IOMethods

January 28, 2017

1 IOMethods

1.1 Importing relevant Modules

```
In [ ]: import os, subprocess, csv
        import numpy as np
        import wave as _wave
        from scipy.io.wavfile import write, read
        from sys import platform
```

1.2 Class for handling audio input/output operations. It supports reading and writing of various audio formats via 'audioRead' & 'audioWrite' methods. Moreover playback can be performed by using 'sound' method. For formats different than '.wav' a coder is needed. In this case libffmpeg is being used, where the absolute path of the static build should be given to the class variable. Finally, energy normalisation and anti-clipping methods are also covered in the last two methods.

1.3 Basic Usage examples:

```
Import the class :
import IOMethods as IO
-For loading wav files:
    x, fs = IO.AudioIO.wavRead('myWavFile.wav', mono = True)
-In case that compressed files are about to be read specify
    the path to the libffmpeg library by changing the 'pathToffmpeg'
    variable and then type:
    x, fs = IO.AudioIO.audioRead()
-For writing wav files:
    IO.AudioIO.audioWrite(x, fs, 16, 'myNewWavFile.wav', 'wav')

-For listening wav files:
    IO.AudioIO.sound(x,fs)
```

- Normalisation parameters for wavreading and writing

```
In [ ]: class AudioIO:
            """ Class for handling audio input/output operations.
```

```python
    It supports reading and writing of various audio formats
    via 'audioRead' & 'audioWrite' methods. Moreover playback
    can be performed by using 'sound' method. For formats
    different than '.wav' a coder is needed. In this case
    libffmpeg is being used, where the absolute path of
    the static build should be given to the class variable.
    Finally, energy normalisation and anti-clipping methods
    are also covered in the last two methods.

    Basic Usage examples:
    Import the class :
    import IOMethods as IO
    -For loading wav files:
        x, fs = IO.AudioIO.wavRead('myWavFile.wav', mono = True)
    -In case that compressed files are about to be read specify
        the path to the libffmpeg library by changing the 'pathToffmpeg'
        variable and then type:
        x, fs = IO.AudioIO.audioRead()
    -For writing wav files:
        IO.AudioIO.audioWrite(x, fs, 16, 'myNewWavFile.wav', 'wav')

    -For listening wav files:
        IO.AudioIO.sound(x,fs)

"""
# Normalisation parameters for wavreading and writing
normFact = {'int8' : (2**7) -1,
            'int16': (2**15)-1,
            'int24': (2**23)-1,
            'int32': (2**31)-1,
            'int64': (2**63)-1,
            'float32': 1.0,
            'float64': 1.0}

# 'Silence' the bash output
FNULL = open(os.devnull, 'w')

# Absolute path needed here
pathToffmpeg = '/home/mis/Documents/Python/Projects/SourceSeparation/MiscFiles'

def __init__(self):
    pass
```

1.4 Function to load audio files such as *.mp3, *.au, *.wma & *.aiff. It first converts them to .wav and reads them with the methods below.Currently, it uses a static build of ffmpeg.

