HW6

October 10, 2022

Group number
3

Group members
Alexander Stoustrup
Mathias Tyranski
Benjamin Simonsen

[668]:

```
import numpy as np
from scipy.optimize import minimize
import scipy.signal as si
import sympy as sp
import control as ct
from typing import List
from sympy.plotting import plot
import matplotlib.pyplot as plt
from IPython.display import display, Latex, Math, Image

def eq_disp(varstring, expr, unit=""):
    if hasattr(expr, "_repr_latex_"):
        expr=(expr._repr_latex_()).replace('$', '')
    else:
        expr=f"{expr}".replace(" ", "\;")
    display(Latex(f"${varstring}={expr} \: {unit}$"))

def reduce_feedback(G_fwd, G_bwd):
    """Assumes feedback is deducted from signal, if not
    change sign of feedback"""
    return sp.simplify(G_fwd/(1+G_fwd*G_bwd))

def RHarray(coeffs: List):
    # first 2 rows from coefficients
    n = len(coeffs)
    arr = sp.zeros(n, n//2+2)
    i = 0
    for i in range(0,n,2):
        arr[0, i//2] = coeffs[i]
    for i in range(1,n,2):
```

```
        arr[1, i//2] = coeffs[i]

    for j in range(2, arr.shape[0]):
        for i in range(arr.shape[1]-1):
            a0 = arr[j-2,0]
            a3 = a1 = arr[j-1,i+1]
            a1 = arr[j-1,0]
            a2 = arr[j-2,i+1]
            arr[j, i] = (a1*a2-a0*a3)/a1
    return arr
```

1 E7.5

[669]:
```
s = sp.symbols('s')
L = (s**2 + 2*s + 10)/(s**4 + 38*s**3 + 515*s**2 + 2950*s + 6000)
eq_disp('L(s)', L)

T = reduce_feedback(L, 1)
eq_disp('T(s)', T)
```

$$L(s) = \frac{s^2 + 2s + 10}{s^4 + 38s^3 + 515s^2 + 2950s + 6000}$$

$$T(s) = \frac{s^2 + 2s + 10}{s^4 + 38s^3 + 516s^2 + 2952s + 6010}$$

And the $P(s)$ of the system is

[670]:
```
P = L
eq_disp('P(s)', P)
```

$$P(s) = \frac{s^2 + 2s + 10}{s^4 + 38s^3 + 515s^2 + 2950s + 6000}$$

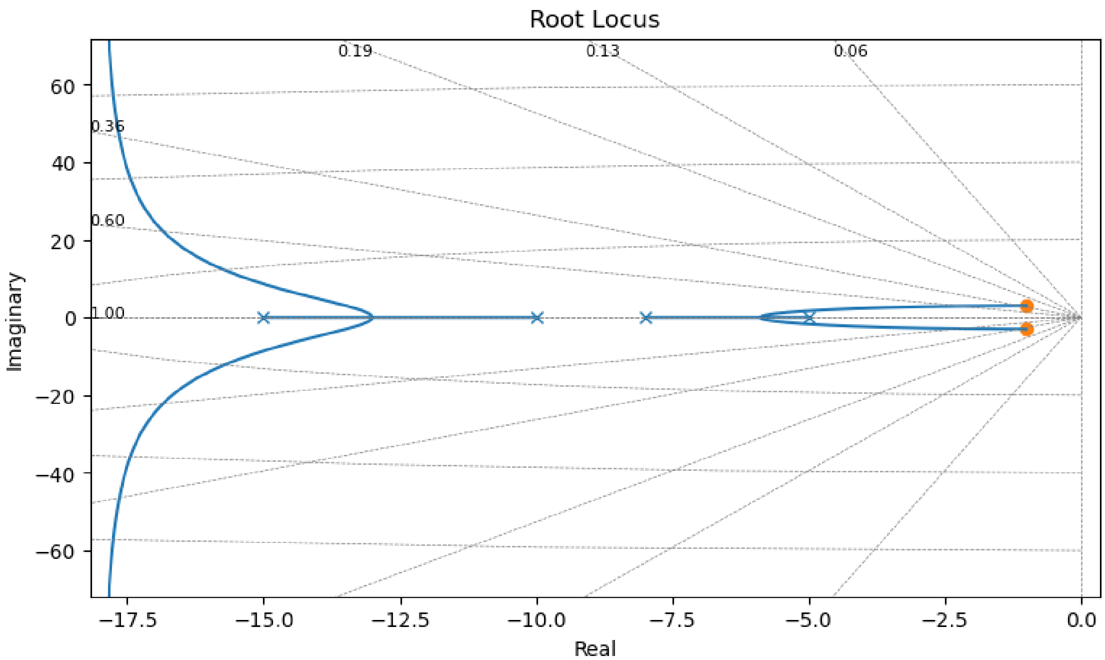
Create transfer function

[671]:
```
s = ct.tf('s')
T = (s**2 + 2*s + 10)/(s**4 + 38*s**3 + 516*s**2 + 2952*s +6010)
P = (s**2 + 2*s + 10)/(s**4 + 38*s**3 + 515*s**2 + 2950*s +6000)
```

1.1 a)

Plotting the Root Locus

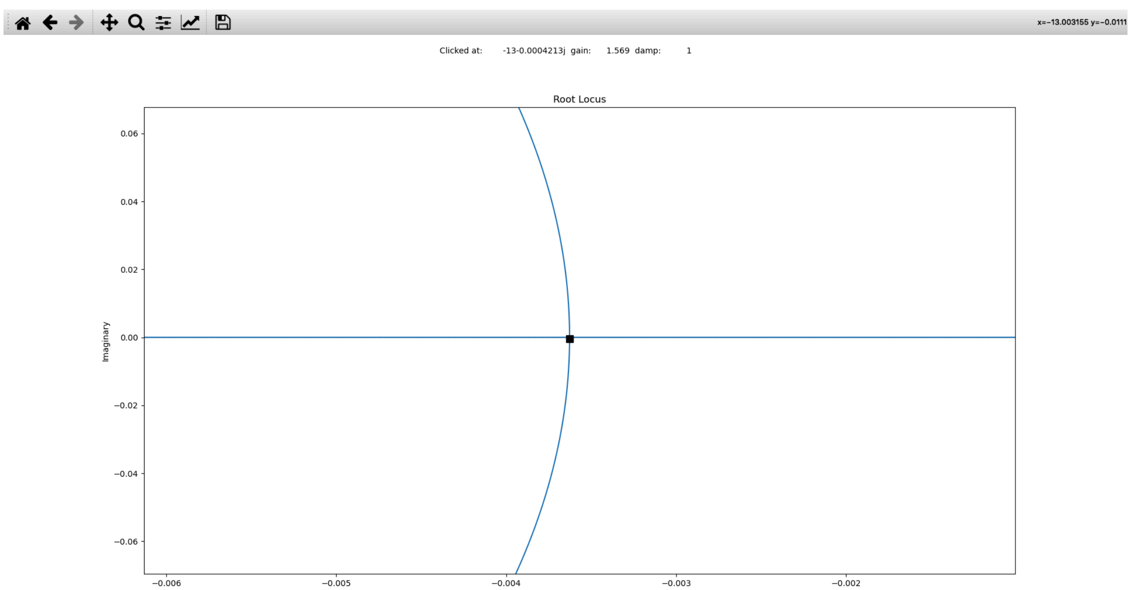
[672]:
```
fig, ax = plt.subplots(figsize=(9, 5))

rlist, klist = ct.rlocus(P)
```



