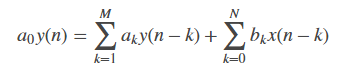
Part A: Simple Signal Processing

* One of the simplest things we can do to a signal is to *scale* it (i.e. multiply it by a constant). Try:   
  >> out1 = .25 \* sig1;  
  >> sound(out1, Fs)
* Or if you're lazy:
* >> sound(.25 \* sig1)
* We can also make the signal bigger:
* >> sound(4 \* sig1)
* About the simplest thing we can do to *two* signals is to add them together:  
  >> out2 = sig1 + sig2;
* Listen to out2 and look at its spectrogram. Can you identify the two different signals in both the audible and visual domains?
* Both the A/D converter which digitized the input signal and the D/A converter which converts the processed samples back to an analog signal have limited range of values that they can represent. By keeping our input signal below 5 V we keep it within the range of the A/D converter and avoid "clipping" the signal. However, when we add two signals together or send then through systems having a gain greater than one, we may create an output signal with values too large for the D/A converter to convert. If this happens you will hear distortion in the output signal. To avoid this, you can use the soundsc() command instead of sound(). This command automatically scales the signal value so that it fits within the range of the D/A converter.
* Another interesting thing we can do is *multiply* the two signals together. Digitally, this is just as easy to do as adding them, but this is very difficult to do in an analog circuit.  
  >> out3 = sig1 .\* sig2;
  + Note the special Matlab construct “.\*” denoting point by point multiplication, as opposed to matrix multiplication.
* Listen to out3 and look at its spectrogram. Can you identify either of the two signals in either the aural or visual domains?

Part B: Simple Filtering

As you saw in ELEC241, filter a digital signal involves forming a weighted sum of the past input and output samples:  


Matlab has a function for performing this operation:  
>> y = filter(b, a, x);  
Choosing the values of the *ak’*s and *bk*’s is the art of *filter design,* which we'll look at in the next part. For now, we'll just try a few simple values for *a* and *b*.

As a tip, record the characteristics of each filter we look at - you will be asked to summarize each type at the end of experiment 9.4.

* A *finite impulse response* (FIR) or non-recursive filter has all the *ak* (except *a*0) equal to zero, i.e. it is a weighted sum of input samples. The simplest FIR filter is the *boxcar* filter in which all the *bk* are equal to 1. Define a length 5 boxcar filter:  
  >> a=1;  
  >>b=[1 1 1 1 1];
* Matlab has a function to compute and display the frequency response of a filter:  
  >> freqz(b, a)  
  Note that the frequency axis is labeled in terms of the Nyquist frequency, which is ½ the sampling rate. So for our sampling rate, 1 corresponds to 5 kHz.  
  Although it is a little distorted because of the logarithmic scaling, this is a close relative of our old friend sin(x)/x. Also note that the jumps in phase are caused by a change in sign of the transfer function (the magnitude is always positive). Taking this into account, the phase is a *linear* function of frequency.
* We can also look at the *unit-sample response* of the filter:  
  >> delta = [1 zeros(1,100)];  
  >> y = filter(b, a, delta);  
  >> plot(y, '.')  
  As you would expect, the unit-sample response is a rectangular pulse, which supposedly looks like the side view of a boxcar.
* Now let's try applying this filter to our signal:  
  >> out1 = filter(b, a, sig1);
* Listen to out1 and look at its spectrogram.  
  >> sound(out1, Fs);  
  >> specgram(out1, 156, Fs)  
  As you can tell from its frequency response plot,  this is a "sort-of" lowpass filter, though not a very good one. **Can you tell this by comparing the sound and spectrogram of the output with the input? What do the blue areas of the spectrogram correspond to?**
* Repeat from the beginning of Part B using a length 10 boxcar filter.
* **What is the relation between the length of the filter and its frequency response?**
* An *infinite impulse response* (IIR) filter has an unit-sample response that doesn't go to zero after a finite number of samples. IIR filters are usually implemented *recursively*, i.e. with non-zero *a* coefficients
* Define a first order recursive filter:

>> a=[1, -.9];  
>> b=[1];

* Using the same technique as above, look at its unit-sample and frequency response. Note that this is also a lowpass filter, with a more conventional frequency response than the boxcar filter.
* Apply this filter to the input signal. Listen to the output and plot its spectrogram. **Can you tell by listening and looking that this is a lowpass filter?**
* An interesting thing happens if we change the sign of a1:  
  >> a = [1 .9];  
  Look at the unit-sample and frequency response of this filter. Apply it to the input signal, listen to the output, and plot its spectrogram. **What kind of filter is this?**

Part C: Fancy Filtering

To get filters that are really effective (i.e. that pass the desired frequencies with minimum distortion and reject undesired frequencies as strongly as possible), we need to have a design procedure to find the optimum set of coefficients for a given specification. If you take ELEC 431, you will learn how to do this. In the meantime, we will utilize pre-constructed algorithms.

* The *Butterworth* filter is one of the classic filter designs. It is characterized by having the "flattest" possible passband. Unfortunately, its performance in the stop band is decidedly mediocre. Design a 5th-order lowpass Butterworth filter having a cutoff frequency of 600 Hz with our 10 kHz sampling frequency:  
  >> [b,a] = butter(5, 600/5000);
* Print out the coefficients:  
  >>a,b
* Look at the unit-sample response and the frequency response. Notice how much closer this is to an ideal low pass filter. Also notice the scale on the y-axis. For a better comparison with previous plots, try changing this scale:  
  >> subplot(2,1,1)  
  >> axis([0 1 -60 0])

Also note that the phase is not linear.

* Now apply the filter to our input signal  
  >> out=filter(b,a,sig1);  
  and listen to the output.
* We can also design FIR filters that are significantly better than our boxcar filter:  
  >> a=1  
  >> b=fir1(5,600/5000);  
  Examine the unit-sample and frequency response of this filter and apply it to the reference signal.
* FIR filters really start to get interesting when we build very long ones. Try the same cutoff frequency with 50 coefficients:  
  >> b=fir1(50,600/5000);  
  Note that we now have a filter of order 50. How would you like to design and build an analog (RLC) filter of this order?!
* **Look at the unit-sample and frequency response of this filter. What function does the unit-sample response remind you of?** Note that this filter has a much more ideal shape than any of the others.
* Finally apply this filter to the reference signal. **Can you tell the difference in the sound or the spectrogram between the effects of this filter and the length 5 filter?**
* You will need your signals sig1 and sig2 for the next section.