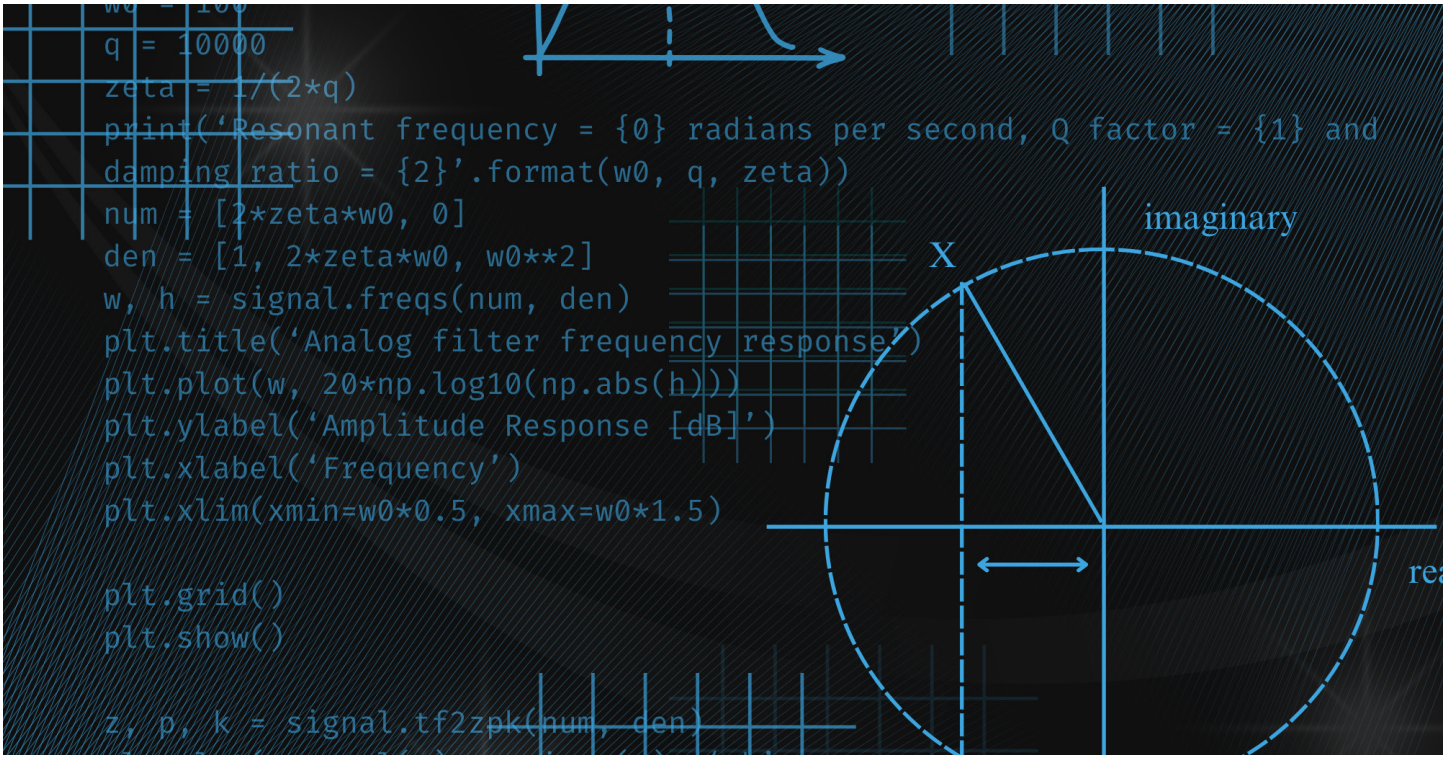
WHITE PAPER

knowles

EXPERIMENTING WITH OPEN-SOURCE RF ANALYSIS TOOLS TO BETTER UNDERSTAND HOW POLES AND ZEROS RELATE TO Q FACTOR

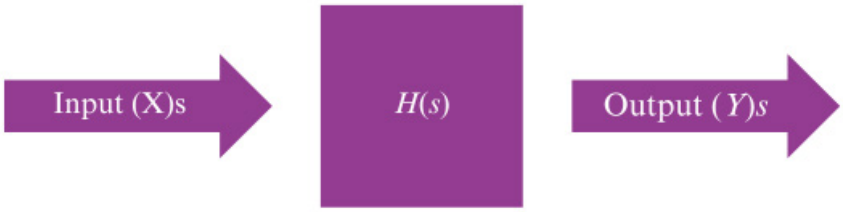


As an RF designer, an in-depth understanding of poles and zeros will improve your filter designs. In this white paper, we will:

- Use the Python Scipy.signal Toolbox to build a transfer function
- Take a closer look at poles and zeros to explore their relationship to Q factor and their function in filter performance

Poles, Zeros and the Transfer Function

To better understand poles and zeros, it's worth revisiting the transfer function. Filters have a transfer function, $H(s)$, that tells us what output signal to expect based on a given input signal. Filter transfer functions are traditionally expressed in terms of the complex variable, s.