The breakaway points are read from the Root Locus

[673]:
```python
Image('plots/breakaway1.png')
```

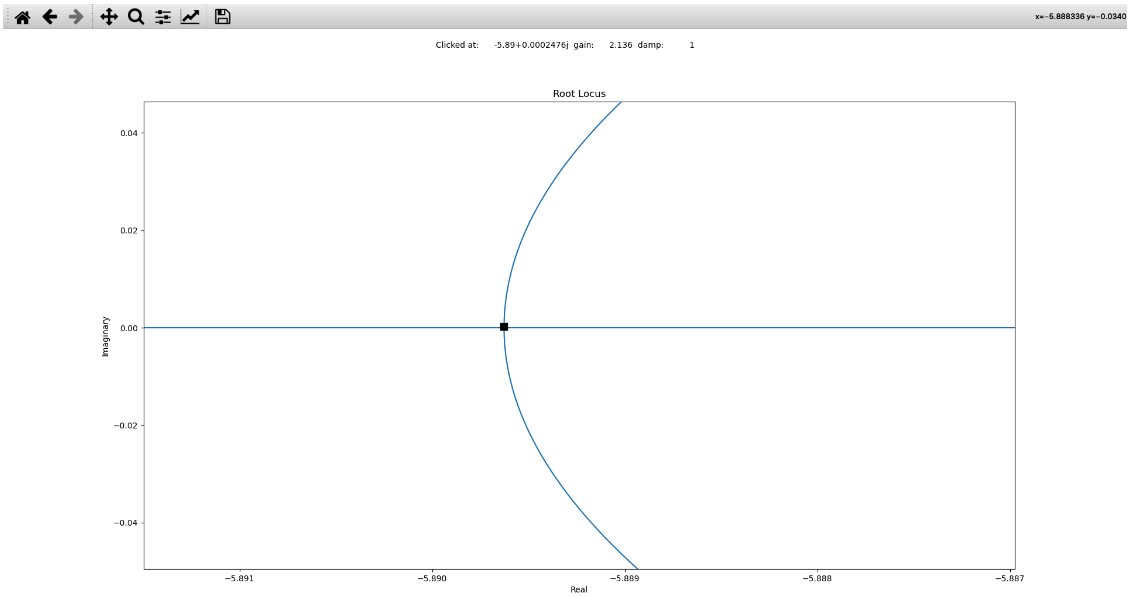
[673]:



[674]:
```python
bp1 = (-13.0,0)
```

[675]:
```
Image('plots/breakaway2.png')
```

[675]:



[676]:
```
bp2 = (-5.89,0)
```

So the breakaway points are

[677]:
```
eq_disp('bp_1', bp1)
eq_disp('bp_2', bp2)
```

$bp_1 = (-13.0,\ 0)$

$bp_2 = (-5.89,\ 0)$

1.2 b)

[678]:
```
zeros = P.zeros()
poles = P.poles()
n = poles.size
M = zeros.size

eq_disp("z_i", zeros)
eq_disp('p_k', poles)
```

$z_i = [-1.+3.j\ \ -1.-3.j]$

$p_k = [-15.+0.j\ \ -10.+0.j\ \ -8.+0.j\ \ -5.+0.j]$

Compute the asymptote centroid

[679]:
```
sigma_A = (sum(poles) - sum(zeros))/(n-M)
eq_disp('\\sigma_A', np.round(sigma_A,3))
```

$\sigma_A = (-18 + 0j)$

The angle of the asymptotes

[680]:
```
phi_A = (2*0 + 1)/(n - M)*180
eq_disp('\\phi_A', phi_A, '^\\circ')
```

$\phi_A = 90.0^\circ$

or

[681]:
```
eq_disp('\\phi_A', -phi_A, '^\\circ')
```

$\phi_A = -90.0^\circ$

1.3 c)

The gains at the breakway points can be read from the Root Locus in question a)

[682]:
```
K_bp1 = 1.57
K_bp2 = 2.14

eq_disp('K_{bp1}', K_bp1)
eq_disp('K_{bp2}', K_bp2)
```

$K_{bp1} = 1.57$

$K_{bp2} = 2.14$

2 E7.9

[683]:
```
D = 10
n_segments = 36
s, K = sp.symbols('s, K')

L = K/(s*(s**2 + 2*s + 5))

P = L/K
eq_disp('P(s)', P)

T = reduce_feedback(L, 1)
eq_disp('T(s)', T)
```

$$P(s) = \frac{1}{s\left(s^2 + 2s + 5\right)}$$

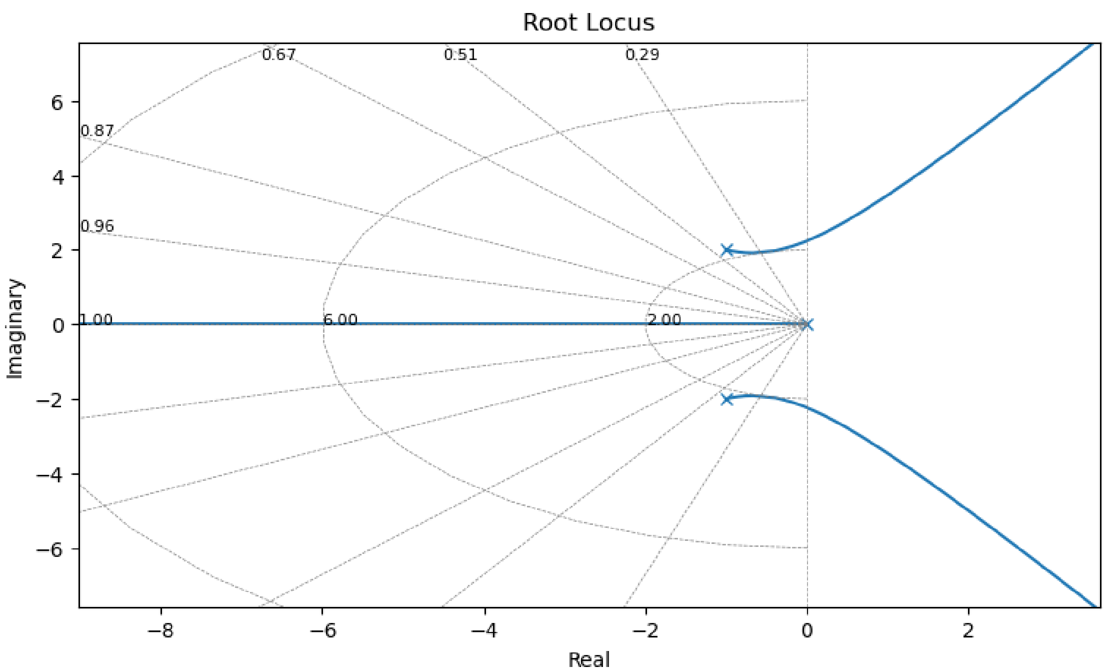
$$T(s) = \frac{K}{K + s\left(s^2 + 2s + 5\right)}$$

