ECE 3413 Lab 10 Closed Loop Gain Determination using Root Locus

Leomar Durán

23th April 2023

1 Introduction

1.1 The setup

For this lab, we will need to install the System Identification Toolbox add-on.

Before starting the installation, be warned that the installation will require escalation to Administrator privileges. One will also need to agree to "The MathWorks, Inc. Software License Agreement" as the System Identification Toolbox add-on as created by MathWorks (as of this writing). After the installation, MATLAB will restart, so save all previous work first.

Additionally, this add-on has no additional dependencies. The download is 61 MByte and the installation is 160 MByte.

1. Once ready, open the Home tab, and find the Add-Ons menu in the ENVIRONMENT section as shown in Fig. 1, and click the Get Add-Ons button highlighted.

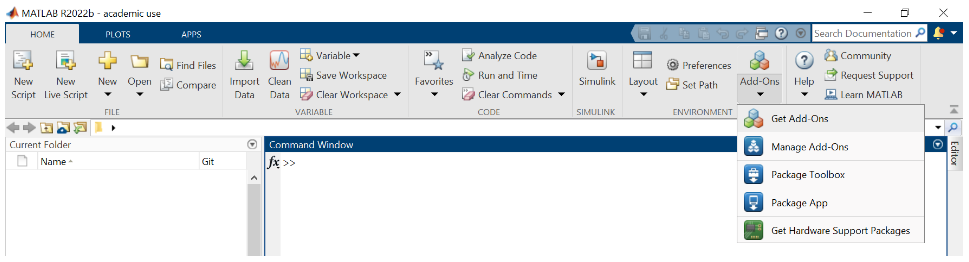
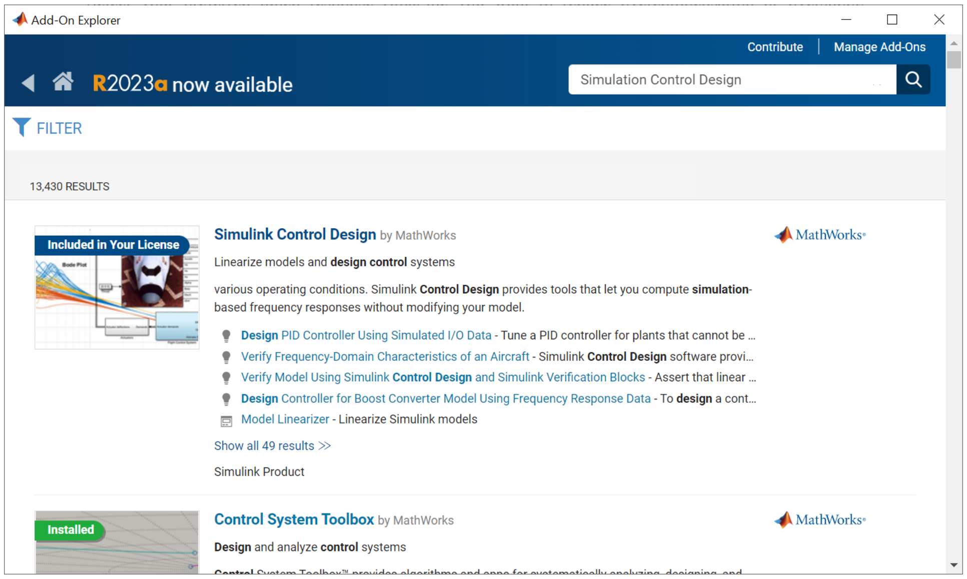


Figure 1: Finding the Add-Ons menu.

2. In the search bar of the "Add-On Explorer" dialog opened, search for "Simulation Control Design". It is the first hit in the results listed in Fig. 2.



Add-On Explorer

Contribute | Manage Add-Ons

R2023a now available

Simulation Control Design

FILTER

13,430 RESULTS

Included in Your License

Simulink Control Design by MathWorks

Linearize models and **design control** systems

various operating conditions. Simulink **Control Design** provides tools that let you compute **simulation**-based frequency responses without modifying your model.