By converting the output signal, $Y(s)$, back into real numbers, we can see how a filter's performance is determined by the structure of the transfer function. This can be observed by identifying values for *s* when the denominator of the transfer function, $D(s)$, approaches zero or the numerator, $N(s)$, approaches zero.

$$H(s) = \frac{N(s)}{D(s)}$$

When $N(s)$ approaches zero and the transfer function gets smaller, those values of s are zeros. When $D(s)$ approaches zero and the transfer function gets larger, those values of s are the poles. In other words, solutions that zero the function are the *zeros*, and roots that make the function approach its maximum limit are the *poles*. For additional detail, here's an example of this premise using a simple RC first-order lowpass filter.

Solutions that zero the function are the *zeros*, and roots that make the function approach its maximum limit are the *poles*.

Creating a Transfer Function for a Second-Order Bandpass Filter Using Python

Using Python, we can model a transfer function as an array of coefficients for the numerator and another array of coefficients for the denominator:

$$H(s) = \frac{\sum_{i=0}^{N} b[N-i]s^i}{\sum_{j=0}^{M} a[M-i]\,s^j}$$

The toolbox allows us to pass in an array of b elements for the numerator and an array of a elements for the denominator to create a transfer function.

An example transfer function for a second-order bandpass filter might be:

$$H_{BP}(s) = \frac{2\zeta\omega_0 s}{s^2 + 2\zeta\omega_0 s + \omega_0^2}$$

Here, ω_0 is the natural frequency and the resonator damping ratio, ζ , is related to Q factor as:

$$\zeta = \frac{1}{2Q}$$

Following the signal toolbox documentation, we would pass in our numerator and denominator coefficients in decreasing order of *s*, so our *b* array would be written as the numeric values for $[2\zeta\omega_0, 0]$ and the a array would be $[1, 2\zeta\omega_0, \omega_0^2]$.

Plotting the Frequency Response Using Input Resonant Frequency and Q Factor

Using numpy, the scipy.signal and matplotlib.pyplot, we can use Python to look at the frequency response for this transfer function.

Start by setting the resonant frequency as 100 radians/second and Q factor as 100.

From there, the damping ratio and transfer function are calculated and the frequency response is plotted (Figure 1):

```
import numpy as np
import scipy.signal as signal
import matplotlib.pyplot as plt
w0 = 100
q = 100
zeta = 1/(2*q)
print('Resonant frequency = {0} radians per second, Q factor = {1} and damping ratio = {2}'
format(w0, q, zeta))
num = [2*zeta*w0, 0]
den = [1, 2*zeta*w0, w0**2]
w, h = signal.freqs(num, den)
plt.title('Analog filter frequency response')
plt.plot(w, 20*np.log10(np.abs(h)))
plt.ylabel('Amplitude Response [dB]')
plt.xlabel('Frequency')
plt.xlim(xmin=w0*0.5, xmax=w0*1.5)
plt.grid()
plt.show()
> Resonant frequency = 100 radians per second, Q factor = 100 and damping ratio = 0.005
```

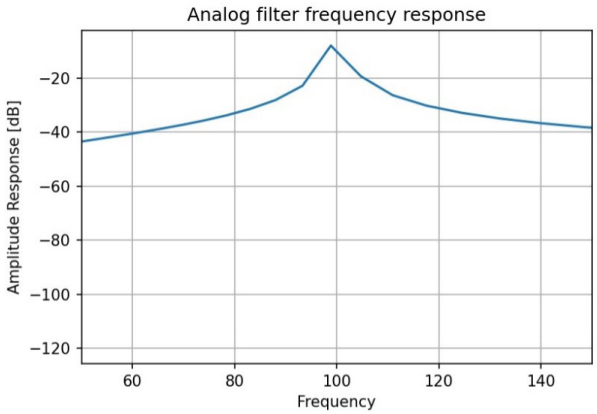


Figure 1: Frequency response for our sample transfer function when the resonant frequency is set to 100 radians/second and Q factor is set to 100

