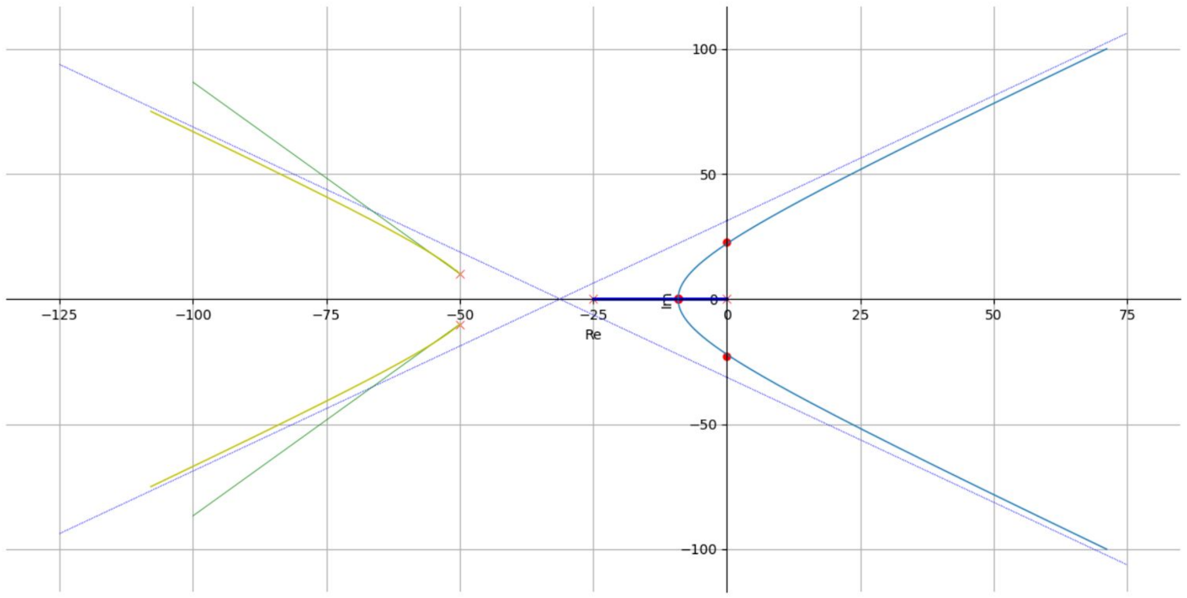


Root Locus

Program to draw the root locus following the rules.

Name: Linh Ahmed Mohamed Salah Eldin

ID: 50

Source code:

https://github.com/linhahmed/Root-Locus

Requirements

Given the following open loop transfer function with four poles at

S = 0 , S= -25, S= -50 + j 10 and S = -50 – j 10 and no zeroes. It is required to write a program to draw the root locus following the rules.

Description

The root locus of a feedback system is the graphical representation in the complex s-plane of the possible locations of its closed-loop poles for varying values of a certain system parameter. The points that are part of the root locus satisfy the angle condition.