- **Design** PID Controller Using Simulated I/O Data - Tune a PID controller for plants that cannot be ...
- Verify Frequency-Domain Characteristics of an Aircraft - Simulink **Control Design** software provi...
- Verify Model Using Simulink **Control Design** and Simulink Verification Blocks - Assert that linear ...
- **Design** Controller for Boost Converter Model Using Frequency Response Data - To **design** a cont...
- Model Linearizer - Linearize Simulink models

Show all 49 results >>

Simulink Product

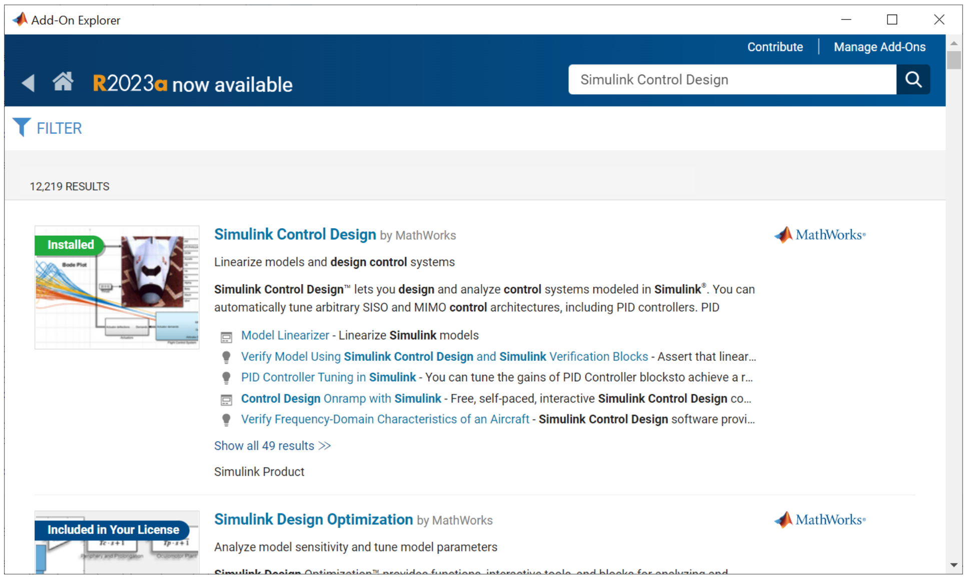
Installed

Control System Toolbox by MathWorks

Design and analyze **control** systems

Figure 2: The “Add-On Explorer” dialog with “Simulation Control Design” hovered in navy blue text.

The Simulation Control Design add-on should be labeled with the navy blue “Included In Your License” label. However, if it was already installed, then you will see the green “Installed” label seen in Fig. 3, in which case, no further action is needed and you may proceed to Subsection 1.2 Background information.



12,219 RESULTS

Simulink Control Design by MathWorks

Linearize models and **design control** systems

Simulink Control Design™ lets you **design** and analyze **control** systems modeled in **Simulink**®. You can automatically tune arbitrary SISO and MIMO **control** architectures, including PID controllers. PID

- Model Linearizer - Linearize **Simulink** models
- Verify Model Using **Simulink Control Design** and **Simulink** Verification Blocks - Assert that linear...
- PID Controller Tuning in **Simulink** - You can tune the gains of PID Controller blocksto achieve a r...
- **Control Design** Onramp with **Simulink** - Free, self-paced, interactive **Simulink Control Design** co...
- Verify Frequency-Domain Characteristics of an Aircraft - **Simulink Control Design** software provi...

Show all 49 results >>

Simulink Product

Simulink Design Optimization by MathWorks

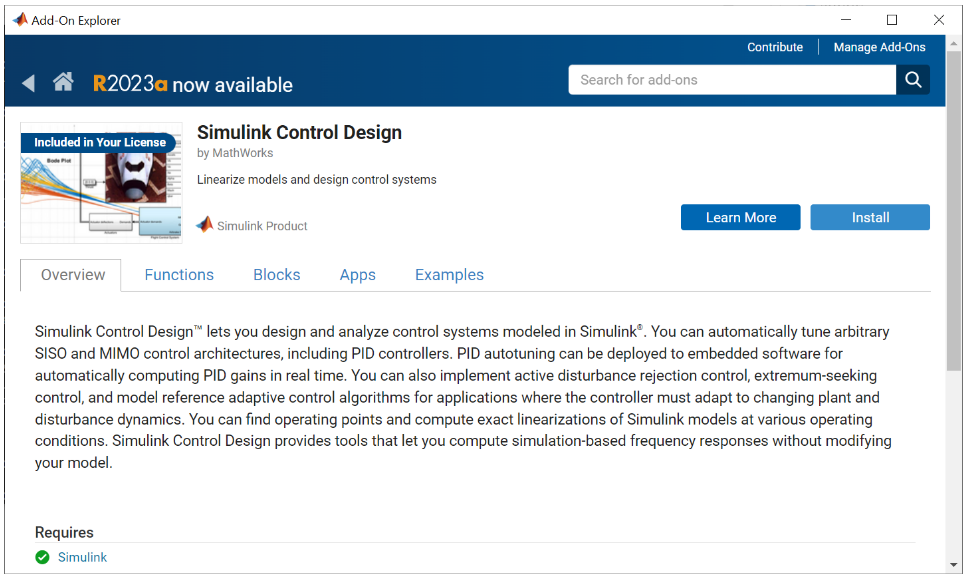
Analyze model sensitivity and tune model parameters

