LAB RECORD

MAT651

1740256

INDEX

1. Revision
2. Inroduction to Complex Numbers
3. Plotting complex numbers
4. $f(z)$ if $u + v/u - v$ are given
5. Check if $f(z)$ is analytic
6. Check if $f(z)$ is harmonic
7. Harmonic Conjugate
8. Bilinear Transformations

LAB 1 ~ Revision - Fundamentals of Python Programming

11/11/2019

Aim -
Check if the number entered is prime or not. If not, list all the factors of the number

In [4]:

```
def prime(n):
    print("Number entered is: ",n)
    if n>1:
        for i in range(2,n):
            if(n%i == 0):
                print(n," is not a prime number")
                print("The factors of ",n," are: ")
                for j in range(1,n+1):
                    if(n%j == 0):
                        print(j)
                break
            else:
                print(n," is a prime number")
                break
    else:
        print(n," is not a prime number")

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

```
Enter a number: 8
Number entered is:  8
8  is not a prime number
The factors of  8  are:
1
2
4
8
```

Conclusion -
From the above output, we see that 6 is not a prime number and the factors are given below.

LAB 2 ~ Basics of Complex Numbers

16/11/2019

AIM

1. Find 1 application of complex analysis

2. Define 2 complex numbers and form the arithmetic operations possible

3. Find the polar & rectangular/cartesian form of a complex number

4. Find the phase of a complex number

5. Extract the real & imaginary part from a complex number

6. Can we take sin() of a complex number?

7. Write the differenc in math & cmath library

8. Find the argument of a complex number

9. Verify Euler's formula

In [8]:

```
import cmath
import math
from cmath import *
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:

```
a1 = 2
a2 = 3
b1 = 4
b2 = 5
```

In [7]:

```
c1 = complex(a1,a2)
c2 = complex(b1,b2)
```

In [8]:

```
print("Complex Number 1: ",c1)
print("Complex Number 2: ",c2)
```

```
Complex Number 1:  (2+3j)
Complex Number 2:  (4+5j)
```

In [10]:

```
print("The sum of the complex numbers are: ",c1+c2)
print("The difference of the complex numbers are: ",c1-c2)3
print("The product of the complex numbers are: ",c1*c2)
print("Division of the complex numbers are: ",np.round(c1/c2,4))
```

```
The sum of the complex numbers are:  (6+8j)
The difference of the complex numbers are:  (-2-2j)
The product of the complex numbers are:  (-7+22j)
Division of the complex numbers are:  (0.561+0.0488j)
```

In [12]:

```
p1 = cmath.polar(c1)[0]
p2 = cmath.polar(c2)
print("The modulus & argument of 1st polar complex number is: ",p1)
print("The modulus & argument of 2nd polar complex number is: ",p2)
```

```
The modulus & argument of 1st polar complex number is:  3.605551275463989
The modulus & argument of 2nd polar complex number is:  (6.4031242374328485,
0.8960553845713439)
```

In [33]:

```
r1 = cmath.rect(3.605551275463989, 0.982793723247329)
r2 = cmath.rect(6.4031242374328485, 0.8960553845713439)
print("The rectangular form of the 1st polar complex number is: ",r1)
print("The rectangular form of the 2nd polar complex number is: ",r2)
```

```
The rectangular form of the 1st polar complex number is:  (2+2.9999999999999
996j)
The rectangular form of the 2nd polar complex number is:  (4+4.9999999999999
99j)
```

In [35]:

```
print("The phase of the 1st complex number is: ",cmath.phase(c1))
print("The phase of the 2nd complex number is: ",cmath.phase(c2))
```

```
The phase of the 1st complex number is:  0.982793723247329
The phase of the 2nd complex number is:  0.8960553845713439
```

In [39]:

```
print("The real part of the 1st complex number is: ",c1.real)
print("The imaginary part of the 1st complex number is: ",c1.imag)
print("The real part of the 2nd complex number is: ",c2.real)
print("The imaginary part of the 2nd complex number is: ",c2.imag)
```

```
The real part of the 1st complex number is:  2.0
The imaginary part of the 1st complex number is:  3.0
The real part of the 2nd complex number is:  4.0
The imaginary part of the 2nd complex number is:  5.0
```

Yes we can find the sin() of a complex number

The cmath library provides accessibility for performing operations with complex numbers

Conclusion - From the above output, we were able to find the real and imaginary part of a complex number, phase and argument of a complex number and different arithmetic operations were performed on them as well.

23/11/2019

Aim -

- -----

Draw a scatterplot, 2d & 3d plot

In [11]:

```
import numpy as np
```

In [13]:

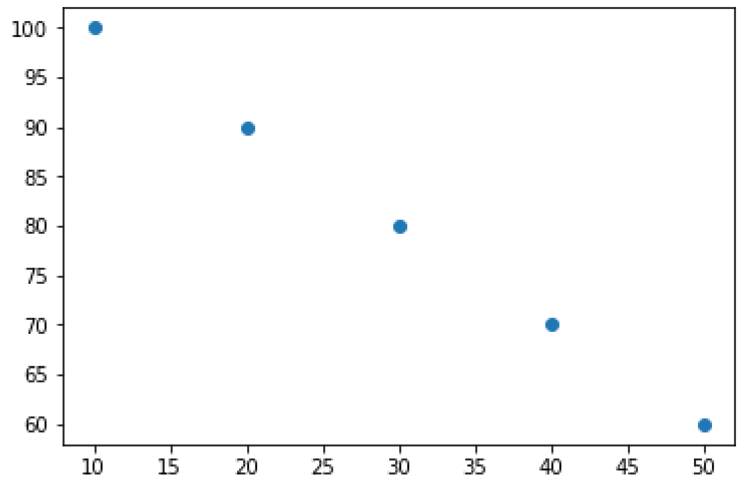
```
import matplotlib.pyplot as plt
from pylab import *
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

In [14]:

```
y = [100,90,80,70,60]
x = [10,20,30,40,50]
plt.scatter(x,y)
```

Out[14]:

```
<matplotlib.collections.PathCollection at 0x292b2ef9400>
```



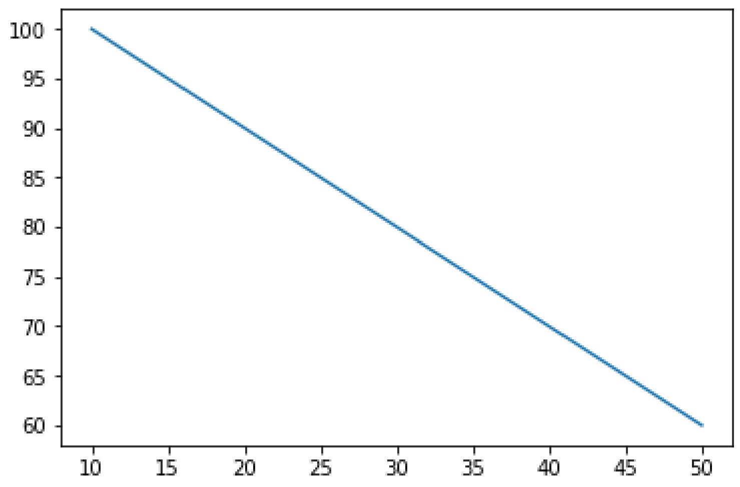
Above is a scatter plot with 2 lists x and y respectively

In [12]:

