LAB 2: CMATH

Date: 04-1-2024

In [1]:
```
from cmath import *
import math
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
import ipywidgets as widgets
import random
```

In [2]:
```
a = 2 + 4j
a
```

Out[2]:
```
(2+4j)
```

> Absolute of Complex Number

In [3]:
```
abs(a)
```

Out[3]:
```
4.47213595499958
```

In [4]:
```
(a.real**2 + a.imag**2)**0.5
```

Out[4]:
```
4.47213595499958
```

> Phase of Complex Number

In [5]:
```
polar(a)[-1]
```

Out[5]:
```
1.1071487177940904
```

In [6]:
```
atan(a.imag/a.real)
```

Out[6]:
```
(1.1071487177940904+0j)
```

> Polar Form of Complex Number

In [7]:
```
polar(a)
```

Out[7]:
```
(4.47213595499958, 1.1071487177940904)
```

In [8]:
```
print(polar(a)[0],"e^",polar(a)[1])
```

```
4.47213595499958 e^ 1.1071487177940904
```

> WAP to print constants in cmath

In [9]:
```python
consts = zip(["e","inf","infj","nan","nanj","pi","tau"],[e,inf,infj,nan,nanj,pi,tau
dict(consts)
```

Out[9]:
```
{'e': 2.718281828459045,
 'inf': inf,
 'infj': infj,
 'nan': nan,
 'nanj': nanj,
 'pi': 3.141592653589793,
 'tau': 6.283185307179586}
```

> WAP to verify the following properties of complex numbers.
>
> - Commutative Addition
> - Commutative Multiplication
> - Associative Addition
> - Associative Multiplication
> - Distributive

In [10]:
```python
def commutativeAdd(a,b,c):
    if(a+b == b+a):
        return True
    return False
def commutativeMult(a,b,c):
    if(a*b == b*a):
        return True
    return False
def associativeAdd(a,b,c):
    if((a+b)+c == a+(b+c)):
        return True
    return False
def associativeMult(a,b,c):
    if((a*b)*c == a*(b*c)):
        return True
    return False
def distributive(a,b,c):
    if(a*(b+c) == a*b + a*c):
        return True
    return False
```

In [11]:
```python
a,b,c = [eval(input("Enter Number: ")) for i in range(3)]
```

In [12]:
```python
print("Satisfied Commutative Law of Addition" if (commutativeAdd(a,b,c)) else "Do N
print("Satisfied Commutative Law of Multiplication" if (commutativeMult(a,b,c)) els
print("Satisfied Associative Law of Addition" if (associativeAdd(a,b,c)) else "Do N
print("Satisfied Associative Law of Multiplication" if (associativeMult(a,b,c)) els
print("Satisfied Distributive Law" if (distributive(a,b,c)) else "Do Not Satisfy Di
```

```
Satisfied Commutative Law of Addition
Satisfied Commutative Law of Multiplication
Satisfied Associative Law of Addition
Satisfied Associative Law of Multiplication
Satisfied Distributive Law
```

In [13]:
```python
funcs = [commutativeAdd,commutativeMult,associativeAdd,associativeMult,distributive
for i in funcs:
    print("Verified" if i(a,b,c) else "Non Verified" ,"Law of", i.__name__)
```

```
Verified Law of commutativeAdd
Verified Law of commutativeMult
Verified Law of associativeAdd
Verified Law of associativeMult
Verified Law of distributive
```

> WAP to verify that the sum of two conjugate numbers is real.

In [14]:
```
a = eval(input("Enter Number: "))
a_conj = a.conjugate()
conjSum = a+a_conj
print(conjSum)
"Verified" if conjSum.imag == 0 else "Not Verified"
```

```
(4+0j)
```

Out[14]:
```
'Verified'
```

> WAP to verify that the product of conjugate numbers is real.

In [15]:
```
a = eval(input("Enter Number: "))
a_conj = a.conjugate()
conjProd = a*a_conj
print(conjProd)
"Verified" if conjProd.imag == 0 else "Not Verified"
```

```
(13+0j)
```

Out[15]:
```
'Verified'
```

> WAP to prove that $|z_1 + z_1| \leq |z_1| + |z_1|$.

In [16]:
```
a,b = [eval(input("Enter Number: ")) for i in range(2)]
True if(abs(a+b) <= abs(a)+abs(b)) else False
```

Out[16]:
```
True
```

> WAP to verify that, if the sum and product of two complex numbers is real, then they are conjugates of each other.

In [17]:
```
a,b = [eval(input("Enter Number: ")) for i in range(2)]
```

In [18]:
```
if((a*b).imag == 0 and (a+b).imag == 0):
    print("Numbers are Conjugate")
else:
    print("Numbers are Not Conjugate")
print(a.conjugate() == b)
```

```
Numbers are Not Conjugate
False
```

