ECE-320 Linear Control Systems
Laboratory 3

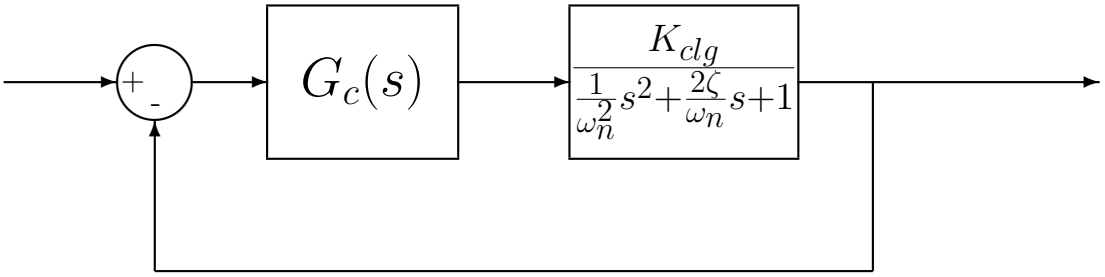
Model Matching (ITAE and Quadratic Optimal)

Preview In this Lab you will obtain a model of a one degree of freedom system using the log decrement method, determine the closed loop gain, and then use model matching techniques to produce a desired response. *You will analyze only one system in this lab.*

Pre-Lab

1) Print out this lab and **read** it.

In what follows, assume we have the unity feedback system shown below. $G_c(s)$ is the controller that you will be implementing, ω_n and ζ are estimates obtained using either time-domain or frequency domain methods, and K_{clg} is the system closed loop gain. Note that we are modelling the motor as contributing only a gain to the system, and we are lumping both the plant's gain and the motor gain together into one parameter.



2) Show that for a *proportional controller*, where

$$G_c(s) \;=\; k_p$$

the steady state output y_{ss} due to a step input of amplitude Amp is given by

$$y_{ss} \;=\; \frac{Amp\ k_p K_{clg}}{1 + k_p K_{clg}}$$

which can be rewritten as

$$K_{clg} \;=\; \frac{y_{ss}}{k_p}\frac{1}{Amp - y_{ss}}$$

This is the expression we will use to estimate the closed loop system gain K_{clg}.

We need to first **identify** the system:

1. Estimate an initial second order system model using time domain analysis (using the **log_dec** program).
2. Measure the frequency response (make one measurement at 1Hz, 2Hz, ..., 7 Hz, and at least 4 points near the resonant peak)
3. Use the **process_data** and **fit_bode** commands to estimate the gain of the system.
4. Use the **opt_fit_bode** command to fine tune the system model.
5. Determine the closed loop system gain K_{clg} (to be described below)

Estimating the Closed Loop Gain K_{clg}

0. Set the units

Click **Setup** → **User Units** and set the units to **cm**.

1. Setting up the controller

Click **Setup** → **Control Algorithm**. Be sure the system is set for *Continuous Time*. Select **PID** under **Control Algorithm**. Click on **Setup Algorithm**. Be sure **Feedback** is from **Encoder 1**. Set k_p to a small number (less than or equal to 0.05) and be sure $k_d = 0$ and $k_i = 0$. Then click **OK**. Next Click **Implement Algorithm**. The click **OK**.

2. Setting up the closed loop trajectory

Click **Command** → **Trajectory**. Select **Step** and click on **Setup**. Select **Closed Loop Step** and set **Step Size** to 1 to 2 cm. Be sure to record this step size (we'll refer to the amplitude as *Amp* below). Set the **Dwell Time** to something like 2000 ms, this is the time the system will be recording data. Finally click **OK**, then **OK** and you should be back to the main menu.

3. Executing the closed loop step

Click **Command Execute**. A menu box will come up with a number of options, and a big green **Run** button. Click on the **Run** button. When the system has finished collecting data, a box will appear indicating the how many sample points of data have been collected. (If you have hit a stop, the system stops recording data. This usually means you're input amplitude was too large or k_p was too large.) Click on **OK** to get back to the main menu.

4. Determining the steady state value

Click **Plotting** → **Setup Plot**, or just **Plotting Data** → **Plot Data**. Look at the steady state value (y_{ss}). You may need to change the dwell time if your system has not reached steady state.

5. Estimating the closed loop gain

Estimate the closed loop gain K_{clg} using the formula derived in the prelab: $K_{clg} = \frac{y_{ss}}{k_p}\frac{1}{A-y_{ss}}$. You need to go through this procedure at least three times. You must use at least two different values of k_p and two different values of input amplitude *Amp*. If none of the steady state values is larger than 0.4 cm, increase either k_p or *Amp*. Average the three results to get your K_{clg} (they should be similar). For the trials I've run, I've got K_{clg} between 10 and 20. Your's may be outside this range though.