```
x = [10,20,30,40,50]
y = [100,90,80,70,60]
plt.plot(x,y)
```

Out[12]:

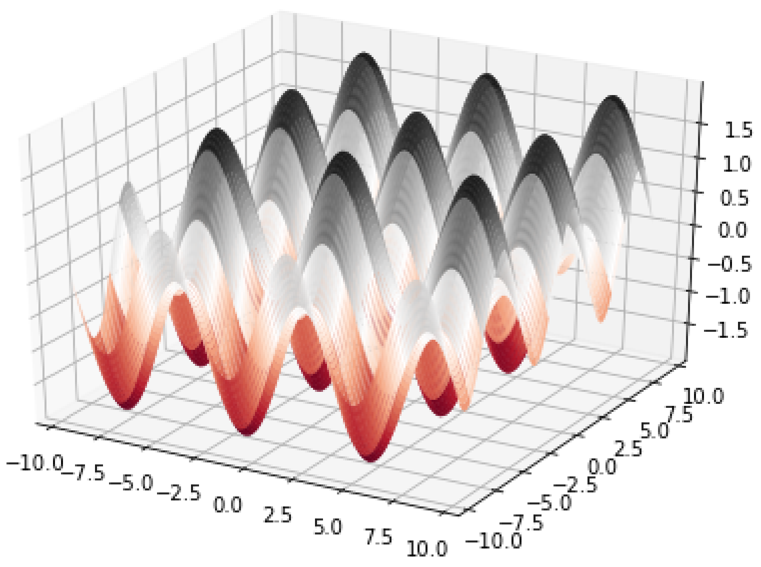
```
[<matplotlib.lines.Line2D at 0x292b2e07828>]
```



Above is a straight line on a 2d plot

In [26]:

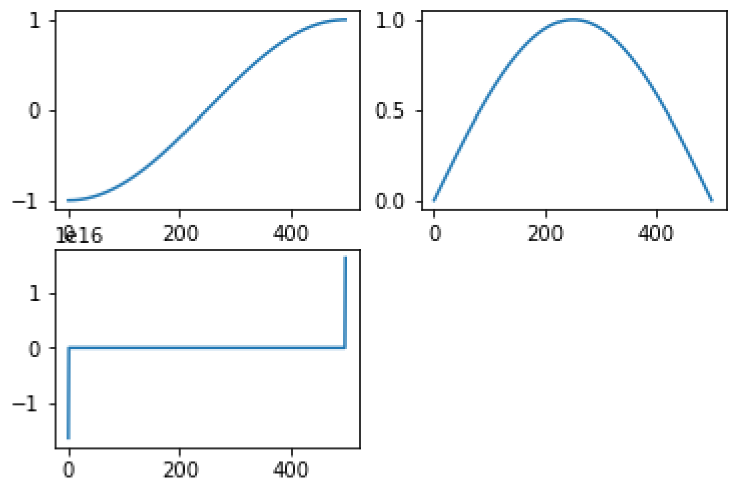
```
ax = Axes3D(figure())
x = arange(-3*pi,3*pi,0.1)
y = arange(-3*pi,3*pi,0.1)
xx,yy = meshgrid(x,y)
z = sin(xx) + sin(yy)
ax.plot_surface(xx,yy,z,cmap = 'RdGy',cstride=1)
show()
```



Above is a 3d plot with the addition of 2 trignometric identities.

In [2]:

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

x = np.linspace(-(np.pi/2),(np.pi/2),500.5)
plt.subplot(221)
plt.plot(np.sin(x))

plt.subplot(222)
plt.plot(np.cos(x))

plt.subplot(223)
plt.plot(np.tan(x))

plt.show()
```



Conclusion
Above are different subplots for different trignometric identities.

LAB 3 ~ Plotting Complex numbers

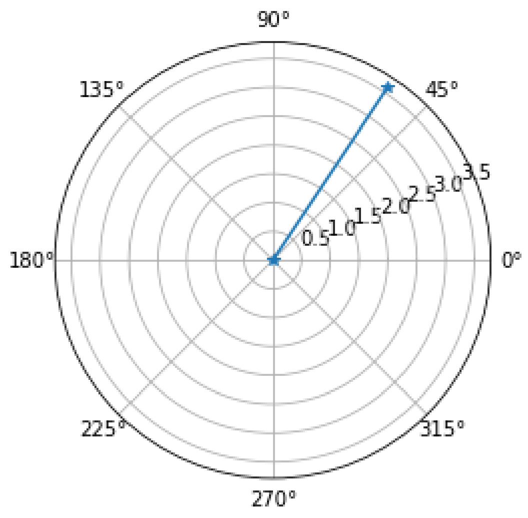
In [25]:

```
a = complex(2,3)
print("The complex number is: ",a)
ang = np.angle(a)
print("The angle of the complex value is: ",ang)
ab = np.abs(a)
print("The absolute value of the complex number is: ",ab)
plt.polar([0,ang],[0,ab],marker="*")
```

```
The complex number is:  (2+3j)
The angle of the complex value is:  0.982793723247329
The absolute value of the complex number is:  3.605551275463989
```

Out[25]:

```
[<matplotlib.lines.Line2D at 0x2286d60e438>]
```



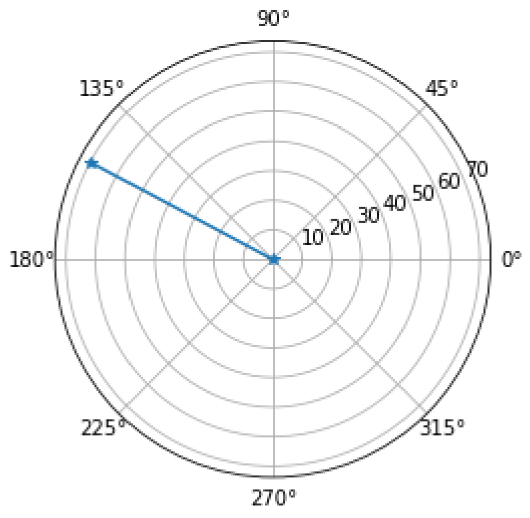
In [12]:

```
a = complex(-62,33)
print("The complex number is: ",a)
ang = np.angle(a)
print("The angle of the complex value is: ",ang)
ab = np.abs(a)
print("The absolute value of the complex number is: ",ab)
plt.polar([0,ang],[0,ab],marker="*")
```

```
The complex number is:  (-62+33j)
The angle of the complex value is:  2.6524728480782644
The absolute value of the complex number is:  70.2353187506115
```