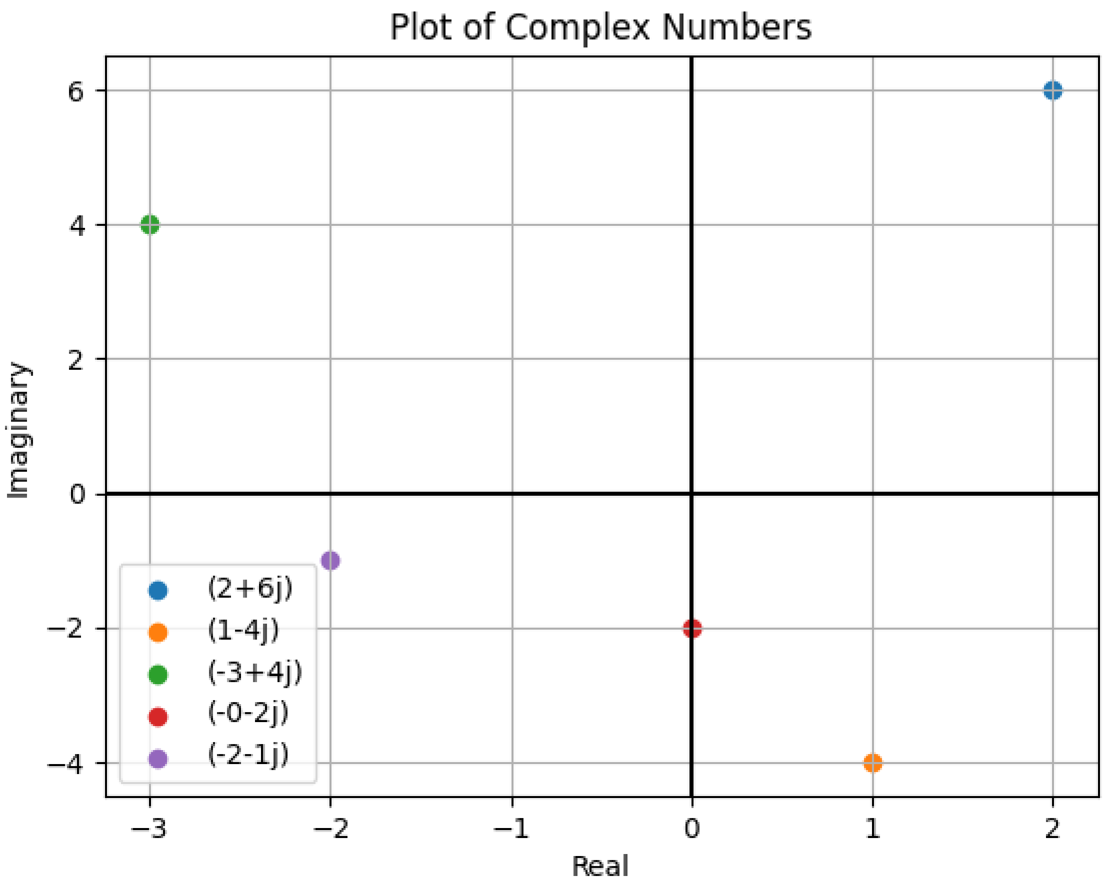
> WAP to verify the Eulers formula
> $$e^{i\theta} = \cos\theta + i\sin\theta$$

In [20]:
```python
a = eval(input("Enter complex number(arg,amp)$"))
#print(a[1]*exp(a[0]*1j),a[1]*(cos(a[0]) + 1j*sin(a[0])))
True if(a[1]*exp(a[0]*1j) == a[1]*(cos(a[0]) + 1j*sin(a[0]))) else False
```

Out[20]:
```
True
```

> WAP to plot the given set of complex numbers.

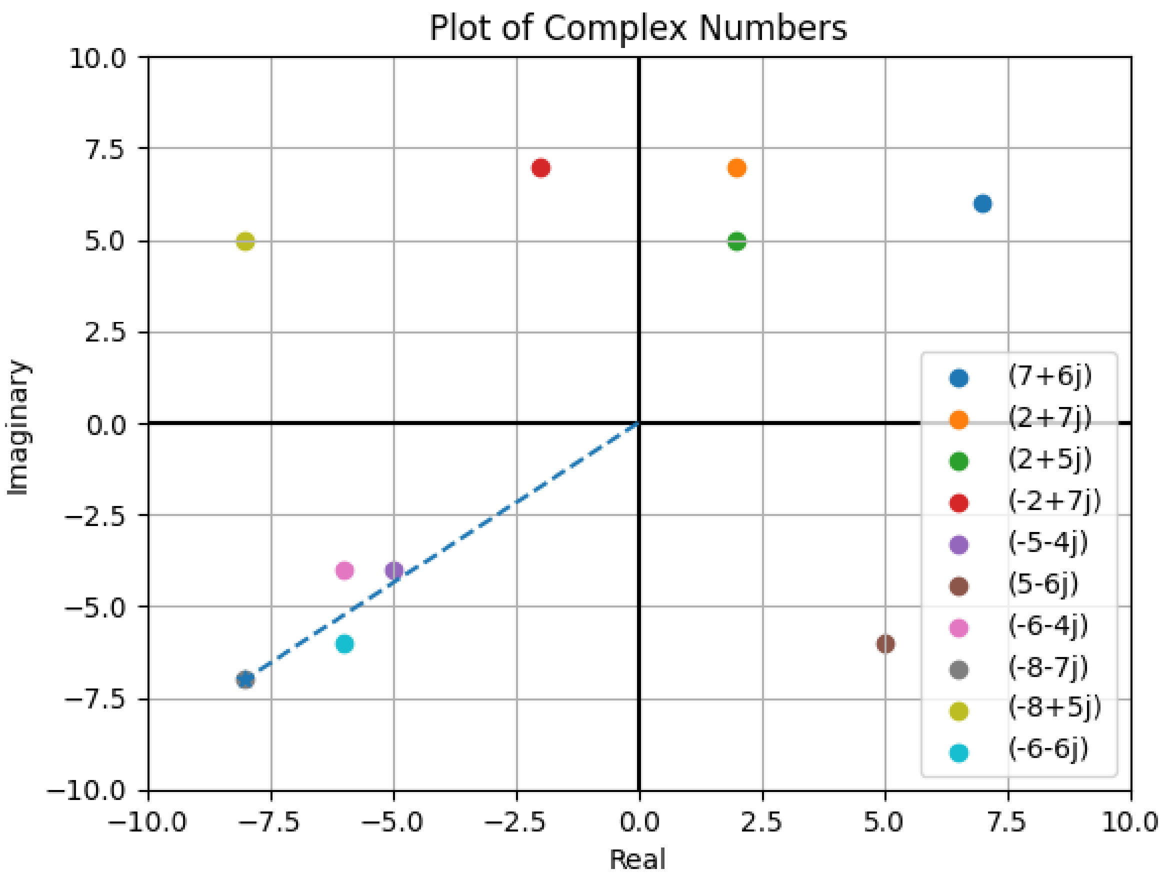
In [2]:
```python
z = eval(input("Enter list of complex numbers:"))
def plotComplex(z,show = True,legend = True):
    f = plt.figure()
    for i in z:
        plt.scatter(i.real,i.imag)
    if(legend):
        plt.legend(z)
    plt.grid()
    plt.axhline(color = "black")
    plt.axvline(color = "black")
    plt.xlabel("Real")
    plt.ylabel("Imaginary")
    plt.title("Plot of Complex Numbers")
    if show:
        plt.show()
    return f
#[2+6j, 1-4j,-3+4j, -2j, -2-1j]
plotComplex(z);
```



> WAP to find the furthest point from the origin

In [3]:
```
#z = eval(input("Enter List of complex numbers:"));
z = [complex(random.randint(-8,8),random.randint(-8,8)) for i in range(10)]
maxz = list(map(lambda z: abs(z),z))
maxz = maxz.index(max(maxz))
maxz = z[maxz]
print("Maximum Distance= ",maxz)
f = plotComplex(z,False)
plt.xlim(-10,10)
plt.ylim(-10,10)
plt.figure(f)
plt.scatter(maxz.real,maxz.imag,marker = "*",label = "Maximum")
plt.plot([0,maxz.real],[0,maxz.imag],linestyle = "--")
#plt.legend()
plt.show()
```

```
Maximum Distance=  (-8-7j)
```



In [34]:
```
z = (3 - 1j)/(2+3j) - (2-2j)/(1-5j)
z
```

Out[34]:
```
(-0.23076923076923073-1.1538461538461537j)
```

> WAP to to plot n^{th} roots of unity

In [5]:
```
def unityRoots(n = 1):
    return np.roots([1]+ [0]*(n-1) + [1])
    #x = sp.Symbol("x")
    #return sp.solve(sp.Eq(x**n,1),x)
unityRoots(4)
```

Out[5]:
```
array([-0.70710678+0.70710678j, -0.70710678-0.70710678j,
        0.70710678+0.70710678j,  0.70710678-0.70710678j])
```

In [6]:
```
def plotRoots(n):
    roots = unityRoots(n)
    f = plt.figure()
    plt.axhline(color = "black")
    plt.axvline(color = "black")
    plt.grid(which="both")
    plt.xlim(-1.2,1.2)
    plt.ylim(-1.2,1.2)
    for i in roots:
        i = complex(i)
        #pll.polar(1,phase(i))
        plt.scatter(i.real,i.imag)
    plt.title(f"n = {n}")
    plt.show()
widgets.interactive(plotRoots,n = (1,10))
```

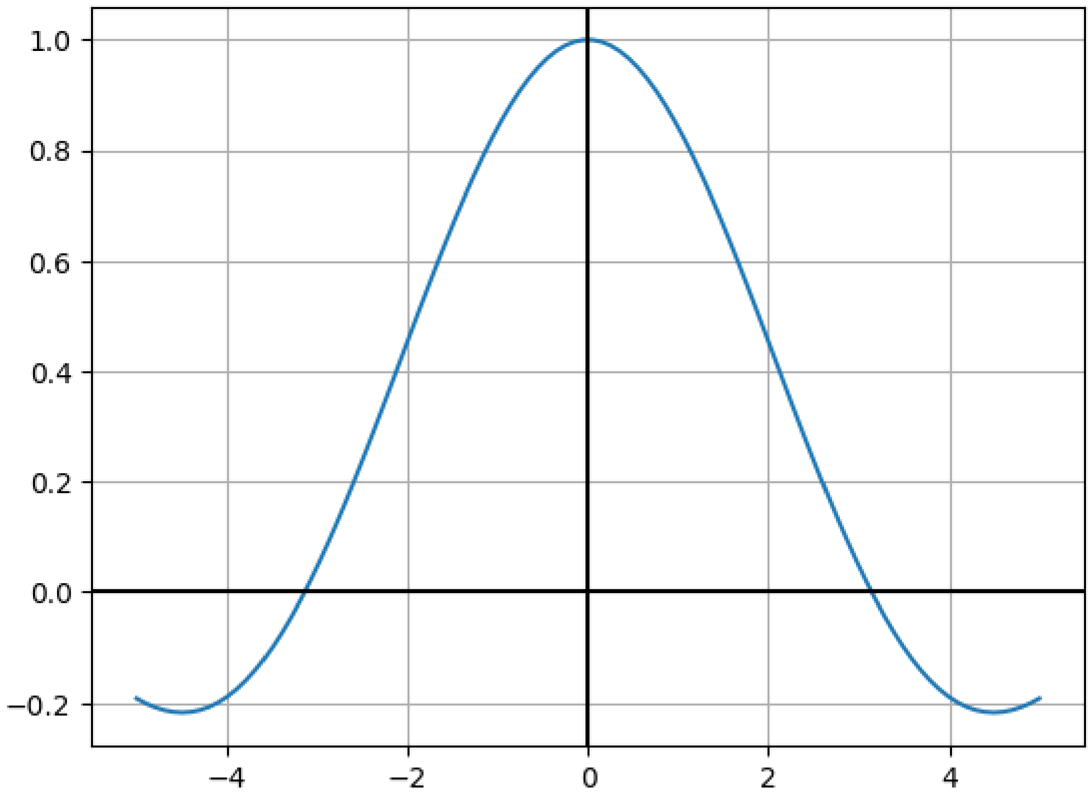
Out[6]:
```
interactive(children=(IntSlider(value=5, description='n', max=10, min=1), Output
()), _dom_classes=('widget-int…
```

