ECE 450 - Homework #7

Collin Heist

October 8, 2019

1 ECE 450 - Homework #5

1.1 Problem 7.7.1

1.1.1 Package Imports

```
In [1]: import numpy as np
        import seaborn as sns
        import pandas as pd
        import matplotlib.pyplot as plt
        from scipy import signal as sig
        from control import margin, tf
```

1.1.2 Generic functions for the step, ramp, and parabolic response

```
In [2]: step = lambda times:     [0 if t < 0 else 1 for t in times]
        ramp = lambda times:     [0 if t < 0 else t for t in times]
        parabola = lambda times: [0 if t < 0 else t * t for t in times]
```

1.1.3 Generic function to convolve any number of equations

```
In [3]: def convolve_all(values):
            temp_conv = values[0]
            if len(values) > 1:
                for next_val in values[1:]:
                    temp_conv = np.convolve(temp_conv, next_val)

            return temp_conv
```

1.1.4 Generic function to generate the magnitude and phase of $H(j\omega)$ values

```
In [27]: def magnitude_phase_response(num, den, omega_range, omega_step=10, gain_num=None, gain_
             if isinstance(gain_num, (np.ndarray, list)) and isinstance(gain_den, (np.ndarray, l
                 num = convolve_all([num, gain_num])
                 den = convolve_all([den, gain_den])

             system = sig.lti(num, den)
             w, h_mag, h_phase = sig.bode(system, np.arange(omega_range[0], omega_range[1], omeg
             _, phase_margin, _, crossover_w = margin(h_mag, h_phase, w)
```

```
    df_list = []
    df = pd.DataFrame(list(zip(w, h_mag)), columns=["$\omega$", "Value"])
    df["Kind"] = "Magnitude Response"
    df_list.append(df)

    df = pd.DataFrame(list(zip(w, h_phase)), columns=["$\omega$", "Value"])
    df["Kind"] = "Phase Response"
    df_list.append(df)

    return pd.concat(df_list, ignore_index=True, axis=0), phase_margin, crossover_w
```

1.1.5 Generic function to obtain response of a system to inputs

```
In [5]: def response_to_inputs(num, den, input_funcs, input_names, time, gain_num=None, gain_den
            df_list = []

            # If a gain equation was given, adjust the system num / dun
            if isinstance(gain_num, (np.ndarray, list)) and isinstance(gain_den, (np.ndarray, li
                num = convolve_all([num, gain_num])
                den = convolve_all([den, gain_den])

            num = np.pad(num, (len(den) - len(num), 0), "constant") # Make arrays same length
            den = np.add(den, num)
            for in_name, in_f in zip(input_names, input_funcs):
                df = pd.DataFrame(list(zip(time, in_f(time))), columns=["Time", "Value"])
                df["Kind"] = df["Name"] = in_name
                df["Error"] = "Response"
                df_list.append(df)

                _, response, _ = sig.lsim((num, den), in_f(time), time)
                df = pd.DataFrame(list(zip(time, response)), columns=["Time", "Value"])
                df["Kind"] = in_name
                df["Name"] = in_name + " - Response"
                df["Error"] = "Response"
                df_list.append(df)

                response_err =  np.subtract(in_f(t), response)
                df = pd.DataFrame(list(zip(time, response_err)), columns=["Time", "Value"])
                df["Kind"] = in_name
                df["Name"] = in_name + " - Error"
                df["Error"] = "Error"
                df_list.append(df)

            return pd.concat(df_list, ignore_index=True, axis=0)
```

1.1.6 Generic function to plot the responses of a system

```
In [110]: def create_plots(df, error=False):
              if error:
                  sns.set(style="whitegrid", font_scale=2.75)
                  g = sns.FacetGrid(df, hue="Name", row="Kind", col="Error", height=7.5, aspect=
                  g.map(sns.lineplot, "Time", "Value", **dict(linewidth=2.5)).add_legend().despi
              else:
                  sns.set(style="whitegrid", font_scale=1.5)
                  g = sns.FacetGrid(df, hue="Kind", row="Kind", height=5.5, aspect=1.75,
                                    sharey=False, gridspec_kws={"hspace":0.3})
                  g.map(sns.lineplot, "$\omega$", "Value", **dict(linewidth=2.5)).add_legend().d

              return g
```

df

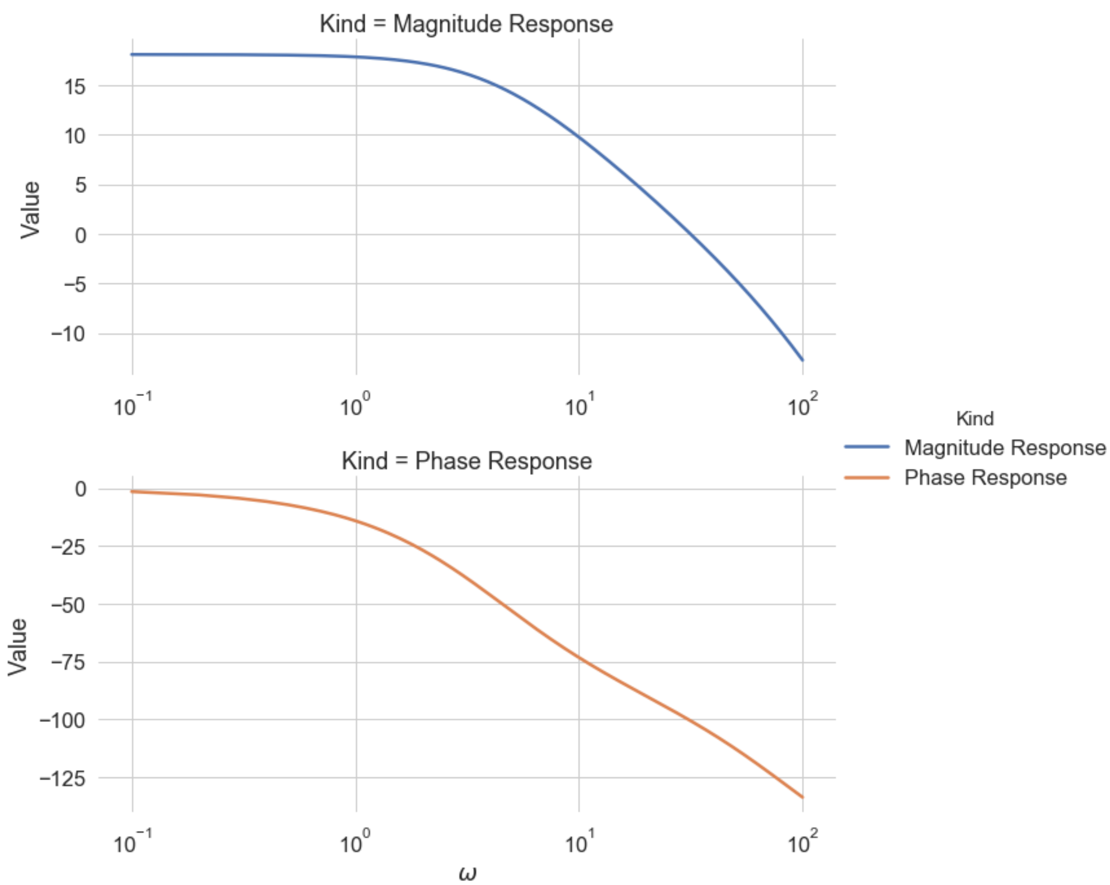
```
In [244]: def compute_phase_lead(df, desired_pm):
              alpha = (1 + np.sin(desired_pm * np.pi / 180)) / (1 - np.sin(desired_pm * np.pi /
              alpha_db = -10 * np.log10(alpha)
              omega_m = df["Value"].iloc[(df["Value"] - alpha_db).abs().argsort()[:1].values[0]]
              omega_p = np.sqrt(alpha) * omega_m
              omega_z = omega_m / np.sqrt(alpha)

              return omega_z, omega_p
```

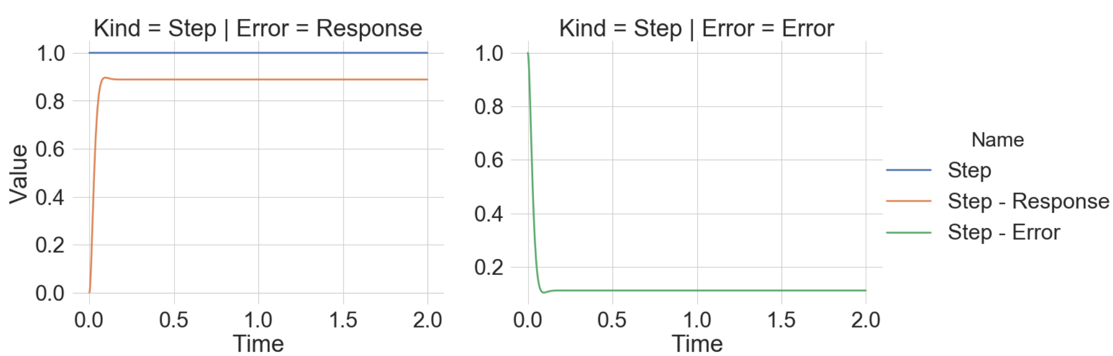
1.1.7 My Solution

```
In [111]: num = [400 * 8]
          den = convolve_all([[1, 100, 400]])

          df, m, w = magnitude_phase_response(num, den, [0.1, 10 ** 2], 0.1)
          g = create_plots(df)
          for ax in g.axes.flatten():
              ax.tick_params(labelbottom=True)
```




```
In [71]: t = np.arange(0, 2, 0.001)
         df = response_to_inputs(num, den, [step], ["Step"], t)
         create_plots(df, True);
```

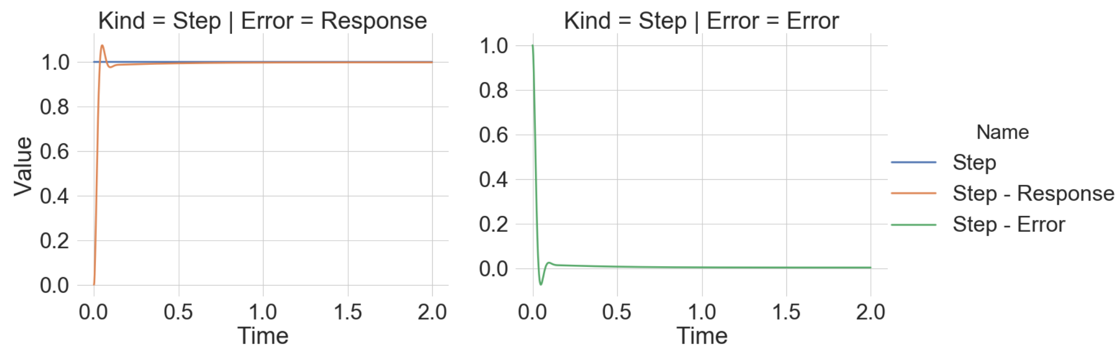


It is now time to design the phase lag network. I'll start with a network designed as such:

$$G_c(s) = 2 \cdot \frac{s+3}{s+0.1}$$

```
In [72]: comp_network_num = np.multiply(2, [1, 3])
         comp_network_den = [1, 0.1]

         t = np.arange(0, 2, 0.001)
         df = response_to_inputs(num, den, [step], ["Step"], t, comp_network_num, comp_network_d
         create_plots(df, True);
```



This seems to satisfy the problem requirements, but let's look at the steady-state error at one second, just to be sure.

```
In [73]: display(df[(df["Time"] == 1) & (df["Error"] == "Error")])
```

```
      Time     Value  Kind          Name  Error
5000   1.0  0.003001  Step  Step - Error  Error
```

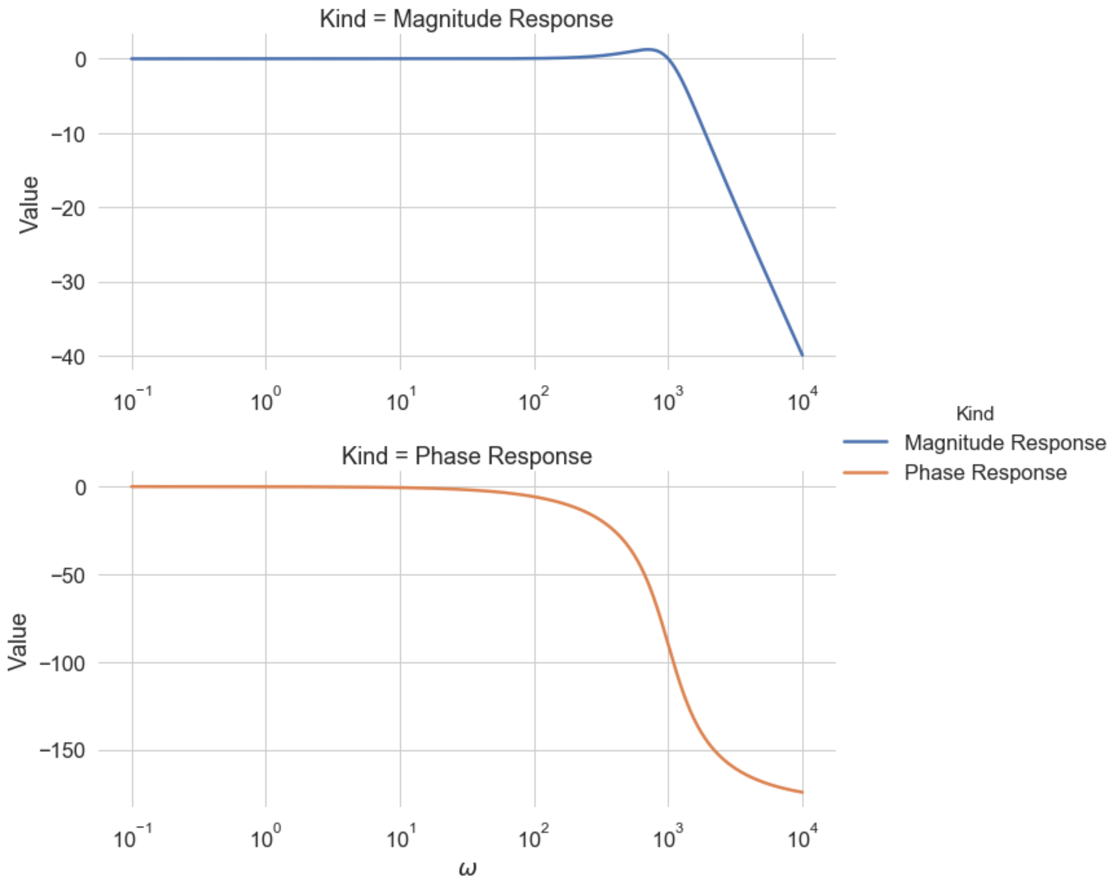
As we can see, at 1 second, the step error for this new transfer function is below 1%, and is actually 0.3%.

1.2 Problem 7.7.2

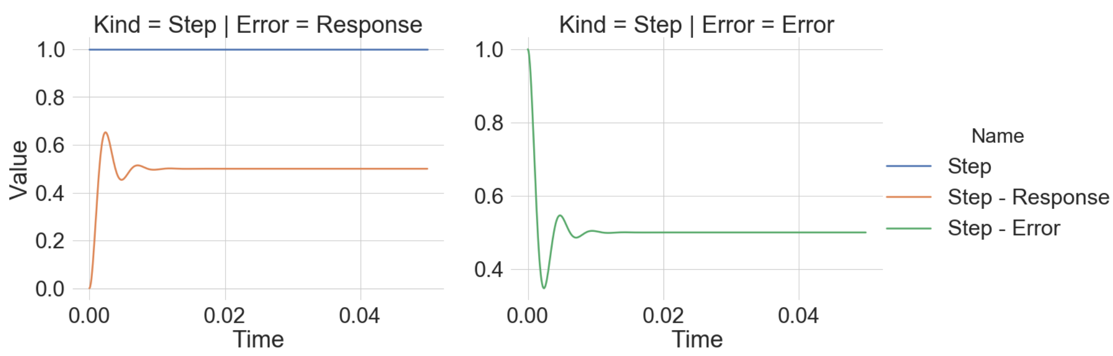
```
In [232]: num = [10 ** 6]
          den = [1, 10 ** 3, 10 ** 6]

          df, phase_margin, crossover_w = magnitude_phase_response(num, den, [0.1, 10 ** 4], 1)
          print ("Crossover at {:.3f} (rad/s) has a phase-margin of {:.3f} degrees".format(cross
          g = create_plots(df)
          for ax in g.axes.flatten():
              ax.tick_params(labelbottom=True)
```

```
Crossover at 1101.540 (rad/s) has a phase-margin of -100.964 degrees
```



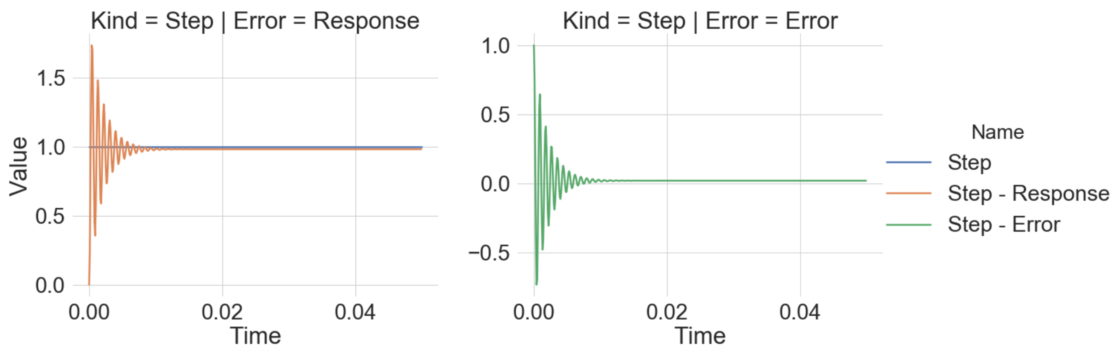

```
In [233]: t = np.arange(0, 0.05, 0.0001)
          df = response_to_inputs(num, den, [step], ["Step"], t)
          create_plots(df, True);
```



For starters, I will add a generic gain of 50 to reduce the error towards 1%.

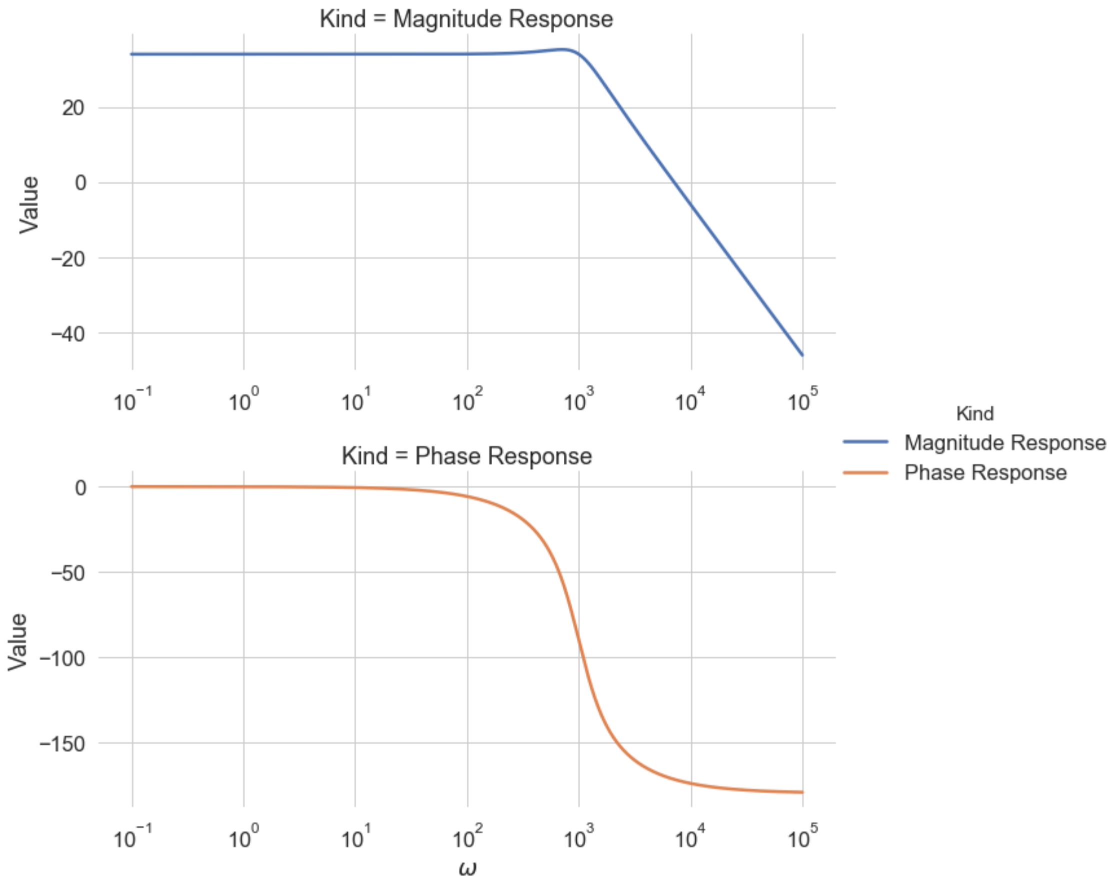
```
In [234]: comp_network_num = [50]
          comp_network_den = [1]
```

```
t = np.arange(0, 0.05, 0.0001)
df = response_to_inputs(num, den, [step], ["Step"], t, comp_network_num, comp_network_
create_plots(df, True);
```




```
In [235]: df, phase_margin, crossover_w = magnitude_phase_response(num, den, [0.1, 10 ** 5], 1,
          print ("Crossover at {:.3f} (rad/s) has a phase-margin of {:.3f} degrees".format(cross
          g = create_plots(df)
          for ax in g.axes.flatten():
              ax.tick_params(labelbottom=True)
```

```
Crossover at 6712.237 (rad/s) has a phase-margin of 8.663 degrees
```



