

Root Locus

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import control
```

Typical design scenario

- System with multiple known parameters, and one unknown or varying parameter (K)
- The unknown parameter makes it impossible to know if system is stable or if meets our design criteria

For example:

$$G(s) = \frac{s^2 + s + 1}{s^3 + 4s^2 + Ks + 1}$$

- Changing K changes the locations of the poles
- Poles are in fact values of s s.t.: $s^3 + 4s^2 + Ks + 1 = 0$

As control engineers we have two questions at this point:

- How to design a system that meets the requirements:
 - What value of K should I choose to meet the requirements (i.e., that places the poles at the correct location in the s plane)
- What is the effect of variations on a control system that has been already designed:
 - How sensitive is the system to a value of K that is slightly different than what we have estimated (or predicted).

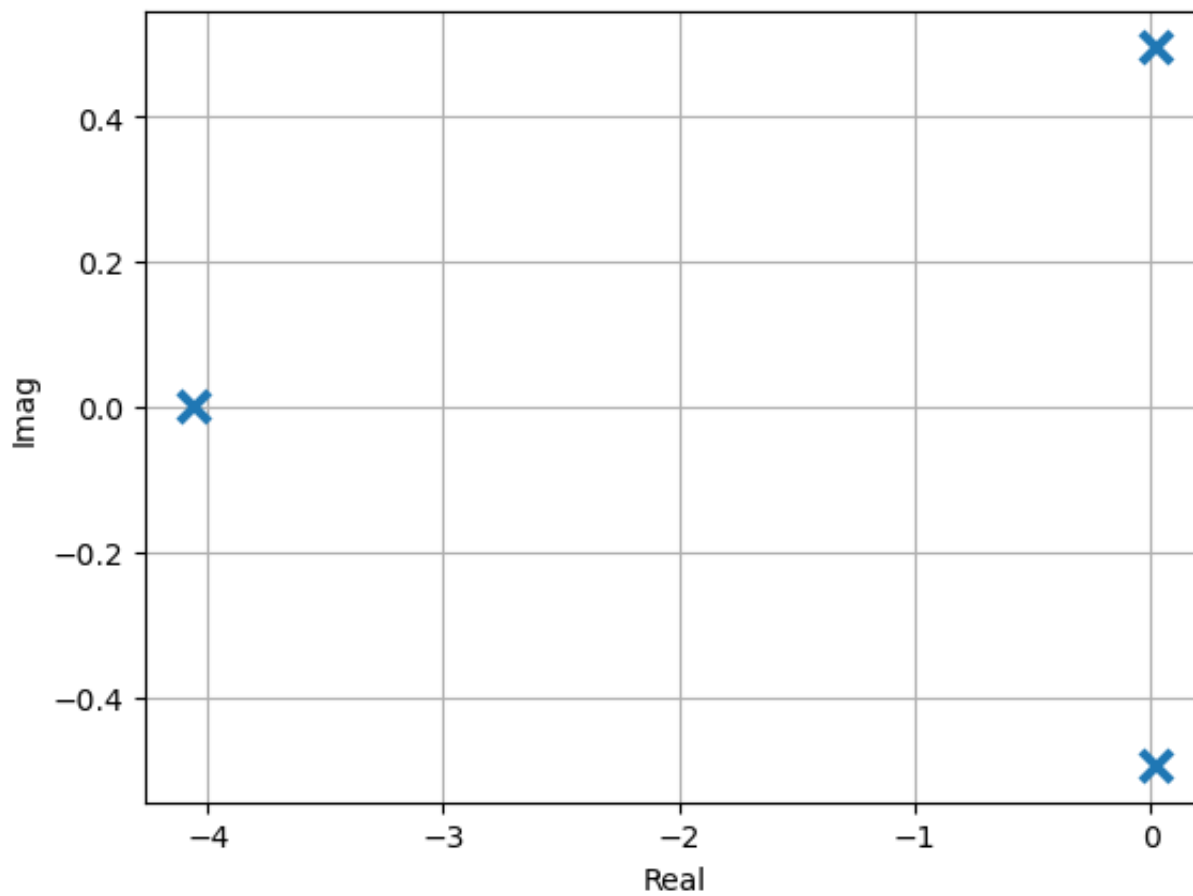
One way: pick a random value of K and plot the poles in the s plane.

For example, if $K = 0$:

```
In [ ]: K=0
poles = np.roots([1, 4, K, 1])
print('Poles for K={}: {}'.format(K, poles))
```

```
Poles for K=0: [-4.06064703+0.j          0.03032351+0.4953248j  0.03032351-0.4953248j]
```

```
In [ ]: plt.plot(np.real(poles), np.imag(poles), marker='x', markersize=10, linestyle='', linewidth=1)
plt.grid()
plt.xlabel('Real');
plt.ylabel('Imag');
```

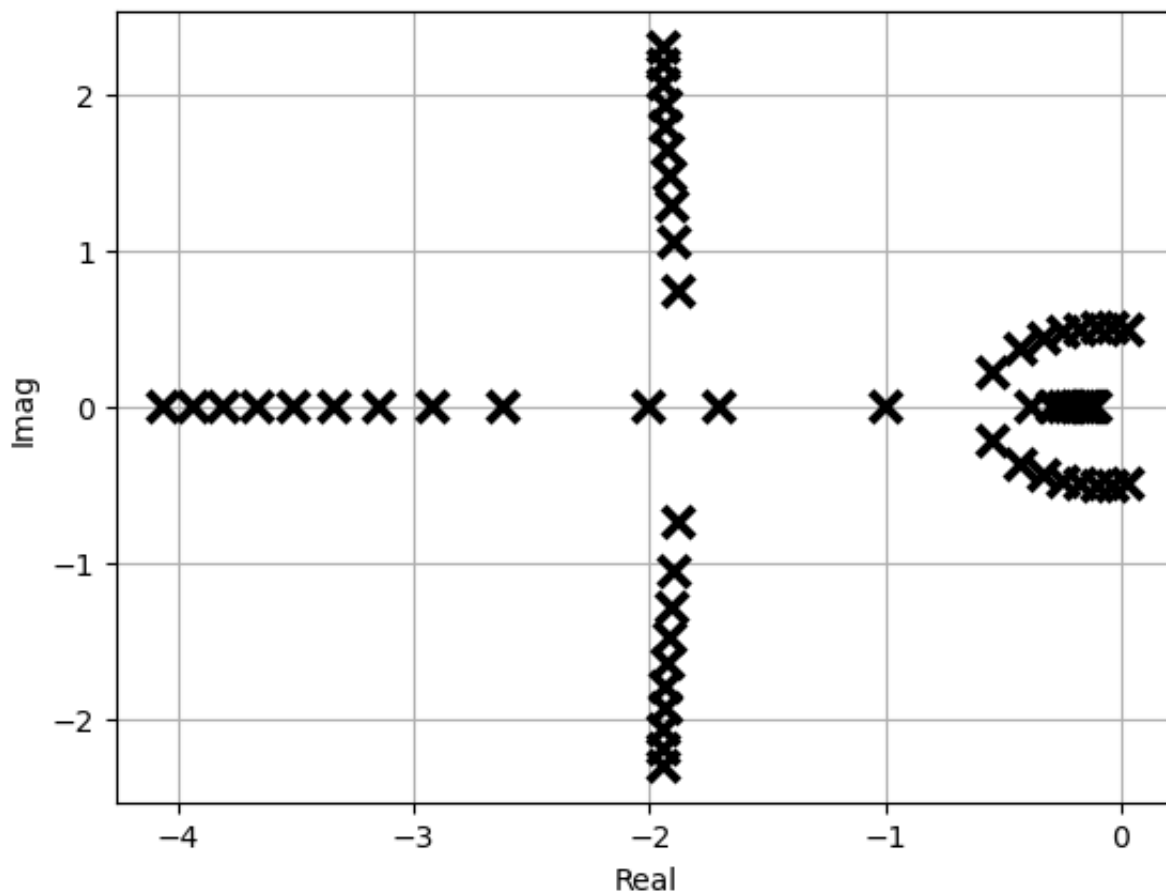


We can do it for multiple values of K , calculating the root of $s^3 + 4s^2 + Ks + 1 = 0$, for $K \rightarrow \infty$ (or until we get to a high enough value of K).

```
In [ ]: poles = []
K_range = np.arange(0, 10, 0.5)
for K in K_range:
    poles.append(np.roots([1, 4, K, 1]))
    # print('Poles for K={}: {}'.format(K, poles[-1]))
```

And plot it:

```
In [ ]: plt.plot(np.real(poles), np.imag(poles), marker='x', markersize=10, linestyle='', color='k')
plt.grid()
plt.xlabel('Real')
plt.ylabel('Imag');
```



Connecting all these points together is a drawing of how all these poles move through the s -plane as we vary K . This is what the Root Locus method does.

Interpreting the location of the poles in the s-plane

- The axis of the s -plane are: Real component (σ) along the x-axis, and Imag component (ω) on the y-axis.
- Each location of the s -plane corresponds to a specific waveform in the time domain through equation: e^{st}
- If s is real: this corresponds to exponential growth ($s>0$) or decay ($s<0$).
 - The further we are in the left half plane the faster the waveform decays
 - The further we are in the right half plane the faster the waveform grows
- If s is complex: this corresponds to a pure oscillatory motion (sinusoidal) in the time domain.
 - Always in pair (complex conjugate of each other). This is due to having to deal with real time.
 - The further they are along the ω axis, the faster the oscillation
- If s is a combination of real and imaginary values mix both exponential and sinusoidal motion.

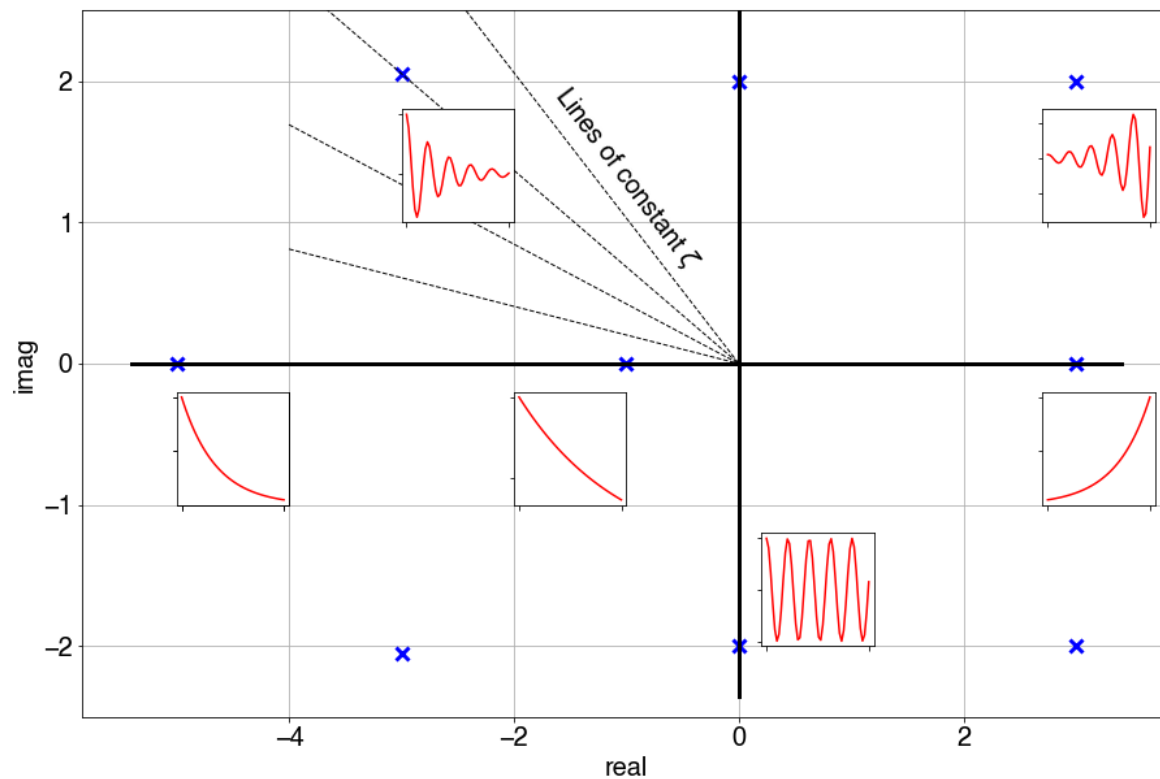
See also notebook `05_System_response`.

When we analyse a system:

- Both poles and zeros are needed to characterise the forced response to an input
- Only the poles are needed to dictate the unforced response of the system to a set of initial conditions
- This links directly to the stability of the system

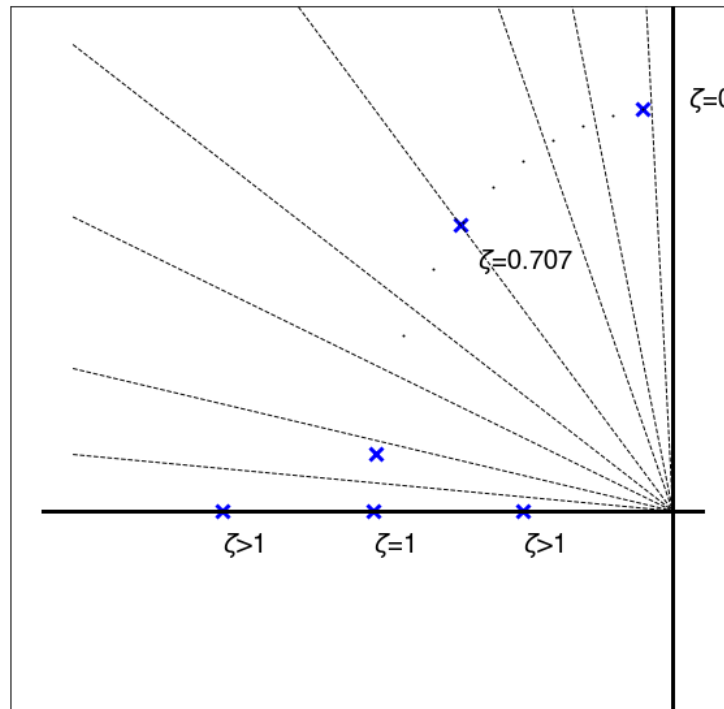
Knowing the location of the poles in the s -plane:

- Determines the stability,
- Determines if the response is oscillatory (and at what frequency), or exponential (and at what rate)
- And more: for example makes it possible to determine the damping ratio $\zeta = \cos(\theta)$: two poles on the same θ line would have the same damping ratio



A note on the damping ratio

- Normalised unitless value (ζ)
- $\zeta = 0$: no dumping. The pole is on the imaginary axis
- $\zeta = 1$: perfectly dumped. The pole is exponential and hence on the real axis.
- $\zeta \in (0, 1)$: damped oscillations (the poles are somewhere in the s-plane).
- $\zeta > 1$: over dumped (still on the real axis).



Damping ratio, natural frequency and pole location

Let's consider a second order system:

$$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

The characteristic equation is:

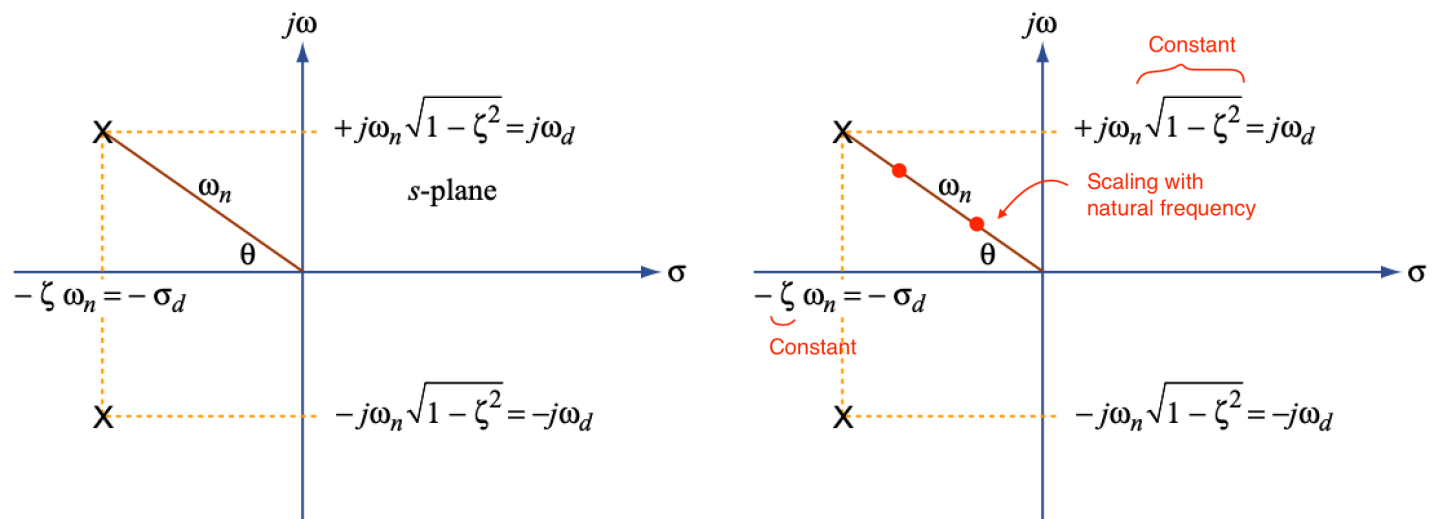
$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$$

and when we solve it:

$$s_{1,2} = \frac{-2\zeta\omega_n \pm \sqrt{4\zeta^2\omega_n^2 - 4\omega_n^2}}{2} = -\zeta\omega_n \pm \omega_n\sqrt{(\zeta^2 - 1)}$$

- if we assume $\zeta < 1$, we have complex roots

$$s_{1,2} = -\zeta\omega_n \pm \omega_n\sqrt{(1 - \zeta^2)}j$$



- If we hold ζ constant, the poles scale with ω_n
- What is the angle θ (sometime I will also call it ϕ)?

$$\sqrt{\zeta^2 \omega_n^2 + \omega_n^2 (1 - \zeta^2)} \cos(\theta) = \omega_n \zeta$$

\Downarrow

$$\cos(\theta) = \frac{\omega_n \zeta}{\sqrt{\zeta^2 \omega_n^2 + \omega_n^2 (1 - \zeta^2)}}$$

\Downarrow

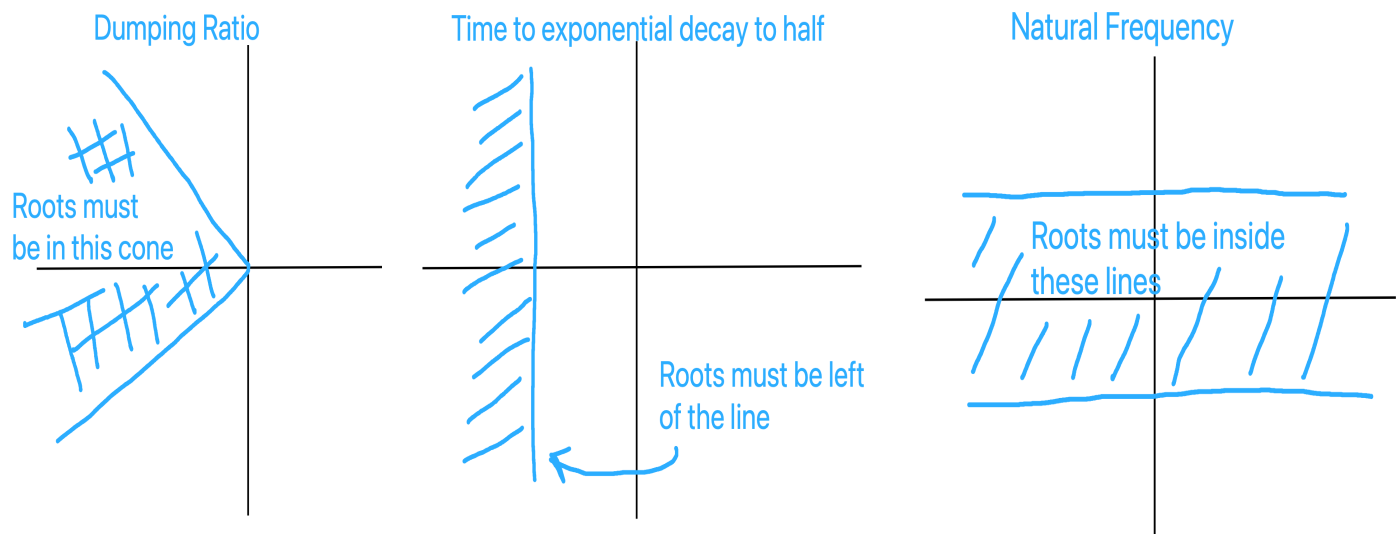
$$\cos(\theta) = \frac{\omega_n \zeta}{\sqrt{\omega_n^2 (\zeta^2 + 1 - \zeta^2)}} = \frac{\omega_n \zeta}{\omega_n} = \zeta$$

- Remember that we assumed $\zeta < 1$
- When $\zeta > 1$, the two roots separate along the real axis: we have two first order real poles.

Why are we interested in this?

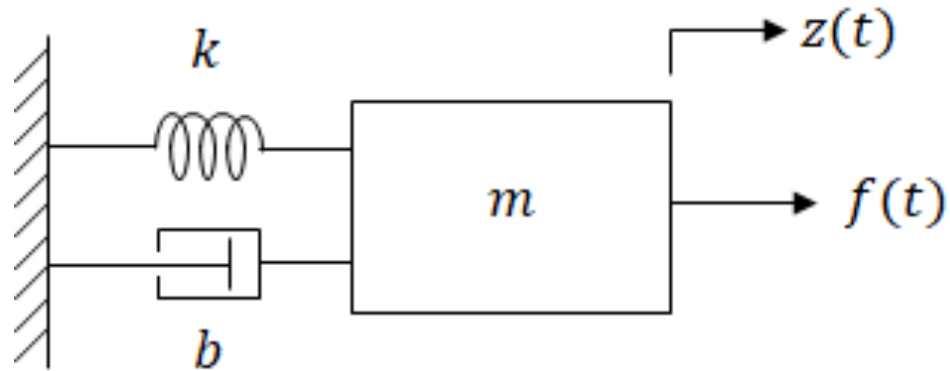
- Requirements most of the time come in the form of:
 - Damping ratio (e.g. greater than some value)
 - Natural frequency
 - Time to half, time to rise, etc.
 - Or any combination of these

And these requirements translate into constraints on the position of the poles.



A simple design problem

Suppose you need to design a spring-dumper-mass system:



- You are given the spring ($k = 1$) and the mass ($m = 1$)
- You can choose the damper that is needed to achieve the requirements
 - System shall have $\xi \geq 0.75$.

The transfer function of the system is:

$$\frac{Z(s)}{F(s)} = \frac{1}{s^2 + bs + 1}$$

The root locus method gives a method to verify how the poles move as we change the value of the parameter b from 0 to infinity.

Let's use the Python again to see what the Root Locus looks like in this case:


```

In [ ]: fig, axs = plt.subplots(1,1, figsize=(10,5))

poles = []
b_range = np.arange(0, 10, 0.001)
for b in b_range:
    poles.append(np.roots([1, b, 1]))

plt.plot(np.real(poles), np.imag(poles), marker='.', markersize=1, linestyle='', color='k',
plt.plot(np.real(poles[0]), np.imag(poles[0]), marker='x', color='g', markeredgewidth=3, li
plt.plot(np.real(poles[-1]), np.imag(poles[-1]), marker='x', color='r', markeredgewidth=3,

# desired damping lines
phi = np.arccos(0.75)
xi = 0.75
tan_theta = np.sqrt(1-xi**2)/xi

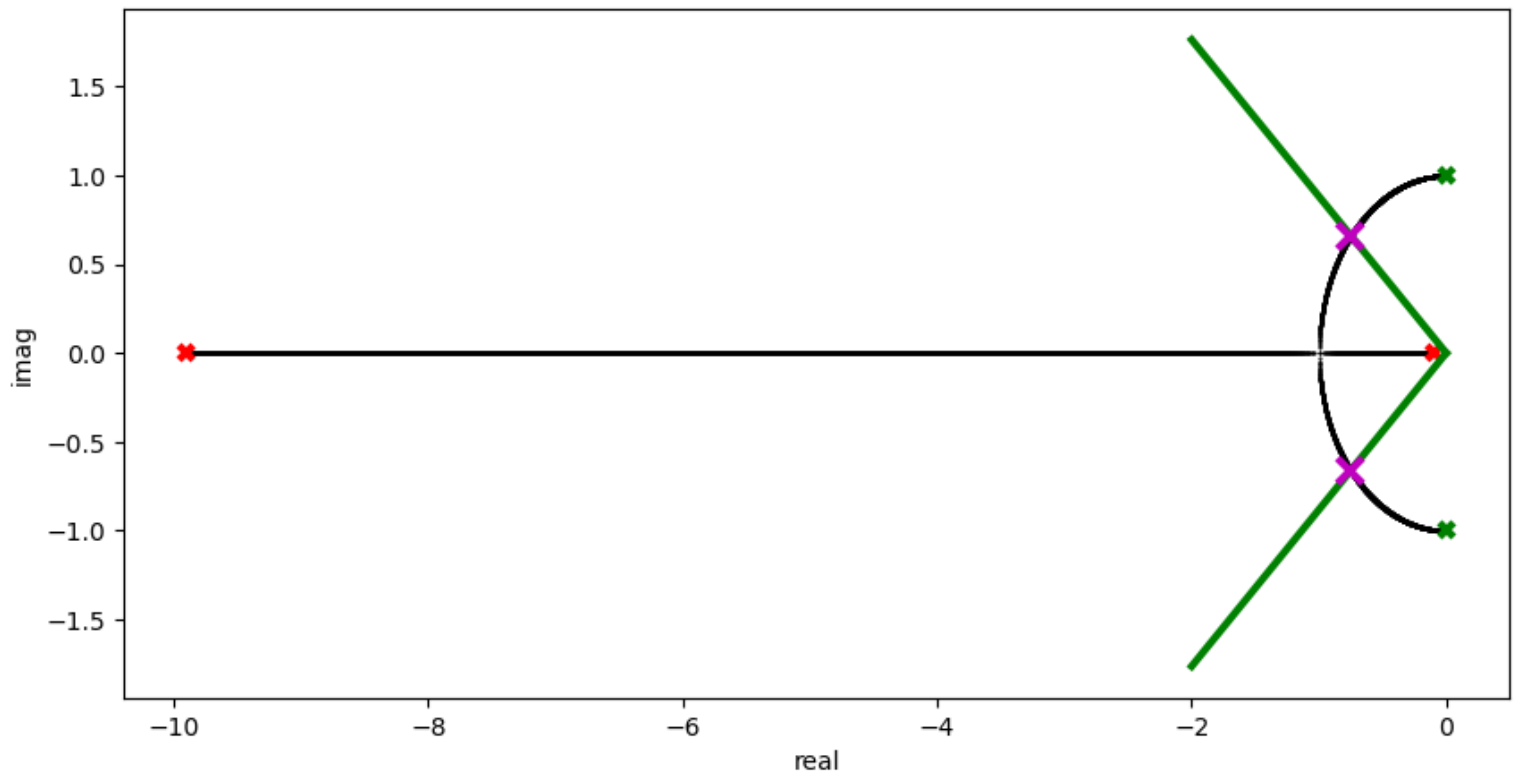
x_range = np.linspace(-2, 0)
plt.plot(x_range, -tan_theta*x_range, linewidth=3, color='g')
plt.plot(x_range, tan_theta*x_range, linewidth=3, color='g')