Create transfer function

[684]:
```
s = ct.tf('s')
P = 1/(s*(s**2 + 2*s +5))
```

2.1 a)

[685]:
```
fig, ax = plt.subplots(figsize=(9, 5))

rlist, klist = ct.rlocus(P)
```



2.2 b)

The angle of departure from the complex poles is found using the angle criterion

$$\angle F(s) = \sum_{i=1}^{M} \angle(s + z_i) - \sum_{k=1}^{n} \angle(s + p_k) = 180^\circ + k360^\circ$$

[686]:
```
zeros = P.zeros()
poles = P.poles()
M = zeros.size
n = poles.size

eq_disp('z_i', zeros)
eq_disp('p_k', poles)
```

$z_i = []$

$p_k = [-1. + 2.j \ \ -1. - 2.j \ \ 0. + 0.j]$

Asymptote center

[687]:
```
sigma_A = (sum(poles) - sum(zeros))/(n-M)
eq_disp('\\sigma_A', np.round(sigma_A,3))
```

$\sigma_A = (-0.667 + 0j)$

Angle of assymptotes

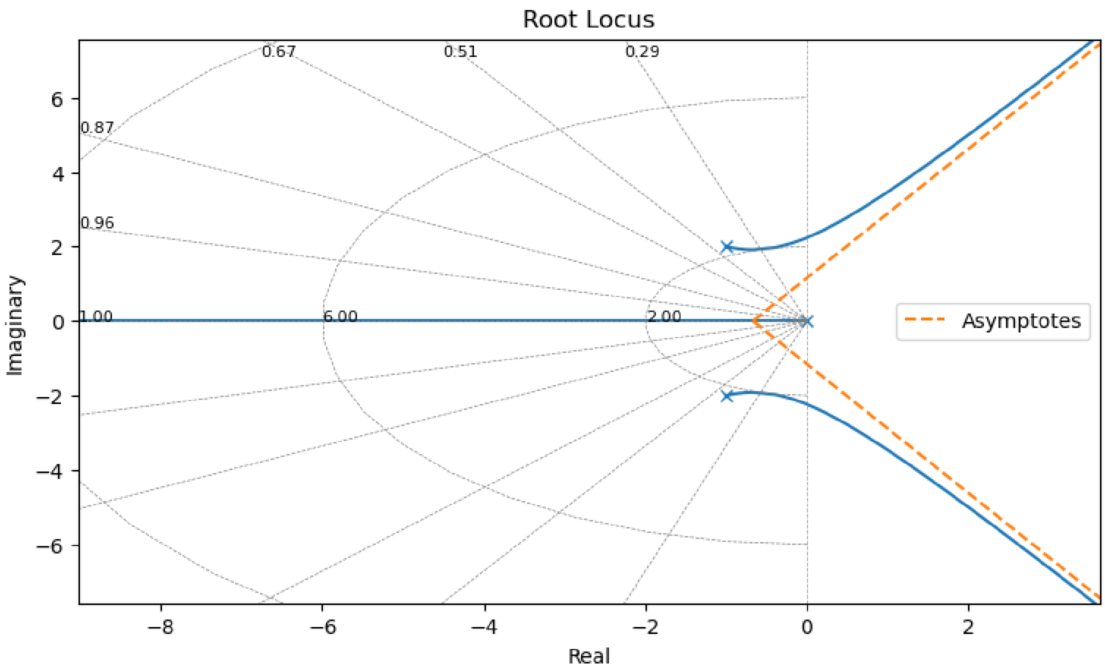
[688]:
```
phi_A = (2*0 + 1)/(n - M)*180
eq_disp('\\phi_A', phi_A, '^\\circ')
```

$\phi_A = 60.0^\circ$

Plotting the asymptotes

[689]:
```
fig, ax = plt.subplots(figsize=(9, 5))

rlist, klist = ct.rlocus(P)

line1, = ax.plot([-0.667, 10], [0, (10 + 0.667)*np.tan(phi_A*np.pi/180)], \
                 linestyle='--', color='tab:orange', label='Asymptotes')
line2, = ax.plot([-0.667, 10], [0, -(10 + 0.667)*np.tan(phi_A*np.pi/180)], \
                 linestyle='--', color='tab:orange')

ax.legend(handles=[line1])
```

[689]: <matplotlib.legend.Legend at 0x17efbc1f6d0>



2.3 b)

The angle of departure from the complex poles is found using the angle criterion

$$\angle F(s) = \sum_{i=1}^{M} \angle (s + z_i) - \sum_{k=1}^{n} \angle (s + p_k) = 180^{\circ} + k360^{\circ}$$

[690]:
```
theta_1sym = sp.symbols('theta_1')
F = -(90 + (180-np.arctan(2)*180/np.pi) + theta_1sym)
theta_1 = sp.solve(F + 180, theta_1sym)[0]
eq_disp('\\theta_1', round(theta_1,2), '^\\circ')
```

$\theta_1 = -26.57^{\circ}$

By symmetry the departure angle for the second complex pole is

[691]:
```
theta_2 = -theta_1
eq_disp('\\theta_2', round(theta_2,2), '^\\circ')
```

$\theta_2 = 26.57^{\circ}$

2.4 c)

The transfer function is

[692]:
```
s = sp.symbols('s')
eq_disp("T", T)
```

$$T = \frac{K}{K + s\left(s^2 + 2s + 5\right)}$$

Finding the coefficients

[693]:
```
p, q = T.as_numer_denom()

coeffs = sp.Poly(q, s).coeffs()
eq_disp('q(s)', sp.Poly(q, s))
for i, k in enumerate(coeffs):
    display(Latex(f"${f's^{len(coeffs)-1-i}'}: {sp.latex(k)}$"))
```

$q(s) = \operatorname{Poly}\left(s^3 + 2s^2 + 5s + K, s, domain = \mathbb{Z}\left[K\right]\right)$

$s^3 : 1$

$s^2 : 2$

$s^1 : 5$

$s^0 : K$

Determining a Routh array to find the gain where the system will be marginally stable

[694]:
```
arr = RHarray(coeffs)
arr
```

[694]:

$$\left[\begin{matrix}1 & 5 & 0 & 0\\2 & K & 0 & 0\\5 - \frac{K}{2} & 0 & 0 & 0\\K & 0 & 0 & 0\end{matrix}\right]$$