Figure 3: The Simulation Control Design add-on already installed in the “Add-On Explorer” dialog.

However, if the label is different from either of these, please contact your administrator about licensing before proceeding.

Proceeding from Fig. 2, click on the Simulation Control Design add-on to select it.

3. To install, click on the Install button, which will highlight in light blue when hovered over like in Fig. 4.



Simulink Control Design
by MathWorks

Linearize models and design control systems

Simulink Product

Simulink Control Design™ lets you design and analyze control systems modeled in Simulink®. You can automatically tune arbitrary SISO and MIMO control architectures, including PID controllers. PID autotuning can be deployed to embedded software for automatically computing PID gains in real time. You can also implement active disturbance rejection control, extremum-seeking control, and model reference adaptive control algorithms for applications where the controller must adapt to changing plant and disturbance dynamics. You can find operating points and compute exact linearizations of Simulink models at various operating conditions. Simulink Control Design provides tools that let you compute simulation-based frequency responses without modifying your model.

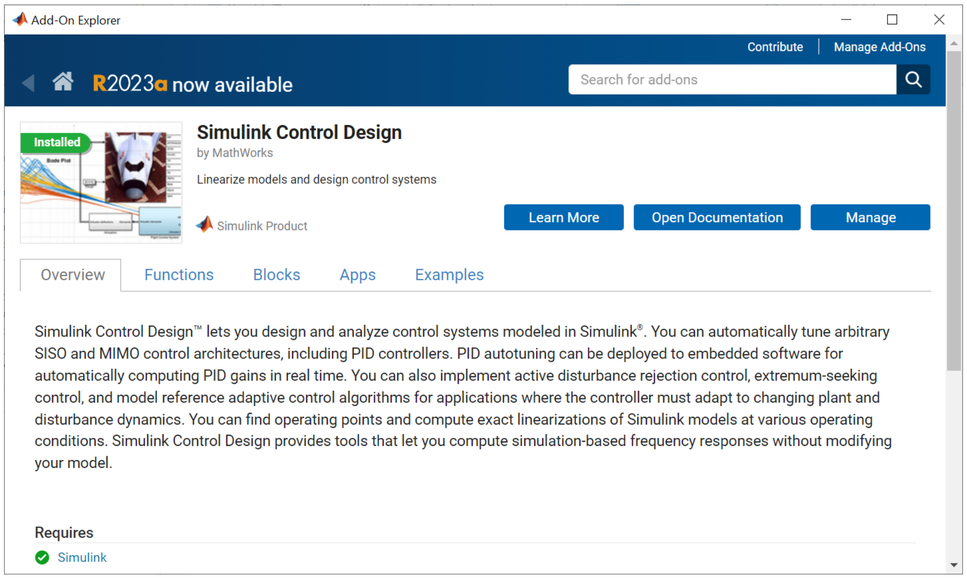
Requires

✓ Simulink

Figure 4: The Simulation Control Design add-on selected and ready for installation.

Follow the onscreen instructions, which will include escalation to Administrator privileges and agreeing to the "The MathWorks, Inc. Software License Agreement".

4. MATLAB will complete the installation process and restart, after which you will be greeted by the dialog in 5.



Simulink Control Design

by MathWorks

Linearize models and design control systems

Simulink Control Design™ lets you design and analyze control systems modeled in Simulink®. You can automatically tune arbitrary SISO and MIMO control architectures, including PID controllers. PID autotuning can be deployed to embedded software for automatically computing PID gains in real time. You can also implement active disturbance rejection control, extremum-seeking control, and model reference adaptive control algorithms for applications where the controller must adapt to changing plant and disturbance dynamics. You can find operating points and compute exact linearizations of Simulink models at various operating conditions. Simulink Control Design provides tools that let you compute simulation-based frequency responses without modifying your model.

Requires

Simulink

Figure 5: Dialog showing complete installation of the Simulation Control Design add-on.