# Choose b = 1.5
b = 1.5
poles_desired = np.roots([1, b, 1])
print('Desired poles (magenta):', poles_desired)
plt.plot(np.real(poles_desired), np.imag(poles_desired),
         marker='x', markersize=10, linestyle='', color='m', markeredgewidth=3)

plt.xlabel('real');
plt.ylabel('imag');

```

Desired poles (magenta): [-0.75+0.66143783j -0.75-0.66143783j]



- We start from the imaginary axis ($b = 0$), as we would expect if we had no dumping: we expect oscillations
- As we increase the damping the poles move in the left half plane and towards the real axis, until for the critical damping value they hit the real axis, where they split and separate on the real line
- We can now plot the desired damping lines ($\xi \geq 0.75$)

For example, if we choose $b = 1.5$, we have poles: $[-0.75+0.66143783j \ -0.75-0.66143783j]$, which meet our requirements.

However...what happens if one of the parameters changes? For example, the spring might be sensitive to temperature variation:

- Hot: 0.9 K_{nominal}
- Cold: 1.1 K_{nominal}

And we can again use the Root Locus method, but this time using:

$$\frac{Z(s)}{F(s)} = \frac{1}{s^2 + 1.5s + k}$$

for $K \in [0.9k, 1.1k]$

```
In [ ]: fig, axs = plt.subplots(1,1, figsize=(10,5))

poles = []
k_range = np.arange(0.9, 1.1, 0.001)
for k in k_range:
    poles.append(np.roots([1, 1.5, k]))

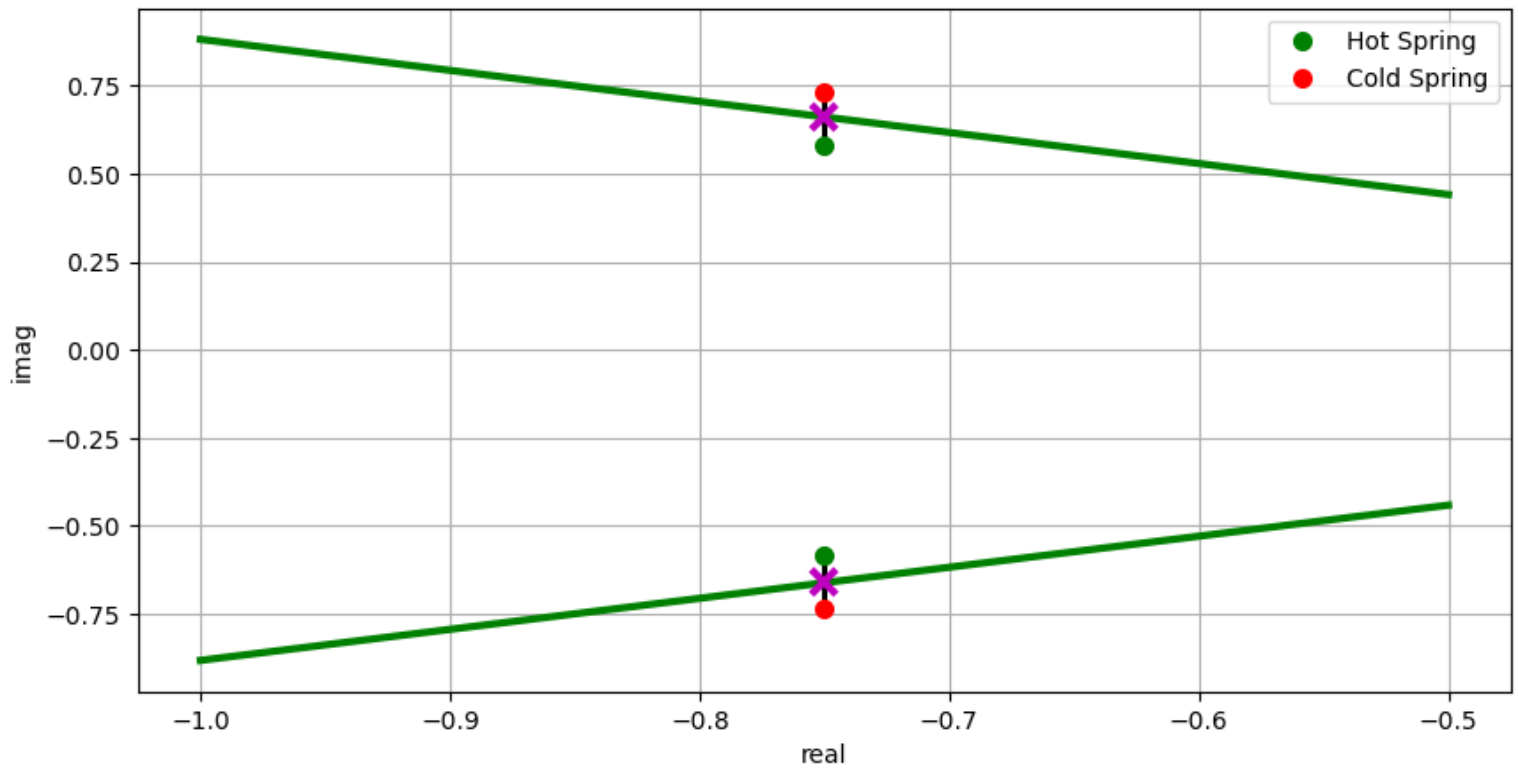
plt.plot(np.real(poles), np.imag(poles), marker='.', markersize=1, linestyle='', color='k',
plt.plot(np.real(poles[0]), np.imag(poles[0]), marker='.', color='g', markeredgewidth=5, li
plt.plot(np.real(poles[-1]), np.imag(poles[-1]), marker='.', color='r', markeredgewidth=5,
plt.legend()
# desired damping lines
phi = np.arccos(0.75)
xi = 0.75
tan_theta = np.sqrt(1-xi**2)/xi

x_range = np.linspace(-1, -0.5)
plt.plot(x_range, -tan_theta*x_range, linewidth=3, color='g')
plt.plot(x_range, tan_theta*x_range, linewidth=3, color='g')

# Choose b = 1.5
b = 1.5
poles_desired = np.roots([1, b, 1])
print(poles_desired)
plt.plot(np.real(poles_desired), np.imag(poles_desired),
         marker='x', markersize=10, linestyle='', color='m', markeredgewidth=3)

plt.grid()
plt.xlabel('real')
plt.ylabel('imag');
```

$[-0.75+0.66143783j \quad -0.75-0.66143783j]$



- When the spring is cold we do not meet our requirements!
- We might need to find a way to restrict the temperature variation, redesign our system or have a warning that performance might degrade depending on temperature. Depends on the application.

Sketching the Root Locus

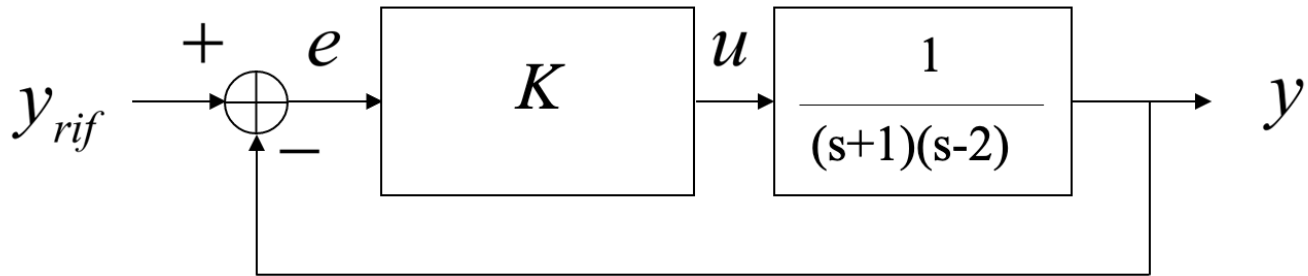
- As control engineers, you might not use an exact sketch in practise (Matlab or the Python Control Library will help you!), but most often than not, you will use an approximate one!
- With practice you will be able to look at the open loop zeros and poles of the system and visualise how the closed loop RL looks like.
- Practise makes perfect!

Let's take an example:

$$G(s) = \frac{1}{(s+1)(s-2)}$$

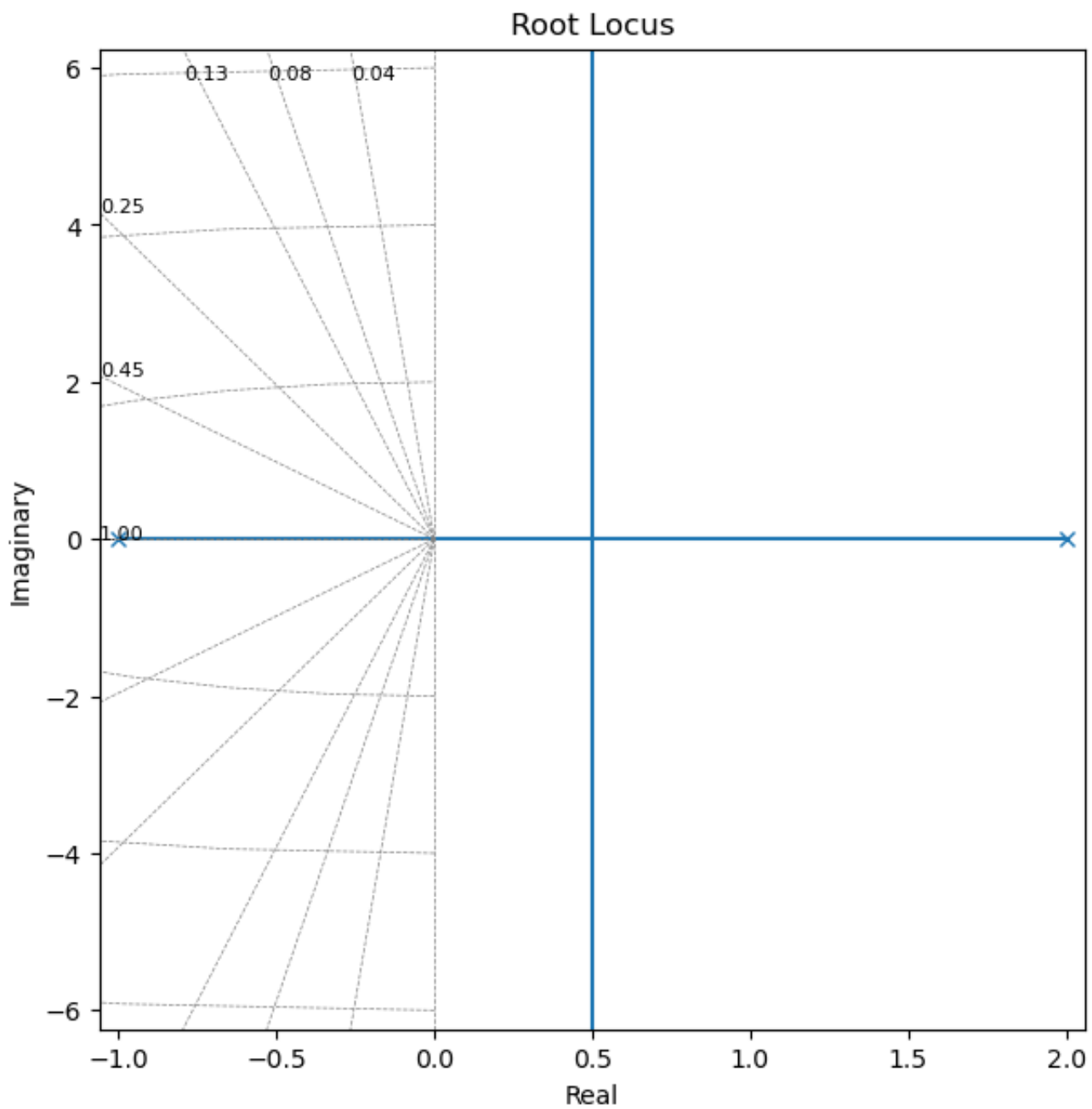
The system is clearly unstable and we need to design a controller to make the closed loop system stable.

One option might be that of using a proportional controller:



- If we plot how the poles change when K changes however, we realise that one root is always in the right half plane.
- Moreover, the root that starts at -1 also goes unstable.

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1],[1, -1, -2]))
```

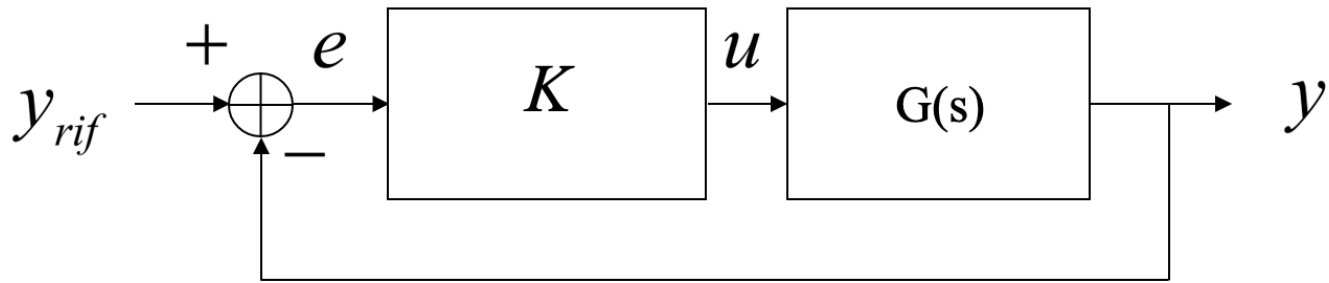


- This problem cannot be solved with a proportional controller only

Design question:

- How do we place poles and zeros in the feedback path so that all poles are in the left half plane?
- Understanding how to place the poles/zeros come from experience drawing the Root Locus

Classic Set up of the Root Locus Problem:

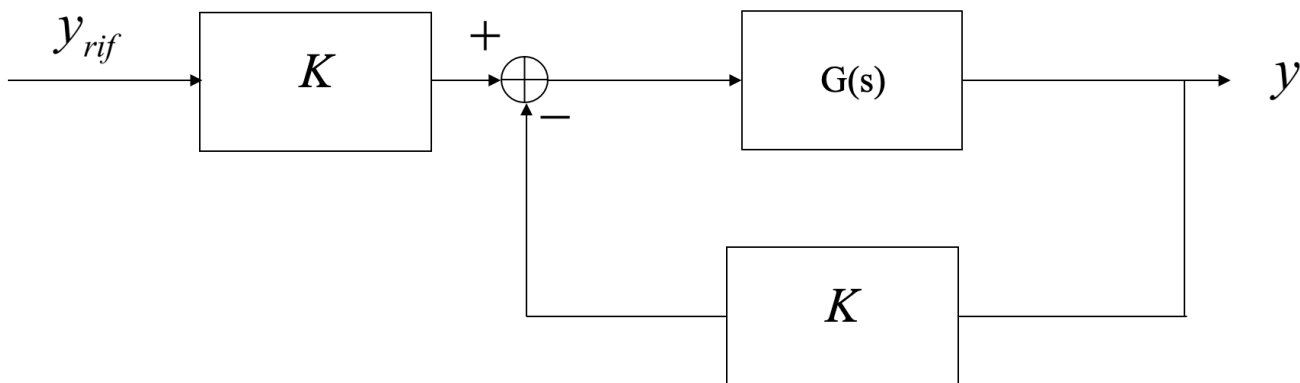


- Gain in the forward path
- Unity feedback

The closed transfer function is:

$$G_{cc} = \frac{KG(s)}{1 + KG(s)}$$

- Given that we are interested in finding the poles of the closed loop system we only need:
 $1 + KG(s) = 0$
- This is the form of the characteristic equation that we need in order to apply the rules that we are going to discuss.
- **Note:** Matlab might draw the root locus starting from a slightly different form, and considers the gain in the feedback path. Note that the denominator does not change (the closed loop system will behave differently to forced inputs but this is not important for our objectives when we plot the root locus). To have the Matlab representation the same, we need to multiply the input by K .
- From a stability perspective there is no difference



- **Note:** Before we analysed

$$G(s) = \frac{s^2 + s + 1}{s^3 + 4s^2 + Ks + 1}$$

which is not in the form that we would like.

- We can translate it back to the form we need.
- We are looking for the roots of the characteristic equation and how they move as we change K .

Steps:

- Write the characteristic equation: $s^3 + 4s^2 + Ks + 1 = 0$
- Group all K terms: $s^3 + 4s^2 + 1 + (Ks) = 0$
- Divide by the non- K terms to obtain the form we need:

$$s^3 + 4s^2 + 1 + (Ks) = 0 \Rightarrow \frac{s^3 + 4s^2 + 1}{s^3 + 4s^2 + 1} + K \frac{s}{s^3 + 4s^2 + 1} = 1 + K \frac{s}{s^3 + 4s^2 + 1}$$

- Note that we are now considering a new $G(s) = \frac{s}{s^3 + 4s^2 + 1}$
- We can use the Root Locus to analyse how any change in one of the parameters affect the results, not only the gain in the forward path. *It is really powerful!*

How to sketch the Root Locus: Drawing Rules

- We use 10 drawing rules

Start from:

$$1 + KG(s) = 0 \Rightarrow 1 + K \frac{Q(s)}{P(s)} = 0$$

- We have re-written $G(s)$ as a ratio of two polynomials
- Always start from this form.

RULE 1

There are n lines (loci) in the Root Locus, where n the maximum between the degree of the denominator and of the numerator.

- We have as many lines as the number of poles or zeros (whichever is greater)
- e.g. 3 poles and 2 zeros: 3 lines

RULE 2

As K increases from 0 to ∞ , the roots move from the poles of $G(s)$ to the zeros of $G(s)$.

Let's unpick this rule a bit and start from:

$$1 + K \frac{Q(s)}{P(s)} = 0 \Rightarrow P(s) + KQ(s) = 0$$

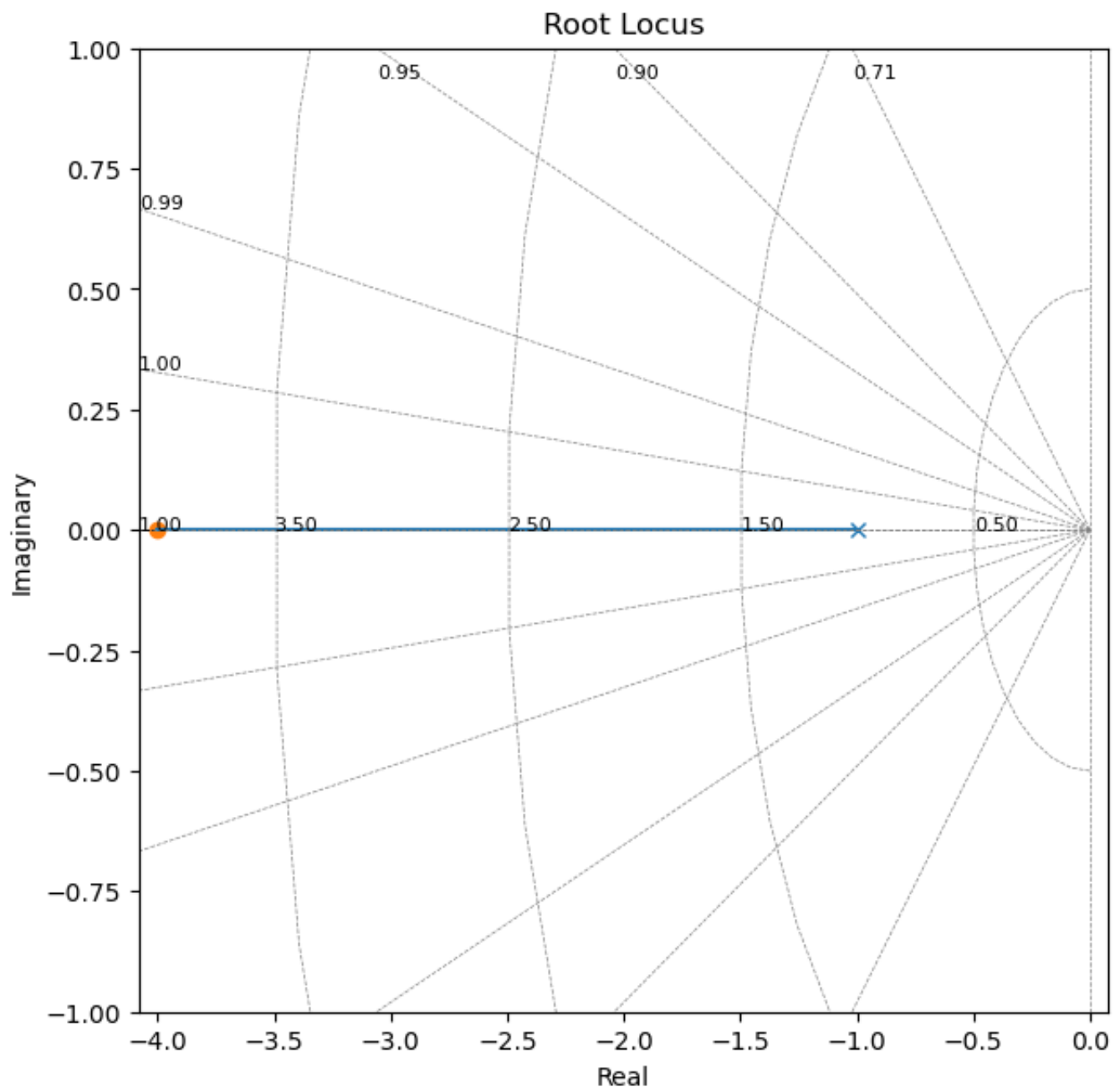
- Poles are when $P(s) = 0$
- Zeros are when $Q(s) = 0$

When $K = 0$, the $P(s) + KQ(s) = 0$ we are left with $P(s) = 0$. So the poles of the closed loop system start at the poles of $G(s)$.

When $K \rightarrow \infty$, we put more importance to $Q(s)$, which dominates our characteristic equation: the root locus ends at the zeros of $G(s)$.

