Massachusetts Institute of Technology

Department of Mechanical Engineering

2.004 *Dynamics and Control II*

Laboratory Session 7: Frequency Domain Control Design

Laboratory Objectives:

(i) To investigate open and closed loop Bode plots of the 2.004 DC motor system.

(ii) To design a lead compensator to achieve given closed-loop specifications.

(iii) To compare root locus and frequency methods.

Introduction: In previous labs we approached controller design through root locus techniques based on pole-zero models. In this session we switch to a controller design method based on frequency analysis. We will determine the accuracy of our flywheel transfer function across a range of frequencies by performing a frequency sweep. From the frequency response plots we can then design a compensator to change the phase margin to a desired value, which will in turn produce a satisfactory closed loop response.

Experiment #1: Open and Closed Loop Bode Plots

To build a Bode plot of our system from experimental data, we will perform a frequency sweep using an input "Chirp" signal (see Appendix D), which varies the frequency of a sinusoidal signal over time. To improve low frequency responses, we will use an Exponential Chirp[1] to sweep the frequency from low (0.05 Hz) to high frequency (10 Hz) logarithmically. This frequency range was chosen in order to capture the dominant features in the frequency response of the open-loop plant.

(a) Download all Lab 7 files from Canvas: `https://canvas.mit.edu/courses/10470/assignments/129632`.

(b) Open the Arduino template file `motor_position_control.ino` and take a look at the code to make sure you understand its operation. Use the exponential chirp signal to the open loop plant by making sure that `#define OPEN_LOOP_PLANT` and `#define EXP_CHIRP` are uncommented and the other options are commented out.

Collect the angular position output by running the `SerialRead.m` MATLAB script. This will collect the set point, position sensor output, and pwm/100 (controller output) signals according to the block diagram in Figure 1. The chirp signal is set up to run for 100 seconds so you can stop the data streaming right after.

[1]`https://en.wikipedia.org/wiki/Chirp`

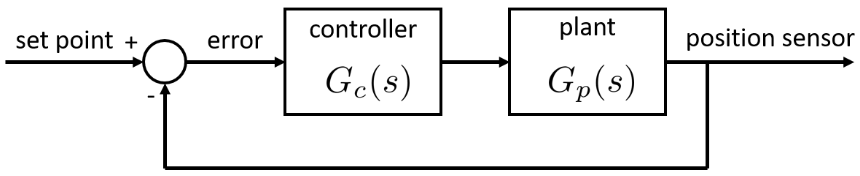


Figure 1: Closed loop block diagram with labeled signals.

(c) Run the MATLAB function `estimate_freq_resp.m` to build the experimental Bode plots for the open loop transfer function. The function requires the variable `data` from the MATLAB workspace, plant transfer function G_p, and controller transfer function G_c. For example, run the command `estimate_freq_resp(data, Gp, 1, 'open_loop')` will plot a pair of open loop Bode plots for both open loop plant and experimental data.

(d) Repeat steps (b) and (c) with the closed-loop system with a proportional controller with gain $G_c(s) = 100$ by uncommenting `#define PID` and commenting out `#define OPEN_LOOP_PLANT` in the Arduino code. Upload the code and use `SerialRead.m` to acquire data. Then run `estimate_freq_resp(data, Gp, 100, 'closed_loop')` to generate closed-loop Bode plots.

Experiment #2: Controller Design

Here we wish to design a controller that will produce a closed-loop step response to meet the following time domain specifications:

- an overshoot of 7%
- settling time $T_s = 0.4$ seconds or less.

A quick inspection of your plots should reveal that the uncompensated system does not achieve these performance specifications. To fix this, we will design a lead compensator of the form (see Appendix A). The lead compensator is of the form:

$$G_c(s) = K_c\left(\frac{T_D s + 1}{\alpha T_D s + 1}\right).$$

We will use the MATLAB script `Lead_compensator_design.m` to aid in our calculations.

(a) Using the steps in Appendix A, determine the required performance parameters: phase margin ϕ_m, and crossover frequency ω_c, from the time domain specifications. Hint: think about how to relate overshoot to ζ and then to ϕ_m, and ω_c to settling time through their respective equations (see Appendix B).