So when the two roots lie on the imaginary axis the gain is

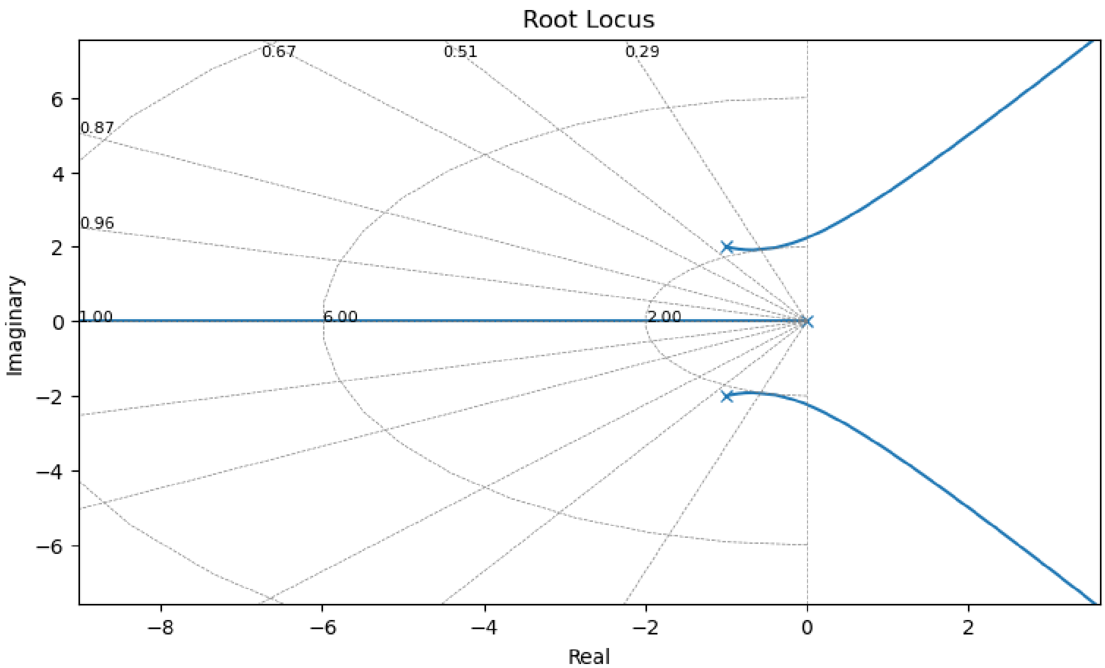
[695]:
```
K_mstable = sp.solve(arr[2,0], K)[0]
eq_disp('K', K_mstable)
```

$K = 10$

2.5 d)

The root locus is already sketched in a):

[696]:
```
fig, ax = plt.subplots(figsize=(9, 5))

rlist, klist = ct.rlocus(P)
```



3 P7.5

3.1 a)

The characteristic equation transferfunction $P(s)$, without the gain factor K, is given by

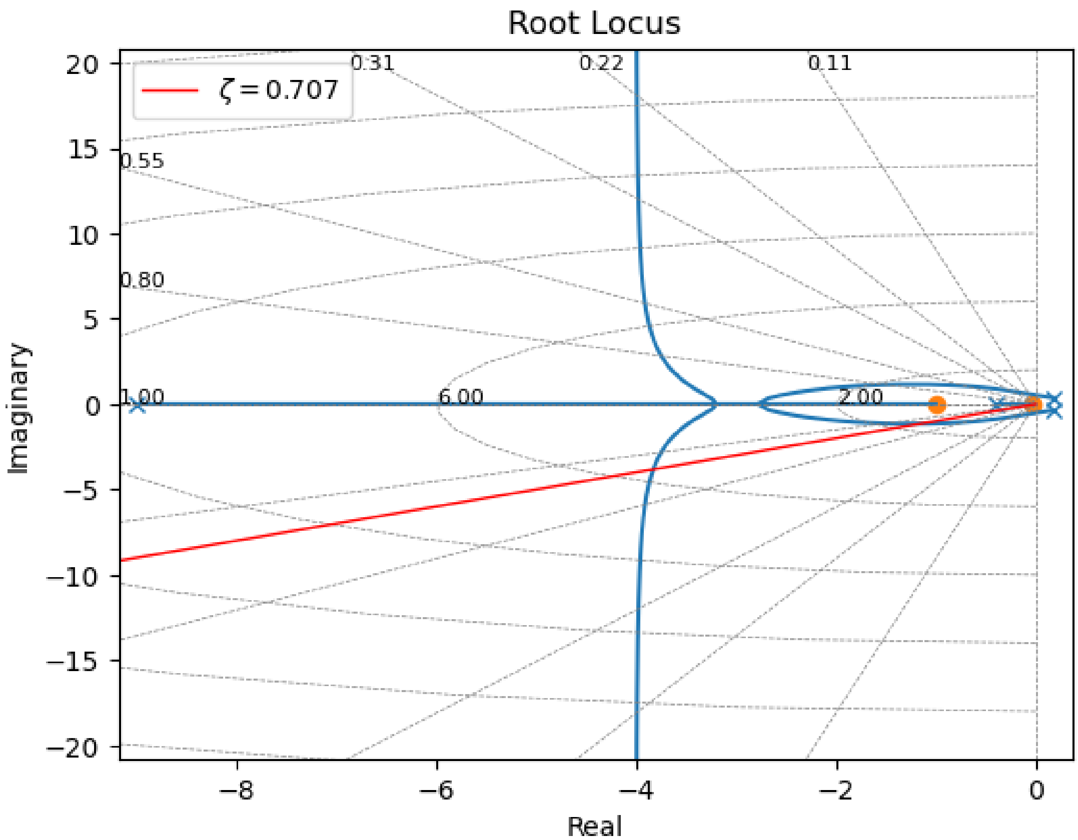
[697]:
```
s = ct.tf('s')
zeta = 0.707
G = 25*(s+0.03)/((s+0.4)*(s**2-0.36*s+0.16))
H = (s+1)/(s+9)
P = G*H
eq_disp('P', P)
```

$P = \frac{25s^2+25.75s+0.75}{s^4+9.04s^3+0.376s^2+0.208s+0.576}$

We can draw a line that represents all points with the given damping ratio, by calculating the corresponding phase angle. We can the find the points where this ratio crosses the loci, which will then be the appropiate K_2 gain

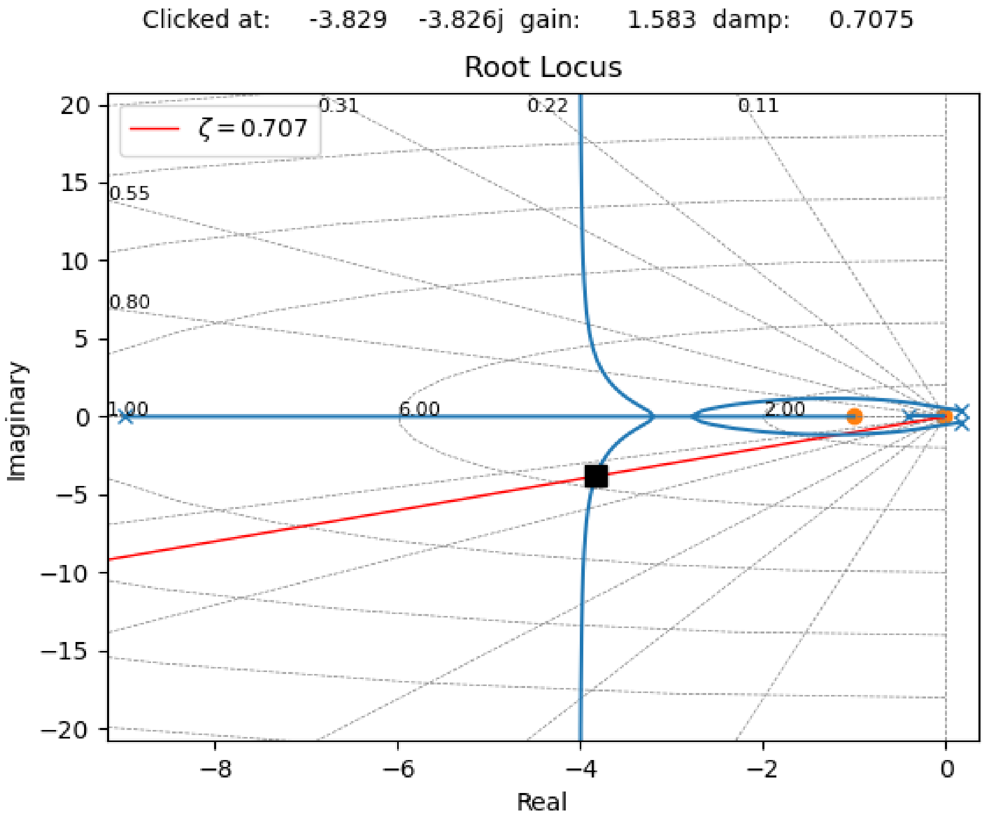
[698]:
```
theta = np.arccos(zeta)
slope = np.sin(theta)/np.cos(theta)
line = lambda x: x*slope
ax = plt.subplot()
rl = ct.rlocus(G*H, ax=ax)
span = np.r_[0:-100:-0.1]
handle = ax.plot(span, line(span), 'r', label=f'$\zeta={zeta}$', linewidth=1)
```

```
ax.legend(handles=handle)
plt.show()
```



