From EQ to Reverb & Distortion: DSP Audio Effects in Matlab

By Henry D. Pfister

1 Overview

These notes are designed for a two-hour summer enrichment workshop for high-school students. The activity introduces the students to digital signal processing (DSP) for audio signals. Students will record and play sound on a computer using the Matlab scripting language and then apply audio effects (e.g., equalization, reverberation, and distortion) to that sound.

Sound is transmitted through the air as a pressure wave that varies both in time and space (e.g., see Figure 1). An **audio signal** is a function that describes the pressure, measured at a fixed point in space (e.g., your ear), as a function of time.

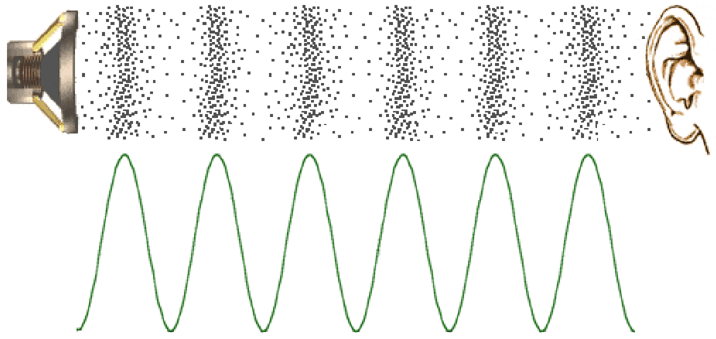


Figure 1: A speaker generates a pressure wave in space by vibrating. This generates a pressure profile that varies in time and across space. High pressure regions in space have more gas molecules and low pressure regions have fewer gas molecules. The sinusoid shows the pressure as a function of space for a fixed instance in time.

