EENG307: Designing Controllers Using Bode Plots, Part 2*

Lecture 34

Elenya Grant, Kathryn Johnson, and Hisham Sager†

Fall 2022

Contents

1 Pre-requisite Material

This lecture assumes that the reader is familiar with the following material:

- Lecture 31: Time/Frequency Relationships and Control Performance Measures
- Lecture 32: Designing Controllers Using Bode Plots, Part 1

Note: this article makes heavy use of Matlab's "Control System Designer" (formerly `sisotool`) and we recommend that you open that tool and follow along with the article there for full benefit. The two sections use different versions of the tool.

2 Analysis of PD Control

Let's return to the control design problem from 32: Designing Controllers Using Bode Plots, Part 1.

Consider the following system:

$$G(s) = \frac{5}{s(s+1)(s+10)}$$

We would like to design a controller so that the feedback system has the following desirable characteristics:

- Zero steady state error to a step input

†Developed and edited by Tyrone Vincent and Kathryn Johnson, Colorado School of Mines, with contributions from Salman Mohagheghi, Chris Coulston, Kevin Moore, CSM and Matt Kupilik, University of Alaska, Anchorage

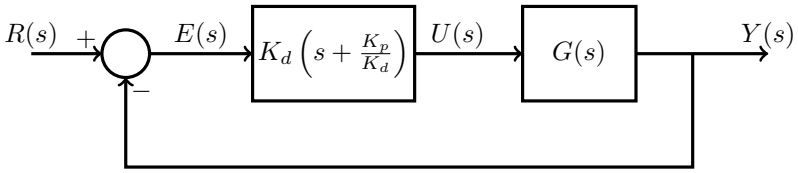
- Rise time of 1.1 s
- % overshoot $\leq 16.5\%$.

We found that these time domain specifications gave rise to the following frequency domain specifications:

- Loop gain contains an integrator (low frequency gain goes to infinity)
- Crossover $\omega_{co} = 2$ rad/s
- Phase margin $\phi_{PM} = 50°$

We were unable to meet these specifications using proportional control alone. In particular, we had a very oscillatory response with large %OS. Let's see if we can make more headway using a PD controller, which adds a zero and can therefore increase the phase margin ϕ_{PM}. Recall our PD design block diagram

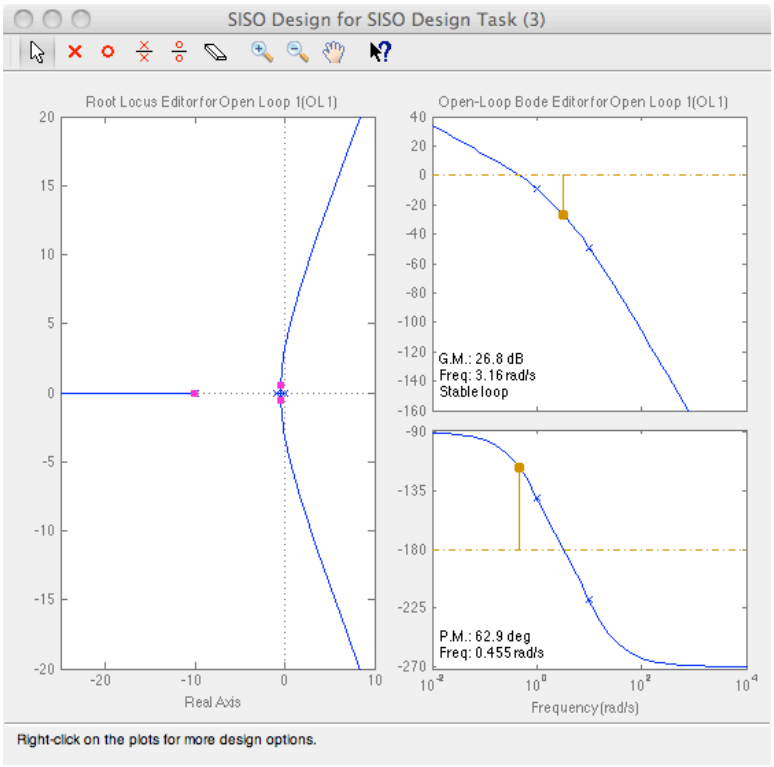
PD Control



In addition to changing the gain at all frequencies by design of K_d, the PD controller allows us to place a zero in the loop gain via the selection of K_p/K_d. Let's start the design process by entering the open loop transfer function $G(s)$, loading `sisotool`, importing $G(s)$, and opening the design plot. A quick way to do this is the following:

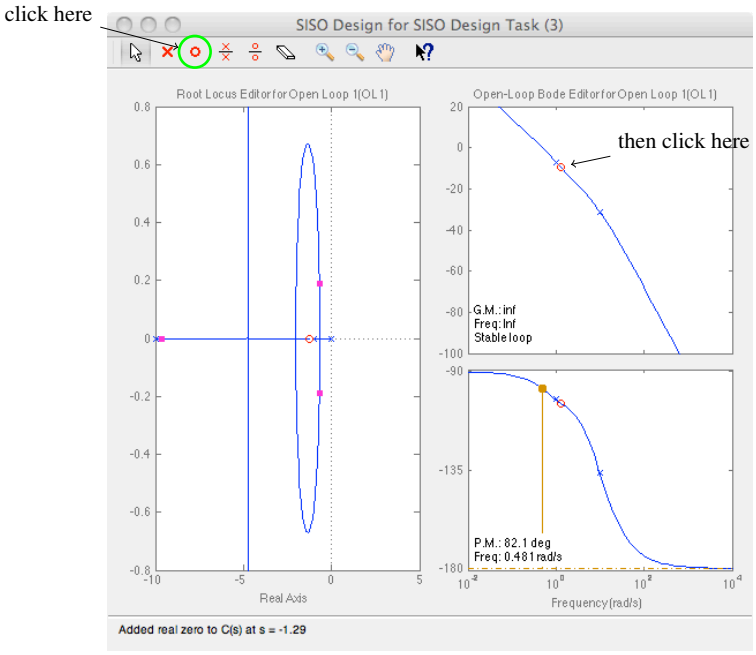
```
s = tf([1 0],1);
G = 5/(s*(s+1)*(s+10));
sisotool(G)
```

The design plot figure should look as follows:



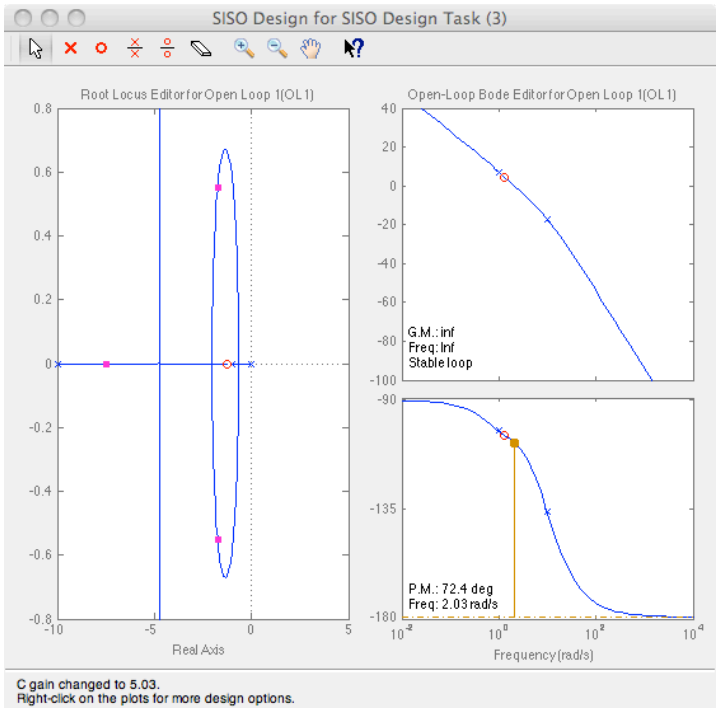
Before, when we increased the gain so that the crossover frequency was $\omega_{co} = 2$ rad/s, we ran into problems because the phase margin was poor. We can improve the phase margin by adding a zero near the crossover frequency. Recall from Lecture 27: Understanding Bode Plots Using Matlab that the Bode plot for a LHP zero term has a phase that goes from $0°$ to $90°$ around the zero's break frequency. Remember also that properties of the `log` and $\angle$ functions mean that Bode plots for different terms can be added together, so by adding this LHP zero we can make the net phase response more positive.

