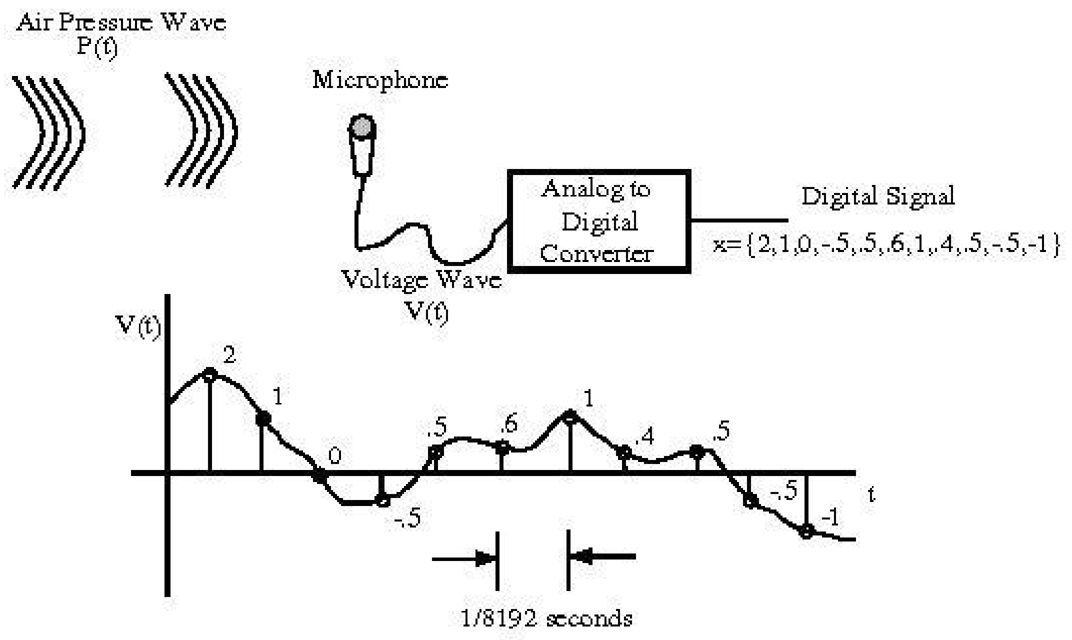
Sound Processing in MATLAB

What is digital sound data?



Getting some pre-recorded sound files (digital sound data)

- You will first need the following sound files. Right click on these and choose “Save Link As…” and put these in a folder and make sure you use the pull down menu option File àSet Path to put that folder in the MATLAB path (or simply put the .wav files in your correct working directory.

  road.wav
  hootie.wav
  lunch.au
  flute.wav
  tenorsax.wav
  mutedtrumpet.wav

Loading Sound files into MATLAB

- We want to read the digital sound data from the .wav file into an array in our MATLAB workspace. We can then listen to it, plot it, manipulate, etc. Use the following command at the MATLAB prompt:

```
[road,fs]=wavread('road.wav'); % loads "the long and winding road" clip
```

- The array **road** now contains the stereo sound data and **fs** is the sampling frequency. This data is sampled at the same rate as that on a music CD (fs=44,100 samples/second).

- See the size of road: **size(road)**

- The left and right channel signals are the two columns of the road array:

```
left=road(:,1);
right=road(:,2);
```

- Let's plot the left data versus time. Note that the plot will look solid because there are so many data points and the screen resolution can't show them all. This picture shows you where the signal is strong and weak over time.

```
time=(1/44100)*length(left);
t=linspace(0,time,length(left));
plot(t,left)
xlabel('time (sec)');
ylabel('relative signal strength')
```

- Let's plot a small portion so you can see some details

```
time=(1/44100)*2000;
t=linspace(0,time,2000);
plot(t,left(1:2000))
xlabel('time (sec)');
ylabel('relative signal strength')
```

- Let's listen to the data (plug in your headphones). Click on the speaker icon in the lower right hand corner of your screen to adjust the volume. Enter these commands below one at a time. Wait until the sound stops from one command before you enter another sound command!

```
soundsc(left,fs)      % plays left channel as mono

soundsc(right,fs)     % plays right channel mono (sound nearly the same)

soundsc(road,fs)      % plays stereo (ahhh…)
```

- Another audio format is the .au file format. These files are read in using

```
[lunch,fs2]=auread('lunch.au');

soundsc(lunch,fs2);
```

- To save an array as a .wav file, use **wavwrite( )**. Use **auwrite( )** for .au format output.

Let's *Mess* With the Signal (perform digital signal processing)

Reverse Playing

- To play the sound backwards, we simply reverse the order of the numbers in the arrays. Let's experiment with a small array. Type in the following commands:

```
y=[1;2;3;4;5]
y2=flipud(y)
```

- Note that **flipud** stands for flip upside-down which flips your array **y** and stores the inverted array in a new array called **y2**.

