Drew Seidel
Cary Renzema
ECE 508 - Python Workshop
Friday, June 9, 2023

Pole/Zero Processor Report/Results

Overview:

The repository organization is outlined in the README, and corresponding documentation is commented throughout the source code. Instructions for running are also in the README, but in general, main.py is run in the src directory.

This report will go through some sample inputs, and corresponding outputs.

The primary functionality is that users will provide a function in the frequency domain, given pole and zero values, and the ‘s’ symbol for the variable to be substituted with $j\omega$.

The user will also provide a limits dictionary describing limits for each pole/zero. One value is a ‘typical’ value, two is ‘minimum’ and ‘maximum’, three is ‘minimum’, ‘typical’, ‘maximum. Both a tuple, or, a singular number can be supplied for limits with only ‘typical’ values.

All user modifications are made in src/test.py.

Here are a few examples:

```python
# single pole low pass 1k rad/s
p1 = Symbol("p1")
sp = 1 / (s * p1 + 1)
limits = {"p1": 1/1e3}
```

```python
# single pole low pass 1M rad/s, 1k rad/s, 10 rad/s
# gain of 1, 2, 3
p1 = Symbol("p1")
z1 = Symbol("z1")
sp = 1 * z1 / (s * p1 + 1)
limits = {"p1": (1/1e6, 1/1e3, 1/1e1), "z1": (1, 2, 3)}
```

The code will parse the inputs, ensure everything is provided, in order, and will interface with `ExpressionClass` to product magnitude (with optional annotations), phase, and time response driven by a unit step (1/s) in the frequency domain. Both the frequency range and time range are calculated for the given transfer functions. All three plots will have their own window due to the amount of information displayed, with an appropriate legend.

Interfacing with the class is done in the following way (in process_fs in process.py).

```
fs = ExpressionClass(  # instantiate ExpressionClass with parsed values
limits=limit_values,
pz=keys,
s_var=s_var,
s_func=s_domain_func,
type=limit_values_type,
)
fs.process_bode()
fs.plot_bode(annotate_plot=True)
fs.process_time_domain()
fs.plot_time_domain()
fs.display_all_plots()
```

If there are too many poles/zeros, the annotations can be cumbersome, so they be turned off in `fs.plot_bode()`, line 69 of process.py.

Some important design considerations or changes to spec:

- Dynamic for-looping needed. The number of poles/zeros in provided determine the number of looping iterations. For example, if two poles are supplied (each with a supplied min, typ, and max) value, then two for loops are needed (with last pole in the supplied dict being the most inner). However, four poles supplied would require four loops, etc. to ensure all possible combinations are iterated through. To do this effectively and dynamically, itertools is utilized, and is heavily documented in expression_class.py
- The spec initially requested that an unsettled time-domain response show the last 5 cycles of oscillation. However, considering the case that some combinations of poles/zeros produce a settled response, and may produce and unsettled response, and all lines are shown on the same figure for easy comparison, the unsettled response will just be shown in the same time frame as the unsettled.
- Timeout for inability to process inverse Laplace transform. If sympy cannot process the inverse Laplace transform within 20 seconds, the program will terminate.

Program Responses:

Single-pole low-pass filter at 1 rad/s (compared with MATLAB for ground truth):

```
# single pole low pass 1k rad/s
p1 = Symbol("p1")
sp = 1 / (s * p1 + 1)
limits = {"p1": 1/1e3}
```