Let's place a zero just before 2 rad/s. To do this, we can click on the zero icon of the design figure circled in green in the figure below. Although we can edit this design in either the root locus or the Bode plot, since this article is about Bode plots we'll use that one. In the Bode plot figure, click on the magnitude plot around 1 rad/s. The result should look like the figure below. If you misplace the zero, you can put the cursor over the zero until it looks like a hand, and the left click to grab the zero and move it right (moving the zero at $s = -\frac{K_p}{K_d}$ to a higher frequency) or left (lower frequency).



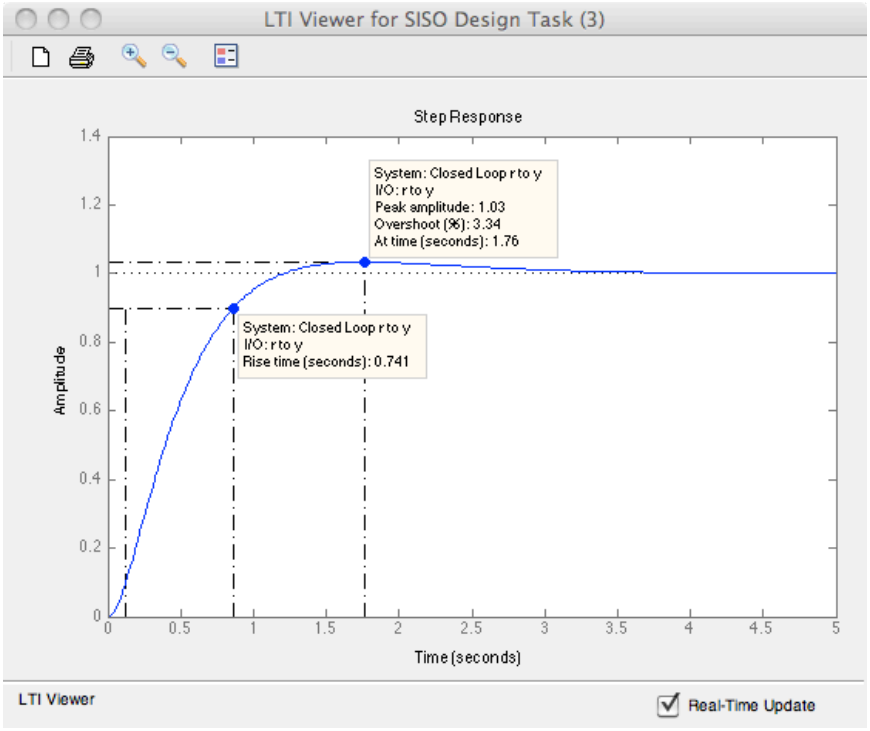
Note that the placement of the zero has changed both the Bode plot and the root locus plot. For the root locus interpretation, refer back to Lecture 19: PD Design Using Root Locus and SISOtool. The phase is higher than for the plant $G(s)$ alone before we added the zero (see Lecture 32) because of the control zero we added.

Next, use the "hand" tool (hover over the magnitude Bode plot until it appears) to move the magnitude plot up until the crossover is 2 rad/s. This change of gain is equivalent to increasing the derivative gain K_d in the "PD Control" block diagram earlier in this lecture. This action results in the following plot:



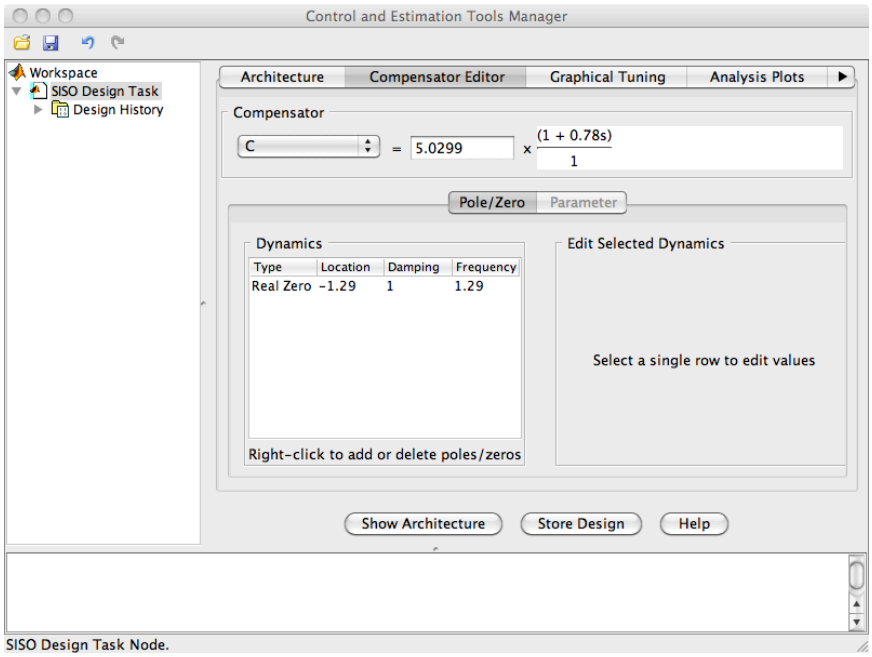
Note that the crossover is 2.03 rad/s (as noted in the lower left corner of the phase plot), and the phase margin is 72.4°. These values of $\omega_{co,G}$ and $\phi_{PM,G}$ meet our specifications from the beginning of this section, so our step response should be acceptable. However, since we have used approximations in the design process (see Lecture 31), we always need to verify our design by plotting the step response.

By clicking on the menu item "Analysis -> Response to Step Command" and right clicking on the plots to show the rise time and peak response, we get the following plot:



To recover our controller, we go back to the "Control and Estimation Tools Manager", and click on "Compensator

Editor."