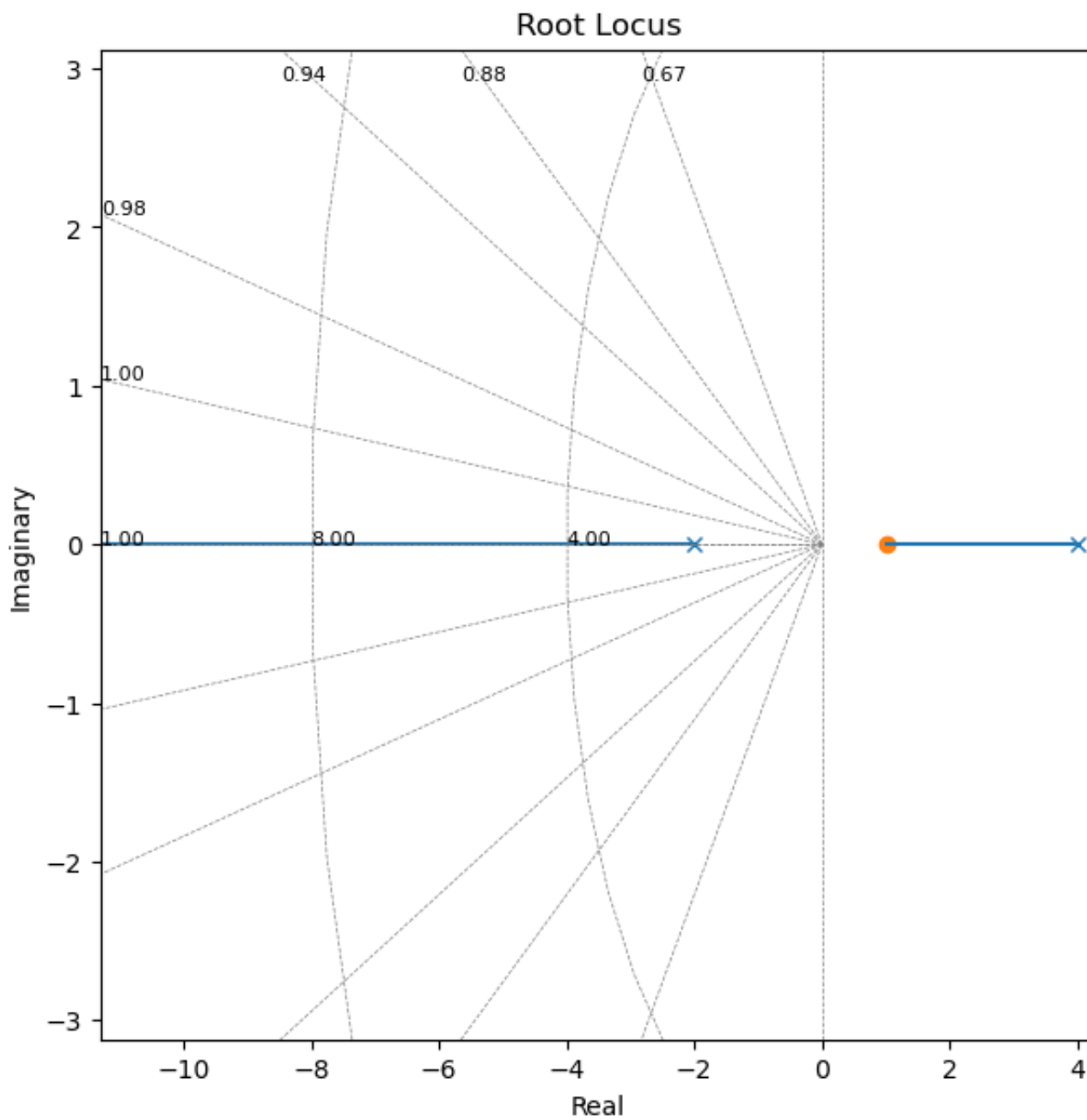
- Rule 2 says: As K increases, the closed loop poles move from the open loop poles to the open loop zeros.
- What happens when we have a different number of poles and zeros?
- **case 1:** If $\text{order}(Q(s)) = \text{order}(P(s))$: each have a pair, and the roots travel from one pole to a zero

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
        [rlist, klist] = control.root_locus(control.tf([1, 4],[1, 1]))
```

- **case 2:** If $\text{order}(P(s)) > \text{order}(Q(s))$, the lines from the extra poles would go to infinity.

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1, -1],[1, -2, -8]))
```

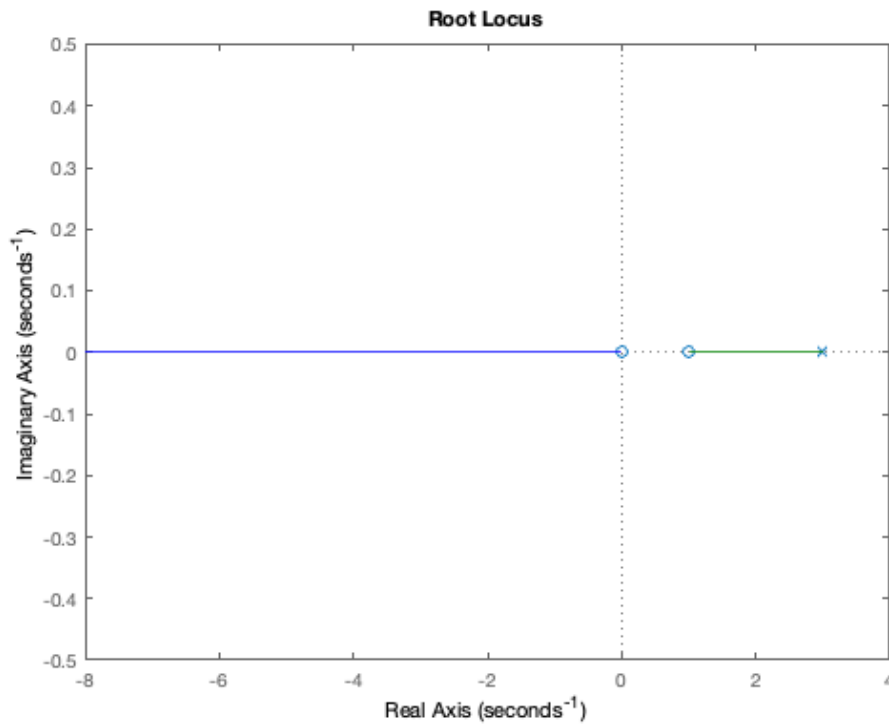


- **case 3:** If $\text{order}(Q(s)) > \text{order}(P(s))$, the extra lines come from infinity.

For example:

$$G(s) = \frac{s^2 - s}{s - 3}$$

```
In [ ]: # [rlist, klist] = control.root_locus(control.tf([1, -1, 0], [1, -3]))
# For some reason python control lib does not plot it..
# The figure below is done with Matlab:
# >> rlocus(tf([1, -1, 0], [1, -3]))
```



RULE 3

When roots are complex they occur in conjugate pairs (we saw this already multiple times! - it is due to having real systems) and they move in pairs.

- The Root Locus is symmetric around the real line

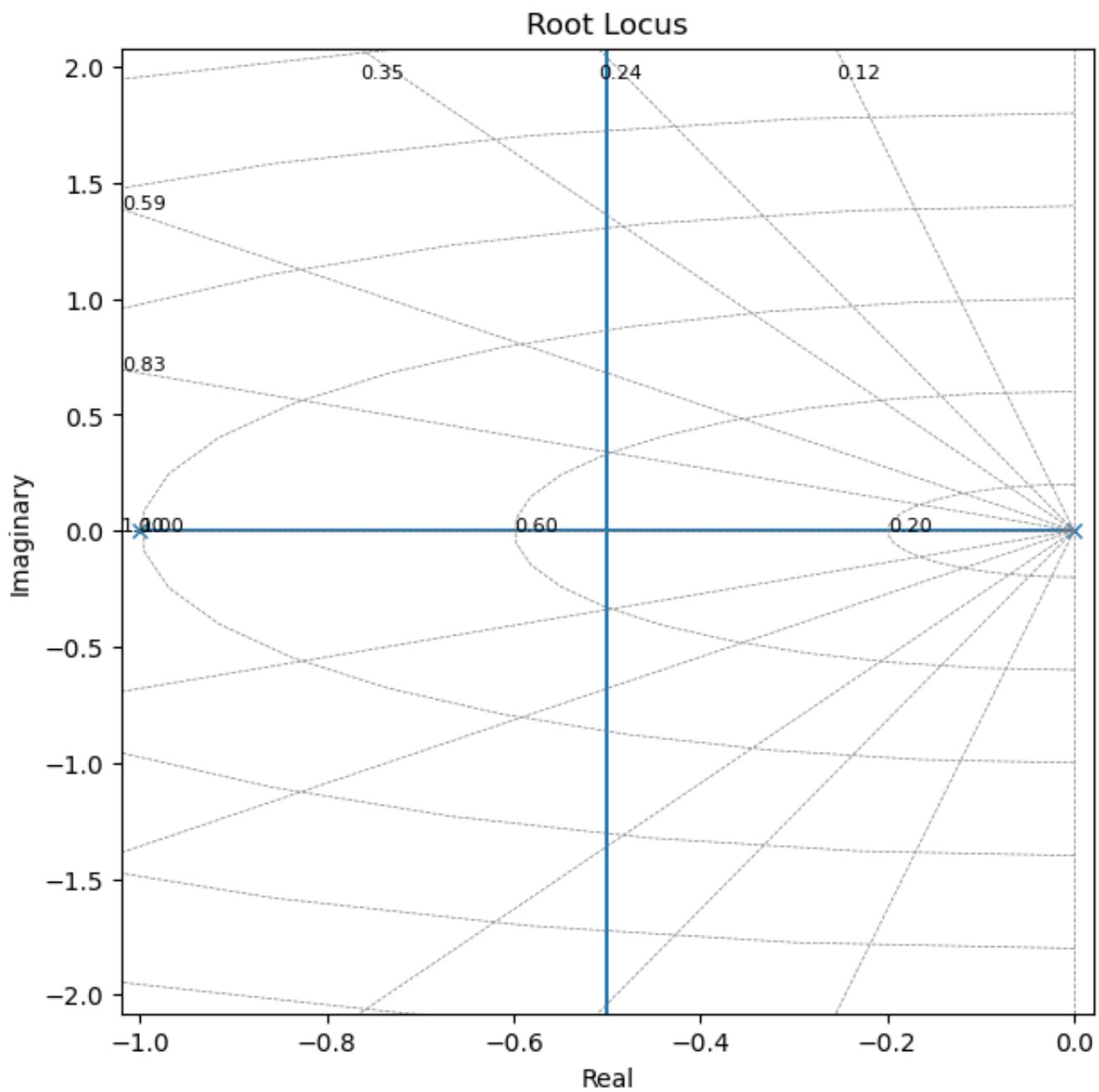
RULE 4

At no time will the same root cross over its path (no loop): no value of s will correspond to more than one value of K .

- Note that paths of two different roots are allowed to intersect.

For example if we have poles in $s = 0$ and $s = 1$:

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1], [1, 1, 0]))
```



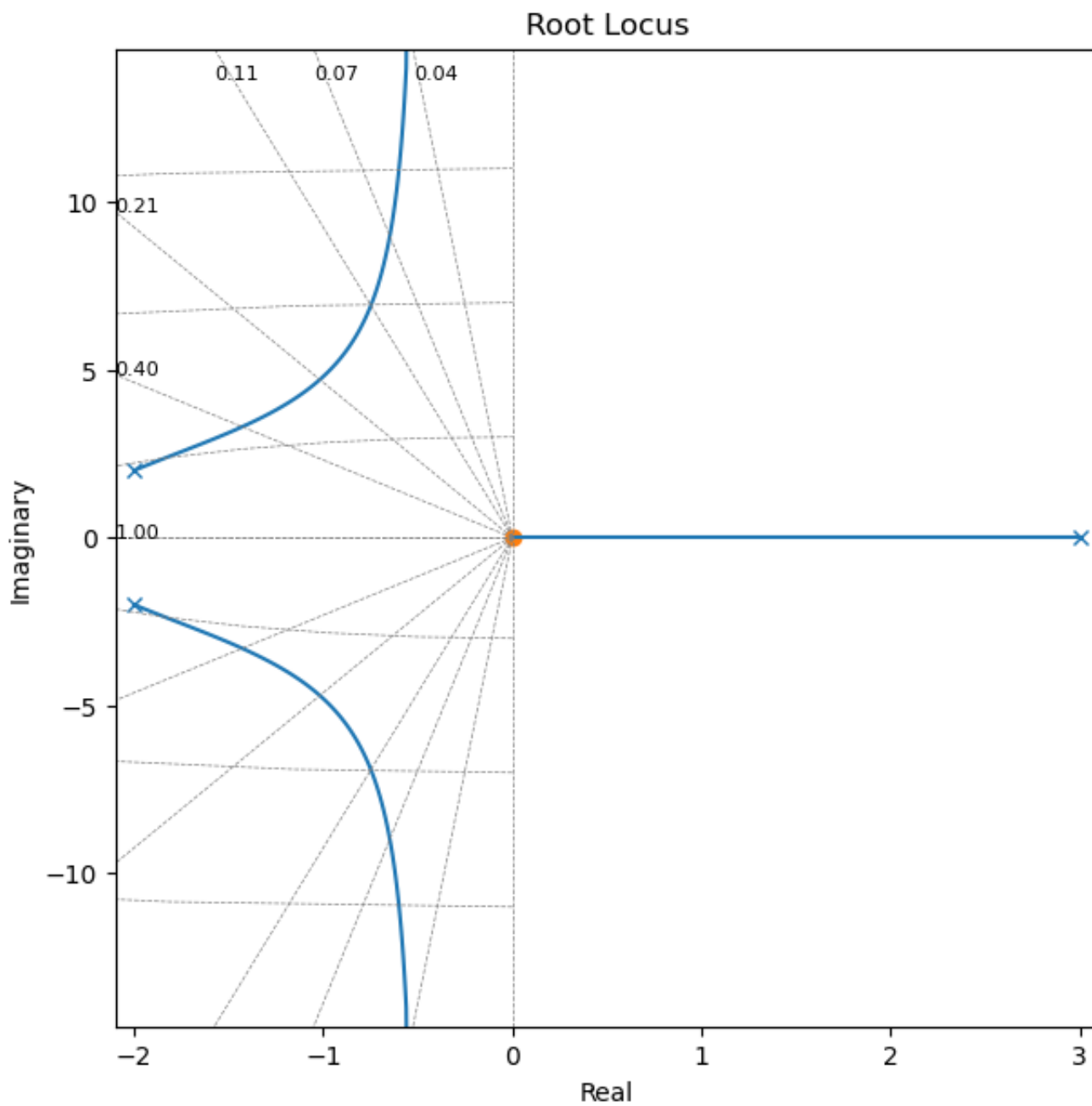
Using the first 4 rules - example:

$$G(s) = \frac{s}{(s - 3)(s + 2 - 2j)(s + 2 + 2j)}$$

- We have 3 poles, so we have 3 lines
- 3 poles, and 1 zeros: 2 lines go to infinity
- The two complex conjugate roots move in pairs (mirrored about the real axis).
- Each root will not have a path that creates loop

Let's now plot the Root Locus with the Python Control Library

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1, 0],[1, 1, -4, -24]))
```



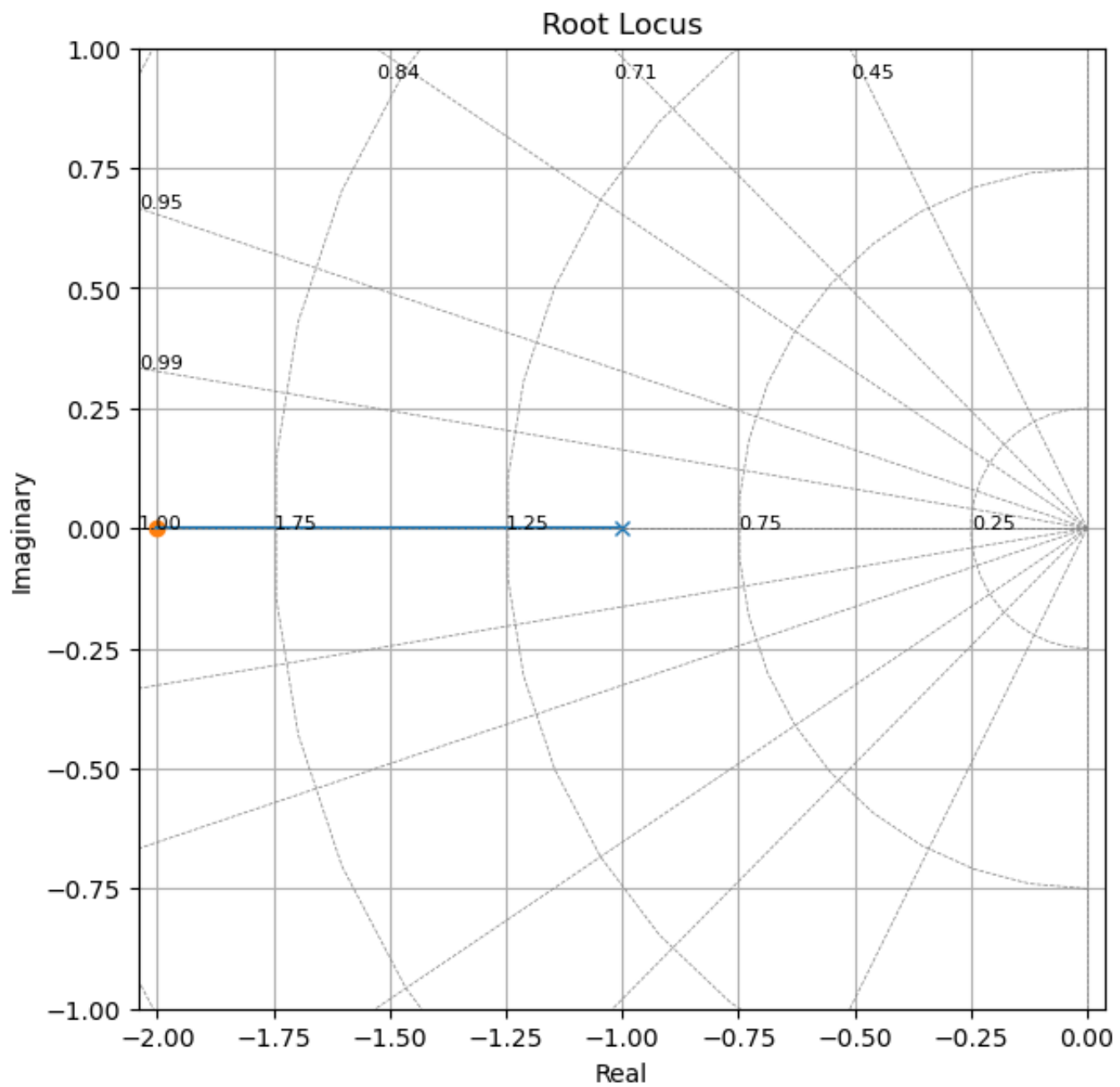
We can check our roots:

```
In [ ]: np.roots([1, 1, -4, -24])
Out[ ]: array([-2.+2.j, -2.-2.j,  3.+0.j])
```

RULE 5

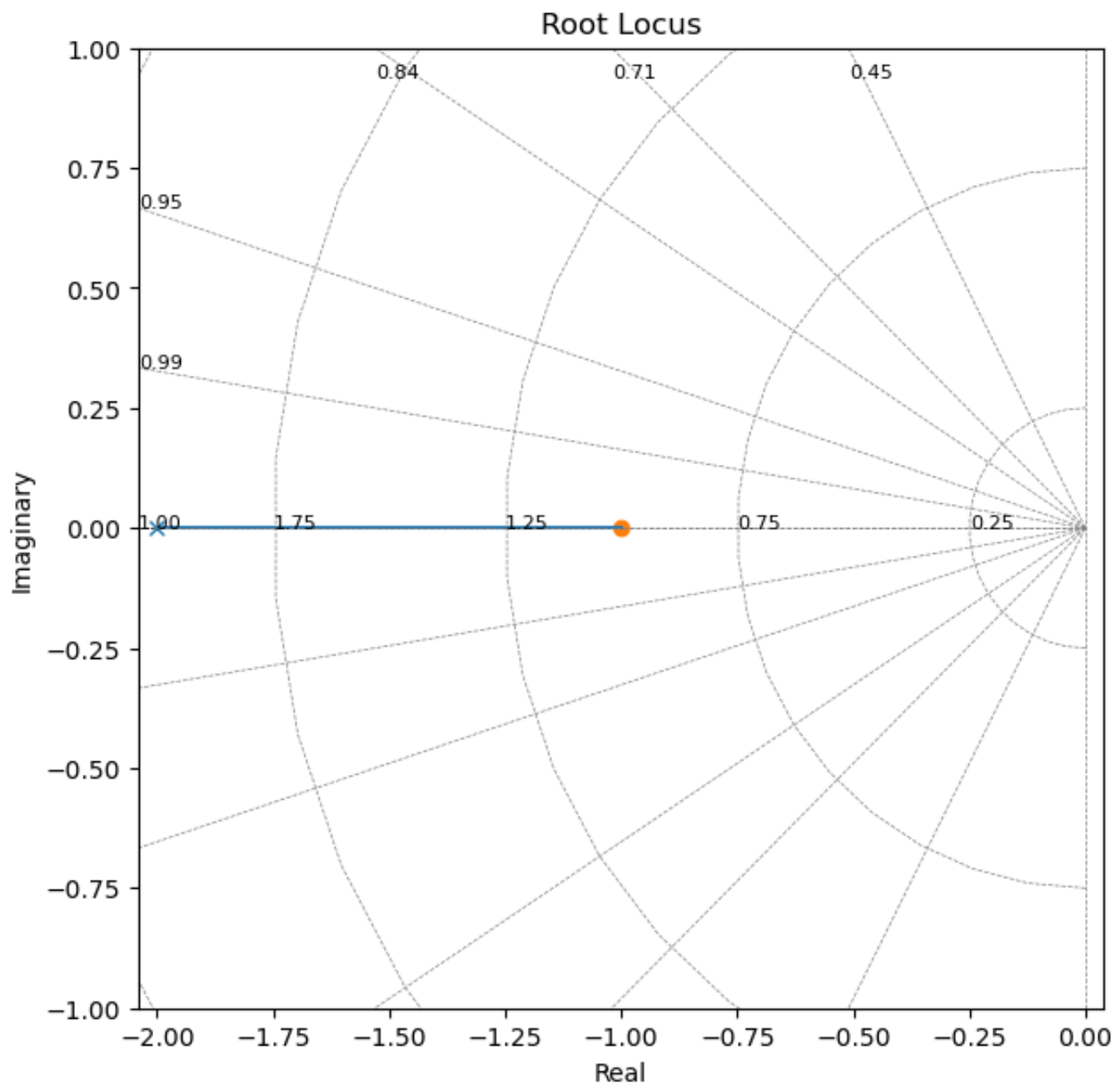
The portion of the real axis to the left of an odd number of open loop poles and zeros (critical frequencies) are part of the loci.

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1, +2],[1, 1]))
plt.grid()
```



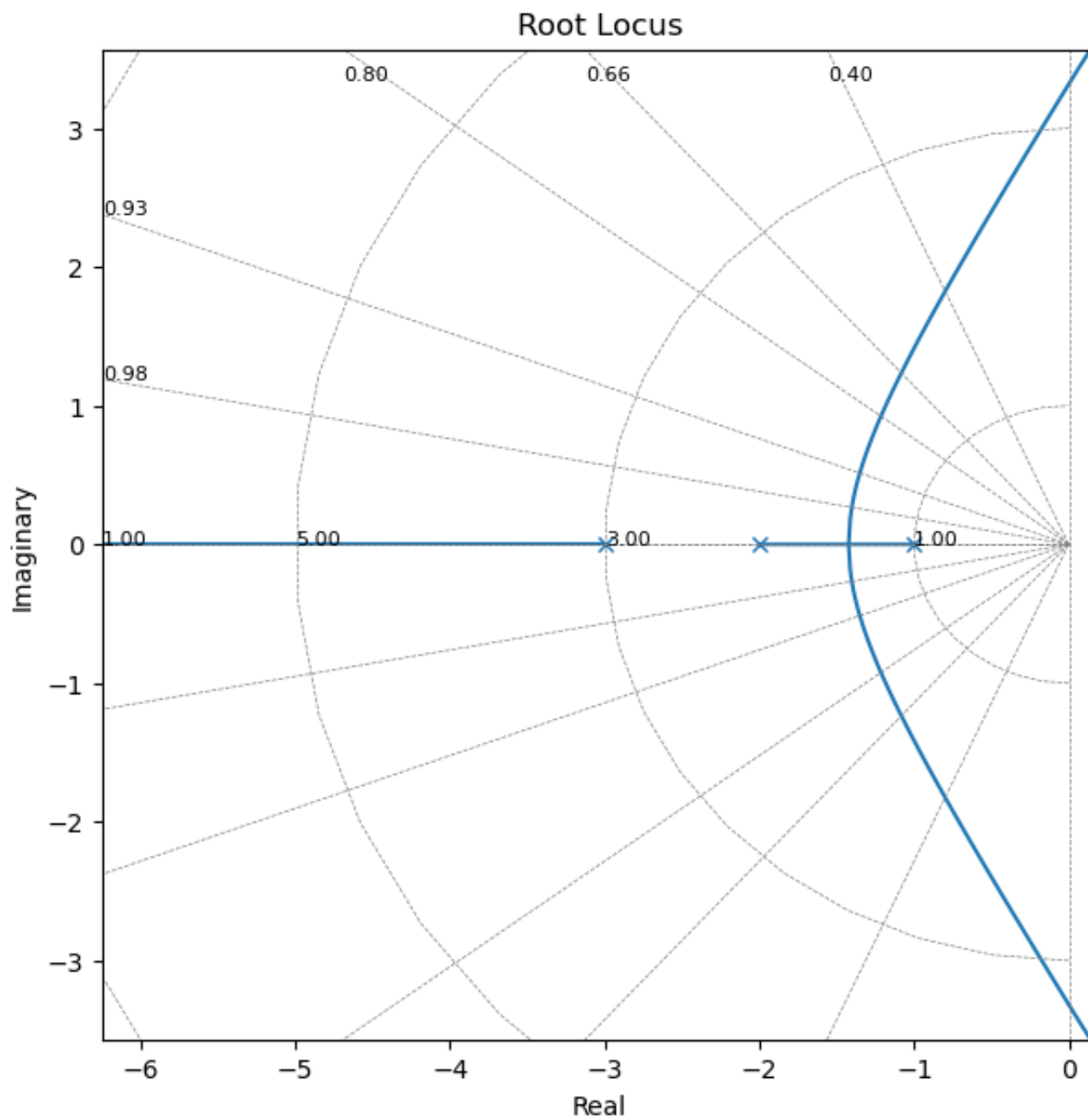
Note that if we swap the location of the pole and zero, the root locus is still between them, but its direction is reversed: it is always from the open loop pole to the open loop zero.

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1, 1],[1, 2]))
```