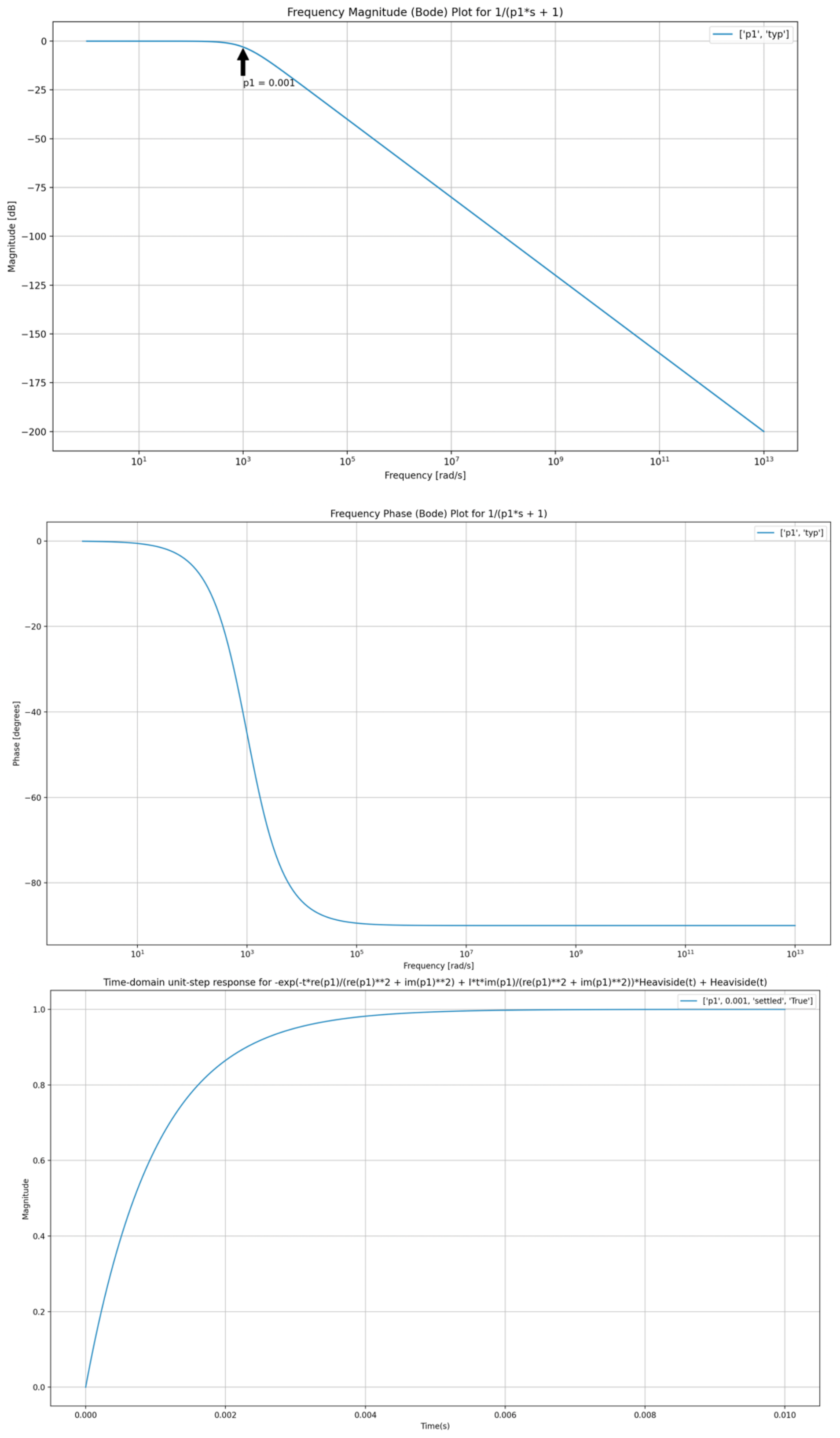


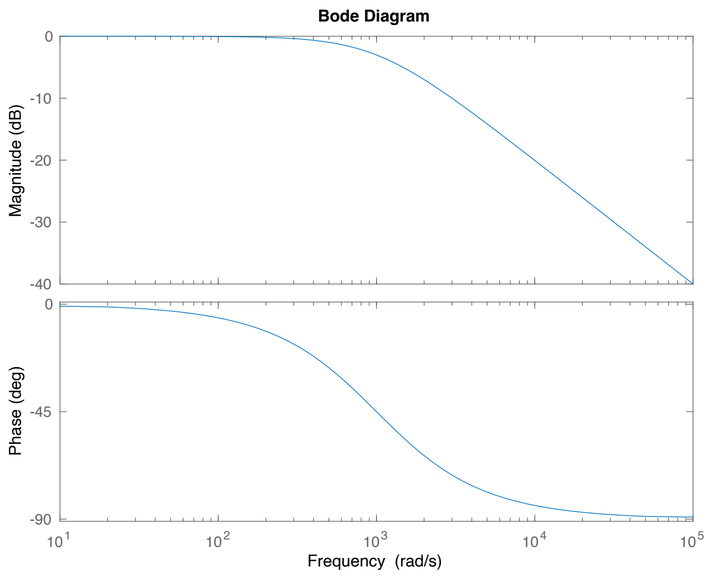
Figure 1: Magnitude, phase, and step response for 1k rad/s low-pass single-pole filter

