pyrecplay_mulawquantizationblock

January 25, 2017

1 This program shows the working of Mu-Law Quantization on a recording and how the quantization works better than uniform quantizers.

1.0.1 Input:

As the program runs, it records through the selected input(microphone) for 8 seconds.

1.0.2 Output:

Output is Mu-Law quantized version of the recording. (Observe that the low sound intensitites(smaller amplitudes) are also quantized which wasn't in the case of Mid-Tread Quantization(as smaller amplitudes are rounded off to zero).

Import the relevant modules.

```
In [1]: """
        PyAudio Example: Make a quantization between input and output (i.e., record a
        few samples, quatize them with a mid-tread or mid-rise quantizer, and play them back imm
        Using block-wise processing instead of a for loop
        Gerald Schuller, Octtober 2014
        """

        import pyaudio
        import struct
        #import math
        #import array
        import numpy as np
        #import scipy
```

Define the variables.

```
In [2]: CHUNK = 5000 #Blocksize
        WIDTH = 2 #2 bytes per sample
        CHANNELS = 1 #2
        RATE = 32000  #Sampling Rate in Hz
        RECORD_SECONDS = 8
```

Initialize the sound card.

```
In [4]: p = pyaudio.PyAudio()

        stream = p.open(format=p.get_format_from_width(WIDTH),
                        channels=CHANNELS,
                        rate=RATE,
                        input=True,
                        output=True,
                        #input_device_index=10,
                        frames_per_buffer=CHUNK)
```

```
In [5]: print("* recording")

        #Loop for the blocks:
        for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
            #Reading from audio input stream into data with block length "CHUNK":
            data = stream.read(CHUNK)
            #Convert from stream of bytes to a list of short integers (2 bytes here) in "samples
            #shorts = (struct.unpack( "128h", data ))
            shorts = (struct.unpack( 'h' * CHUNK, data ));
            samples=np.array(list(shorts),dtype=float);

            #start block-wise signal processing:

            ###mu-Law compression:###
            y=np.sign(samples)*(np.log(1+255*np.abs(samples/32768.0)))/np.log(256);

            ####Quantization, ####
            #16 steps for normalized range -1<=x<=1
            q=2.0/16.0;

            #Mid Tread quantization:
            indices=np.round(y/q)
            #Mid -Rise quantizer:
            #indices=np.floor(y/q)

            #### De-Quantization: #####
            #Mit-Tread:
            yrek=indices*q;
            #Mid -Rise quantizer:
            #yrek=indices*q+q/2;

            #no quantization:
            #yrek=y

            #### mu-law expanding function: ###
            #we use: exp(log(256)*yrek)=256^yrek
            samples=np.sign(yrek)*(np.exp(np.log(256)*np.abs(yrek))-1)/255*32768.0
```

```
    #end signal processing
    samples=np.clip(samples,-32000,32000)
    #converting from short integers to a stream of bytes in "data":
    data=struct.pack('h' * len(samples), *samples);
    #Writing data back to audio output stream:
    stream.write(data, CHUNK)

print("* done")

stream.stop_stream()
stream.close()

p.terminate()
```

```
* recording


c:\python27\lib\site-packages\ipykernel\__main__.py:41: DeprecationWarning: integer argument exp


* done
```