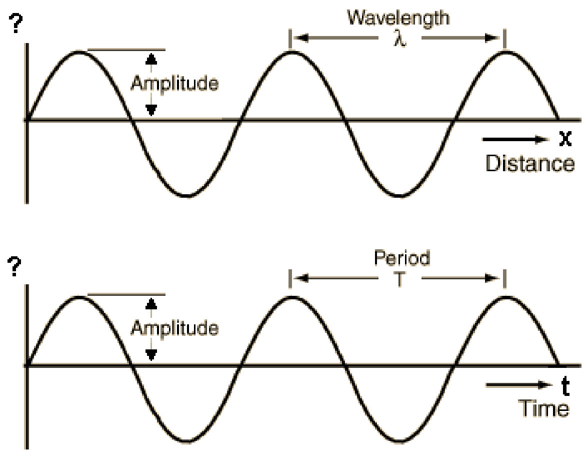


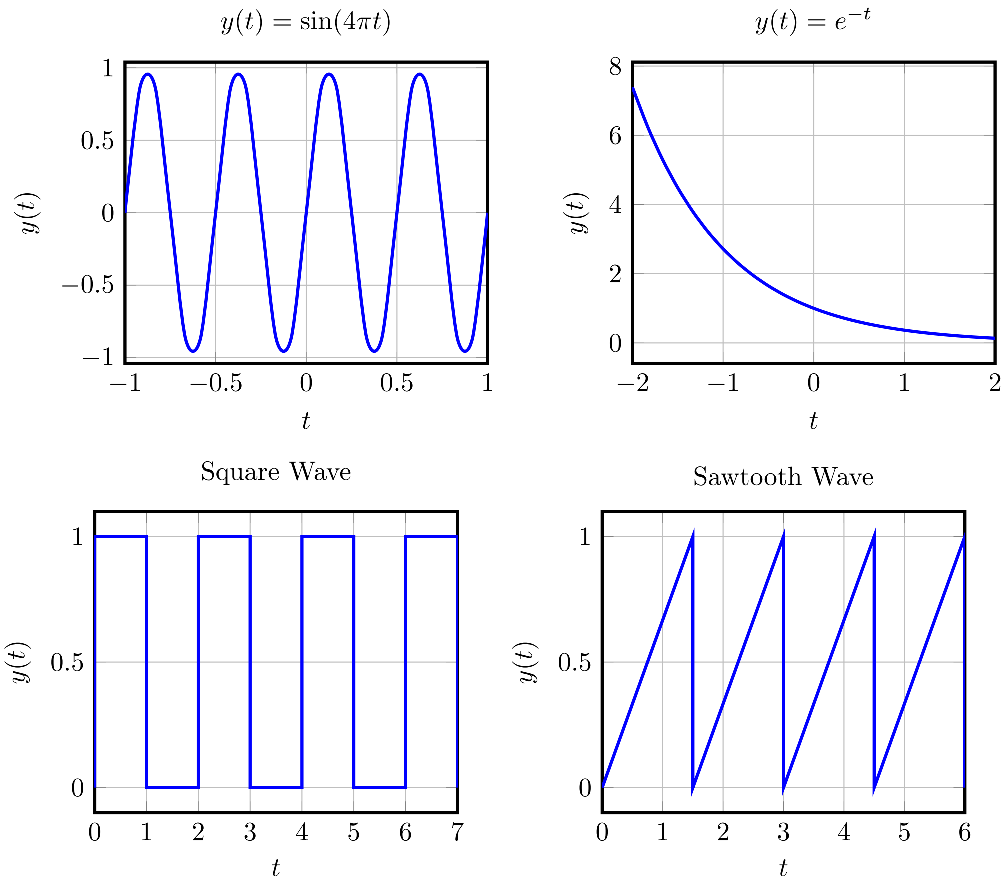
Figure 2: For a pressure wave, the distance between peaks in space is called the wavelength while the interval between peaks in time is called the period.

1.1 Signals

An audio signal $y(t)$ is **periodic** with **period** T seconds if it is formed by repeating a single segment of duration T seconds. The smallest possible T for which this is true is called the **fundamental period** of the signal. A signal with fundamental period T repeats one cycle every T seconds and has a **fundamental frequency** of $F = 1/T$ cycles per second. The terms "cycles per second" is also known as Hertz and is abbreviated Hz.

Example 1. Since the function $\sin(2\pi t)$ is periodic with a fundamental frequency of 1 Hz, the function $y(t) = \sin(2\pi f t)$ is periodic with fundamental frequency f Hz and fundamental period $1/f$ sec.

Example 2. For each signal below, determine if the signal is periodic and, if so, what is its fundamental period and fundamental frequency.



1.2 Digital Representations

Signals related to physical quantities, such as pressure, are typically considered to take **continuous values** that lie in some real interval and to be **continuous time** (i.e., the signal value is defined for all real valued times). On the other hand, digital computers can only represent a finite number of signal values and time instants.

A **discrete-time signal** is defined only for integer time instants. One can convert a continuous-time signal into a discrete-time signal by **sampling** at uniformly-spaced times. For example, $y[n] = y(nT_s)$ is a discrete-time sampled version of $y(t)$ with a sampling period of T_s. The **sampling frequency** $F_s = 1/T_s$ is defined to be the number of samples taken per second. Figure 3 shows some example discrete-time signals.

A **quantized signal** is a signal where the amplitude values are limited to finite set. If the set is large enough, then this detail can typically be ignored. For example, the compact disc (CD) audio standard

