# Audio_Record_Playback_new

June 14, 2025

[1]:
```
from numpy import *
from matplotlib.pyplot import *
import sk_dsp_comm.sigsys as ss
# import sk_dsp_comm.pyaudio_helper as pah
import pyaudio_helper_old as pah
import sk_dsp_comm.fir_design_helper as fir_d
import scipy.signal as signal
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
from IPython.display import Audio, display
from IPython.display import Image, SVG
```

[2]:
```
%config InlineBackend.figure_formats=['svg'] # SVG inline viewing
#%config InlineBackend.figure_formats=['pdf'] # render pdf figs for LaTeX
```

## 1 Introduction

In this notebook (also the Appendix for ECE 4680 Lab 2) we consider audio processing, real-time and near teal-time, in the Jupyter notebook. If you are reading the PDF rendering of this notebook, it was first exported from the notebook version as LaTeX and then the `.tex` file was edited in VS Code using the LaTeX Workshop extension, to include screenshots of GUI controls from the running Jupyter notebook. Finally a PDF file was produced.

There are two topics to be covered here:

1. Using `pyaudio_helper.py` from Scikit-DSP-Comm for audio recording and playback/steaming via audio looping
2. Using the `Ipython.display` function Audio() to playback existing `wav` files or `ndarray` 1D signals

### 1.1 1.0 Pyaudio_helper for Recording and Playback

In this first subsection we consider the use of PyAudio via `scikit-dsp-comm.pyaudio_helper` to record and playback audio samples streams is described. The `ipywidgets` or `jupyterwidgets` are used to provide GUI controls inside the notebook. The widgets must be installed on your system. In particular when using `Jupyter Lab`, a recently released upgrade to the older `Jupyter Notebook` interface, requires additional installation steps.

A simplified block diagram of PyAudio *streaming-based* (nonblocking) signal processing when using `pyaudio_helper` and `ipython` widgets is shown in the figure below.

Once `pyaudio_helper` is imported (here via `import sk_dsp_comm.pyaudio_helper as pah`) you can see what audio devices are available on your or lab system you are in front of:

[3]:
```
pah.available_devices()
```

[3]:
```
{0: {'name': 'iMic USB audio system', 'inputs': 2, 'outputs': 2},
 1: {'name': 'MacBook Pro Microphone', 'inputs': 1, 'outputs': 0},
 2: {'name': 'MacBook Pro Speakers', 'inputs': 0, 'outputs': 2},
 3: {'name': "Mark's iPhone Microphone", 'inputs': 1, 'outputs': 0}}
```

Note if you plugin additional USB audio devices or you PC is *docked* in a docking station, additional devices will be displayed. If devices are added after the Python kernel starts, you will have stop and restart the kernel for the new devices to be identified.

### 1.1.1 Single Channel Recording with Record Monitor Level Control

Here we make a single channel recording (mono) using an external mic input, then save the output to a `wav` file for later playback in a repeating loop. The raw capture samples could also be manipulated in Python statically before playback or in real-time while being played back.

To aid the recording process to slider widgets are configured: (1) To set the record level into the capture buffer `DSP_IO_1.data_capture` and (2) to set the playback level so that if output device 5, in this case, can be brought down to avoid unwanted feedback getting into the mic input. Due to the latency with the PyAudio processing, feedback from output input will result in echo.

[12]:
```
Record_gain = widgets.FloatSlider(description = 'Record Gain',
                continuous_update = True,
                value = 1.0,
                min = 0.0,
                max = 2.0,
                step = 0.01,
                orientation = 'horizontal')
Monitor_gain = widgets.FloatSlider(description = 'Monitor Gain',
                continuous_update = True,
                value = 0.1,
                min = 0.0,
                max = 2.0,
                step = 0.01,
                orientation = 'horizontal')
```