1.2 Background information

The root locus technique may be used to find the optimal gains for a controller on a plant.

We will be applying this technique to a plant on noisy data.

2 Procedure

2.1 Task 00 – Parameters

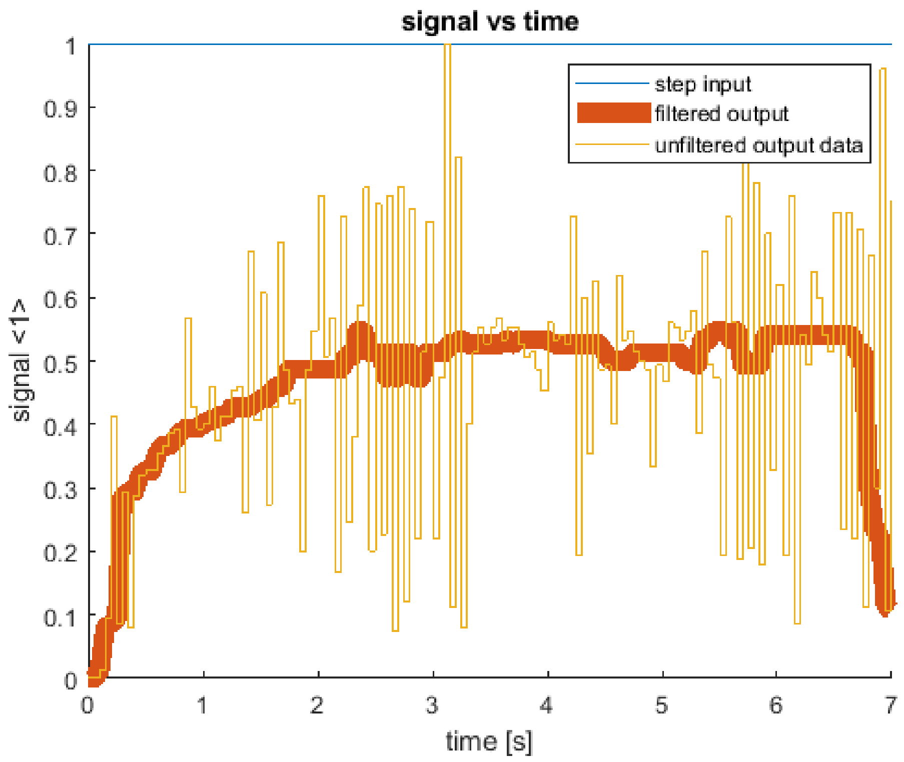


Figure 6: Comparing the step input, raw output data and filtered output data.

We start by loading the noisy data. From this data, we can reconstruct a step response triggered on $t = 0$, as well as apply a median filter to the data to create a smoother signal with similar rise and settling time characteristics.

To find the rise and settling time characteristics, we have to smoothen the curve in the first place. In light of this, a median filter with 19o seems to smoothen out the function satisfactorily by inspection as can be seen in Fig. 6.

This data parameterizes the transfer functions which we will estimate. The script for this parameterization is available in Appendix subsection A.1

Next we use the Matlab `ident` function to call up the System Identification Toolkit.