Model Matching with Zero Position Error ITAE Models

Our general goals are as follows:

- track a step input of amplitude 1 cm
- produce a position error of less than 0.05
- reach steady state within 0.4 seconds (the faster the better)
- have as little overshoot as you can manage

The second, third, and fourth order zero position error ITAE systems have the following closed loop transfer functions

$$\begin{aligned} G_0(s) &= \frac{\omega_0^2}{s^2 + 1.4\omega_0 s + \omega_0^2} \\ G_0(s) &= \frac{\omega_0^3}{s^3 + 1.75\omega_0 s^2 + 2.15\omega_0^2 s + \omega_0^3} \\ G_0(s) &= \frac{\omega_0^4}{s^4 + 2.1\omega_0 s^3 + 3.4\omega_0^2 s^2 + 2.7\omega_0^3 s + \omega_0^4} \end{aligned}$$

Keep in mind, the larger ω_0, the faster the system responds and the harder the motor has to work. I have used values of ω_0 from 10 to 50, and lived to tell the tale..

Once we know the order of ITAE system we want (which should be at least as large as that of our plant), we can compute the controller as

$$G_c(s) = \frac{G_0(s)}{G_p(s)(1 - G_0(s))}$$

The program **ITAE_0** (see below) with do this for you. You need to choose ω_0 and an order of ITAE system, look at the model results, implement the controller on the ECP system, and then compare the simulated and modelled results.

Specifically, you need to:

1. Construct your plant transfer function in the Matlab workspace. For this part, use the form

$$G_p(s) = \frac{K_{clg}}{\frac{1}{\omega_n^2}s^2 + \frac{2\zeta}{\omega_n}s + 1}$$

2. Run the program **ITAE_0**. This routine has the input arguments
   - The amplitude of the step input
   - The plant transfer function $G_p(s)$

- The value of ω_0
- The order of the ITAE system (2-4)
- The length of time to plot the results
- The filename with the ECP data (in single quotes), used to compare the model and the real system. If there is no file, type ''.

3. If the performance of the model is acceptable, you need to implement the controller on the ECP system. To do this[1]
   (a) Click **Setup → Control Algorithm**
   (b) Select **Dynamic Forward Path**
   (c) Click **Setup Algorithm**
   (d) Click **Import**
   (e) Select **values.par** and click **open**. You should see values entered into the R and S arrays. These should look like the values the program **ITAE_0** spits to the screen.
   (f) Click **OK**
   (g) Click **Implement Algorithm**
   (h) Click **OK**
4. Once the ECP system has run acceptably, you need to export the data, edit it, and the rerun **ITAE_0** to produce a plot of the predicted response and the model response.
5. Write down the controller $G_c(s)$ for each system, so you can refer to them later.
6. You may need to reset the controller often, such as every time you want to implement a new controller. Click **Utility → Reset Controller**. Only do this **before** you have implemented a controller.
7. You may need to rephase the motor. Click **Utility → Rephase Motor**
8. Be sure to **Implement** the controller you have designed.

You should try three different values of ω_0 with a second order ITAE model, and then use one of these values of ω_0 with the third and fourth order ITAE model. For each of these you should have a plot of the predicted and real response.

[1] I have tried to automate this a bit. However, if you really want to enter all the numbers by hand, be my guest. I did not mean to deprive you...

Model Matching with Zero Velocity Error ITAE Models

Our general goals are as follows:

- track a step input of amplitude 1 cm
- produce a position error of less than 0.05
- reach steady state within 0.4 seconds (the faster the better)
- have as little overshoot as you can manage

The second, third, and fourth order velocity position error ITAE systems have the following closed loop transfer functions

$$\begin{aligned} G_0(s) &= \frac{3.2\omega_0 s + \omega_0^2}{s^2 + 3.2\omega_0 s + \omega_0^2} \\ G_0(s) &= \frac{3.25\omega_0^2 s + \omega_0^3}{s^3 + 1.75\omega_0 s^2 + 3.25\omega_0^2 s + \omega_0^3} \\ G_0(s) &= \frac{5.14\omega_0^3 s + \omega_0^4}{s^4 + 2.41\omega_0 s^3 + 4.93\omega_0^2 s^2 + 5.14\omega_0^3 s + \omega_0^4} \end{aligned}$$

Once we know the order of ITAE system we want (which should be at least as large as that of our plant), we can compute the controller as

$$G_c(s) = \frac{G_0(s)}{G_p(s)(1 - G_0(s))}$$

The program **ITAE_1** (see below) with do this for you. You need to choose ω_0 and an order of ITAE system, look at the model results, implement the controller on the ECP system, and then compare the simulated and modelled results.

Specifically, you need to:

1. Construct your plant transfer function in the Matlab workspace. For this part, use the form

$$G_p(s) = \frac{K_{clg}}{\frac{1}{\omega_n^2}s^2 + \frac{2\zeta}{\omega_n}s + 1}$$

2. Run the program **ITAE_1**. This routine has the input arguments
   - The amplitude of the step input
   - The plant transfer function $G_p(s)$
   - The value of ω_0
   - The order of the ITAE system (2-4)
   - The length of time to plot the results

- The filename with the ECP data (in single quotes), used to compare the model and the real system. If there is no file, type ''.