If we have three poles:

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1],[1, 6, 11, 6]))
```



Note that:

- To the left of the third pole, the root locus goes to infinity
- Between the first and the second critical frequencies the lines come together and then they go off the real axis. We will see in a minute where they go.

RULE 6

Lines leave (or break out) and enter (break in) the real line at 90° (in search of an open loop zero)

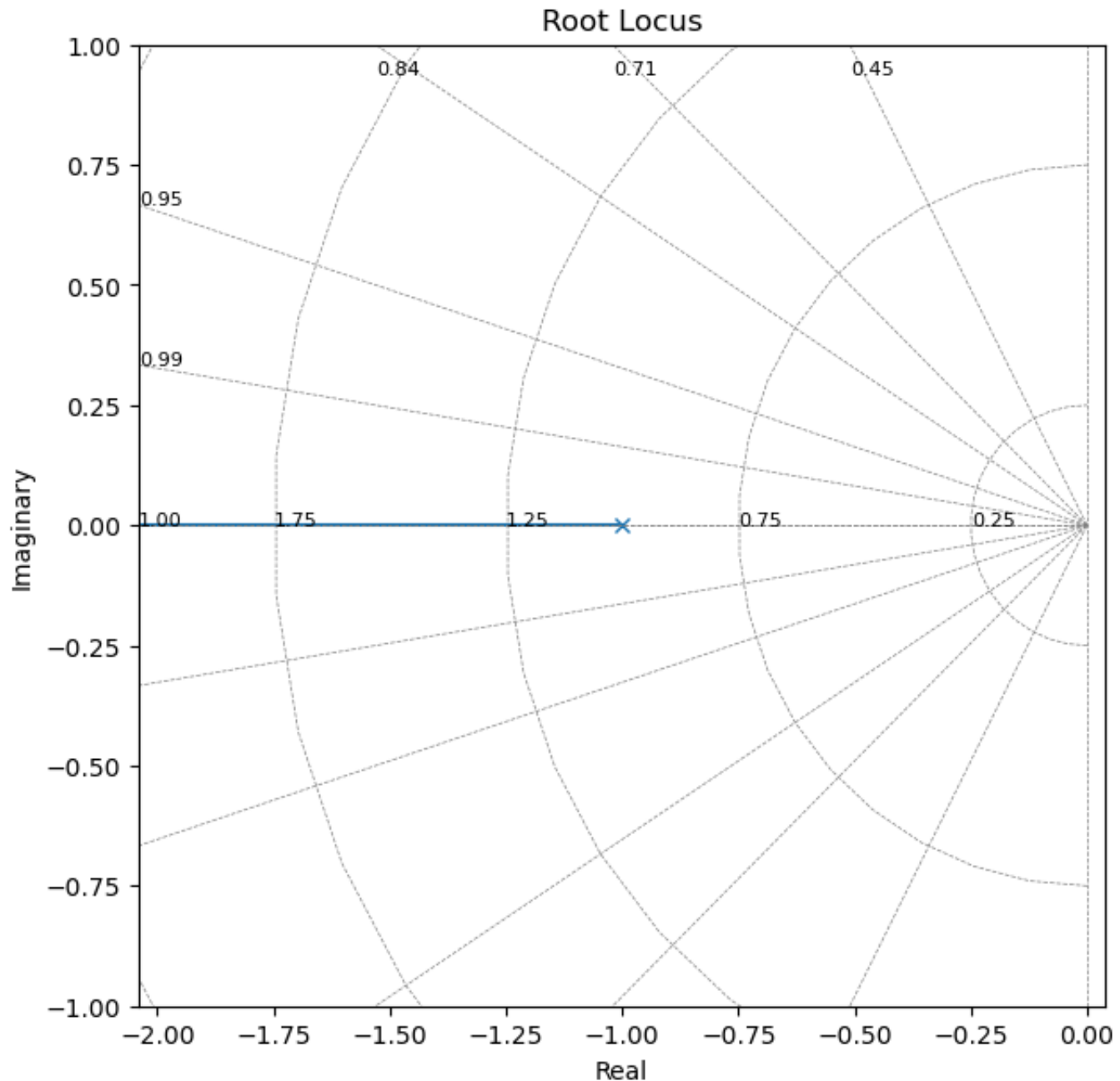
RULE 7

If there are not enough poles or zeros to make a pair then the extra lines go to or come from infinity

For example:

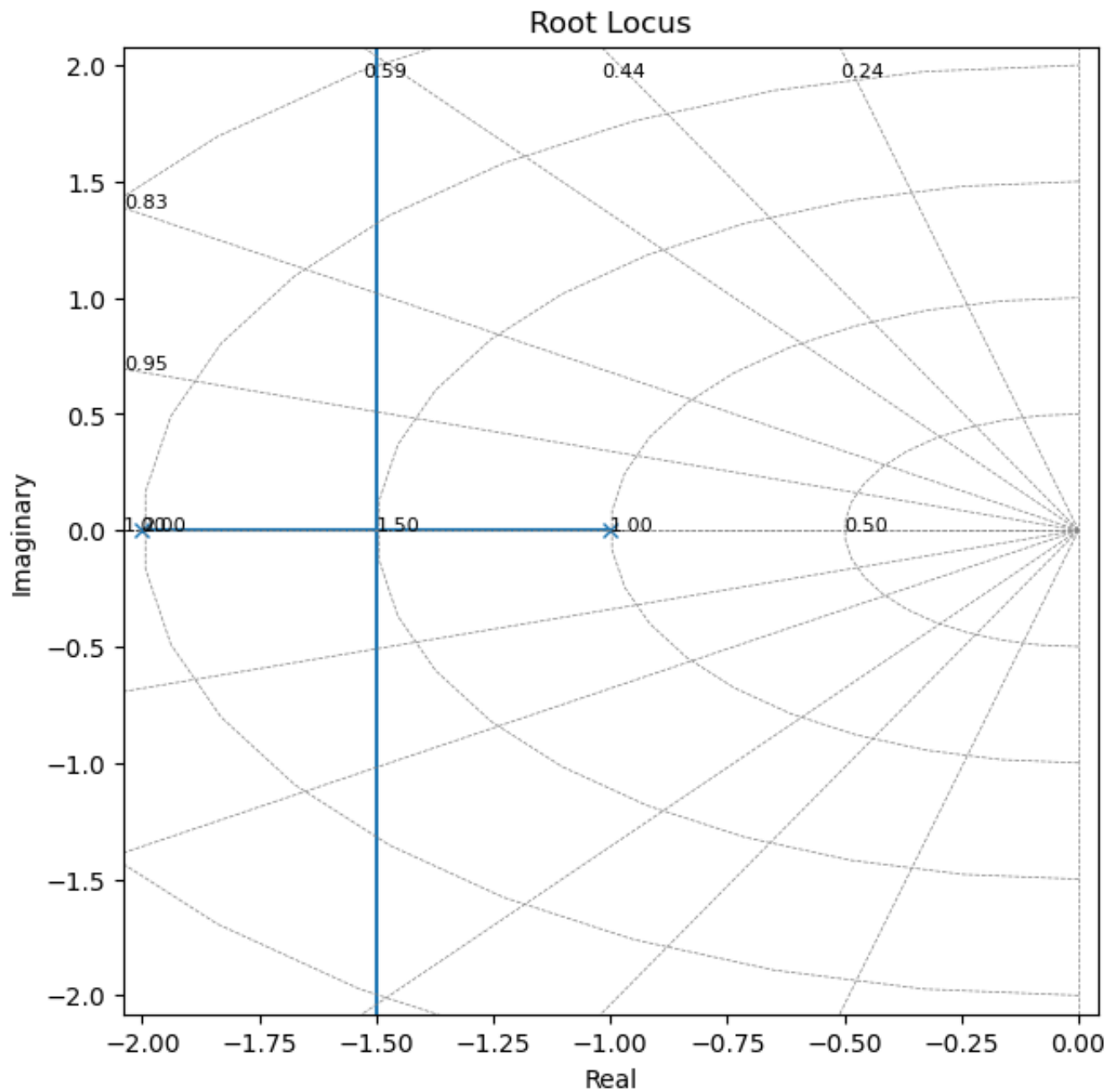
- One pole, no zero: **1 extra pole**: the left side of the pole is part of the locus and it goes to infinity

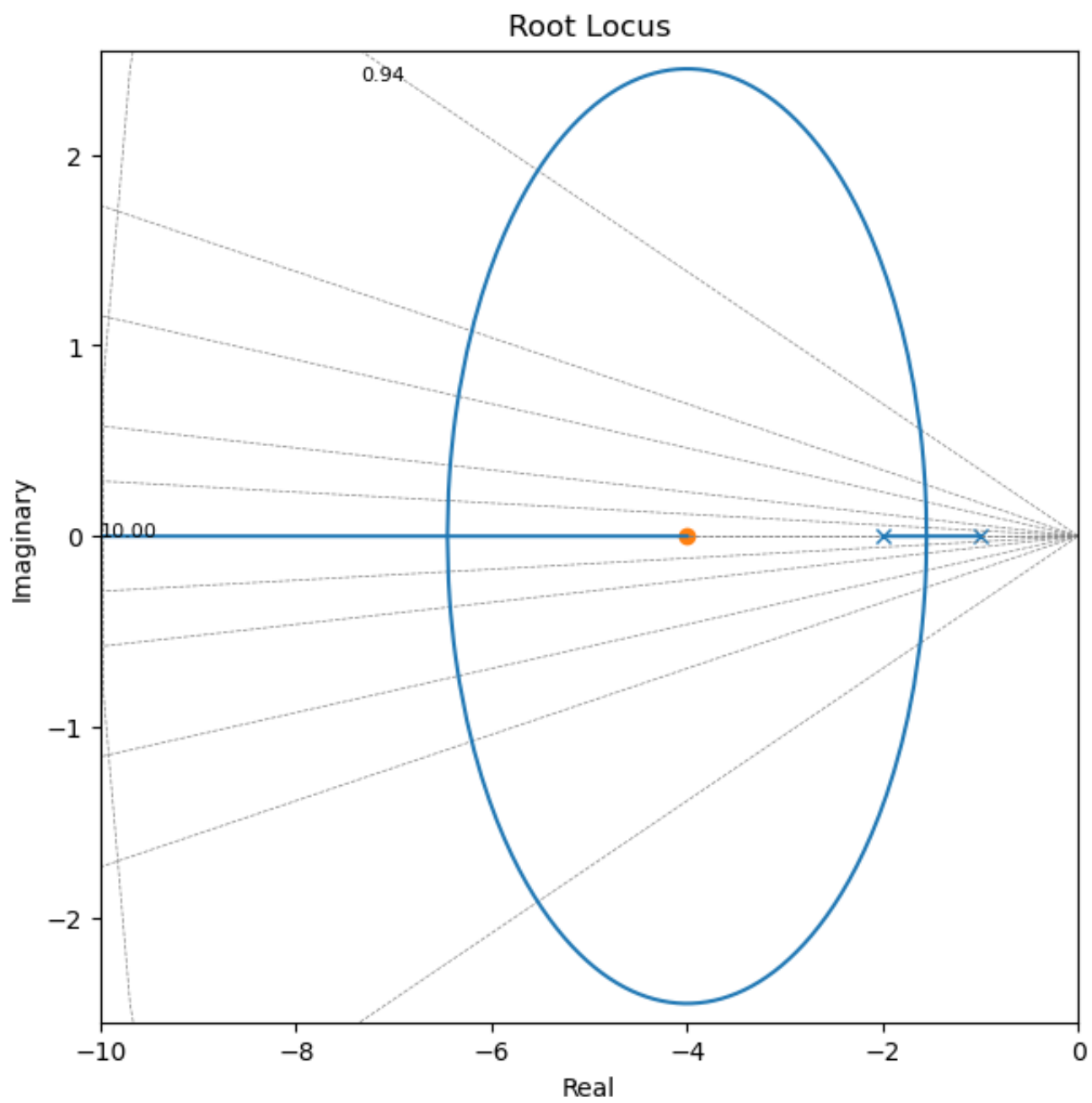
```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))  
[rlist, klist] = control.root_locus(control.tf([1],[1, 1]))
```



- Two poles and no zeros, **2 extra poles**: the left side of an odd number of poles in part of the locus and they leave the real axis at 90 deg towards infinity

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1],[1, 3, 2]))
```





- What happens when we have two extra zeros?
- Two lines would be coming in *from* infinity

Given that we have lines going to (or coming from) infinity, at what angles do they do that?

RULE 8

Lines go to infinity along asymptotes

- We have as many asymptotes as many unmatched pole-zero pairs ($n - m$)
- The angles of the asymptotes is (in deg):

$$\Phi_A = \frac{2q + 1}{n - m} \cdot 180$$

Where $n - m$ is the number of unmatched pairs (#poles-#zeros), and $q = 0, 1, 2, \dots, (n - m - 1)$

- The centroid of the asymptotes is:

$$C = \frac{\sum \text{Finite poles} - \sum \text{Finite zeros}}{n - m}$$

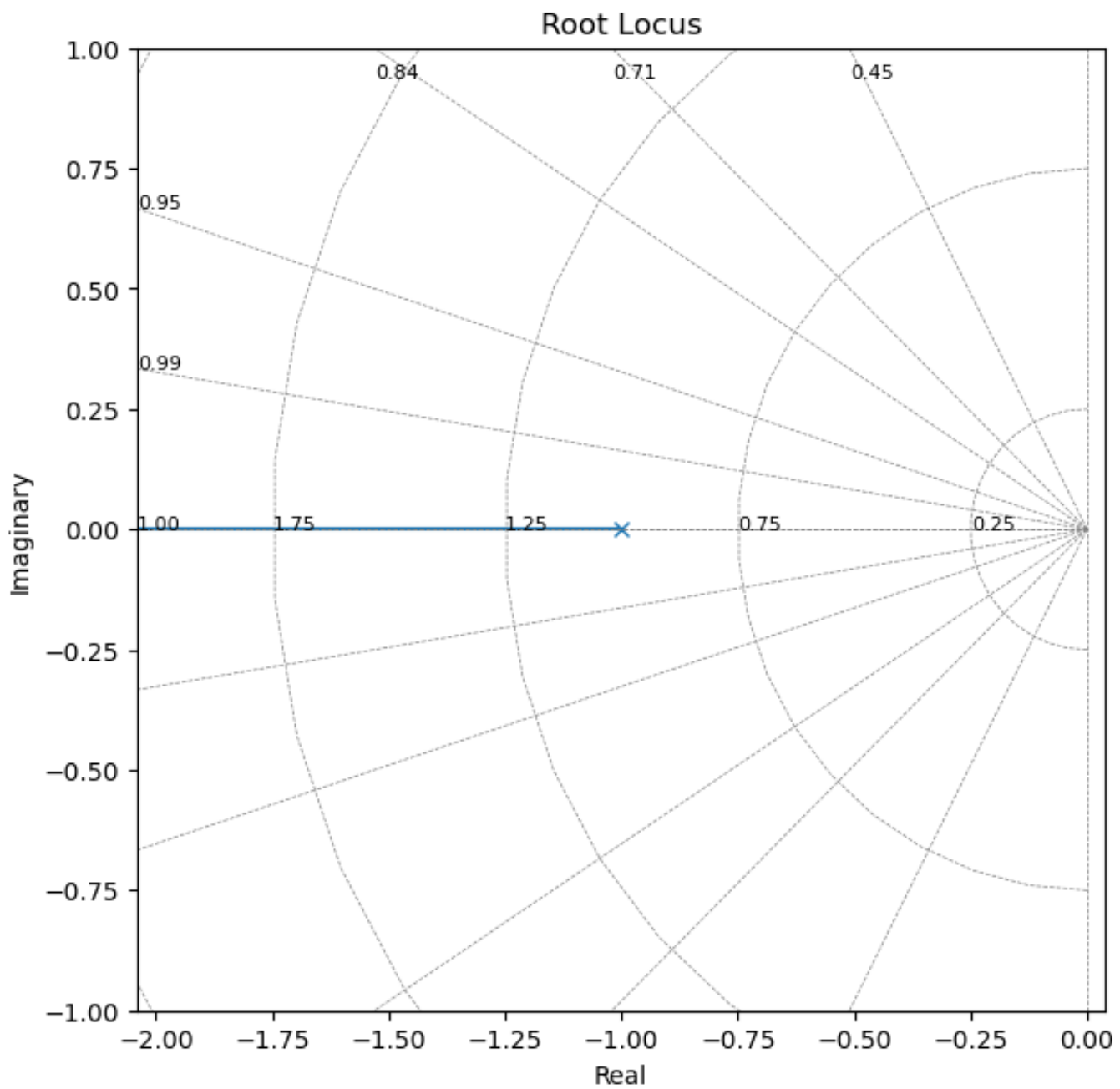
- This is the point on the real axis where all asymptotes come together.

For example:

- $n - m = 1$ (1 line that goes to infinity)

$$\Phi_A = \frac{2 \cdot 0 + 1}{1} \cdot 180 = 180$$

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1],[1, 1]))
```



For example:

- $n - m = 2$ (2 lines that go to infinity)

$$\Phi_{A_0} = \frac{2 \cdot 0 + 1}{2} \cdot 180 = 90$$

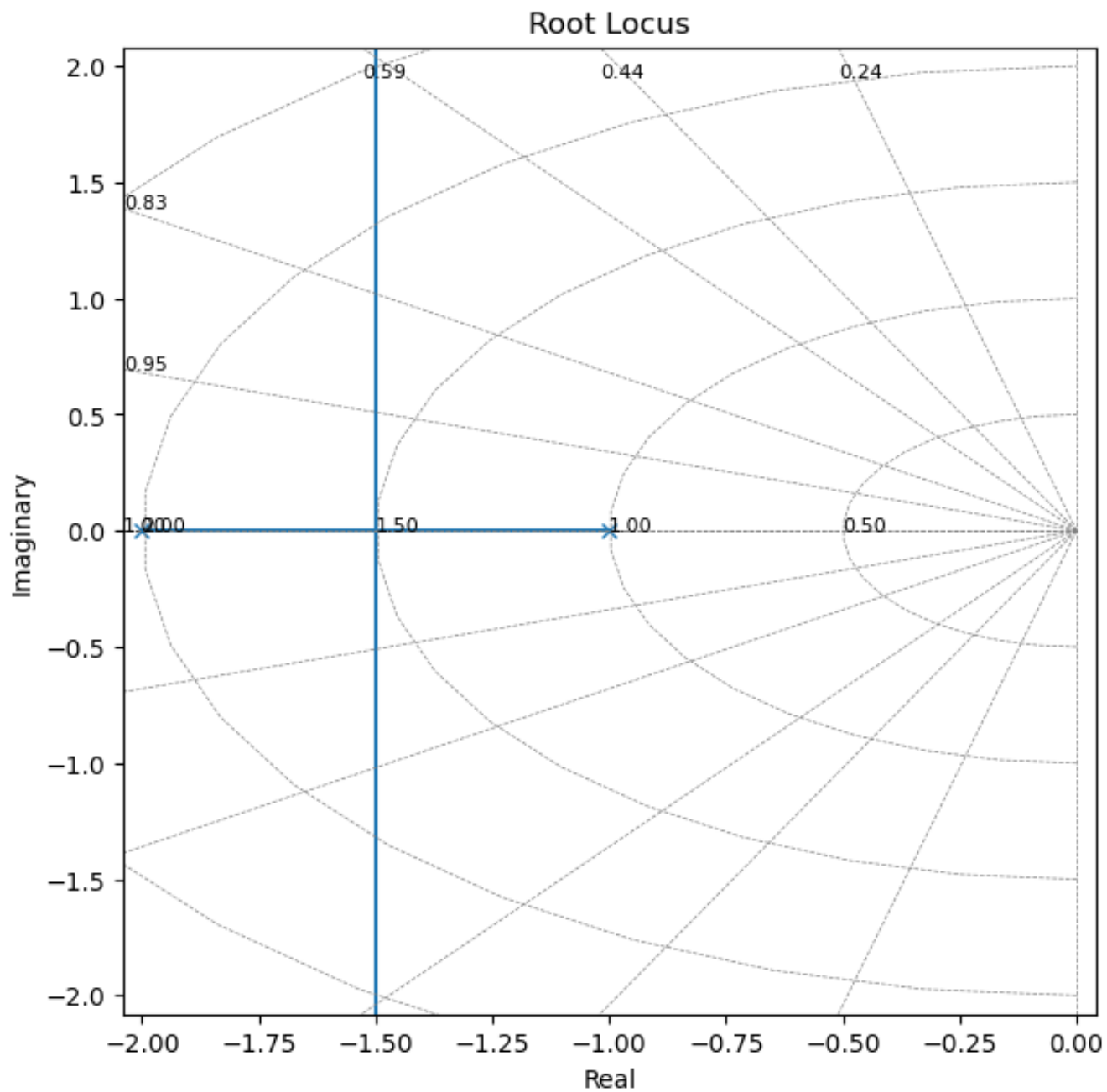
$$\Phi_{A_1} = \frac{2 \cdot 1 + 1}{2} \cdot 180 = 270 = -90$$

with centroid (this determines exactly where they go off to ± 90 deg):

$$C = \frac{\sum \text{Finite poles} - \sum \text{Finite zeros}}{n - m} = \frac{(-2 - 1) - 0}{2} = -1.5$$

- half way between the poles in this case where we only have two poles.

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1],[1, 3, 2]))
```



- What would you expect if we added one more pole and one more zero to the left of the -2 pole?

```
In [ ]: #!/cat answers/solution_root_locus_1
```

- $n - m = 3$ (3 lines that go to infinity)

$$G(s) = \frac{1}{(s+1)(s+2)(s+3)}$$

Centroid:

$$\frac{-3 - 2 - 1}{3} = -2$$

Angles:

$$\Phi_{A_0} = 60$$

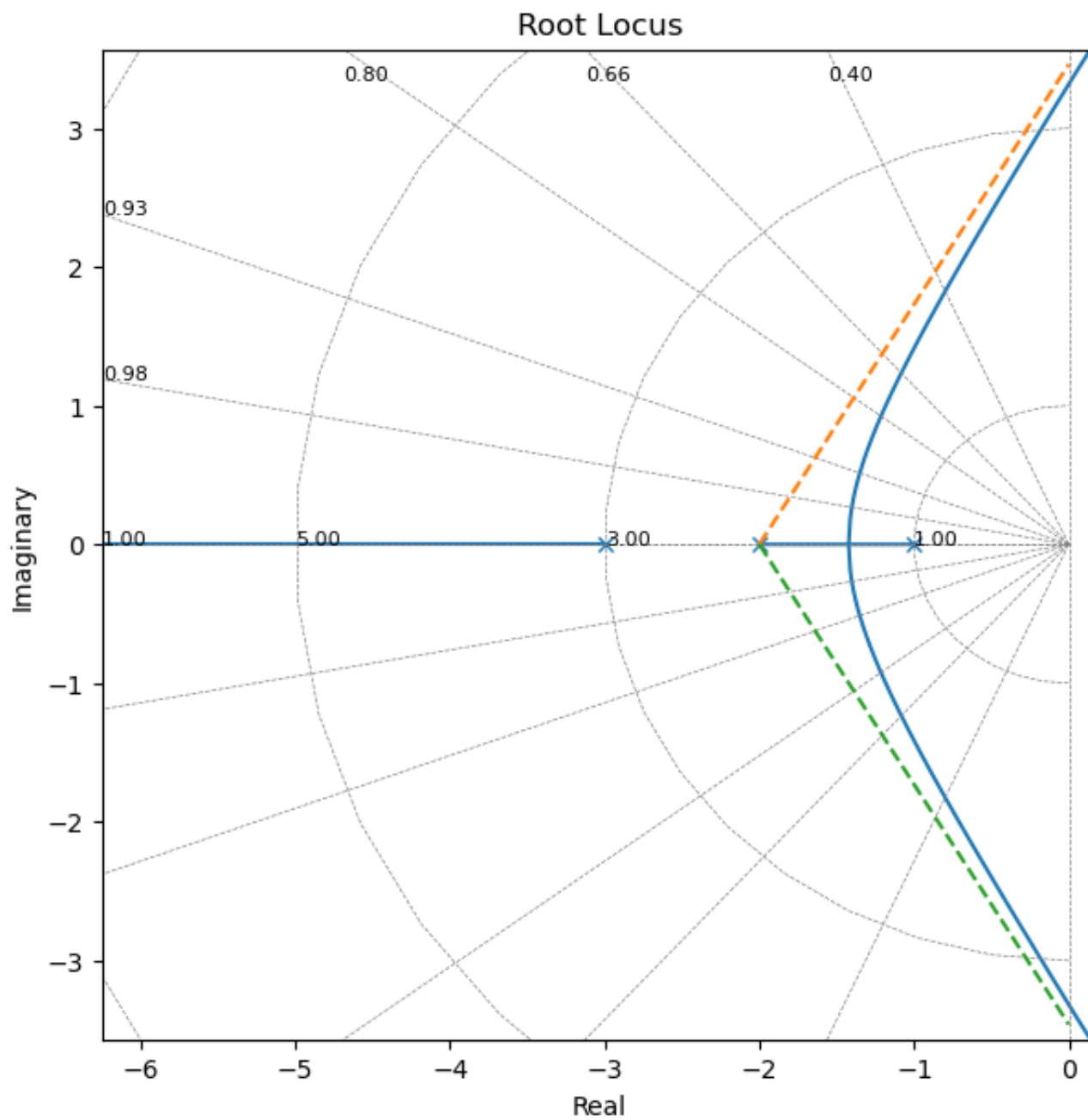
$$\Phi_{A_1} = 180$$

$$\Phi_{A_2} = -60$$

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))