3 Results

3.1 Task 01 – Varying the proportional terms

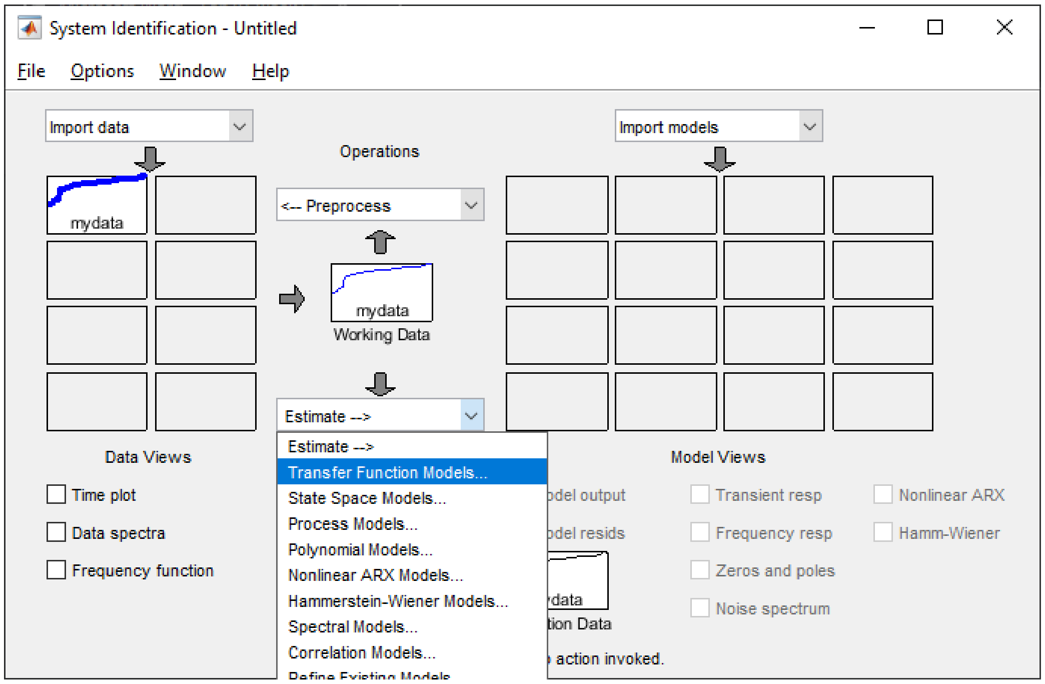


Figure 7: All transfer functions estimated in the System Identification Toolkit.

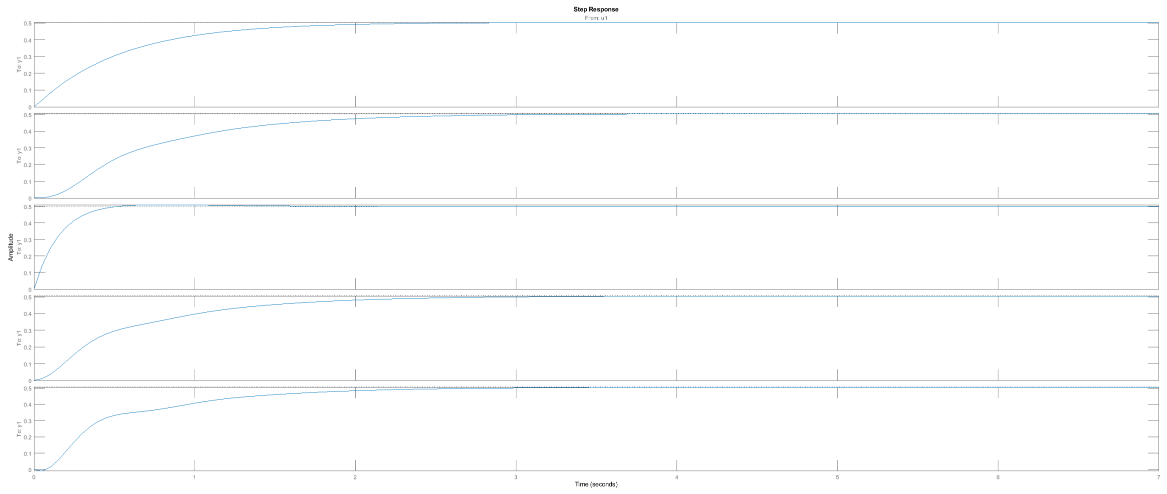


Figure 8: Step responses of all the transfer functions estimated.