3. If the performance of the model is acceptable, you need to implement the controller on the ECP system. To do this:
   (a) Click **Setup → Control Algorithm**
   (b) Select **Dynamic Forward Path**
   (c) Click **Setup Algorithm**
   (d) Click **Import**
   (e) Select **values.par** and click **open**. You should see values entered into the R and S arrays. These should look like the values the program **ITAE_1** spits to the screen.
   (f) Click **OK**
   (g) Click **Implement Algorithm**
   (h) Click **OK**
4. Once the ECP system has run acceptably, you need to export the data, edit it, and the rerun **ITAE_1** to produce a plot of the predicted response and the model response.
5. Write down the controller $G_c(s)$ for each system, so you can refer to them later.
6. You may need to reset the controller often, such as every time you want to implement a new controller. Click **Utility → Reset Controller**. Only do this **before** you have implemented a controller.
7. You may need to rephase the motor. Click **Utility → Rephase Motor**
8. Be sure to **Implement** the controller you have designed.

You should try three different values of ω_o with a second order ITAE model, and then use one of these values of ω_0 with the third and fourth order ITAE model. For each of these you should have a plot of the predicted and real response. You will probably notice that the system is much slower with the zero velocity error ITAE systems than with the zero position error systems (with comparable values of ω_0)

Model Matching with Quadratic Optimal Systems

Our general goals are as follows:

- track a step input of amplitude 1 cm
- produce a position error of less than 0.05
- reach steady state within 0.4 seconds (the faster the better)
- have as little overshoot as you can manage

These systems minimize the performance index

$$J = \int_0^\infty \left[q(y(t) - r(t))^2 + u(t)^2\right] dt$$

where $r(t)$ is the input (reference signal), $y(t)$ is the system output, $u(t)$ is the control effort, and q is a weighting factor. The program **quadratic** will automatically determine the optimal $G_0(s)$ for you, once you have determined the value of q you want. Once we know $G_0(s)$, we can compute the controller as

$$G_c(s) = \frac{G_0(s)}{G_p(s)(1 - G_0(s))}$$

The program **quadratic** will do this for you. You need to choose q, look at the model results, implement the controller on the ECP system, and then compare the simulated and modelled results.

Specifically, you need to:

1. Construct your plant transfer function in the Matlab workspace. For this part, use the form

$$G_p(s) = \frac{K_{clg}\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

2. Run the program **quadratic**. This routine has the input arguments
   - The amplitude of the step input
   - The plant transfer function $G_p(s)$
   - The value of q. (You should try using values of q less than 0.1 or so. If q is too large the model converges to the reference value very quickly, but the real system no longer matches the model, and, in fact, produces an increasingly poor result.)
   - The length of time to plot the results
   - The filename with the ECP data (in single quotes), used to compare the model and the real system. If there is no file, type ''.

3. If the performance of the model is acceptable, you need to implement the controller on the ECP system. To do this:
   (a) Click **Setup → Control Algorithm**
   (b) Select **Dynamic Forward Path**
   (c) Click **Setup Algorithm**
   (d) Click **Import**
   (e) Select **values.par** and click **open**. You should see values entered into the R and S arrays. These should look like the values the program **quadratic** spits to the screen.
   (f) Click **OK**
   (g) Click **Implement Algorithm**
   (h) Click **OK**
4. Once the ECP system has run acceptably, you need to export the data, edit it, and the rerun **quadratic** to produce a plot of the predicted response and the model response.
5. Write down the controller $G_c(s)$ for each system, so you can refer to them later.
6. You may need to reset the controller often, such as every time you want to implement a new controller. Click **Utility → Reset Controller**. Only do this **before** you have implemented a controller.
7. You may need to rephase the motor. Click **Utility → Rephase Motor**
8. Be sure to **Implement** the controller you have designed.

You should try three different values of q. For each of these you should have a plot of the predicted and real response. You will probably notice that the model and real system do not match very well as the values for q get larger.

Memo Your memo should compare (briefly) the response of the model and the response of the real system. How close is the predicted behavior to the behavior of the real system? You should have some description of the configuration of the system you were trying to control. Specifically, it should include:

1. A table showing k_p, y_{ss}, Amp, and K_{clg} for the three tries at estimating K_{clg}.
2. The values of ζ, ω_n and the final value of K_{clg} for the system you modelled.
3. For the system analyzed you should have (as a minimum):
   - 5 plots for the ITAE zero position error systems
   - 5 plots for the ITAE zero velocity error systems
   - 3 plots for the quadratic optimal systems
4. A brief discussion of at least one fundamental difference between the controllers designed using the ITAE methods and the controllers designed using the quadratic optimal methods. For which type of controller do you think you need a more accurate model?