We see that the controller has the form

$$C(s) = 5.0299(1 + 0.78s)$$

or, multiplying through,

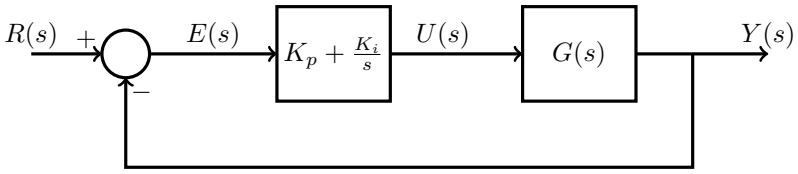
$$C(s) = 5.0299 + 3.923s$$

Thus, we should pick $K_p = 5.0299$ and $K_d = 3.923$

3 Analysis of PI Control

From our analysis of steady state error, we know that the system type[1] of the loop gain plays a key role. In particular, if zero steady state error for a step input is required, then the system Type must be 1 (or higher). If the plant to be controlled, $G(s)$, does not have a pole at $s = 0$, then this must be supplied by the controller. A Proportional-Integral (PI) controller can do this and has the following structure:

PI Control



With a little bit of algebra (combining the controller terms over a common denominator and factoring out K_p), the loop gain is thus

$$L(s) = \frac{K_p\left(s + \frac{K_i}{K_p}\right)}{s} G(s)$$

[1] Recall that system Type is the number of poles at $s = 0$, also called the number of pure integrators

Note that the controller has a pole at $s = 0$, and a zero at $s = K_i/K_p$. If we move the integrator over with $G(s)$, we can write the loop gain as

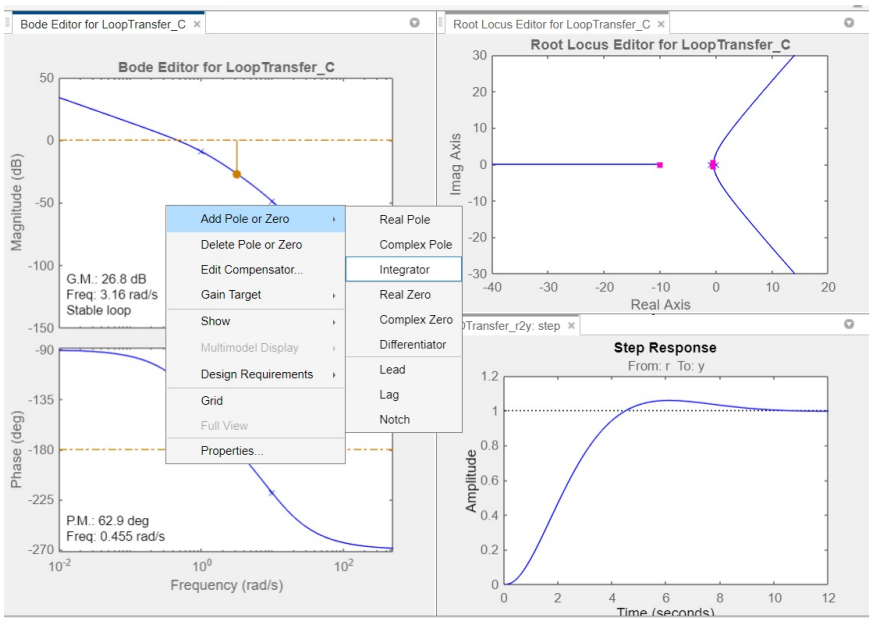
$$L(s) = K_p \left(s + \frac{K_i}{K_p} \right) \frac{G(s)}{s}$$

This looks very much like a PD controller applied to a new "augmented" plant $G(s)/s$! The control design can thus be approached in the same way as for PD control from Section 2, except using the augmented plant $G(s)/s$, rather than the original $G(s)$.

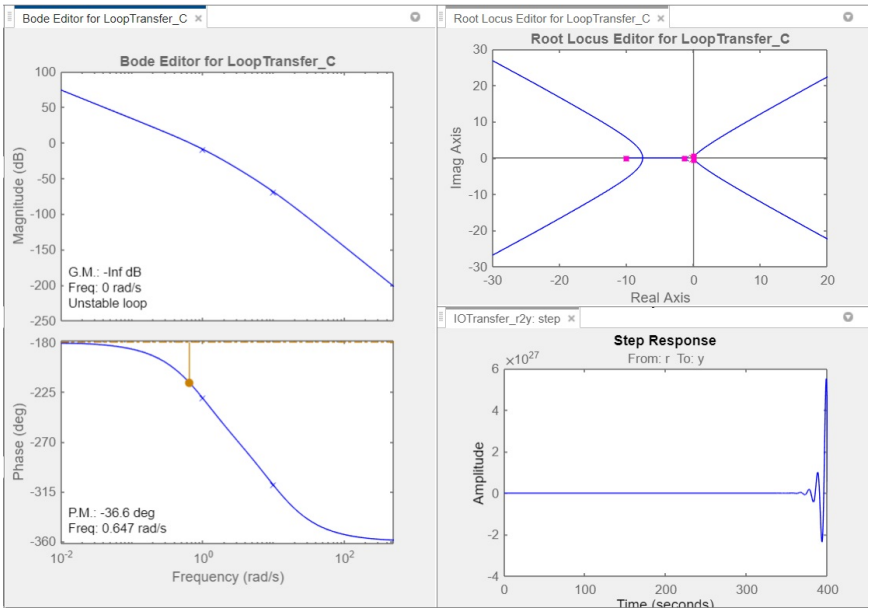
Let's revisit the example from Section 2. Since this example was already Type 1 (from the plant $G(s)$) and would therefore not illustrate the PI control design process, we'll update our specifications to be

- stable closed-loop
- zero steady-state error to a *ramp* input $r(t) = tu(t)$
- $\%OS \leq 16.5\%$ for a step input $r(t) = u(t)$
- rise time of 1.1 s

To achieve the new steady-state error requirement for the ramp input, we need a Type 2 system, which means we need to add an integrator via the PI controller, as shown in the figure below (menu accessed via right click in the Bode plot).

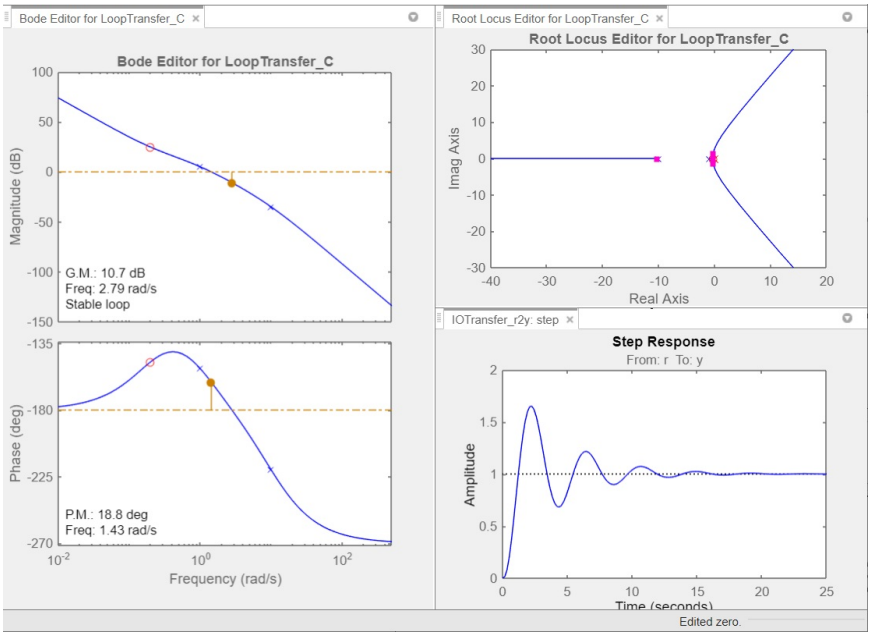


After adding this new integrator, our system is now unstable in the closed-loop!