Date: 17-01-24

LIMITS OF COMPLEX FUNCTION

In [7]:
```
def plotF(f:"Callable",x_lim):
    X = np.linspace(-x_lim,x_lim,100)
    plt.plot(X,[f(i) for i in X])
    plt.grid()
    plt.axhline(color = "black")
    plt.axvline(color = "black")
    plt.show()
```

In [8]:
```
def f(x):
    return np.sin(x)/x
plotF(f,5)
```



In [9]:
```python
def getLimit(f:"sympy.Function",z = sp.Symbol("z"), z0 = 0):
    lim = sp.limit(f,z,z0)
    return complex(lim.simplify())
```

In [10]:
```python
getLimit("sin(z)/z",z0 = 1+1j)
```

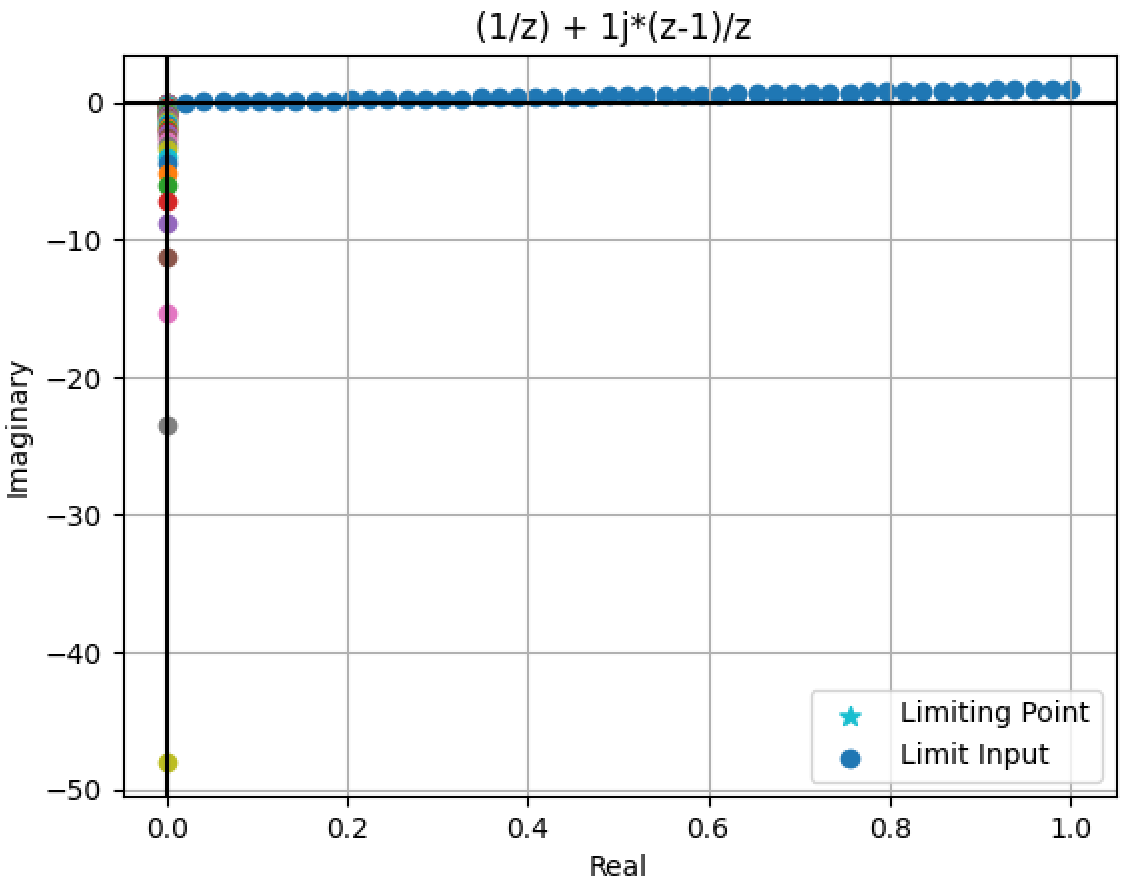
Out[10]:
```
(0.9667107481003567-0.3317468333156206j)
```

