root locus wooo

Contents

1 Introduction

The *root locus* is a graphical technique that shows how the *closed–loop poles* of a feedback system move in the complex plane as a real scalar gain K varies from 0 to ∞.

Throughout these notes we are assuming a **unity feedback** system

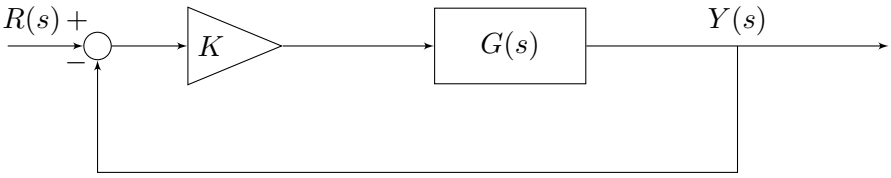


Figure 1: unity feedback system!

Characteristic equation. The closed–loop transfer function is

$$\frac{Y(s)}{R(s)} = T(s) = \frac{KG(s)}{1+KG(s)}, \qquad \Rightarrow \qquad \underbrace{1+KG(s)=0}_{\text{characteristic equation}} .$$

The roots of $1 + KG(s)$ are the closed–loop poles. The root locus is the *plot* of those roots as K varies in the range $[0, \infty)$.

Let's assume that $G(s)$ is a transfer function that can be expressed as a fraction $\frac{N(s)}{D(s)}$. This means we can express $T(s)$ as:

$$T(s) = \frac{\frac{KN(s)}{D(s)}}{1+\frac{KN(s)}{D(s)}} = \frac{KN(s)}{KN(s)+D(s)}$$

Note that if K is really small:

$$T(s) = \frac{-}{\approx 0 + D(s)}$$

We can see that the poles of $T(s)$ for a small K are the roots of $D(s)$, also known as the poles of the open loop TF $G(s)$! Similarly, if K is really large, we can say that the $KN(s)$ term dominates!

$$T(s) = \frac{\uparrow KN(s)}{\uparrow KN(s) + D(s)} = \frac{KN(s)}{KN(s) + \approx 0}$$

We can see that as $K \to \infty$, the poles of the transfer function are the same as the roots of $N(s)$, which are the zeroes of the open loop TF $G(s)$!

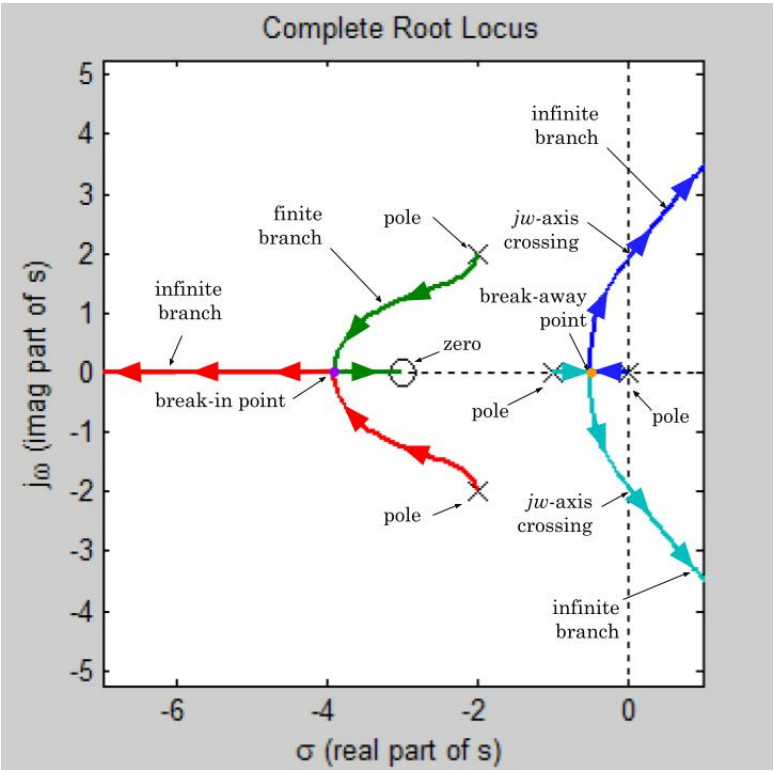
So for a given open loop transfer function $G(s)$ in unity feedback with gain K, the **root locus plots how the poles of the closed loop transfer function move from the open loop poles to the open loop zeroes, as K goes from 0 to ∞.**

2 Terminology and Notation

- **Branch/Infinite Branch** - The term *branch* will often have conflicting notation in controls books. It is sometimes used to **solely** refer to a line between an infinite pole/zero and a finite pole/zero. For clarity in this note, we will call this an *infinite* branch.
- **Finite Branch** - A line on the root locus between a finite pole and a finite zero.
- **Finite Pole/Zero** - This is a pole or zero with a finite value, plotted on the graph. A pole is noted as a $\times$, and a zero is noted as a $\circ$.
- **Infinite Pole/Zero** - On a root locus, all lines have a *direction*: *from* an open-loop pole *to* an open-loop zero. Sometimes, we have more finite open-loop poles than finite open-loop zeroes, or vice versa. In this case, we will have sections of the graph that continue to infinity. These *infinite branches* are meant to visualize infinite poles/zeroes, referring to the other end of the branch that does not have a finite value.
- **Break-away point**: A point on the root locus where a single point splits *away* into multiple lines
- **Break-in point**: A point on the root locus where multiple lines converge *away* into a single point.