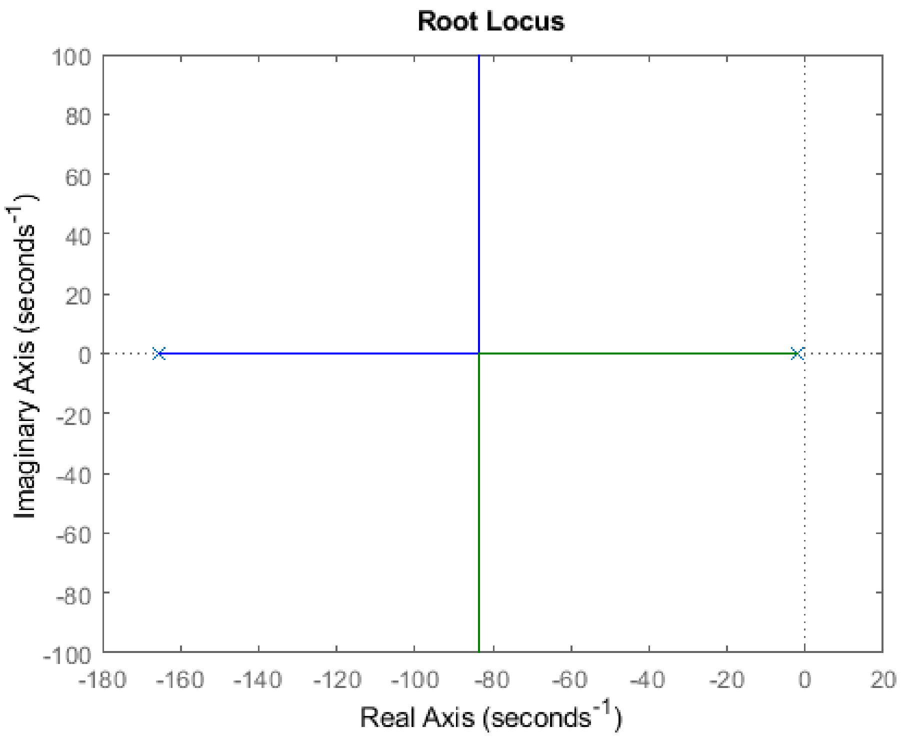


Figure 9: Root locus of 2^o transfer function with 0 zeroes.

This transfer function produces

- overdamped in $(0, 42.9)$
- critically damped at $\{42.9\}$
- underdamped in $(42.9, \infty)$

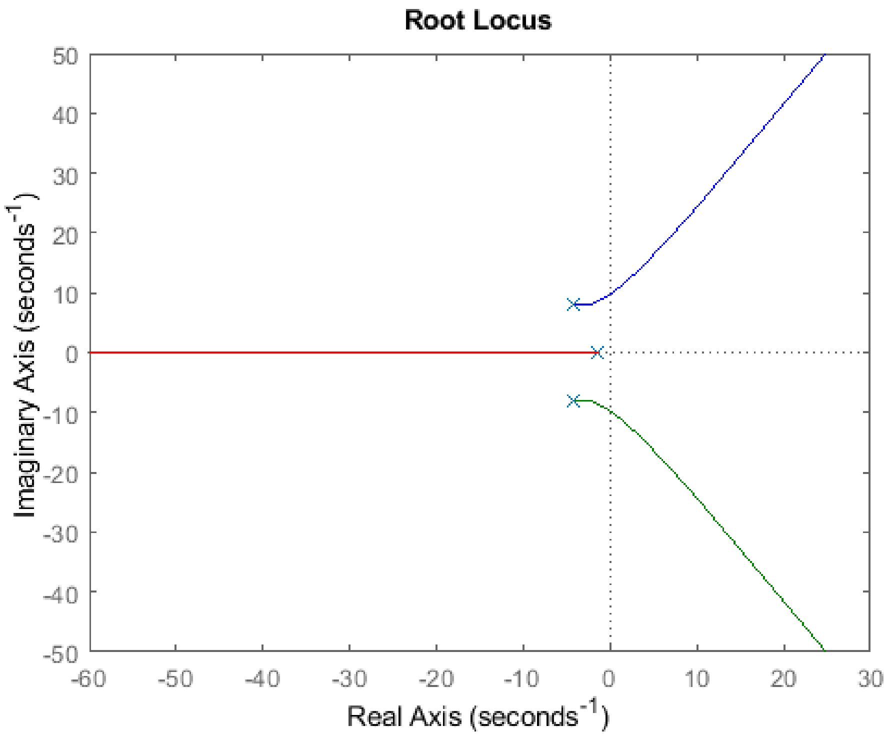


Figure 10: Root locus of 3^o transfer function with 0 zeroes.

