MUSI 2525: Assignment 4

Logan Bussell and Matthew Arnold

2018/11/12

Filtering Noise

Using the following code, we can generate noise that we can then use to test our custom single-pole filter implementation.

```
x = randn(44100);
```

We can test our single-pole filter implementation by plotting it against MATLAB's own implementation. Using $\alpha = 0.99$, the output can be seen in Figure 1.

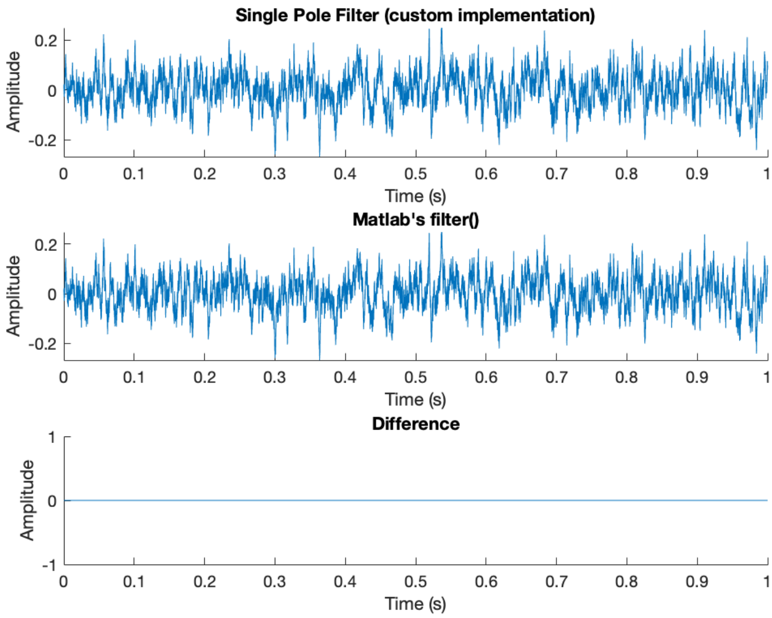

Figure 1: Noise filtered with a low pass filter

Notice that the difference between the filters is exactly zero. We can confirm this by making sure the following command returns a logical 1:

```
>> all(xFiltered == xMatlabFiltered)
ans =
  logical
   1
```

Filtering a Signal

Filtering a given audio signal `cathy_2.wav` using values $\alpha = 0.5, 0.9, 0.99, 0.999$, gives the plots shown in Figure 2.

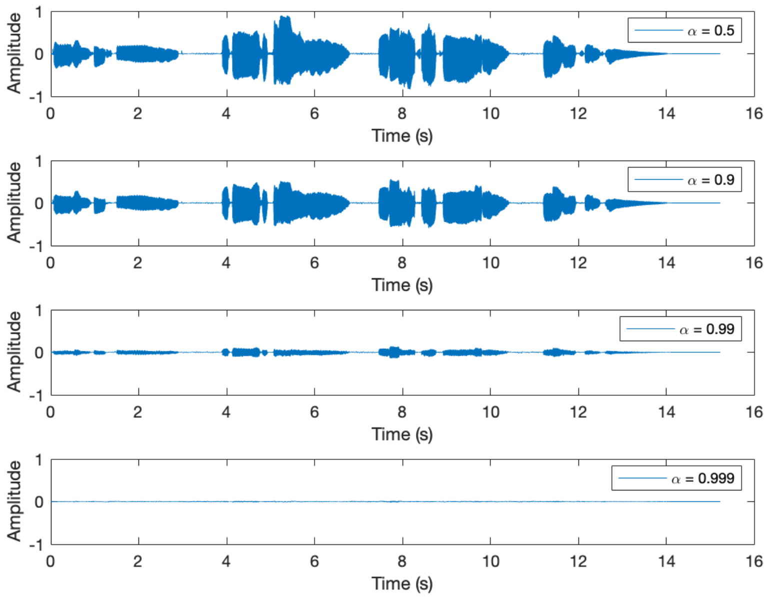

Figure 2: One audio signal filtered with varying values of α

After testing the difference of auditory perception using the `sound()`, we can determine that as α approaches 1, the low pass effect becomes more evident. The low pass effect is difficult to distinguish using only the plots, so the auditory test is helpful to verify proper filter characteristics.

The overall amplitude of the samples also decreases as α approaches 1. This is clearly displayed by the plots and by auditory perception using the `sound()` function.

Determining Filter Cutoff

Using the MATLAB function `freqz()`, we can plot the magnitude response of our single pole filter using different values of α.

We used linear interpolation to estimate and plot the cutoff frequency of two filters with $\alpha = 0.9, 0.99$, which are marked with an x in Figure 3.

The plot with $\alpha = 0.9$ has a cutoff frequency of approximately 738 Hz, and the plot with $\alpha = 0.99$ has a cutoff frequency of approximately 70 Hz.

In this scenario, α controls the placement of the cutoff frequency and the slope of the gradual decline in the low pass transfer function (relative to logarithmic frequency axis for perceptual purposes). For example, plot 2 ($\alpha = 0.99$) has a much lower frequency cutoff than plot 1, but a much more gradual decline throughout the frequency range above the cutoff frequency.

