ECE 4680 DSP Laboratory 2: Speech Signal Processing Using Python Sound Functions

This lab is to be worked in teams of most two. Each team will submit a lab report describing their results. An E-mail ZIP package containing .wav files will also be turned in or a lab demo to the instructor, to allow evaluation of the processing algorithms.

Introduction

Using a basic PC sound system and Python (~~MATLAB~~) wave file processing functions, *near real-time* audio signal processing can be investigated. In this experiment you will become familiar with `*.wav` files and Python functions in `ssd.py` for reading and writing wave files from the Jupyter notebook of the qtconsole. When worked in the Jupyter notebook there is a very nice audio *widget* available that allows direct playback via your PC sound system. Audio interface functions for Python (see Figure 1) are available in module `ssd.py`. An Jupyter notebook example using the audio control will be placed on the Web Site.

ECE 4680 and ECE 4650/5650

Playing Wavefiles

Beyond the typical `imports`, you now include the last line below to be able to use the `Audio` control. The module `ssd` contains the functions `fs, x = ssd.from_wave('file.wav')` and `ssd.to_wave('file_name',fs,x)` for importing and exporting wav files to the Python workspace.

In [2]:
```
%pylab
%matplotlib inline
import scipy.signal as signal
import ssd
from IPython.display import Audio, display
```

```
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib
```

In [21]:
```
fs,x = ssd.from_wav('zero.wav')
```

The wave file must be in your path for this to work.

In [22]:
```
Audio('zero.wav')
```

Out[22]:

In [23]:
```
y = 1.2*x
print('Maximum value in y is %4.4f' % (max(max(x[:,0]),max(x[:,1]))))
```

```
Maximum value in y is 0.4441
```

In [24]:
```
ssd.to_wav('zero1.wav',fs,y)
```

The basic investigation for this experiment will be the use of *butt splicing* to either speed up or slow down the playing time (delivery) of a recorded speech signal (in Python/Numpy an `ndarray`), without altering the pitch. Think about this for a moment. If a signal is played back at a sampling rate of twice the original record value, the play back time is cut in half, and the pitch is

doubled. Likewise if a signal is played back at a sampling rate of one half the original record value, the play back time is doubled, and the pitch is halved. Using butt splicing of speech segments it is possible to maintain the recorded pitch while either slowing down or speeding up the play back time.

Digital Signal Processing and Windows Wave Files

With the multimedia enhancements to Windows came the ability to handle digitized audio signals in what is known as the wave (`*.wav`) file format. A Wave audio file is a binary file format for storing continuous-time audio source material, in sampled data form, as a result analog-to-digital (A/D) conversion. A sound system equipped PC generally includes two channels for recording and two digital-to-analog (D/A) converters for play back. The block diagram of Figure 2 depicts the DSP capability that a sound system equipped PC typically has.

The current DSP capability connected with wave file manipulation is not geared toward real-time processing. It is best suited to what one might call near real-time signal processing. Signals can be processed after being initially recorded, then saved as a new wave file for play back at another time.

