A digital Butterworth filter will be designed to meet the following specification with a time step of $\delta t = 0.01s$:

$$\text{Pass band (-3dB): } 1\ rad/s < \omega < 5\ rad/s$$
$$\text{Stop band (-20dB): } \omega < 0.3\ rad/s, \quad \omega > 15\ rad/s$$

Create Analog Filter

A bandpass Butterworth filter is typically designed by creating a lowpass filter, scaling it to the bandwidth, and transforming it to a bandpass filter. In this case, the bandwidth ($5 - 1 = 4$ rad/s) is higher than the center frequency ($\frac{5+1}{2} = 3$ rad/s), so it will be better to cascade a high pass filter and a low pass filter.

The lowpass section extends from the cutoff frequency of 5 rad/s to 15 rad/s, a roll-off of about -35 dB/dec ($\frac{20-3}{\log(15/5)=35}$), so a two-pole filter should suffice. The high pass section from 0.3 rad/s to 1 rad/s has a roll-off of about -20 dB/dec, but I found that a single pole was insufficient, so that will be a two-pole filter as well. This means the bandpass filter will use four poles.

A two pole low pass Butterworth filter with cutoff 1 rad/s takes the form:

$$H_0(s) = \frac{1}{(s + e^{j45°})(s + e^{-j45°})} = \frac{1}{s^2 + 1.414s + 1}$$

To make the high pass filter, transform $H_0(s)$ with:

$$H_{hp}(s) = H_0(1/s) = \frac{s^2}{s^2 + 1.414s + 1}$$

The highpass filter cutoff is 1 rad/s in this case, so no shifting is needed.
The low pass filter must be shifted to a cutoff frequency of $\omega_c = 5$ rad/s:

$$H_{lp} = H_0\left(\frac{s}{\omega_c}\right) = \frac{25}{s^2 + 1.414 \cdot 5s + 25}$$

The bandpass filter is created by multiplying the lowpass and the highpass filters. Then, divide by the peak value of the transfer function (0.9568) so that the passband flattens out at 0 dB:

$$H(s) = \frac{25}{s^2 + 1.414 \cdot 5s + 25} \cdot \frac{s^2}{s^2 + 1.414s + 1} \cdot \frac{1}{0.9568} = \frac{26.13s^2}{(s^2 + 1.414s + 1)(s^2 + 7.07s + 25)}$$

The bode plot of this transfer function is shown in Figure 1. The Python code used to create the plot can be found in Appendix A.