Date: 18-01-24

LIMIT OF A COMPLEX SEQUENCE

In [11]:
```python
def limitingPoints(f:"sympy.Function",z = sp.Symbol("z"), zStart = 1,zLimit = 0):
    func = sp.lambdify(z,f,"numpy")
    points = np.linspace(zStart,zLimit,50)[:-1]
    fig = plotComplex([func(i) for i in points],show=False,legend=False)
    limit = getLimit(f,z0 = zLimit)
    plt.scatter(limit.real,limit.imag,marker = '*',s = 50,label ="Limiting Point")
    plt.scatter([i.real for i in points],[i.imag for i in points],label = "Limit Ir
    plt.legend()
    plt.title(f"{f}")
    plt.show()
    return
```

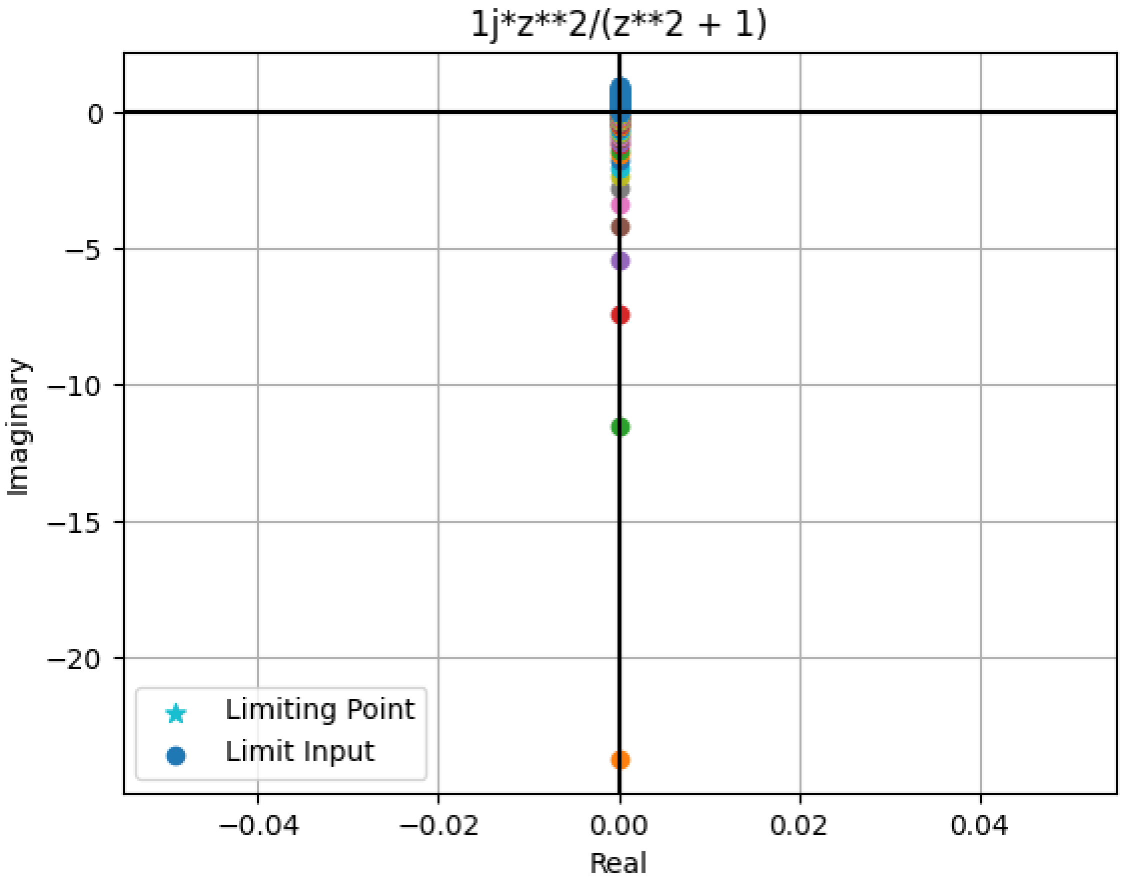
In [12]:
```python
limitingPoints("(1/z) + 1j*(z-1)/z ",zStart=complex(1,1),zLimit=0)
```