To verify that this response is correct, this simple example can be replicated in MATLAB.

```
s = tf('s'); %single-pole low pass at 1k rad/s test
p1 = 1/1e3;
sp = 1/(s*p1 + 1);

figure
bodeplot(sp) %bode plot

figure()
step(sp);    %step response
```



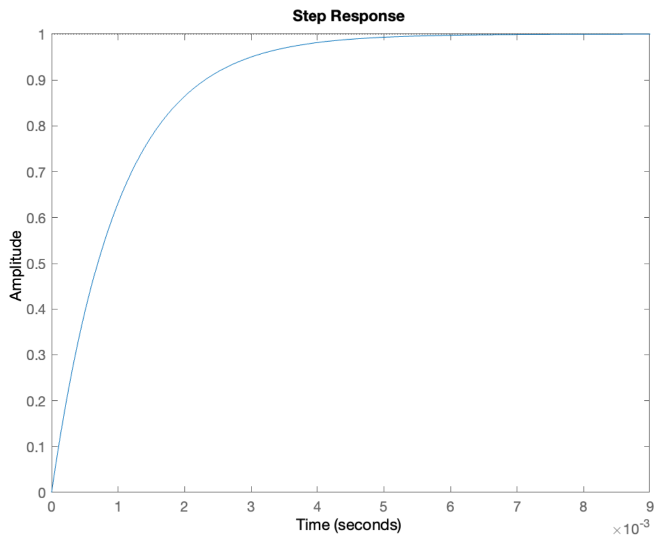
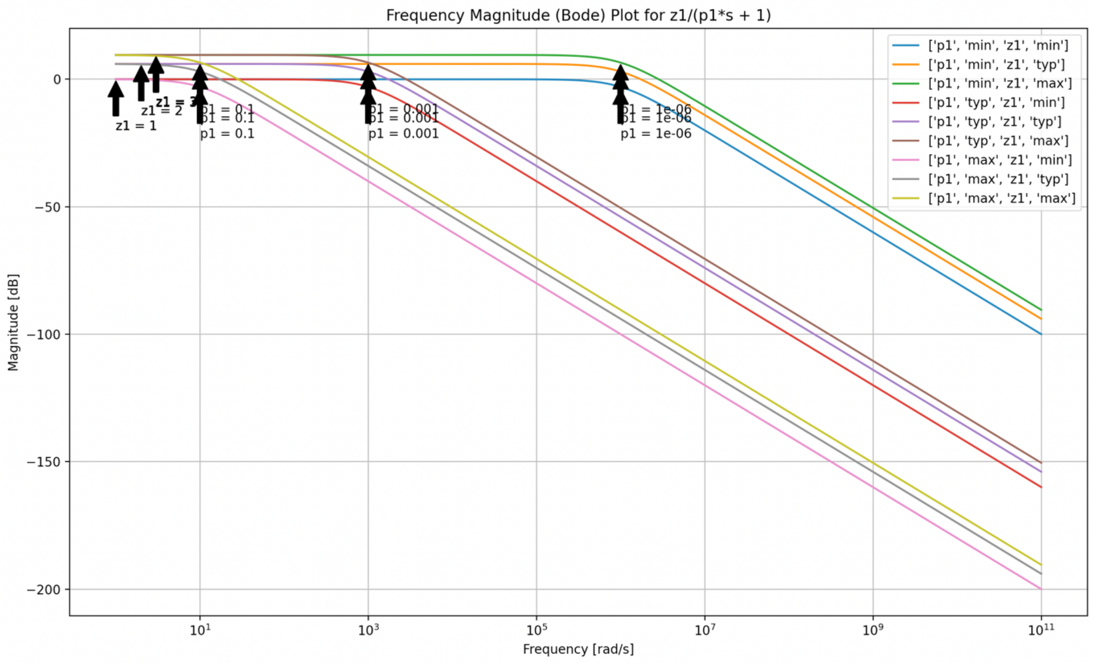