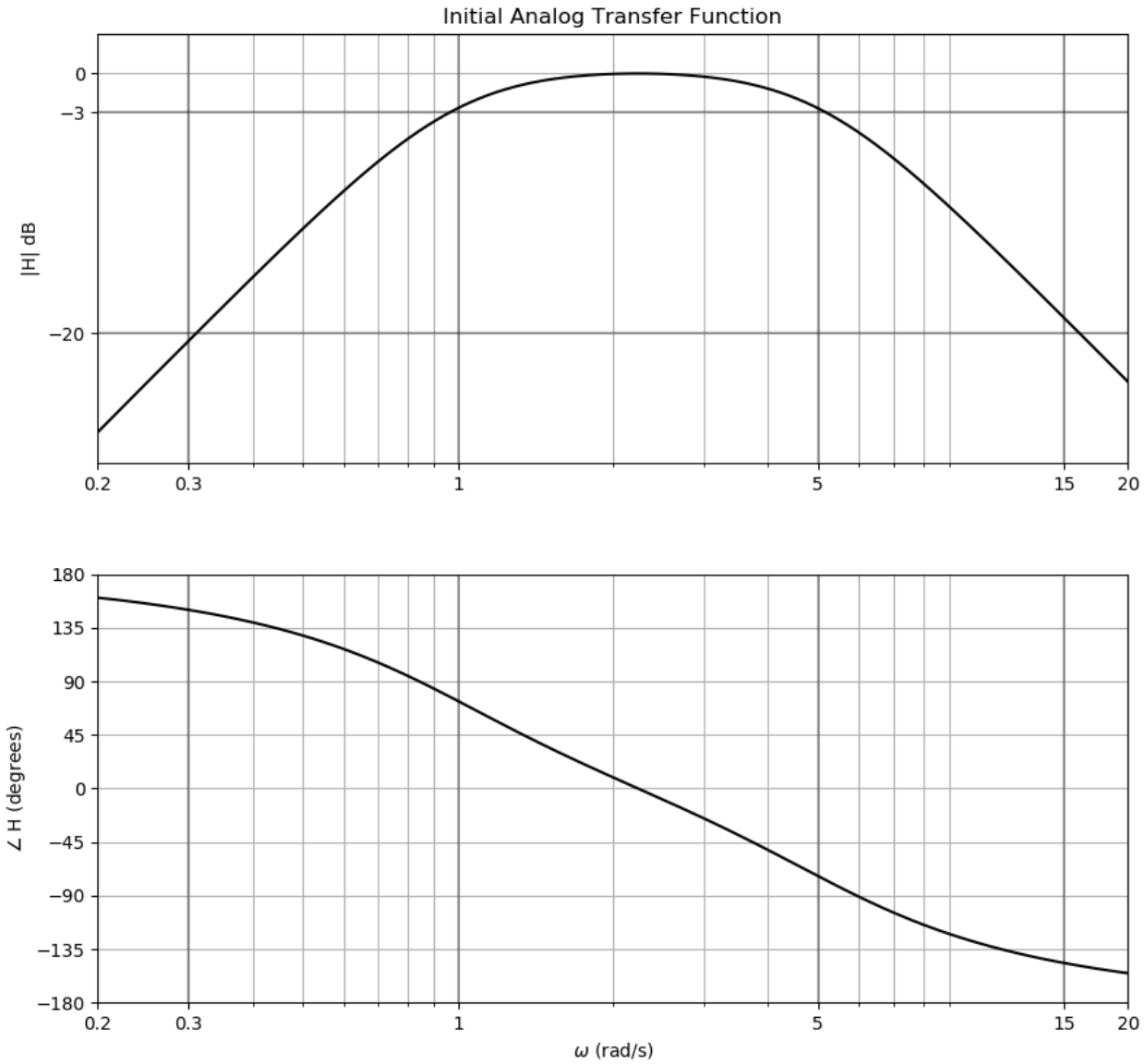

Figure 1: Initial analog filter Bode plot

We see that the highpass section looks good, but the lowpass section should be shifted a bit to the left. To do this, I substituted s with $\frac{s}{0.94}$ in the lowpass section, making the transfer function:

$$H(s) = \frac{23.09s^2}{(s^2 + 1.414s + 1)(s^2 + 6.65s + 22.09)}$$

which is shown in Figure 2 and appears to meet specifications.