  Note: A point can be both a break-in and break-away point.

Below is an example root locus with some of these terms marked!



3 Angle and Magnitude Criteria [Proofs Link]

Here are the two core root–locus conditions that define all points on the root locus:

Angle (Phase) Criterion : A point s lies on the locus if and only if

$$\boxed{\sum_j \angle(s - z_j) - \sum_i \angle(s - p_i) = (2k+1)\pi,\ k \in \mathbb{Z}.}$$

Here, $\angle(s - z_i)$ refers to the angle created by the line from the point s to a given zero z_i and the +real axis.

Magnitude Criterion : For any point s satisfying the angle criterion, the gain that places a closed–loop pole there is $\boxed{K = \dfrac{\prod_j |s - p_j|}{K_g \prod_i |s - z_i|}.}$ Here, the gain K_g is the inherent constant gain of the transfer function, if there is one. For example, $H(s) = \frac{2(s-2)}{s+3}$ would have $K_g = 2$.

The angle criterion determines *where* the branches are; the magnitude criterion determines *for which* K. All graphical rules stem from these two formulas.

4 Root–Locus Construction Rules [Proofs Link]

1. **Direction & Infinite Branches:** As K increases from 0 to ∞, it moves from the open loop poles to the open loop zeroes. If there are differing numbers of finite poles and zeroes, there will be $|\#p - \#z|$ *infinite* poles/zeroes (whichever one has fewer in the open-loop TF)

2. **Real–axis segments:** On the real axis the locus exists to the *left* of an odd number of real poles and zeros.

3. **Symmetry:** The locus is symmetric about the real axis.

4. **Asymptote angles:**

$$\theta_k = \frac{(2k+1)\pi}{\#p - \#z},\ k = 0, 1, \ldots, \#p - \#z - 1$$

5. **Breakaway/Break–in points:** Let $G(s) = \frac{N(s)}{D(s)}$ We can solve either of the following equations for s:

$$\frac{dN(s)}{ds}D(s) - N(s)\frac{dD(s)}{ds} = 0$$

$$\sum \frac{1}{s - p_i} = \sum \frac{1}{s - z_i}$$

6. **Angle of departure/arrival** from a complex pole/zero s_p:

$$\phi_{\text{dep}} = 180^\circ - \Big(\sum_{\substack{\text{angles to } s_p \\ \text{from other poles}}} - \sum_{\substack{\text{angles to } s_p \\ \text{from zeros}}} \Big).$$

7. **Imaginary–axis crossing:** Apply the Routh–Hurwitz criterion, and plug in the K for marginal stability into the characteristic equation. There's an example on the next page!

5 Example Manual Root Locus (Hard)

Consider the open–loop transfer function

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

5.1 Pole–Zero Map

- Poles: $p_{1,2} = \pm 2j$, $p_3 = -3$.
- Zeros: $z_1 = -1$, $z_2 = +2$.
- $n = 3$, $m = 2 \Rightarrow \#p - \#z = 1$ asymptote.

5.2 Characteristic Equation

Close the loop with gain K:

$$1 + KG(s) = 0 \implies (s^2+4)(s+3) + 2K(s+1)(s-2) = 0$$

Expand and collect terms:

$$\underbrace{s^3 + 3s^2 + 4s + 12}_{1\times G(s)^{-1}} + 2K(s^2 - s - 2) = 0$$

Grouping like powers:

$$\big[1\big]s^3 + \big[3 + 2K\big]s^2 + \big[4 - 2K\big]s + \big[12 - 4K\big] = 0$$

The Routh array determines for which K the closed–loop poles lie in the left half–plane.

5.3 Routh–Hurwitz Stability Range

$$\begin{array}{c|cc} s^3 & 1 & 4-2K \\ s^2 & 3+2K & 12-4K \\ s^1 & a & 0 \\ s^0 & 12-4K & \end{array}$$

where $a = \dfrac{(3+2K)(4-2K) - (12-4K)}{3+2K} = \dfrac{6K - 4K^2}{3+2K}$. Positivity of the first column gives

$$1 > 0, \quad 3 + 2K > 0, \quad \frac{6K - 4K^2}{3+2K} > 0, \quad 12 - 4K > 0 \implies K > -\frac{3}{2}, \quad 0 < K < \frac{3}{2}, \quad 0 < K < 3$$

The tightest inequality satisfying all of these is our stability range:

$$\boxed{\text{stable system when } 0 < K < \frac{3}{2}}$$

5.4 Root–Locus Sketch

1. Start points (OL poles, $\times$): $\pm 2j$, -3.

2. End points (OL zeroes, o): -1, $+2$.

3. Real–axis segments: Based off the real-axis segments rule, they will be from $-\infty$ to -3 and from -1 to $+2$.

4. Infinite Branch / Asymptote: $\#p - \#z = 1$ branch to ∞. We have 3 poles and 2 zeroes, so our angle formula is:

$$\theta_a = \frac{(2k+1)\pi}{\#p - \#z} = (2k+1)\pi$$

   This means that we have one asymptote at $\theta_a = \pi$. We can intuit this being from the pole at -3 to $-\infty$, as it does not have a corresponding finite zero, and the real axis segment to the left of -3 is on the root locus.