In [ ]:
```
@staticmethod
def audioRead(fileName, mono=False, startSec=None, endSec=None):
    """ Function to load audio files such as *.mp3, *.au, *.wma & *.aiff.
        It first converts them to .wav and reads them with the methods below.
        Currently, it uses a static build of ffmpeg.

    Args:
        fileName:       (str)       Absolute filename of WAV file
        mono:           (bool)      Switch if samples should be converted to mono
        startSec:       (float)     Segment start time in seconds (if None, segment
        endSec:         (float)     Segment end time in seconds (if None, segment en
    Returns:
        samples:        (np array)  Audio samples (between [-1,1]
                                    (if stereo: numSamples x numChannels,
                                    if mono: numSamples)
        sampleRate:     (float):    Sampling frequency [Hz]
    """

    # Get the absolute path
    fileName = os.path.abspath(fileName)

    # Linux
    if (platform == "linux") or (platform == "linux2"):
        convDict = {
            'mp3':[os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux')
                + ' -i ' + fileName + ' ', -3],
            'au': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux')
                 + ' -i ' + fileName + ' ', -2],
            'wma':[os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux')
                 + ' -i ' + fileName + ' ', -3],
            'aiff': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux')
                 + ' -i ' + fileName + ' ', -4]
                    }

    # MacOSX
    elif (platform == "darwin"):
        convDict = {
            'mp3':[os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx')
                + ' -i ' + fileName + ' ', -3],
            'au': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx')
                 + ' -i ' + fileName + ' ', -2],
            'wma':[os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx')
                 + ' -i ' + fileName + ' ', -3],
            'aiff': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx')
```

```
                + ' -i ' + fileName + ' ', -4]
                  }
    # Add windows support!
    else :
        raise Exception('This OS is not supported.')

    # Construct

    if fileName[convDict['mp3'][1]:] == 'mp3':
        print(fileName[convDict['mp3'][1]:])
        modfileName = os.path.join(os.path.abspath(fileName[:convDict['mp3'][1]] + '
        subprocess.call(convDict['mp3'][0]+modfileName, shell = True,  stdout=AudioI
        samples, sampleRate = AudioIO.wavRead(modfileName, mono, startSec, endSec)
        os.remove(modfileName)

    elif fileName[convDict['au'][1]:] == 'au':
        print(fileName[convDict['au'][1]:])
        modfileName = os.path.join(os.path.abspath(fileName[:convDict['au'][1]] + 'w
        subprocess.call(convDict['au'][0]+modfileName, shell = True,  stdout=AudioIO
        samples, sampleRate = AudioIO.wavRead(modfileName, mono, startSec, endSec)
        os.remove(modfileName)

    elif fileName[convDict['wma'][1]:] == 'wma':
        print(fileName[convDict['wma'][1]:])
        modfileName = os.path.join(os.path.abspath(fileName[:convDict['wma'][1]] + '
        subprocess.call(convDict['wma'][0]+modfileName, shell = True,  stdout=AudioI
        samples, sampleRate = AudioIO.wavRead(modfileName, mono, startSec, endSec)
        os.remove(modfileName)

    elif fileName[convDict['aiff'][1]:] == 'aiff':
        print(fileName[convDict['aiff'][1]:])
        modfileName = os.path.join(os.path.abspath(fileName[:convDict['aiff'][1]] +
        subprocess.call(convDict['aiff'][0]+modfileName, shell = True,  stdout=Audio
        samples, sampleRate = AudioIO.wavRead(modfileName, mono, startSec, endSec)
        os.remove(modfileName)

    else :
        raise Exception('This format is not supported.')

    return samples, sampleRate
```

1.5 Write samples to WAV file and then converts to selected format using ffmpeg.

In [ ]:

```
@staticmethod
def audioWrite(y, fs, nbits, audioFile, format):
    """ Write samples to WAV file and then converts to selected
    format using ffmpeg.
    Args:
```

```
    samples:         (ndarray / 2D ndarray) (floating point) sample vector
                    mono: DIM: nSamples
                    stereo: DIM: nSamples x nChannels

    fs:                 (int) Sample rate in Hz
    nBits:                 (int) Number of bits
    audioFile:         (string) WAV file name to write
    format:                (string) Selected format
                    'mp3'          : Writes to .mp3
                    'wma'          : Writes to .wma
                    'wav'          : Writes to .wav
                    'aiff'         : Writes to .aiff
                    'au'         : Writes to .au
"""

# Linux
if (platform == "linux") or (platform == "linux2"):
    convDict = {
        'mp3':  [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux') + ' -i ', -3
        'au':   [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux') + ' -i ', -2
        'wma':  [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux') + ' -i ', -3
        'aiff': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_linux') + ' -i ', -4
                }

# MacOSX
elif (platform == "darwin"):
    convDict = {
        'mp3':  [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx') + ' -i ', -3],
        'au':   [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx') + ' -i ', -2],
        'wma':  [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx') + ' -i ', -3],
        'aiff': [os.path.join(AudioIO.pathToffmpeg, 'ffmpeg_osx') + ' -i ', -4]
                }

# Add windows support!
else :
    raise Exception('This OS is not supported.')

if (format == 'mp3'):
    prmfileName = os.path.join(os.path.abspath(audioFile[:convDict['mp3'][1]] +
    AudioIO.wavWrite(y, fs, nbits, prmfileName)
    subprocess.call(convDict['mp3'][0] + prmfileName + ' ' + audioFile,
                    shell = True,  stdout=AudioIO.FNULL, stderr=subprocess.STDOU
    os.remove(prmfileName)

elif (format == 'wav'):
    AudioIO.wavWrite(y, fs, nbits, audioFile)

elif (format == 'wma'):
```

```
            prmfileName = os.path.join(os.path.abspath(audioFile[:convDict['wma'][1]] +
            AudioIO.wavWrite(y, fs, nbits, prmfileName)
            subprocess.call(convDict['wma'][0] + prmfileName + ' ' + audioFile,
                            shell = True,  stdout=AudioIO.FNULL, stderr=subprocess.STDOU
            os.remove(prmfileName)

        elif (format == 'aiff'):
            prmfileName = os.path.join(os.path.abspath(audioFile[:convDict['aiff'][1]] +
            AudioIO.wavWrite(y, fs, nbits, prmfileName)
            subprocess.call(convDict['aiff'][0] + prmfileName + ' ' + audioFile,
                            shell = True,  stdout=AudioIO.FNULL, stderr=subprocess.STDOU
            os.remove(prmfileName)

        elif (format == 'au'):
            prmfileName = os.path.join(os.path.abspath(audioFile[:convDict['au'][1]] + '
            AudioIO.wavWrite(y, fs, nbits, prmfileName)
            subprocess.call(convDict['au'][0] + prmfileName + ' ' + audioFile,
                            shell = True,  stdout=AudioIO.FNULL, stderr=subprocess.STDOU
            os.remove(prmfileName)
        else :
            raise Exception('This format is not supported.')
```