The closer α gets to 1, the more the bass frequencies will stand out due to the lowered cutoff frequency and "leftward" movement of the low pass filter. A lower α results in a higher cutoff frequency and a steeper cutoff slope, and vice versa.

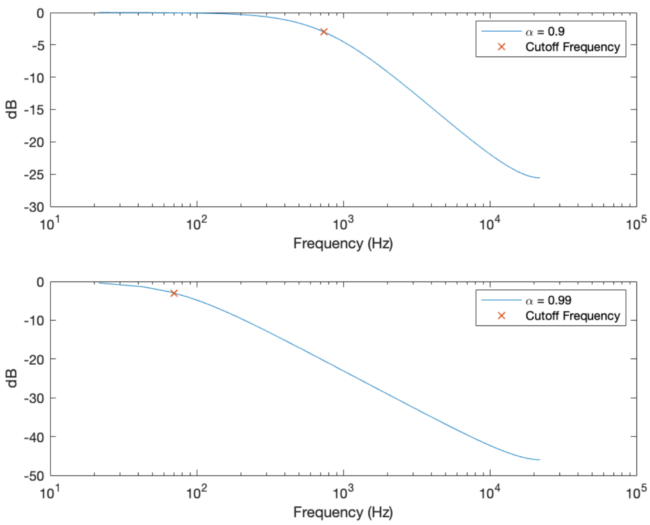

Figure 3: The cutoff frequencies of different low-pass filters

Using $\alpha > 1$ has an unintended side effect: it makes the signal 0 for the majority of the length, but with a sharp peak at the very end (positive or negative, depending on the α value).

Peak Filtering

Using a given set of arguments (frequency, Q, and gain), we can calculate the coefficients to pass into MATLAB's `filter()` function that will result in a peak filtered signal, and wrap this in our own function.

Using the function `fvtool()` we can analyze the magnitude response of a filter using our generated coefficients to verify that our filter behaves as expected:

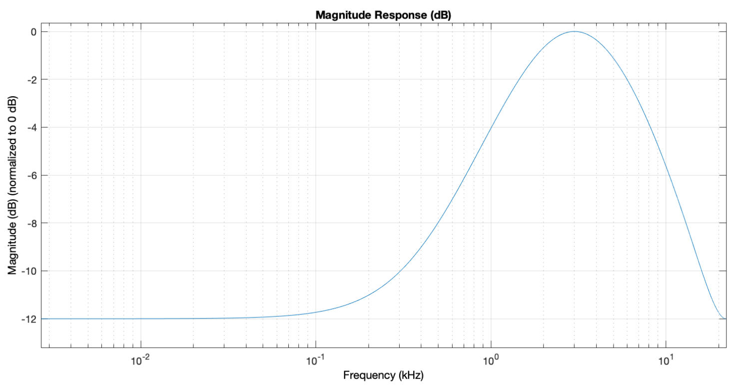

Figure 4: Magnitude response of a peak filter with frequency = 3000 Hz, Q = 4, and gain = +12

As expected, adjusting the frequency parameter moves the peak, adjusting Q tightens/spreads out the peak, and adjusting gain changes the intensity of the peak.

The results of filtering audio with the above filter can be seen in Figure 5.

Since we are applying a peak filter, it makes sense that the output would be louder at certain frequencies, causing the overall amplitude in some places in the signal to be greater.

Aurally, the effect of the filter is obvious. The signal sounds louder and more "saturated" due to the increase in level in the 3000 Hz range.

Note: an attempt at a custom peak filter implementation was included with this homework submission, but as of this writing it doesn't perform as expected.

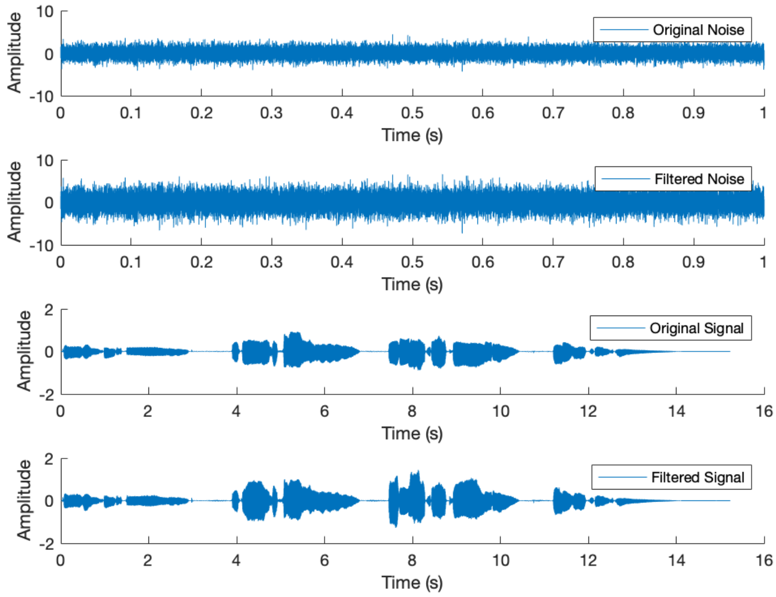

Figure 5: Audio peak filtered with frequency = 3000 Hz, Q = 4, gain = +12