[rlist, klist] = control.root_locus(control.tf([1],[1, 6, 11, 6]))

x_range = np.linspace(-2, 0);
plt.plot(x_range, np.tan(60/180*3.14)*x_range+np.tan(60/180*3.14)*(2), linestyle='--');
plt.plot(x_range, np.tan(-60/180*3.14)*x_range+np.tan(-60/180*3.14)*(2), linestyle='--');
```



RULE 9

If there is at least two lines to infinity, then the sum of all roots is constant

- This does not help sketching the roots but helps us with intuition

Suppose we have the system above:

$$G(s) = \frac{1}{(s+1)(s+2)(s+3)}$$

- The sum of the roots is -6 . This is the sum of the roots when $K = 0$ (the roots start at the open loop roots).
- When increase K the roots starts too move, but their sum must still be -6

For example in the picture below, we can plot how the roots move for $K = 0.29$

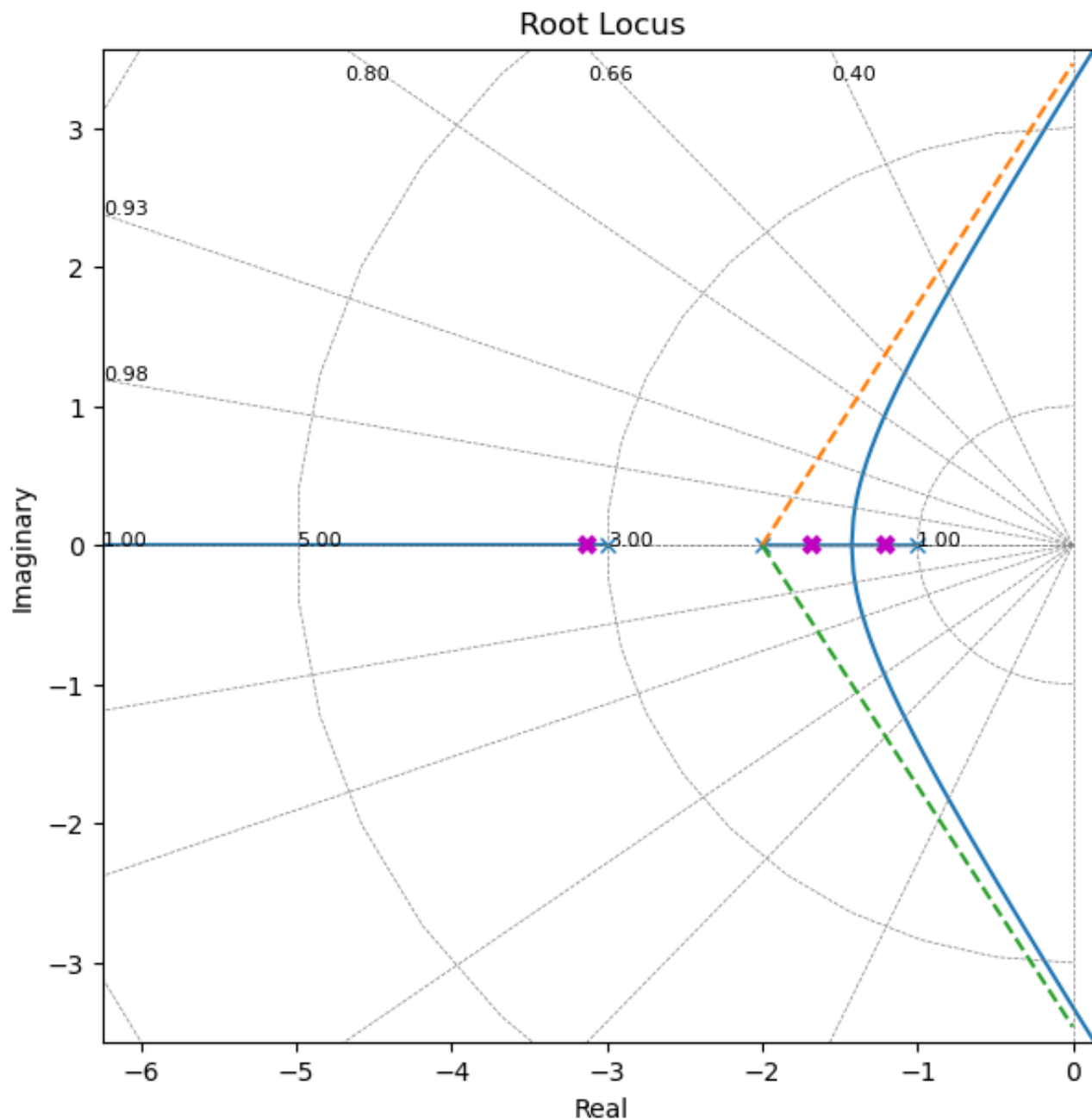
```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))

[rlist, klist] = control.root_locus(control.tf([1],[1, 6, 11, 6]))

x_range = np.linspace(-2, 0)
plt.plot(x_range, np.tan(60/180*3.14)*x_range+np.tan(60/180*3.14)*(2), linestyle='--')
plt.plot(x_range, np.tan(-60/180*3.14)*x_range+np.tan(-60/180*3.14)*(2), linestyle='--')

k_id=3
print('K=', klist[k_id])
plt.plot(np.real(rlist[k_id]), np.imag(rlist[k_id]), marker='x', color='m', linestyle='', m

K= 0.28867513459481264
Out[ ]: [<matplotlib.lines.Line2D>]
```



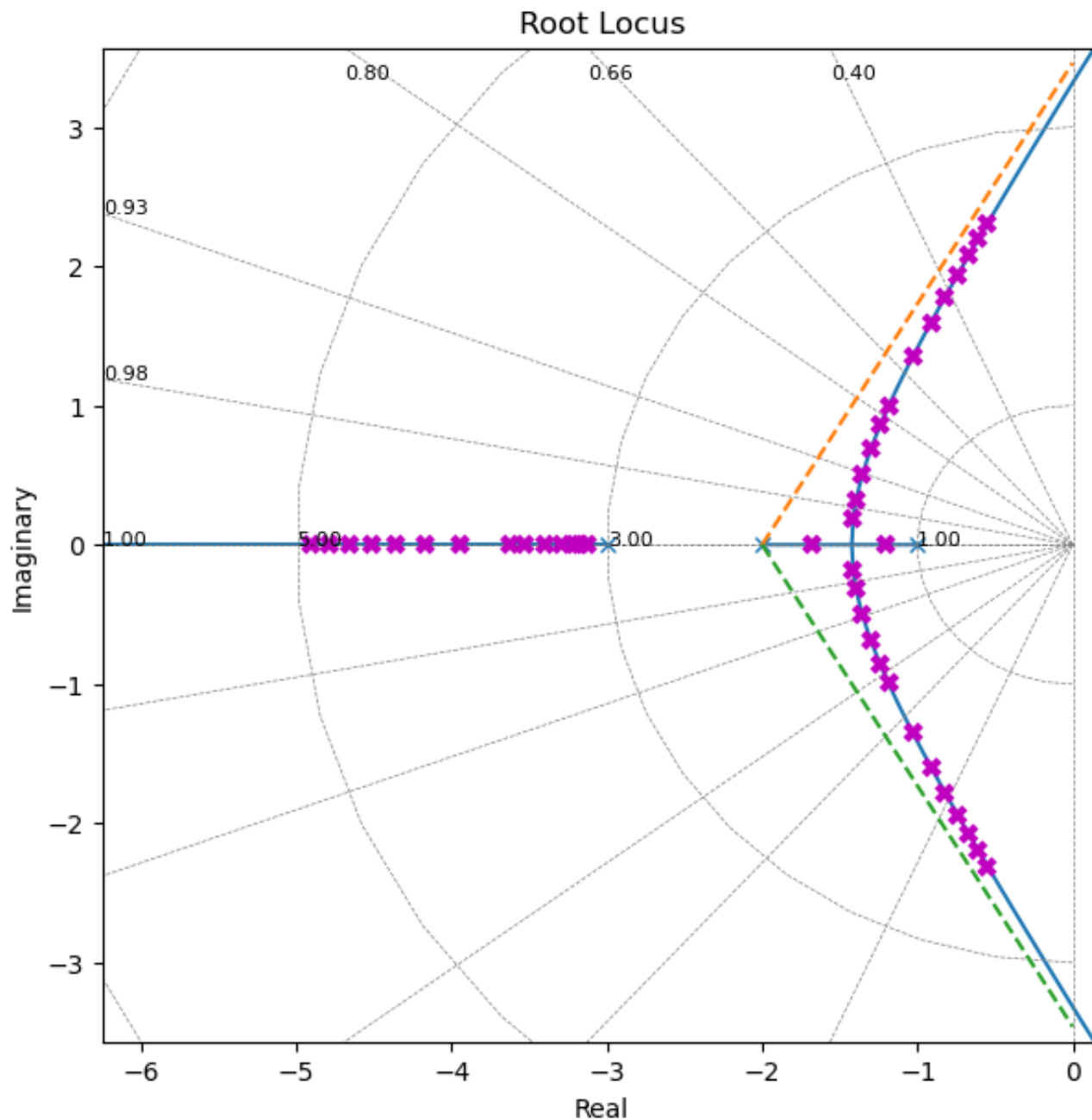
- What is important is that given that the sum is constant, we are destabilising the complex pair half as fast as how we move the real root
- This is useful when trying to decide the gain of our system

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))

[rlist, klist] = control.root_locus(control.tf([1],[1, 6, 11, 6]))

x_range = np.linspace(-2, 0)
plt.plot(x_range, np.tan(60/180*3.14)*x_range+np.tan(60/180*3.14)*(2), linestyle='--')
plt.plot(x_range, np.tan(-60/180*3.14)*x_range+np.tan(-60/180*3.14)*(2), linestyle='--')

for k_id in range(3, 30, 2):
    plt.plot(np.real(rlist[k_id]), np.imag(rlist[k_id]), marker='x', color='m', linestyle='')
```



- So far we have always considered $K \in [0, +\infty]$, let's see instead when we would like to analyse a negative gain.

RULE 10 - Negative K

The root locus when K goes from 0 to $-\infty$ can be obtained reversing RULE 5 and adding $+180^\circ$ to the asymptote angles

- For $K < 0$, the root locus is on the right of the odd critical frequencies

A few more comments

These set of rules only provide a sketch of the root locus. For example we have not discussed where exactly is on the real axis the break in and break away points or the angle of departure for complex roots, or where the locus crosses the imaginary axis, or what is the gain associated to one specific point on the locus.

- Use references to go deeper in the topic (see [00_Syllabus](#))
- Use software tool such as the Matlab SISO tool or the Python Control Library.

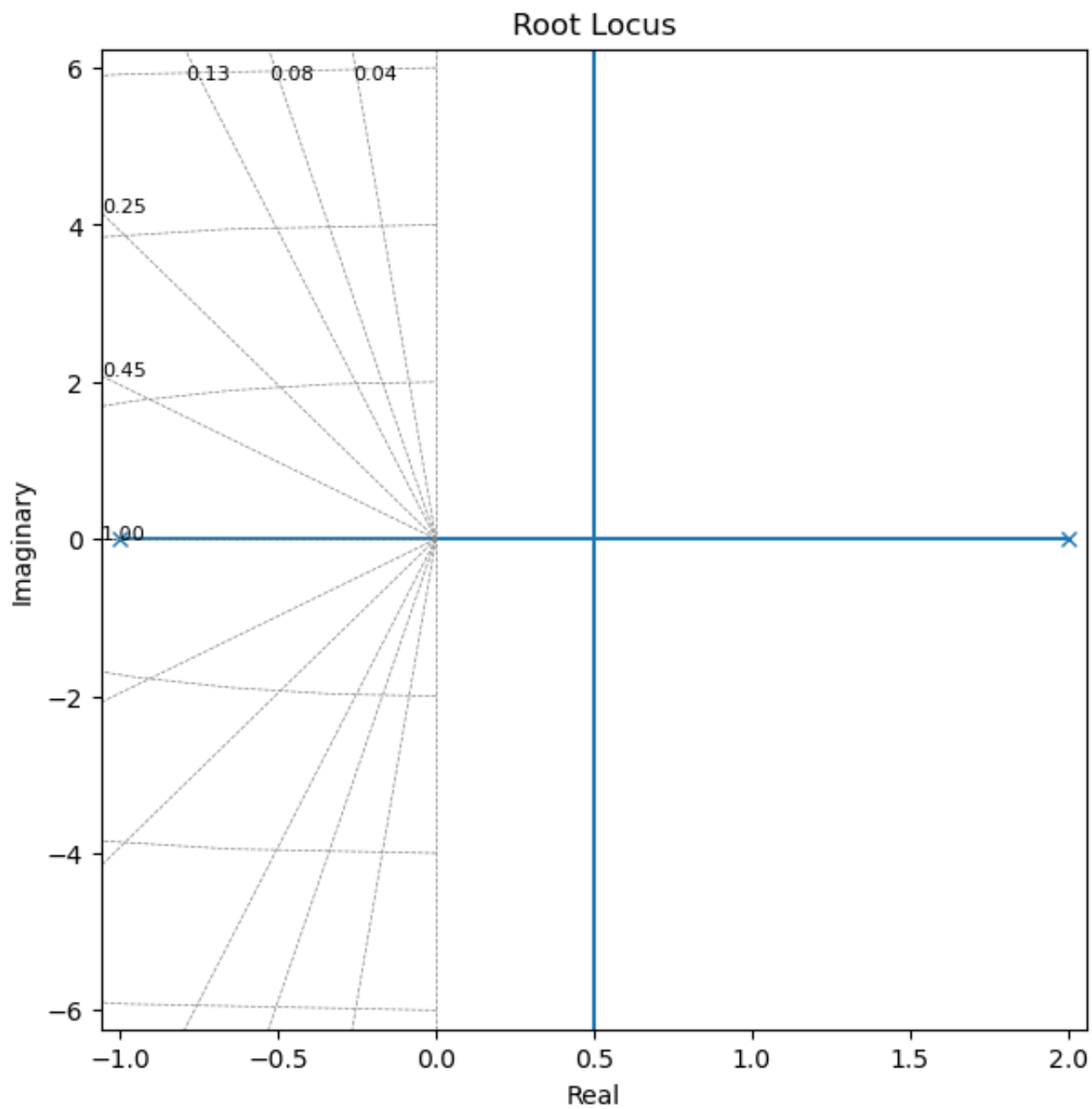
A simple control problem

We have

$$G(s) = \frac{1}{(s+1)(s-2)}$$

And the root locus is:

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
        [rlist, klist] = control.root_locus(control.tf([1],[1, -1, -2]))
```



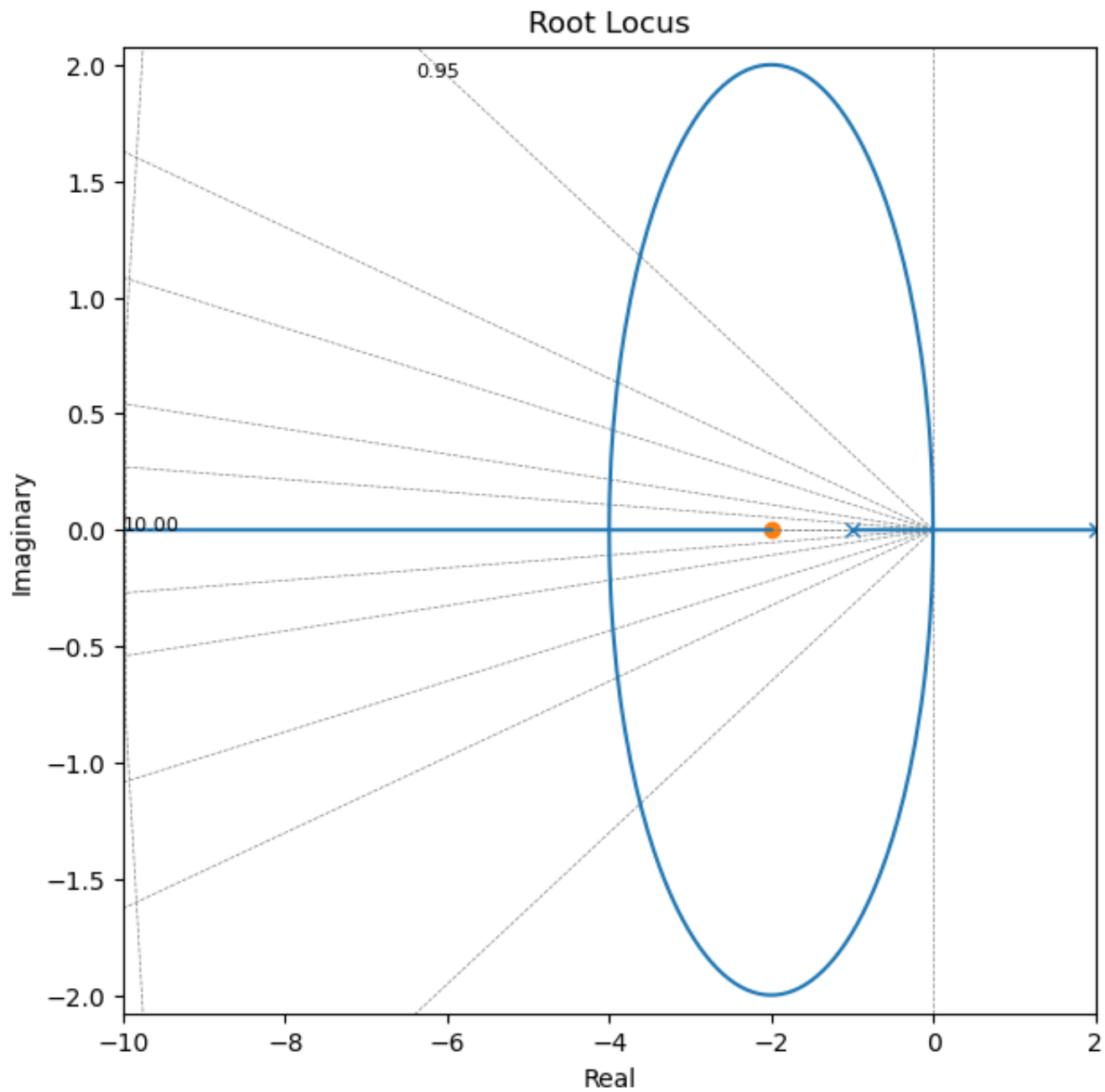
- System is unstable and there is no K that can stabilise it
- How can we make this stable?

-
- Given that we have two unmatched poles, they go off to infinity with $\pm 90^\circ$
 - However, we know that if we have only one unmatched pair, the unmatched root would go to $+180^\circ$
 - This means that if we had one more zero to the left of our poles, we should obtain the desired behaviour.

For example, we could add a zero in $s = -2$.

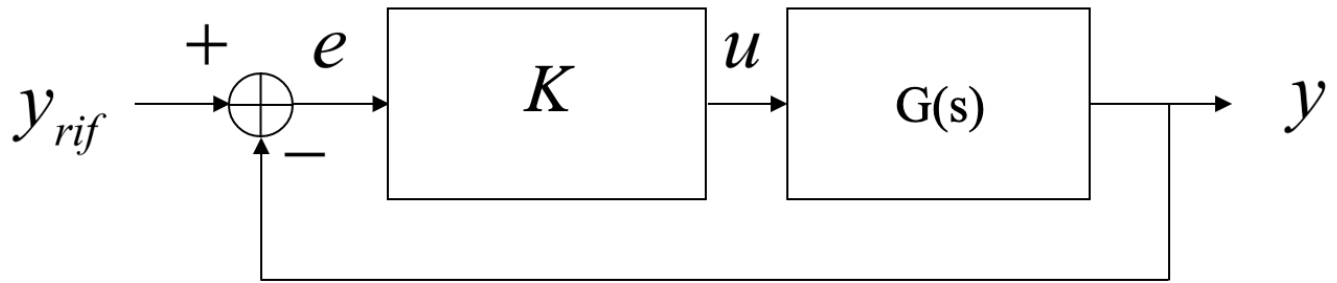
Let's use `root_locus` to confirm:

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(7, 7))
[rlist, klist] = control.root_locus(control.tf([1, 2],[1, -1, -2]))
plt.xlim((-10, 2));
```

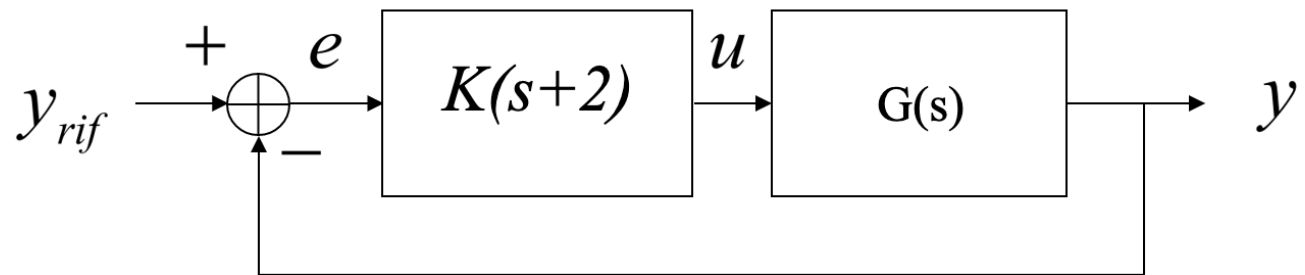


- Adding an open loop zero and choosing the correct gain has stabilised the system

To be more precise we have started from this:



and added a zero:



- The controller now has a proportional controller (K) and a derivative controller ($s + 2$)
- We have designed a PD controller in the s -domain

Additional properties

Brake-in and break-away

Correspond to those points along the real axis where the root locus goes in or away from the it. These points corresponds to multiple roots of the characteristic equation.

Let's write the characteristic equation $1 + KG(s) = 0$ as:

$$f(s) = B(s) + KA(s) = 0$$

$f(s)$ has multiple roots where $\frac{d}{ds}f(s) = 0$

We can see this considering a root of order $r \geq 2$:

$$f(s) = (s - s_1)^r (s - s_2) \dots$$

We can derive and calculate $\frac{d}{ds}f(s) = 0$ when $s = s_1$:

$$\left. \frac{d}{ds}f(s) \right|_{s=s_1} = 0$$

This means that the multiple roots of $f(s)$ satisfy our equation.

This means:

$$\frac{d}{ds}f(s) = \frac{d}{ds}B(s) + K \frac{d}{ds}A(s) = 0 \rightarrow B'(s) + KA'(s) = 0$$

and if we plug this back in the characteristic equation:

$$B(s) + KA(s) = 0 \rightarrow B(s) - \frac{B'(s)}{A'(s)}A(s) = 0$$

which we can also write as:

$$B(s)A'(s) - B'(s)A(s) = 0$$

when we solve this equation with respect to s we obtain the points where multiple roots occur.

Let's also note that:

$$B(s) + KA(s) = 0 \rightarrow K = -\frac{B(s)}{A(s)}$$

and that:

$$\frac{dK}{ds} = -\frac{B(s)A'(s) - B'(s)A(s)}{A^2(s)}$$

If dK/ds is set equal to zero, we get the same equation as before.

In summary to find the break-in/break-away points we can simply find the roots of:

$$\frac{dK}{ds} = 0$$

Note that only those points that are on the root locus are actual breakaway/break-in points.

Stated differently, if at a point at which $dK/ds = 0$ the value of K takes a real positive value, then that point is an actual breakaway or break-in point (we have assumed $K > 0$).

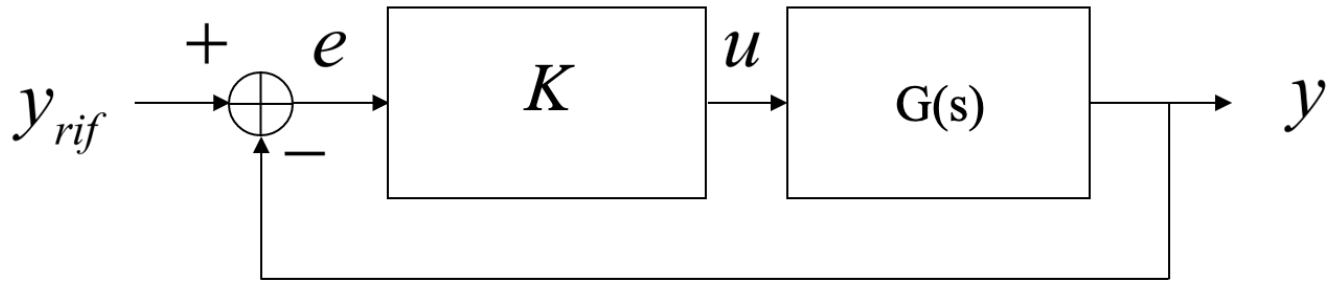
Intersections with the imaginary axis

- The points where the root locus intersects the imaginary axis can be determined using the Routh criterion: the parameters K^* and ω^* provided both the value of parameter $K = K^*$ and the frequency $\omega = \omega^*$ for which the root locus intersects the imaginary axis.

Alternatively, we can also set $s = j\omega$ in the characteristic equation and set to zero the real and imaginary parts to solve for ω and K .

Example

Let's consider a negative feedback system:



where

$$G(s) = \frac{K}{s(s+1)(s+2)}$$

with $K \geq 0$

1. Determine the root loci on the real axis.
2. Determine the asymptotes of the root loci:

$$\phi_A = \frac{2q+1}{n-m} 180$$

where $q = 0, 1, n-m-1$

We obtain: 60, 180, 300

With centroid:

$$c = \frac{0-1-2}{3} = -1$$

The asymptotes are almost parts of the root loci in regions very far from the origin.

