Octave's External Audio Package
CS177a Final Project

Jeremy Patton and Nadav Havivi

May 14, 2011

1 Pre-requisite: SoX

The audio package is designed to use SoX. This is a powerful command-line tool for playing, recording, and editing audio. Windows and MAC: http://sox.sourceforge.net/ Linux: Open up a terminal and type the following command:

```
Debian-based Linux:
sudo apt-get install sox
Fedora Core Linux:
sudo yum install sox
```

2 Loading, Saving, and Analyzing

2.1 au.m

The usage of this tool is `au(x, fs, lo [,hi])`, where `x` is the array of values for the sound, `fs` is the frequency (or speed), `lo` is the lower bound of the section of the array, and `hi` the upper bound. If `hi` is not given, then this function returns only the value at `lo`, but if it is [] then the array is clipped to the end. Similarly, if `lo` is [], the returned array will start at the beginning of x. This function takes x and returns the interval defined by `[(lo*fs/1000)+1:(hi*fs/1000)+1]`.

2.2 auload.m

As can be surmised by its name, this function in its most basic use loads a sound file and returns an array corresponding to that file's data. A full use looks like this: `[data, rate, sampleformat] = auload(path)`, where `data` is the array of sound values, `rate` is the `fs` as described before, and `sampleformat` is how the saund data is aggregated and put together (something which we will not go into again in this paper). As of this version of the package, `auload.m` only accepts .wav, .au, and .aiff filetypes.

2.3 ausave.m

This function operates in the opposite direction as `auload.m`. Its use is in the form of `ausave('filename.ext', x [, fs, sampleformat])`. `filename.ext` is the name of the file to be saved, where `.ext` is the extension. This version of the package, again, only accepts .wav, .au, and .aiff filetypes. `x` is the sound data array to be saved, and optional arguments `fs` and `sampleformat` are the same as in previously discussed functions.

2.4 auplot.m

Possibly the most interesting function in the package, `auplot.m` does exactly what is says: it plots an array of sound data. Its use is `[y, t, scale] = auplot(x [, fs [, offset [, plotstr]]])`, where `fs` is initialized to 8000 if not secified. `offset` is the offset from the beginning of the file to plot, and `plotstr` is the style of plotting to use, such as `"-g"` or `*b`. This function will plot each channel of a sound file on a separately, so that they can be read individually. See attached image ding.png for an example of this.

3 Playback, Clipping, and some Editing

3.1 sound

`sound(x [, fs, bs])`

The sound function provides playback of an audio file. The argument 'x' represents the audio file *after* the file has been loaded with **auload**. 'fs' is the frequency, or speed, at which the audio plays. If this argument is not

entered, the default value is 8000 hertz. Increasing this number will speed up the playback. Conversely, decreasing the number will slow the playback. Finally, 'bs' is the buffer size. Adjusting this will determine how much of the audio file is loaded into the buffer before playback begins. If this argument is not entered, the default value is equal to the frequency or 'fs'. Decreasing this number will result in a stuttered playback. Increasing this number will consume unnecessary memory. It is suggested you do not enter this argument.

```
Example
octave:1>[x,fs]=auload("example.wav");
octave:2>sound(x,fs)
play WARN au: header size 24 is too small
-: (au)
Encoding: Signed PCM
Channels: 1 @ 16-bit
Samplerate: 22050Hz
Replaygain: off
Duration: unknown
In:0.00 00:00:02.97 [00:00:00.00] Out:65.5k [    -=|=-    ]        Clip:0
```

3.2 soundsc

`soundsc(x, fs, limit) or soundsc(x, fs, [lo, hi])`

This function allows for scaling of the amplitude of the audio. The results of this scaling can be saved as a vector. Using the first form of the function, limit will simply scale the amplitude by this factor. If, for example, the amplitude at a certain point is .5, scaling by .1 will have the following effect on the amplitude:

$$.5/.1 = 5 \tag{1}$$

Using the second form of the function will scale the bottom half of the wave by 'lo', and the top half of the wave by 'hi'. If 'lo' and 'hi' are the same values, then this function has the same effect as using the limit argument described above.

```
Example
octave:1>x=auload("example.wav");
octave:2>y=soundsc(x,fs,.1);
```

```
octave:3>x(1),y(1)
ans =  0.0013581
ans =  0.013581
```

3.3 clip

`clip(x) or clip(x, hi) or clip(x,[lo,hi])`

Clip will put a ceiling and/or floor on the amplitude of the sound wave. If no values for either lo or hi are entered, 0 and 1 are assumed respectively. The output of this function may also be saved as a vector.

Example

```
octave:1>x=auload("example.wav");
octave:2>y=clip(x,.001);
octave:3> x(1),y(1)
ans =  0.0013581
ans =  0.0010000
```