[9]:
```
# define callback (3)
# Here we configure the callback to play back a wav file
def callback1(in_data, frame_count, time_info, status):
    global DSP_IO_1
    DSP_IO_1.DSP_callback_tic()

    # convert byte data to ndarray
```

```
    in_data_nda = np.frombuffer(in_data, dtype=np.int16)
    # Ignore in_data when generating output only
    #*******************************************
    x = in_data_nda.astype(float32)
    y = Record_gain.value*x
    # Note wav is scaled to [-1,1] so need to rescale to int16
    #y = 32767*x.get_samples(frame_count)
    # Perform real-time DSP here if desired
    #
    #*******************************************
    # Save data for later analysis
    # accumulate a new frame of samples
    DSP_IO_1.DSP_capture_add_samples(y)
    #*******************************************
    # Monitor level control
    y *= Monitor_gain.value
    # Convert from float back to int16
    y = y.astype(int16)
    DSP_IO_1.DSP_callback_toc()
    return y.tobytes(), pah.pyaudio.paContinue
```

[13]:
```
# fs, x_wav2 = ss.from_wav('Music_Test.wav')
# x_wav = (x_wav2[:,0] + x_wav2[:,1])/2
# x = pah.loop_audio(x_wav)
DSP_IO_1 = pah.DSP_io_stream(callback1,1,2,fs=44100,Tcapture=20)
DSP_IO_1.interactive_stream(Tsec=10)
widgets.HBox([Record_gain,Monitor_gain])
```

```
interactive(children=(ToggleButtons(description=' ', index=1, options=('Start␣
↪Streaming', 'Stop Streaming'), v…
```

[13]:
```
HBox(children=(FloatSlider(value=1.0, description='Record Gain', max=2.0,
step=0.01), FloatSlider(value=0.1, d…
```

### 1.1.2 Save Captured Samples to a *wave* File

Before saving to a `wav` file the short capture made above, at $f_s = 44100$ Hz, was first trimmed to remove a startup recording glitch and then dead-air at the back end was removed by setting the sample length to just $N_{\text{samples}} = 100,000$ less $N_{\text{trim}} = 6000$. For Lab 2 when sampling at just $f_s = 22050$ Hz, trimming is optional, and clearly the total length will be greater that 100,000 samples, as $30 \times 22,050 = 661,500$ samples.