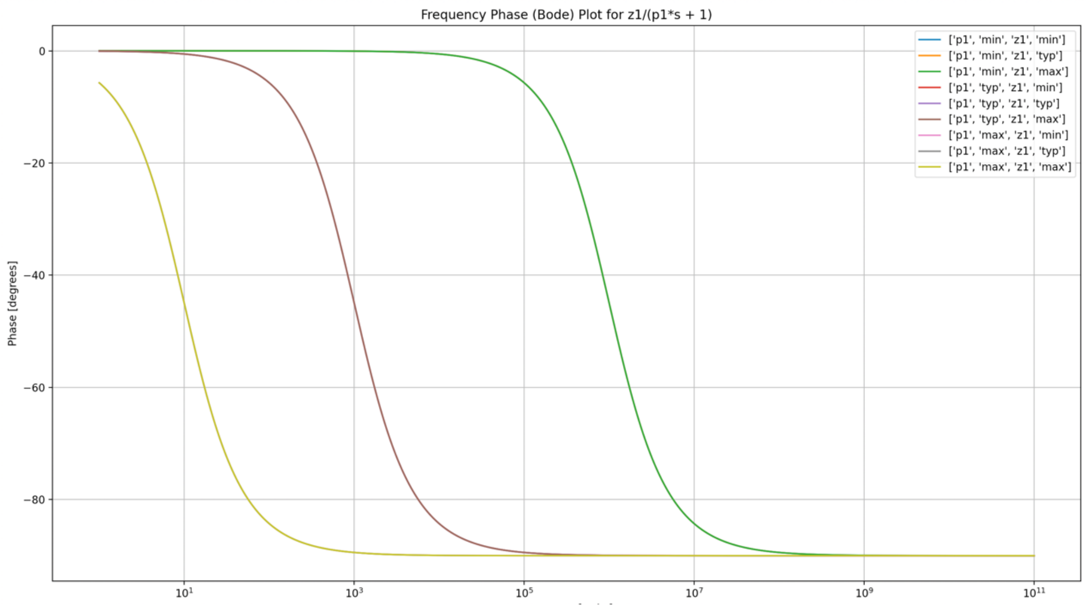
Figure 2: MATLAB verification of program functionality

MATLAB has verified the functionality of the plots produced in Figure 1.

Single-pole low-pass filter at 1 rad/s, 1k rad/s, 1M rad/s, with gain of 1,2,3:

```
# single pole low pass 1M rad/s, 1k rad/s, 10 rad/s
# gain of 1, 2, 3
p1 = Symbol("p1")
z1 = Symbol("z1")
sp = 1 * z1 / (s * p1 + 1)
limits = {"p1": (1/1e6, 1/1e3, 1/1e1), "z1": (1, 2, 3)}
```