1.6 Function to load WAV file.

In [ ]:

```
    @staticmethod
    def wavRead(fileName, mono=False, startSec=None, endSec=None):
        """ Function to load WAV file.

        Args:
            fileName:       (str)        Absolute filename of WAV file
            mono:           (bool)       Switch if samples should be converted to mono
            startSec:       (float)      Segment start time in seconds (if None, segment
            endSec:         (float)      Segment end time in seconds (if None, segment en
        Returns:
            samples:        (np array)   Audio samples (between [-1,1]
                                         (if stereo: numSamples x numChannels,
                                         if mono: numSamples)
            sampleRate:     (float):     Sampling frequency [Hz]
        """
        try:
            samples, sampleRate = AudioIO._loadWAVWithWave(fileName)
            sWidth = _wave.open(fileName).getsampwidth()
            if sWidth == 1:
                #print('8bit case')
                samples = samples.astype(float) / AudioIO.normFact['int8'] - 1.0
            elif sWidth == 2:
                #print('16bit case')
                samples = samples.astype(float) / AudioIO.normFact['int16']
```

```
            elif sWidth == 3:
                #print('24bit case')
                samples = samples.astype(float) / AudioIO.normFact['int24']
        except:
            #print('32bit case')
            samples, sampleRate = AudioIO._loadWAVWithScipy(fileName)

        # mono conversion
        if mono:
            if samples.ndim == 2 and samples.shape[1] > 1:
                samples = (samples[:, 0] + samples[:, 1])*0.5

        # segment selection
        songLenSamples = samples.shape[0]
        if startSec is None:
            startIdx = 0
        else:
            startIdx = int(round(startSec*sampleRate))
        if endSec is None:
            endIdx = songLenSamples-1
        else:
            endIdx = int(round(endSec*sampleRate))
        if startIdx < 0 or startIdx > songLenSamples:
            raise Exception("Segment start sample index out of song boundaries!")
        if endIdx < startIdx or endIdx > songLenSamples:
            raise Exception("Segment end sample index out of song boundaries!")
        if samples.ndim == 1:
            samples = samples[startIdx:endIdx]
        else:
            samples = samples[startIdx:endIdx, :]

        return samples, sampleRate
```

1.7 Load samples & sample rate from 24 bit WAV file.(Using Wave module of Python)

In [ ]:
```
    @staticmethod
    def _loadWAVWithWave(fileName):
        """ Load samples & sample rate from 24 bit WAV file """
        wav = _wave.open(fileName)
        rate = wav.getframerate()
        nchannels = wav.getnchannels()
        sampwidth = wav.getsampwidth()
        nframes = wav.getnframes()
        data = wav.readframes(nframes)
        wav.close()
        array = AudioIO._wav2array(nchannels, sampwidth, data)

        return array, rate
```