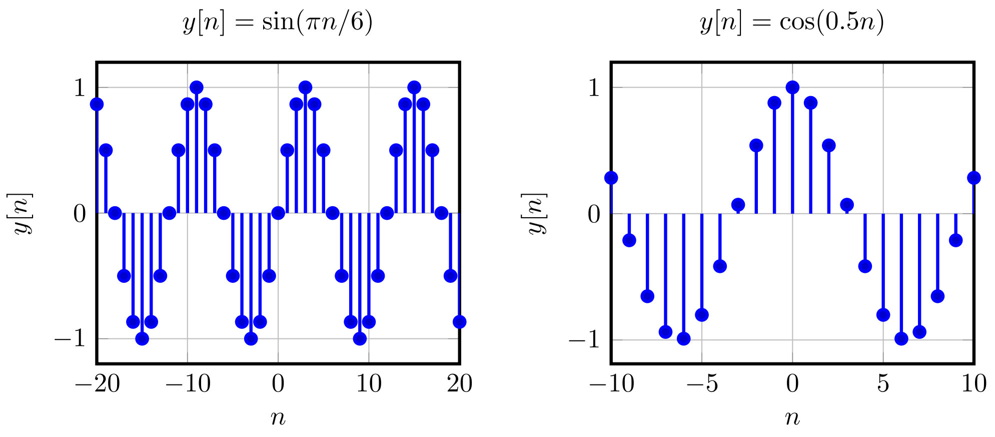


Figure 3: Examples of discrete-time signals

uses a sampling rate of 44,100 Hz and quantizes the signal to one of 65536 different values. These values were chosen to so that, for typical listeners, the perceived loss in quality is negligible. Also, the number $65536 = 2^{16}$ can be represented by 16 bits.

1.3 Matlab

Matlab is an interpreted language that makes it easy to manipulate matrices and vectors. Now, we will start to explore the use of MATLAB to generate and plot signals. To learn more about any function is Matlab, you can type "`help [function name]`" in the MATLAB command window. If you don't know the exact function name, you may also type "`lookfor [keyword]`". This will list all functions related to the keyword. Type in the following commands to see a few Matlab commands in action:

```
1+2
2-1
3*2
3/2
pi
cos(pi)
a1 = [1 2 3 4 5]
a2 = 1:5
max(a2)
b1 = [0 2 4 6 8]
b2 = 0:2:8
length(b2)
min(b2)
5*a1
a1+b1
b1-a1
a1 .* b1
help .*
a1 ./ b1
help ./
ones(1,10)
zeros(1,10)
```

Type (or cut and paste) the following commands to see how Matlab can be used to define, plot, and analyze signals:

```
t = -30:30              % Setup time indices
y = cos(2*pi*t/7)       % Discrete-time signal y
plot(t,y)               % Plot signal y
xlabel('t');            % Label plot
ylabel('y(t)');         % Label plot
title('cos(2*pi*t/7)')  % Label plot

Fs = 22050;             % Setup sample rate
Ts = 1/Fs;              % Setup sample period
t = 1:Ts:3;             % Generate time indices in seconds
y = cos(2*pi*t*800)     % Generate 800 Hz signal
soundsc(y,Fs)           % Scale and play sound through speaker
```

Exercise 3. Change the previous example to play a 400 Hz tone.

Exercise 4. Consider the following signal:

$$y(t) = -2\sin(2\pi ft) + 4\cos(\pi ft + p),$$

where $f = 5$ Hz, $p = \pi/4$, $F_s = 100$ Hz, and $t = 0 : 1/F_s : 2$. Use MATLAB to:

(a) Plot y

(b) Find the length of y

(c) Find the maximum of y

(d) Find the minimum of y

2 Audio Processing

For this section, you will need to download the accompanying files "guitar4.wav" and "guitar6.wav".

2.1 Importing WAV files

Matlab makes it fairly easy to import audio files. Type "`help audioread`" to find out more details. Use the following command to load and play a WAV file guitar riff:

```
[y,Fs] = audioread('guitar4.wav');     % Read WAV file
soundsc(y,Fs);                         % Play WAV file
```

Exercise 5. Repeat the previous steps with the file "guitar6.wav".

Next, we will extract a portion of the WAV file and determine the fundamental frequency of one note

```
index = 33001:34985;    % Index samples for second note
yy = y(index);          % Extract indexed samples into yy
soundsc(yy,Fs)          % Play extracted waveform
plot(index,yy)          % Plot extracted waveform
```

From the picture, we see that this portion of the signal appears to be periodic. One can estimate the fundamental frequency by counting the number of samples S in one cycle. Analyzing the units, one finds that

$$Hz = \frac{cycles}{second} = \frac{samples}{sec} \cdot \frac{cycles}{sample} = F_s/S.$$

Exercise 6. Can you think of a more accurate way to estimate the fundamental period and frequency from this plot. Describe your method and use it to estimate the frequency in Hertz.