5. Breakaway: Let's use the formula $\sum \frac{1}{s+p} = \sum \frac{1}{s+z}$, and solve with Wolfram Alpha!

$$\frac{1}{s+2i} + \frac{1}{s-2i} + \frac{1}{s+3} = \frac{1}{s+1} + \frac{1}{s-2} \implies s = 0.10692, s = 5.56$$

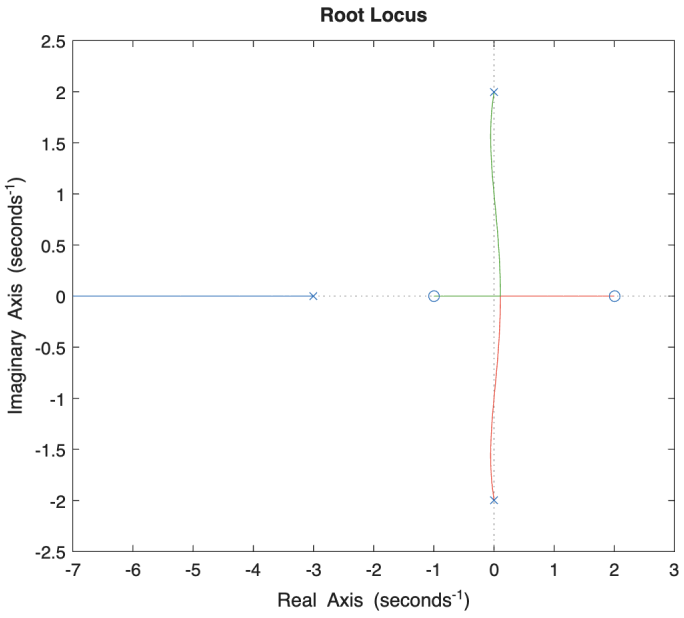
   We know our real axis segments are between -1 and 2, and -3 to ∞. The only value that lies in a range here is $\boxed{s = 0.10692}$, so this is our breakaway point!

6. Imaginary crossings: We know this occurs when the system is marginally stable, which occurs at $K = 1.5$. Plugging this into our characteristic equation gives us:

$$[1]s^3 + [3+2K]s^2 + [4-2K]s + [12-4K] = s^3 + 6s^2 + s + 6 = (s+6)(s^2+1) \implies s = -6, \pm j$$

   Our nonzero imaginary axis crossings are at $\boxed{s = \pm j}$!

Below is the plot of this transfer function, which has all the features that we have mentioned above!



6 HW6 - Design using Root Locus

This section will provide as an intro to designing systems with desired characteristics using root locus plots. For now, I'll briefly go over how this is accomplished, and what you can do with it.

6.1 Introduction

Typical system design targets such as *percent overshoot*, *settling time*, and *peak time* can be met by forcing the **dominant closed–loop poles** to approximate those of a *second–order system*. Remembering our second order poles, we get:

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

where

$$\omega_n = \sqrt{\sigma^2+\omega^2}, \qquad \zeta = -\frac{\sigma}{\omega_n}, \qquad \theta = \cos^{-1}\zeta.$$

Performance relations (2^{nd}-order model)

$$\%PO \approx e^{-\pi\zeta/\sqrt{1-\zeta^2}} \times 100\%,$$

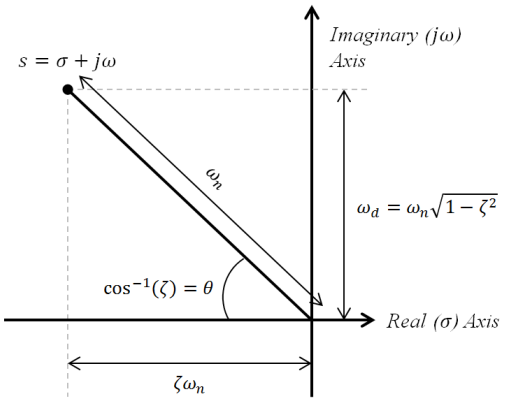
$$T_{\text{s}} \approx \frac{4}{\zeta\omega_n}, \qquad T_{\text{r}}\ (10\text{–}90\%) \ \approx \ \frac{1}{\omega_n}(1.768\zeta^3 - 0.417\zeta^2 + 1.039\zeta + 1),$$

$$\omega_{\text{d}} = \omega_n\sqrt{1-\zeta^2}.$$

Therefore, *specifications* $\{\%PO, T_{\text{s}}, T_{\text{r}}\}$ map to a *desired region* in the s-plane bounded by

$$\text{constant–}\zeta \ \ (\theta \text{ rays}) \quad \text{and constant–}\omega_n \ \ (\text{circles}).$$