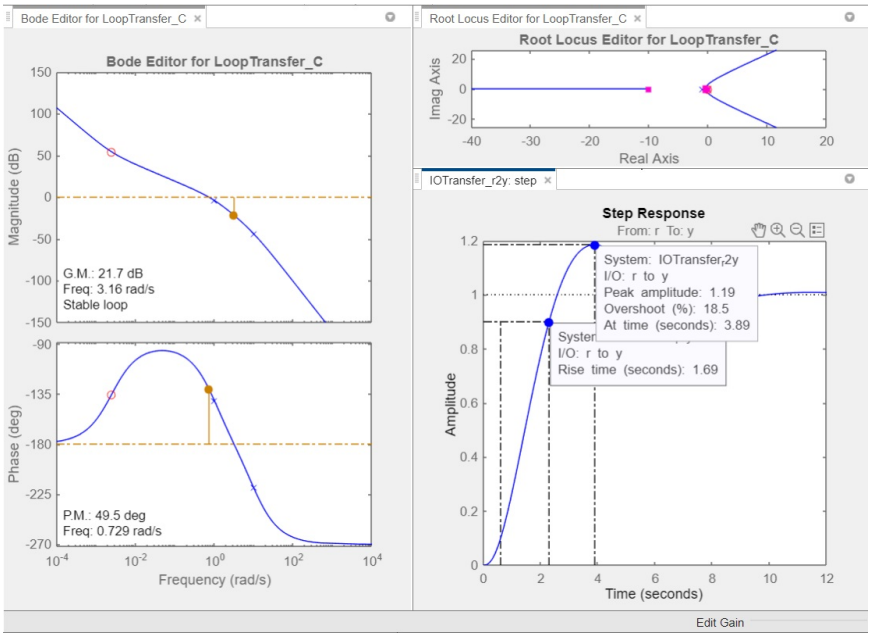
The reason for this instability is that the integrator we added caused the phase Bode plot to start at -180° deg and never rise above it, so there can never be a positive phase margin no matter how we change the gain. We must therefore incorporate the controller zero at $s = -\frac{K_i}{K_p}$ to have any hope of achieving our specifications. Recall that a LHP zero will cause the phase angle to increase.

Again by right clicking on the Bode plot in `sisotool`, we can add a real LHP zero. Recalling that poles and zeros impact the phase plot from one decade below to one decade above their break frequencies, let's try adding it at around one decade below the desired crossover frequency, or 0.2 rad/s.



As you can see, we have successfully met the closed-loop stability requirement, but not yet the percent overshoot requirement (steady-state error and rise time not shown). The phase margin $\phi_{PM} = 18.8^\circ$, not the required 50°. Matlab's `sisotool` allows us to experiment with controller tuning to change our phase margin and crossover frequency. As we move the controller zero to the left (lower frequency), we can increase the amount by which the phase plot rises above -180°, which gives us more opportunity to achieve our desired phase margin. However, we notice a tradeoff: the peak phase occurs at lower frequencies, which makes it harder to achieve our crossover frequency requirement to achieve the desired rise time.

We can adjust the location of the controller zero (at $-\frac{K_i}{K_p}$) and gain (K_p) to try to achieve a balance between our rise time and percent overshoot requirements. One example that doesn't quite meet either is shown below.

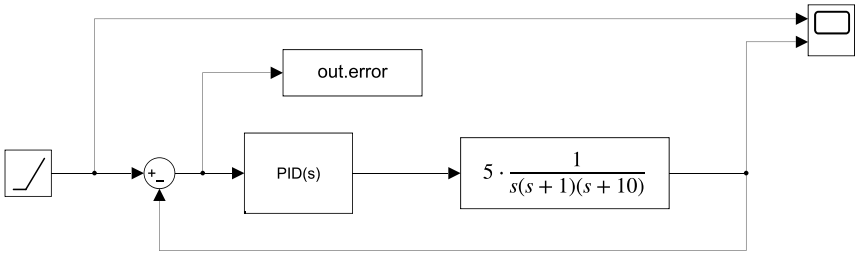


Matlab tells us in the lower left corner of this version of the Control System Designer (or by double-clicking on "C") that the controller is

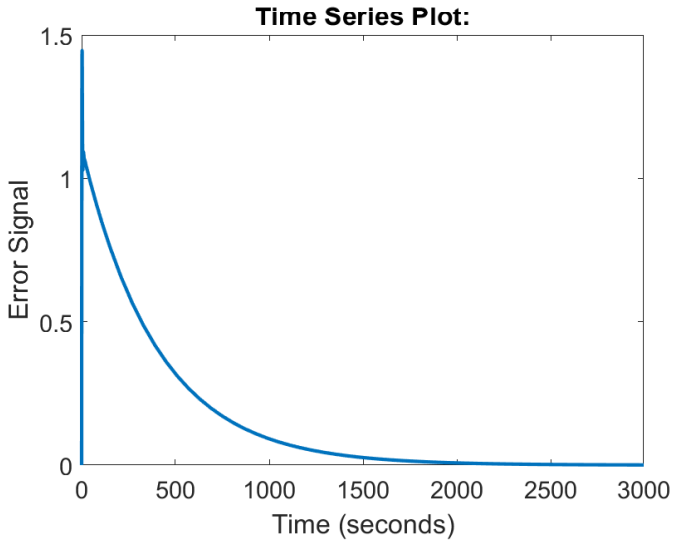
$$\frac{K_p\left(s+\frac{K_i}{K_p}\right)}{s} = \frac{1.81\,(s+0.0025)}{s}$$

and therefore $K_p = 1.81$ and $K_i = 0.0025K_p \Rightarrow K_i = 0.0045$.

We also need to check our steady-state error results. Since `sisotool` doesn't have a built-in ramp input, we can quickly check this in Simulink using the configuration shown below.



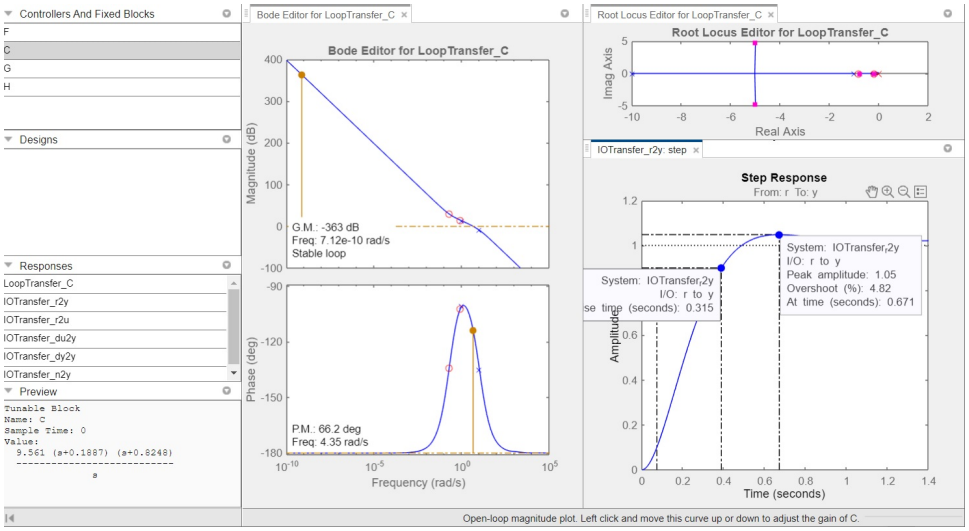
The steady-state error does approach zero but very slowly, so this is not a great controller design overall.