1.8 Load samples & sample rate from WAV file.(Using Scipy module)

In [ ]:

```
@staticmethod
def _loadWAVWithScipy(fileName):
    """ Load samples & sample rate from WAV file """
    inputData = read(fileName)
    samples = inputData[1]
    sampleRate = inputData[0]

    return samples, sampleRate
```

1.9 Function to convert the content of wave file which is bytes to string format

In [ ]:

```
@staticmethod
def _wav2array(nchannels, sampwidth, data):
    """data must be the string containing the bytes from the wav file."""
    num_samples, remainder = divmod(len(data), sampwidth * nchannels)
    if remainder > 0:
        raise ValueError('The length of data is not a multiple of '
                         'sampwidth * num_channels.')
    if sampwidth > 4:
        raise ValueError("sampwidth must not be greater than 4.")

    if sampwidth == 3:
        a = np.empty((num_samples, nchannels, 4), dtype = np.uint8)
        raw_bytes = np.fromstring(data, dtype = np.uint8)
        a[:, :, :sampwidth] = raw_bytes.reshape(-1, nchannels, sampwidth)
        a[:, :, sampwidth:] = (a[:, :, sampwidth - 1:sampwidth] >> 7) * 255
        result = a.view('<i4').reshape(a.shape[:-1])
    else:
        # 8 bit samples are stored as unsigned ints; others as signed ints.
        dt_char = 'u' if sampwidth == 1 else 'i'
        a = np.fromstring(data, dtype='<%s%d' % (dt_char, sampwidth))
        result = a.reshape(-1, nchannels)
    return result
```

1.10 Write samples to WAV file

In [ ]:

```
@staticmethod
def wavWrite(y, fs, nbits, audioFile):
    """ Write samples to WAV file
    Args:
        samples: (ndarray / 2D ndarray) (floating point) sample vector
                    mono: DIM: nSamples
                    stereo: DIM: nSamples x nChannels

        fs:         (int) Sample rate in Hz
        nBits:          (int) Number of bits
```

```
        fnWAV:          (string) WAV file name to write
    """
    if nbits == 8:
        intsamples = (y+1.0) * AudioIO.normFact['int' + str(nbits)]
        fX = np.int8(intsamples)
    elif nbits == 16:
        intsamples = y * AudioIO.normFact['int' + str(nbits)]
        fX = np.int16(intsamples)
    elif nbits > 16:
        fX = y

    write(audioFile, fs, fX)
```

1.11 Plays a wave file using the pyglet library. But first, it has to be written. Termination of the playback is being performed by any keyboard input and the press 'Enter'.

In [ ]:

```
@staticmethod
def sound(x,fs):
    """ Plays a wave file using the pyglet library. But first, it has to be written.
        Termination of the playback is being performed by any keyboard input and Ent
        Args:
        x:                      (array) Floating point samples
        fs:                     (int) The sampling rate
    """
    import pyglet as pg
    global player
    # Call the writing function
    AudioIO.wavWrite(x, fs, 16, 'testPlayback.wav')
    # Initialize playback engine
    player = pg.media.Player()
    # Initialize the object with the audio file
    playback = pg.media.load('testPlayback.wav')
    # Set it to player
    player.queue(playback)
    # Sound call
    player.play()
    # Killed by "keyboard"
    kill = raw_input()
    if kill or kill == '':
        AudioIO.stop()
    # Remove the dummy wave write
    os.remove('testPlayback.wav')
```

1.12 Stops a playback object of the pyglet library. It does not accept arguments, but a player has to be already initialized by the above "sound" method.

In [ ]:
```
    @staticmethod
    def stop():
        """ Stops a playback object of the pyglet library.
            It does not accept arguments, but a player has to be
            already initialized by the above "sound" method.
        """
        global player
        # Just Pause & Destruct
        player.pause()
        player = None
        return None
```

1.13 Function to perform energy normalisation of two audio signals, based on envelopes acquired by Hilbert transformation.