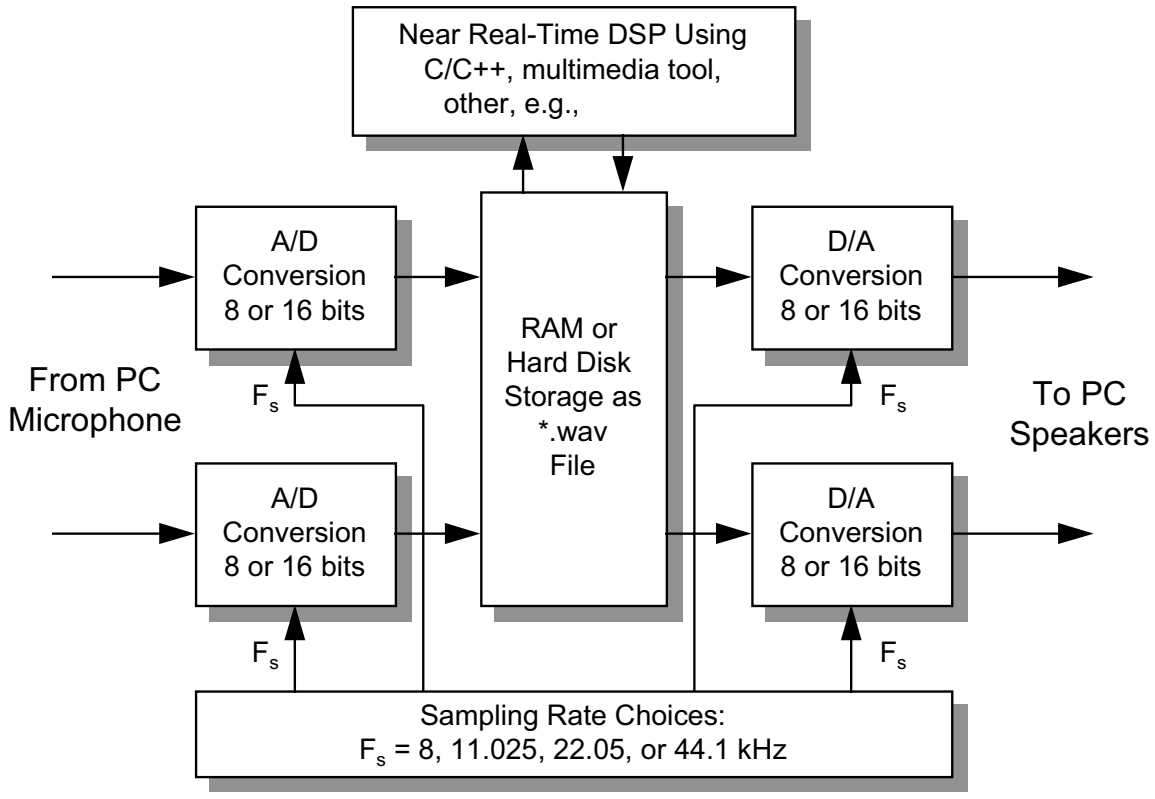

Figure 2: DSP capability of the typical sound card equipped PC.

Wave Files and Sound in Python Using Numpy ndarrays

In Python, in particular IPython/Jupyter notebook, the ability to read and write `*.wav` files is available via the module `ssd.py`. The functions are summarized in Table 1. The functions in

Table 1: Python sound functions for Win/Mac/Linux OS's.

Python Sound Related Functions	
`to_wav( filename,rate,x)`	Write a wave file. A wrapper function for `scipy.io.wavfile.write` that also includes `int16` scaling and conversion. Assume input `x` is [-1,1] values.
`from_wav( filename)`	Read a wave file. A wrapper function for `scipy.io.wavfile.read` that also includes `int16` to float [-1,1] scaling.

`ssd` are simply wrapper functions of a portion of the scipy module io. In particular access just `scipy.io.wavefile` via

```
from scipy.io import wavfileta
```

To play a wave file you use the Audio control available in the IPython notebook (see Figure 1) or use the audio player of you OS to play the wave file directly. There are sound playing functions from within Python itself, but some system configuration is required.

Simple Rate Changing Independent of Pitch

Without getting too detailed this section explains a simple technique for either increasing or decreasing the playing time of a sound vector, without altering the sound pitch. Increasing the rate refers to decreasing the playing time, while decreasing the rate implies increasing the playing time. Both operations can be solved in an approximate way by *butt splicing* speech segments of say 45 ms or so. Butt splicing in DSP is analogous to adding or removing segments of magnetic recording tape by end-to-end taping them together to form a new edited tape. Butt splicing of speech sequences thus implies that no transition smoothing is used, just a hard end-to-end connection.

Decreasing Playback Time

To decrease the playback time, yet retain the proper pitch, all we need do is to periodically remove short segments of the original speech vector, butt splice the remaining pieces back together, then play it back at the original recording rate. If the pattern is save 45 ms, discard 45 ms, save 45 ms, etc., the new sound vector will be half as long as the original, thus it will play in half the time. A graphical description of the operation in terms of Python `ndarrays` is shown in Figure 3. The

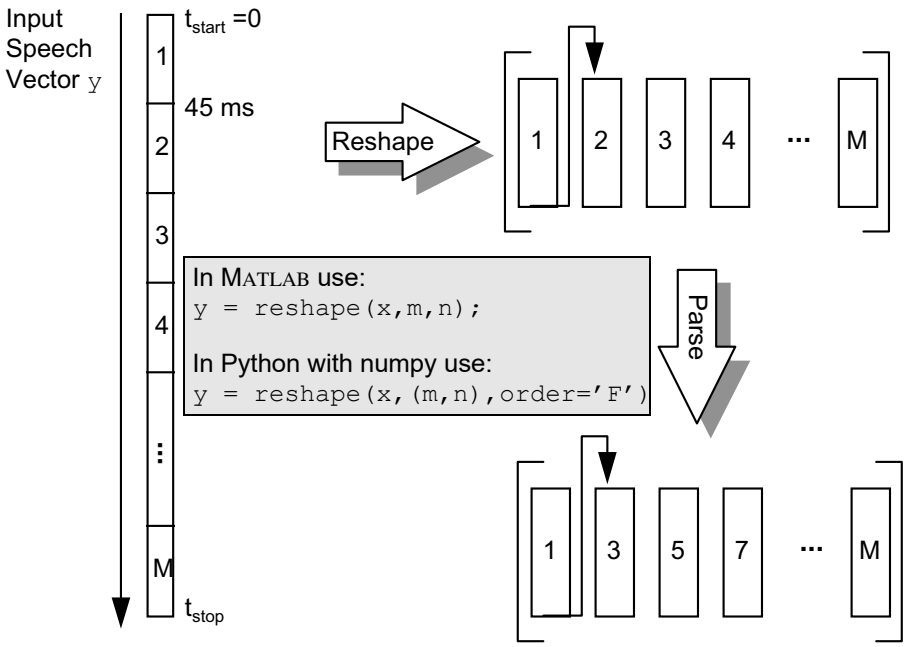

Figure 3: Butt splicing to decreasing playing time yet maintain sound pitch.

segment length may need to be adjusted for best sound quality. Note for the case of Python, the option `order='F'` insures that Fortran ordering is taken in the reshape. This also matches the way MATLAB does things.

