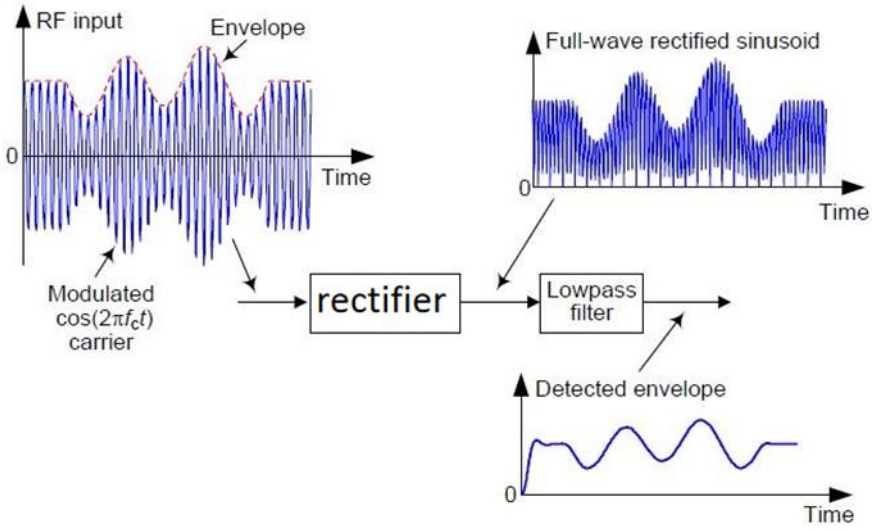
Lab #2: Nonlinear Systems, Filters, and Convolution

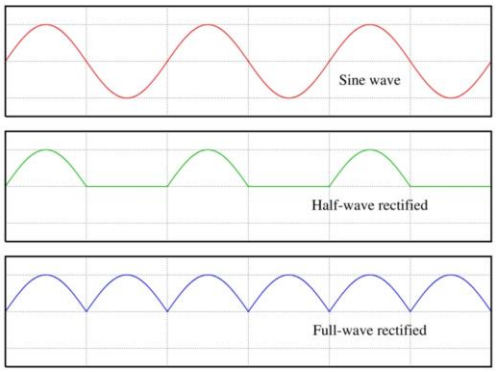
Part #1 Nonlinear rectifier and envelope extraction

An envelope detector takes a (relatively) high-frequency amplitude modulated signal as input and provides an output which is the envelope of the original signal. An envelope detector is sometimes called a peak detector. The signal's envelope is equivalent to its outline, and an envelope detector connects all the peaks in the signal. Envelope detection has numerous applications in the fields of signal processing and communications, one of which is amplitude modulation (AM) detection. In this lab, you will also learn how to design simple filters in Matlab.



An envelope detector consists of **two main elements**:

1) ***Rectifier*** - serves to enhance one half of the received signal over the other. Rectification is also used to convert AC (Alternating Current) to DC (Direct Current). Rectification methods include **full-wave** rectification and **half-wave** rectification. Full-wave rectification rectifies the negative component of the input voltage to a positive voltage or it takes the absolute value (**abs** in Matlab) of the input. In contrast, half-wave rectification removes just the negative components of the input. Rectifiers are nonlinear systems.



Exercise #1:

The following codes perform the function of loading a sound signal and then passing it through a full-wave rectifier. You will hear the distortions creating by the nonlinear rectifier.

```
[signal, Fs]=audioread('sent001.wav'); sound(signal, Fs);pause(3);
Rectified_signal=abs(signal); sound(Rectified_signal, Fs);
```

Use subplots to show a small section of both signals to see what a full-wave rectifier does. You can add a new figure using the **figure()** command. In addition, please modify the codes above to implement a **half-wave rectifier** and then compare the input and output signals using subplots.

2) ***Low pass filter*** – a low-pass filter is required to remove the **high-frequency elements** that remain within the signal after rectification. The low-pass filter can be implemented in MATLAB. The filters are linear.

