Root Locus

Name : Hamza Hassan Mohammed
Id : 26

Contents

Objective and statement

Given the poles of a system

$$S = 0 , S = -25 , S = -50 + 10\,j , S = -50 - 10\,j$$

We are required to draw the root locus and to calculate the different parameters which help in drawing using the rules.

Algorithm details

1- Getting the angles

Getting the angles is computed easily by substituting in the equation

$$\theta = \frac{\pm 180(2\mathrm{q} + 1)}{\mathrm{n} - \mathrm{m}}$$

n: the number of poles

m: the number of zeros

```
def get_angles():
    res = set(())
    for i in range(0, n - m):
        res.add((((int)(180 * (2 * i + 1) / (n - m))) % 360 + 360) % 360)
        res.add((((int)(-180 * (2 * i + 1) / (n - m))) % 360 + 360) % 360)

    return res
```

2- Getting the centroid

The centroid Is calculated using the formula

$$\text{centroid} = \frac{\sum \text{poles} - \sum \text{zeros}}{\mathrm{n} - \mathrm{m}}$$

n: the number of poles

m: the number of zeros

```
def get_centroid():
    sum_poles = sum(poles_real)
    return sum_poles / (n - m)
```

3- Get the breaking away points

The points are calculated by getting the roots of the derivative of the function

$$(s)(s+25)\big(s-(-50+10j)\big)\big(s-(-50-10j)\big)+K=0$$

$$-k = s^{\{4\}} + 125s^3 + 5100s^2 + 65000s$$

$$\frac{d\mathrm{k}}{ds} = 4s^3 + 374s^2 + 10200s + 65000$$

$$putting\ \frac{d\mathrm{k}}{ds} = 0$$

We found out that $s = -19.15039$ is the only option (as it leads to a positive K)

```
def get_breaking_down_s():
    function_string = get_funtion_string()
    function = Function(function_string)
    roots_of_derivatives = function.get_derivative_root()
    res = list()
    function = Function("-1 *" + function_string)
    for root in roots_of_derivatives:
        try:
            val = eval(str(function.get_value_at(root)))
            if (val > 0):
                # print(root.evalf())
                res.append(root.evalf())
        except:
            pass

    return res
```

4- Angle of departure

We iterate for all poles which has an imaginary part and calculate the angle of departure by the rule

$$\theta_d = 180 - \left(\sum \text{angle resulting by connecting this pole with the all other poles}\right)$$

```
def get_angle_of_depature():
    res = []
    sz = len(den)
    for i in range(0, n):
        sum = 0
        if poles_complex[i] != 0:
            for j in range(0, n):
                if j == i:
                    continue
                else:
                    denominator = poles_real[i] - poles_real[j]
                    if abs(denominator) < EPS:
                        if (poles_complex[i] - poles_complex[j] > 0):
                            sum += 90
                        else:
                            sum += 270
                    else:
                        sum += math.atan(
                            (poles_complex[i] - poles_complex[j]) / (poles_real[i] - poles_real[j])) * 180 / math.pi
            res.append(((180 - sum) % 360 + 360) % 360)
        else:
            res.append(None)
    return res
```