(b) Look for "`% TO DO`" in the script and edit/complete the code. Lines you need to complete include the adjusted theoretical transfer function (Lines 7 and 8), and the required performance parameters ϕ_m and ω_c (Lines 32 and 33), the phase to be added and the location of the target frequency, respectively.

(c) Run the code to determine the compensator, and to verify from the plots that it has the expected phase and gain margins from the Bode plot.

(d) Using the Matlab `stepinfo` function, determine if your newly compensated system meets the time domain performance requirement: `stepinfo(feedback(Gc*Gp,1))`. Tweak the values if required.

(e) Using `sisotool`, as in previous labs, generate the root locus, step response and Bode plots of the controller you've designed to check their relationship.

Experiment #3: Controller Testing and Verification

Now we want to implement the controller we designed to validate its theoretical behavior. This will be done by implementing the lead compensator in the Arudino code.

(a) Enter your newly designed lead controller in the Arduino code by modifying the constants `Kc`, `T_1`, and `T_2` to match with your designed lead controller. Find the open loop Bode plot with the lead controller ($G_c(s)G_p(s)$) by running the exponential chirp again and making sure that `#define OPEN_LOOP_LEAD` is uncommented and the other controller options are commented out.

(b) Using the newly determined `Gc` as an input to the `estimate_freq_resp.m` function, and build the experimental and theoretical Bode plots for the open loop frequency response by running the script.

(c) Now apply an input of a sine wave in the Arduino code by uncommenting `#define SINE_WAVE` and commenting out the other input options. Apply the sine wave with a frequency equal to the crossover frequency (ω_c) in the open-loop Bode plot and obtain the set_point and sensor signals of pure sinusoidal time domain response. Manually measure the magnitude and phase of the two signals and compare them with your Bode plots. Refer to Appendix C for determining gain and phase of two sine waves.

(d) Find the closed loop Bode plot with the lead controller by running the exponential chirp again and making sure that `#define LEAD_LAG` option is uncommented.

(e) Change the option to `'closed_loop'` for the `estimate_freq_resp.m` function, and build the experimental and theoretical Bode plots for the closed loop frequency response by running the function.

(f) Find the step response of the system using an input of a square wave in the Arduino code by uncommenting `#define SQUARE_WAVE` and commenting out the other input options. Run this on your plant and save the step response. Verify the overshoot and settling time requirements.

Appendix A: Lead Controller Frequency Design

You may find the following formulas relating the phase margin and frequency of a lead compensator to its transfer function useful. The following plot shows the contribution of the compensator in the region of interest.

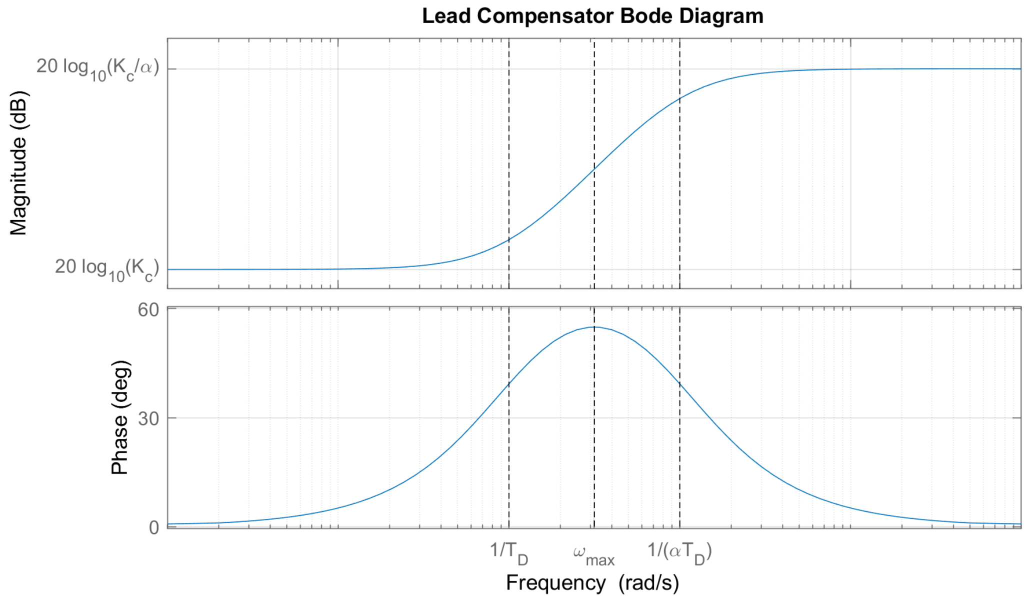


Figure 2: Bode plot for a lead compensator in the form of Equation 1.