The very slow response is related to the slower-than-desired natural frequency as well as to the fact that we don't have a standard second-order system, so all of our design tools are approximations.

4 Alternative Controller Architectures

There are many types of controllers beyond PID. Within `sisotool`, it is easy to add additional poles and zeros of the controller to achieve desired objectives. Since our issue in Section 3 was largely that we struggled to achieve enough phase margin, we could quickly add a second controller zero to increase the phase further, as illustrated in the plot below, which has characteristics $t_r = 0.3$ s and $\%OS = 4.8\%$.



In the lower left pane, you can see that this controller is of the form

$$C(s) = \frac{K\,(s + z_1)\,(s + z_2)}{s}$$

which has a tunable gain K and two tunable zeros z_1 and z_2. This controller structure does not have a name and can be problematic to implement in the "real world" (since it is not proper), but does make it easier to meet our objectives from a theoretical sense.

If you are interested in tuning controllers beyond the structures we have discussed in this class, two other common types are *lead* and *lag* compensators, which have the form

$$C(s) = K\frac{s+z}{s+p}$$

and can be found in many control systems tutorials and textbooks. They both have a tunable gain K and tunable zero z and pole p, and the main difference is whether the pole has a lower frequency than the zero (in which case the magnitude and phase Bode plots start to drop before they start to rise) or the zero at a lower frequency than the pole (in which case the magnitude and phase Bode plots start to rise first, then drop), assuming both the pole and zero are in the LHP.

5 Lecture Highlights

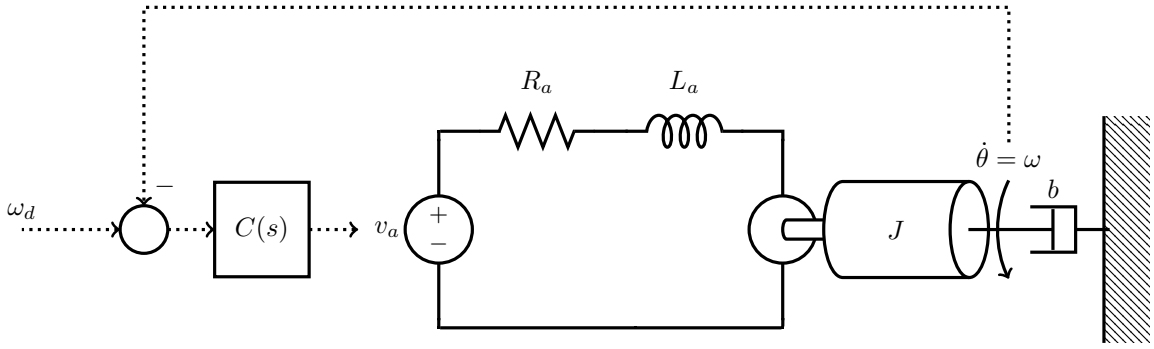
The primary takeaways from this article include

1. the illustrated step-by-step design of a proportional derivative (PD) controller for an example system using frequency response control design techniques and Matlab's `sisotool`.
2. the illustrated step-by-step design of a proportional integral (PI) controller for an example system using frequency response control design techniques and Matlab's `sisotool`.
3. a brief discussion of controller architectures beyond PID

6 Quiz Yourself

6.1 Questions

1. A motor is used to rotate an inertia to a desired speed. A sensor monitors the angular velocity of the inertia, and a feedback control system is implemented by setting the motor voltage as $V_a(s) = C(s)(\omega_d(s) - \omega(s)y)$ where ω_d is the desired angular velocity of the inertia. The dotted lines indicate the information flow that implements this controller. The component parameters are $R_a = 1$, $L_a = 1$, $J = 2$, $b = 4$, $K_t = 3$, $K_e = 4$.



   (a) Find the transfer function from v_a to ω. Call this transfer function $G(s)$. Draw the block diagram of the control system in terms of $C(s)$ and $G(s)$.

   (b) Use MATLAB to plot the bode plot of $G(s)$.

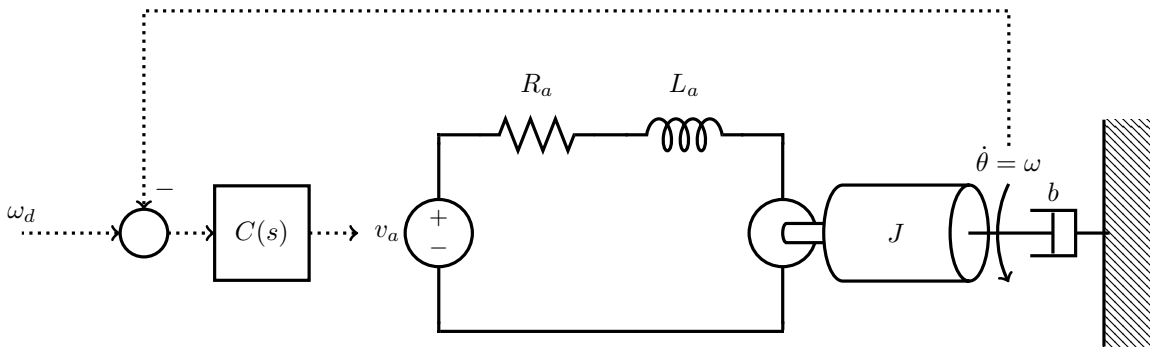
   (c) Design a PD compensator to meet the following specifications:

   - low frequency gain > 20 dB
   - cross-over frequency > 5 rad/s
   - phase margin $\phi_{PM} > 45^\circ$.

   (Hint: Choose K so that both the steady state requirement and the bandwidth requirement are met - one of them may be exceeded.) Using Matlab, plot the frequency response of both your resulting controller $C(s)$ and the loop gain $C(s)G(s)$.

(d) Use MATLAB to simulate the step response of the closed loop system $C(s)G(s)/(1+C(s)G(s))$.

2. A motor is used to rotate an inertia to a desired speed. A sensor monitors the angular velocity of the inertia, and a feedback control system is implemented by setting the motor voltage as $V_a(s) = C(s)(\omega_d(s) - \omega(s))y)$ where ω_d is the desired angular velocity of the inertia. The dotted lines indicate the information flow that implements this controller. The component parameters are $R_a = 1$, $L_a = 1$, $J = 2$, $b = 4$, $K_t = 3$, $K_e = 4$.



(a) Find the transfer function from v_a to ω. Call this transfer function $G(s)$. Draw the block diagram of the control system in terms of $C(s)$ and $G(s)$.

(b) Design a PI compensator to meet the following specifications:

- infinite low frequency gain
- cross-over frequency > 4 rad/s
- phase margin $> 45^\circ$.

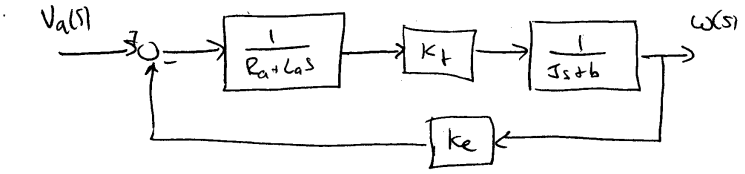
Using Matlab, plot the frequency response of both your resulting controller $C(s)$ and the loop gain $C(s)G(s)$.