This transfer function produces

- overdamped in $(0, \infty)$
- underdamped in $(0, 13)$

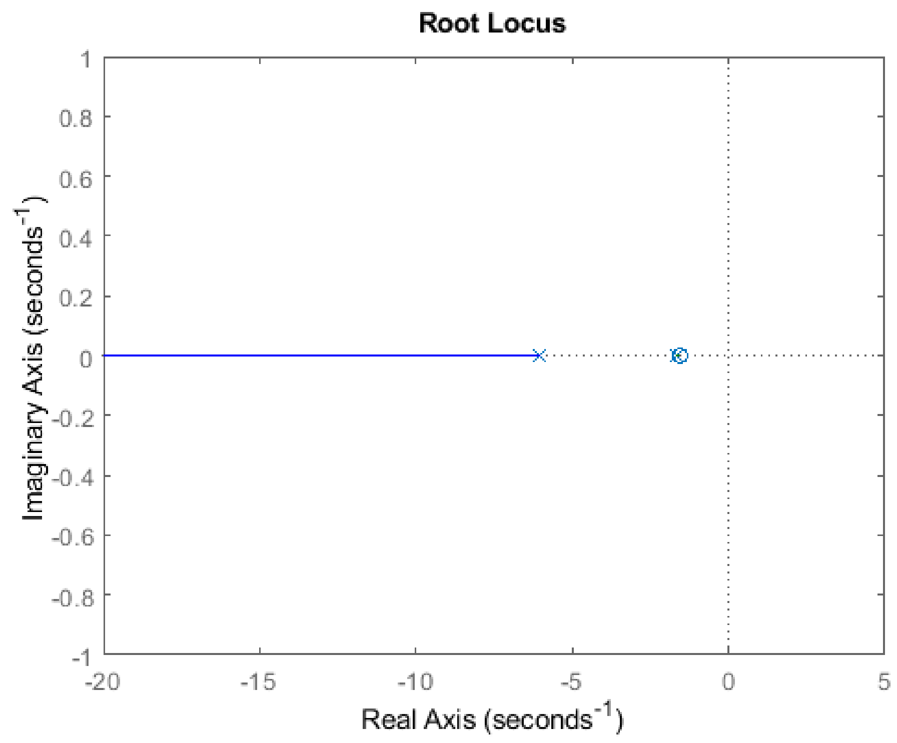


Figure 11: Root locus of 2^o transfer function with 1 zeroes.

This transfer function produces

- overdamped in $(0, \infty)$

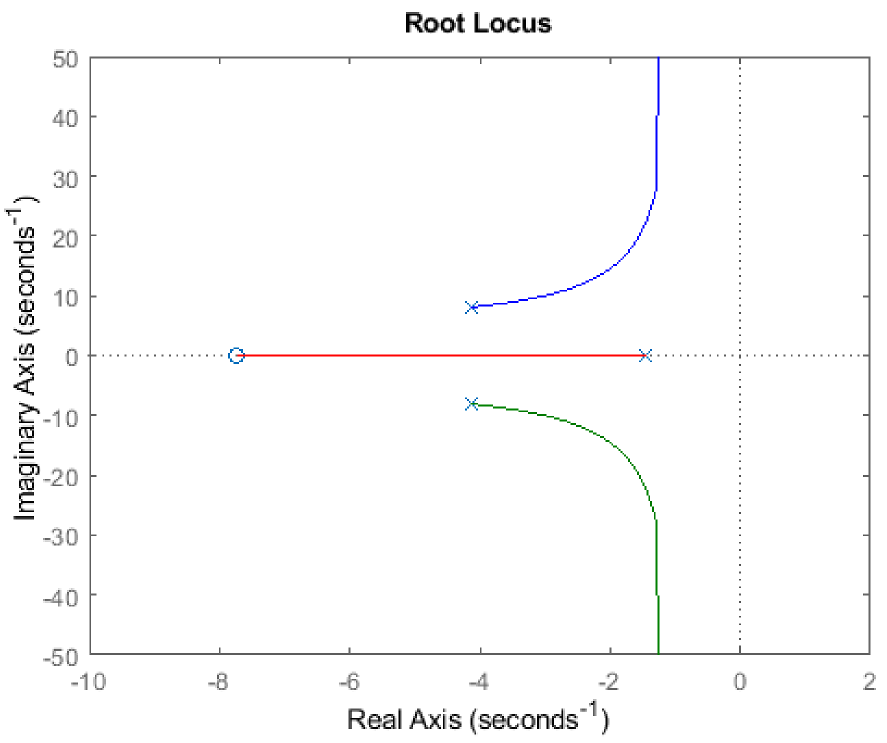


Figure 12: Root locus of 3^o transfer function with 1 zeroes.