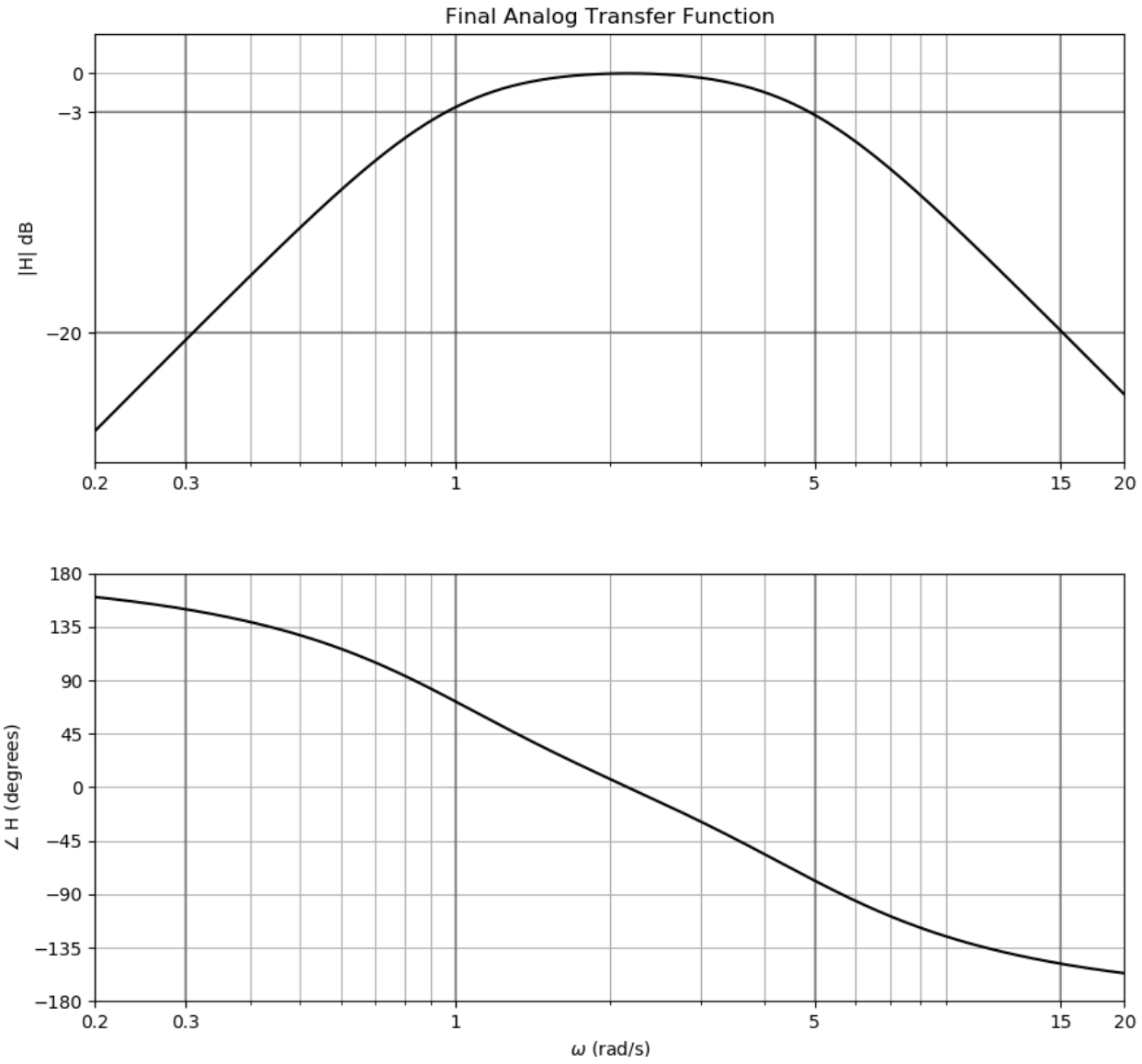

Figure 2: Final analog filter Bode plot (slightly shifted lowpass section)

Convert to Digital Filter

An analog filter can easily be converted to a digital FIR filter with the backward linear approximation: $s \approx \frac{(1-z^{-1})}{T}$. This substitution is easily made in software (see code in Appendix B). The same code also creates a z-domain bode plot, shown in Figure 3. I found that shifting the lowpass filter with a factor 0.96 rather than 0.94 worked better after the approximation. Since the timestep is 0.01 s, the analog frequencies of 0.3, 1, 5 and 15 rad/s correspond to the z-domain angles of $0.171°, 0.572°, 2.86°$ and $8.59°$, respectively.