(c) Use MATLAB to simulate the step response of the closed loop system $C(s)G(s)/(1+C(s)G(s))$.

6.2 Solutions

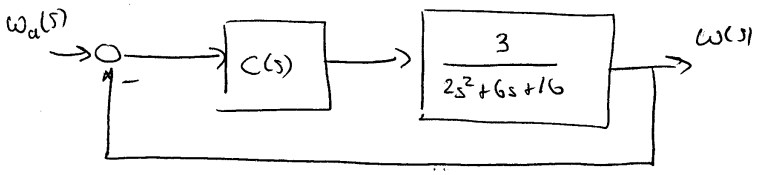
1.

(a)



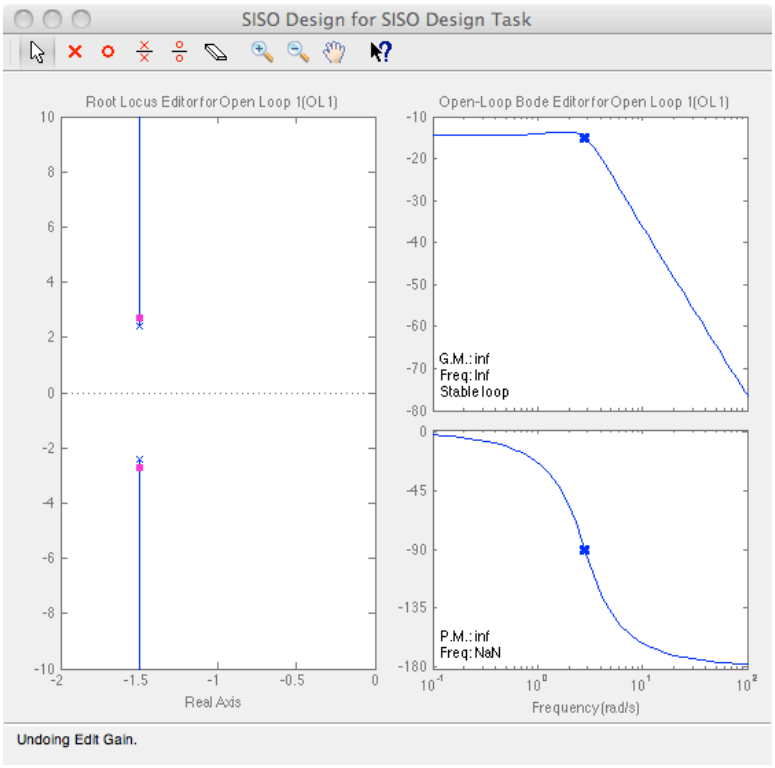
$$\frac{\omega(s)}{V_a(s)} = \frac{\frac{K_t}{(R_a + L_a s)(Js + b)}}{1 + \frac{K_t K_e}{(R_a + L_a s)(Js + b)}} = \frac{K_t}{JL_a s^2 + (JR_a + L_a b)s + R_a b + K_t K_e}$$

$$= \frac{3}{2s^2 + 6s + 4 + 12}$$

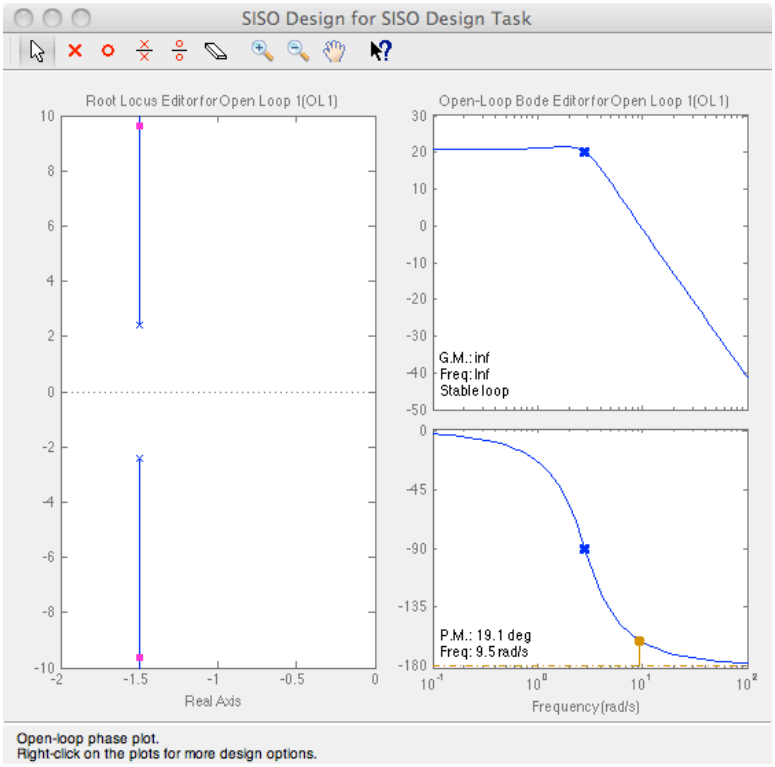


(b) We will use `sisotool` to plot the Bode plot of $G(s)$.

```
» G = tf(3,[2 6 16])
» sisotool(G)
```

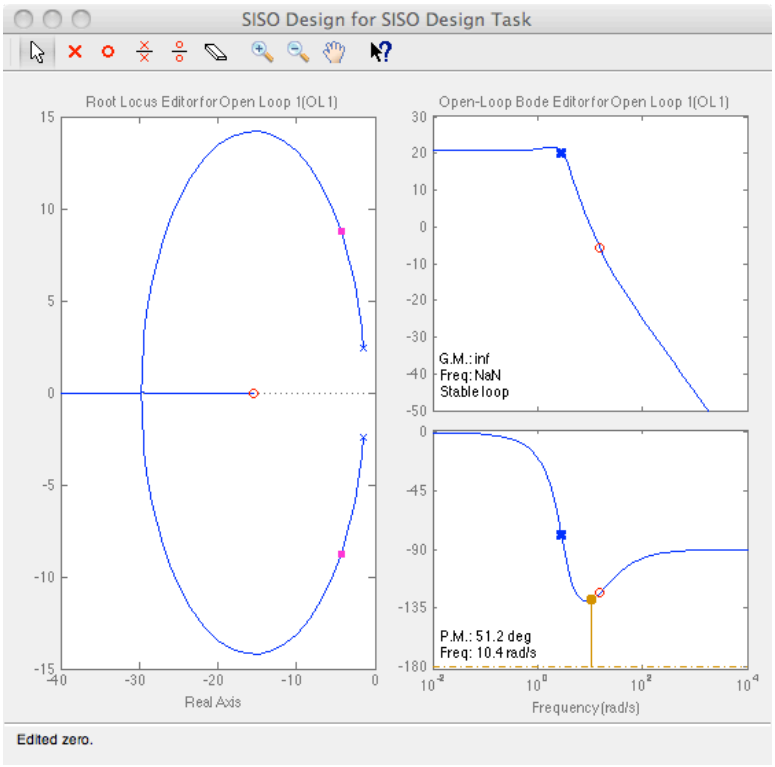


(c) We will need to increase the gain to meet the low frequency gain and/or crossover specifications. Let's try meeting the low frequency gain spec first, and see where that puts the crossover. Grabbing the magnitude plot and moving it up until the low frequency gain is just above 20dB gives us the following plot:

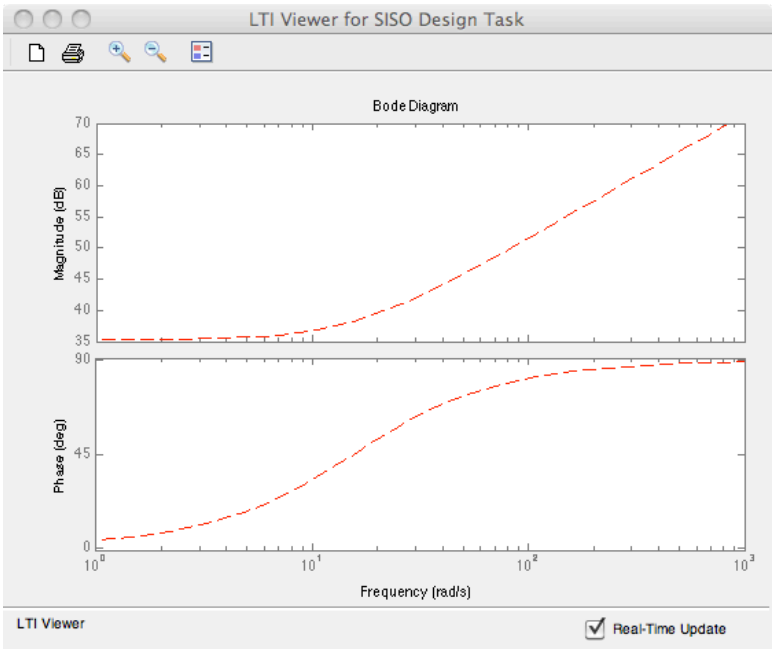


We see that crossover is 9.5 rad/s, which meets the crossover specification, so we don't need to increase the gain any further. However, the phase margin is 19.1°, indicating the need for a zero. We place a zero

at crossover, and get the following

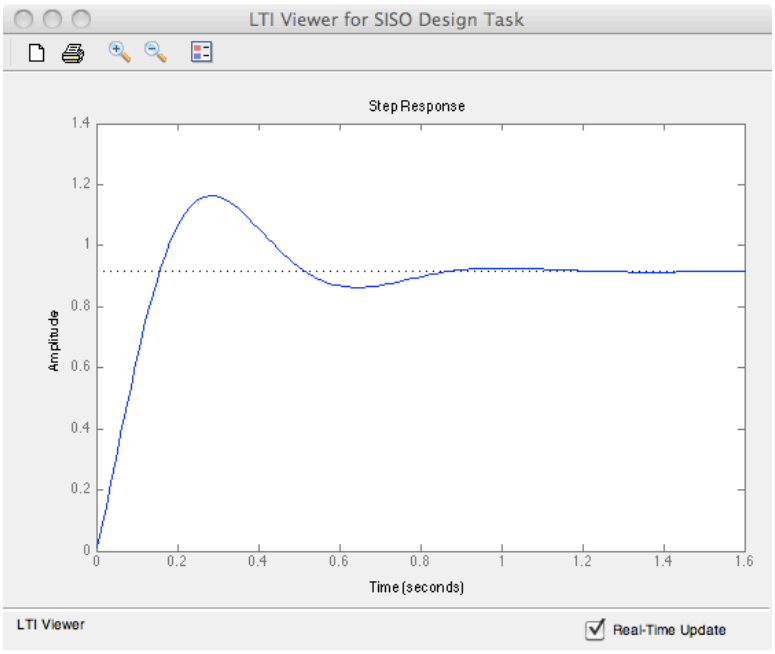


the bode plot for $C(s)G(s)$ is on the right. To see the frequency response of $C(s)$ alone, we can select from the menu "Analysis -> Compensator Bode"



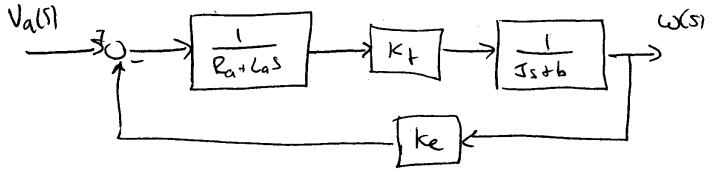
This PD controller will increase the loop gain at high frequency, and subsequently increase the phase, as shown. This is also called a *phase lead* compensator.

(d) To see the closed loop step response, select from the menu "Analysis -> Response of Step Command".



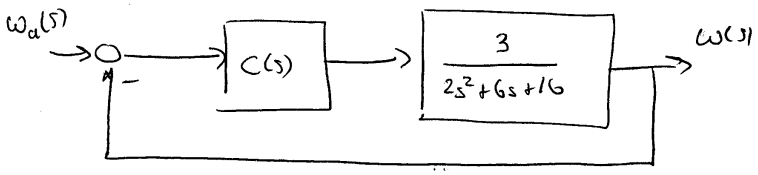
2.

(a)

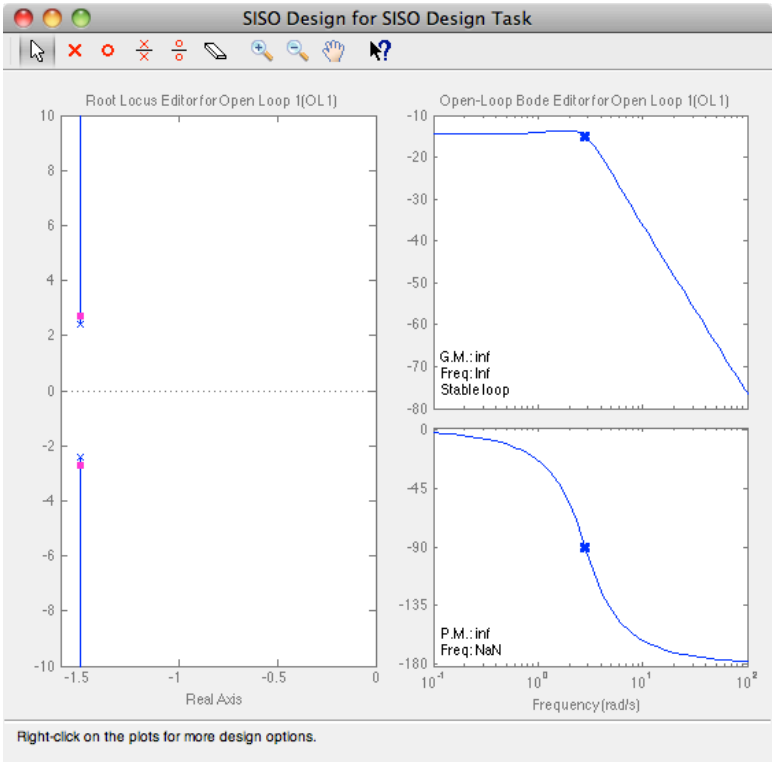


$$\frac{\omega(s)}{V_a(s)} = \frac{\frac{K_t}{(R_a+L_as)(Js+b)}}{1+\frac{K_tK_e}{(R_a+L_as)(Js+b)}} = \frac{K_t}{JL_as^2+(JR_a+L_ab)s+R_ab+K_tK_e}$$