Data Structure Used

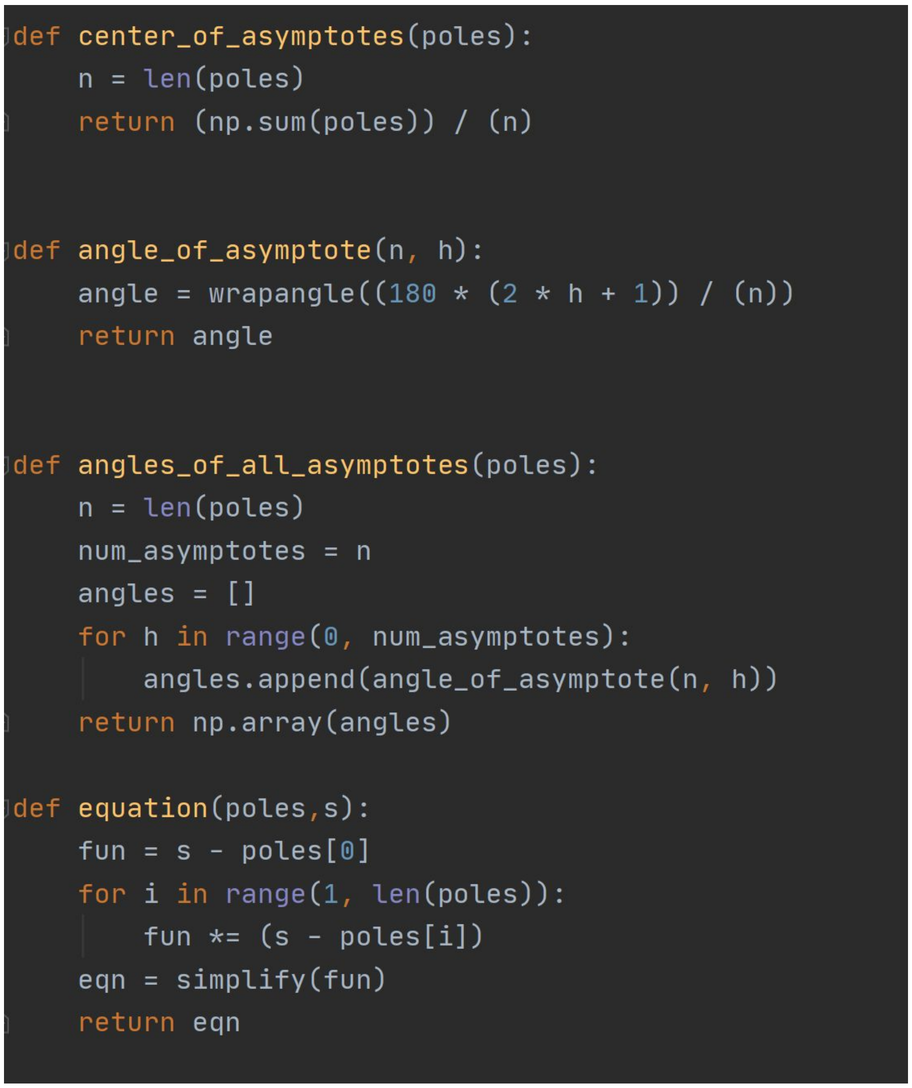
- ➔ Numpy arrays for storing iterations.
- ➔ Sympy for substituting equations.
- ➔ Matplotlib for plotting graph.

Algorithms

- Drawing the poles (there are no zeros).
- Draw the asymptotes lines(the blue dotted lines).
- Getting break away points between any two real poles and drawing them.
- Getting the intersections with the imaginary axis using Routh and drawing them
- Getting Departure angles for complex poles and drawing the lines(the green lines)
- Drawing approximated curves that start at a complex pole or the break away point and end at the intersection with the imaginary axis following the asymptotes.