Wikipedia's root locus page has a fantastic image for visualizing this! (see below)



From this image, we can gain a few important geometric insights:

- The angle that the pole makes with respect to the -real axis is defined by the damping coefficient ζ:

$$\zeta = \cos\theta$$

- The distance of the pole from the origin is its natural frequency ω_n:

$$r = \omega_n$$

6.2 Controllers & Compensators Overview

Assume our desired system characteristics define specific poles/regions in the s-plane, based off the definitions on the previous page.

Based off the rules for root locus, and the second-order approximations above, controllers and compensators add poles and zeroes such that the system's dominant poles now exist in favorable locations on the system's root locus! Once these points exist on the root locus, we choose the corresponding K as our gain and our system is going to follow our desired specifications.

Below is a brief outline of this design process:

1. **Plot design templates.** Draw $\theta = \cos^{-1}\zeta$ rays (damping-ratio lines) and ω_n circles that satisfy the transient specs.

2. **Plot the uncompensated/uncontrolled root locus.**

   - If it already passes through the desired region, adjust the gain K accordingly $\left(K = \frac{\prod|s-p_j|}{K_g\prod|s-z_i|}\right)$.
   - If not, *shape* the locus with compensators/controllers.

3. **Shape with compensators/controllers.**

   - **Lead Compensator/PD Controller** - $C(s) = \underbrace{K_d s + K_p}_{PD} \approx \underbrace{K_p\frac{s+z_c}{s+p_c}}_{\text{lead}}$.

     This adds $z_c >> p_c$ to help our desired pole s satisfy the **angle criterion**.

   - **Lag Compensator/ PI Controller** - $C(s) = \underbrace{K_p + \frac{K_i}{s}}_{PI} \approx \underbrace{K_p\frac{s+z_c}{s+p_c}}_{\text{lag}}$.

     This adds p_c, z_c with $p_c >> z_c$. The intent here is to increase the type of the system and **reduce steady state error** without affecting the position of the desired pole on the RL.

   - **Lag–Lead/PID**

     This controller cascades the two controllers above to simultaneously achieve desired transient and steady-state characteristics!

4. **[Mandatory for PID/Lag-Lead]: Re-plot and iterate** until the dominant compensated poles lie inside the desired template region and all other poles reside sufficiently left to be non-dominant ($\mathrm{Re}\{s_{\mathrm{nd}}\} < -5/T_{\mathrm{s}}$ is a common rule).

5. **Validate** in the time domain (step response) and frequency domain (Bode plot) to ensure the second-order approximation remains adequate after full closed-loop dynamics are included. If this isn't satisfactory, return to iteration.

The last two steps are necessary due to us approximating the system poles as second order during the whole system design process.

6.3 Controller/Compensator Design Examples (no `sgrid()`)

For the purposes of this subsection, I will *not* be using the function `sgrid()`, due to it being unstable and not working for many people. My discussion notes cover a manual method of solving this, but for these examples, I will be using **an alternative set of functions** that seem to (in my limited testing) **only work on `.m` files. For me, this method did not work with `.mlx` files**.

6.3.1 PI

We will be going over the following example:

32. Design a PI controller to drive the step-response error to zero for the unity feedback system shown in Figure P9.1, where

$$G(s) = \frac{K}{(s+1)(s+3)(s+10)}$$

The system operates with a damping ratio of 0.5.

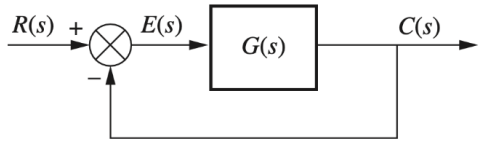


FIGURE P9.1

Remember that a PI controller (of the form $\frac{K(s+z_c)}{s}$) removes steady state error by **increasing the type of the system by adding a pole at the origin, while minimally affecting the root locus.** To accomplish this, we pick a zero z_c close enough to the origin, and calculate the gain!

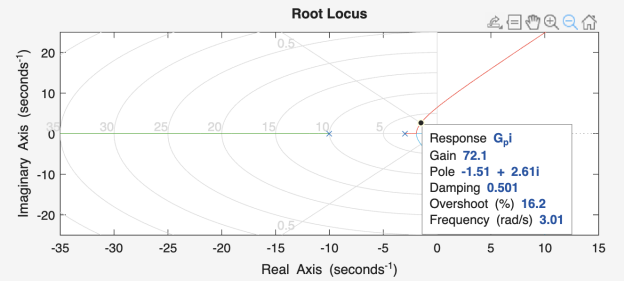
A common value for z_c is ≈ 0.1, but feel free to use slightly smaller values. A really small zero would be prone to noise and make the system sensitive, although this topic is out of scope.