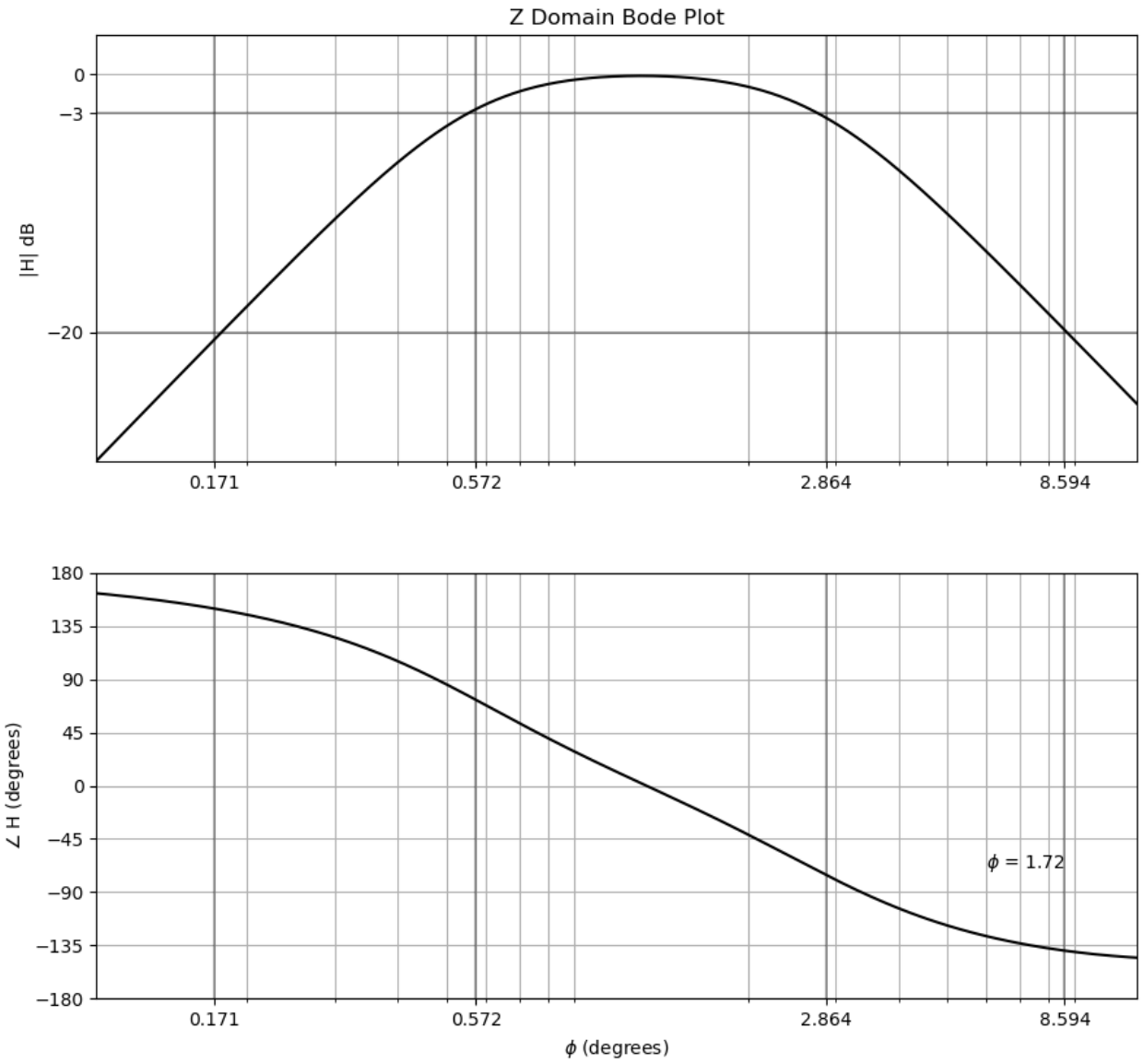

Figure 3: Z domain Bode plot. Note that the x-axis is no longer rad/s, rather it is degrees.

With the slight shift and approximation the digital transfer function is:

$$H(z) = H(s)\big|_{s=\frac{(1-z^{-1})}{0.01}} = \frac{0.002219(1 - 2z^{-1} + z^{-2})}{(1 - 1.98586z^{-1} + 0.98596z^{-2})(1 - 1.93227z^{-1} + 0.934426z^{-2})}$$

Verify Accuracy in Time Domain

This filter can easily be implemented in the time domain using the Python code in Appendix C. The coefficients of the lowpass and highpass sections are put in separately, and then the output of the lowpass section is made the input to the highpass section. The input to the lowpass section is a sinusoid at one of the four corner frequencies, demonstrating that the output is attenuated the specified amount. The output is displayed in Figure 4.