A lead compensator has the form

$$G_c(s) = K_c \frac{T_D s + 1}{\alpha T_D s + 1} \qquad 0 \leq \alpha < 1 \tag{1}$$

where $\frac{1}{\alpha}$ is the ratio between the pole/zero break-point frequencies. The maximum phase lead ϕ_{max} of the compensator occurs at a frequency that lies midway between the two break-point frequencies ω_{max} on a logarithmic scale (geometric mean), which is given by the formulas:

$$\sin(\phi_{max}) = \frac{1-\alpha}{1+\alpha} \tag{2}$$
$$\omega_{max} = \frac{1}{T_D\sqrt{\alpha}}$$

Additionally, by the definition of the crossover point, the plant and controller magnitude at the crossover frequency ω_c follow the equation

$$1 = |G_c(j\omega_c)G_p(j\omega_c)|$$

which may be used to find the crossover frequency from a known plant. We wish to add the maximum lead compensator phase ϕ_{max} to the crossover frequency of the plant, ω_c, in order to increase the closed loop phase margin most effectively. Therefore, we want to set $\omega_c = \omega_{max}$.

One possible design procedure for a lead compensator can be summarized as follows:

1. Evaluate the phase margin (PM) of the open loop system.
2. Allow for approximately $5°$ to $10°$ of extra margin and determine needed phase lead ϕ_{max} to add to the open loop plant to meet the phase margin specification at the crossover frequency.
3. Determine α from Equation (2).
4. Pick ω_{max} to be at the crossover frequency, which implies that the zero is at $z = \frac{1}{T_D} = \omega_{max}\sqrt{\alpha}$ and the pole is at $p = \frac{1}{\alpha T_D} = \frac{\omega_{max}}{\sqrt{\alpha}}$.
5. Determine the compensator gain K_c such that the crossover frequency is the same before and after adding the lead compensator.
6. Draw the compensated frequency response and check the PM.
7. Iterate on the design.

Note that, when adding a lead compensator to the closed loop, if the gain K_c is kept the same, then the cross-over frequency will increase. Alternatively, if the crossover frequency is kept the same, then the closed loop gain will decrease with the addition of the lead compensator. This may lead to a slightly different design procedure where the gain K_c is adjusted as well to help meet specifications. If steady-state error or low frequency response is a concern then we can add a lag compensator to raise low frequency gain, or add integral control action to eliminate steady-state error.

Typically, we use $\alpha \geq 0.1$ to avoid placing poles at high frequency, resulting in large gains. Additionally, if more than $\approx 60°$ of phase is needed, it is suggested to use multiple lead compensators in series.