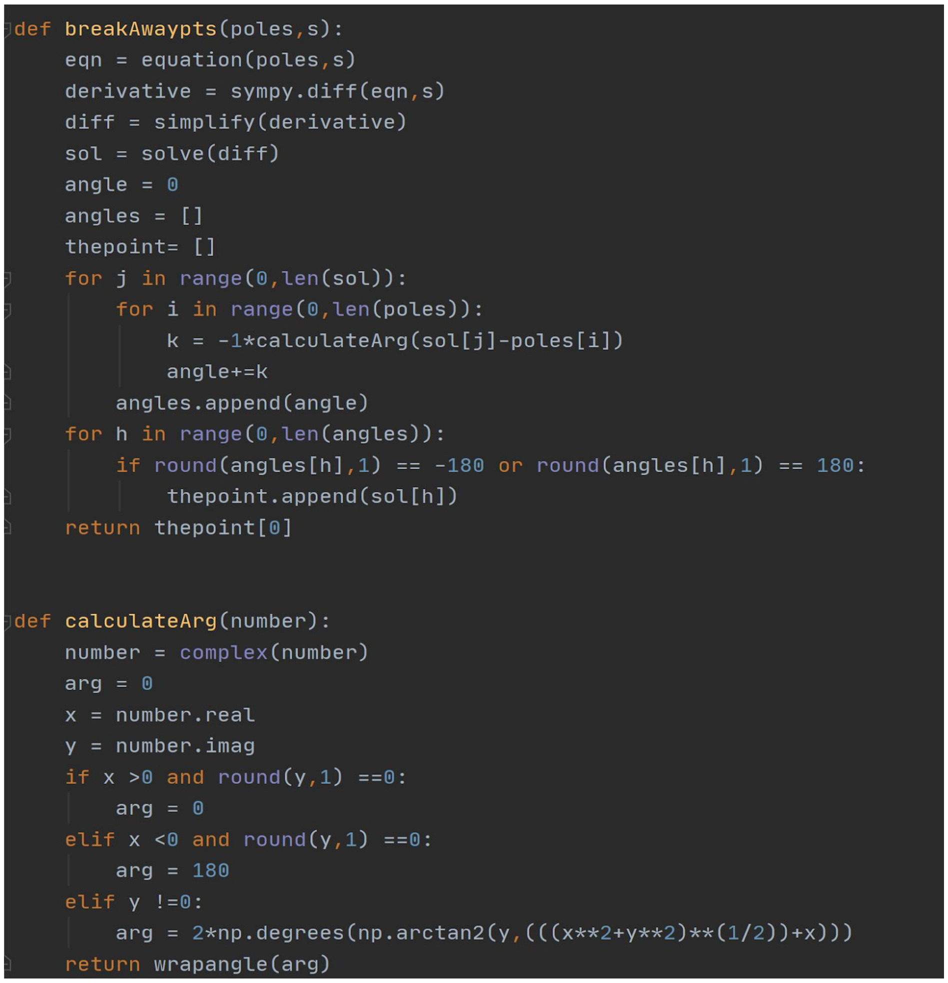
Code Snippets

Calculating the asymptotes center and angles



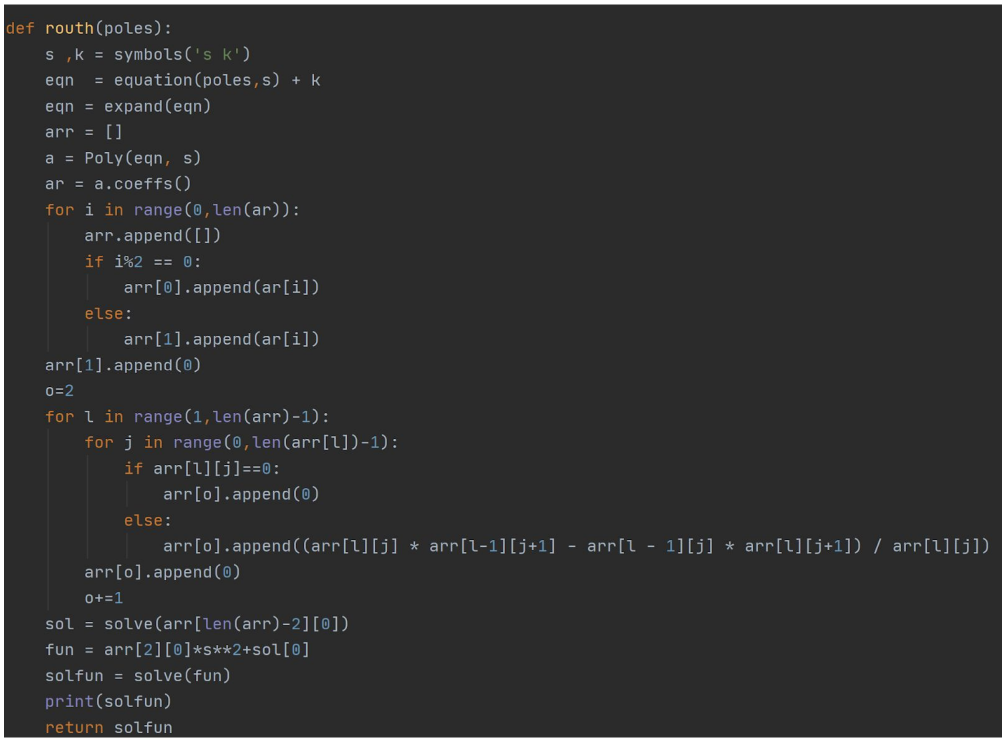
```python
def center_of_asymptotes(poles):
    n = len(poles)
    return (np.sum(poles)) / (n)


def angle_of_asymptote(n, h):
    angle = wrapangle((180 * (2 * h + 1)) / (n))
    return angle


def angles_of_all_asymptotes(poles):
    n = len(poles)
    num_asymptotes = n
    angles = []
    for h in range(0, num_asymptotes):
        angles.append(angle_of_asymptote(n, h))
    return np.array(angles)

def equation(poles,s):
    fun = s - poles[0]
    for i in range(1, len(poles)):
        fun *= (s - poles[i])
    eqn = simplify(fun)
    return eqn
```