Plotting Poles and Zeros

We can also plot (Figure 2) the poles and zeros for this transfer function in the s plane:

```
print('Resonant frequency = {0} radians per second, Q factor = {1} and damping ratio = {2}'.format(w0, q, zeta))
z, p, k = signal.tf2zpk(num, den)
plt.plot(np.real(z), np.imag(z), 'ob', markerfacecolor='none')
plt.plot(np.real(p), np.imag(p), 'xr')
plt.legend(['Zeros', 'Poles'], loc=1)
plt.title('Pole / Zero Plot')
plt.xlabel('Real')
plt.ylabel('Imaginary')
plt.grid()
plt.show()
```

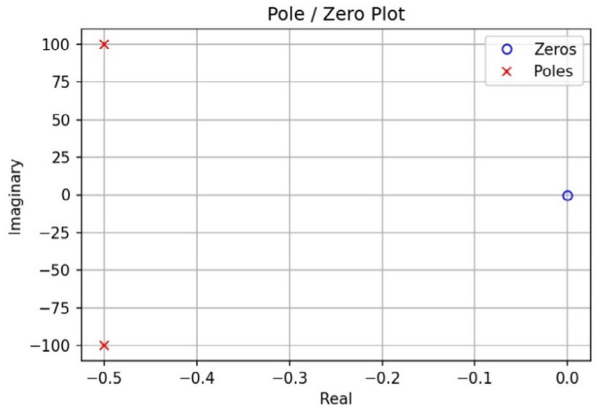


Figure 2: Poles and zeros plotted in the s plane for our sample transfer function when the resonant frequency is set to 100 radians/second and Q factor is set to 100.

Adjusting Input Parameters and Inspecting the Response

Already mentioned in this white paper, quality factor, or Q factor, is a common shorthand figure of merit (FOM) for RF filters. In its simplest form, Q factor is expressed as the ratio of stored energy to lost energy per oscillation cycle. Q factor comes up a lot in RF engineering, but determining this value has its intricacies because it's impacted by a confluence of factors.

In the complex plane, one way to approach Q factor is to look at the poles and zeros, or pole Q factor. For example, to go from minimum transmission to maximum transmission as sharply as possible, you need to have a zero pulling the signal down and a pole pushing the signal up. In theory, by moving the poles and zeros on the frequency axis, the slope gets steeper or shallower, impacting the sharpness of the filter. In practice, you can place zeros near frequencies you'd like to reject and poles near frequencies you'd like to pass. Ultimately, moving poles and zeros around on the plot crystallizes the filter response into the right shape for the customer.

We gain information about the Q factor when we know where the poles and zeros are. The magnitude of the vector formed by coordinates in the s plane (Figure 3) tell us a lot about the resonant frequency of a pole.

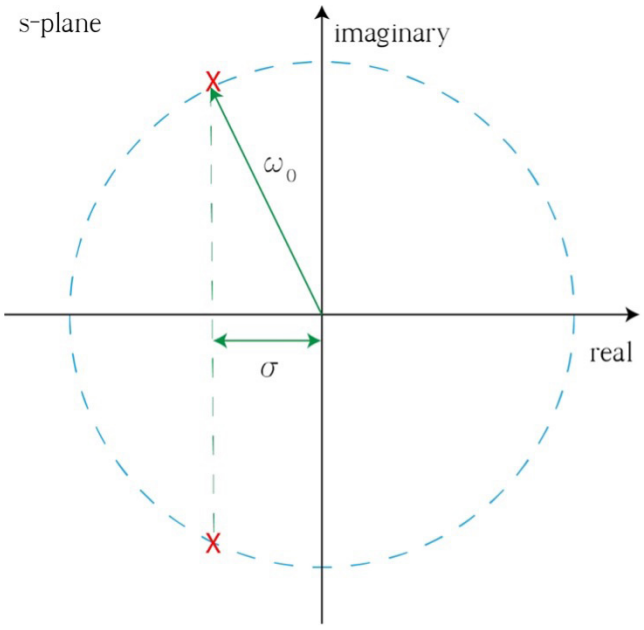


Figure 3: Resonant frequency of a pole can be found by looking at the magnitude of the vector formed by coordinates in the s plane

Using the Python script, we can observe how sigma behaves on the pole/zero plot by keeping frequency consistent and increasing Q factor.

We gain information about the Q factor when we know where the poles and zeros are. The magnitude of the vector formed by coordinates in the s plane (Figure 3) tell us a lot about the resonant frequency of a pole.

To start, find the pole resonant frequency of the first pole:

```
print('Upper left pole: {:.1}'.format(p[0]))
> Upper left pole: (-0.5+1e+02j)
print('Pole resonant frequency: {:.1f}'.format(np.abs(p[0])))
> Pole resonant frequency: 100.0
```

This matches our input ω_0 of 100 radians/second.