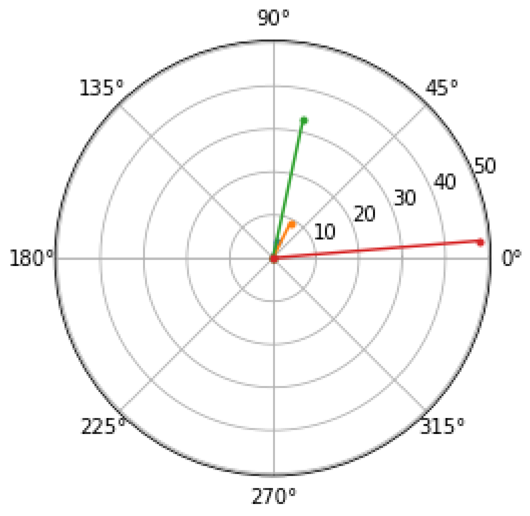
Out[12]:

```
[<matplotlib.lines.Line2D at 0x2a2b16495c0>]
```



In [27]:

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

lc = [[1,4],[4,8],[7,32],[48,4]]
for i in lc:
    i = complex(i[0],i[1])
    plt.polar([0,np.angle(i)],[0,np.abs(i)],marker=".")

plt.show()
```



Conclusion - The above graph shows a list of complex numbers plotted

02/12/2019

Write the following complex numbers in polar form:

(i) $1+i$
(ii) $5-5i$
(iii) $6i$
(iv) $-\sqrt{3}+i$
(v) $-2-\sqrt{3}i$

In [5]:

```
z1 = complex(1,1)
z2 = complex(5,-5)
z3 = complex(0,6)
z4 = complex(-np.sqrt(3),1)
z5 = complex(-2,-np.sqrt(3))
print(polar(z1))
print(polar(z2))
print(polar(z3))
print(polar(z4))
print(polar(z5))
```

```
(1.4142135623730951, 0.7853981633974483)
(7.0710678118654755, -0.7853981633974483)
(6.0, 1.5707963267948966)
(1.9999999999999998, 2.6179938779914944)
(2.6457513110645907, -2.4278682746450277)
```

Sketch the graph for:

$|z + i| = 2$
$real(x) = 5$
$imaginary(z) = -2$
$imaginary(\overline{z} + 3i) = 6$

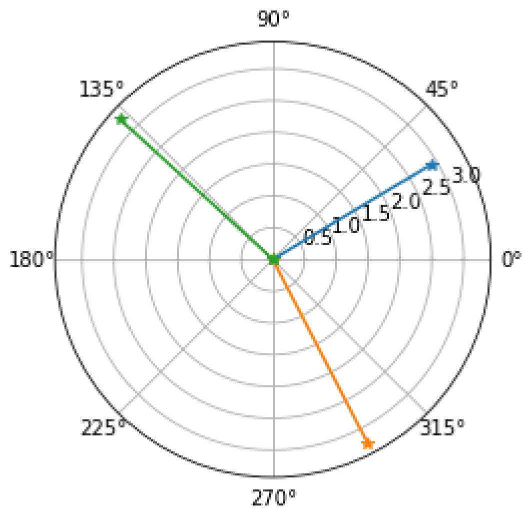
Find out which point is farthest away from the origin:

$z1 = 2.5 + 1.5i$
$z2 = 1.5 - 2.9i$
$z3 = -2.4 + 2.2i$

In [13]:

```
z = [[2.5,1.5],[1.5,-2.9],[-2.4,2.2]]
for i in z:
    i = complex(i[0],i[1])
    print("The distance from origin is: ",np.sqrt(np.real(i)**2+np.imag(i)**2))
    plt.polar([0,np.angle(i)],[0,np.abs(i)],marker="*")

plt.show()
```

```
The distance from origin is:  2.9154759474226504
The distance from origin is:  3.2649655434629015
The distance from origin is:  3.2557641192199416
```



If $a + ib = \frac{3-i}{2+3i} + \frac{2-2i}{1-5i}$. Find a and b

In [4]:

```
z1 = complex(3,-1)
z2 = complex(2,3)
z3 = complex(2,-2)
z4 = complex(1,-5)
print(z1/z2)
print(z3/z4)
z5 = (z1/z2)+(z3/z4)
print("a,b = ",z5)
```

```
(0.23076923076923078-0.8461538461538461j)
(0.4615384615384615+0.3076923076923077j)
a,b =  (0.6923076923076923-0.5384615384615384j)
```

Find:

$(1+i)^{1/3}$
$(1+i)^{1/5}$
$(3+4i)^{1/2}$
$(\frac{16i}{1+i}^{1/8}$
$(-i)^{1/4}$

$(-i)^{1/3}$
$(-1-\sqrt{3i})^{1/2}$
$(\frac{1+i}{\sqrt{3}i+i})^{1/6}$

In [14]:

```
print(complex(1,1)**(1/3))
```

```
(1.0842150814913512+0.2905145555072514j)
```

In [15]:

```
print(complex(1,1)**(1/5))
```

```
(1.0585781527063765+0.16766230825618095j)
```

In [16]:

```
print(complex(3,4)**(1/2))
```

```
(2+1j)
```

In [22]:

```
print(complex(0,16)/complex(1,1)**(1/8))
```

```
(1.5017845623084753+15.247874546598545j)
```

In [17]:

```
print(complex(0,-1)**(1/4))
```

```
(0.9238795325112867-0.3826834323650898j)
```

In [18]:

```
print(complex(0,-1)**(1/3))
```

```
(0.8660254037844387-0.49999999999999994j)
```

In [20]:

```
print(complex(-1,-np.sqrt(3)**(1/2)))
```

```
(-1-1.3160740129524924j)
```

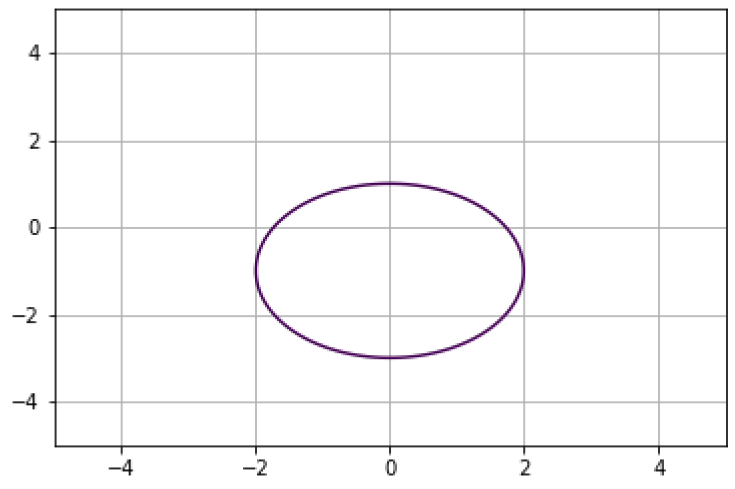
In [21]:

```
print(complex(1,1)/complex(np.sqrt(3),1)**(1/6))
```

```
(0.9651555190405962+0.8098616400556805j)
```

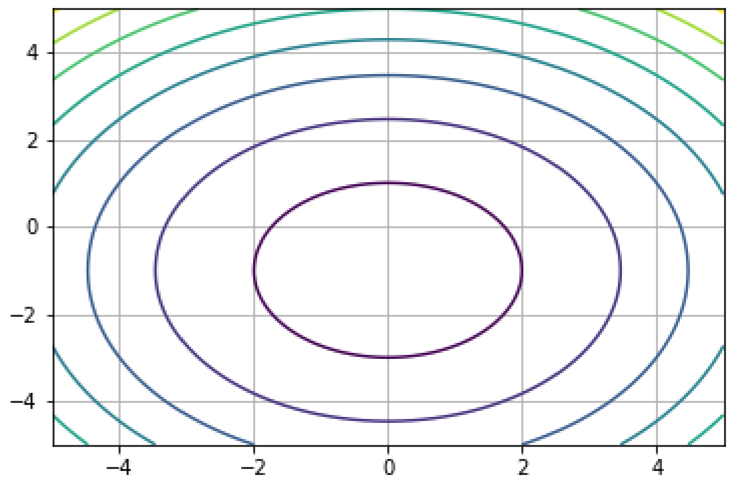
In [2]:

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

x = np.linspace(-5,5,100)
y = np.linspace(-5,5,100)
X,Y = np.meshgrid(x,y)
F = X**2 + (Y+1)**2 - 4
plt.contour(X,Y,F,[0])
plt.grid(True)
plt.show()
```



In [3]:

```
x = np.linspace(-5,5,100)
y = np.linspace(-5,5,100)
X,Y = np.meshgrid(x,y)
F = X**2 + (Y+1)**2 - 4
plt.contour(X,Y,F)
plt.grid(True)
plt.show()
```



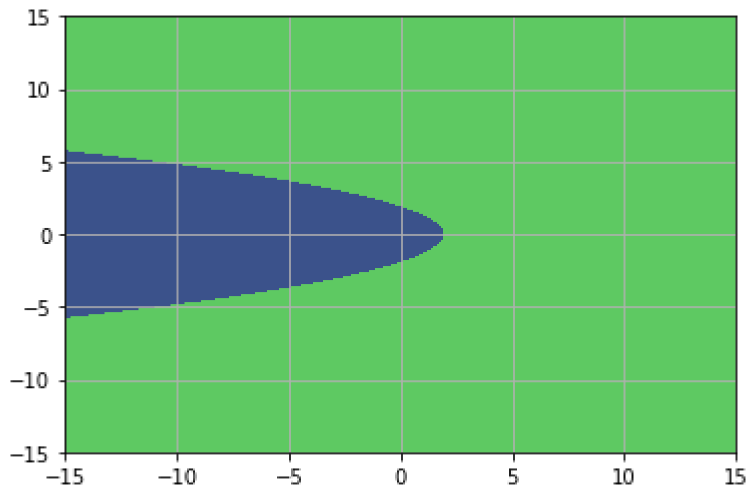
$$|2z + 3| < 1$$

$$|Z| \leq |2z + 1|$$

$$|2(x + y)| + 1$$

In [6]:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-15,15,100)
y = np.linspace(-15,15,100)
X,Y = np.meshgrid(x,y)
F = X*2 + (Y)**2 - 4
plt.contourf(X,Y,F,0)
plt.grid(True)
```



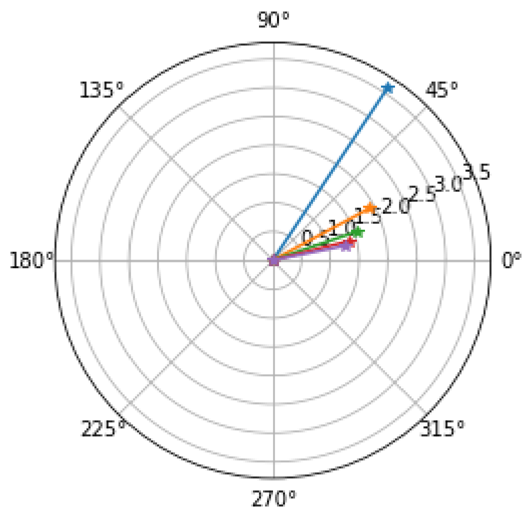
18/01/2020

Find the nth root of a complex number and plot it's root on the polar plane

In [8]:

```
import cmath
import math
from cmath import *
import numpy as np
import matplotlib.pyplot as plt

def comproot(a,b,n):
    z = complex(a,b)
    for i in range(1,n+1):
        r = z**(1/i)
        print("The ",i,"th root of the complex number is: ",r)
        plt.polar([0,np.angle(r)],[0,np.abs(r)],marker="*")

a = int(input("Enter the 1st parameter of a complex number: "))
b = int(input("Enter the 2nd parameter of a complex number: "))
n = int(input("Enter the nth root of a complex number: "))
comproot(a,b,n)
```

```
Enter the 1st parameter of a complex number: 2
Enter the 2nd parameter of a complex number: 3
Enter the nth root of a complex number: 5
The  1 th root of the complex number is:  (2+3j)
The  2 th root of the complex number is:  (1.6741492280355401+0.895977476129
838j)
The  3 th root of the complex number is:  (1.4518566183526649+0.493403534104
00467j)
The  4 th root of the complex number is:  (1.3365960777571289+0.335171369660
65714j)
The  5 th root of the complex number is:  (1.2675064916851109+0.252398387219
317j)
```



Find limit of real and complex function

LAB 4 ~ Find f(z) if u + v, u - v is given.

$$u + v = e^x(cosy + siny)$$

In [17]:

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

def fn():
    x,y,z = symbols('x y z')
    expr = exp(x)*(cos(y) + sin(y))
    fx = diff(expr,x)
    print("Differentiating function with respect to x: ",fx)
    fy = diff(expr,y)
    print("Differentiating function with respect to x: ",fy)
    fadd = fx+fy
    print("Adding the functions we get: ",fadd)
    fsub = fx-fy
    print("Subtracting the functions we get: ",fsub)
    fdiffz = fadd + fsub*1j
    print("The differentiated function, we get: ",fdiffz)
    fdiffz.replace(x,z)
    fdiffz.replace(y,0)
    print(fdiffz)

fn()
```

```
Differentiating function with respect to x:  (sin(y) + cos(y))*exp(x)
Differentiating function with respect to x:  (-sin(y) + cos(y))*exp(x)
Adding the functions we get:  (-sin(y) + cos(y))*exp(x) + (sin(y) + cos(y))*
exp(x)
Subtracting the functions we get:  -(-sin(y) + cos(y))*exp(x) + (sin(y) + co
s(y))*exp(x)
The differentiated function, we get:  1.0*I*(-(-sin(y) + cos(y))*exp(x) + (s
in(y) + cos(y))*exp(x)) + (-sin(y) + cos(y))*exp(x) + (sin(y) + cos(y))*exp
(x)
1.0*I*(-(-sin(y) + cos(y))*exp(x) + (sin(y) + cos(y))*exp(x)) + (-sin(y) + c
os(y))*exp(x) + (sin(y) + cos(y))*exp(x)
```

In [1]:

```
import cmath
import math
from cmath import *
import numpy as np
import matplotlib.pyplot as plt

def func(a,b):
    z = complex(a,b)
    f = z.real + z.imag*1j
    print("The function is: ",f)

a = int(input("Enter the 1st parameter of a complex number: "))
b = int(input("Enter the 2nd parameter of a complex number: "))
func(a,b)
```

```
Enter the 1st parameter of a complex number: 2
Enter the 2nd parameter of a complex number: 3
The function is:  (2+3j)
```

Can you give argument of a function as function? Justify.

Does complex function take arguments as symbols? Justify.

Yes, we can give argument of a function as a function

Yes, a complex function take arguments as symbols

LAB 5 ~ To check whether $f(z)$ is analytic or not.

In [10]:

```
import cmath
import sympy as sy

def analytic(u,v):
    print("Given expression f(z):",(u+1j*v))
    ux=sy.diff(u,x)
    print("\nDerivative of u wrt x:",ux)
    uy=sy.diff(u,y)
    print("Derivative of u wrt y:",uy)
    vx=sy.diff(v,x)
    print("Derivative of v wrt x:",vx)
    vy=sy.diff(v,y)
    print("Derivative of v wrt y:",vy)
    if(ux == vy and uy == -vx):
        print("\nf(z) is an analytic function.")
        return True
    else:
        print("\nf(z) is not an analytic function.")
        return False
```

In [11]:

```
from sympy import *

x,y = symbols('x y')
u = sy.exp(x)*sy.cos(y)
v = sy.exp(x)*sy.sin(y)
analytic(u,v)
```

```
Given expression f(z): 1.0*I*exp(x)*sin(y) + exp(x)*cos(y)

Derivative of u wrt x: exp(x)*cos(y)
Derivative of u wrt y: -exp(x)*sin(y)
Derivative of v wrt x: exp(x)*sin(y)
Derivative of v wrt y: exp(x)*cos(y)

f(z) is an analytic function.
```

Out[11]:

```
True
```

In [12]:

```
u=sy.tan(x)+1
v=x**2
analytic(u,v)
```

```
Given expression f(z): 1.0*I*x**2 + tan(x) + 1

Derivative of u wrt x: tan(x)**2 + 1
Derivative of u wrt y: 0
Derivative of v wrt x: 2*x
Derivative of v wrt y: 0

f(z) is not an analytic function.
```

Out[12]:

```
False
```

In [13]:

```
u=sy.sin(x)+sy.cos(x)
v=sy.sqrt(x)
analytic(u,v)
```

```
Given expression f(z): 1.0*I*sqrt(x) + sin(x) + cos(x)

Derivative of u wrt x: -sin(x) + cos(x)
Derivative of u wrt y: 0
Derivative of v wrt x: 1/(2*sqrt(x))
Derivative of v wrt y: 0

f(z) is not an analytic function.
```

Out[13]:

```
False
```

The objective of the above code was to check whether a given function is analytic or not

LAB 6 ~ To check whether a function is harmonic or not

In [14]:

```
x,y,z,c= sy.symbols('x y z c')
def harmonic(u,v):
    expr=u+1j*v
    print("Given expression f(z):",expr)
    dfx2 = expr.diff(x,x)
    print("\nSecond order partial derivative of f(z) wrt x: ",dfx2)
    dfy2 = expr.diff(y,y)
    print("Second order partial derivative of f(z) wrt y: ",dfy2)
    diffsum = dfx2 + dfy2
    if(diffsum == 0):
        print("\nf(z) is a harmonic function.")
        return True
    else:
        print("\nf(z) in not a harmonic function.")
        return False
```

In [15]:

```
u = (x**2)-(y**2)
v = 2*x*y
harmonic(u,v)
```

```
Given expression f(z): x**2 + 2.0*I*x*y - y**2

Second order partial derivative of f(z) wrt x:  2
Second order partial derivative of f(z) wrt y:  -2

f(z) is a harmonic function.
```

Out[15]:

```
True
```

In [16]:

```
u=sy.sin(x)+sy.cos(x)
v=x**2
harmonic(u,v)
```

```
Given expression f(z): 1.0*I*x**2 + sin(x) + cos(x)

Second order partial derivative of f(z) wrt x:  -sin(x) - cos(x) + 2.0*I
Second order partial derivative of f(z) wrt y:  0

f(z) in not a harmonic function.
```

Out[16]:

```
False
```

In [17]:

```
u=sy.tan(x)
v=x**3
harmonic(u,v)
```

```
Given expression f(z): 1.0*I*x**3 + tan(x)

Second order partial derivative of f(z) wrt x:  6.0*I*x + 2*(tan(x)**2 + 1)*
tan(x)
Second order partial derivative of f(z) wrt y:  0

f(z) in not a harmonic function.
```

Out[17]:

```
False
```

The code above was to determine whether a function is harmonic or not

LAB 7 ~ To check if $v(x, y)$ is a harmonic conjugate of $u(x, y)$

In [18]:

```
def harmonic(u):
    U1 = sy.diff(u, x, 2)
    U2 = sy.diff(u, y, 2)
    if (U1 + U2 == 0):
        return True
    else:
        return False

def harmonic_conj(U, V):
    print("\nGiven U(x,y) = ",U)
    print("Given V(x,y) = ",V, "\n")

    if (analytic(U,V) == True and harmonic(V) == True):
        print("\n", U, "is the harmonic conjugate of ", V)
    else:
        print("\nU(x,y) is not the harmonic conjugate of V(x,y)")
```

In [19]:

```
harmonic_conj(sy.sin(x),sy.tan(y))
```

```
Given U(x,y) =  sin(x)
Given V(x,y) =  tan(y)

Given expression f(z): sin(x) + 1.0*I*tan(y)

Derivative of u wrt x: cos(x)
Derivative of u wrt y: 0
Derivative of v wrt x: 0
Derivative of v wrt y: tan(y)**2 + 1

f(z) is not an analytic function.

U(x,y) is not the harmonic conjugate of V(x,y)
```