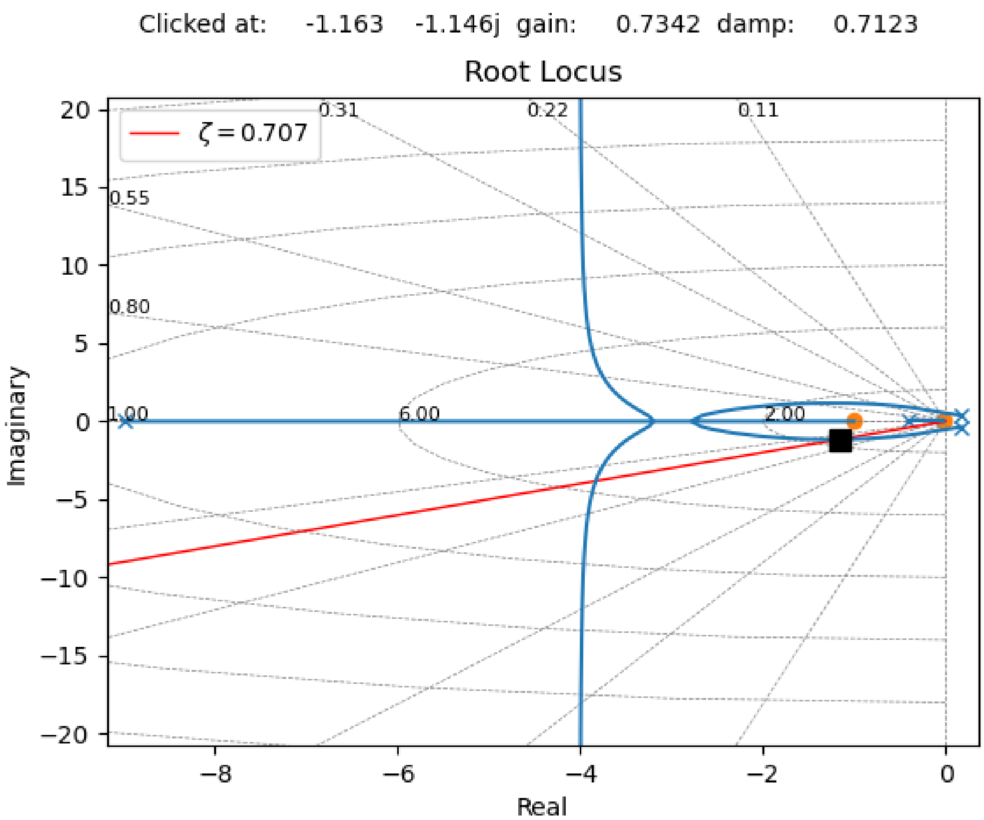
[699]:
```
Image("plots/P7.5.png")
```

[699]:



[700]:
```
Image("plots/P7.5_2.png")
```

[700]:



2 gains will result in the desired damping ratio:

$$K_2 = 0.7342$$

$$K_2 = 1.583$$

3.2 b)

We calculate the steady state error for both of the possible gains K_2

[701]:
```
K2_1 = 1.583
K2_2 = 0.7342
T_ho_1 = ct.feedback(G, K2_1*H)
T_ho_2 = ct.feedback(G, K2_2*H)
eq_disp('y_{ss}(K_2=1.583)', round(ct.dcgain(T_ho_1),3))  # calculates step␣
↪input steady state error for system transferfunc
```

$y_{ss}(K_2 = 1.583) = 3.828$

[702]:
```
eq_disp('y_{ss}(K_2=0.7342)', round(ct.dcgain(T_ho_2),3))
```

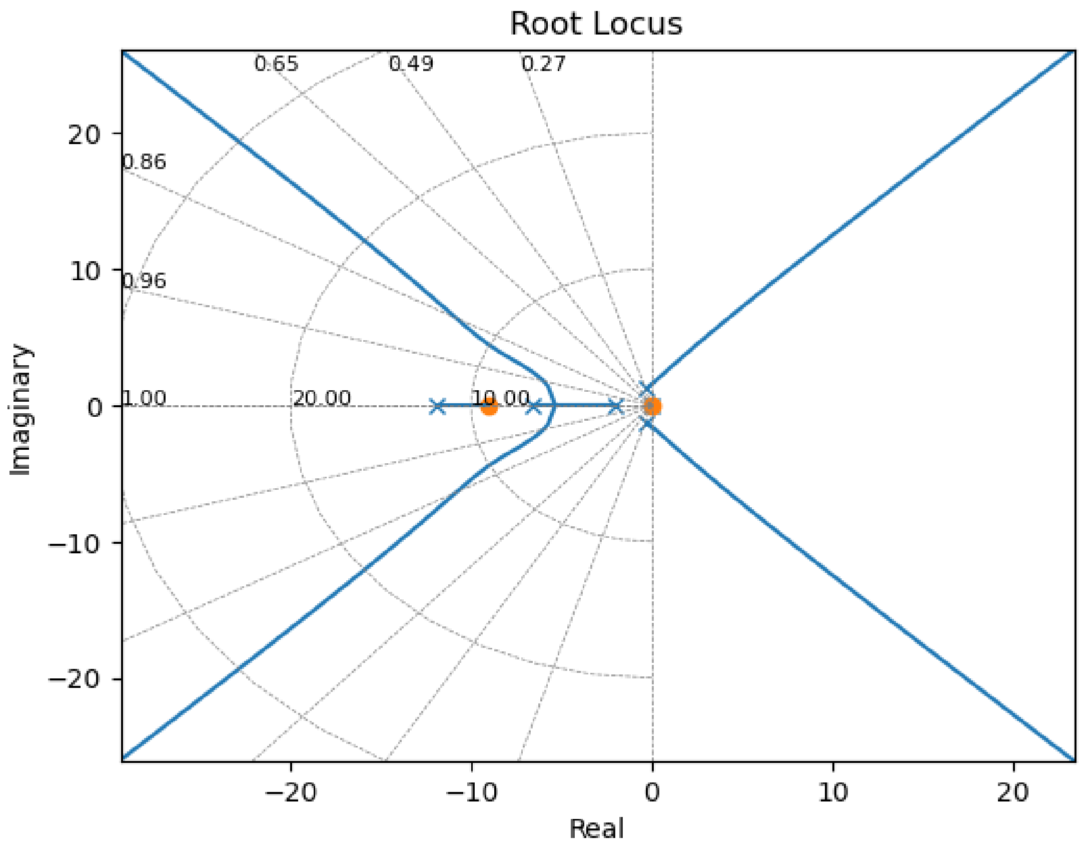
$y_{ss}(K_2 = 0.7342) = 5.991$

3.3 c)