5- Intersection with imaginary axis

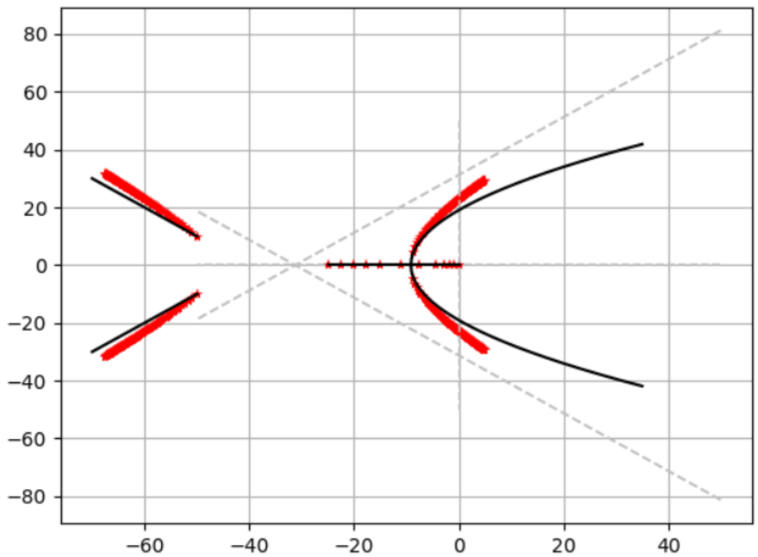
This part is calculated using Routh stability criteria and is used to get the ω Corresponding to the k at which the system is unstable (or critically stable)

```
def get_intersection_with_img_axis():
    sz = len(den)

    raws = n
    cols = 1 + (int)(sz / 2)
    table = [["0" for i in range(cols)] for j in range(raws)]
    k = 0
    for i in range(0, sz, 2):
        table[0][k] = "(" + str(den[i]) + ")"
        k += 1
    k = 0
    for i in range(1, sz, 2):
        table[1][k] = "(" + str(den[i]) + ")"
        k += 1
    if (sz % 2 == 1):
        if eval(table[0][-1]) != 0:
            table[0][-1] += 'x'
        else:
            table[0][-1] = 'x'
    else:
        if eval(table[1][-1]) != 0:
            table[1][-1] += 'x'
        else:
            table[1][-1] = 'x'
    # steps of routh
    cols = len(table[0])
    for i in range(2, n):
        for j in range(0, cols - 1):
            current = "("
            current += "(" + str(table[i - 1][0]) + "*" + table[i - 2][j + 1] + "-" + table[i - 2][0] + "*" + \
                    table[i - 1][j + 1] + ")"
            current += "/" + str(table[i - 1][0])
            current += ")"
            try:
                val = eval(current)
                table[i][j] = "(" + str(val) + ")"
            except:
                table[i][j] = current
            # table[i].append( (table[i-1][0] "*" table[i-2][j+1] "+" table[i-2][0] "*" table[i-1][j+1])/table[i-1][0])

    # print(table)
    equation = table[-1][0]
    function = Function(equation)
    x = function.get_root()[0]
    table[-2][1] = eval(table[-2][1])
    w = math.sqrt(eval(str(table[-2][1]) + "/" + str(table[-2][0])))
    return [w, -w]
```

6- Plotting



Gray dotted lines are the asymptotic lines

Black lines are root locus drawn manually

Red curves are root locus drawn by changing the values of K

a- Drawing the real locus by changing values of k

```
# Draw the real locus by changing the values of K
xs = []
ys = []
s = get_funtion_string()
for k in range(0, 5000000, 50000):
    f = Function(s + "+" + str(k))
    l = f.get_root()
    for root in l:
        c = complex(root)
        xs.append(c.real)
        ys.append(c.imag)
plt.plot(xs, ys, 'r*', mew=0.05)
```

b- Drawing the locus manually

For the breaking away curve, the curve is substituted by a parabola with suitable parameters

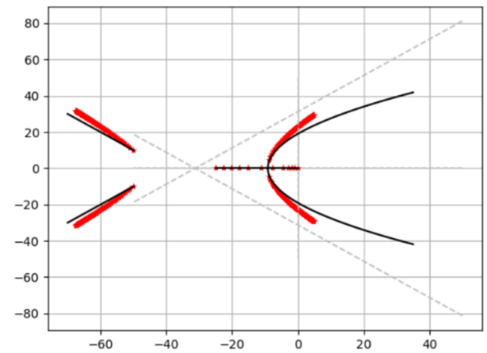
For the locus on the left side, they were substituted by lines which are parallel to the asymptotic lines.

Output and snippet

when running the program, the methods described above is called and the results appears in the console.

```
---------The results  of the program-----
The equation you enter is  (1* x**4+125* x**3+5100* x**2+65000* x**1+0* x**0)
angles of asymptotes are  {225, 315, 45, 135}
Astoroid is  -31.25
S which leads to a break down is  -9.15039013681293
angles of departure for the given poles are  [None, None, 123.11134196037199, 236.8886580396279]
The Intersection with the img axis occurs at  [22.80350850198276, -22.80350850198276]
```

Then the plotting window appears



Libraries used

1- Matplotlib

   The library which is used to plot the points

2- Sympy

- It helped finding the roots of the derivative when calculating the breaking away Point

Link to the repository

https://github.com/Hamzawy63/Root_Locus