- Now let's try it on one of our sound arrays:

```
left2=flipud(left);
soundsc(left2,fs)
```

Digital Delay Experiment 1:

- Now let's add an echo to our sound data by adding to each sound sample, the sample from a previous time:

```
leftout=left;  % set up a new array, same size as old one
 N=10000;  % delay amount N/44100 seconds
for n=N+1:length(left)
        leftout(n)=left(n)+left(n-N);  % approximately ¼ second echo
end
```

- Note that these arrays are large and it may take some time for the processing to be completed. Compare the input and output by typing the following after the cursor returns, indicating that the processing is done:

```
soundsc(left,fs) % original

 % Wait until the sound stops before moving to next sound command

soundsc(leftout,fs) % signal with new echo
```

- This program first sets the output to be the input. This is simply a quick way to initialize the output array to the proper size (makes it operate faster). The loop starts at n=10001 and goes up to the full length of our array left. The output is the sum of the input at sample time n plus the input at sample time n-10000 (10000 samples ago, 10000/44100 seconds ago since the samples are spaced by 1/44100 seconds). Try some different delay amounts.

- Try it in stereo and we will echo left-to-right and right-to-left!

```
out=road;  % set up a new array, same size as old one
 N=10000;  % delay amount N/44100 seconds
for n=N+1:length(road)
         out(n,1)=road(n,1)+road(n-N,2);  % echo right-to-left!
        out(n,2)=road(n,2)+road(n-N,1);  % echo left-to-right!
end

soundsc(road,fs) % original
soundsc(out,fs) % echo
```

Digital Delay Experiment 2:

- Try the following variation on this theme, which keeps adding to the signal itself from 1000 samples ago slightly softened (multiplied by 0.8). Note that for this sound data the samples are spaced by T=1/8192 sec (fs2=8192 samples/sec).

```
[lunch,fs2]=auread('lunch.au');

out=lunch; % set up a new array, same size as old one
N=1000;  % delay amount N/8192 seconds
for n=N+1:length(lunch)
        out(n)=.8*out(n-N)+lunch(n);  % recursive echo
end

soundsc(out,fs2) % echo
```

- This echo process is like looking into a mirror and seeing a mirror with a reflection of the first mirror, etc! The echo goes on forever, but gets slightly quieter each time.

Digital Tone Control

- The following program (or "digital filter") is designed to soften high frequency components from the signal (treble). It retains the low frequency components (bass). Applying this digital filter has the same effect as turning down the treble tone control on your stereo. The design of this code is not so obvious. The Electrical Engineering students will learn more about this type of frequency selective digital filtering in ECE334 Discrete Signals and Systems.

```
[hootie,fs]=wavread('hootie.wav'); % loads Hootie
out=hootie;

for n=2:length(hootie)
```

```
        out(n,1)=.9*out(n-1,1)+hootie(n,1); % left
        out(n,2)=.9*out(n-1,2)+hootie(n,2); % right

    end
```

- Compare the input and output as before. Note that the modified signal sounds muffled in comparison to the input data. This is because the high frequency components have been suppressed in the output.

```
soundsc(hootie,fs)      % original

soundsc(out,fs)         % low pass filtered
```

- A small change in our digital filter allows us to boost high frequencies and suppress low frequencies:

```
out=hootie;

for n=2:length(hootie)

        out(n,1)=hootie(n,1)-hootie(n-1,1); % left
        out(n,2)=hootie(n,2)-hootie(n-1,2); % right

end

soundsc(out,fs)         % high pass filtered
```

Changing the Speed

- The sampling frequency **fs** tells us how much time goes between each sample (T=1/fs). If we play the song with more or less time between samples than was originally there when recorded, the speed will seem off, producing interesting effects...

```
soundsc(hootie,fs/1.5) % How slow can you go?

soundsc(hootie,fs*1.5) % The Chimpmonks!
```

Removing (Minimizing) Vocals

- In most popular music recordings, the vocal track is the same on the left and right channels (or very similar). The volume of the various instruments are more unevenly distributed between the two channels. Since the voice is the same on both channels, what would happen if we subtract one channel from the other and listen to the result?

  ```
  soundsc(left,fs);         % Original left channel

  soundsc(left-right,fs);   % Long and winding road, virtually no vocal
  ```

  Notice the voice is virtually eliminated…

- Try it with Hootie…

  ```
  soundsc(hootie(:,1),fs);                  % Original left channel

  soundsc(hootie(:,1)-hootie(:,2),fs);      % Hootie, reduced vocal
  ```

- You still hear some vocal here because this song uses a stereo reverberation effect and the echo moves back and forth between left and right channels (like our stereo delay above). This makes the voice unequal from left to right channels.