Calculating the break away points



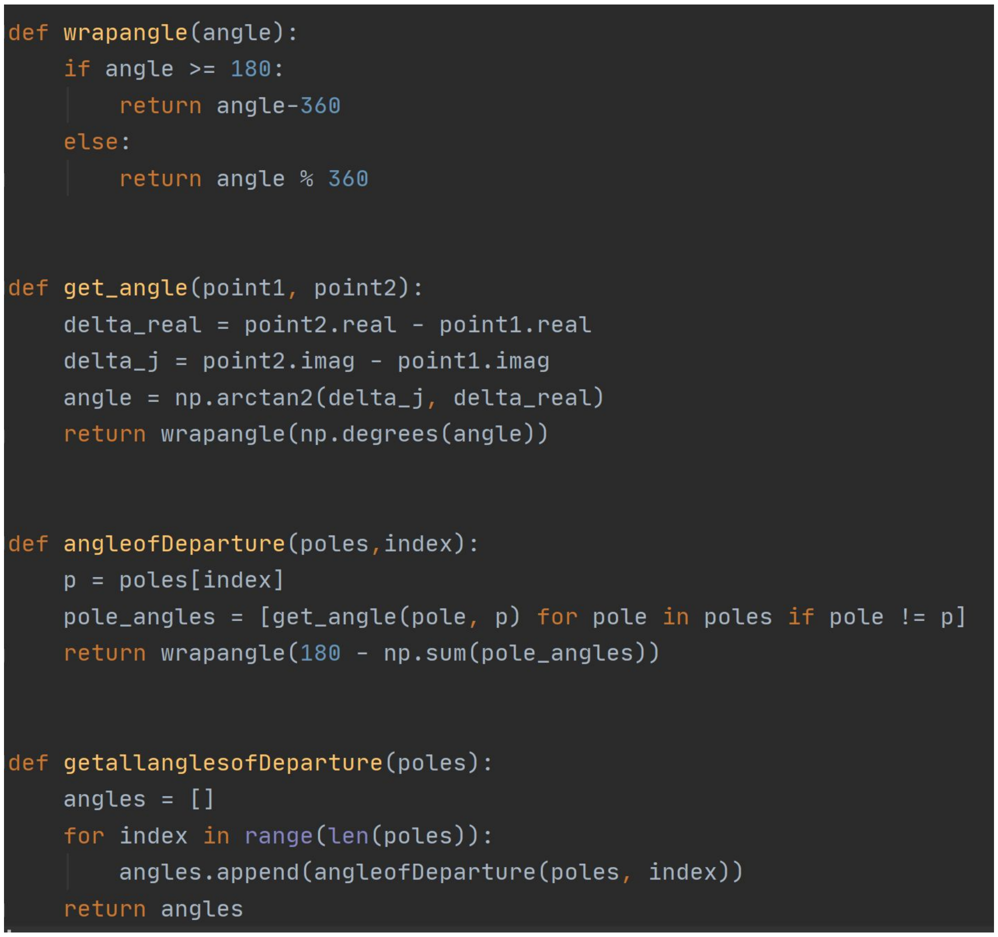
```python
def breakAwaypts(poles,s):
    eqn = equation(poles,s)
    derivative = sympy.diff(eqn,s)
    diff = simplify(derivative)
    sol = solve(diff)
    angle = 0
    angles = []
    thepoint= []
    for j in range(0,len(sol)):
        for i in range(0,len(poles)):
            k = -1*calculateArg(sol[j]-poles[i])
            angle+=k
        angles.append(angle)
    for h in range(0,len(angles)):
        if round(angles[h],1) == -180 or round(angles[h],1) == 180:
            thepoint.append(sol[h])
    return thepoint[0]


def calculateArg(number):
    number = complex(number)
    arg = 0
    x = number.real
    y = number.imag
    if x >0 and round(y,1) ==0:
        arg = 0
    elif x <0 and round(y,1) ==0:
        arg = 180
    elif y !=0:
        arg = 2*np.degrees(np.arctan2(y,(((x**2+y**2)**(1/2))+x)))
    return wrapangle(arg)
```

Calculating the imaginary crossings using routh



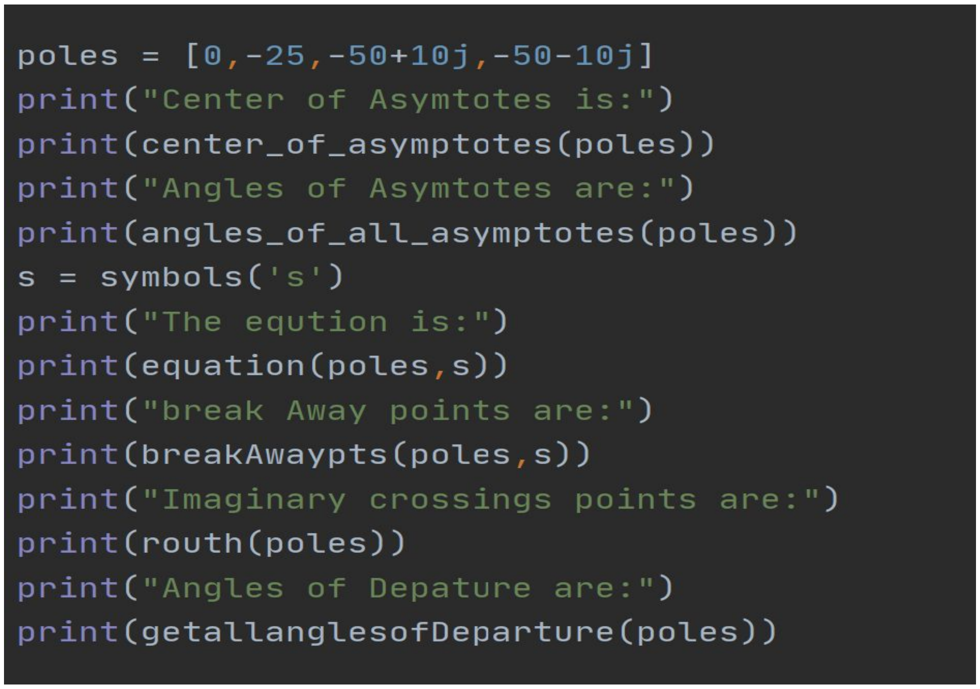
```python
def routh(poles):
    s ,k = symbols('s k')
    eqn  = equation(poles,s) + k
    eqn = expand(eqn)
    arr = []
    a = Poly(eqn, s)
    ar = a.coeffs()
    for i in range(0,len(ar)):
        arr.append([])
        if i%2 == 0:
            arr[0].append(ar[i])
        else:
            arr[1].append(ar[i])
    arr[1].append(0)
    o=2
    for l in range(1,len(arr)-1):
        for j in range(0,len(arr[l])-1):
            if arr[l][j]==0:
                arr[o].append(0)
            else:
                arr[o].append((arr[l][j] * arr[l-1][j+1] - arr[l - 1][j] * arr[l][j+1]) / arr[l][j])
        arr[o].append(0)
        o+=1
    sol = solve(arr[len(arr)-2][0])
    fun = arr[2][0]*s**2+sol[0]
    solfun = solve(fun)
    print(solfun)
    return solfun
```