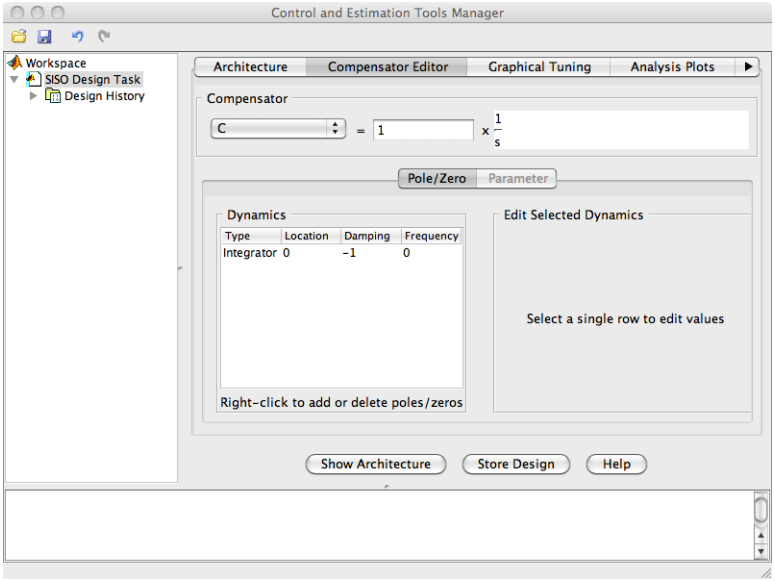
$$= \frac{3}{2s^2+6s+4+12}$$



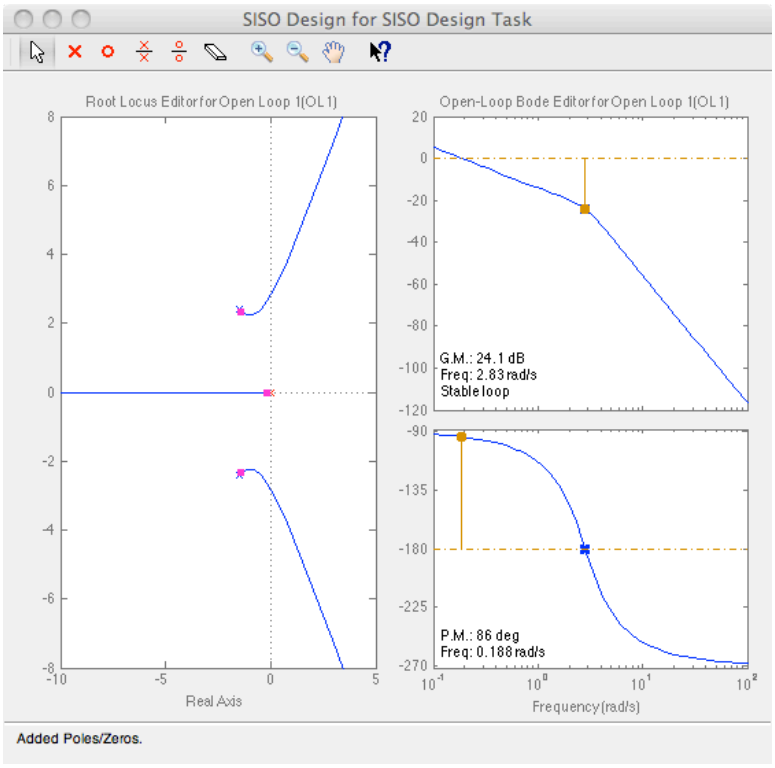
(b) `sisotool(tf(3,[2 6 16]))`



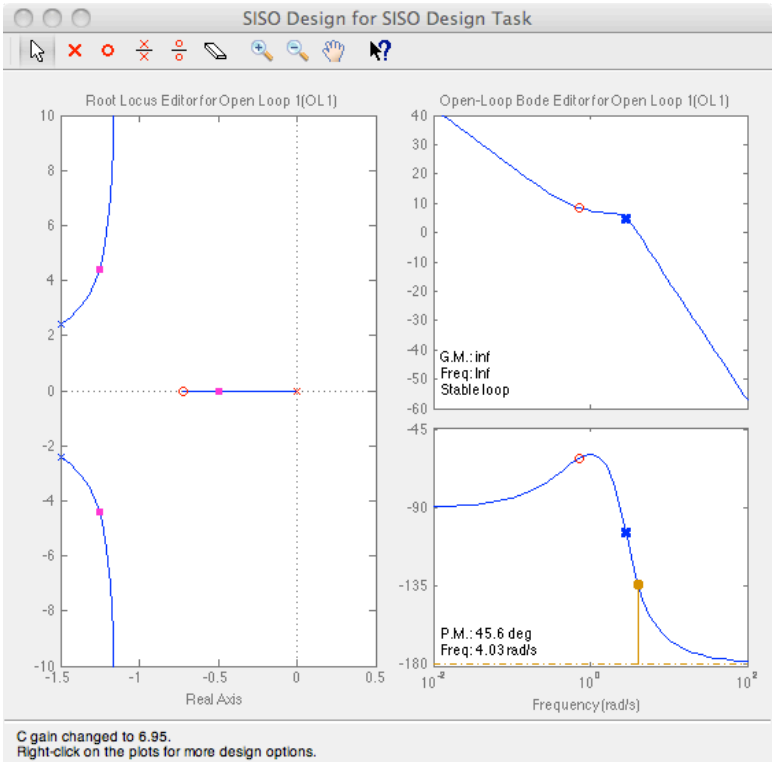
Edit the controller to add an integrator



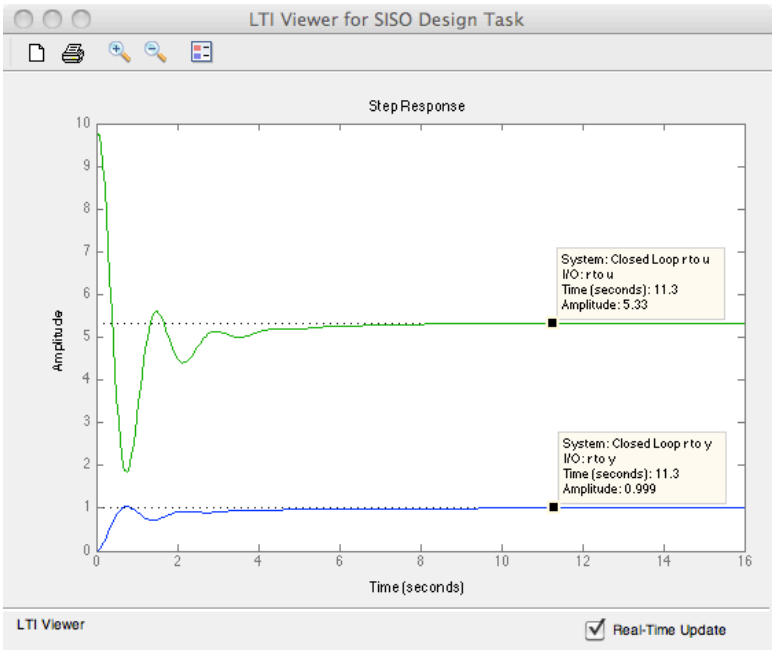
The loop gain frequency response is now



If we tried to move the crossover frequency to 4 rad/s, the phase margin would be negative. Thus, we need to add a zero before the desired crossover, and then increase the gain



This gives the following closed loop behavior:



We can recover the controller from the compensator editor: