ECE 450 - Homework #8

Collin Heist

October 17, 2019

1 ECE 450 - Homework #8

1.1 Problem 8.1.2

1.1.1 Package Imports

```
In [67]: 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
         import warnings
         warnings.filterwarnings('ignore')
```

1.1.2 Generic function to generate the $H(s)$ for a Buttersworth Filter of a given n poles

```
In [24]: def buttersworth(order):
             pole_list = [np.sin((np.pi * (2 * k - 1))/(2 * order)) + complex(0, np.cos((np.pi *
             s_pole_list = [[1, pole] for pole in pole_list]

             return convolve_all(s_pole_list)
```

1.1.3 Generic function to get the minimum n for a desired pass band cut-off

```
In [229]: def minimum_n_passband(deviation, omega_p):
              return int(np.ceil(np.log10((1 / (1 - deviation) ** 2) - 1) / (2 * np.log10(omega_
```

1.1.4 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
```

1.1.5 Generic function to solve a set of state matrices

```
In [ ]: def state_solver(A_matrix, B_matrix, force_function, initial_conditions,
                         time_range=[0, 10], dt=0.01):
            time_values = np.arange(time_range[0], time_range[1], dt)
            x_vals = np.array(initial_conditions)
            state_variables = [[] for _ in range(len(initial_conditions))]

            # Loop through each instance in time, calculate the state variable at that time
            for time in time_values:
                if isinstance(force_function(time), np.ndarray):
                    x_vals = x_vals + dt * (A_matrix @ x_vals) + dt * (B_matrix @ force_function
                else:
                    x_vals = x_vals + (dt * (A_matrix @ x_vals)) + dt * (B_matrix * force_functi
                for index, _ in enumerate(state_variables):
                    state_variables[index].append(x_vals[index][0])

            return state_variables, time_values
```

1.1.6 Generic function to plot the responses of a system

```
In [93]: # Color list for multiple lines on each subplot
         colors = ["red", "blue", "green", "gray", "purple", "orange"]
         step_size = 0.005

         # Generic Function to create a plot
         def create_plot(x, y, xLabel=["X-Values"], yLabel=["Y-Values"],
                         title=[("Plot", )], num_rows=1, size=(18, 14), logx=False):
             plt.figure(figsize=size, dpi=300)
             for c, (x_vals, y_vals, x_labels, y_labels, titles) in enumerate(zip(x, y, xLabel,
                 for c2, (y_v, t) in enumerate(zip(y_vals, titles)):
                     plt.subplot(num_rows, 1, c + 1)
                     # Add a plot to the subplot, use transparency so they can both be seen
                     plt.plot(x_vals, y_v, label=t, color=colors[c2], alpha=0.70)
                     plt.ylabel(y_labels)
                     plt.xlabel(x_labels)
                     plt.grid(True)
                     plt.legend(loc='lower right')
                     if logx:
                         plt.xscale("log")

             plt.show()
```

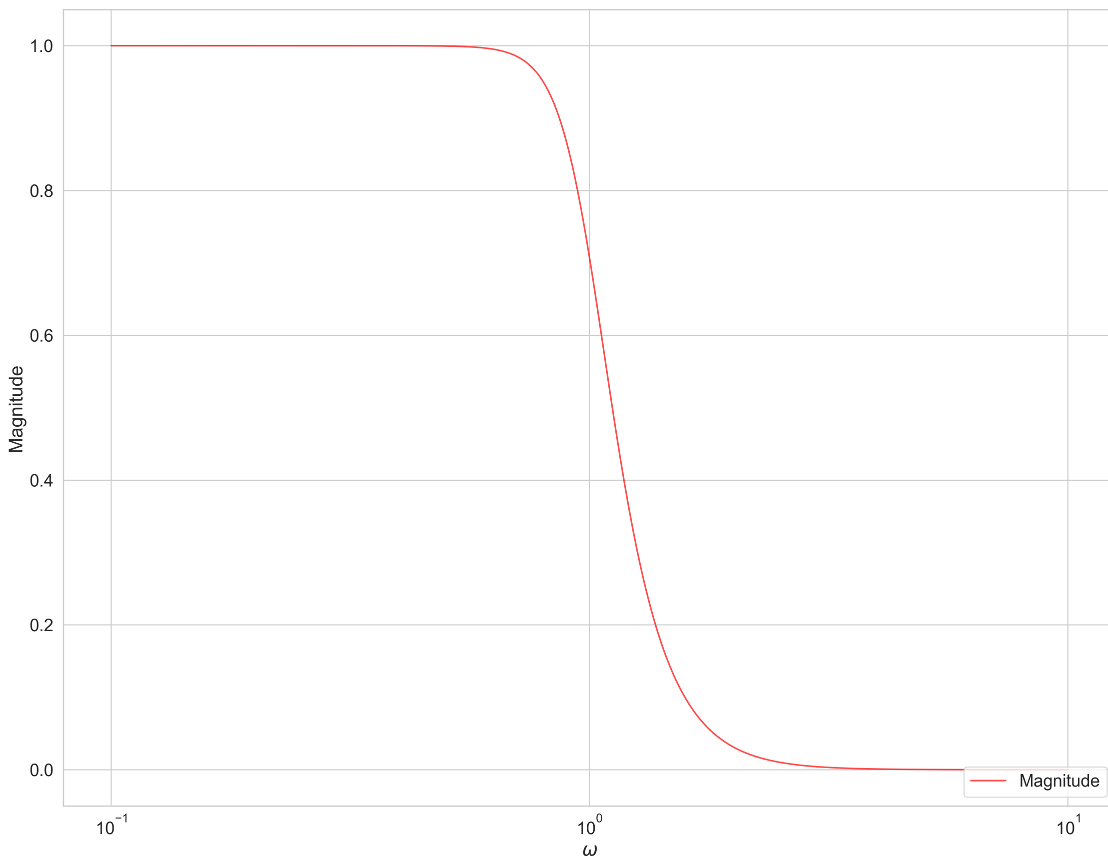
1.1.7 Generic function to generate the magnitude and phase of $H(j\omega)$ values

```
In [166]: 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,
                  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], ome
    _, phase_margin, _, crossover_w = margin(h_mag, h_phase, w)

    return w, h_mag, h_phase, phase_margin, crossover_w
```

1.2 Problem 8.1.2

In [225]:
```
num = [1]
den = np.real(buttersworth(5))

w, mag, _ = sig.bode(sig.lti(num, den), np.arange(10 ** -1, 10, 0.0005))
create_plot([w], [(10**(0.05*mag), )], ["$\omega$"], ["Magnitude"], [("Magnitude", )],
```



In [215]:
```
df = pd.DataFrame(np.asarray([w, 10**(0.05*mag)]).T, columns=["w", "value"])
df[round(df["value"], 3).isin([0.95, 0.707, 0.1])]
```

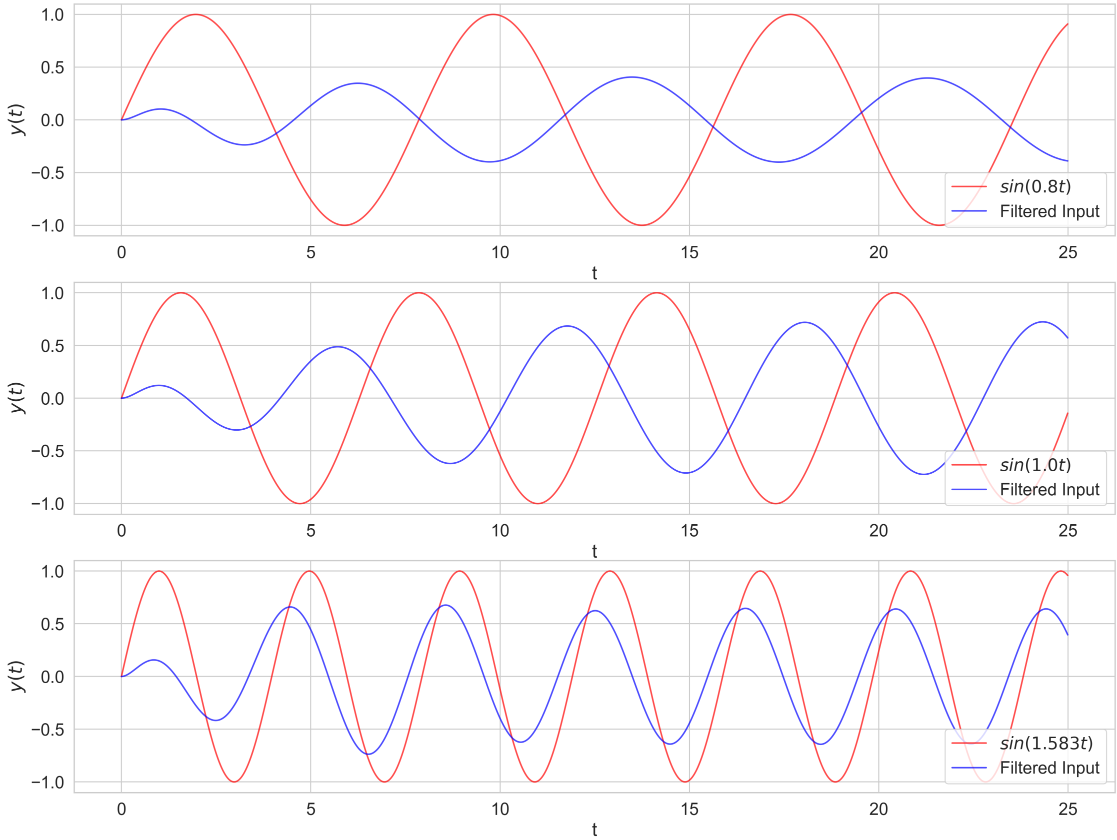
Out[215]:

	w	value
1400	0.8000	0.950283

```
1401  0.8005  0.949994
1402  0.8010  0.949704
1800  1.0000  0.707107
2964  1.5820  0.100408
2965  1.5825  0.100251
2966  1.5830  0.100094
2967  1.5835  0.099938
2968  1.5840  0.099782
2969  1.5845  0.099626
```

The above values show that the frequencies that correspond to the output amplitudes of 0.95, 0.707, and 0.1 are 0.8, 1, and 1.583 (rad/s) accordingly.

```
In [223]: input0950 = lambda t: np.sin(0.8 * t)
          input0707 = lambda t: np.sin(1 * t)
          input0100 = lambda t: np.sin(1.583 * t)

          a, b, c, d = sig.tf2ss(num, den)
          name, t_range, dt = "Filtered Input", [0, 25], 0.01
          vars0950, t = state_solver(a, b, input0950, [0, 0, 0, 0, 0], t_range, dt)
          vars0707, _ = state_solver(a, b, input0707, [0, 0, 0, 0, 0], t_range, dt)
          vars0100, _ = state_solver(a, b, input0100, [0, 0, 0, 0, 0], t_range, dt)
          create_plot([t, t, t],
                      [(input0950(t), vars0950[0]), (input0707(t), vars0707[0]), (input0100(t),
                      ["t", "t", "t"], ["$y(t)$", "$y(t)$", "$y(t)$"],
                      [("$sin(0.8t)$", name), ("$sin(1.0t)$", name), ("$sin(1.583t)$", name)], 3
```

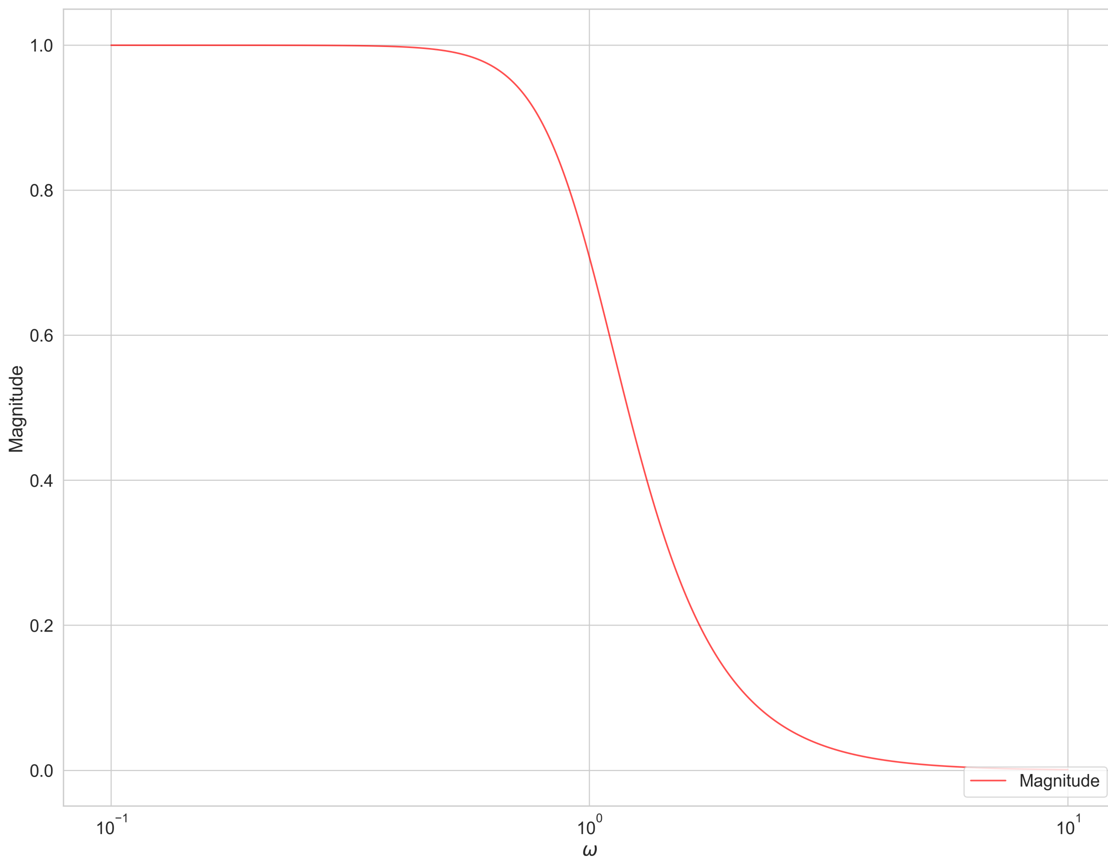


This time-domain simulation does not match the frequency domain amplitude plot.

1.3 Problem 8.1.3

```
In [232]: min_order = minimum_n_passband(0.2, 0.9)
          print ("The minimum order required is n =", min_order)

          num = [1]
          den = np.real(buttersworth(min_order))

          w, mag, _ = sig.bode(sig.lti(num, den), np.arange(10 ** -1, 10, 0.0005))
          create_plot([w], [(10**(0.05*mag), )], ["$\omega$"], ["Magnitude"],
                      [("Magnitude", )], 1, logx=True)
```

```
The minimum order required is n = 3
```


1.0
0.8
0.6
0.4
0.2
0.0
Magnitude
10^{-1}
10^{0}
10^{1}
ω
Magnitude