Using $K_2 = 0.7342$ we draw the root locus of the system with the pilot in the loop

[703]:
```
Gp = 1/(s**2 + 12*s + 1)
T2 = ct.feedback(Gp*T_ho_2, 1)
```

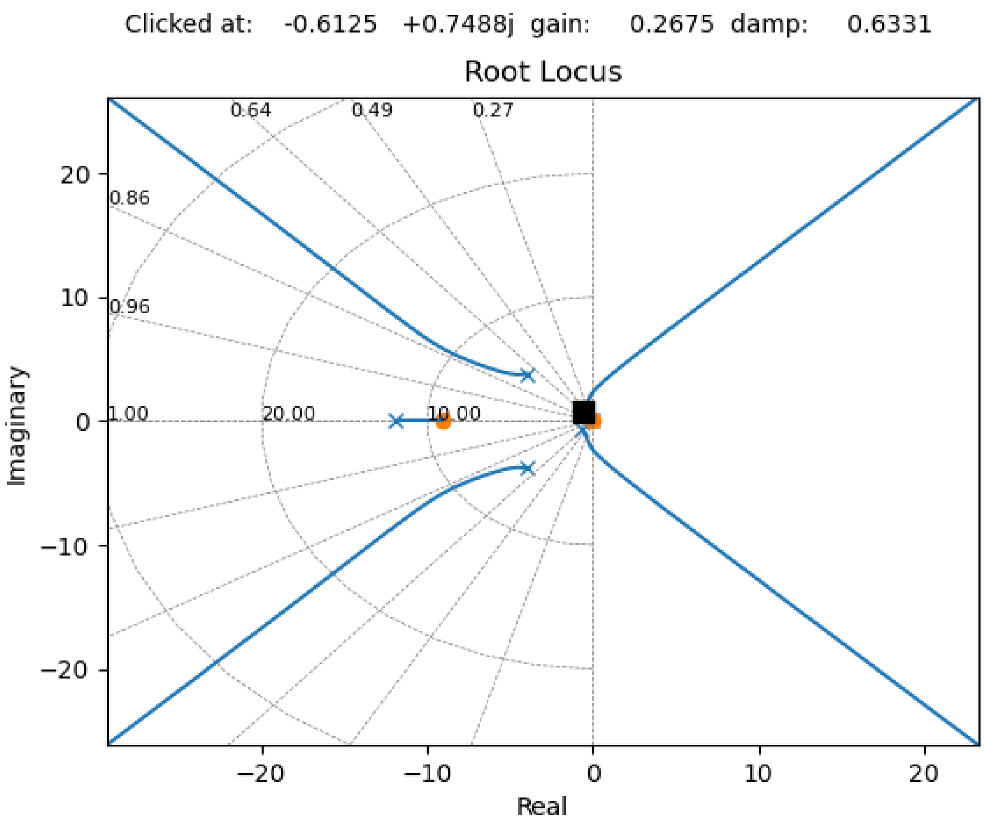
[704]:
```
%matplotlib inline
rl = ct.rlocus(T2)
```



We chose a K_1 that allows poles to be on left side of imagninary axis

[705]:
```
Image("plots/P7.5_3.png")
```

[705]:



3.4 d)

Steady state is recalcualted using $K_1 = 0.2675$

[706]:
```
K1 = 0.2675
T = ct.feedback(T_ho_1, K1*Gp)
eq_disp('y_{ss}(K_1=0.2675)',round(ct.dcgain(T),3))
```

$y_{ss}(K_1 = 0.2675) = 1.891$

4 P7.11

[707]:
```
Kb = 0.4
Kp = 1
K1 = 2
r = 0.2
tau1 = 1
taua = tau1
J_empty = 2.5*10**(-3)
J_full = 5*10**(-3)
```

```
tau_m = 0.5
KT_LJ = 2

t_rev = 6  # time to reverse must be 6ms
Ts = 3   #settling time must be 3ms
```

4.1 a)

First we simplify the system transferfunction, we then rewrite the characteristic equation to the form:

$$1 + K_a P(s) = 0$$

The characteristic equation after rewritiing the feedback loops is

[708]:
```
Ka, s = sp.symbols('Ka, s')
x = sp.symbols('x')
K2 = 10
J = J_full
photocell = 0.5*K1/(tau1*s+1)
amp = Ka/(taua*s+1)
tach = K2
motor = KT_LJ/((s+1/tau_m)/J)
sys =Kp*photocell*r/s*reduce_feedback(amp*reduce_feedback(motor*(1/(J*s)), Kb),␣
↪K2)
eq_disp('charEQ', 1 + sys)
```

$$charEQ = \frac{80.0Ka}{s\left(4000.0Ka + (s+1)\left(s\left(200.0s + 400.0\right) + 160.0\right)\right)(s+1)} + 1$$

We then rewrite to the standard form to find $P(s)$

[709]:
```
char_eq = ((1+sys)*sys.as_numer_denom()[1]).simplify()
args = sp.Add.make_args(char_eq.expand().collect(Ka))
non_Ka_terms = sum(x for x in args if not Ka in x.free_symbols)
Ka_terms = [x for x in args if Ka in x.free_symbols][0]
Psym = Ka_terms.subs(Ka,1)/non_Ka_terms
eq_disp('P(s)', Psym)
```

$$P(s) = \frac{4000.0s^2 + 4000.0s + 80.0}{200.0s^5 + 800.0s^4 + 1160.0s^3 + 720.0s^2 + 160.0s}$$

Get the factors of the numerator and denomerator, to calculate the numerical rlocus

