)
Enter imaginary part of function in terms of n: (24*n/5)
Limit of  -7*n/5 + 4.8*I*n is  (-inf+infj)
```

In [4]:

```
## 6. (3/2 + 5/n)^(n+1) + i((5+n)/n)

n=Symbol('n')
real_z=eval(input("Enter real part of function in terms of n: "))
imag_z=eval(input("Enter imaginary part of function in terms of n: "))

limit_real_z=limit(real_z,n,inf)
limit_imag_z=limit(imag_z,n,inf)

limit_z=complex(limit_real_z,limit_imag_z)

print('Limit of ',real_z + 1j*imag_z,'is ',limit_z)
```

```
Enter real part of function in terms of n: (3/2 + 5/n)**(n+1)
Enter imaginary part of function in terms of n: ((5+n)/n)
Limit of  (1.5 + 5/n)**(n + 1) + 1.0*I*(n + 5)/n is  (inf+1j)
```

7. Plot the sequences as 2D plot with the following customizations:

a. Limit has a unique color and marker of higher size than that of sequence.

b. Label the limit such that "lim f(z) = value of the limit computed"

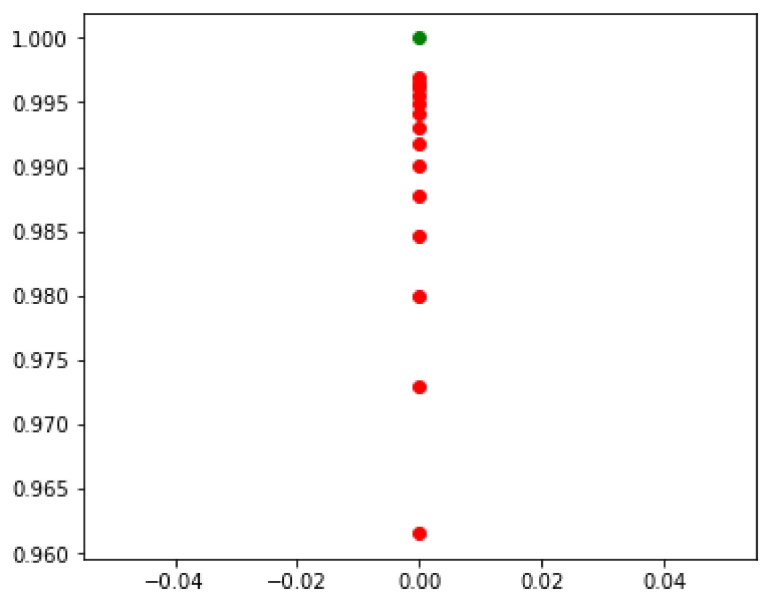
c. Appropriate labels for axes and plot title

If the sequence does not have a limit, state the reason.

In [5]:

```
import matplotlib.pyplot as plt

n=Symbol('n')
a=complex(limit(0,n,inf),limit(n**2/(n**2+1),n,inf))

print(a)

x=[]
y=[]

l=int(input("Enter the lower limit of range: "))
u=int(input("Enter the lower limit of range: "))
b=l
plt.figure(figsize=(6,5))
for i in range(0,u-1):
    re = 0
    im = b**2/(b**2+1)
    x.append(re)
    y.append(im)
    b+=1
plt.scatter(x,y,color='r')
plt.scatter(a.real,a.imag,color='g')
plt.show()
```

```
1j
Enter the lower limit of range: 5
Enter the lower limit of range: 15
```



Conclusion:

In summary, this lab practical improved our understanding of complex function and sequence limits. We were taught to use different methods, including L'Hopital's rule, the squeeze theorem, and algebraic manipulation, to determine the limits. Additionally, we learned to use matplotlib.pyplot library to create visualizations of the limits, which aided in comprehending the functions' behavior near the limits.

LAB 4:

Analytic function and Cauchy-Reimann equations

Date: 30/01/23 to 04/02/23

Aim:

The purpose of this exercise is to comprehend the concept of the Cauchy-Riemann equations and develop skills in verifying if functions meet the equations using loops. Additionally, we aimed to practice solving several problems associated with the Cauchy-Riemann equations.

In [1]:

```
from sympy import *
from numpy import *

x,y,u,v=symbols('x, y, u, v')

u=eval(input("Enter the Real part of the complex number: "))
v=eval(input("Enter the Imaginary part of the Complex number: "))
```

```
Enter the Real part of the complex number: 3*x + 2*y
Enter the Imaginary part of the Complex number: x**2 - 4*y
```

In [2]:

```
a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and b==-c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")

```

```
Does not satisfy Cauchy-Reyman
```

1. f(z) = x^2 + y^2 - 2xyi

2. f(z) = 2xi + 2xy
3. f(z) = 2x + ixy^2
4. f(z) = xy + iy
5. f(z) = (x+iy)/(x^2 + y^2)
6. f(z) = (x -iy^)^3
7. f(z) = z bar
8. f(z) = z bar/ mod z
9. Verify the Algebraic Properties of Limit of Complex functions

(a) Sum of limit of Sequence = limit of the sum of sequence

(b) lim(c f(z)) = c[lim f(z)]

10. Check CR eqn for:

a) e^z

b) (x^2 + y^2) + i(2xy)

c) f(z) = z

d) f(z) = z^2

e) f(z) = z^2 + z

In [27]:

```
# 1. f(z) = x^2 + y^2 - 2xyi

u=x**2 + y**2
v= -2*x*y

a=u.diff(x)

b=v.diff(x)

c=u.diff(y)

d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [28]:

```
# 2. f(z) = 2xi + 2xy

u= 2*x*y
v= 2*x

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [29]:

```
# 3. f(z) = 2x + ixy^2

u= 2*x
v= x*y**2

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [31]:

```
# 4. f(z) = xy + iy

u= x*y
v= y

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [33]:

```
# 5. f(z) = (x+iy)/(x^2 + y^2)

u= x/(x**2 + y**2)
v= y/(x**2 + y**2)

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [34]:

```
# 6. f(z) = (x -iy^)^3

u= x/(x**2 + y**2)
v= y/(x**2 + y**2)

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [35]:

```
# 7. f(z) = z bar

u= x
v= -y

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [36]:

```
# 8. f(z) = z bar/ mod z

u= x/(x**2 + y**2)**1/2
v= -y/(x**2 + y**2)**1/2

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

```
Does not satisfy Cauchy-Reyman
```

In [ ]:

```
u= x/(x**2 + y**2)
v= y/(x**2 + y**2)

a=u.diff(x)
b=v.diff(x)
c=u.diff(y)
d=v.diff(y)

if (a==d and -b==c):
    print("Satisfies Cauchy Reyman")
else:
    print("Does not satisfy Cauchy-Reyman")
```

Conclusion:

In conclusion, this lab practical provided us with a solid understanding of the Cauchy-Riemann equations and how to use loops to check if functions satisfy the equations. We practiced solving a range of problems related to the Cauchy-Riemann equations, which helped us to develop our analytical skills and ability to apply the concepts to real-world problems.

LAB 5 :

Harmonic Function and Harmonic Conjugates

Date: 06/02/23 to 18/02/23

Aim:

The goal of this lab is to comprehend the ideas of harmonic function and harmonic conjugates and verify whether a real or imaginary part of a complex function is a harmonic conjugate of another.

Harmonic Function

In [3]:

```
from sympy import *

# for the best pretty printing , use the init_printing() function

init_printing (use_latex=True)

#define harmonic

x,y= symbols('x,y')

def harmonic(q):
    print("The function is: ",q)
    dqdx = diff(q,x,2)
    dqdy = diff(q,y,2)
    if dqdx + dqdy == 0:
        print('Thus is a harmonic function.')
    else:
        print('This is a not a harmonic function.')

```

In [4]:

```
# 1. x**3 - 3*x*y**2

harmonic(x**3 - 3*x*y**2)
```

```
The function is:  x**3 - 3*x*y**2
Thus is a harmonic function.
```

In [5]:

```
## 2.  x^2 - y^2 + x + 1

harmonic(x**2 - y**2 + x + 1)
```

```
The function is:  x**2 + x - y**2 + 1
Thus is a harmonic function.
```

In [6]:

```
# 3. x^2y - y^2 x

harmonic(x**2*(y) - y**2*(x))
```

```
The function is:  x**2*y - x*y**2
This is a not a harmonic function.
```

Harmonic Conjugates

Let u(x,y) be a harmonic function then a function v(x,y) is said to be a harmonic conjugate of u(x,y) if

1. v(x,y) is harmonic
2. u and v satisfies the CR equations

In [16]:

```
def harmonic_conjugate(u,v):
    print("the function is:",u)
    dudx=diff(u,x,2)
    dudy=diff(u,y,2)
    if dudx+dudy==0:
        print("The function is harmonic")
        ux=u.diff(x)
        vy=v.diff(y)
        uy=u.diff(y)
        vx=v.diff(x)
        if (ux==vy and uy==-vx):
            print("The function is harmonic conjugate")
        else:
            print("The function is harmonic but does not satisfy CR ec
    else:
        print("The function is not harmonic itself, thus is not a harm
```

In [17]:

```
## 1. Check if -x^2/ 2 + y^2/2 is the harmonic conjugate of x^2 - y^2

harmonic_conjugate(-x**2/2 + y**2/2, x**2 - y**2 + x + 1)
```

```
the function is: -x**2/2 + y**2/2
The function is harmonic
The function is harmonic but does not satisfy CR equation
```

In [18]:

```
harmonic_conjugate(x**2 - y**2 + x + 1,-x**2/2 + y**2/2)
```

```
the function is: x**2 + x - y**2 + 1
The function is harmonic
The function is harmonic but does not satisfy CR equation
```

In [19]:

```
## 2. Check if -x^2/2 + y^2/2 is the harmonic conjuagte of xy

harmonic_conjugate(-x**2/2 + y**2/2, x*y)
```

```
the function is: -x**2/2 + y**2/2
The function is harmonic
The function is harmonic but does not satisfy CR equation
```

In [14]:

```
from sympy import *

import math

def conjugate():
    init_printing(use_latex=True)
    x,y,c = symbols("x,y,c")
    u=input("Enter the function u: ")
    print(" ")
    dx2=diff(u,x,2)
    dy2=diff(u,y,2)
    if dx2 + dy2 == 0:
        print("The function ", u, "is harmonic.")
        dux=diff(u,x) #xy
        duy=diff(u,y) #-vx
        iv=integrate(-duy, x)
        vy=diff(iv,y)
        gy=dux - vy
        ig=integrate(gy,y)
        v=iv + ig
        print("The harmonic conjugate of u is: ")
        display(v)
    else:
        print("The function ", u, "is not harmonic.")
```

1. Check if -x^2/2 + y^2/2 is the harmonic conjugate of x^2 - y^2 + x + 1
2. Check if -x^2/2 + y^2/2 is the harmonic conjugate of xy
3. Check if x^2 - y^2 is the harmonic conjugate of 2xy + 3
4. Check if x^2 - y^2 is the harmonic conjugate of 2xy

In [19]:

```
from sympy import *
import math
init_printing(use_latex=True)
x,y,c = symbols('x,y,c')
```

In [22]:

```
def harmonic_conjugate(u,v):
    print("the function is:",u)
    dudx=diff(u,x,2)
    dudy=diff(u,y,2)
    if dudx+dudy==0:
        print("The function is harmonic")
        ux=u.diff(x)
        vy=v.diff(y)
        uy=u.diff(y)
        vx=v.diff(x)
        if (ux==vy and uy==-vx):
            print("The function is a harmonic conjugate")
        else:
            print("The function is harmonic but does not satisfy the C
    else:
        print("The function is not harmonic itself, thus it is not a h
```

In [23]:

```
# 1. Check if -x^2/2 + y^2/2 is the harmonic conjugate of x^2 - y^2 +

harmonic_conjugate(x**2 - y**2 + x +1, -x**2/2 + y**2/2)
```

```
the function is: x**2 + x - y**2 + 1
The function is harmonic
The function is harmonic but does not satisfy the CR equations
```

In [24]:

```
# 2. Check if -x^2/2 + y^2/2 is the harmonic conjugate of xy

harmonic_conjugate(x*y, -x**2/2 + y**2/2)
```

```
the function is: x*y
The function is harmonic
The function is a harmonic conjugate
```

In [25]:

```
# 3. Check if x^2 - y^2 is the harmonic conjugate of 2xy + 3

harmonic_conjugate(2*x*y + 3, x**2 - y**2)
```

```
the function is: 2*x*y + 3
The function is harmonic
The function is harmonic but does not satisfy the CR equations
```

In [26]:

```
# 4. Check if x^2 - y^2 is the harmonic conjugate of 2xy

harmonic_conjugate(x**2 - y**2, 2*x*y)
```

```
the function is: x**2 - y**2
The function is harmonic
The function is a harmonic conjugate
```

Find the harmonic conjugate of

1. x^3 - 3xy^2
2. x^2 - y^2 + 2

3. e^x sin(y)
4. x^2 y + 2x

In [27]:

```
import math

def conjugate(u):
    init_printing(use_latex=True)
    x,y,c = symbols("x,y,c")
    u=input("Enter the function u: ")
    print(" ")
    dx2=diff(u,x,2)
    dy2=diff(u,y,2)
    if dx2 + dy2 == 0:
        print("The function ", u, "is harmonic.")
        dux=diff(u,x) #xy
        duy=diff(u,y) #-vx
        iv=integrate(-duy, x)
        vy=diff(iv,y)
        gy=dux - vy
        ig=integrate(gy,y)
        v=iv + ig
        print("The harmonic conjugate of u is: ")
        display(v)
    else:
        print("The function ", u, "is not harmonic.")
```

In [14]:

```
#1 x^3 - 3xy^2

conjugate(x**3 - 3*x*y**2)
```

```
Enter the function u: x**3 - 3*x*y**2

The function  x**3 - 3*x*y**2 is harmonic.
The harmonic conjugate of u is:
```

$$3x^2y - y^3$$

In [16]:

```
#2 x^2 - y^2 + 2

conjugate(x**2 - y**2 + 2)
```

```
Enter the function u: x**2 - y**2 + 2

The function  x**2 - y**2 + 2 is harmonic.
The harmonic conjugate of u is:
```

$$2xy$$

In [17]:

```
#3 e^x sin(y)

conjugate(e**x *sin(y))
```

```
Enter the function u: e**x *sin(y)

The function  e**x *sin(y) is not harmonic.
```

In [41]:

```
#4 x^2 y + 2x

conjugate((x**2)*y + 2*x)
```

```
Enter the function u: (x**2)*y + 2*x

The function  (x**2)*y + 2*x is not harmonic.
```

Conclusion

In conclusion, this lab practical helped us to develop a better understanding of harmonic functions and harmonic conjugates. We learned how to identify whether an imaginary or real part of a complex function is a harmonic conjugate of another function. We also learned about the Laplace equation and its relationship to harmonic functions.

LAB 6

MILNE THOMSON METHOD

Date - 20/02/23 to 04/03/23

Aim:

The aim of this lab is to demonstrate the method of Milne-Thomson method which is used to find real or imaginary part of a complex function.

It is a method for finding a holomorphic function whose real and imaginary part is given. let f be a complex variable function given by $f(z) = u(x, y) + iv(x, y)$ where $z = x + iy$.

Then f(z) can be constructed if either u or v is given

Case 1: When u is given.

Step 1: Find $\frac{\partial u}{\partial x}$ and equate it to u_x

Step 2: Find $\frac{\partial u}{\partial y}$ and equate it to u_y

Step 3: Find $f'(x, y)$ by the formula $f'(x, y) = u_x - iu_y$

Step 4: Replace x by z and y by 0 in $f'(x)$ to get $f'(z, 0)$

Step 5: Find $f(z)$ by the formula $f(z) = \int f'(z, 0)dz + c$

Case 2: When v is given.

Step 1: Find $\frac{\partial v}{\partial x}$ and equate it to v_x

Step 2: Find $\frac{\partial v}{\partial y}$ and equate it to v_y

Step 3: Find $f'(x, y)$ by the formula $f'(x, y) = v_x - iv_y$

Step 4: Replace x by z and y by 0 in $f'(x)$ to get $f'(z, 0)$

Step 5: Find $f(z)$ by the formula $f(z) = \int f'(z, 0)dz + c$

Example: Considering $e^x(xcosy + ysiny)$ as

(i) the real part and

(ii) the imaginary part

of the corresponding function, find the analytic function for the same

In [10]:

```
from sympy import *

x,y,z=symbols('x,y,z')

def u_giv(u):
    ux=diff(u,x)
    print("ux=",ux)
    uy=diff(u,y)
    print("uy=",uy)
    fp=simplify(ux-1j*uy)
    print("f'(x,y)=",str(fp))
    g=fp.subs({x:z,y:0})
    print("f'(z,0)=",g)
    f=integrate(g,z)
    return("f="+str(f))

def v_giv(v):
    vx=diff(v,x)
    print("vx=",vx)
    vy=diff(v,y)
    print("vy=",vy)
    fp=vy+1j*vx
    print("f'(x,y)=",str(fp))
    g=fp.subs({x:z,y:0})
    print("f'(z,0)=",g)
    f=integrate(g,z)
    return("f="+str(f))

pprint(u_giv(exp(x)*(x*cos(y) - y*sin(y))))
print()

pprint(v_giv(1j*exp(x)*(x*cos(x))- y*sin(y)))
```

```
ux= (x*cos(y) - y*sin(y))*exp(x) + exp(x)*cos(y)
uy= (-x*sin(y) - y*cos(y) - sin(y))*exp(x)
f'(x,y)= (x*cos(y) - y*sin(y) + 1.0*I*(x*sin(y) + y*cos(y) + sin(y)) + co
s(y))*exp(x)
f'(z,0)= (z + 1)*exp(z)
f=z*exp(z)

vx= -1.0*I*x*exp(x)*sin(x) + 1.0*I*x*exp(x)*cos(x) + 1.0*I*exp(x)*cos(x)
vy= -y*cos(y) - sin(y)
f'(x,y)= -y*cos(y) + 1.0*I*(-1.0*I*x*exp(x)*sin(x) + 1.0*I*x*exp(x)*cos
(x) + 1.0*I*exp(x)*cos(x)) - sin(y)
f'(z,0)= 1.0*I*(-1.0*I*z*exp(z)*sin(z) + 1.0*I*z*exp(z)*cos(z) + 1.0*I*ex
p(z)*cos(z))
f=-1.0*z*exp(z)*cos(z)
```

Conclusion:

Milne Thomson method is another method for computing real or imaginary part and the workings of the same was seen through this lab

LAB 7

ORTHOGONAL FAMILIES OF CURVE

Date: 05/03/2023

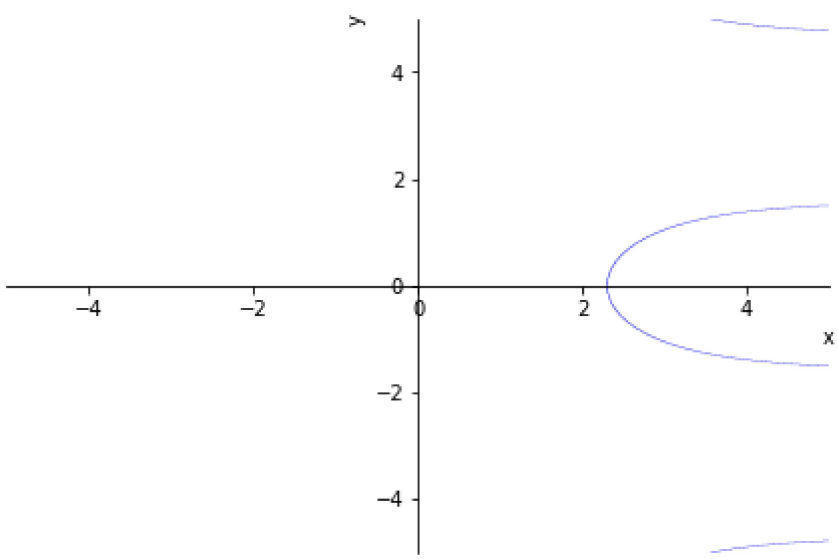
Orthogonal Families of curves

Suppose the function $f(z) = u(x, y) + iv(x, y)$ is an analytic in some domain D. Then the real and imaginary parts of f can be used to define two families of curves in D. The equations $u(x, y) = c1$ and $v(x, y) = c2$ where $c1$ and $c2$ are arbitrary real constants, are called level curves of u and v respectively. The level curves are orthogonal families.

1. Suppose the function $f(z) = e^x cosy + ie^y sinx$ is analytic in some domain D. Write a python program to find the orthogonal family of curves corresponding to real and imaginary parts of the function. Also plot the curves.

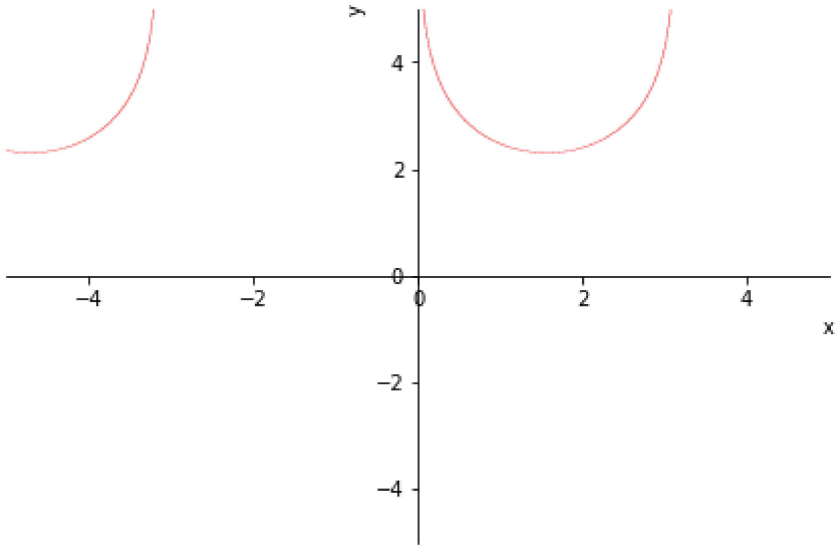
In [2]:

```
from numpy import *
from sympy import *

x,y = symbols('x,y')
p1=plot_implicit(Eq(exp(x)*cos(y),10), line_color='blue', line_width=4
None
```



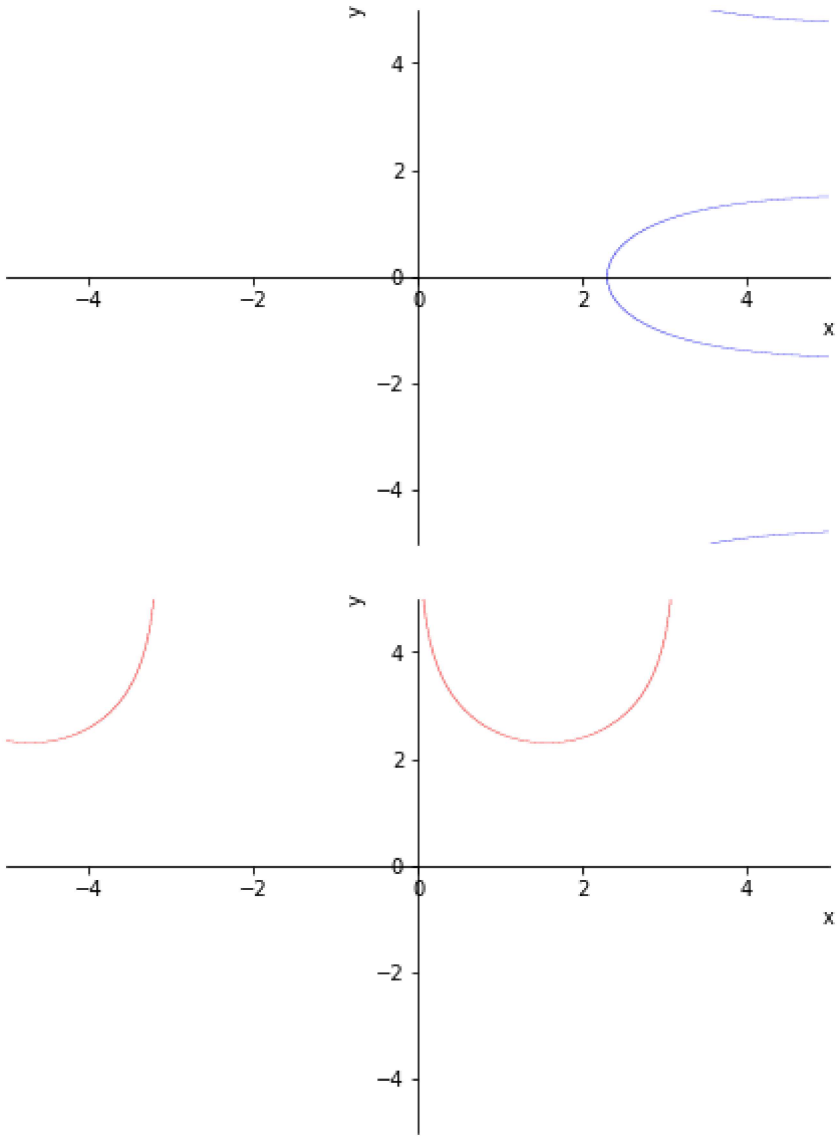
In [4]:

```
from numpy import *
from sympy import *

x,y = symbols('x,y')
p2=plot_implicit(Eq(exp(y)*sin(x),10), line_color='red', line_width=4)
None
```



In [6]:

```
from numpy import *
from sympy import *

x,y = symbols('x,y')
p1=plot_implicit(Eq(exp(x)*cos(y),10), line_color='blue', line_width=4
p2=plot_implicit(Eq(exp(y)*sin(x),10), line_color='red', line_width=4)
None
p1.extend(p2)
p1
p1.show
None
```



2. $x^3y - xy^3 = C$

In [7]:

```
import numpy as np
def ortho_family(x,y):
    u=x**3* y
    v=x*y**3
    return u,v
```

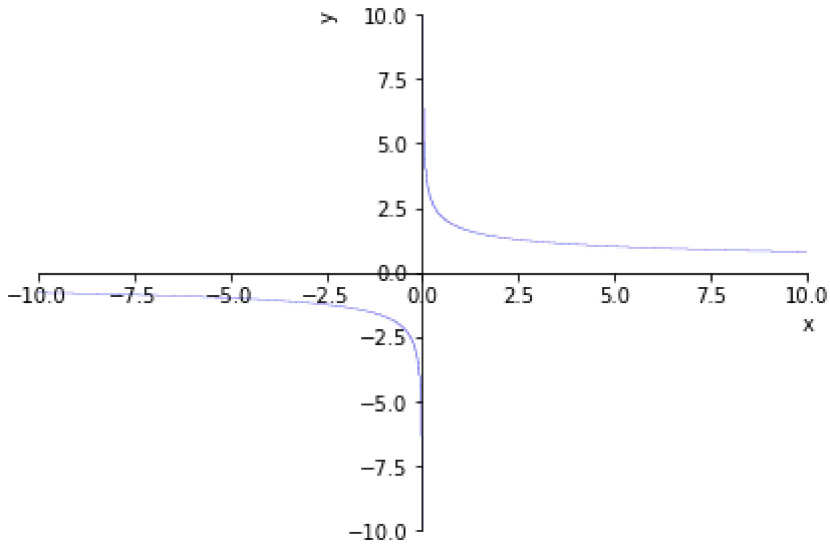
In [8]:

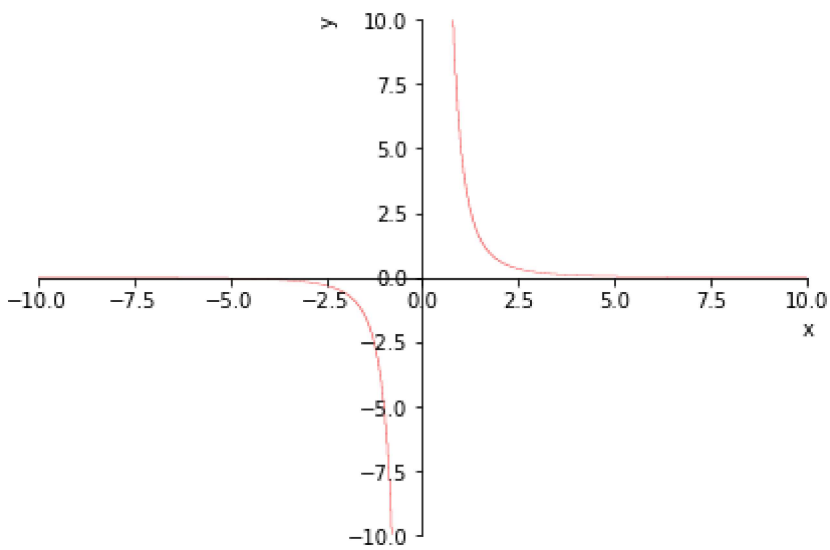
```
ortho_family(2,4)
```

Out[8]: (32, 128)

In [12]:

```
from numpy import *
from sympy import *
import matplotlib.pyplot as plt
x,y = symbols('x,y')
p1=plot_implicit(Eq(x*y**3,5),(x,-10,10),(y,-10,10), line_color='blue'
p2=plot_implicit(Eq(x**3*y,5),(x,-10,10),(y,-10,10), line_color='red',
None
p1.extend(p2)
p1
p1.show
None
```



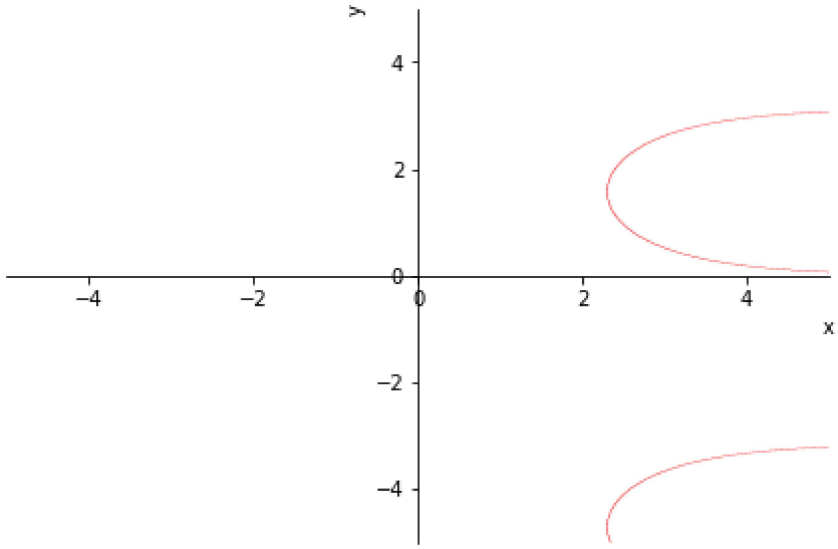


For the following problems find the corresponding conjugate and plot the orthogonal families of curves:

3. $e^x sin(y) = c$
4. $e^{(} - x)cos(y) + xy = C$
5. $sin(x)cos(hx)$

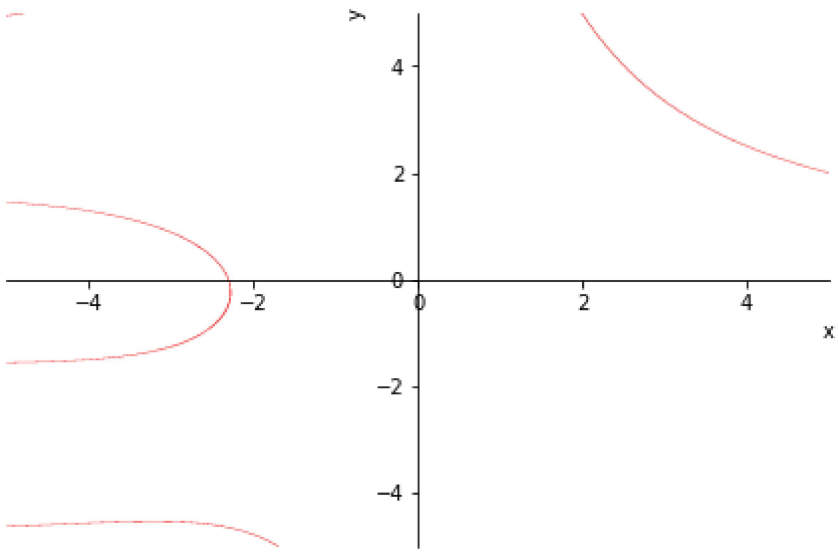
In [2]:

```
from numpy import *
from sympy import *

x,y = symbols('x,y')
p1=plot_implicit(Eq(exp(x)*sin(y),10), line_color='red', line_width=4)
None
```



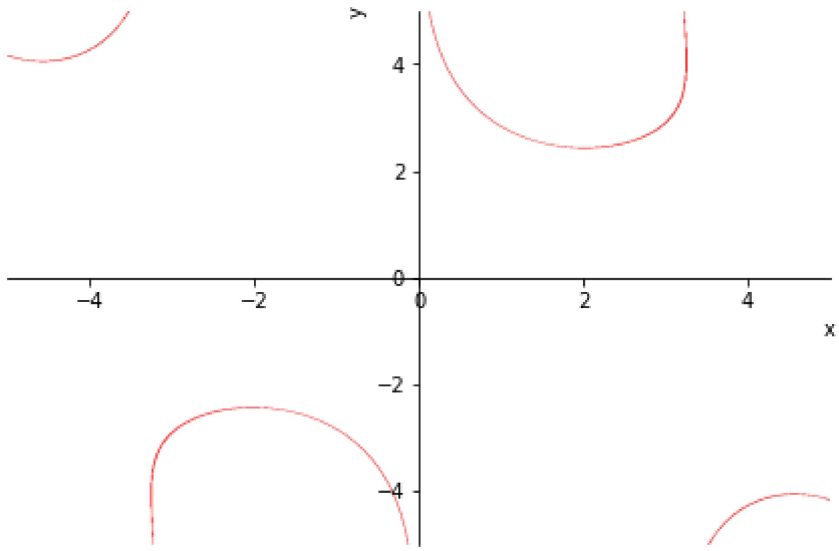
In [3]:

```
from numpy import *
from sympy import *

x,y = symbols('x,y')
p1=plot_implicit(Eq(exp(-x)*cos(y) + x*y,10), line_color='red', line_w
None
```



In [6]:

```
x,y = symbols('x,y')
p1=plot_implicit(Eq(sin(x)*sinh(y) + x*y,10), line_color='red', line_w
None
```



Conclusion:

In this lab, we learnt the concept of orthogonal families of curves, found the conjugate of the functions and plotted the orthogonal curves.

Lab 8

Conformal Mapping

27.03.23

Aim:

The aim of this lab is to learn about an angle-preserving transformation that is conformal mapping and checking if various functions are conformal.

A conformal mapping, also called a conformal map, conformal transformation, angle-preserving transformation, or biholomorphic map, is a transformation. that preserves local angles. An analytic function is conformal at any point where it has a nonzero derivative.

A transformation is conformal when T(z) is analytic and T'(z) is not equal to 0. Algo

1. Defome x y z as symbols
2. define function in terms of z and real and imagianry parts seperate.
3. check whether the given function satisfies CR equation or not. If yes set a=1.
4. check whether the given function have any critical points or not . If no, set b=1.

5. if a=1, b=1, the functions is conformal.

In [1]:

```
import math
import cmath
import sympy as sp

#### check if following are conformal.

x,y,z=sp.symbols('x,y,z')
```

Q1 $e^z = e^x(cosy + isiny)$

In [3]:

```
f=sp.exp(z)
r=sp.exp(x)*sp.cos(y)
i=sp.exp(x)*sp.sin(y)
du_dx=sp.diff(r,x)
dv_dy=sp.diff(i,y)
du_dy=sp.diff(r,y)
dv_dx=sp.diff(i,x)
if du_dx==sp.simplify(dv_dy) and du_dy==sp.simplify(-dv_dx):
    a=1
else:
    a=0
print("a=",a)
d=sp.diff(f)
print("derivative=",d)
s=sp.solve(d,z)
print('critical points=',s)
if len(s)==0:
    b=1
else:
    b=0
print('b=',b)
if a==1 and b==1:
    print("This is a conformal mapping")
else:
    print("this is not a conformal mapping")
```

```
a= 1
derivative= exp(z)
critical points= []
b= 1
This is a conformal mapping
```

Q2 $z^2 = x^2 + 2xiy$

In [6]:

```
f=z**2
r=x**2 - y**2
i=2*x*y

du_dx=sp.diff(r,x)
dv_dy=sp.diff(i,y)
du_dy=sp.diff(r,y)
dv_dx=sp.diff(i,x)
if du_dx==sp.simplify(dv_dy) and du_dy==sp.simplify(-dv_dx):
    a=1
else:
    a=0
print ('a=',a)
d=sp.diff(f)
print('derivative=',d)
s=sp.solve(d,z)
print('critical points = ', s)
if len(s) == 0:
    b=1
else:
    b=0
print('b=',b)
if a==1 and b==1:
    print("This is a conformal transformation")
else:
    print("This is not a conformal transformation")
```

```
a= 1
derivative= 2*z
critical points =  [0]
b= 0
This is not a conformal transformation
```

Q3 $sin(z) = cosh(-y)cos(x) + isinh(-y)sin(x)$

In [8]:

```
f=sp.sin(z)
r=sp.cosh(-y)*sp.cos(x)
i=sp.sinh(-y)*sp.sin(x)

du_dx=sp.diff(r,x)
dv_dy=sp.diff(i,y)
du_dy=sp.diff(r,y)
dv_dx=sp.diff(i,x)
if du_dx==sp.simplify(dv_dy) and du_dy==sp.simplify(-dv_dx):
    a=1
else:
    a=0
print ('a=',a)
d=sp.diff(f)
print('derivative=',d)
s=sp.solve(d,z)
print('critical points = ', s)
if len(s) == 0:
    b=1
else:
    b=0
print('b=',b)
if a==1 and b==1:
    print("This is a conformal transformation")
else:
    print("This is not a conformal transformation")
```

```
a= 1
derivative= cos(z)
critical points =  [pi/2, 3*pi/2]
b= 0
This is not a conformal transformation
```

Conclusion:

We covered the topic of conformal mapping and discussed the conditions that a function must satisfy in order to be considered a conformal transformation. Additionally, we delved into the concept of conformal transformation and explored its intricacies.

LAB 9

Transformations

03.04.23

Aim -

The aim of this lab is to learn about four basic transformations of complex numbers i.e., Reflection, Translation, Rotation/Magnification and Inversion. In this lab, we will plot an original point and its corresponding elementary transformation.

The basic elementary transformations of complex numbers are reflection, translation, rotation and dilation

Reflection

Reflection of point (5,6) with respect too x axis is (5,-6) and wrt y axis is (-5,6)

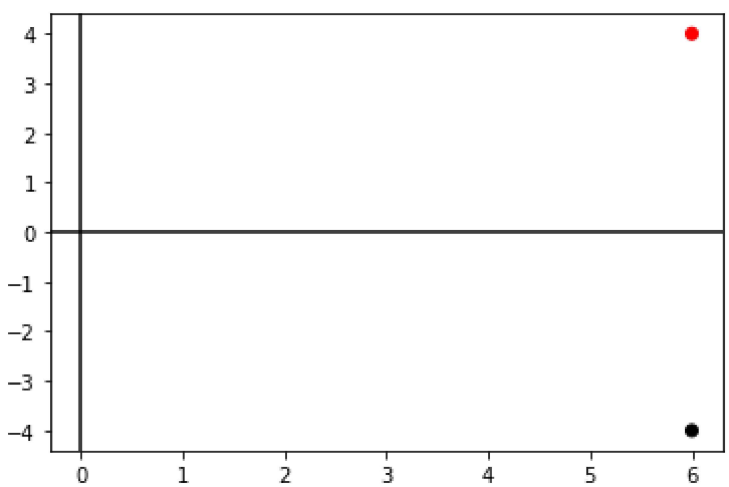
In [2]:

```
import matplotlib.pyplot as plt
x=int(input("Enter the value of x coordinate: "))
y=int(input("Enter the value of y coordinate: "))
axis=input("Reflection wrt to which axis: ")

if axis=='y':
    xref=-x
    yref=y
elif axis=='x':
    xref=x
    yref=-y

x=[x,xref]
y=[y,yref]

plt.scatter(x,y,color=['red','black'])
plt.axhline(0,color='black')
plt.axvline(0,color='black')
```

```
Enter the value of x coordinate: 6
Enter the value of y coordinate: 4
Reflection wrt to which axis: x
```

Out[2]:

```
<matplotlib.lines.Line2D at 0x26083c73310>
```



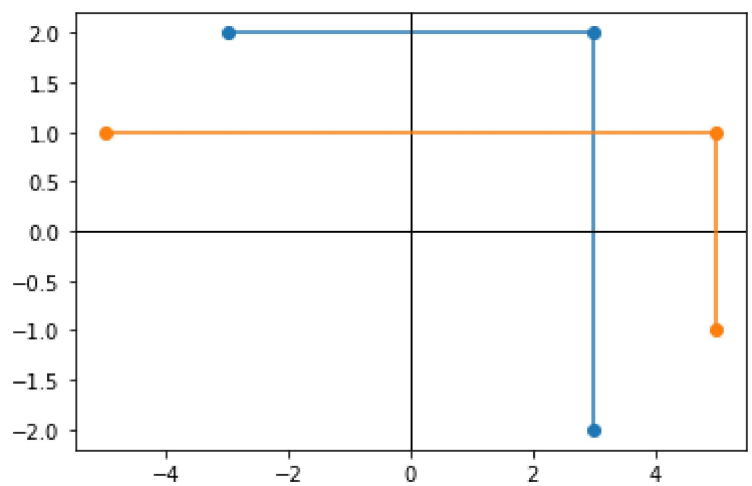
In [3]:

```
import cmath
print("The reflection of points (3,2) and (5,1)")
A=[-3,3,3]
B=[2,2,-2]
plt.plot(A,B,marker='o')
C=[-5,5,5]
D=[1,1,-1]
plt.axhline(0,linewidth=1,color='black')
plt.axvline(0,linewidth=1,color='black')
plt.plot(C,D,marker='o')
```

```
The reflection of points (3,2) and (5,1)
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x26085d8f610>]
```



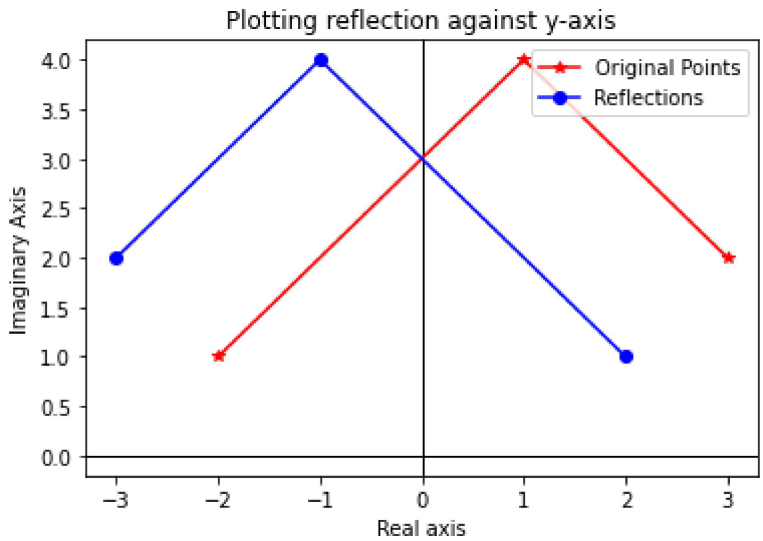
The reflection of a complex number z about the real axis is its conjugate z bar, and the reflection about imaginary axis is -z bar.

Plot the reflection points 1+4j, 3+2j, -2+j wrt to imaginary axis in plane

In [4]:

```
from pylab import *
z=[-2+1j,1+4j,3+2j]
w= [-conjugate(x) for x in z]
print(w)
X1 = [x.real for x in z]
Y1 = [x.imag for x in z]
X2 = [x.real for x in w]
Y2 = [x.imag for x in w]
plt.axhline(0,linewidth=1,color='black')
plt.axvline(0,linewidth=1,color='black')
plot(X1,Y1,color='red',label='Original Points',marker='*')
plot(X2,Y2,color='blue',label='Reflections',marker='o')
xlabel("Real axis")
ylabel("Imaginary Axis")
title("Plotting reflection against y-axis")
legend()
show()
```

```
[(2+1j), (-1+4j), (-3+2j)]
```

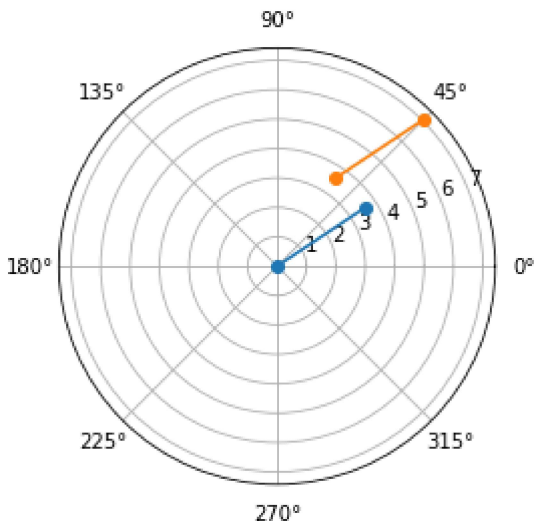


Translation

In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
def translation(x,y,p,q):
    u=complex(x,y)
    plt.polar([0,np.angle(u)],[0,np.abs(u)],marker='o',label="Original
    v=complex(p,q)
    w=u+v
    print(w)

    plt.polar([np.angle(v),np.angle(w)],[np.abs(v),np.abs(w)],marker='

translation(3,2,2,3)
```

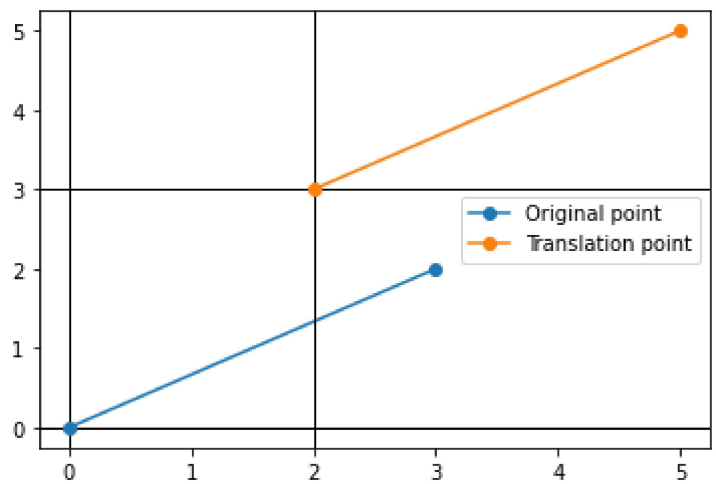
```
(5+5j)
```



In [6]:

```
## cartesian
def translation1(x,y,p,q):
    u=complex(x,y)
    plt.axhline(0,linewidth=1,color="black")
    plt.axvline(0,linewidth=1,color="black")
    plt.plot([0,x],[0,y],marker="o",label="Original point")
    v=complex(p,q)
    w=u+v
    print(w)
    plt.axhline(3,linewidth=1,color="black")
    plt.axvline(2,linewidth=1,color="black")
    plt.plot([2,w.real],[3,w.imag],marker="o",label="Translation point
    plt.legend()
translation1(3,2,2,3)
```

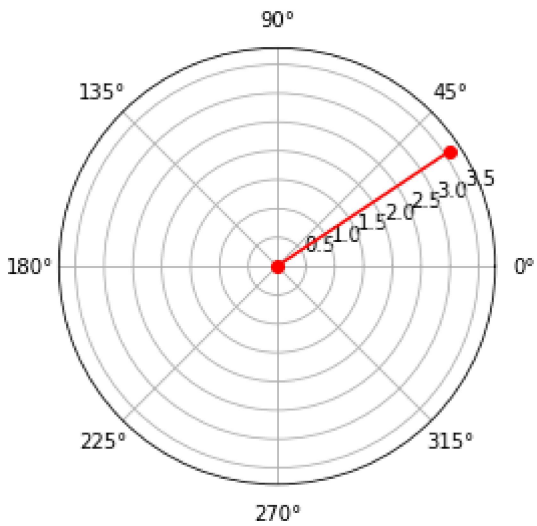
```
(5+5j)
```



Rotation

In [7]:

```
a=complex(3,2)
cc=complex(2,3)
alpha=angle(cc)
w1=a*exp(alpha*1j)
plt.polar([0,np.angle(a)],[0,np.abs(a)],marker="o",color='red')
None
```



Bilinear Transformation

Transformation of the form w= T(z)=az+b/cz+d are bilinear transformations, where ad-bc not equal to 0

In [8]:

```
from sympy import *
def BT(z1,z2,z3,w1,w2,w3):
    z=Symbol('z')
    w=Symbol('w')
    Z = ((z-z1)*(z2-z3))/((z1-z2)*(z3-z))
    W = ((w-w1)*(w2-w3))/((w1-w2)*(w3-w))
    eq = Z-W
    print("the bilinear transformation is: ")
    print((solve(eq,w)))

```

In [9]:

```
### 1. Find the bilinear transformation that takes (1,i,-1)
BT(1,1j,-1,1,0,-1)
```

```
the bilinear transformation is:
[(z - I)/(-I*z + 1.0)]
```

In [10]:

```
BT(2,1,1j,1j,1,2)
```

```
the bilinear transformation is:
[(-z*(2.0 + I) + 3.0 + 2.0*I)/(-z + 2.0 + I)]
```

In [11]:
```
BT(1,1j,20,2,9,-1)
```

```
the bilinear transformation is:
[(z*(27.0 + 160.0*I) - 407.0 - 540.0*I)/(z*(3.0 - 130.0*I) - 193.0 - 60.0
*I)]
```

In [12]:
```
BT(5,-1,-3,1,1j,-1)
```

```
the bilinear transformation is:
[(z*(2.0 + I) + 2.0 + 7.0*I)/(z*(1.0 + 2.0*I) + 7.0 + 2.0*I)]
```

Conclusion -

In this lab, we learnt about the concept of Bilinear Transformation. We learnt how The Bilinear Transformation maps points from one complex plane to another, and it is characterized by its linearity in both the numerator and the denominator of the transformation formula. It is commonly used in areas such as image warping, image scaling, and geometric transformations in computer graphics and image processing.

LAB 10

Convergence and Divergence of Complex Series

Aim:

We aim to explore the generation of Maclaruin and Taylor series using for various fucntions.

20-4-23

Maclaurin series

z=1/(1-z)

In [13]:

```
import numpy as np
from sympy import *
import math
def factorial(n):
    fact=1
    for i in range(1,n+1):
        fact=fact+1
    return fact
x,z,z0=symbols('x,z,z0')
z0=0
fun=1/(1-x)
series=0

dif=fun
for i in range(8):
    if(i!=0):
        dif=diff(dif)
        exp=dif.subs(x,0)* (z)**i/factorial(i)
        series=series+exp
    else:
        exp=dif.subs(x,0)*(z)**i/factorial(i)
        series=series+exp
pprint(series)
```

```
                6                4       3       2
     7    720⋅z         5    24⋅z    3⋅z     2⋅z     z
630⋅z  + ───────── + 20⋅z  + ──────── + ───── + ───── + ─ + 1
           7                   5       2       3       2
```

Taylor Series

z=1/(1-z)

In [15]:

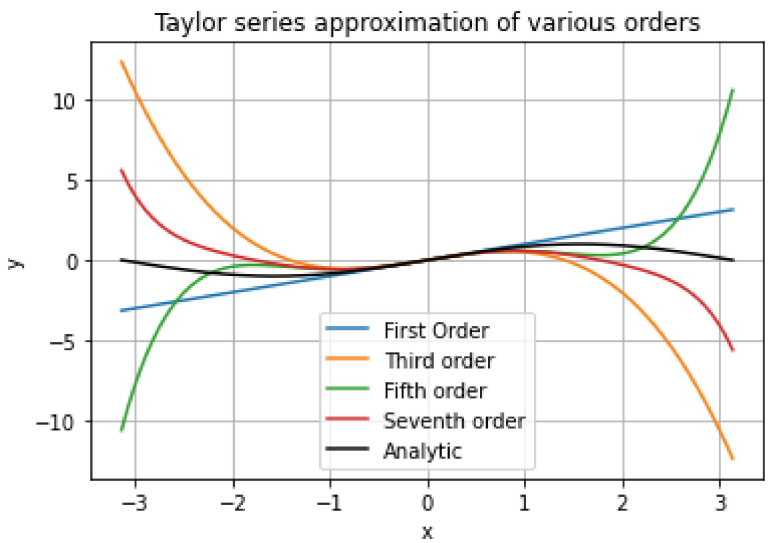
```
import numpy as np
from sympy import *
import math
def factorial(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    return fact
x,z,z0=symbols('x,z,z0')
z0=2*I
fun=1/(1-x)
series=0

dif=fun
for i in range(4):
    if(i!=0):
        dif=diff(dif)
        exp=dif.subs(x,z0)* (z-z0)**i/factorial(i)
        series=series+exp
    else:
        exp=dif.subs(x,z0)*(z-z0)**i/factorial(i)
        series=series+exp
pprint(series)
```

$$\frac{(z - 2\cdot i)^3}{(1 - 2\cdot i)^4} + \frac{(z - 2\cdot i)^2}{(1 - 2\cdot i)^3} + \frac{z - 2\cdot i}{(1 - 2\cdot i)^2} + \frac{1 + 2\cdot i}{5}$$

Graphical approximation of taylor series of various orders

In [16]:

```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt

x=np.linspace(-np.pi,np.pi,200)
y=np.zeros(len(x))

labels=['First Order','Third order','Fifth order','Seventh order']

for n,label in zip(range(7),labels):
    y=y+((-i)**n * (x)**(2*n+1)) / np.math.factorial(2*n+1)
    plt.plot(x,y,label=label)

plt.plot(x,np.sin(x),'k',label='Analytic')
plt.grid()
plt.title("Taylor series approximation of various orders")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



Conclusion:

In conclusion, during this lab, we gained an understanding of Taylor series and Maclaurin series, which is a specific case of Taylor series. We also observed graphical representations of Taylor series approximations at different orders.

LAB 11

Applications Of Complex Analysis

24.04.23

Aim:

The objective of this lab is to explore practical applications of Complex Analysis, with a specific focus on the Julia Set as one of the examples.

Julia Set

It is a set of complex numbers which do not converge to any limit when a given mapping is repeatedly applied to them. In some cases the result is a connected fractcal set.

The function julia_set takes the arguements.

1. c an arbitary complex number

2. num_iter the number of iterations to perform

3. N the number of grid points on each axis in the grid and

4. x0 is the limits of the grid.

During each step in the for loop, after z=z^2+c, we check which points in z are still smaller than infinity. The number of iterations taken by each point to blow up to iinifity is recorded in F. By plotting F as a 2 dimensional image, the Julia set finally reveals itself

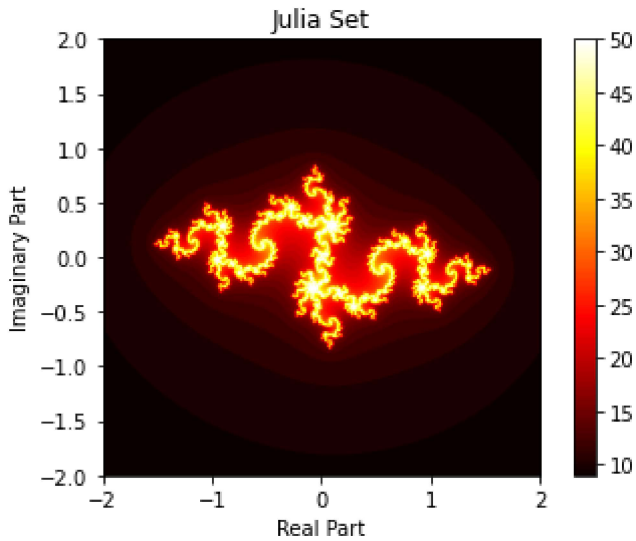
In [18]:

```
import numpy as np
import matplotlib.pyplot as plt
def julia_set(c=-0.835-0.2321*1j,num_iter=50, N=1000, X0=np.array([-2,

    """
    This function creates the Julia set.
    Inputs
        c: (Complex) arbitary complex number to add to the grid z
        nume_iter: (int) number of iterations to perform
        N: (int) Number of grid points on each axis of z
        X0: (array) limits of the grid z
    Output
        x: (Array) value of the real x axis used in the grid
        y: (Array) values of the imaginary y axis used in the grid
        F: (Array) the complex grid containing julia set

    """
    x0=X0[0]
    y0=X0[2]
    x1=X0[1]
    y1=X0[3]
    # Set up complex grid. Each element in the grid
    # is a complex number x+iy

    x,y=np.meshgrid(np.linspace(x0,x1,N),
                    np.linspace(y0,y1,N)*1j)
    z=x+y
    # F keeps track of which grid points are bounded
    # even after many iterations of z:= z**2+c
    F=np.zeros([N,N])
    # Iterate through operation z:= z**2+c
    for j in range(num_iter):
        z=z**2+c
        index=np.abs(z) < np.inf
        F[index] = F[index] + 1
    return np.linspace(x0,x1,N), np.linspace(y0,y1,N), F
```

In [17]:

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

def juliaset(c=-0.835-0.2321*1j,num_iter=50,N=1000, X0=np.array([-2,2,
    x0=X0[0]
    x1=X0[1]
    y0=X0[2]
    y1=X0[3]

    x,y=np.meshgrid(np.linspace(x0,x1,N),
                    np.linspace(y0,y1,N)*1j)

    z=x+y
    F=np.zeros([N,N])
    for j in range(num_iter):
        z=z**2+c
        index=np.abs(z)<np.inf
        F[index]=F[index]+1
    return np.linspace(x0,x1,N),np.linspace(y0,y1,N),F

# Set the parameters for the Julia set
c = -0.835 - 0.2321*1j
num_iter = 50
N = 1000
X0 = np.array([-2, 2, -2, 2])

# Generate the Julia set
x, y, F = juliaset(c, num_iter, N, X0)

# Plot the Julia set
plt.imshow(F, cmap='hot', extent=[x[0], x[-1], y[0], y[-1]])
plt.xlabel('Real Part')
plt.ylabel('Imaginary Part')
plt.title('Julia Set')
plt.colorbar()
plt.show()

```

```
<ipython-input-17-c8e6f023be1e>:16: RuntimeWarning: overflow encountered
in square
  z=z**2+c
<ipython-input-17-c8e6f023be1e>:16: RuntimeWarning: invalid value encount
ered in square
  z=z**2+c
<ipython-input-17-c8e6f023be1e>:17: RuntimeWarning: overflow encountered
in absolute
  index=np.abs(z)<np.inf
```



Conclusion:

Thus we learnt how Julia set is a fractal set of complex numbers that are associated with a certain iterative function. The set is named after the French mathematician Gaston Julia who studied its properties in the early 20th century. Julia sets are closely related to the Mandelbrot set and are used to study the dynamics of complex functions. They have applications in fields such as physics, engineering, and computer graphics.

Conclusion of MAT651

We convered various topics regarding complex analysis such as: Basics and Visualization of complex numbers:
Limit of complex number
Analytic Function
CR Equations
Harmonic Functions
Harmonic Conjugates
Milne Thmonson Method
Orthogonal Families of curve
Conformal Mapping
Bilinear Transformations

This lab helped us develop an innated understanding of the working of complex numbers, complex analysis and its various applications using Python and various of its tools like visualisation and solving complex problems.