Increasing Playback Time

To increase the playback time, yet retain the proper pitch, all we need do is to periodically repeat short segments of the original speech vector, again using a butt splicing technique, then play it back at the original recording rate. If the pattern is say 45 ms, repeat previous 45 ms, save next 45 ms, etc., the new sound vector will be twice as long as the original, thus it will play in twice the time. Python Hint:

```
In [226]: A = array([[1,2],[3,4]]) # make sure to create an ndarray

In [227]: hstack((A,A))
Out[227]:
array([[1, 2, 1, 2],
       [3, 4, 3, 4]])

In [228]: hstack((A.T,A.T))
Out[228]:
array([[1, 3, 1, 3],
       [2, 4, 2, 4]])

In [229]: hstack((A.T,A.T)).T
Out[229]:
```

```
array([[1, 2],
       [3, 4],
       [1, 2],
       [3, 4]])
```

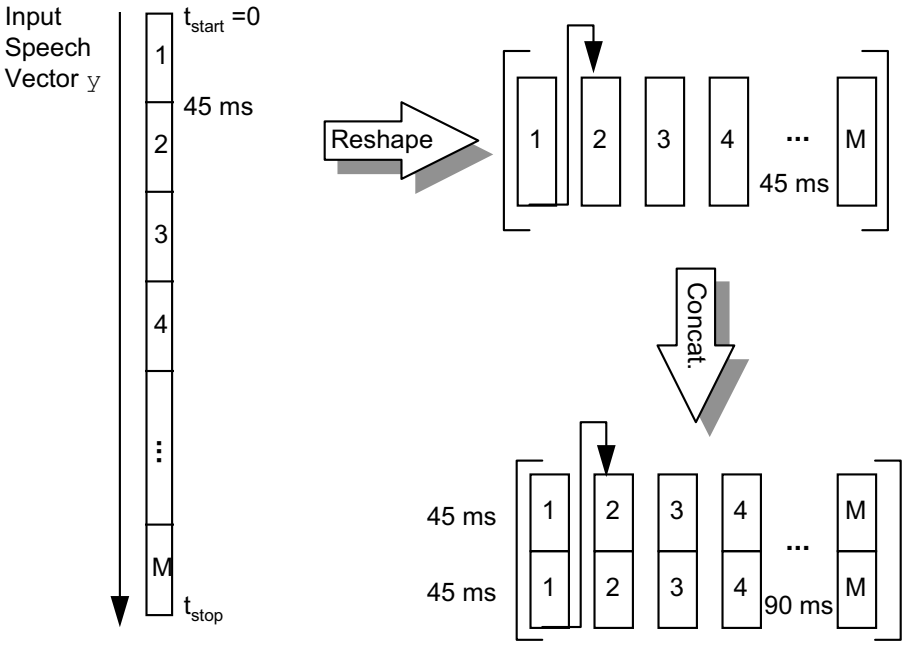

Figure 4: Butt splicing to increase playing time yet maintain sound pitch.

Experiments

1. Create a short wave file using 16 bits per sample resolution, $F_s = 11.025$ ksps, and duration of about 30 seconds, using GoldWave or a similar wave recording application.
2. Import the wave file into the IPython/Jupyter notebook workspace using the function

   ```
   fs,y = ssd.from_wav('test_file.wav')
   ```

3. Verify that `sound(y,Fs)` plays the vector through the sound system, and that raising or lowering Fs changes both the duration and pitch of the message. Using the length of `y` and `Fs` determine the exact time duration of the played sound vector.
4. Verify that MATLAB plays a matrix by columns. First convert the vector y into a matrix using

   ```
   >> reshape(y, M, N),
   ```

   where `M*N = length(y)`. Then play the matrix using sound. Verify that scrambled speech is heard if you play the transpose of your matrix.
5. Butt splice y by removing alternate 45 ms speech segments to halve the delivery time and maintain the same pitch. Export your modified speech vector back into a wave file and verify that it is playable.
6. Butt splice `y` by inserting redundant 45 ms speech segments to double the delivery time and maintain the same pitch. Export your modified speech vector back into a wave file and verify that it is playable.
7. Try variations on 5 and 6 to improve the quality, or further alter the rate change.
8. Analyze your speech waveforms using the spectrogram function

   ```
   >> spectrogram(y,FFTLENGTH,Fs,[WIND],OVERLAP)
   ```

   Recall from ECE 2610 that a spectrogram is a frequency spectrum versus time plot.
9. Summarize your findings.
10. Prepare for instructor demo.

Bibliography/References

[1] Tim Kientzle, *A Programmers Guide to Sound*, Addison-Wesley, Reading, Ma, 1998.