Calculating the angles of departure

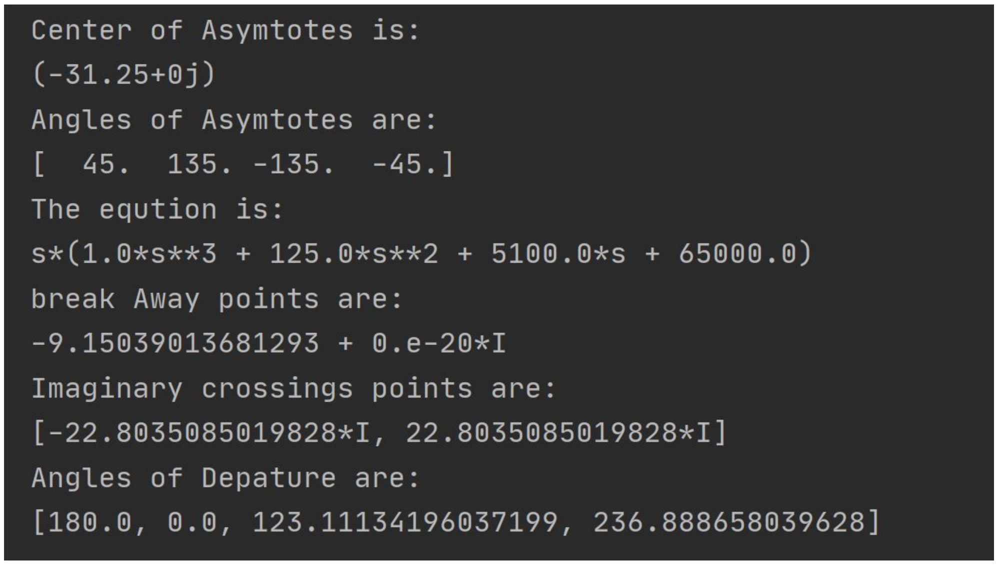


```python
def wrapangle(angle):
    if angle >= 180:
        return angle-360
    else:
        return angle % 360


def get_angle(point1, point2):
    delta_real = point2.real - point1.real
    delta_j = point2.imag - point1.imag
    angle = np.arctan2(delta_j, delta_real)
    return wrapangle(np.degrees(angle))


def angleofDeparture(poles,index):
    p = poles[index]
    pole_angles = [get_angle(pole, p) for pole in poles if pole != p]
    return wrapangle(180 - np.sum(pole_angles))


def getallanglesofDeparture(poles):
    angles = []
    for index in range(len(poles)):
        angles.append(angleofDeparture(poles, index))
    return angles
```

Calculations



```
poles = [0,-25,-50+10j,-50-10j]
print("Center of Asymtotes is:")
print(center_of_asymptotes(poles))
print("Angles of Asymtotes are:")
print(angles_of_all_asymptotes(poles))
s = symbols('s')
print("The eqution is:")
print(equation(poles,s))
print("break Away points are:")
print(breakAwaypts(poles,s))
print("Imaginary crossings points are:")
print(routh(poles))
print("Angles of Depature are:")
print(getallanglesofDeparture(poles))
```



```
Center of Asymtotes is:
(-31.25+0j)
Angles of Asymtotes are:
[  45.  135. -135.  -45.]
The eqution is:
s*(1.0*s**3 + 125.0*s**2 + 5100.0*s + 65000.0)
break Away points are:
-9.15039013681293 + 0.e-20*I
Imaginary crossings points are:
[-22.8035085019828*I, 22.8035085019828*I]
Angles of Depature are:
[180.0, 0.0, 123.11134196037199, 236.888658039628]
```