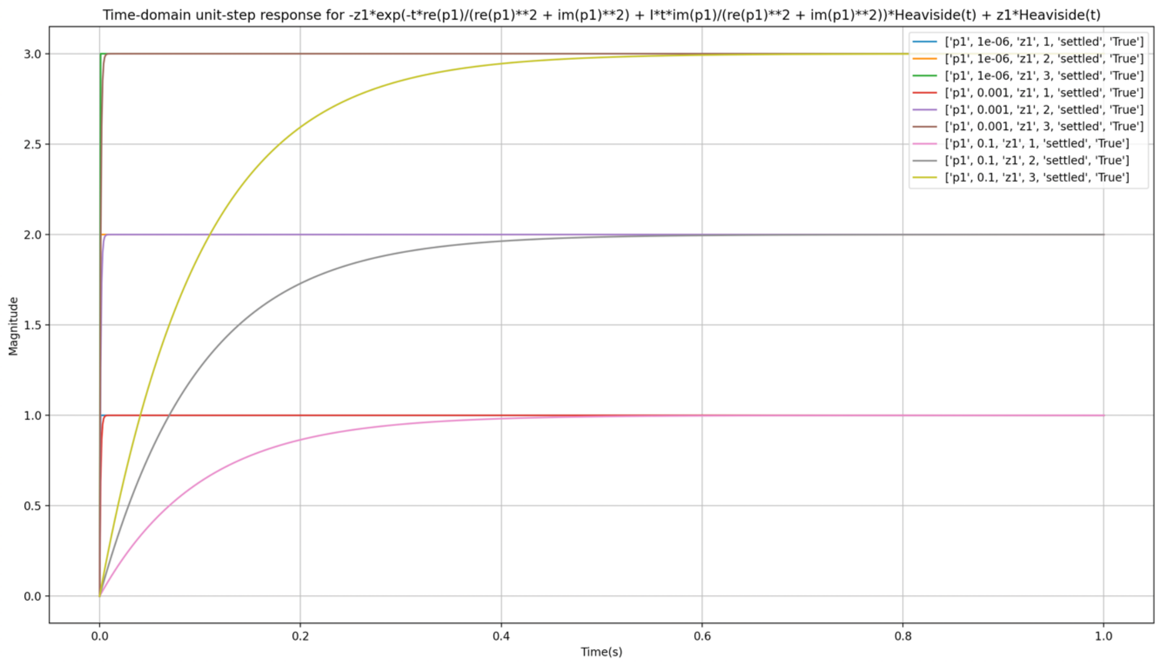


Figure 3: Magnitude, phase, and step response for 1 rad/s, 1k rad/s, 1M rad/s low-pass single-pole filter with gain of 1, 2, and 3

Bandpass response with $\omega_0 = 1k\ rad/s,\ Q = 10,\ Gain = 1,1000$

$$H(s) := G \cdot \frac{\frac{\omega_0}{Q} \cdot s}{s^2 + \frac{\omega_0 \cdot s}{Q} + \omega_0^2}$$

Where in this case it will be modeled as:

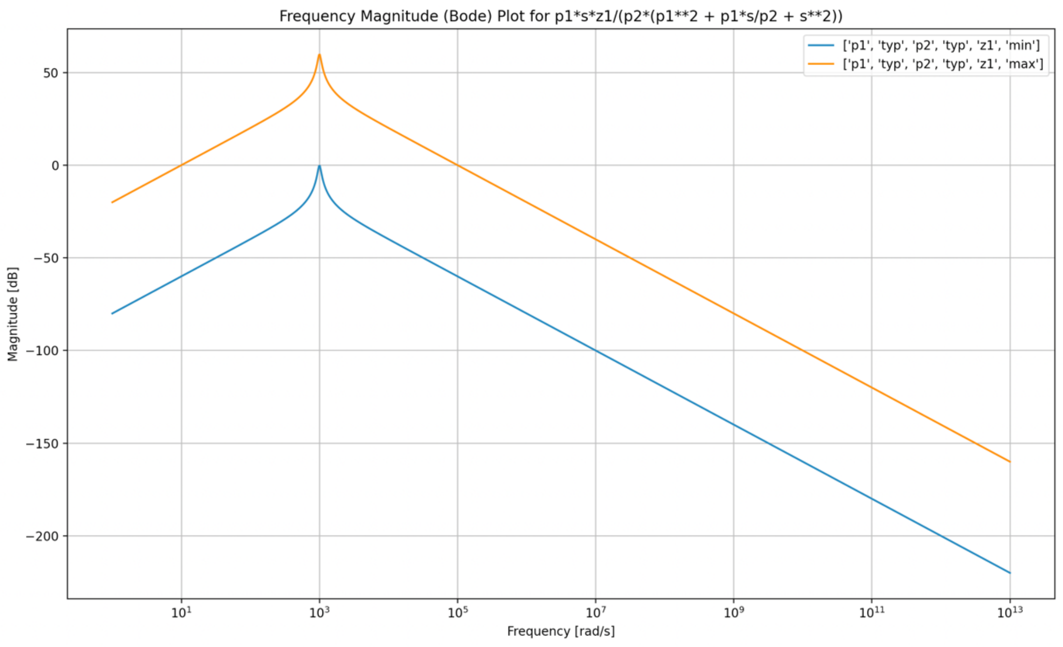
$p1 = \omega_0$
$p2 = Q$
$z1 = G$

```
# band-pass filter with wc (p1) = 1k rad/s, Q(p2) = 10, Gain(z1) = (1,1000)
# sympy cannot process inverse laplace of unit-step driven function
# to look at bode plots, uncomment lines 70,71 of process.py
p1 = Symbol("p1")
p2 = Symbol("p2")
z1 = Symbol("z1")
sp = z1 * ((p1/p2) * s) / (s ** 2 + ((p1 * s) / p2) + p1 ** 2)
limits = {"p1": (1e3,), "p2": 10, "z1": (1, 2, 3)}
```

Running this, sympy could not process the inverse Laplace function, but the class interaction can be modified to look at the magnitude/phase response. This also demonstrates the ability for both tuple, and single number input for typical-only responses, as well as (min, max) inputs as well.

```
fs.process_bode()
fs.plot_bode(annotate_plot=False)
#fs.process_time_domain()
#fs.plot_time_domain()
fs.display_all_plots()
```

Note annotations are also turned off here, as the format does not follow typical pole/zero convention as poles are in the numerator in this representation, etc.