We can set up our plant, add our zero, and check the root locus with the following code:

```
s = tf('s');
G = 1/((s+1)*(s+3)*(s+10)); %OL TF

zeta = 0.5

z_pi = -0.1; %PI zero, chosen to be close to origin, with minimal change to RL
G_pi = G*(s-z_pi)/s; %PI controlled plant, no K

figure
rlp = rlocusplot(G_pi)
rlp.AxesStyle.GridVisible = "on";
rlp.AxesStyle.GridDampingSpec = zeta; %zeta requirement shown on plot
```

[plot on following page]

We can select the intersection of the root locus with our desired damping spec line ($\zeta = 0.5$) to find our desired pole location!



As we can see, our desired pole is $s_0 = -1.61 + 2.61i$. We can calculate K using the following:

$$K = \frac{\prod |s - p_i|}{\prod |s - z_i|} = \frac{1}{|G(s_0)|}$$

Putting this into code, we get:

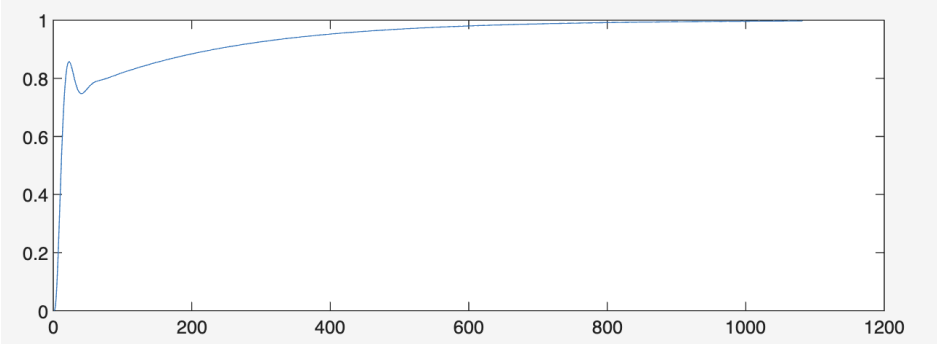
```
s0 = -1.51+2.61i;
K = 1/abs(norm(evalfr(G_pi, s0)))
T = K*G_pi
```

From this code, we get $K = 72.16$, meaning our final controller is:

$$\boxed{\frac{72.16(s + 0.1)}{s}}$$

We can then check our final specs with the following!

```
T_cl = feedback(T, 1)
damp(T_cl)
plot(step(T_cl))
```



Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-7.28e-02	1.00e+00	7.28e-02	1.37e+01
-1.51e+00 + 2.61e+00i	5.00e-01	3.01e+00	6.63e-01
-1.51e+00 - 2.61e+00i	5.00e-01	3.01e+00	6.63e-01
-1.09e+01	1.00e+00	1.09e+01	9.16e-02

6.3.2 PD

We will be going over the following example:

6. The unity feedback system shown in Figure P9.1 with

$$G(s) = \frac{K(s+6)}{(s+2)(s+3)(s+5)}$$

is operating with a dominant-pole damping ratio of 0.707. Design a PD controller so that the settling time is reduced by a factor of 2. Compare the transient and steady-state performance of the uncompensated and compensated systems. Describe any problems with your design. [Section: 9.3]

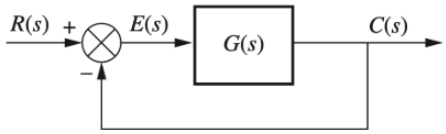


FIGURE P9.1

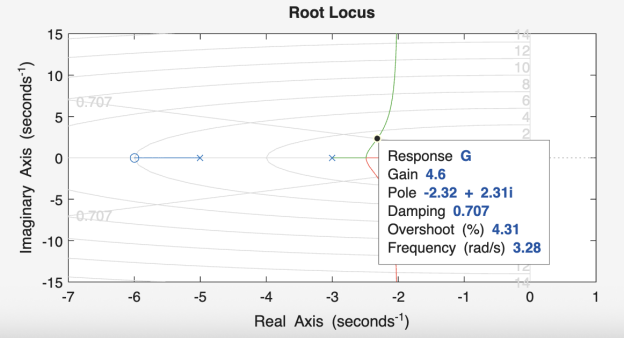
The core idea of PD controllers (of the form $K(s + z_{PD})$ is as follows:

1. Find the position of our desired closed loop poles, given the transient specifications given.
2. We want to place this pole on the root locus by adding a zero such that it satisfies the angle rule:
$$\sum_j \angle(s - z_j) - \sum_i \angle(s - p_i) = (2k + 1)\pi,\ k \in \mathbb{Z}$$
To do this, we must calculate the current angle contribution at the desired pole, $(\sum_j \angle(s - z_j) - \sum_i \angle(s - p_i))$ and find how much angle we need to "add" with the zero to satisfy the rule.
3. Solve for K at the desired pole once our zero is added!