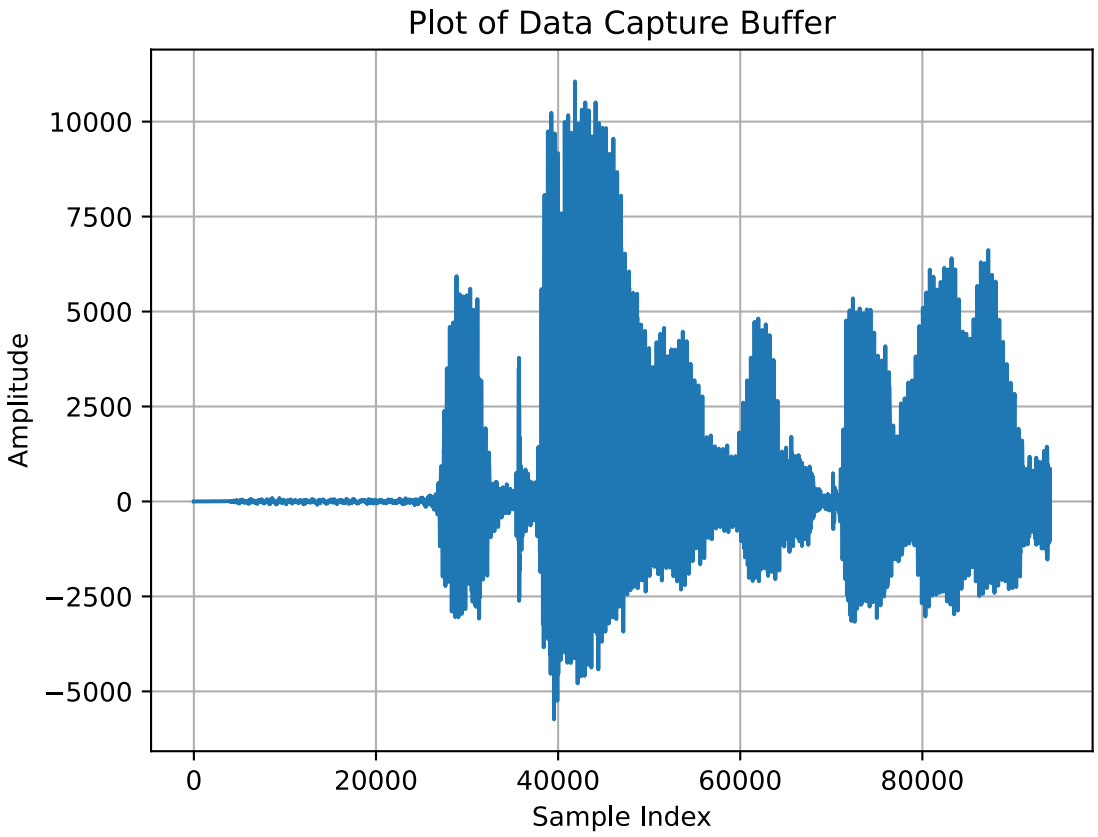
[19]:
```
# Scale values to [-1,1]
Ntrim = 6000
Nsamps = 100000
ss.to_wav('my_record.wav',44100,
          DSP_IO_1.data_capture[Ntrim:Nsamps]/max(abs(DSP_IO_1.
↪data_capture[Ntrim:Nsamps])))
```

```
# No trimming
ss.to_wav('my_record.wav',44100,
          DSP_IO_1.data_capture/max(abs(DSP_IO_1.data_capture)))
```

[9]:
```
DSP_IO_1.data_capture
```

[9]:
```
array([0., 0., 0., …, 0., 0., 0.], shape=(250880,))
```

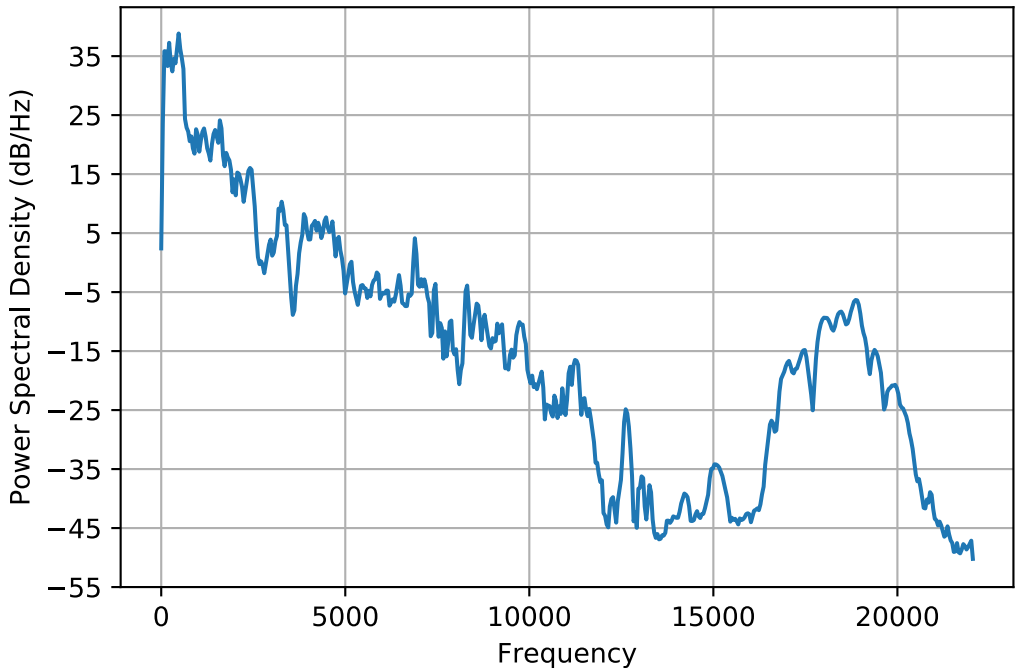
[14]:
```
# mwickert saying "Hello there, is this really working?"
# A little echo is present due to the monitor gain being 0.1. on a MacBook
Ntrim = 6000
Nsamps = 100000
plot(DSP_IO_1.data_capture[Ntrim:Nsamps])
title(r'Plot of Data Capture Buffer')
xlabel(r'Sample Index')
ylabel(r'Amplitude')
grid();
```