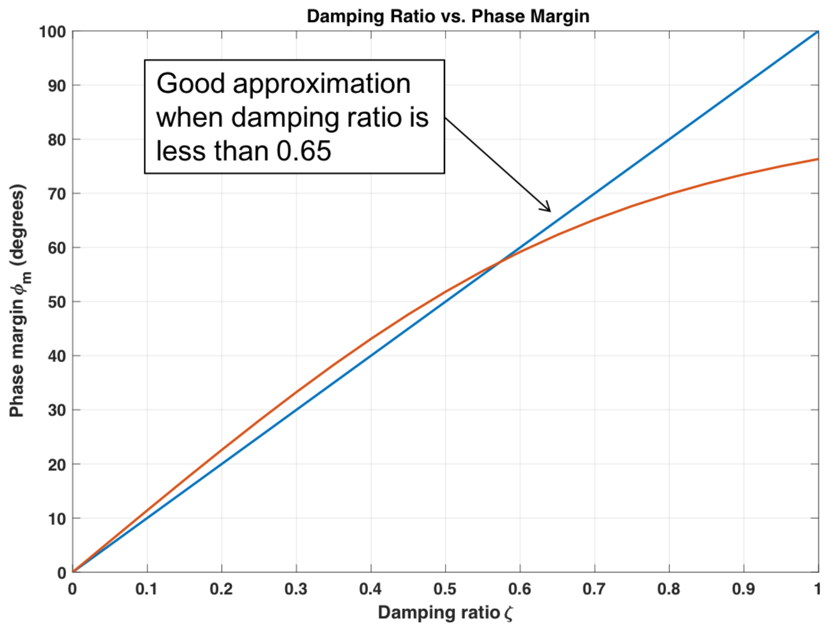
Appendix B: Mathematical Relationships

These equations allow you to map a percentage overshoot requirement to the damping ratio ζ and to a phase margin (PM) requirement.

$$\zeta = \frac{-\ln(\%OS/100)}{\sqrt{\pi^2 + \ln^2(\%OS/100)}}$$

$$PM = \tan^{-1}\left(\frac{2\zeta}{\sqrt{\sqrt{4\zeta^4+1} - 2\zeta^2}}\right) \approx 100\zeta \qquad \text{if } \zeta \leq 0.65$$

From the plots below, observe where these functions are valid, and factor this uncertainty in if required.

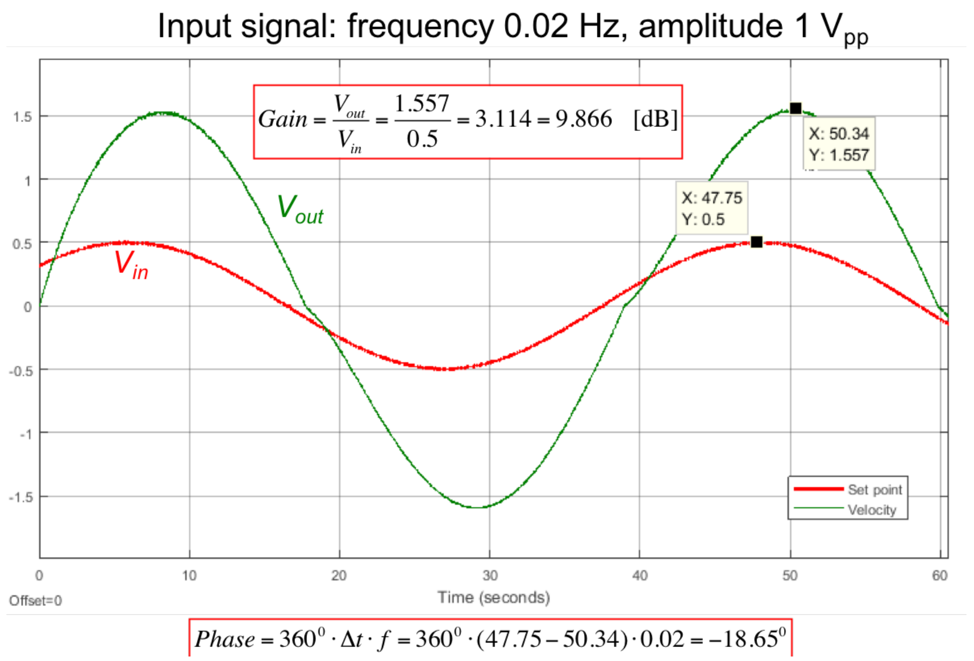


When designing controllers in the frequency domain, we typically assume that the system is dominated by 2nd order poles. For a 2nd order system, the crossover frequency is usually close to the natural frequency. Therefore, we assume that