We are given that $\zeta = 0.707$, and we want to halve the settling time. To accomplish this, we first need to find the locations of the **current dominant closed loop poles**. To do this, we look at the intersection of the $\zeta = 0.707$ line with the plant's root locus:

```
s = tf('s');
G = (s+6)/((s+2)*(s+3)*(s+5)); %OL TF

zeta = 0.707
figure
rlp = rlocusplot(G)
rlp.AxesStyle.GridVisible = "on";
rlp.AxesStyle.GridDampingSpec = zeta; %zeta requirement shown on plot
```

We can see that our current closed-loop poles are at $s_0 = -2.32 \pm 2.31i$.



To ensure that our settling time is halved, we need to halve our time constant ($\frac{1}{\omega_n \zeta}$), which means **doubling the real component of the dominant pole ($\omega_n \zeta$)**.

As a result, we know we must place our pole s_0 such that the real component is -4.64, and lies on the line $\zeta = 0.707$. Since, the $\zeta = 0.707$ line is approximately the $45°$ line, we can say our real and imaginary components must be equal and our desired pole is:

$$s_0 = -4.64 + 4.64i$$

Next, let's calculate the current angle contribution at the desired pole s_0:

$$\sum \angle(s - z_i) - \sum \angle(s - p_i) = \angle G(s_0)$$

We can do this with the following code:

```
s0 = -4.64 + 4.64i; %desired pole
angle_zp = angle(evalfr(G, s0)); %sum of angles from zero - sum of angles from pole
```

We can then calculate the angle deficiency (mod implementation out of scope) and zero position!

```
angle_contribution = -(mod(angle_zp, 2*pi)-pi); %the needed additional angle from z_{pd}
z_pd = real(s0) - imag(s0)/tan(angle_contribution)
```

We get a zero position of -7.21! Finally, we can calculate our desired K for the PD-controlled system, using the same gain criterion from the previous example:

```
T = G * (s-z_pd); %PD controlled plant, without K
K = 1/abs(norm(evalfr(T, s0)))
```

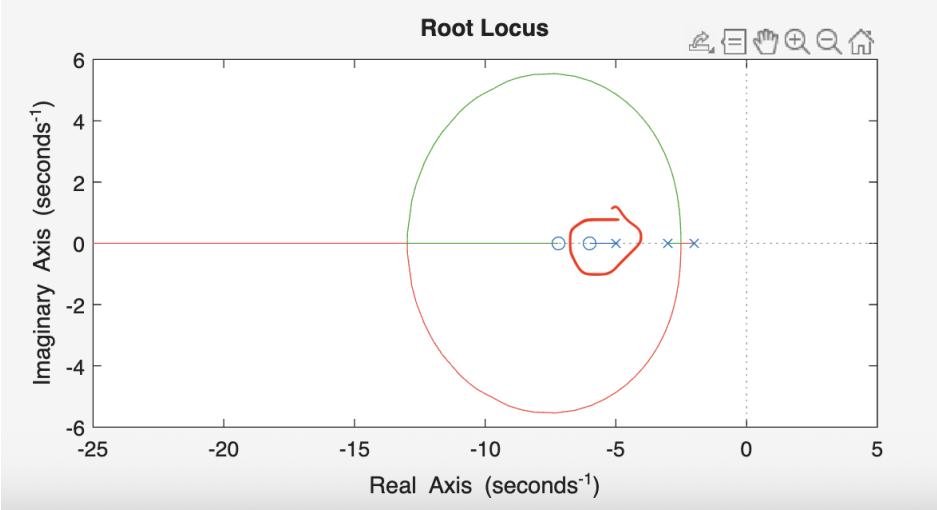
We get $K = 4.7667$, and we can fully write out our controller now!

$$\boxed{4.7667(s + 7.21)}$$

Checking our specs:

```
T_cl = feedback(K*T, 1); %CL transfer function
Ts_old = 4/2.32
Ts_PD = stepinfo(T_cl).SettlingTime
damp(T_cl)
```

: Running this, we see our damping spec is met ($\zeta = 0.707$), but our T_s spec is not quite there! This discrepancy is due to the our second order approximation being not quite valid:



The presence of the pole on the finite branch in red ensures that there is a pole close to our desired dominant poles at $-4.64 \pm 4.64i$, regardless of K! As a result, our second-order approximation is not truly valid, and we will have to iterate to match our actual spec. We will go over an example of this iteration process in the PID example!

This area was intentionally left blank for formatting reasons.
The **PID** example is on the following page.

6.3.3 PID

We will be going over the following example:

25. For the unity feedback system in Figure P9.1, with

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

design a PID controller that will yield a peak time of 1.122 seconds and a damping ratio of 0.707, with zero error for a step input. [Section: 9.4]

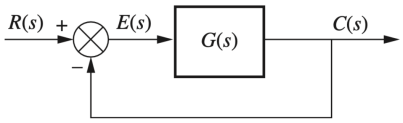


FIGURE P9.1

The general procedure for designing PID/lag-lead system is as follows:

1. Add a PI controller to the system to remove the steady state error.
2. Design a PD controller for the PI compensated system to meet the transient specifications.
3. Test to see if it meets the specifications, and adjust values of the controller as necessary.