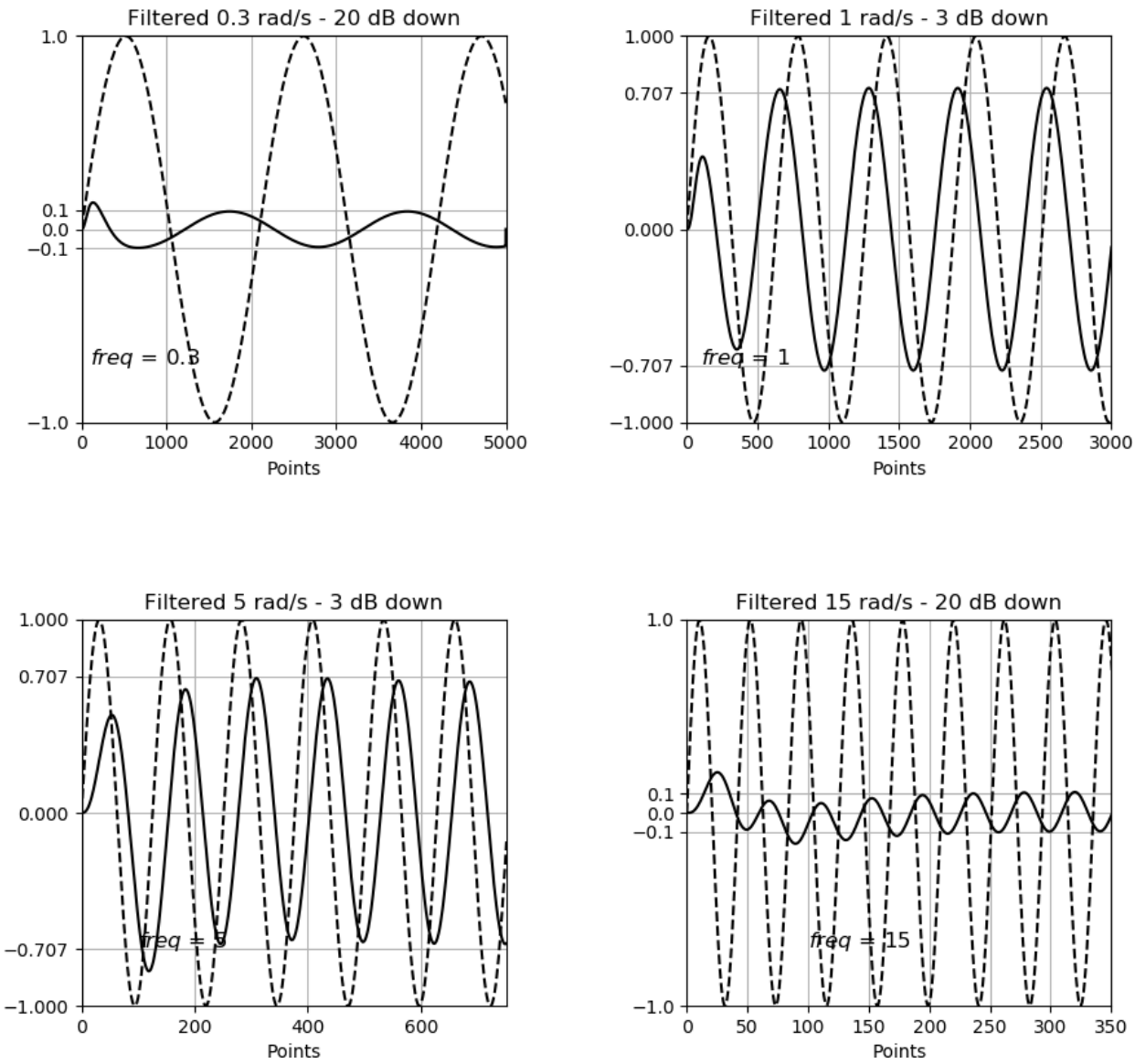

Figure 4: Time domain output at corner frequencies. Dashed lines are input, solid lines are output.