2.2 Echo and Reverberation

An echo occurs when a sound reflects off a distance surface. Due to the longer path, the reflected sound arrives later than the original sound and can be heard as a distinct second copy of the original sound.

```
delay = 0.2;                % Delay in seconds
index = round(delay*Fs);    % Delay in samples
yy = [zeros(index,1); y];   % Zero pad the beginning to add delay
yy = yy(1:length(y));       % Cut vector to correct length
yy = yy+y;                  % Add original sound to echo
soundsc(yy,Fs);             % Play it
```

Exercise 7. Play with the value of delay to see the effect.

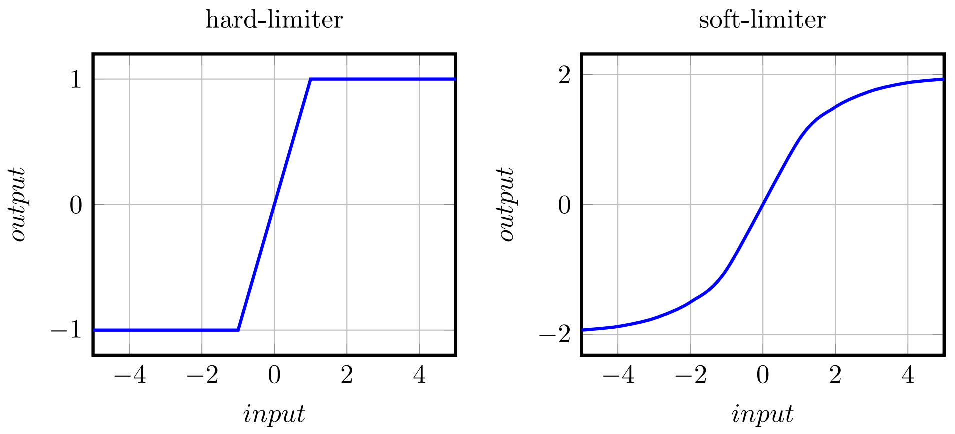
Reverberation (or reverb) is the effect generated by many copies of a sound with different delays summing together due to reflections off the walls of a room. In contrast to echo, the individual copies are not distinguishable but overall effect typically makes the sound richer or fuller.

```
delay1 = round(Fs*0.008);   % FIR Delay
delay2 = round(Fs*0.025);   % IIR Delay
coef = 0.7;                 % IIR Decay rate
yy = filter([1 zeros(1,delay1) coef],[1 zeros(1,delay2) -coef],y);
soundsc(yy,Fs);             % Play it
```

Exercise 8. Adjust the values of `delay1`, `delay2`, and `coef` to make a reverb that sounds better to you.

2.3 Distortion

An ideal amplifier simply takes the input signal and multiplies it by some constant larger than 1. Some amplifier designs are nearly ideal when the output level is not too large. But, their characteristics change for larger output values. In particular, the signal is distorted when the amplifier is driven into saturation. For vacuum tube amplifiers, this distortion has a soft edge and many rock musicians enjoy the resulting sound.



For our experiment, we will use a soft-limiter based on the inverse tangent function atan(x). Another good choice is the hyperbolic tangent tanh(x).

```
yy = atan(8*y);
soundsc(yy,Fs);
```

Exercise 9. Adjust the constant "8" in the previous example to find your favorite distortion.