As a note, steps 1 and 2 are *interchangeable*. Let's do one step at a time:

1. **PI Controller:** For now, we can simply have our PI zero at $z_{pi} = -0.1$ since all we need to do is eliminate steady state error. Let's write this in matlab!

```
s = tf('s');
G = 1/(s+1)/(s+3); %OL TF
zeta = 0.707;
Tp = 1.122;

z_pi = -0.1; %sufficiently far away from other OL poles and zeroes and close to
    origin!
G_pi = G*(s-z_pi)/s; %PI controlled plant (no K)
```

2. **PD Controller:** Our desired specifications are $T_p = 1.122, \zeta = 0.707$. We can solve for our desired natural frequency specification by rearranging our equation for T_p:

$$T_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \implies \omega_n = \frac{\pi}{T_p\sqrt{1-\zeta^2}} = 3.9592$$

This means that we require the poles to be at:

$$s_{1,2} = -\omega_n\zeta \pm \omega_n\sqrt{1-\zeta^2}j \approx -2.8 \pm 2.8j$$

We can shortcut to the desired phase contribution using `angle(evalfr(G, s))`! This will give us $\sum \angle(s - z_i) - \sum \angle(s - p_i)$. Let's write all of our work so far in MATLAB !

```
s = tf('s');
G = 1/(s+1)/(s+3); %OL TF
zeta = 0.707;
Tp = 1.122;
wn = pi/(Tp*sqrt(1-zeta^2));
s1 = -wn*zeta + wn*sqrt(1-zeta^2)*i; %desired CL poles!
curr_angle = angle(evalfr(G, s1)); %find current total angle at desired pole
angle_deficiency = (mod(curr_angle, 2*pi)-pi); %needed extra angle
```

We can then calculate the location of the zero, using a triangle with the desired pole and the angle we just solved for:

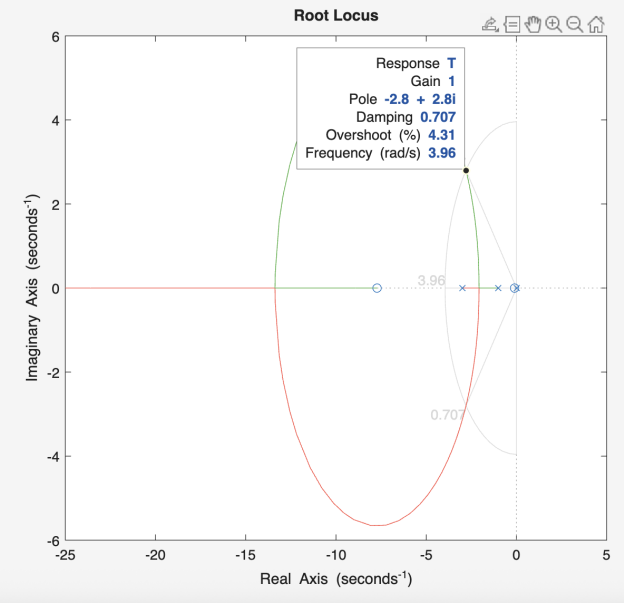
```
z_pd = real(s1)-imag(s1)/tan(angle_contribtution) %remember that tan(angle) = imag(
    s1)/(real(s1)-z_pd)
```

We can now define our controlled system and calculate K.

```
ControlledG = G_pi*(s-z_pd) %open-loop PID controlled plant form without K
K = 1/abs(norm(evalfr(ControlledG, s1))) %distance to poles/distance to zeroes
T = K*ControlledG %controlled plant OL tf with K!
```

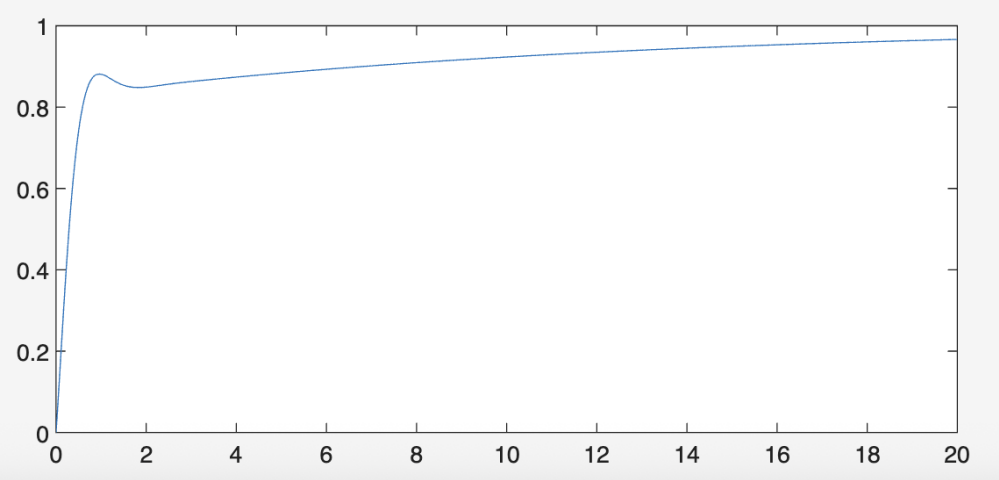
We can then see that our root locus passes through the desired poles, and satisfies our spec given a second order approximation!

```
figure
rlp = rlocusplot(T);
rlp.AxesStyle.GridVisible = "on";
rlp.AxesStyle.GridDampingSpec = zeta; %plots damping spec
rlp.AxesStyle.GridFrequencySpec = wn; %plots frequency spec
```