1. Determine the breakaway points:

Let's consider the following characteristic equation $G(s) + 1 = 0$:

$$G(s) + 1 = \frac{K}{s(s+1)(s+2)} + 1 = 0$$

This means:

$$K = -(s^3 + 3s^2 + 2s)$$

Setting

$$\frac{dK}{ds} = 0$$

we obtain:

$$\frac{dK}{ds} = -(3s^2 + 6s + 2) = 0$$

and:

$$s = -0.4226 \quad s = -1.5774$$

- The breakaway point must lie on a root locus between 0 and -1 .
- $s = -0.4226$ corresponds to the actual breakaway point.
- $s = -1.5774$ is not on the root locus.

Evaluation of the values of K corresponding:

- $K = 0.3849$, for $s = -0.4226$
- $K = -0.3849$, for $s = -1.5774$

```
In [ ]: ## first point
s = -0.4226

K=-(s**3+3*s**2+2*s)
print(f'K={K} for s={s}')
```

```
## second point
s = -1.5774

K=-(s**3+3*s**2+2*s)
print(f'K={K} for s={s}')
```

K=0.3849001751759995 for s=-0.4226

K=-0.3849001751759995 for s=-1.5774

Determine the intersection with the imaginary axis:

The characteristic equation is:

$$s^3 + 3s^2 + 2s + K = 0$$

We can compute the Routh array as:

s^3	1	2
s^2	3	K
s^1	$(6 - K)/3$	
s^0	K	

The value of K that makes the s^1 term in the first column equal zero is $K = 6$ (see notebook 09_Routh_hurwitz_criterion)

The crossing points can then be found solving the auxiliary polynomial obtained from the s^2 row:

$$3s^2 + K = 3s^2 + 6 = 0$$

which gives us:

$$s = \pm\sqrt{2}j$$

- The frequencies at the crossing points on the imaginary axis are thus $\omega = \pm\sqrt{2}j$.
- The gain value corresponding to the crossing points is $K = 6$.

Alternatively we can set $s = j\omega$ in the characteristic equation, equate both the real part and the imaginary part to zero, and then solve for ω and K .

In this case:

$$(j\omega)^3 + 3(j\omega)^2 + 2(j\omega) + K = 0$$

$$(K - 3\omega^2) + j(2\omega - \omega^3) = 0$$

$$(K - 3\omega^2) = 0, \quad (2\omega - \omega^3) = 0$$

from which:

$$\omega = \pm\sqrt{2}j \text{ and } K = 6$$

and

$$\omega = 0 \text{ and } K = 0$$

1. solve for system requirements

We can now determine a pair of dominant complex-conjugate closed loop poles such that the damping ratio is $\xi = 0.5$.

- We know that they must lie on a line passing through the origin and such that:

$$\cos(\theta) = \xi \rightarrow \cos^{-1}(\xi) = \theta$$

from which:

$$\cos^{-1}(\xi) = 60^\circ$$

From the picture of the root locus we can obtain that closed-loop poles having $\xi = 0.5$ are obtained approximately as follows:

$$s = -0.3 \pm j0.58$$

We can then determine the value of K that yields such poles from what is called the **magnitude condition** (see also end of notebook):

$$K = -(s^3 + 3s^2 + 2s)$$

$$|K| = |-(s^3 + 3s^2 + 2s)|_{s=-0.3 \pm j0.58}$$

or $K = 1.066$

```
In [ ]: s=-0.3+0.58j
np.abs(s**3+3*s**2+2*s)
```

```
Out[ ]: 1.0662593604484791
```

Knowing $K = 1.066$ we can determine the third pole from:

$$s^3 + 3s^2 + 2s + 1.066$$

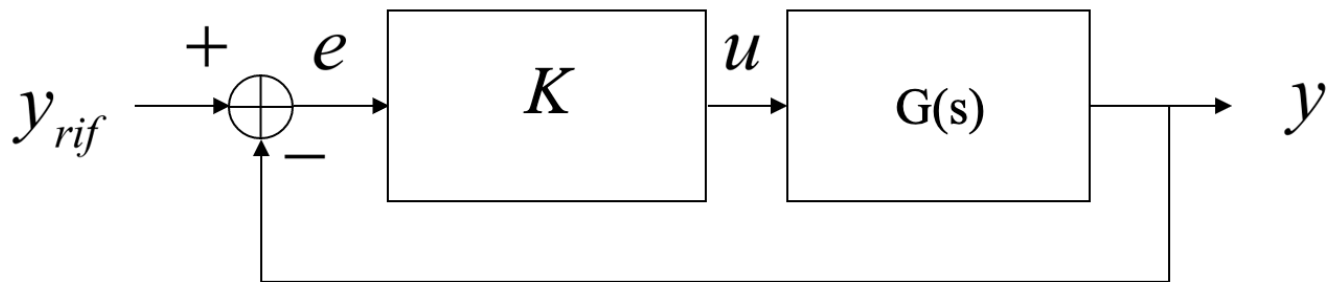
as $s = -2.3399$

```
In [ ]: np.roots([1, 3, 2, 1.066])
```

```
Out[ ]: array([-2.33997629+0.j, -0.33001185+0.5887719j,
              -0.33001185-0.5887719j])
```

- Note that for $K = 6$, the dominant poles would be on the imaginary axis with $\omega = \pm j\sqrt{2}$ and the system will have sustained oscillations.
- For $K > 6$ the dominant poles would be on the right-half plane and the system would be unstable.

Yes, But Why? Root Locus Development



Recall that the closed-loop characteristic equation is:

$$1 + KG(s) = 0$$

where $G(s) = G_{ol}(s)R(s)$ is the open-loop transfer function.

- How can we tell if an arbitrary point $s = \sigma + j\omega$ lies on the root locus?
- We seek conditions that determine whether s is a root of the characteristic equation.
- s is a root if $KG(s) = -1 + j0$.

or in polar form:

$$KG(s) = |KG(s)|e^{j\angle(KG(s))} = -1 + j0 = 1 \cdot e^{j(2n+1)\pi} \text{ for } n = 0, 1, 2, \dots$$

In fact:

$$1 + KG(s) = 0 \Rightarrow KG(s) = -1$$

Since this is a complex quantity, we can obtain a phase and a magnitude equation.

This tells us that for any point $s = \sigma + j\omega$ on the root locus:

$$|KG(s)| = 1$$

and

$$\angle(G(s)) = (2n + 1)\pi$$

This gives two important conditions:

- **The angle condition:** $\angle(G(s)) = (2n + 1)\pi$
- **The magnitude condition:** $|KG(s)| = 1$
- The angle condition is used to determine whether a point s lies on the root locus
- If a point lies on the root locus, the magnitude condition is used to determine the gain K associated with that point, since $K = 1/|G(s)|$.
- Values of s that satisfies both the phase and magnitude equations are the roots of the characteristic equation, i.e., the closed loop poles.
- If only the phase equation is satisfied we obtain the root locus.

When $K < 0$ (positive feedback) the phase relationship must be modified as:

$$\angle(G(s)) = (2n)\pi$$

Note that we can always use the angle condition to test if any point lies on the root locus

Determine the angle of departure from the complex-conjugate open-loop poles

The presence of a pair of complex-conjugate open-loop poles requires the determination of the angle of departure from these poles. Knowledge of this angle is important, since the root locus near a complex pole yields information as to whether the locus originating from the complex pole migrates toward the real axis or extends toward the asymptote.

If we choose a test point and move it in the very vicinity of the complex open-loop pole at $s = -p_1$, we find that the sum of the angular contributions from the pole at $s = p_2$ and zero at $s = -z_1$ to the test point can be considered remaining the same.

If the test point is to be on the root locus, then the sum of ϕ'_1 (phase of the zero), $-\theta_1$, and $-\theta'_2$ must be $\pm 180^\circ(2k + 1)$.

- *The angle of departure (or angle of arrival) of the root locus from a complex pole (or at a complex zero) can be found by subtracting from 180° the sum of all the angles of vectors from all other poles and zeros to the complex pole (or complex zero) in question, with appropriate signs included.*

Angle of departure from a complex pole = $180^\circ - (\text{sum of the angles of vectors to a complex pole in question from other poles}) + (\text{sum of the angles of vectors to a complex pole in question from zeros})$

Angle of arrival at a complex zero = $180^\circ - (\text{sum of the angles of vectors to a complex zero in question from other zeros}) + (\text{sum of the angles of vectors to a complex zero in question from poles})$

For example:

$$G(s) = \frac{K(s+2)}{s^2 + 2s + 3}$$

$$s = -1 \pm j\sqrt{2}$$

- The real axis $[-\infty, -2]$ is part of the loci
- Since there are two open-loop poles and one zero, there is one asymptote, which coincides with the negative real axis.
- The angle of departure from the complex-conjugate poles is:

$$\phi'_1 - (\theta_1 + \theta'_2) = \pm 180^\circ(2k + 1)$$

or

$$\theta_1 = 180 - \theta'_2 + \phi'_1 == 180 - \theta_2 + \phi_1 =$$

which is:

$$\theta_1 = 180 - \theta_2 + \phi_1 == 180 - 90 + 55 = 145^\circ$$

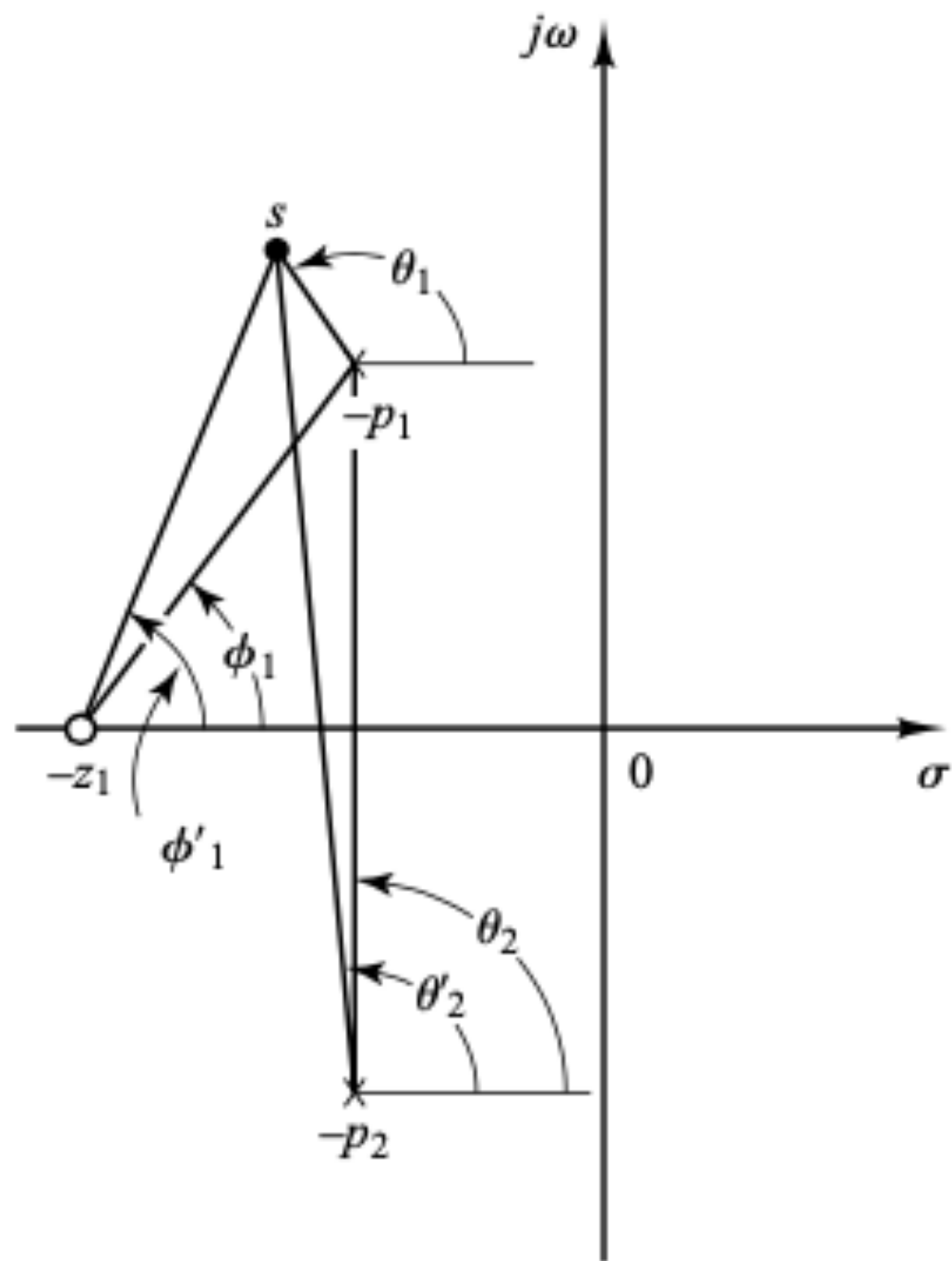


Figure From Ogata, Modern Control Engineering

Break-in points:

$$K = -\frac{s^2 + 2s + 3}{s + 2}$$

$$\frac{dK}{ds} = -\frac{(2s + 2)(s + 2) - (s^2 + 2s + 3)}{(s + 2)^2} = 0$$

which gives:

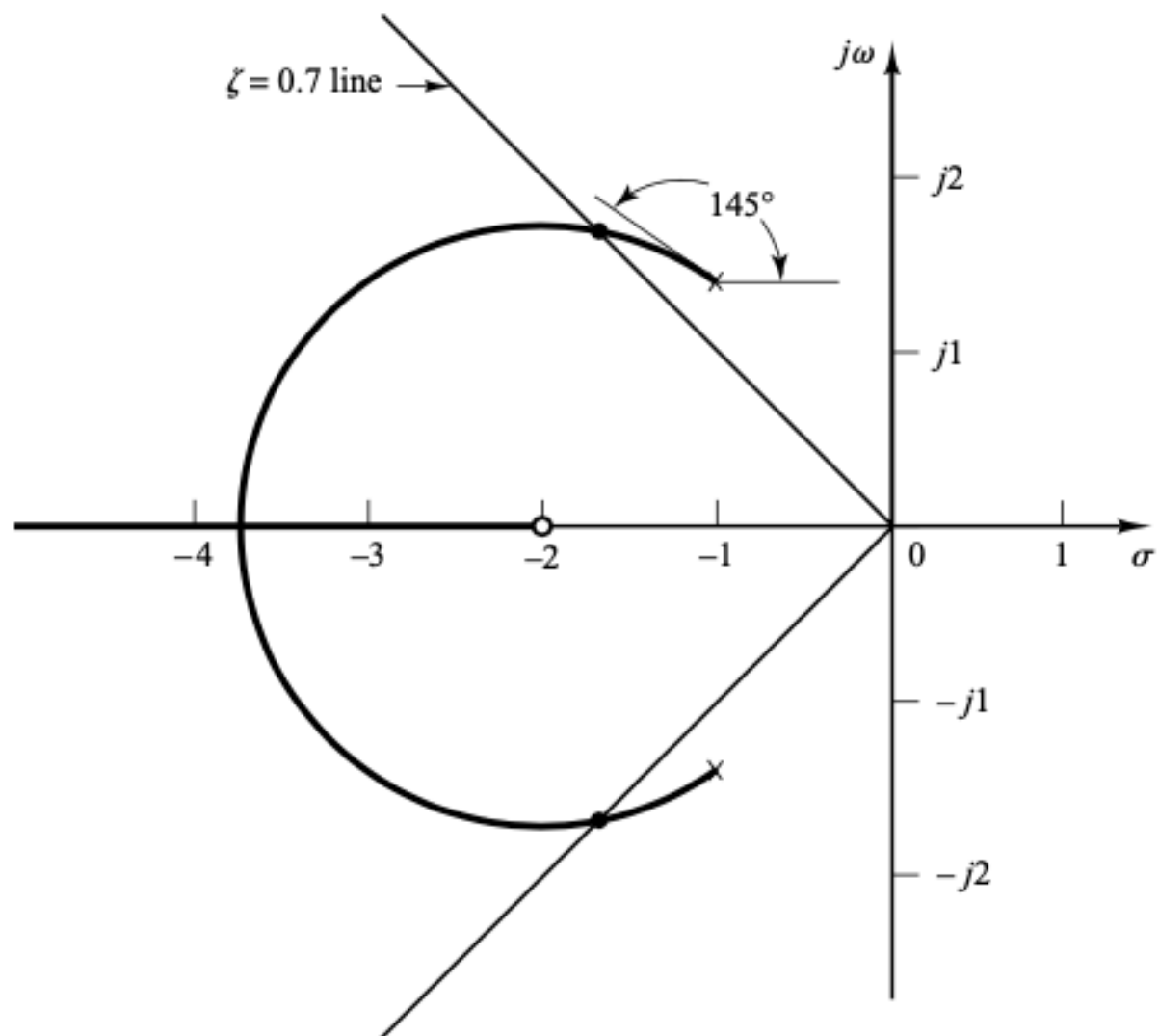
$$s^2 + 4s + 1 = 0$$

from which:

$$s = -3.7320, \quad s = -0.2680$$

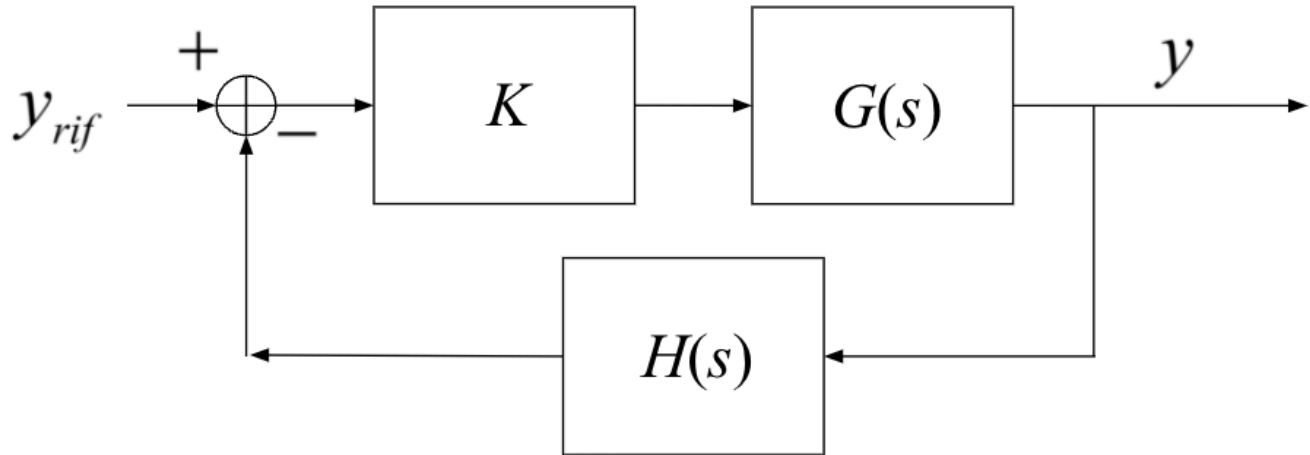
$s = -3.732$ is on the root locus and hence it is a break-in point.

- To determine accurate root loci, several points must be found by trial and error between the break-in point and the complex open-loop poles. (To facilitate sketching the root-locus plot, we should find the direction in which the test point should be moved by mentally summing up the changes on the angles of the poles and zeros.)



Example

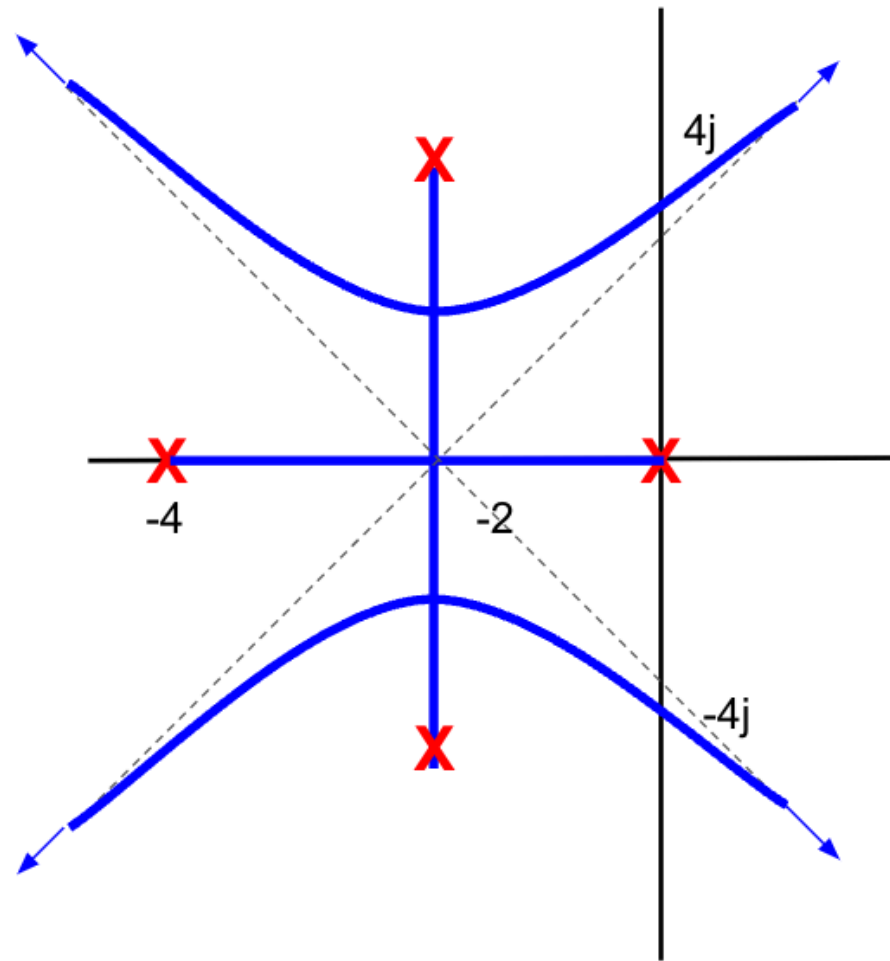
$$G(s)H(s) = \frac{1}{s(s+4)(s^2+4s+20)}$$



- Poles:

$$s = 0, -4, -2 \pm 4j$$

- 4 asymptotes:
 - angles: 45, 135, 225, 315 (deg)
 - centroid: -2 (everything is symmetric)
- How do the poles get to the asymptotes?



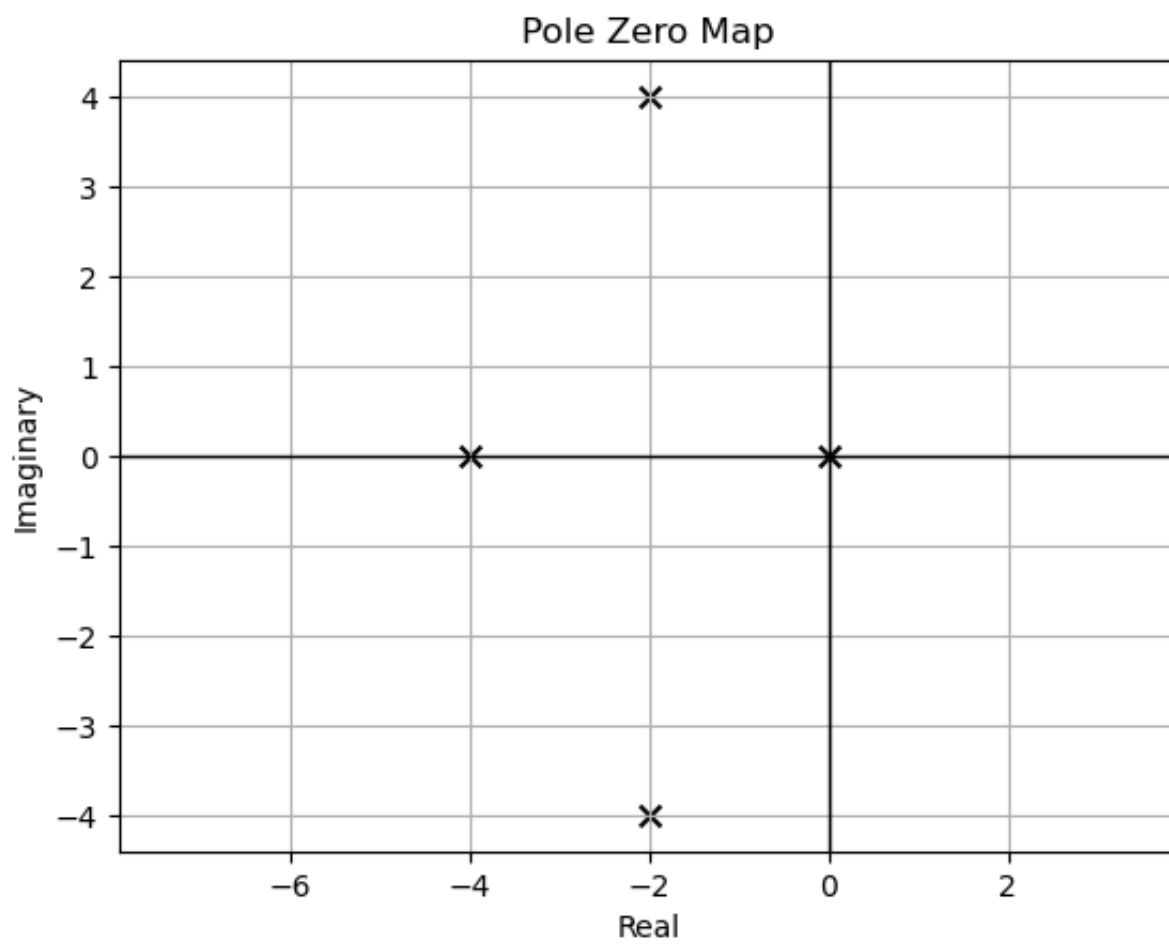
Let's use what the root locus looks like:

```
In [ ]: s = control.tf([1, 0], [1])
K = 1
GH = 1/(s*(s+4)*(s**2+4*s+20))
print(GH)
```

```

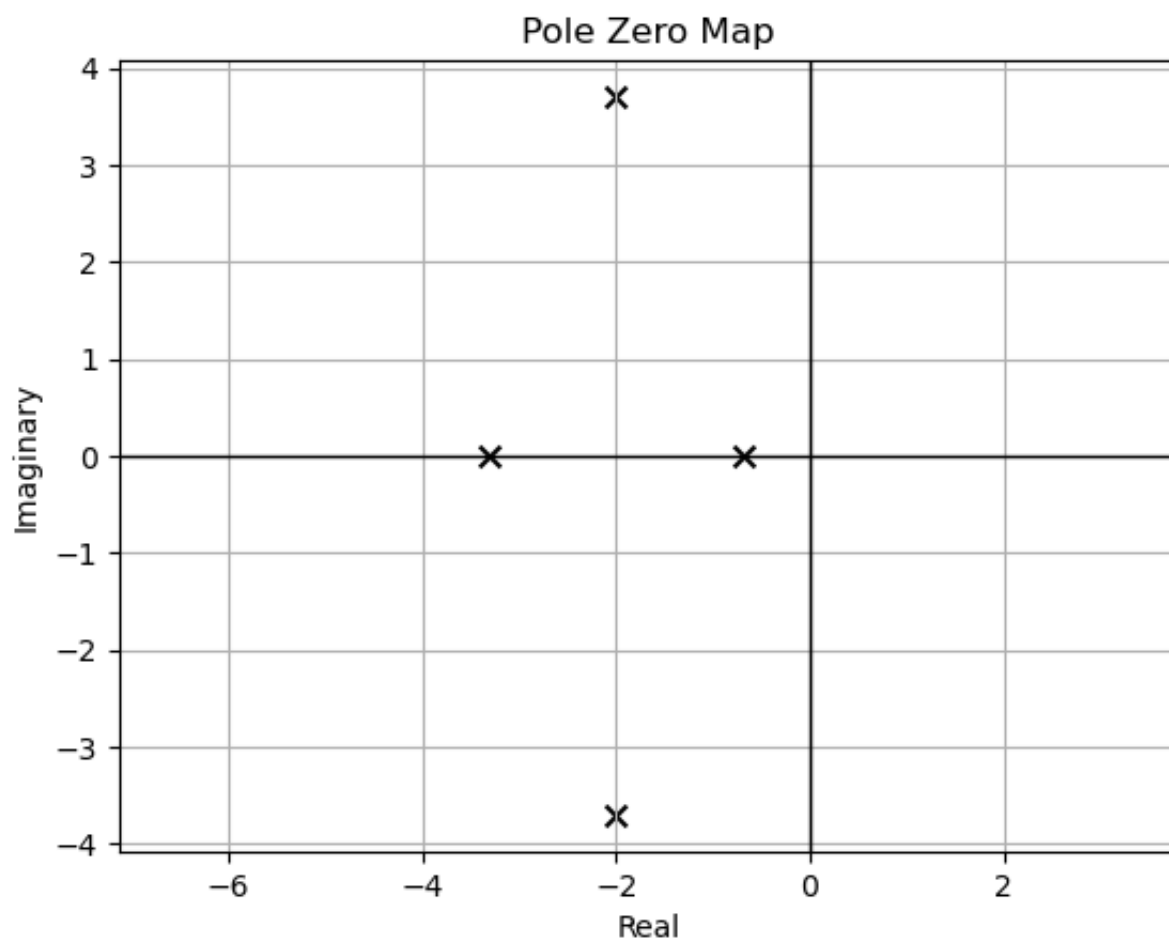
      1
-----
s^4 + 8 s^3 + 36 s^2 + 80 s
```

```
In [ ]: control.pzmap(GH)
plt.grid()
```

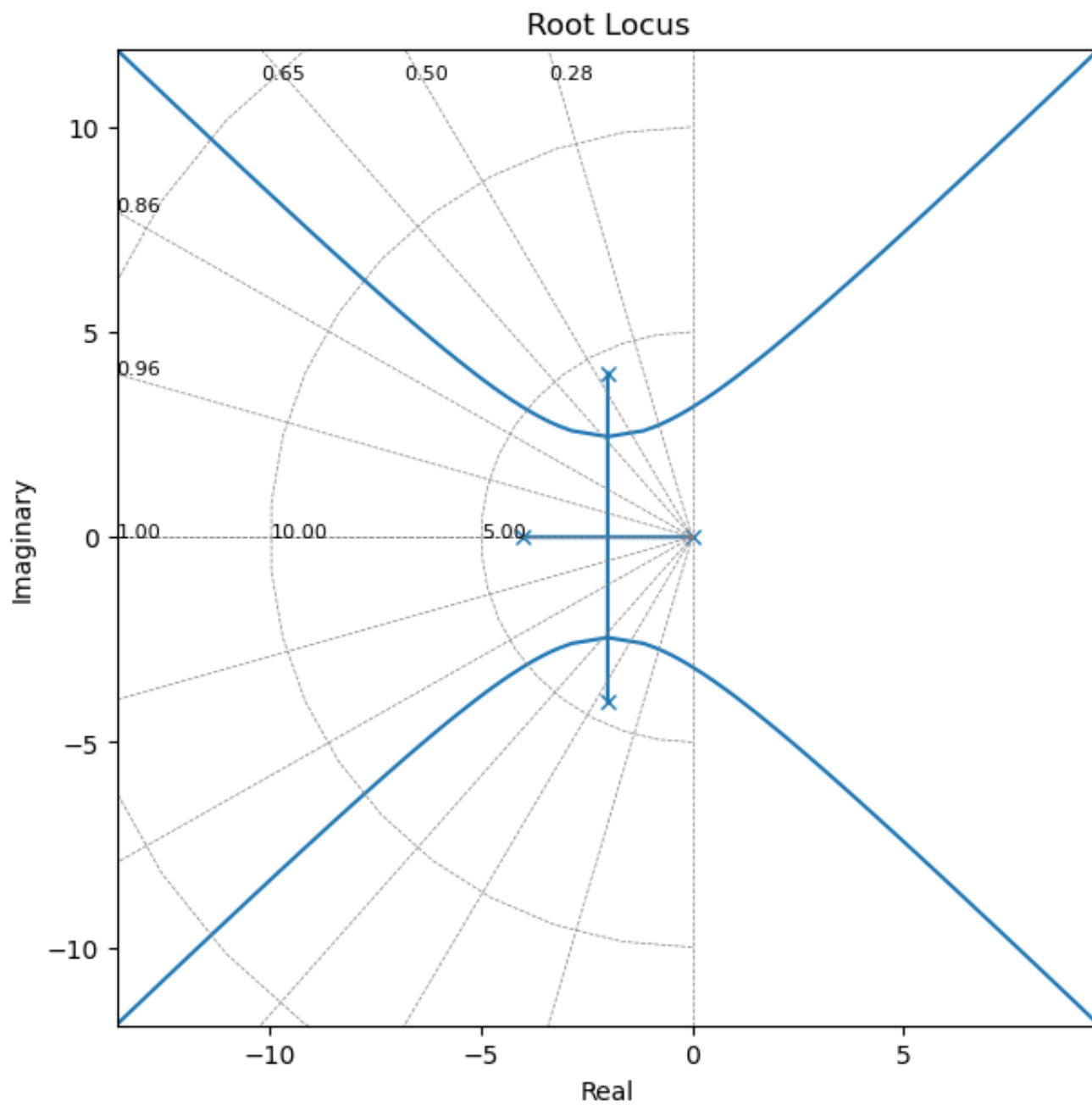


And for the closed loop system:

```
In [ ]: # Change K from 1 to 105
control.pzmap(control.feedback(GH*40, 1));
plt.grid()
```

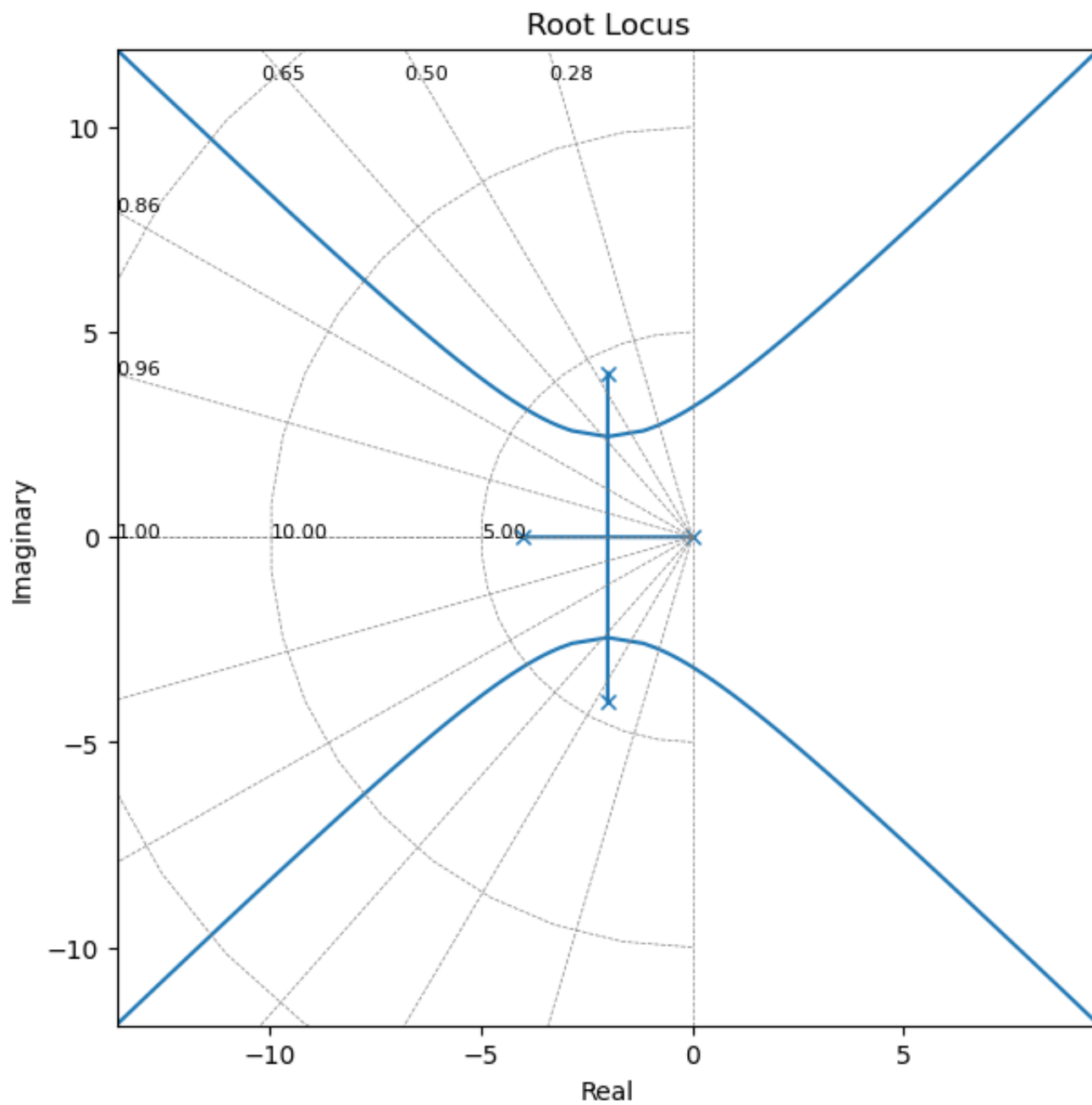


```
In [ ]: fig, axes = plt.subplots(1,1,figsize=(7, 7))
control.rlocus(GH);
```



- `control.rlocus` parameters:
 - `kvect` (list or ndarray, optional) – List of gains to use in computing diagram.
 - `print_gain` (bool) – If True (default), report mouse clicks when close to the root locus branches, calculate gain, damping and print.

```
In [ ]: # %matplotlib notebook
fig, axs = plt.subplots(1,1,figsize=(7, 7))
control.rlocus(GH);
```



- The MATLAB sisotool has also additional tools that can help you in your controller design.

Comments on the Root Locus Plot

- A slight change in the pole-zero configuration may cause significant changes in the root-locus configurations.
- The next two plots demonstrate the fact that a slight change in the location of a zero or pole will make the root-locus configuration look quite different.


```
In [ ]: s = control.tf([1, 0], [1])

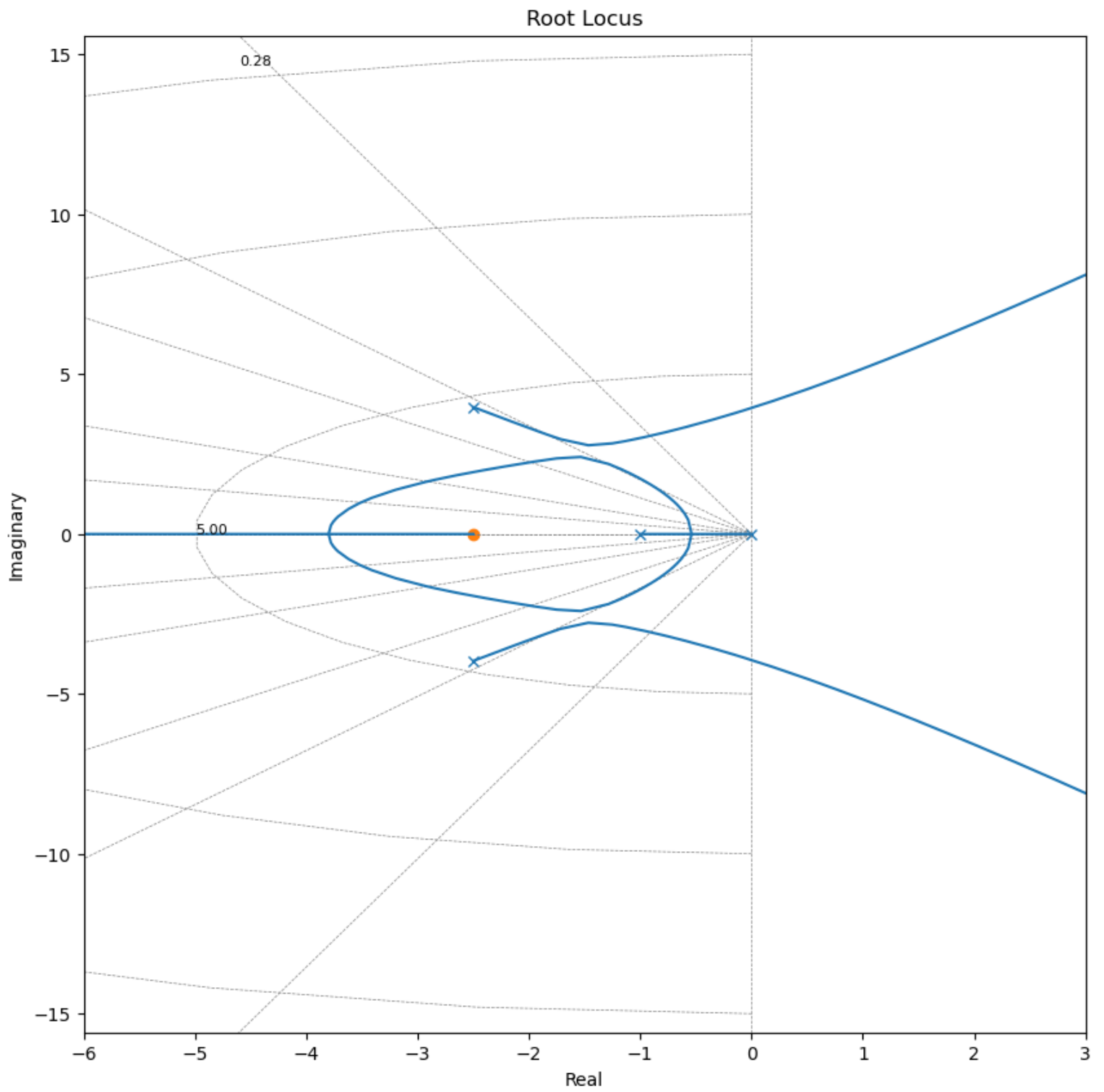
K = 1
GH = (s+2.5)/(s*(s+1)*(s**2+5*s+22))

print(GH)
```

$$\frac{s + 2.5}{s^4 + 6 s^3 + 27 s^2 + 22 s}$$

```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(10, 10))
control.rlocus(GH);
axs.set_xlim((-6,3))
```

```
Out[ ]: (-6.0, 3.0)
```

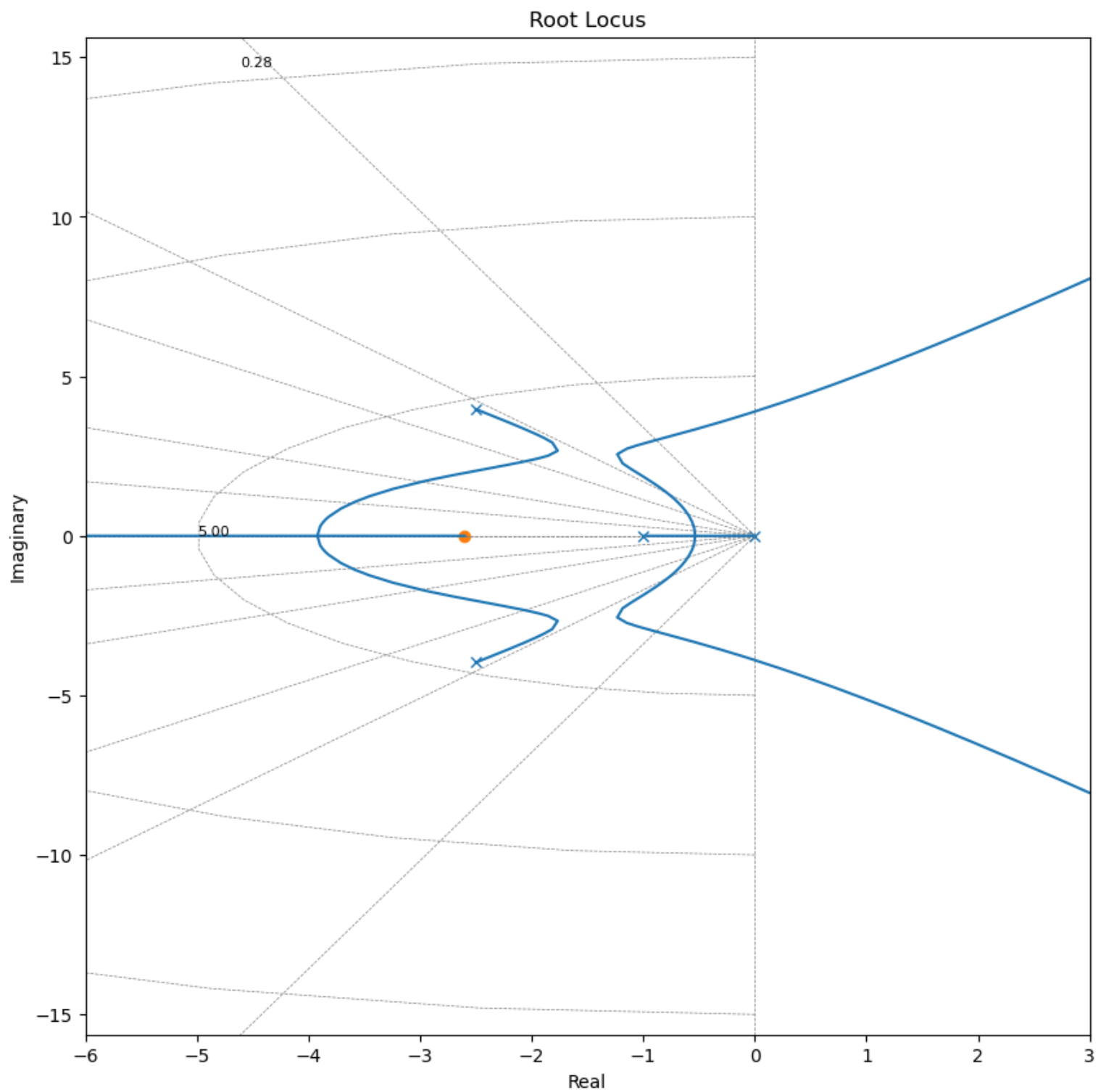


```
In [ ]: s = control.tf([1, 0], [1])
K = 1
GH = (s+2.6)/(s*(s+1)*(s**2+5*s+22))
print(GH)
```

$$\frac{s + 2.6}{s^4 + 6s^3 + 27s^2 + 22s}$$

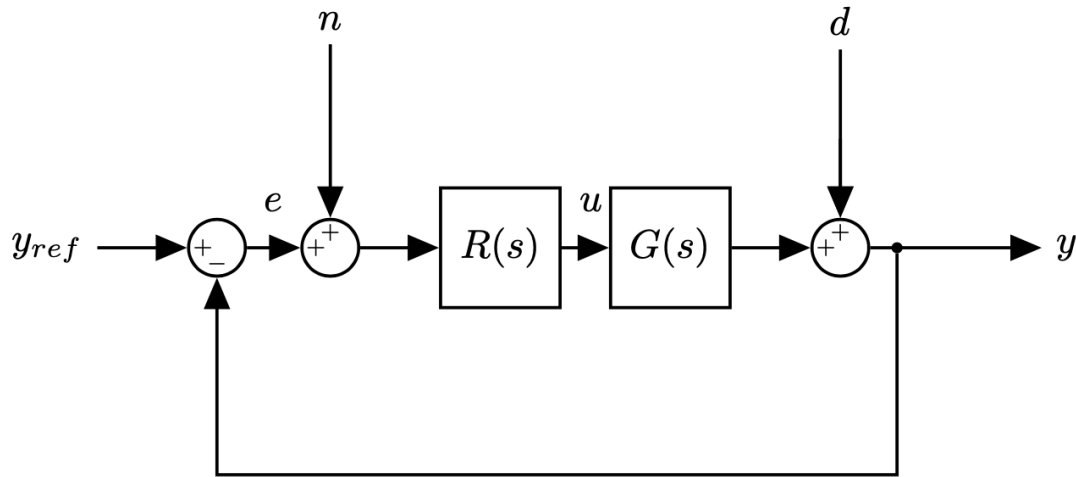
```
In [ ]: fig, axs = plt.subplots(1,1,figsize=(10, 10))
control.rlocus(GH);
axs.set_xlim((-6,3))
```

```
Out[ ]: (-6.0, 3.0)
```



Cancellations of Poles of $G(s)$ with Zeros of $H(s)$

The more general case



- It is important to note that if the denominator of $G(s)$ and the numerator of $R(s)$ involve common factors, then the corresponding open-loop poles and zeros will cancel each other, reducing the degree of the characteristic equation by one or more.
- The root-locus plot of $G(s)R(s)$ does not show all the roots of the characteristic equation, only the roots of the reduced equation.
- To obtain the complete set of closed-loop poles, we must add the canceled pole of $G(s)R(s)$ to those closed-loop poles obtained from the root-locus plot of $G(s)R(s)$.
- The important thing to remember is that the canceled pole of $G(s)R(s)$ is a closed-loop pole of the system

Further comments

- Plotting the damping ratio line given a desired overshoot S (e.g. $S = 15\%$)

We know that:

$$\xi = \frac{|\ln(S/100)|}{\sqrt{\pi^2 + \ln^2(S/100)}}$$

and the angle $\beta = \cos^{-1}(\xi)$

Let's see this in action:

```
In [ ]: # Percentage Overshoot
S = 15

# Equivalent damping ratio (from the equation)
xi = np.abs(np.log(S/100)/np.sqrt(np.pi**2+np.log(S/100)**2))

print('Damping ratio (xi): {:.4f}'.format(xi))
```

Damping ratio (xi): 0.5169

```
In [ ]: # desired damping lines
phi = np.arccos(xi)
print('Angle phi: {:.4f} deg'.format(phi*180/np.pi))
```

Angle phi: 58.8734 deg

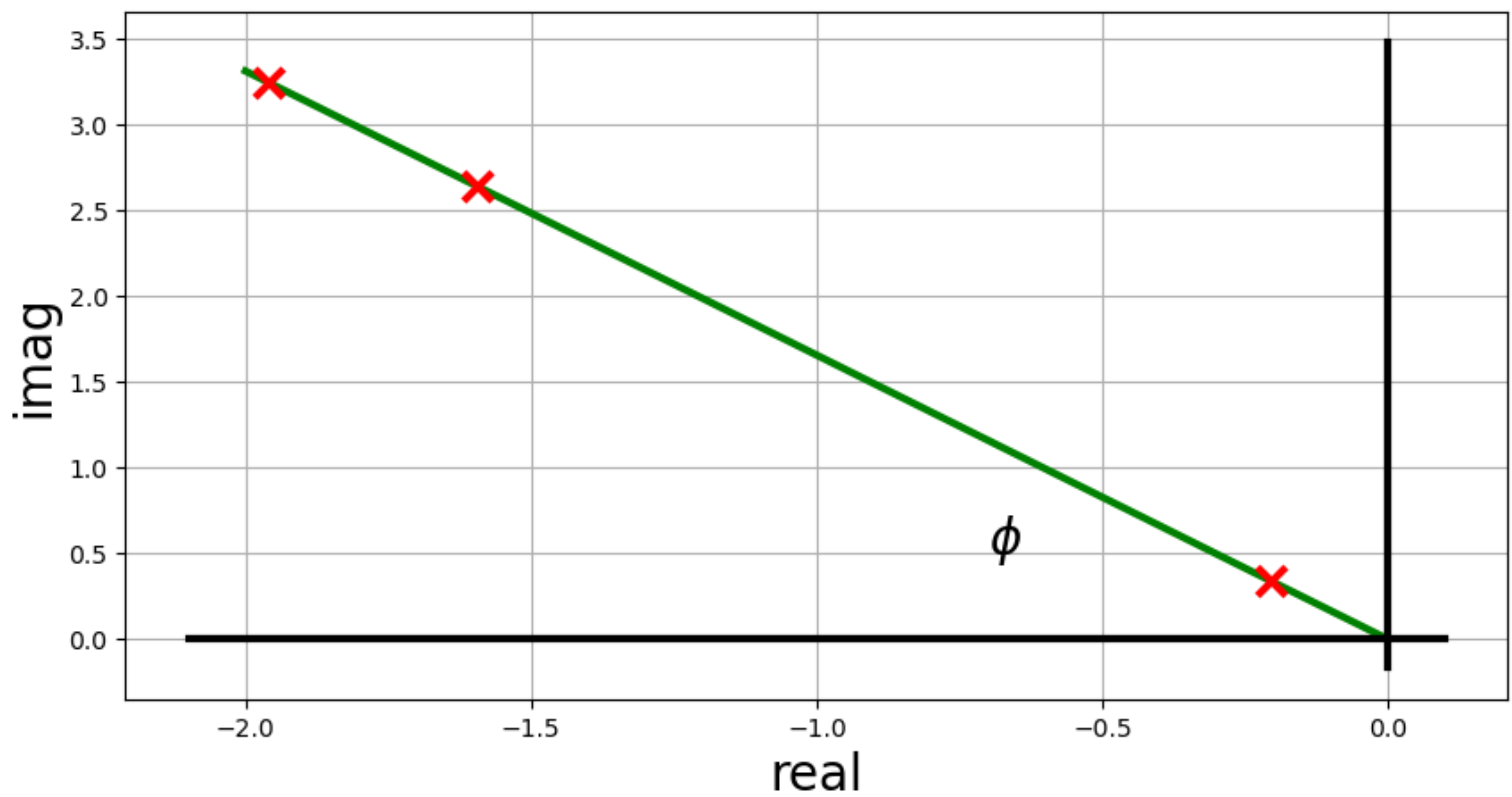
We can plot it:

```
In [ ]: fig, axs = plt.subplots(1,1, figsize=(10,5))

x_range = np.linspace(-2, 0) # points on the constant damping ratio line

plt.plot(x_range, -np.tan(phi)*x_range,
         linewidth=3, color='g')

plt.plot([x_range[1], x_range[10], x_range[-6]],
         -np.tan(phi)* np.array([x_range[1], x_range[10], x_range[-6]]),
         linewidth=3, linestyle='', color='r', marker='x', markeredgewidth=3, markersize=12)
plt.plot(plt.xlim(), [0, 0], linewidth=3, color='k')
plt.plot([0, 0], plt.ylim(), linewidth=3, color='k')
plt.text(-0.7, 0.5, "$\phi$", fontsize=20)
plt.grid()
plt.xlabel('real', fontsize=20)
plt.ylabel('imag', fontsize=20);
```



- All points along the line have the same damping ratio

Now we can go deeper in our analysis and define a second order transfer function.