> $$i\frac{n^2}{n^2+1}$$

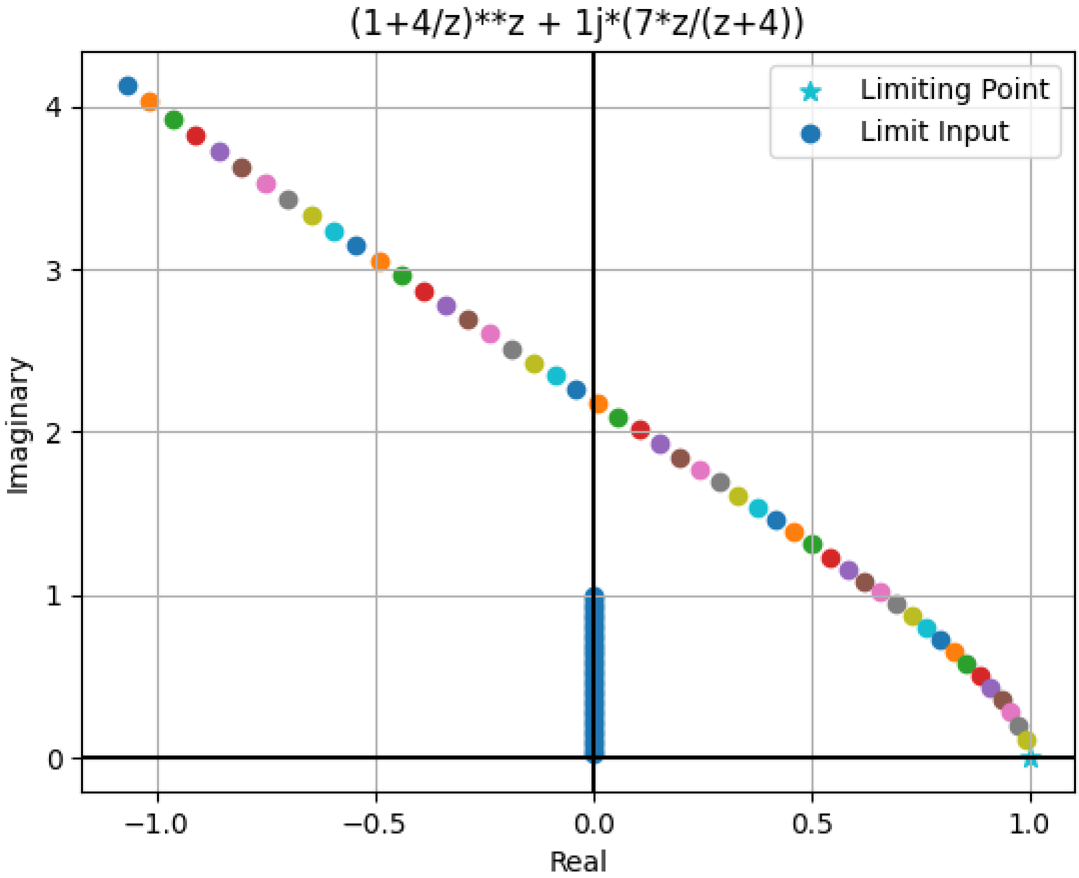
In [16]:
```python
limitingPoints("1j*z**2/(z**2 + 1)",zStart=1j,zLimit=0)
```

```
&lt;lambdifygenerated-4&gt;:2: RuntimeWarning: divide by zero encountered in cdouble_sca
lars
  return 1j*z**2/(z**2 + 1)
&lt;lambdifygenerated-4&gt;:2: RuntimeWarning: invalid value encountered in cdouble_scal
ars
  return 1j*z**2/(z**2 + 1)
```



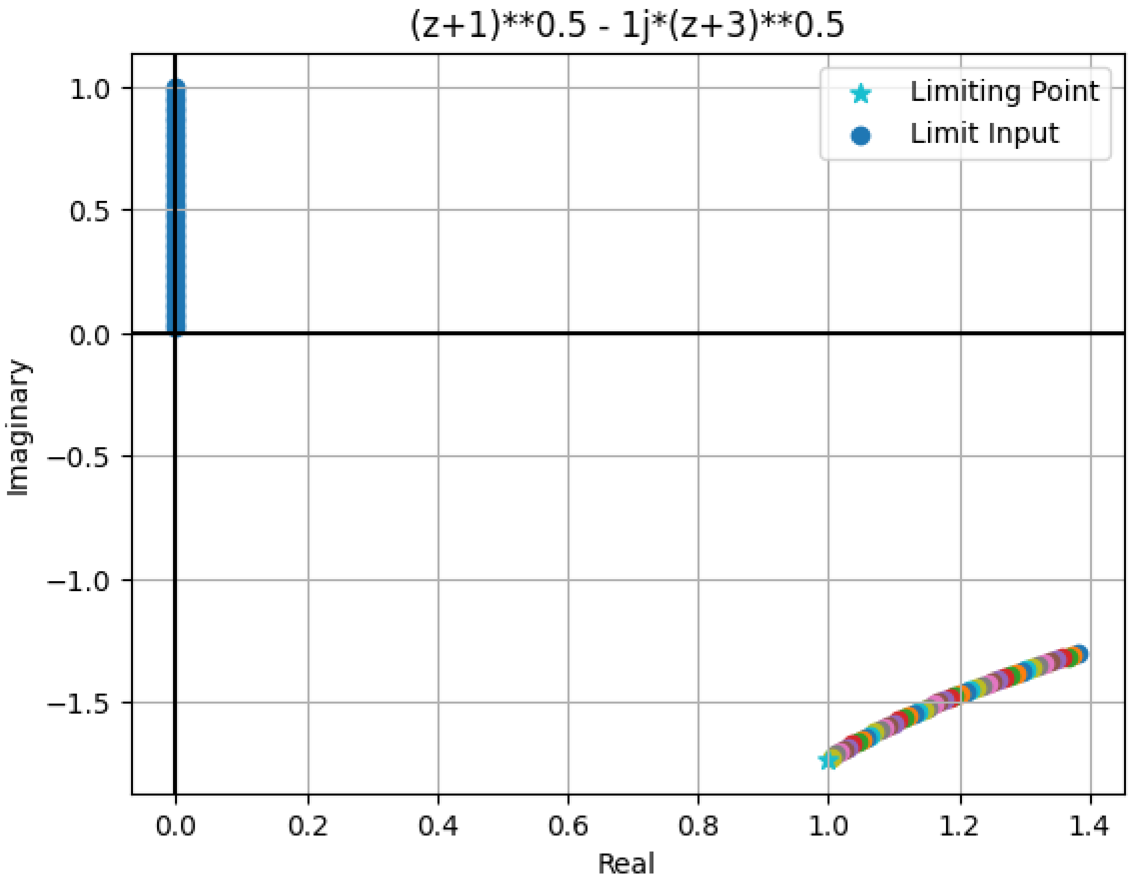
> $(1+\frac{4}{z})^z + i\frac{7z}{z+4}$

In [17]:
```
limitingPoints("(1+4/z)**z + 1j*(7*z/(z+4))",zStart=1j,zLimit=0)
```