You can use the MATLAB command "**butter**" to design a Butterworth **low-pass/high-pass** filter, with the following format:

[b, a] = butter(order, w_n); % b and a contain the filter's coefficients

The input arguments to **butter** can be explained here:

order: order of the filter (normally 1 to 4). A larger order leads to a better filtering effect.

w_n: normalized **cutoff frequency** within the range of 0 and 1. When the sampling frequency is Fs, the maximum allowable frequency in frequency-domain processing is Fs/2 (**Nyquist frequency**). Therefore, the normalized cutoff frequency w_n is equal to the real cutoff frequency divided by Fs/2.

For example:

```
Fs=8000;
[bb, aa] = butter(2, 1000/(Fs/2));
%  2nd order low-pass filter with a cutoff frequency of 1000 Hz for signals sampled at Fs.
```

Note: If you receive an error message for the 'Butter' command, you need to install the "**signal processing toolbox**". Click the link and follow the instructions to install it.

The magnitude and phase responses of the filter can be shown by:

```
freqz(bb,aa,128, Fs);
```

To apply the low-pass filter to a signal (or vector) **ss**, you can use the **filter** command:

```
[ss, Fs]=audioread('sent001.wav');
Filtered_signal=filter(bb, aa, ss);
```

Exercise #2 investigating the effect of low-pass filtering

Create a function **low_pass_filter** (no arguments) and copy the following code to the function file. Then, call the function in the Matlab command window. Describe what kind of sound you hear for the low-pass filtered speech. Use subplots to show how different they are.

```
[signal, Fs]=audioread('sent001.wav');
sound(signal,Fs);pause(3);
[bb, aa] = butter(2, 500/(Fs/2)); % 2nd order low-pass filter at 500 Hz
filtered_signal=filter(bb, aa, signal);
sound(filtered_signal, Fs);
```

Exercise #3: investigating the effect of high-pass filtering

Now, change the low-pass filter to a high-pass filter at 4000 Hz as below and repeat Exercise #7. Show the magnitude and phase responses of your high-pass filter.

```
[bb, aa] = butter(2, 4000/(Fs/2),'high'); % 2nd order high-pass filter at 4000 Hz
```

Exercise #4:

Finally, you should be able to write your own Matlab function called **rectifier_envelope**, which can load a .wav file and perform envelope extraction. It should look like the following:

```
function rectifier_envelope
```

% load sent001.wav

%…add codes here

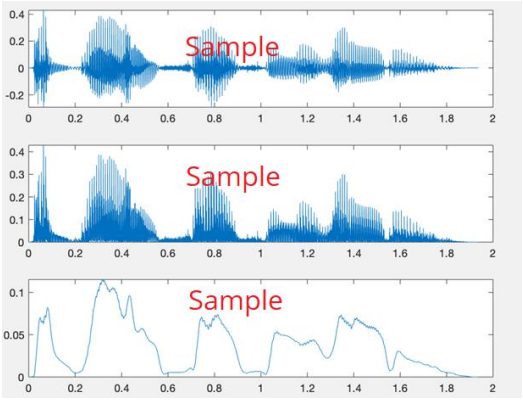
% rectify the speech signal (either full-wave or half-wave)

%…add codes here

% pass it through a low-pass filter at 20 Hz

%… add codes here

Your code should be able to plot the original signal, the rectified signal, and the extracted envelope on the same window using subplots. You can change the cutoff frequency to 5, 50, 500 Hz to see how it would affect the envelope extracted. Include these plots in your lab reports.



Part #2 Convolution

Introduction

In this part of the lab, we will explore convolution and how it can be used with signals such as audio.

Since we are working on a computer, we are working with finite-length, discrete-time versions of signals. It is important to note that convolution in continuous-time systems cannot be exactly replicated in a discrete-time system, but using MATLAB's `conv` function for convolution, we can explore the basic effects and gain insight into what is going on.

When you are explicitly working with discrete-time signals, you would plot them with `stem`. However, since we want to think of these as continuous-time signals, we will still use the `plot`

command. An artifact that you may notice is that discontinuities (as in a step function) are not instantaneous -- they have a small slope in the plot. Also, you need to represent impulses with the height in discrete-time equal to the area in continuous time.