- Note that to do this, we need to choose the natural frequency ω_n .
- Natural frequency however does not influence overshoot, we are free to select any value. For example:
 $\omega_n = 1 \text{ rad/s}$.

```
In [ ]: w = 1
```

Define the transfer function using the Python Control Library.

Let's use another way to do it, creating a transfer function object 's' first:

```
In [ ]: s = control.tf([1, 0],[1])
print(s)
```

```
s
-
1
```

And now we can use the 's' object to define more complex transfer functions more easily:

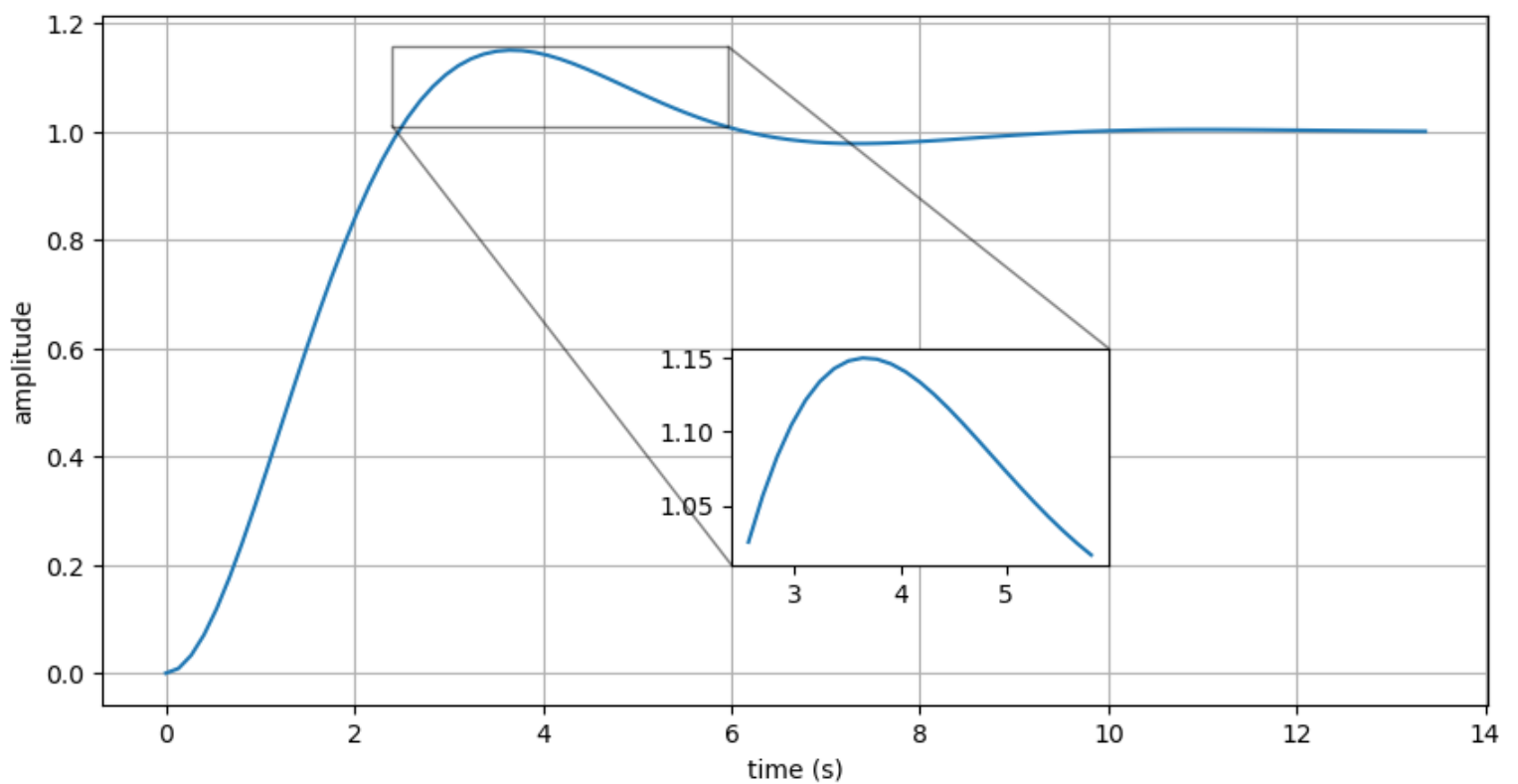
```
In [ ]: G = w**2/(s**2+2*xi*w*s+w**2)
print(G)
```

$$s^2 + 1.034 s + 1$$

And we can plot the step response:

```
In [ ]: [tout, yout] = control.step_response(G)
```

```
In [ ]: fig, ax = plt.subplots(1,1, figsize=(10,5))
ax.plot(tout, yout)
axin1 = ax.inset_axes(
    [6, 0.2, 4, .4], transform=ax.transData)
axin1.plot(tout[yout>1.01], yout[yout>1.01])
ax.inset_zoom(axin1, edgecolor="black")
plt.xlabel('time (s)')
plt.ylabel('amplitude')
plt.grid()
```



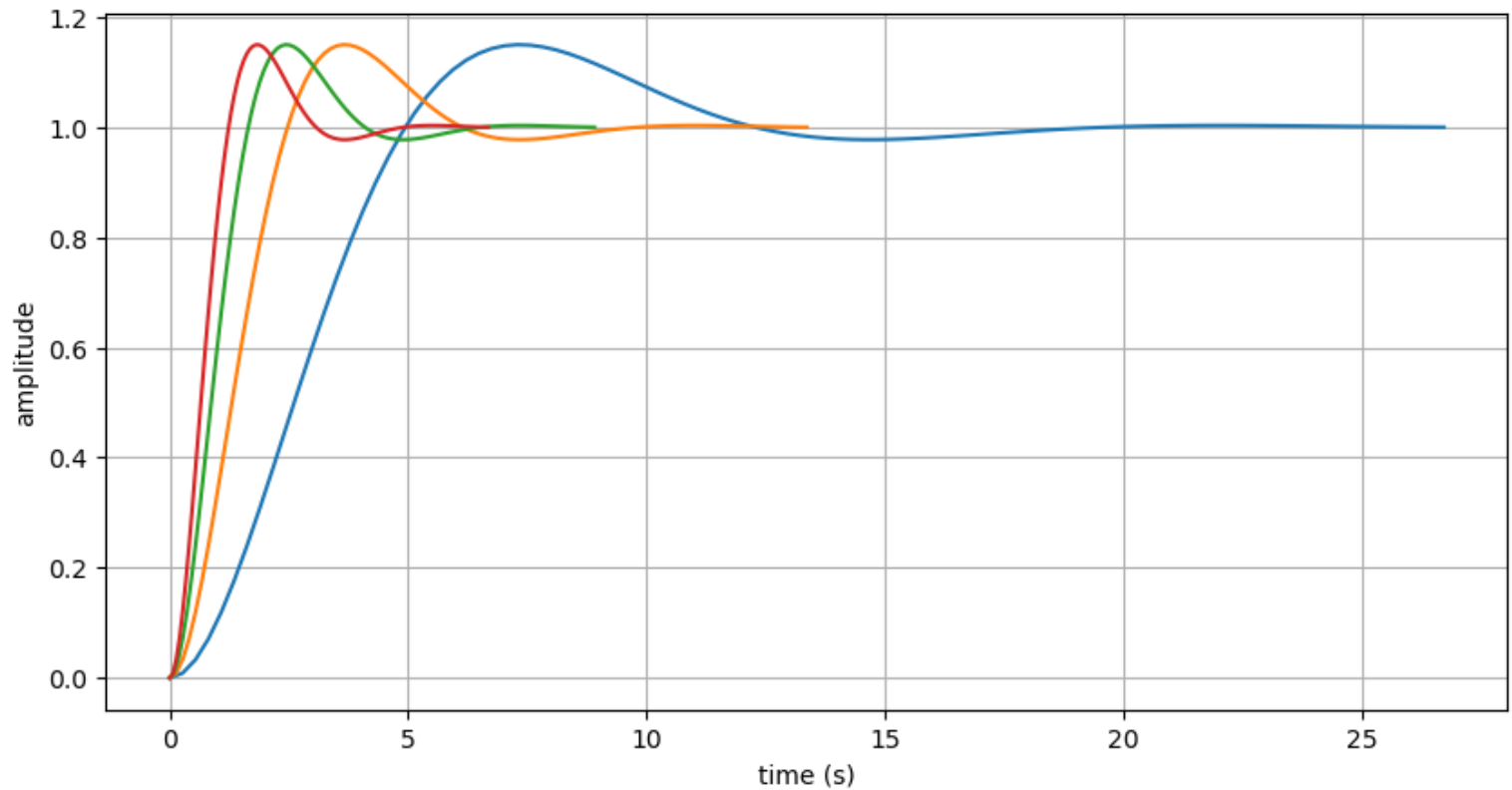
- As expected there is 15% overshoot

Let's now verify that the natural frequency does not impact overshoot:

```
In [ ]: fig, ax = plt.subplots(1,1, figsize=(10,5))
for w in np.arange(0.5, 2.1, 0.5):
    print('w', w)
    G = w**2/(s**2+2*xi*w*s+w**2)
    [tout, yout] = control.step_response(G)
    plt.plot(tout, yout)

plt.xlabel('time (s)')
plt.ylabel('amplitude')
plt.grid()
```

```
w 0.5
w 1.0
w 1.5
w 2.0
```



- All responses have exactly 15% overshoot
- The rise time and settling time are influenced by the natural frequency

Let's see how the poles are impacted:


```

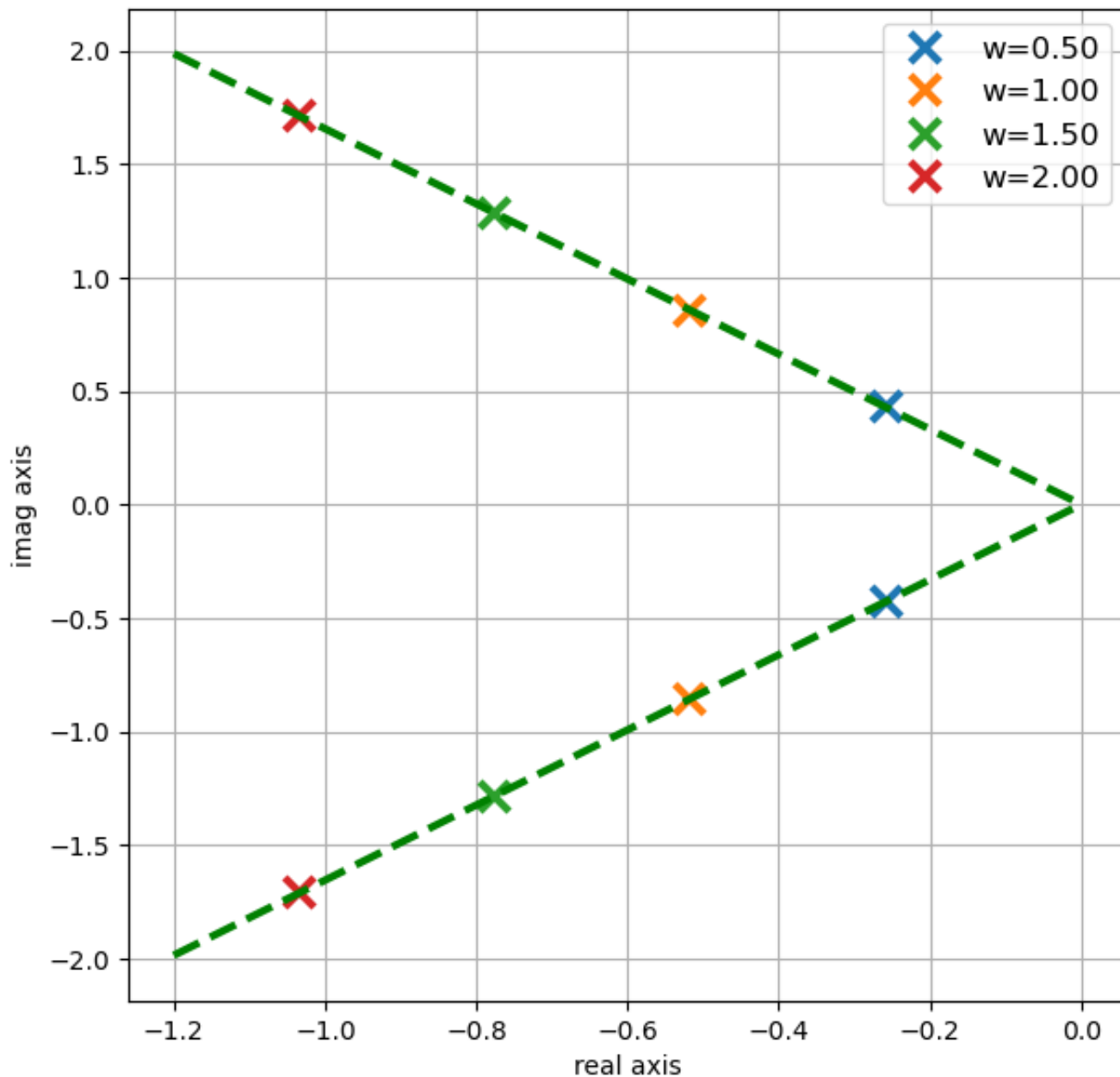
In [ ]: fig, ax = plt.subplots(1,1, figsize=(7,7))

for w in np.arange(0.5, 2.1, 0.5):
    roots = np.roots([1,+2*xi*w,+w**2])
    plt.plot(np.real(roots), np.imag(roots),
             linestyle='', marker='x', markeredgewidth=3, markersize=12, label='w={:.2f}'.f

x_range = np.linspace(-1.2, 0)
plt.plot(x_range, -np.tan(phi)*x_range, linewidth=3, color='g', linestyle='--')
plt.plot(x_range, +np.tan(phi)*x_range, linewidth=3, color='g', linestyle='--')

plt.legend(fontsize=12)
plt.xlabel('real axis')
plt.ylabel('imag axis')
plt.grid()

```



Requirements

- Time requirements
 - Raise time
 - Peak time
 - Settle time
 - Overshoot
 - Steady state error
- Frequency requirements
 - Bandwidth
 - Gain Margin
 - Phase Margin
- **S-Domain**
 - Pole/Zero locations
- Obviously they are all related
- We have tools in the s-domain or in the frequency domain that we do not have in the time domain (e.g. root locus)

When focusing on S-Domain, the location of poles and zeros can be expressed:

- rectangular coordinates: $x + yj$
- polar coordinates: ω_n, ξ

```

In [ ]: fig, ax = plt.subplots(1,1, figsize=(7,7))

xi = 0.5
w = 1
phi = np.arccos(xi)

our_roots = np.roots([1, 2*xi*w, w**2])

for xi in np.arange(0, 1, 0.05):
    r = np.roots([1, 2*xi*w, w**2])
    plt.plot(np.real(r), np.imag(r),
             color='k', linestyle='', marker='.', markeredgewidth=1, markersize=1)

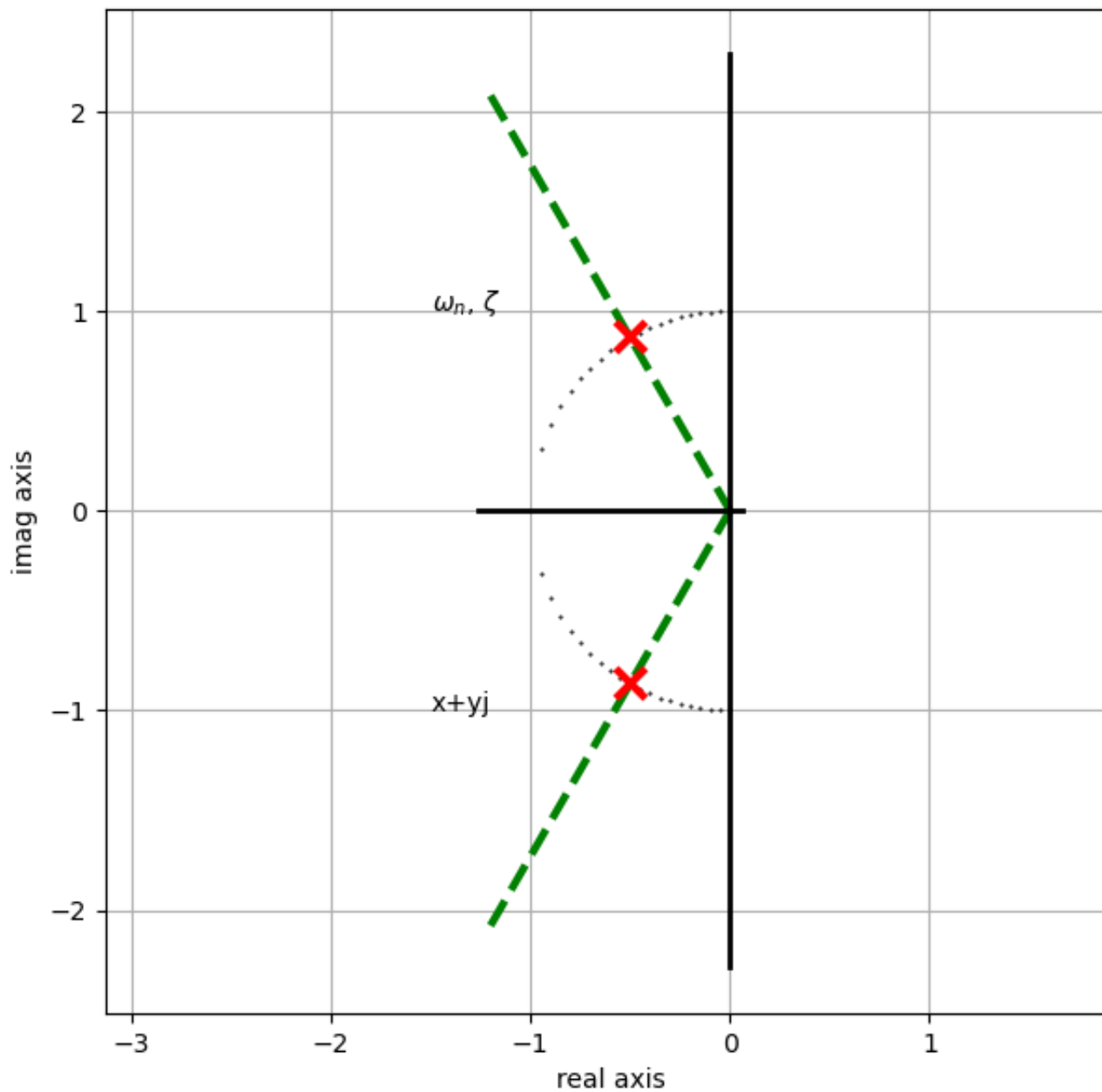
# constant damping ratio line
x_range = np.linspace(-1.2, 0)
plt.plot(x_range, -np.tan(phi)*x_range, linewidth=3, color='g', linestyle='--')
plt.plot(x_range, +np.tan(phi)*x_range, linewidth=3, color='g', linestyle='--')

# poles
plt.plot(np.real(our_roots), np.imag(our_roots),
        color='r', linestyle='', marker='x', markeredgewidth=3, markersize=12)

# axis
plt.plot(plt.xlim(), [0, 0], linewidth=2, color='k')
plt.plot([0, 0], plt.ylim(), linewidth=2, color='k')

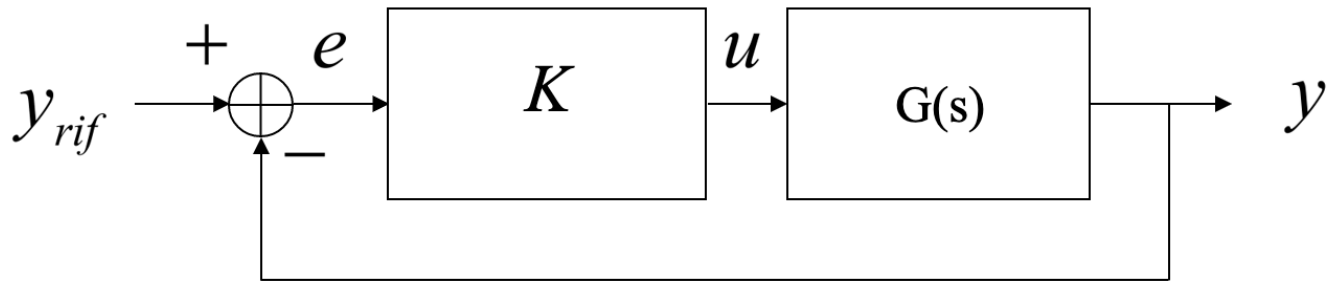
plt.text(-1.5, -1, 'x+yj')
plt.text(-1.5, 1, '$\omega_n$, $\zeta$')
plt.xlabel('real axis')
plt.ylabel('imag axis')
plt.axis('equal')
plt.grid()

```



- Damping ratio and natural frequency are only defined for 2nd order system
- They lose their meaning in higher order systems
- We saw that many higher order systems exhibit 2nd order behaviours (e.g. dominant poles)
- When we have a requirement expressed in terms of damping ratio or natural frequency we are assuming that the system is 2nd order
- Play around with high order systems and verify what happens!

Solving for gain in a root locus plot



Let's consider an arbitrary $G(s)$:

$$G(s) = \frac{5}{(s+4)(s+2)}$$

- Determine what value of K gives a 15% overshoot

Let's start plotting the root locus

```
In [ ]: s = control.tf([1, 0],[1])
```

```
In [ ]: fig, axs = plt.subplots(1,1, figsize=(10, 5))
control.rlocus(5/((s+4)*(s+2)));

# desired damping ratio line
x_range = np.linspace(-4, 0)
plt.plot(x_range, -np.tan(1.0276)*x_range, linestyle='--')
plt.ylim(-5.5, 5.5)
```

```
Out[ ]: (-5.5, 5.5)
```