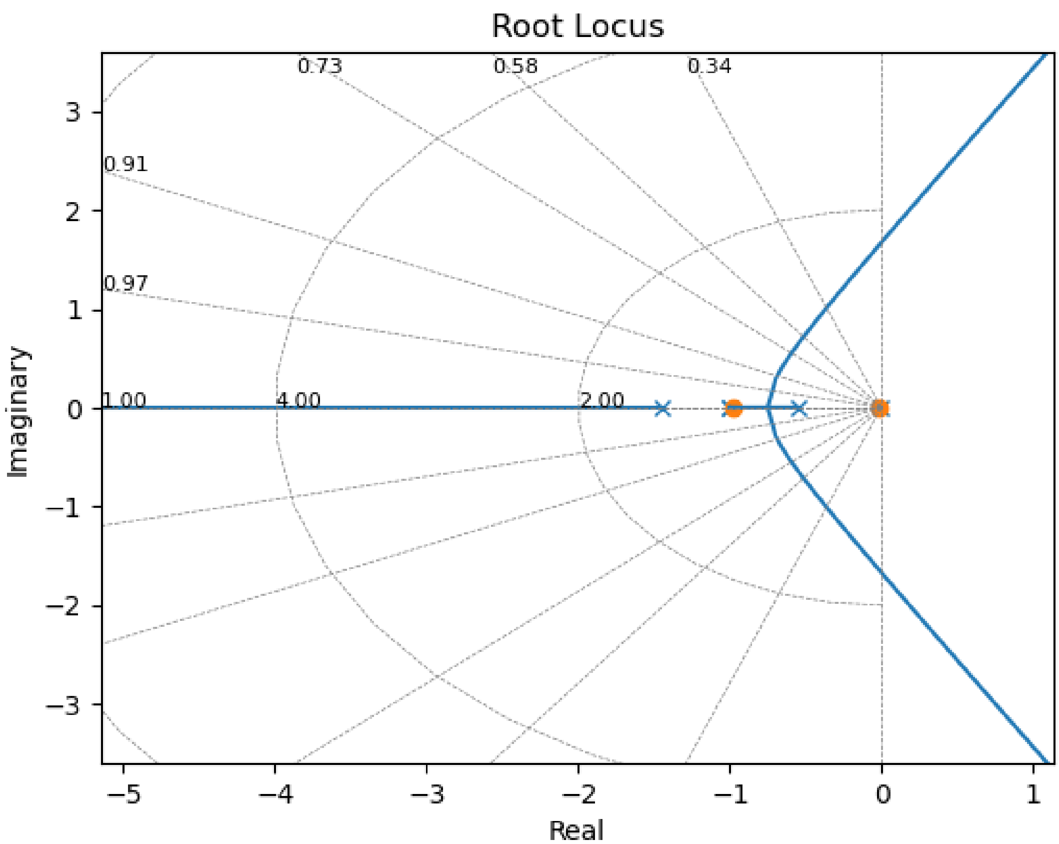
[710]:
```
num, denom = Psym.as_numer_denom()
num_c = [float(x) for x in sp.Poly(num).all_coeffs()]
denom_c = [float(x) for x in sp.Poly(denom).all_coeffs()]
display(num_c, denom_c)
```

```
[4000.0, 4000.0, 80.0]

[200.0, 800.0, 1160.0, 720.0, 160.0, 0.0]
```

```
[711]: P = ct.tf(num_c, denom_c)
eq_disp('P', P)
```

$P = \frac{4000s^2+4000s+80}{200s^5+800s^4+1160s^3+720s^2+160s}$

```
[712]: rlist, klist = ct.rlocus(P)
```

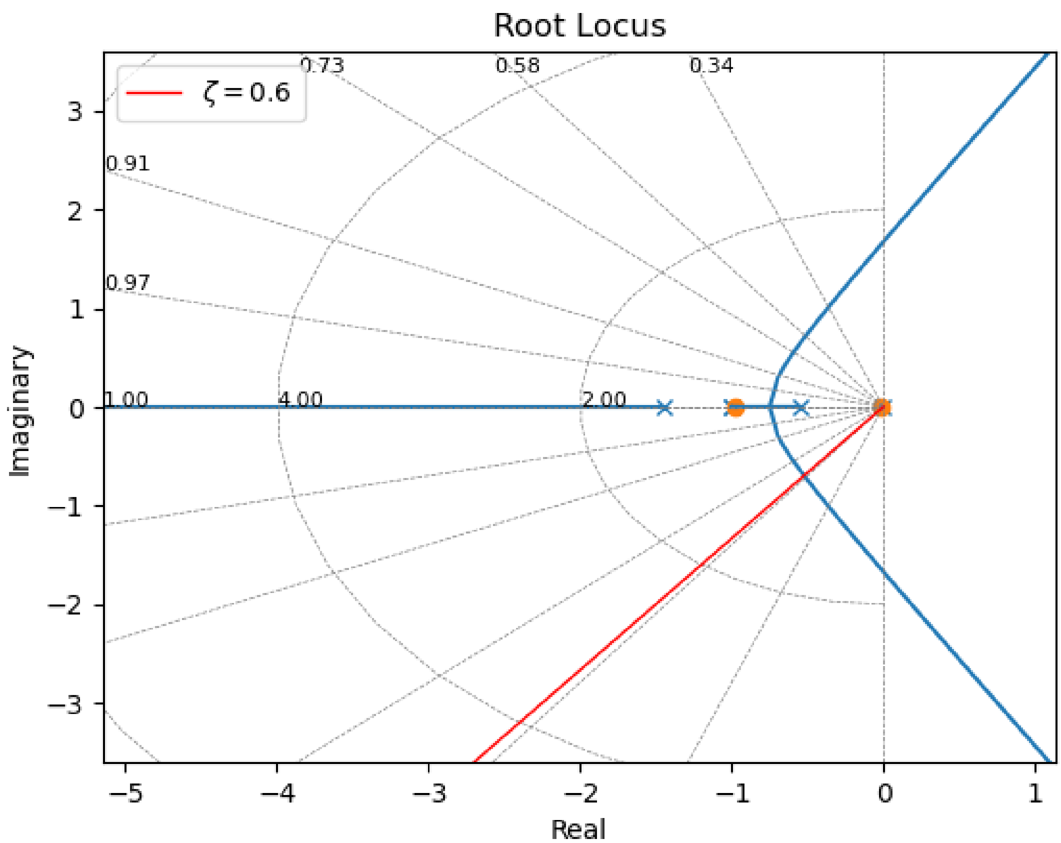


4.2 b)