Graph

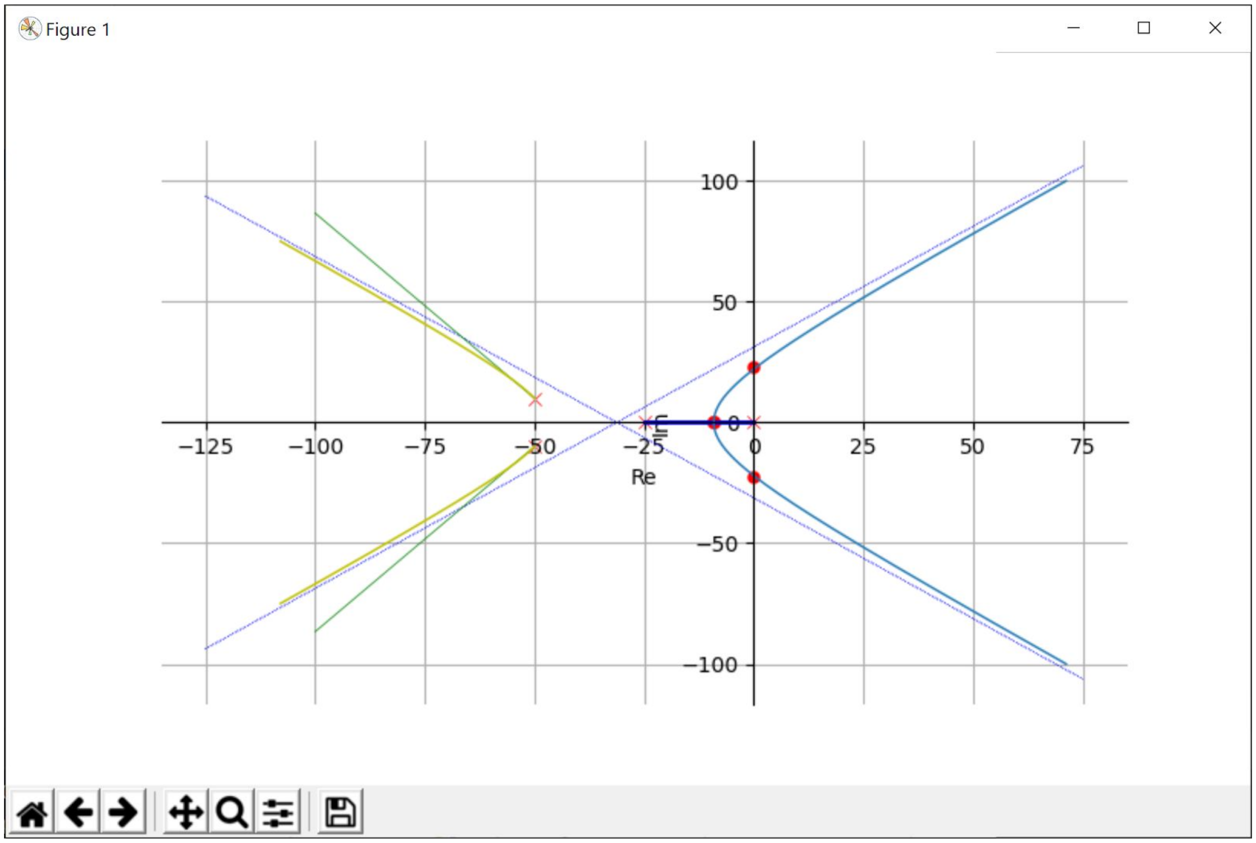


Figure 1
Re
Im