Lab 2
Functions in matlab and Playing Sounds

EE 235: Continuous-Time Linear Systems
Department of Electrical Engineering
University of Washington

1 Introduction

In this lab, you will learn about matlab function files, which are similar to the scripts that you saw in Lab 1. Function files also have the file extension `*.m` and thus are also called 'M-Files'. You will create a number of functions for manipulating sound signals, and use them to create a groove or short song.

The difference between a function and a script is that a script is simply a sequence of commands to be run, while a function can take parameters and return values. A function is therefore a generalized and flexible script.

A side effect of functions is that any variables created within the function are not available after the function has finished running. That is, the variables inside a function are only active within the `scope` of the function, and simply don't exist after the function exits. By contrast, the variables in a script are added to your workspace and you can look at them anytime afterward. To learn more, read the following help pages:

```
help function
help script
```

To create function files, you need to use a text editor such as `Notepad` on a Windows PC or `emacs` on Linux and Mac computers. matlab also has an internal editor that you can use within the matlab GUI. You can start the editor by clicking on an M-file within the matlab file browser. All of these editors are standard tools and will produce plaintext files that matlab can read. **Make sure that your new file is in matlab's working directory, or else you won't be able to run it.**

[1]Last revision: Fri Apr 9 01:56:34 EDT 2010
[2]http://www.creativecommons.org/licenses/by/2.0

2 Sound in MATLAB

Download the sound samples from the sound resources page [3] and save them to your working directory. Use `wavread` to load `*.wav` files, and use `load` to load `*.mat` files. Note that `load` will instantiate any variables that are named inside a `.mat` file, but you need to explicitly capture the output of `wavread` via something like:

```
y = wavread('SoundFileName');
```

You may need to read the `help` pages for these functions.

Sounds are represented digitally on a computer, which means that the analog signal is sampled at fixed time intervals and only these samples are stored. You'll learn more about this later. For now, you just need to keep track of the time interval T_s (or, equivalently, $F_s = 1/T_s$, which is the sampling rate) when playing sounds in MATLAB.

To learn how the time domain signal sounds, use the `sound` command to play it. You must specify the playback sample rate F_s, which ought to be the same as the rate at which the sound was originally sampled. The sound files provided on the website for this lab have a sample rate of 8000Hz.

For example, to play a hypothetical sound called `bell`, you would enter:

```
>> Fs=8000;
>> sound(bell, Fs);
```

If your value for F_s is different from the sampling rate, then you will effectively be performing time scaling. An F_s that is lower than the sampling rate will slow down the sound, and a larger F_s one will speed it up.

You also need to know the sampling rate in order to accurately plot sounds against time. As you may recall from Lab 1, `plot(t,y)` only works with vectors that are the same length. If you know the sampling frequency for `y`, and also how many samples are in `y`, then you can figure out how long `y` is, in seconds. You can now easily construct an appropriate time vector against which to plot sound `y`. You will need to construct a new time vector to plot each sound, unless you have two sounds that have the same duration.

Exercise 1:
- Plot each of the sound samples from the resource page, and predict how the volume will change over time when you play them.

Exercise 2:
- Play each of the sound samples. How good were your predictions?

When working with sound in MATLAB, it is important to remember that the values of the audio signals are in the range $[-1, 1]$. Anything out of that range will be clipped to 1 or

[3] http://cnx.org/content/m13854/latest/

-1 when you play the sound, which will distort it. Keep this in mind when you write your functions! Your functions should expect inputs with values in the range $[-1, 1]$ and produce outputs within that same range.

3 Function Files

3.1

You need to write functions that can modify sound signals before you can combine them to create your musical groove. After all, wouldn't it be boring to make a tune that consists of exactly one note over and over again? You will soon create functions that can time-scale, reverse, delay, fade, and repeat sounds. You will also write a simple mixer that can layer two sounds on top of each other. Remember that the function files you create must have comments, just as you made in Lab 1. However, instead of adding a header to your function files, add the information as a footer (at the very end of the file). This way the first lines of the function contain the function definition and help comments.

MATLAB has many built-in functions. Two closely-related functions that you might use in this lab are `fliplr` and `flipud`, which allow you to time-reverse a signal in one operation. Read `help fliplr` and `flipud` for details.

Exercise 3:
- Time-reverse one of the sounds that you downloaded.
- Plot both the original and reversed signals, and play them.

Download the function file `timescale.m`[4] into your working directory. Read the M-file to see how to run it.

Exercise 4:
- Use the `timescale` function on a sound file to both speed up and slow down one of the sounds.
- Why does the pitch change?

3.2 Fader

Download the function file `fade.m`[5] into your working directory. Read the code, including comments. See that function files start with something like:

```
function [ output1, output2 ] = FunctionName( parameter1, parameter2 )
```

[4]http://cnx.org/content/m13555/latest/timescale.m

[5]See course website

This function MUST be written in a file called `FunctionName.m` (use a descriptive name, obviously). Then this first line means that the function will accept some parameters, and will output whatever is in `outputVariable` when it's done running. If `output1` isn't defined in the function, then it won't return anything. Also note that the `output1` is only defined within the scope of the function. You can't use this value unless you explicitly capture the function's return value at the MATLAB prompt, like so:

```
>> x = FunctionName(param1, param2)
```

The first set of comments (immediately after the first line in the function) is what will be displayed on screen if you type `help FunctionName`. This should be a general description of your function and its syntax.

Now enter `help fade` at the MATLAB prompt. If you saved the file correctly, you will see the help text from the M-file in response to `help fade`. Notice that you can use the new command that you just added to MATLAB as if it were a built-in function.

Enter the following at the MATLAB prompt:

```
>> time = 0:0.01:1;
>> y = cos(time .* pi .* 0.25);
>> plot(time, fade(y));
```

You can see in the plot that `fade` does fade out the cosine wave. You can use this function on audio signals as well. This function works fine, but offers the user no control.

Therefore, your assignment is to edit the `fade` function so that you can adjust the slope of the ramp that fades the signal. In order to do so, notice in the code that there's an unused parameter named `level`. The variable `level` should be the final volume level after fading, as a percentage of the original input.

You must ensure that `level` only takes on values between 0 and 1, because you can't fade to less than 0% nor more than 100% of the original input volume. You also need to handle the case where the user does not specify the value of `level`. The desired behavior in this case is specified in the code. You may find the `exist` command helpful.

Exercise 5:
- Modify the `fade` function to use the parameter `level` as described above. Remember to comment your code, and add a footer with your personal information.
- Write down what you predict will happen if you `fade` the cosine wave `y` with `level` set to each of the following: `0, 1, 0.25, -2, 100`.
- Now show what happens when you run `fade` with each of those values for `level`.

Plot the input `y` and each of the faded outputs.

3.3 Repeater

Now that you've seen the insides of a function file, it's time to write your own. A repeater is a function that plays a particular sound a specified number of times. It seems logical that the function should take a sound file `sound` as an argument, in order to know what to play. You wouldn't want to stifle anyone's creativity, so `repeat` should also take a parameter `N` in order to let the user specify how many times the sound should be played.

Therefore, the first line of your function might look like this:
`function [ out ] = repeat(sound, N)`

A simple way to repeat things an arbitrary number of times is with a `for` loop. Inside the `for` loop you will need to concatenate the sound signals. If you have two vectors `x` and `y`, you can concatenate them as follows:

```
>> x = [1 4 2 2];
>> y = [3 6 8 0];
>> x = [x y]
x  =
   1 4 2 2 3 6 8 0
```

Exercise 6:
• Implement `repeat.m` as described above. It should take two parameters, `sound` and `N`.
• Demonstrate what happens when you set `N` to be each of `3,0,-1`.

3.4 Delay

The next function to implement is one that time-delays a signal by some amount of time `delay`. A time delay is the same as prepending silence to the original signal. Because you're working with digital data, a `0` sounds like silence. You can therefore implement a time delay by zero-padding the start of the vector containing the signal. To zero-pad something means to fill a space with only zeros, so in this case you're concatenating some number of zeros with the signal. The number of zeros to add will depend on the time delay and the sample rate of the signal. Adding 4000 zeros to a signal sampled at 4kHz will delay it by 1 second; adding 4000 zeros to a signal sampled at 8kHz will delay it by only 0.5 seconds. You may find the MATLAB command `zeros` to be useful.

Most of the sound files in these labs are sampled at 8000Hz, but this is not always true. You may want to change this later, so it's good programming practice to have the sample rate `Fs` be another input to the function.

Exercise 7:
• Implement `delay.m` to time-delay a sound, as described above.
It should take three parameters: `sound`, `delay`, and `Fs`.
• Plot the original signal and a delayed version of it (use `subplot`, of course) in order to verify your output.
• What happens when you delay by a negative amount?

3.5 Mixer

The last helper function you will write is a mixer (`mixer.m`). This mixer should layer two sounds on top of each other so that they play simultaneously. A simple way to do this is to add the signals together. Your code should be able to handle two sounds that are not the same length.

However, there are some considerations: Audio signals have a range of $[-1, 1]$, thus the output of your mixer must also fall within this range of $[-1, 1]$. Note that the summed signal will not satisfy this requirement by default! Anything out of range will be distorted, which is bad, so you must re-scale the summed sound before you output it. You can find one way to solve this problem by looking at the source code to the MATLAB function `soundsc` (type "`type functionName`" to see the source code for a function). If you find the maximum value that your summed input takes, and you can scale that to fall within $[-1, 1]$, then your whole signal will also fall within that range.

Exercise 8:
- Implement the function `mix.m`, which mixes two sounds as described above.
- Pick two different sounds, plot them, mix them, and plot the result to verify your work.

4 Rock On

It's time to use the tools you've created! Write a script (not a function) to assemble a 10-30 second musical groove in one long sound vector. Use concatenation and your repeater to play sounds in sequence in a track, fade or shift them, and use your mixer to layer the tracks together. You can make additional functions if you want. For example, you could modify `repeat` to allow you to insert silence between repeats, and so on.

After you've created your sound file, you should save it with the `wavwrite` command. Remember to specify your sample rate F_s, which is 8000Hz for the provided sounds!

Exercise 9:
- Write a script `groove_YourName.m` that creates your musical groove and writes it to the file `groove_YourName.wav`.
- Play your groove for the TA, and use its plot explain how you created it.

EOF