We can check the distance, σ_x, by using:

```
print('Real value: {:.1f}'.format(np.real(p[0])))
> Real value: -0.5
```

Next, find the pole quality factor. This is given as:

$$\frac{\omega_0}{2\sigma_x}$$

```
print('Pole quality factor:
{:.1f}'.format(np.abs(np.abs(p[0])/(2*np.real(p[0])))))

> Pole quality factor: 100.0
```

Now, we can experiment. Let's try increasing the Q factor in our transfer function and see how that impacts the pole Q factor (Figure 4, 5):

```
w0 = 100
q = 10000
zeta = 1/(2*q)
print('Resonant frequency = {0} radians per second, Q factor = {1} and damping ratio = {2}'.format(w0, q, zeta))
num = [2*zeta*w0, 0]
den = [1, 2*zeta*w0, w0**2]
w, h = signal.freqs(num, den)
plt.title('Analog filter frequency response')
plt.plot(w, 20*np.log10(np.abs(h)))
plt.ylabel('Amplitude Response [dB]')
plt.xlabel('Frequency')
plt.xlim(xmin=w0*0.5, xmax=w0*1.5)

plt.grid()
plt.show()

z, p, k = signal.tf2zpk(num, den)
plt.plot(np.real(z), np.imag(z), 'ob', markerfacecolor='none')
plt.plot(np.real(p), np.imag(p), 'xr')
plt.legend(['Zeros', 'Poles'], loc=1)
plt.title('Pole / Zero Plot')
plt.xlabel('Real')
plt.ylabel('Imaginary')

plt.grid()
plt.show()

print('Upper left pole: {:.1}'.format(p[0]))
print('Pole resonant frequency: {:.1f}'.format(np.abs(p[0])))
print('Real value: {}'.format(np.real(p[0])))
print('Pole quality factor: {:.1f}'.format(np.abs(np.abs(p[0])/(2*np.real(p[0])))))

> Resonant frequency = 100 radians per second, Q factor = 10000 and damping ratio = 5e-05
```

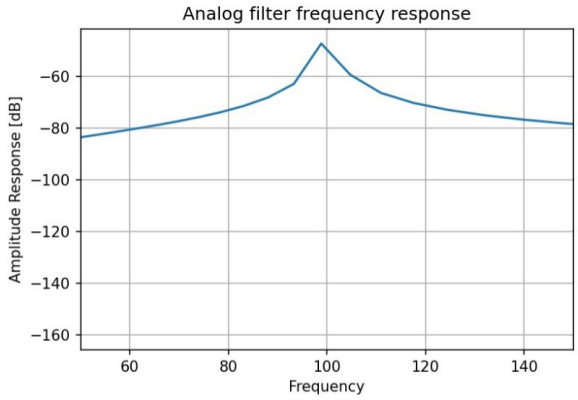


Figure 4: Frequency response for our sample transfer function when the resonant frequency is set to 100 radians/second and Q factor is set to 10000

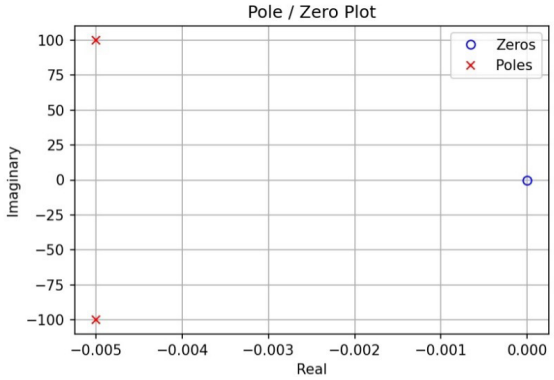


Figure 5: Poles and zeros plotted in the s plane for our sample transfer function when the resonant frequency is set to 100 radians/second and Q factor is set to 10000

```
> Upper left pole: (-0.005+1e+02j)
> Pole resonant frequency: 100.0
> Real value: -0.005000000000000001
> Pole quality factor: 10000.0
```

This experiment demonstrates that as a pole moves closer to the imaginary axis, the Q factor of that pole will increase.

Using Poles and Zeros to Enhance Filter Designs

Here, we've used Python's free, open-source libraries to explore poles and zeros. This exercise emphasizes the following observations:

- A filter's transfer function will increase relative to the distance between your frequency of interest and a pole
- A filter's transfer function decreases the further you are from a pole
- Zeros have the opposite effect of poles

In this paper, we explained some of the theory behind poles and zeros and their relationship with Q factor. In practice, creating poles and zeros involves building resonators. For more information on how this is done, tune in to our on-demand webinar. The webinar covers how resonators work, what types of resonators to use and when and what happens when you combine different resonator types.