This transfer function produces

- overdamped in $(0, \infty)$
- underdamped in $(0, \infty)$

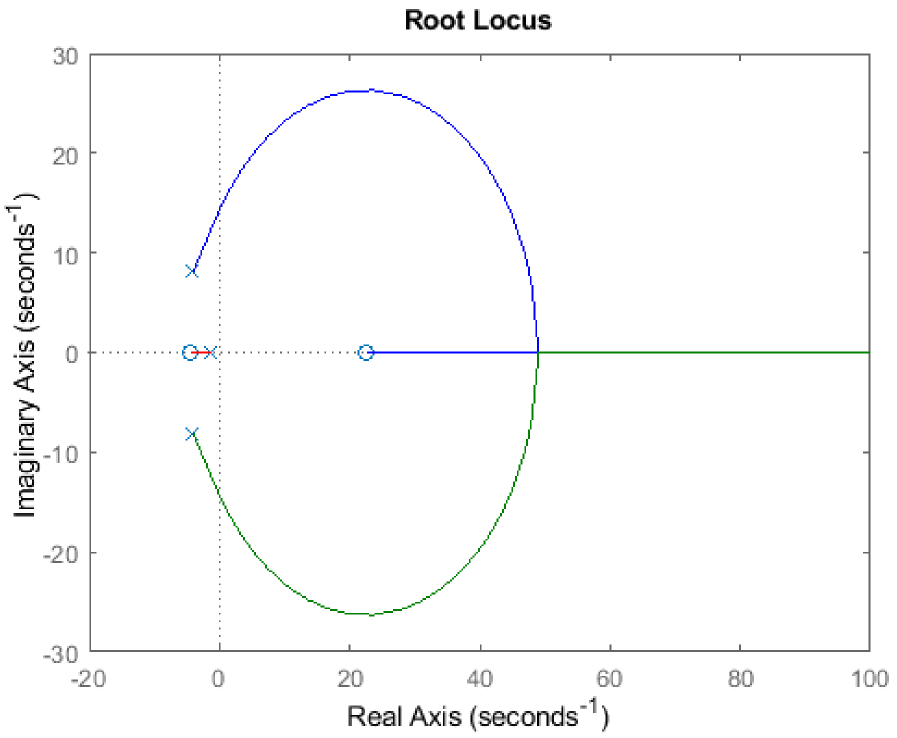


Figure 13: Root locus of 3^o transfer function with 2 zeroes.

This transfer function produces

- overdamped in $(0, \infty)$
- underdamped in $(0, 10.3)$

4 Discussion

(Note that I accidentally deleted the images used to setup the System Identification Toolkit, so I recycled the ones from the Simulink Control Design in Lab 09. I will fix this in the final version of Lab 10.)

A Appendix

A.1 Task 01 – Analyzing data and initial parameters, Matlab script

```
%% week01_task01_params.m
% Analyzing the input data.
% By       : Leomar Duran
% When     : 2023-04-19t21:40Q
% For      : ECE 3413 Classical Control Systems
%

clear

% load the time series data from the data file
output_load = load('Lab10_data.mat')

% get the time domain
t = output_load.raw.Time;
% number of samples
[Nsamp, ~] = size(t)
% starting time for runtime
StartTime = t(1)
% sampling time
Tsamp = (t(end) - StartTime)/Nsamp

% get the output data
output_raw = output_load.raw.Data;
% recreate the step input
input_step = ones(size(output_raw));
% as a time series
input_step_ts = timeseries(input_step, t)

% apply the median filter
% Find a way to filter the data to the best of your abilities and
% retain the rise time and settling time characteristics.
% However, to find the rise and settling time characteristics, we have
% to smoothen the curve in the first place.  In light of this, a median
% filter with 19o seems to smoothen out the function satisfactorily by
% inspection.
output_response = medfilt1(output_raw, 19);

% plot the data for inspection
hold on
plot(input_step_ts)
pOutput = plot(t, output_response);
pOutput.LineWidth = 8;
plot(output_load.raw)
hold off
title('signal vs time')
xlabel('time [s]')
ylabel('signal <1>')
legend(["step input", "filtered output", "unfiltered output data"])

% create workspace structure
data = table(input_step, output_response);
```

A.2 Task 03–07 – Estimating transfer functions based on order, plot step responses, root locus

```
%% week01_task03_07_step_rlocus.m
% Analyzing the input data.
% By       : Leomar Duran
% When     : 2023-04-26t22:53Q
% For      : ECE 3413 Classical Control Systems
%

clear

% load parameters
week01_task01_params

% encapsulate output and input data with time sampling
```

```
mydata = iddata(data.output_response, data.input_step, Tsamp)

% order of each transfer function
orders = [2 3 2 3 3]';
% # zeroes for each transfer function
zeroes = [0 0 1 1 2]';
% tabularize this data
tf_orders = table(orders, zeroes)

tf = [];

for tfIdx = 1:height(tf_orders)
    % load next row
    tf_order = tf_orders(tfIdx,:);
    % estimate next transfer function
    nextTf = tfest(mydata, tf_order.orders, tf_order.zeroes)
    % add next transfer function to array
    tf = [tf ;  nextTf];
    % plot the root locus
    figure
    rlocus(nextTf)
end

% plot the step responses
figure
step(tf, t)
```