In [20]:

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

```
Given U(x,y) =  x**2 - y**2
Given V(x,y) =  2*x*y

Given expression f(z): x**2 + 2.0*I*x*y - y**2

Derivative of u wrt x: 2*x
Derivative of u wrt y: -2*y
Derivative of v wrt x: 2*y
Derivative of v wrt y: 2*x

f(z) is an analytic function.

 x**2 - y**2 is the harmonic conjugate of  2*x*y
```

In [21]:

```
harmonic_conj(sy.cos(x),sy.exp(x))
```

```
Given U(x,y) =  cos(x)
Given V(x,y) =  exp(x)

Given expression f(z): 1.0*I*exp(x) + cos(x)

Derivative of u wrt x: -sin(x)
Derivative of u wrt y: 0
Derivative of v wrt x: exp(x)
Derivative of v wrt y: 0

f(z) is not an analytic function.

U(x,y) is not the harmonic conjugate of V(x,y)
```

LAB 8 ~ BILINEAR TRANSFORMATIONS

In [33]:

```
from sympy import *
x,y,c = symbols('x y c')
u = x**2 - y**2
v = 2*x*y + c
print(u)
ux = diff(u,x)
print("ux: ",ux)
uy = diff(u,y)
print("uy: ",uy)
print("From C.R.E's: ")
print("ux = vy and uy = -vx")
vy = ux
vx = -uy
print("vy: ",vy)
print("vx: ",-vx)
print(integrate(vy,y) + c)
```

```
x**2 - y**2
ux:  2*x
uy:  -2*y
From C.R.E's:
ux = vy and uy = -vx
vy:  2*x
vx:  -2*y
c + 2*x*y
```

10/02/2020

Find the B.L.T that maps $1, i, -1$ onto $i, 0, -i$.

In [23]:

```
import sympy as sy
from sympy import *
from cmath import *
import cmath
import math
import matplotlib.pyplot as plt
```

In [21]:

```
def cross(z2,z3,z4,w2,w3,w4):
    z,w = sy.symbols('z w')
    z1 = z
    w1 = w
    cr = 0
    eqn1 = ((z1-z2)*(z3-z4))/((z2-z3)*(z4-z1))
    eqn2 = ((w1-w2)*(w3-w4))/((w2-w3)*(w4-w1))
    cr = sy.solve(eqn1,eqn2)
    print("The cross ratio is: ".format(cr))
```

How do I pass i as parameter?

In [22]:

```
cross(1,2,-1,3,5,8)
```

```
The cross ratio is:
```

In [ ]:

```
z2,z3,z4,w2,w3,w4 = symbols('z2 z3 z4 w2 w3 w4')
```

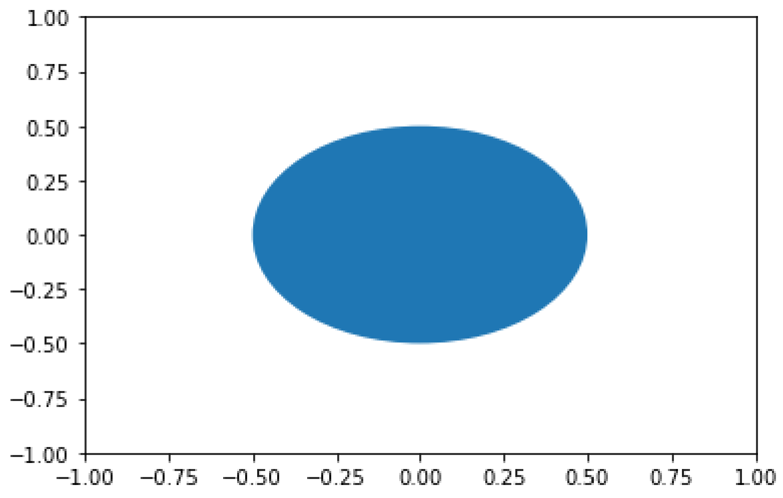
Find the image of $|Z| \leq 1$

In [24]:

```
fig, ax = plt.subplots()

ax.set(xlim=(-1, 1), ylim = (-1, 1))
a_circle = plt.Circle((0, 0), .5)
ax.add_artist(a_circle)
```

Out[24]:

```
<matplotlib.patches.Circle at 0x284be06d358>
```



Find out how 1 graph becomes another graph

Reflection

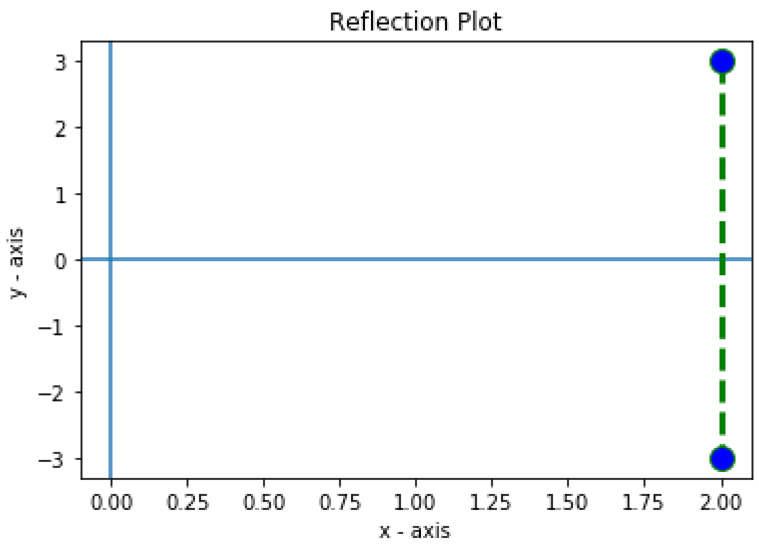
In [63]:

```
def ref(a,b):
    z = complex(a,b)
    zl = [a,a]
    print("The entered complex number is: ",z)
    w = z.conjugate()
    wl = [b,-b]
    print("The reflection of ",z," is: ",w)
    plt.axhline()
    plt.axvline()
    plt.plot(zl,wl,color='green', linestyle='dashed', linewidth = 3, marker='o', markerface
    plt.xlabel('x - axis')
    plt.ylabel('y - axis')
    plt.title('Reflection Plot')
    plt.show()
```

In [64]:

```
x = int(input("Enter the real part: "))
y = int(input("Enter the imaginary part: "))
ref(x,y)
```

```
Enter the real part: 2
Enter the imaginary part: 3
The entered complex number is:  (2+3j)
The reflection of  (2+3j)  is:  (2-3j)
```