However, this clearly introduced a lot more sinusoidal overshoot than desired. To address this, I will increase the phase margin using a phase lead network, as follows:

```
In [245]: w_z, w_p = compute_phase_lead(df, 55 - 9)
          print (w_z, w_p)
```

```
-3.180602276292162 -19.484544261264286
```

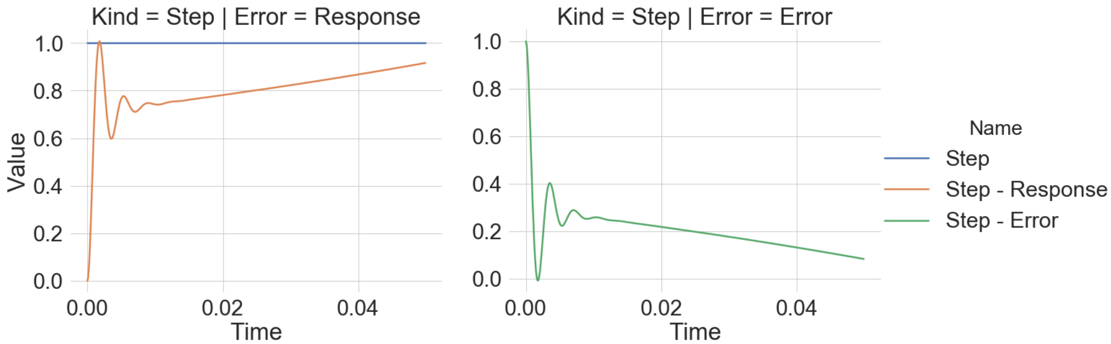
13602 radians per second, according to the chart. Thus $\omega_m = 13602$

$$\omega_p = 50205.94, \omega_z = 3685.11$$

$$G_c(s) = 50 \cdot \frac{50205.94}{3685.11} \cdot \frac{s + 3685.11}{s + 50205.94}$$

```
In [275]: comp_network_num = np.multiply(2.4 , [1, w_z])
          comp_network_den = [1, w_p]

          t = np.arange(0, 0.05, 0.0001)
          df2 = response_to_inputs(num, den, [step], ["Step"], t, comp_network_num, comp_network
          create_plots(df2, True);
```


Kind = Step | Error = Response
Kind = Step | Error = Error
Value
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.02
0.04
Time
Name
Step
Step - Response
Step - Error