2.4 Recording Sound in Matlab

Now, we will try some of these effects on sounds that we have recorded. The following code excerpt shows how to record sound in Matlab.

```
r = audiorecorder(Fs,16,1);          % Get recorder with Fs samples/sec, 16 bit, mono
record(r);                           % Start speaking or singing after this line
stop(r);                             % Execute this line when you are done
mywav = getaudiodata(r,'double');    % Get sound
soundsc(mywav,Fs);                   % Replay sound
```

Exercise 10. Now try recording your singing or humming. Then, plot the signal to find a nice periodic section. Based on the earlier example, try to estimate the fundamental frequency of that periodic section.

Exercise 11. Now try downloading a WAV or MP3 file from the internet. This file can be loaded into Matlab and processed in a similar. Ideally, the sound file should use a sampling rate of 44100 HZ. Also, if the file is stereo (i.e., it has left/right channels), then you will need to take the left channel by writing `y=y(1,:);`.

3 Filters and Equalization

A filter is a device that amplifies different frequencies by different amounts. Many of you have probably seen a picture of the "graphical equalizers" that were popular in the 1980s.

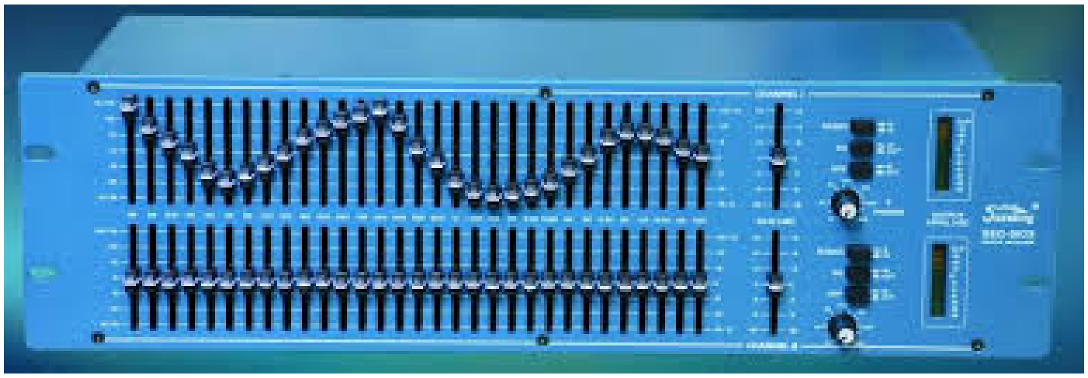


Figure 4: Graphical equalizer

The theory behind filtering a bit too mathematical for this workshop but it is covered in a junior-level course called "Signals and Systems". For now, we will take a cookbook approach and use Matlab to design 3 filters for us: a low-pass filter, a high-pass filter, and a band-pass filter.

```
lowpass = fir1(200,200/22050);                          % low-pass filter with cutoff 200 Hz
yylow = filter(lowpass,1,y);                            % filter signal
soundsc(yylow,Fs);pause;                                % play filtered signal, wait for keypress
bandpass = fir1(200,[200/22050 1000/22050],'DC-0');     % band-pass filter that passes 200-2000 Hz
yyband = filter(bandpass,1,y);                          % filter signal
soundsc(yyband,Fs);pause;                               % play filtered signal, wait for keypress
highpass = fir1(200,2000/22050,'DC-0');                 % high-pass filter with cutoff 2000 Hz
yyhigh = filter(highpass,1,y);                          % filter signal
soundsc(yyhigh,Fs);pause;                               % play filtered signal, wait for keypress
soundsc(yylow+yyband+yyhigh,Fs);pause;                  % play the sum off all bands, await keypress
yyeq = yylow+4*yyband+yyhigh;                           % 3-band EQ for mid-range boost
soundsc(yyeq,Fs);                                       % Play EQ with mid-range boost
```

4 Multiple Effects

Now that we have constructed 3 different effects in Matlab, we can put them together in an arbitrary fashion.

```
yyeqd = atan(8*yyeq);
yyfinal = filter([1 zeros(1,delay1) coef],[1 zeros(1,delay2) -coef],yyeqd);
soundsc(yyfinal,Fs);
```

Exercise 12. There are many other applications of DSP for audio. Try to list a few of them.