Appendix A - Analog Bode Plot Code (my_Fbode.py)

```
# based on Dr. Sullivan's notes
import numpy as np
import matplotlib.pyplot as plt
from math import pi

NN = 2000
w_min = 0.2
w_max = 20
#W = np.linspace(w_min,w_max,NN)
W = np.logspace(-1,2,NN)

s = 1j*W
shift=0.94
H = 25*s**2 / (((s/shift)**2 + 1.414*5*(s/shift) + 25)*(s**2 + 1.414*s + 1))
Hmax = np.real(max(H))
print(Hmax)
H = H/Hmax

Hmag = np.absolute(H)
plt.subplot(211)
ticks = [0.2, 0.3, 1, 5, 15, 20]
plt.semilogx(W,20*np.log10(Hmag),'k')
plt.axis([w_min, w_max, -30, 3])
plt.ylabel('|H| dB')
plt.yticks([-20, -3, 0])
plt.xticks(ticks,[str(t) for t in ticks])
for tick in ticks[1:-1]:
    plt.axvline(tick,color='k',alpha=0.3)
for tick in [-3,-20]:
    plt.axhline(tick,color='k',alpha=0.3)
plt.title('Final Analog Transfer Function')
plt.grid(which='both')

aaa = np.angle(H)
for n in range(NN):
    if aaa[n] > pi:
        aaa[n] = aaa[n] - 2*pi

plt.subplot(212)
plt.semilogx(W,(180/pi)*aaa,'k')
plt.ylabel(r'$\angle$ H (degrees)')
plt.xlabel('$\omega$ (rad/s)')
plt.xticks(ticks,[str(t) for t in ticks])
plt.yticks(np.arange(-180,181,45))
for tick in ticks[1:-1]:
    plt.axvline(tick,color='k',alpha=0.3)
plt.axis([w_min, w_max, -180, 180])
plt.grid(which='both')
```

```
plt.tight_layout()
plt.show()
```

Appendix B - Z Domain Bode Plot Code (approx_zbode.py)

```
# based on Dr. Sullivan's notes
# combines approx_zbode and my_zbode

import numpy as np
import matplotlib.pyplot as plt
from math import pi

NN = 1000000
phi = np.linspace(0,2*pi,NN)

z = np.exp(1j*phi)
dt = .01

s = (1-z**-1)/dt # backward linear
shift = 0.96
#H = 25*s**2 / (((s/shift)**2 + 1.414*5*(s/shift) + 25)*(s**2 + 1.414*s + 1))/0.9568
H = 0.002219*(1-2*z**-1+z**-2)/((1-1.98586*z**-1+0.98596*z**-2)*(1-1.93227*z**-1+0.934426*z**-2))
#H = 0.002219*z**2*(z-1)**2*z**-4/((z**2-1.98586*z+0.98596)*(z**2-1.93227*z+0.934426)*z**-4)

Hmag = np.absolute(H)
ticks = np.array([0.3, 1, 5, 15])
ticks = ticks*dt/pi*180
plt.subplot(211)
plt.semilogx((180/pi)*phi,20*np.log10(Hmag),'k')
plt.axis([0.1, 12, -30, 3])
plt.ylabel('|H| dB')
plt.yticks([-20, -3, 0])
plt.xticks(ticks,[str(t)[:5] for t in ticks])
for tick in ticks:
    plt.axvline(tick,color='k',alpha=0.3)
for tick in [-3,-20]:
    plt.axhline(tick,color='k',alpha=0.3)
plt.title('Z Domain Bode Plot')
plt.grid(which='both')

aaa = np.angle(H)
for n in range(NN):
    if aaa[n] > pi:
        aaa[n] = aaa[n] - 2*pi

plt.subplot(212)
plt.semilogx((180/pi)*phi,(180/pi)*aaa,'k')
plt.axis([0.1, 12, -180, 180])
plt.ylabel(r'$\angle$ H (degrees)')
plt.yticks(np.arange(-180,181,45))
plt.xticks(ticks,[str(t)[:5] for t in ticks])
for tick in ticks:
    plt.axvline(tick,color='k',alpha=0.3)
```

```
plt.text(6,-70,'$\phi$ = {}'.format(round(1.72,2)))
plt.grid(which='both')
plt.xlabel('$\phi$ (degrees)')

plt.tight_layout()
plt.show()
```