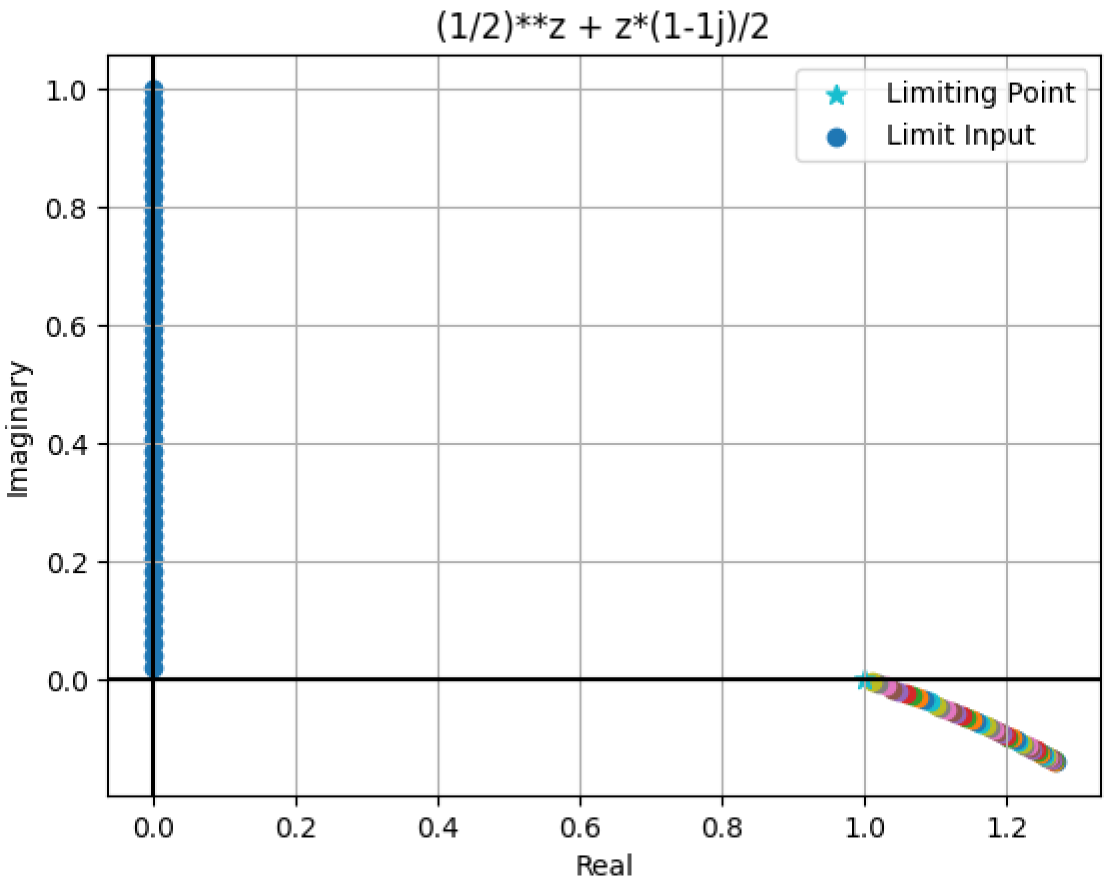
> $\sqrt(z+1) - i\sqrt(z+3)$

In [19]:
```
limitingPoints("(z+1)**0.5 - 1j*(z+3)**0.5",zStart=1j,zLimit=0)
```



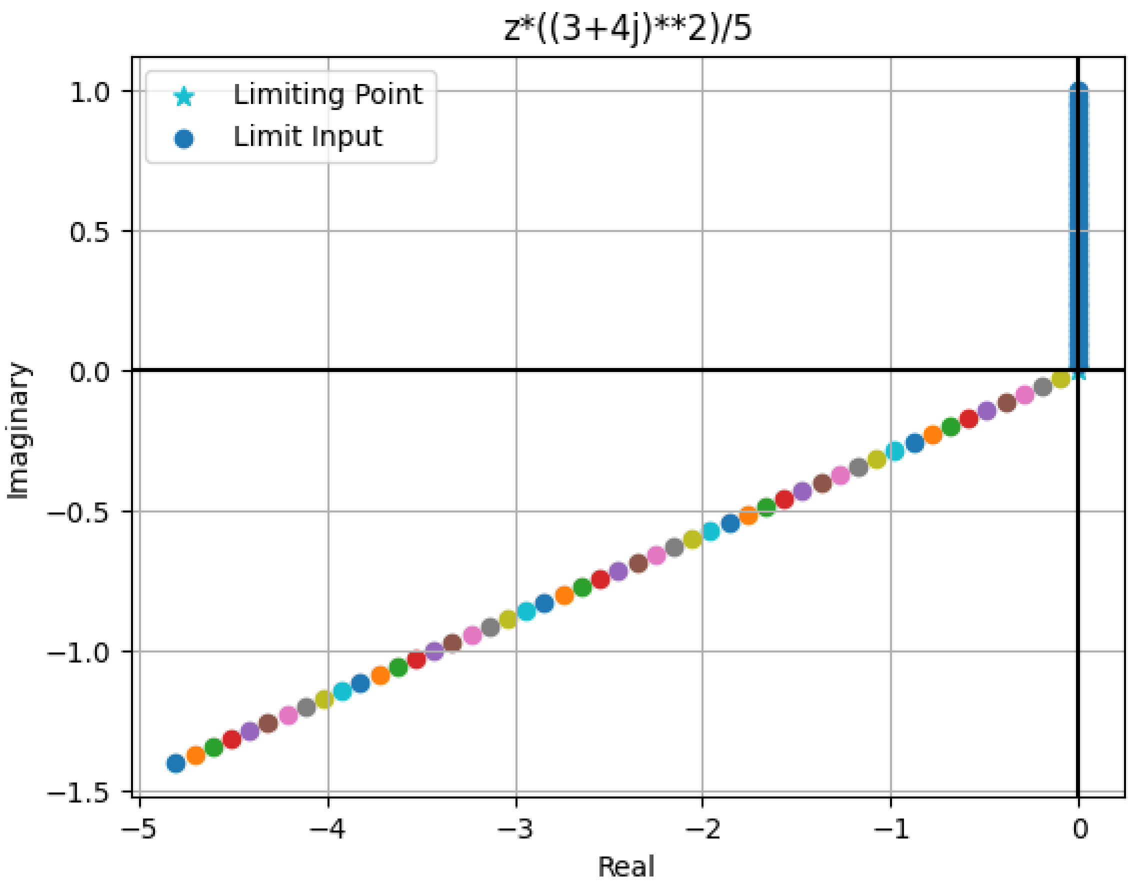
> $$\frac{1}{2}^z + z\frac{1-i}{2}$$

In [20]:
```
limitingPoints("(1/2)**z + z*(1-1j)/2",zStart=1j,zLimit=0)