When you want to play or plot the discrete-time signal, you need to specify the time increment Ts between samples. As you found in the previous lab, when playing a sound, you specify Fs=1/Ts. (Fs is set for you when you load a sound using audioread.) When plotting, you need to define a time vector, e.g.

```
>>L=length(fall);Fs=8000;Ts=1/Fs;
>> t=[0:Ts:(L-1)*Ts];
```

Convolution

MATLAB has a function called `conv(x,h)` that you can use to convolve two discrete-time functions `x(n)` and `h(n)`. It assumes that the time steps are the same in both cases. The input signals must be finite length, and the result of the convolution has a length that is the sum of the lengths of the two signals you are convolving (actually L1+L2-1).

1. Recall that a linear time-invariant system is completely described by its impulse function. In MATLAB, the impulse response must be discrete. For example, consider the system with impulse response:

   ```
   h = [1 zeros(1,20) .5 zeros(1,10)];
   ```

   Plot the impulse response using the `plot` command.

2. Consider an input to the system,

   ```
   x = [0 1:10 ones(1,5)*5 zeros(1,40)];
   ```

   Plot the input with the `plot` command.

3. Use the command `conv` to convolve `x` and `h` like this,

   ```
   y = conv(x, h);
   ```

   Use **subplot** to show the impulse response, input, and output of the convolution. Note that you need to add zeros to the end of x and h (to make them the same length as y or define a time vector for each signal in order to make the timing comparable in the different subplots.

4. Every non-zero coefficient of the impulse response `h`, acts as an echo. When you convolve the input `x` and impulse response `h`, you add up all the time-shifted and scaled echoes. Try making the second coefficient negative. How does this change the final result?

Convolution and Smoothing

Exercise #5: Build a box impulse response:

```
h2=[ones(1,50)/20 zeros(1,20)];
```

Create a new signal **y2** by convolving **"fall"** with **h2** (Note: `load('fall.mat'); % if necessary`). Please plot both signals (subplots) and incorporate them into your report. Listen to the fall signal before and after convolution and report the differences you can hear.

Exercise #6: Next, to see what this system does to the input signal, zoom in on part of the signal:

```
subplot(2,1,1), plot(6400:6500, fall(6400:6500));
subplot(2,1,2), plot(6400:6500, y2(6400:6500));
```

The convolved signal should look a little smoother to you. This is because this impulse response applies a low-pass filter to the signal. We'll learn more about filters a bit later in his course, but basically, the idea is that the original signal is made up of sounds at many different frequencies, and the lower frequencies pass through the system, but the higher frequencies are attenuated. This affects how it sounds as well as how it looks.

Box Function

Exercise #7: Download the function called boxt.m from the course webpage. The function should take three parameters, a time vector t that specifies the finite range of the signal and the start time t1 and end time t2 in seconds for the box function. It outputs a vector of the same size as t, which contains the values of the box function.

Try 'help boxt' in your Matlab command window to see how to use it. Create and plot a box function with time span of (-10,10) (e.g. t=-10:0.1:10;). Remember you need to use the correct time vector when plotting a signal.

```
t=-10:0.1:10;
boxsig = boxt(t,-3,3);
plot(t,boxsig);
```

Exercise #8: Create a script file called boxtscript.m that uses the function to create a box that starts at time t = -1 and ends at time t = 1, where the whole signal lasts from time t = -3 to t = 3 (e.g. t=-3:0.1:3). Generate three different versions of this box using three different time granularities, where the finest granularity (set step size to 0.01, 0.1, and 0.5) has very sharp edges similar to the ideal box and the coarsest granularity has a step size of 0.5.

Note:

The different versions should all have the same time span; the difference in the plots should only be at the edges of the box because of artifacts in continuous plotting of a discrete-time signal.

Exercise #9: Plot all three versions in one figure using subplots and include it in your report.

Time: If u is a vector of length n with time span tu = t1:del:t2, and v is a vector of length m with time span tv = t3:del:t4, and both have the same time step del, then the result of conv(u,v) will be a vector of length n + m - 1 with a time span tc = (t1+t3):del:(t2+t4).

Exercise #10: Using the box function that you wrote previously with a sufficiently fine-grained step size (for example, del = 0.01), create box signals from (0,4) and (-1,1), with time span of (-5,10). Find and plot the result of the convolution of the two boxes. Use the above discussion of time to create the appropriate time vector in your plot. Verify that the timing of signal rising and falling matches what you expect in theory.