Appendix C - Time Domain Test Code (sim_BP.py)

```
# based on Dr. Sullivan's notes
from IPython import get_ipython
get_ipython().magic('reset -sf') # clear variables

import numpy as np
import matplotlib.pyplot as plt

NN = 5000
n = np.arange(NN)
dt = 0.01

x = np.zeros(NN)
y = np.zeros(NN)
w = np.zeros(NN)
ytime = np.zeros(NN)

# --- input ---
x1 = np.sin(0.3*n*dt)
x2 = np.sin(1*n*dt)
x3 = np.sin(5*n*dt)
x4 = np.sin(15*n*dt)

# --- filter ---
b0 = 0.002219
b1 = 0
b2 = 0
a1 = 1.93227
a2 = -0.934426

d0 = 1
d1 = -2
d2 = 1
c1 = 1.98586
c2 = -0.98596

start_step = 2
nsteps = NN-2-start_step
plt.figure()
T = start_step
for _ in range(nsteps):
    T=T+1
    y[T] = a1*y[T-1] + a2*y[T-2] + b0*x1[T] + b1*x1[T-1] +  b2*x1[T-2]
    w[T] = c1*w[T-1] + c2*w[T-2] + d0*y[T] + d1*y[T-1] +  d2*y[T-2]
    ytime[T] = w[T]

plt.subplot(221)
plt.plot(x1,'k--')
plt.plot(ytime,'k')
```

```
plt.text(100,-.7,'$freq$ = 0.3',fontsize=12)
plt.axis([0,NN,-1,1])
plt.title('Filtered 0.3 rad/s - 20 dB down')
plt.yticks([-1,-0.1,0,0.1,1])
plt.xlabel('Points')
plt.grid()
plt.show()

T = start_step
nsteps=3000
for _ in range(nsteps):
    T=T+1
    y[T] = a1*y[T-1] + a2*y[T-2] + b0*x2[T] + b1*x2[T-1] +  b2*x2[T-2]
    w[T] = c1*w[T-1] + c2*w[T-2] + d0*y[T] + d1*y[T-1] +  d2*y[T-2]
    ytime[T] = w[T]

plt.subplot(222)
plt.plot(x2,'k--')
plt.plot(ytime,'k')
plt.text(100,-.7,'$freq$ = 1',fontsize=12)
plt.axis([0,nsteps,-1,1])
plt.title('Filtered 1 rad/s - 3 dB down')
plt.yticks([-1, -0.707, 0, 0.707, 1])
plt.xlabel('Points')
plt.grid()
plt.show()

T = start_step
nsteps = 750
for _ in range(nsteps):
    T=T+1
    y[T] = a1*y[T-1] + a2*y[T-2] + b0*x3[T] + b1*x3[T-1] +  b2*x3[T-2]
    w[T] = c1*w[T-1] + c2*w[T-2] + d0*y[T] + d1*y[T-1] +  d2*y[T-2]
    ytime[T] = w[T]

plt.subplot(223)
plt.plot(x3,'k--')
plt.plot(ytime,'k')
plt.text(100,-.7,'$freq$ = 5',fontsize=12)
plt.axis([0,nsteps,-1,1])
plt.title('Filtered 5 rad/s - 3 dB down')
plt.yticks([-1, -0.707, 0, 0.707, 1])
plt.xlabel('Points')
plt.grid()
plt.show()

T = start_step
nsteps = 350
for _ in range(nsteps):
    T=T+1
```

```
    y[T] = a1*y[T-1] + a2*y[T-2] + b0*x4[T] + b1*x4[T-1] +  b2*x4[T-2]
    w[T] = c1*w[T-1] + c2*w[T-2] + d0*y[T] + d1*y[T-1] +  d2*y[T-2]
    ytime[T] = w[T]

plt.subplot(224)
plt.plot(x4,'k--')
plt.plot(ytime,'k')
plt.text(100,-.7,'$freq$ = 15',fontsize=12)
plt.axis([0,nsteps,-1,1])
plt.title('Filtered 15 rad/s - 20 dB down')
plt.yticks([-1,-0.1,0,0.1,1])
plt.xlabel('Points')
plt.grid()
plt.show()
plt.tight_layout()
```