In [ ]:
```
    @staticmethod
    def energyNormalisation(x1, x2, wsz = 1024):
        """ Function to perform energy normalisation of two audio signals,
        based on envelopes acquired by Hilbert transformation.

        Args:
            x1          :    (np array)      Absolute filename of WAV file
            x2          :    (np array)      Switch if samples should be converted to mono
            wsz :         (int)                       Number of samples to take into acc
                                computation of the analytic function. If set
                                to zero the whole signal will be analysed
                                at once.
        Returns:
            y1          :          (np array)            Energy normalised output signal
            y2          :          (np array)            Energy normalised output signal
        """
        x1.shape = (len(x1), 1)
        x2.shape = (len(x2), 1)

        if wsz == 0:
            xa1 = AF.HilbertTransformation(x1, mode = 'global', wsz = wsz)
            xa2 = AF.HilbertTransformation(x2, mode = 'global', wsz = wsz)

            energy1 = np.mean(np.abs(xa1) ** 2.0)
            energy2 = np.mean(np.abs(xa2) ** 2.0)

            if energy1 > energy2:
                rt = energy1/energy2
                y2 = x2 * rt
                y1 = x1
```

```
        else :
            rt = energy2/energy1
            y1 = x1 * rt
            y2 = x2

        y1 = AudioIO.twoSideClip(y1, -1.0, 1.0)
        y2 = AudioIO.twoSideClip(y2, -1.0, 1.0)

    else:

        if len(x1) > len(x2):
            x1 = np.append(x1, np.zeros(len(x1)%wsz))
            x2 = np.append(x2, np.zeros(len(x1) - len(x2)))
        else:
            x2 = np.append(x2, np.zeros(len(x2)%wsz))
            x1 = np.append(x1, np.zeros(len(x2) - len(x1)))

        xa1 = AF.HilbertTransformation(x1, mode = 'local', wsz = wsz)
        xa2 = AF.HilbertTransformation(x2, mode = 'local', wsz = wsz)

        y1 = np.empty(len(x1))
        y2 = np.empty(len(x2))

        energy1 = np.abs(xa1)
        energy2 = np.abs(xa2)

        pin = 0
        pend = len(x1) - wsz

        while pin <= pend :

            lclE1 = np.mean(energy1[pin : pin + wsz])
            lclE2 = np.mean(energy2[pin : pin + wsz])

            if (lclE1 > lclE2) and (lclE1 > 1e-4) and (lclE2 > 1e-4):
                rt = lclE1/lclE2
                bufferY2 = x2[pin : pin + wsz] * rt
                bufferY1 = x1[pin : pin + wsz]

            elif (lclE1 < lclE2) and (lclE1 > 1e-4) and (lclE2 > 1e-4):
                rt = lclE2/lclE1
                bufferY1 = x1[pin : pin + wsz] * rt
                bufferY2 = x2[pin : pin + wsz]

            else:
                bufferY1 = x1[pin : pin + wsz]
                bufferY2 = x2[pin : pin + wsz]
```

```
                y1[pin : pin + wsz] = AudioIO.twoSideClip(bufferY1, -1.0, 1.0)
                y2[pin : pin + wsz] = AudioIO.twoSideClip(bufferY2, -1.0, 1.0)

                pin += wsz

        y1.shape = (len(y1),1)
        y2.shape = (len(y2),1)
        return y1, y2
```

1.14 Method to limit an input array inside a given range.

```
In [ ]:     @staticmethod
            def twoSideClip(x, minimum, maximum):
                """ Method to limit an input array inside a given
                    range.
                Args:
                    x                 :       (np array)      Input array to be limited
                    minimum         :       (int)                     Minimum value to be consid
                    maximum :                 (int)                       Maximum value to be co

                Returns:
                    x                 :                 (np array)            Limited output array
                """

                for indx in range(len(x)):
                    if x[indx] < minimum:
                        x[indx] = minimum
                    elif x[indx] > maximum:
                        x[indx] = maximum

                return x
```

1.15 Testing the IOMethods

```
In [ ]: if __name__ == "__main__":
            # Define File
            myReadFile = 'EnterYourWavFile.wav'
            # Read the file
            x, fs = AudioIO.wavRead(myReadFile, mono = True)
            # Gain parameter
            g = 0.5
            # Listen to it
            AudioIO.sound(x*g,fs)
            # Make it better and write it to disk
            x2 = np.empty((len(x),2), dtype = np.float32)
            try :
                x2[:,0] = x * g
                x2[:,1] = np.roll(x*g, 512)
```

```python
except ValueError:
    x2[:,0] = x[:,0] * g
    x2[:,1] = np.roll(x[:,0] * g, 256)
# Listen to stereo processed
AudioIO.sound(x2*g,fs)
AudioIO.audioWrite(x2, fs, 16, 'myNewWavFile.wav', 'wav')
```