$$\omega_c \approx \omega_n. \tag{3}$$

Appendix C: Measurement of Sinusoids

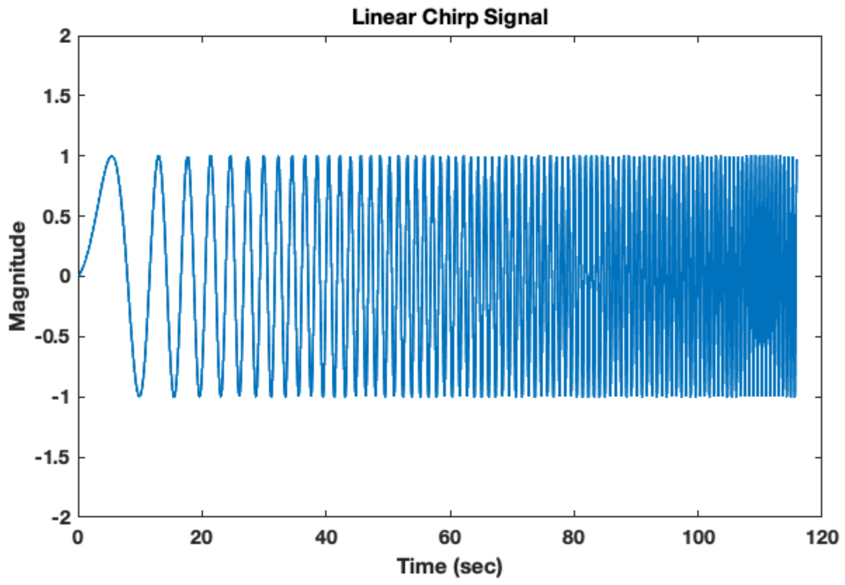
The graph below demonstrates how one can measure the magnitude and phase relationship between input and output sinusoid signals. This technique can be used to build a Bode plot, or to calculate the complex value of the transfer function ($H(i\omega) = Ae^{i\phi}$) for a given frequency ω.

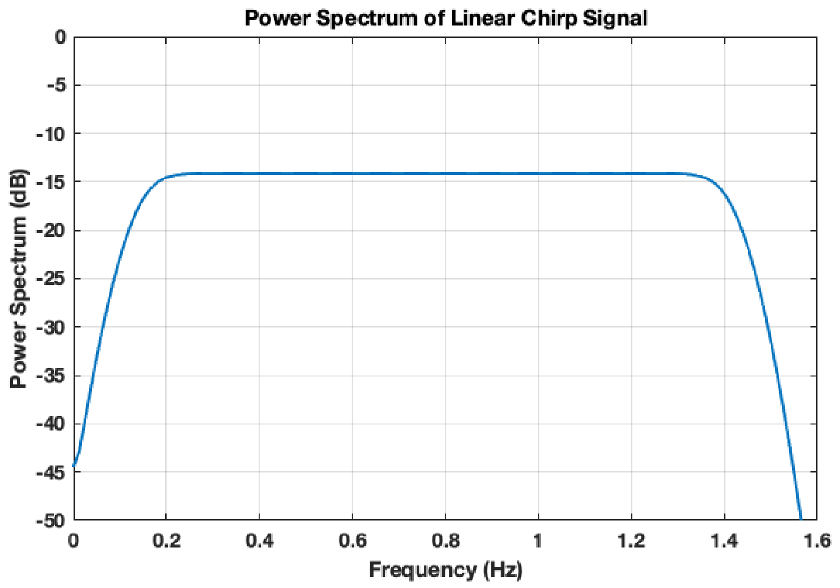


Appendix D: Chirp (Swept-Sine) Function

A (linear) chirp function is a sinusoidal function that has its frequency vary in a linear manner between an initial and final frequency. It is usually represented in the following time domain form, where T is the sweep time, ω_0 is the initial frequency, ω_1 the final frequency, and ϕ_0 is the initial phase angle.

$$f(t) = \sin\left(\omega_0 t + \frac{(\omega_1 - \omega_0)}{2T} t^2 + \phi_0\right)$$





The drawback of linear chirp is that the time spent at low frequencies is the same as at high frequencies, and as such it may not give us an accurate response at low frequency. To rectify this problem, an exponential (or geometric) chirp signal can be used, which varies the frequency exponentially over time. The exponential chirp function is defined as

$$f(t) = \sin\left(\omega_0 \left(\frac{k^t - 1}{\ln(k)}\right) + \phi_0\right)$$

where

$$k = \left(\frac{\omega_1}{\omega_0}\right)^{1/T}.$$

With this property the exponential chirp function will allocate more time for low frequencies, and as such the low frequency response is improved.