Translation

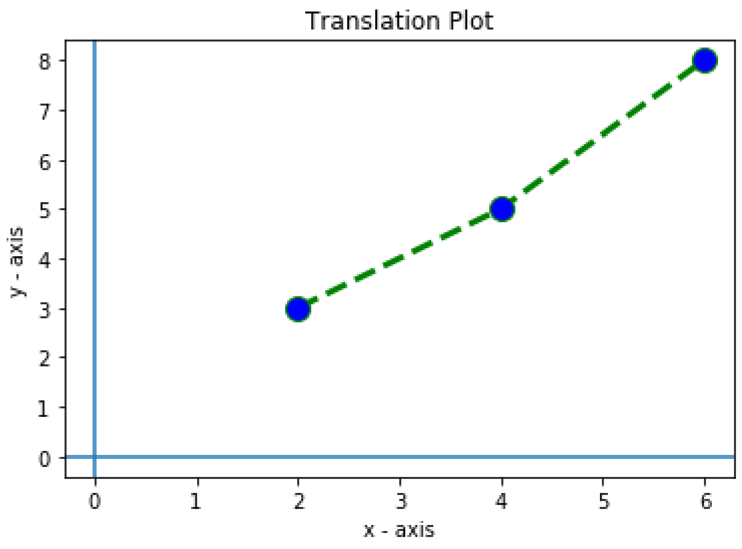
In [78]:

```
def trans(a,b,c,d):
    z = complex(a,b)
    print("The entered complex number is: ",z)
    c = complex(c,d)
    print("The entered complex constant is: ",c)
    w = z + c
    print("The translation of ",z," is: ",w)
    wr = w.real
    wi = w.imag
    zl = [a,c,wr]
    cl = [b,d,wi]
    plt.axhline()
    plt.axvline()
    plt.plot(zl,cl,color='green', linestyle='dashed', linewidth = 3, marker='o', markerface
    plt.xlabel('x - axis')
    plt.ylabel('y - axis')
    plt.title('Translation Plot')
    plt.show()
```

In [79]:

```
x = int(input("Enter the real part of complex number: "))
y = int(input("Enter the imaginary part of complex number: "))
c = int(input("Enter the real part of complex constant: "))
d = int(input("Enter the imaginary part of complex constant: "))
trans(x,y,c,d)
```

```
Enter the real part of complex number: 2
Enter the imaginary part of complex number: 3
Enter the real part of complex constant: 4
Enter the imaginary part of complex constant: 5
The entered complex number is:  (2+3j)
The entered complex constant is:  (4+5j)
The translation of  (2+3j)  is:  (6+8j)

C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\core\_asarray.py:85: Compl
exWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



In [13]:

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

In [26]:

```
z = Symbol('z')
def bil(w):
    eqn = w - z
    s = solve(eqn,z)
    print(s)
```

In [3]:

```
bil((z-1)/(z+1))
```

```
[-I, I]
```

In [4]:

```
bil((1-z)/(1+z))
```

```
[-1 + sqrt(2), -sqrt(2) - 1]
```

In [5]:

```
bil((2*z-1)/z)
```

```
[1]
```

In [6]:

```
bil((z-(1+I))/(z+2))
```

```
[-1/2 - sqrt(-3 - 4*I)/2, -1/2 + sqrt(-3 - 4*I)/2]
```

In [ ]:

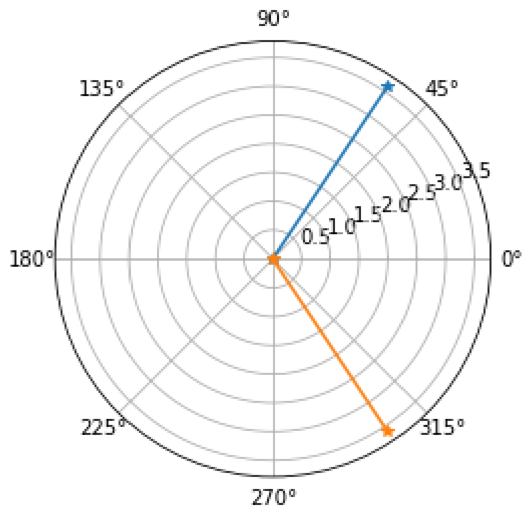
```

```

In [16]:

```
w = complex(2,3)
c = complex(conjugate(w))
plt.polar([0,np.angle(w)],[0,np.abs(w)],marker='*')
plt.polar([0,np.angle(c)],[0,np.abs(c)],marker='*')
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x194eaf5c748>]
```



In [1]:

```
import sympy as sp
from sympy import *
import numpy as np
def bilinear(d,r):
    w,z = symbols('w z')
    LHS = ((z-d[0])/(d[0]-d[1]))*((d[1]-d[2])/(d[2]-z))
    RHS = ((w-r[0])/(r[0]-r[1]))*((r[1]-r[2])/(r[2]-w))
    k1 = sp.Eq(LHS,RHS)
    k2 = sp.solve(k1,w)
    print(k2)
```

In [13]:

```
bilinear([0,-1j,-1],[1j,1,0])
```

```
[-I*(z + 1.0)/(z - 1.0)]
```

In [5]:

```
bilinear([1,1j,-1],[1j,0,-1j])
```

```
[-(I*z + 1.0)/(I*z - 1.0)]
```

In [23]:

```
bilinear([0,1j,1/0],[1,-1j,-1])
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-23-cbe3389e9257> in <module>()
----> 1 bilinear([0,1j,1/0],[1,-1j,-1])

ZeroDivisionError: division by zero
```

In [24]:

```
bilinear([0,1j,-1],[0,np.infty,-1])
```

```
False
```

In [11]:

```
bilinear([1,1j,-1],[1j,0,-1j])
```

```
[-(I*z + 1.0)/(I*z - 1.0)]
```

1. Plot the reflection of the points $(3,2)$ and $(5,1)$ with respect to both the X-axis and Y-axis in the same plane.

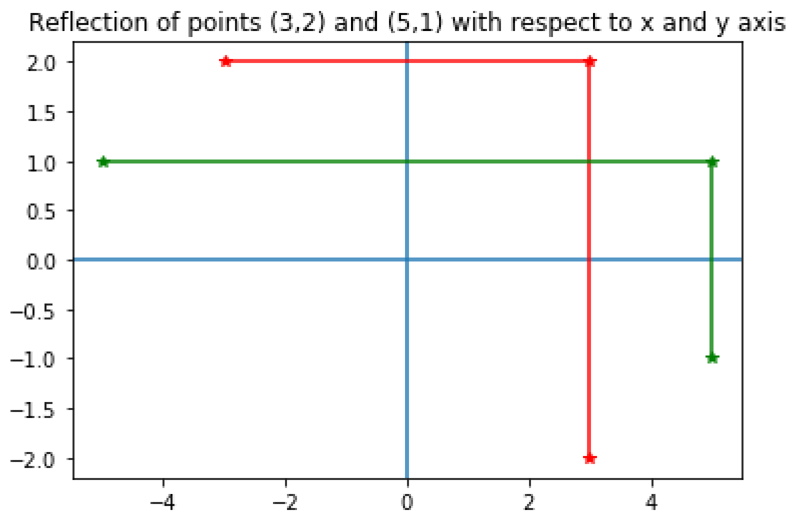
In [1]:

```
def ref(a,b,c,d):
    import matplotlib.pyplot as plt
    import cmath
    import sympy as sp
    z1 = complex(a,b)
    z2 = complex(c,d)
    z1xx = [-a,a,a]
    z1yx = [b,b,-b]
    z2xx = [-c,c,c]
    z2yx = [d,d,-d]
    z1c = sp.conjugate(z1)
    z2c = sp.conjugate(z2)
    print("The conjugate of the 1st complex number is: ",z1c)
    print("The conjugate of the 2nd complex number is: ",z2c)
    plt.axhline()
    plt.axvline()
    plt.plot(z1xx,z1yx,marker="*",color="red")
    plt.plot(z2xx,z2yx,marker="*",color="green")
    plt.title("Reflection of points (3,2) and (5,1) with respect to x and y axis")
    #plt.legend("X-Axis Y-Axis (3,2) (5,1)")
```

In [3]:

```
a = int(input("Enter the x - coordinate of the 1st point: "))
b = int(input("Enter the y - coordinate of the 1st point: "))
c = int(input("Enter the x - coordinate of the 2nd point: "))
d = int(input("Enter the y - coordinate of the 2nd point: "))
ref(a,b,c,d)
```

```
Enter the x - coordinate of the 1st point: 3
Enter the y - coordinate of the 1st point: 2
Enter the x - coordinate of the 2nd point: 5
Enter the y - coordinate of the 2nd point: 1
The conjugate of the 1st complex number is:  3.0 - 2.0*I
The conjugate of the 2nd complex number is:  5.0 - 1.0*I
```



2. Find the Bilinear Transformation which maps $z = 1, i, -1$ onto $w = 1, 0, -1$ respectively.

In [39]:

```
import sympy as sp


def bil(a,r):
    z,w = sp.symbols('z w')
    LHS = ((z-a[0])/(a[0]-a[1]))*((a[1]-a[2])/(a[2]-z))
    RHS = ((w-r[0])/(r[0]-r[1]))*((r[1]-r[2])/(r[2]-w))
    k1 = sp.Eq(LHS,RHS)
    k2 = sp.solve(k1,w)
    sp.pprint(k2)
```

In [41]:

```
a = [0,-1,-1j]
b = [0,1,1j]
bil(a,b)
```

```
[]
```

3. Plot the translation of the point $u = 3 + 2i$ using the complex constant $c = 2 + 3i$ in the same polar plane.

In [7]:

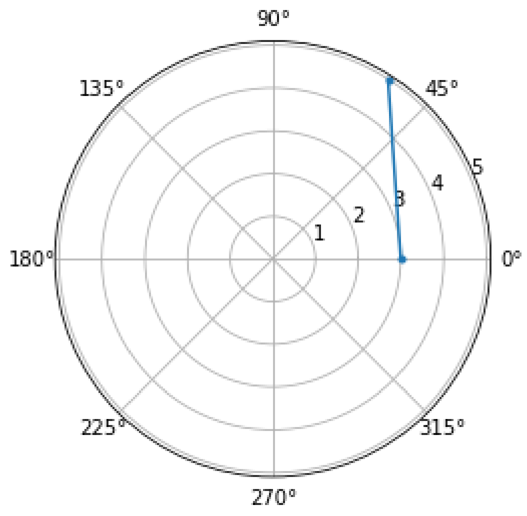
```
def trans(r,i,rc,ic):
    import cmath
    import matplotlib.pyplot as plt
    z = complex(r,i)
    zc = complex(rc,ic)
    print("The entered complex number is: ",z)
    print("The entered complex constant is: ",zc)
    tr = z + zc
    print("The translation is: ",tr)
    plt.polar([z,tr],marker=".")
    plt.show()
```

In [8]:

```
a = int(input("Enter the real part of the complex number: "))
b = int(input("Enter the imaginary part of the complex number: "))
c = int(input("Enter the real part of the complex constant: "))
d = int(input("Enter the imaginary part of the complex constant: "))
trans(a,b,c,d)
```

```
Enter the real part of the complex number: 3
Enter the imaginary part of the complex number: 2
Enter the real part of the complex constant: 2
Enter the imaginary part of the complex constant: 3
The entered complex number is:  (3+2j)
The entered complex constant is:  (2+3j)
The translation is:  (5+5j)
```

```
C:\Users\Jeevan\Anaconda3\lib\site-packages\numpy\core\_asarray.py:85: Compl
exWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



Check whether the following are conformal. If yes, find its real and imaginary parts

$(i) e^z$

In [6]:

```
import sympy as sp
import cmath as cm
import numpy as np
z = sp.Symbol('z')
q = sp.Symbol('q')
q = exp(z)
d = diff(q,z)
print("Derivative: ",d)
if(d!=0):
    print("Conformal")
else:
    print("Not conformal")
```

```
Derivative:  exp(z)
Conformal
```

z =

$(ii) z^2$

In [7]:

```
z = sp.Symbol('z')
q = sp.Symbol('q')
q = z**2
d = diff(q,z)
print("Derivative: ",d)
if(d!=0):
    print("Conformal")
else:
    print("Not conformal")
```

```
Derivative:  2*z
Conformal
```

$(iii) sin(z)$

In [8]:

```
z = sp.Symbol('z')
q = sp.Symbol('q')
q = sin(z)
d = diff(q,z)
print("Derivative: ",d)
if(d!=0):
    print("Conformal")
else:
    print("Not conformal")
```

```
Derivative:  cos(z)
Conformal
```

In [54]:

```
def bil(a,r):
    z,w = sp.symbols('z w')
    LHS = ((z-a[0])/(a[0]-a[1]))*((a[1]-a[2])/(a[2]-z))
    RHS = ((w-r[0])/(r[0]-r[1]))*((r[1]-r[2])/(r[2]-w))
    eq=sp.simplify(LHS-RHS)
    k1 = sp.Eq(eq,0)
    k2 = sp.solve(k1,w)
    sp.pprint(k2)
```

In [55]:

```
a = [0,-1,-1j]
b = [0,1,1j]
bil(a,b)
```

```
[-z]
```

The above codes have been used to find out whether functions are conformal or

not based on their derivative. Elementary transformations such as reflection and translation are plotted on both xy and complex plane. The method as to how to find different bi-linear transformations based on given inputs have also been coded.