3. **Iteration:** Unfortunately, our real response isn't *exactly* where we want it, due to us using a second order approximation throughout our design for both controllers.

```
T_cl = feedback(T, 1);
[y, t] = step(T_cl, [0, 20]);
[peakval, idx] = max(y(1:50, :)); %matlab peaktime will return the last value in the
    system, due to the peak being < 1. this checks the very start of the system!
Tp_pid = t(idx)
damp(T_cl)
plot(t, y)
```




```
Tp_pid =

    0.9539
```

Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-8.27e-02	1.00e+00	8.27e-02	1.21e+01
-2.80e+00 + 2.80e+00i	7.07e-01	3.96e+00	3.57e-01
-2.80e+00 - 2.80e+00i	7.07e-01	3.96e+00	3.57e-01

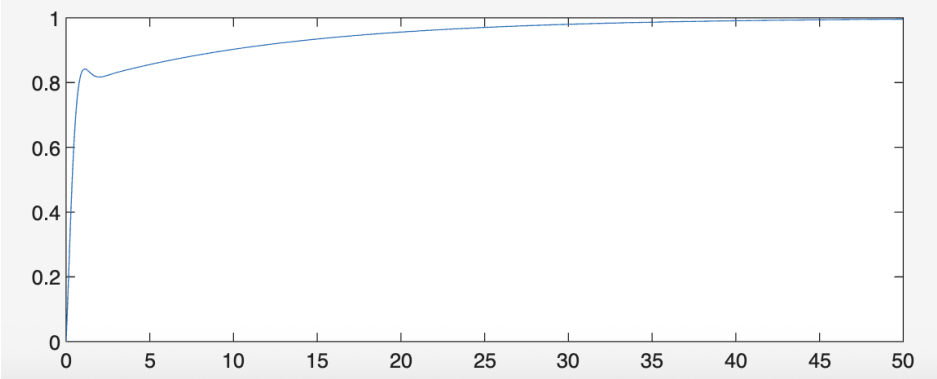
While our damping is spot on, our peak time spec isn't quite right! Since it is too fast, we must *decrease* our ω_n spec!

This area was intentionally left blank for formatting reasons.
The rest of the example is on the following page.

With this, you can adjust your ω_n spec, check the resulting peak time, and adjust again as necessary. I ended up decreasing it by 0.3605. Note that past a specific point, Matlab (by default) does not have enough decimal precision to account for tuning, so your answers may be within ± 0.05!

```
Tp_pid =

    1.1224
```

Pole	Damping	Frequency (rad/seconds)	Time Constant (seconds)
-7.90e-02	1.00e+00	7.90e-02	1.27e+01
-2.54e+00 + 2.55e+00i	7.07e-01	3.60e+00	3.93e-01
-2.54e+00 - 2.55e+00i	7.07e-01	3.60e+00	3.93e-01



Checking our values of K, z_{PD}, and z_{PI}, we get our controller TF as:

$$\boxed{\frac{1.1676(s+0.1)(s+8.7668)}{s}}$$

This area was intentionally left blank for formatting reasons.
The **Full PID Code** is on the following page.

PID Code

```
s = tf('s');
G = 1/(s+1)/(s+3); %OL TF
zeta = 0.707;
Tp = 1.122;

z_pi = -0.1; %sufficiently far away from other OL poles and zeroes and close to origin!
G_pi = G*(s-z_pi)/s; %PI controlled plant (no K)

wn = pi/(Tp*sqrt(1-zeta^2))-0.3605;
s1 = -wn*zeta + wn*sqrt(1-zeta^2)*i %desired pole
curr_angle = angle(evalfr(G_pi, s1)) %find current total angle at desired pole
angle_contribution = -(mod(curr_angle, 2*pi)-pi) %needed extra angle
z_pd = real(s1)-imag(s1)/tan(angle_contribution) %PD zero location!

ControlledG = G_pi*(s-z_pd) %open-loop controlled plant form without K
K = 1/abs(norm(evalfr(ControlledG, s1))) %distance to poles/distance to zeroes
T = (K)*ControlledG %controlled plant OL tf with K!


figure
rlp = rlocusplot(T)
rlp.AxesStyle.GridVisible = "on";
rlp.AxesStyle.GridDampingSpec = zeta;
rlp.AxesStyle.GridFrequencySpec = wn;

T_cl = feedback(T, 1);
[y, t] = step(T_cl, [0, 50]);
[peakval, idx] = max(y(1:50, :));
Tp_pid = t(idx)
damp(T_cl)
plot(t, y)
```