We calculate the phase angle corresponding to the given damping ratio

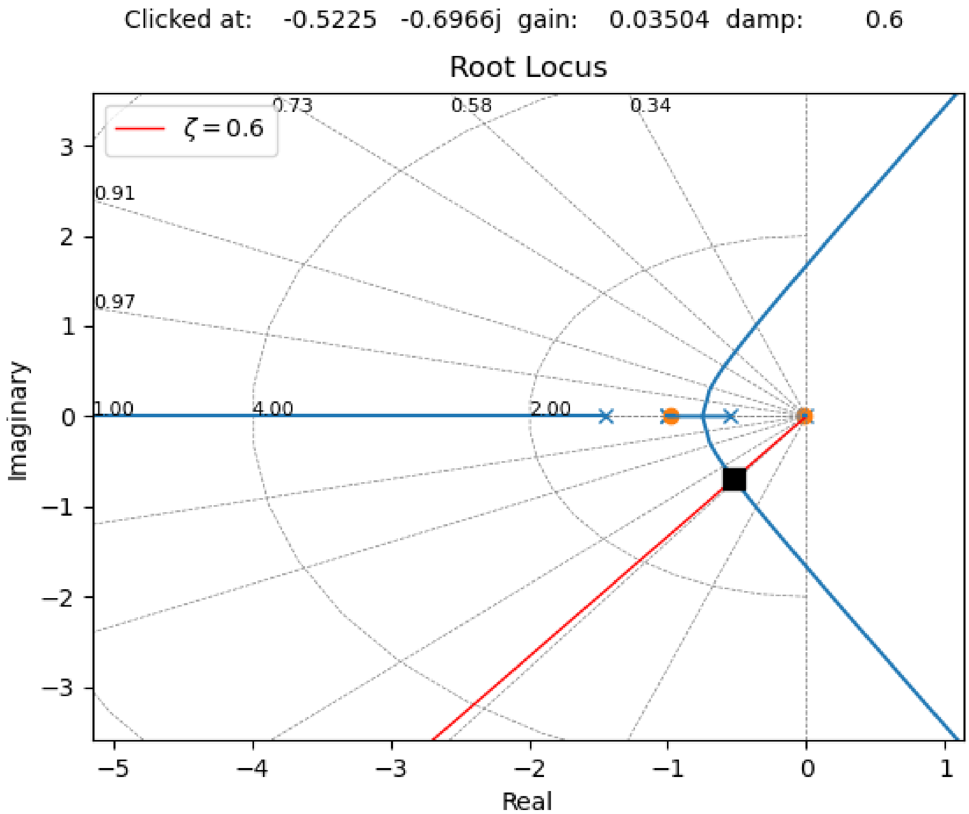
```
[713]: zeta = 0.6
theta = np.arccos(zeta)
slope = np.sin(theta)/np.cos(theta)
line = lambda x: x*slope
ax = plt.subplot()
rl = ct.rlocus(P, ax=ax)
span = np.r_[0:-100:-0.1]
handle = ax.plot(span, line(span), 'r', label=f'$\zeta={zeta}$', linewidth=1)
ax.legend(handles=handle)
```

```
plt.show()
```



[714]:
```
Image('plots/P7_11b.png')
```

[714]:



when $K_a > 0.035$ all roots have damping grater than or equal to $\zeta = 0.6$

4.3 c)

We find the standard form of the characteristic equation, like earlier

[715]:
```
s, K2 = sp.symbols('s, K2')
Ka=0.035
x = sp.symbols('x')
J = J_full
photocell = 0.5*K1/(tau1*s+1)
amp = Ka/(taua*s+1)
tach = K2
motor = KT_LJ/((s+1/tau_m)/J)
sys =Kp*photocell*r/s*reduce_feedback(amp*reduce_feedback(motor*(1/(J*s)), Kb),
    K2)
eq_disp('charEQ', 1 + sys)
```

$$charEQ = 1 + \frac{2.8}{s\left(14.0K_2 + (s+1)\left(s\left(200.0s + 400.0\right) + 160.0\right)\right)(s+1)}$$

We then rewrite to the standard form to find $P(s)$

[716]:
```
char_eq = ((1+sys)*sys.as_numer_denom()[1]).simplify()
args = sp.Add.make_args(char_eq.expand().collect(K2))
non_Ka_terms = sum(x for x in args if not K2 in x.free_symbols)
Ka_terms = [x for x in args if K2 in x.free_symbols][0]
Psym = Ka_terms.subs(K2,1)/non_Ka_terms
eq_disp('P(s)', Psym)
```

$$P(s) = \frac{14.0s^2 + 14.0s}{200.0s^5 + 800.0s^4 + 1160.0s^3 + 720.0s^2 + 160.0s + 2.8}$$

Get the poynomial factors of the numerator and denomerator, to calculate the numerical rlocus

[717]:
```
num, denom = Psym.as_numer_denom()
num_c = [float(x) for x in sp.Poly(num).all_coeffs()]
denom_c = [float(x) for x in sp.Poly(denom).all_coeffs()]
display(num_c, denom_c)
```

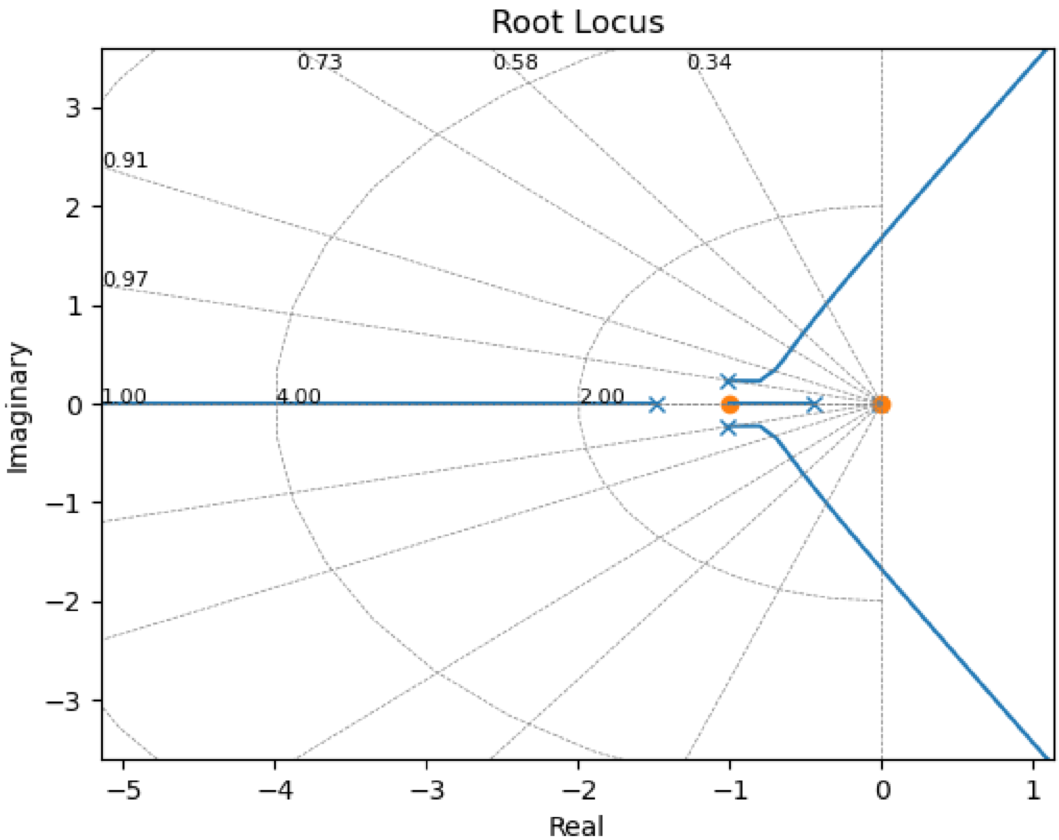
```
[14.000000000000002, 14.000000000000002, 0.0]
```

```
[200.0, 800.0, 1160.0, 720.0, 160.0, 2.8000000000000007]
```

[718]:
```
P = ct.tf(num_c, denom_c)
eq_disp('P(s)', P)
```

$$P(s) = \frac{14s^2+14s}{200s^5+800s^4+1160s^3+720s^2+160s+2.8}$$

[719]:
```
rlist, klist = ct.rlocus(P)
```


Root Locus
Imaginary
Real
0.73
0.58
0.34
0.91
0.97
1.00
4.00
2.00
3
2
1
0
−1
−2
−3
−5
−4
−3
−2
−1
0
1