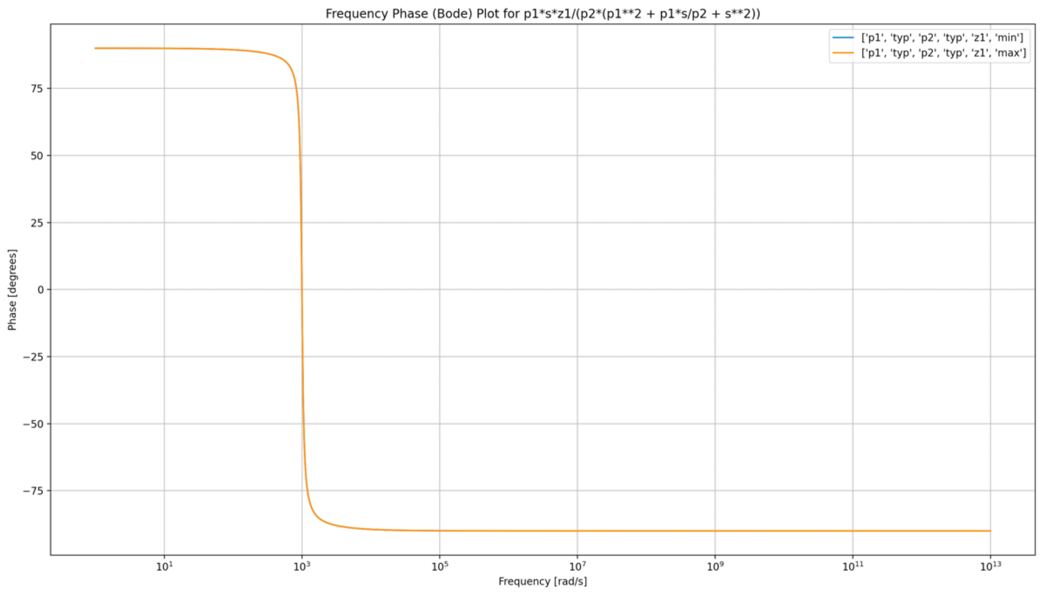
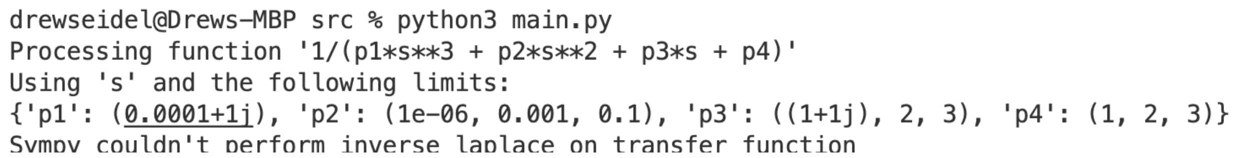


Figure 4: Band-pass response with Q = 10, $\omega_0 = 1k\ rad/s, G_{min} = 1, G_{max} = 1000$

Unknown complex response:

```
# random/unknown response, expression, test complex input
p1 = Symbol("p1")
p2 = Symbol("p2")
p3 = Symbol("p3")
p4 = Symbol("p4")
sp = 1 / (
(p1 * (s**3)) + (p2 * (s**2)) + (p3 * s) + p4
)
limits = {"p1": 1/10e3+1j, "p2": (1/1e6, 1/1e3, 1/1e1), "p3": (1+1j, 2, 3), "p4":
(1, 2, 3)}
```



```
drewseidel@Drews-MBP src % python3 main.py
Processing function '1/(p1*s**3 + p2*s**2 + p3*s + p4)'
Using 's' and the following limits:
{'p1': (0.0001+1j), 'p2': (1e-06, 0.001, 0.1), 'p3': ((1+1j), 2, 3), 'p4': (1, 2, 3)}
Sympy couldn't perform inverse laplace on transfer function
```

Figure 5: Terminal output of a function that sympy cannot perform the inverse Laplace on the transfer function (within 20 seconds)