```



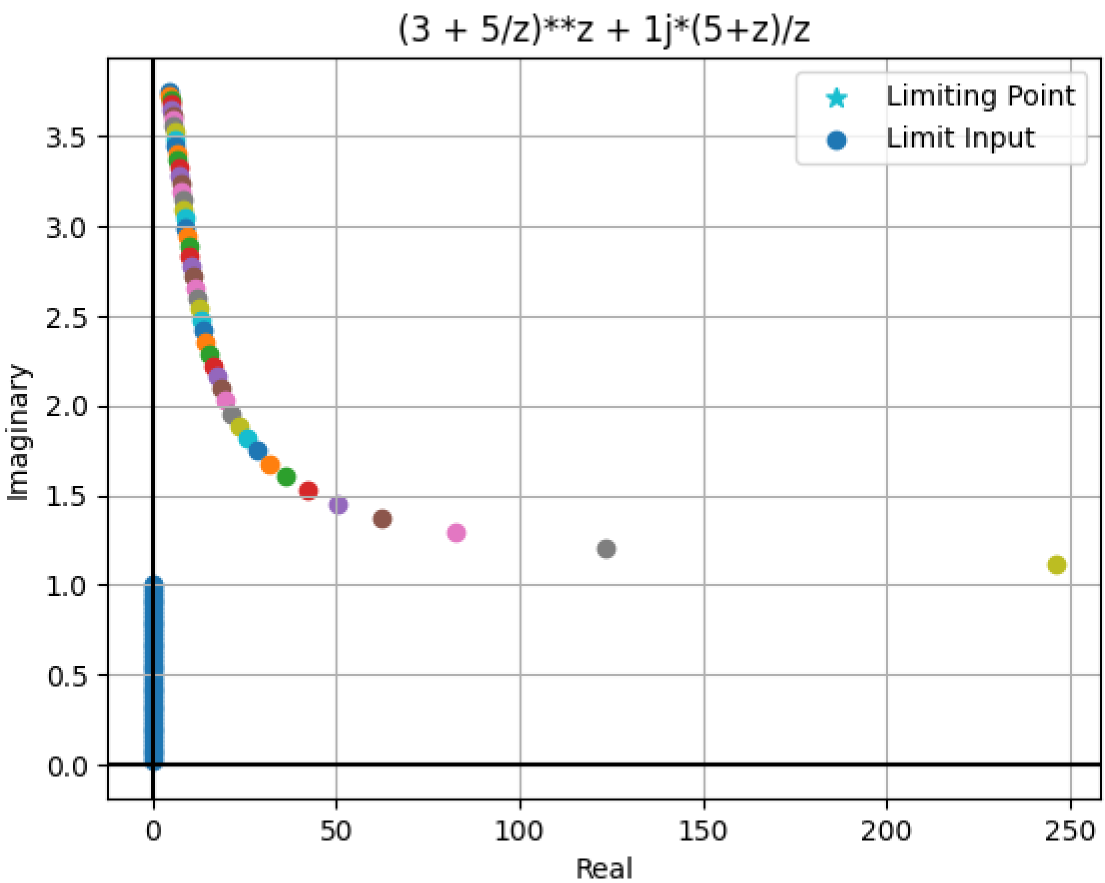
> $$z\frac{(3+4i)^2}{5}$$

In [21]:
```
limitingPoints("z*((3+4j)**2)/5",zStart=1j,zLimit=0)
```



> $$(3 + \frac{5}{z})^z + i\frac{5+z}{z}$$

In [22]:
```
limitingPoints("(3 + 5/z)**z + 1j*(5+z)/z",zStart=1j,zLimit=0)
```



```
In [ ]:
```