[15]:
```
Audio(DSP_IO_1.data_capture,rate=44100)
```

[15]:
```
<IPython.lib.display.Audio object>
```

[22]:
```
psd(DSP_IO_1.data_capture[Ntrim:Nsamps],2**10,44100);
```




### 1.1.3 Single Channel Loop Playback

Here we take the captured samples that have been saved in a wavefile and play them back in a repeating audio loop. The samples could also be manipulated first before playback.

No audio controls are used for the playback other that the standard start and stop buttons.

[17]:
```
# define callback (3)
# Here we configure the callback to play back a wav file
def callback2(in_data, frame_count, time_info, status):
    global DSP_IO, x_loop
    DSP_IO.DSP_callback_tic()

    # convert byte data to ndarray
    #in_data_nda = np.frombuffer(in_data, dtype=np.int16)
    # Ignore in_data when generating output only
    #*********************************************
    #x = in_data_nda.astype(float32)
    #y = Record_gain.value*x
    # Note wav is scaled to [-1,1] so need to rescale to int16
    y = 32767*x_loop.get_samples(frame_count)
```

```
    # Perform real-time DSP here if desired
    #
    ##########################################
    # Save data for later analysis
    # accumulate a new frame of samples
    DSP_IO.DSP_capture_add_samples(y)
    ##########################################
    # Convert from float back to int16
    y = y.astype(int16)
    DSP_IO.DSP_callback_toc()
    return y.tobytes(), pah.pyaudio.paContinue
```

In the below cell before the `DSP_IO` object is created we first load the `wav` file into a `loop_audio` object and then pass that object, `x_loop`, into the above callback function, `callback2`. This then allows infinite looping of the original `wav` file to output device 5.

[20]:
```
fs, x_rec = ss.from_wav('my_record.wav')
x_loop = pah.loop_audio(x_rec)
DSP_IO = pah.DSP_io_stream(callback2,0,1,fs=44100,Tcapture=0)
DSP_IO.interactive_stream(Tsec=0)
```

```
interactive(children=(ToggleButtons(description=' ', index=1, options=('Start␣
↪Streaming', 'Stop Streaming'), v…
```

### 1.1.4 Stereo Gain Sliders with a Stereo Audio Loop

In this example two audio channels are employed (stereo) to loop the `wav` file `Music_test.wav`. Two slider widgets are used to independently control the gain of the left and right audio paths sent to output device 5.

[4]:
```
L_gain = widgets.FloatSlider(description = 'L Gain',
                 continuous_update = True,
                 value = 1.0,
                 min = 0.0,
                 max = 2.0,
                 step = 0.01,
                 orientation = 'vertical')
R_gain = widgets.FloatSlider(description = 'R Gain',
                 continuous_update = True,
                 value = 1.0,
                 min = 0.0,
                 max = 2.0,
                 step = 0.01,
                 orientation = 'vertical')

#widgets.HBox([L_gain, R_gain])
```

[5]:
```
# L and Right Gain Sliders
def callback(in_data, frame_count, time_info, status):
    global DSP_IO_3, L_gain, R_gain, x_loop_stereo
    DSP_IO_3.DSP_callback_tic()
    # convert byte data to ndarray
    in_data_nda = np.frombuffer(in_data, dtype=np.int16)
    # separate left and right data
    #x_left,x_right = DSP_IO.get_LR(in_data_nda.astype(float32))
    # Use a loop object as a source of stereo samples
    # Note since wave files are scaled to [-1,1] we scaling to
    # rescale to the dynamic range of int16
    new_frame = x_loop_stereo.get_samples(frame_count)
    x_left = 20000*new_frame[:,0]
    x_right = 20000*new_frame[:,1]
    #***********************************************
    # DSP operations here
    y_left = x_left*L_gain.value
    y_right = x_right*R_gain.value

    #***********************************************
    # Pack left and right data together
    y = DSP_IO_3.pack_LR(y_left,y_right)
    # Typically more DSP code here
    #***********************************************
    # Save data for later analysis
    # accumulate a new frame of samples
    DSP_IO_3.DSP_capture_add_samples_stereo(y_left,y_right)
    #***********************************************
    # Convert from float back to int16
    y = y.astype(int16)
    DSP_IO_3.DSP_callback_toc()
    # Convert ndarray back to bytes
    #return (in_data_nda.tobytes(), pyaudio.paContinue)
    return y.tobytes(), pah.pyaudio.paContinue
```

[16]:
```
fs, x_in_stereo = ss.from_wav('Music_Test.wav')
x_loop_stereo = pah.loop_audio(x_in_stereo)
DSP_IO_3 = pah.DSP_io_stream(callback,0,2,fs=44100,Tcapture=0)
DSP_IO_3.interactive_stream(0,2)
widgets.HBox([L_gain, R_gain])
```

```
interactive(children=(ToggleButtons(description=' ', index=1, options=('Start␣
↪Streaming', 'Stop Streaming'), v…
```

[16]:
```
HBox(children=(FloatSlider(value=1.01, description='L Gain', max=2.0,
orientation='vertical', step=0.01), Floa…
```

### 1.1.5 Real-Time DSP

To implement one or two channel real-time DSP in the Jupyter notebook, see the 2018 Scipy Conference paper Real-Time Digital Signal Processing Using pyaudio_helper and the ipywidgets.

## 1.2 2.0 Using IPython.display.Audio to Stream Wavefiles

In this subsection we provides examples of how to use the `Audio` function that is imported from `Ipython.display` at the top of the notebook. The module `sk_dsp_comm.sigsys`, i.e.,

```
import sk_dsp_comm.sigsys as ss
```

is also useful as it contains the functions

```
fs, x = ss.from_wave('file.wav')
```

and

```
ss.to_wave('file_name',fs,x)
```

for importing and exporting wav files to the Python workspace.

### 1.2.1 Using the Test File *zero.wav*

Included in the ZIP of this notebook is a stereo test file of a voice staying the word *zero*. We use that file in a few experiments below.

[3]:
```
fs,x = ss.from_wav('zero.wav')
```

[16]:
```
Audio('zero.wav')
```

[16]:
```
<IPython.lib.display.Audio object>
```

From the `Audio` function API we can also feed it with one or two numpy `ndarrays`. The `zero.wav` test file is stereo, so `x` is a 2D array containing two columns. We must first reduce the 2D array to a 1D array using slicing.

[20]:
```
x.shape
```

[20]:
```
(56568, 2)
```

- Direct mono playback:

[14]:
```
Audio(x[:,0],rate=fs) # play the left channel
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[14], line 1
----> 1 Audio(x[:,0],rate=fs) # play the left channel

NameError: name 'x' is not defined
```

- Direct stereo playback:

[15]:
```
Audio([x[:,0],x[:,1]],rate=fs) # play the left and right channels
```

[15]:
```
<IPython.lib.display.Audio object>
```

- Take a look at the waveform and its scaling relative to the normalized wavefile amplitude range of $[-1, 1]$

[4]:
```
# mwickert saying the word "Zero"
t = arange(len(x))/float(fs)
plot(t,x[:,0])
xlabel('Time (s)')
ylabel('Amplitude')
grid();
```