MAE 298 – Homework 1
Computation of Sound Pressure Level and Octave Band Spectrum

Logan Halstrom
PhD Graduate Student Researcher
Center for Human/Robot/Vehicle Integration and Performance
Department of Mechanical and Aerospace Engineering
University of California, Davis
Email: ldhalstrom@ucdavis.edu

1 Background

This assignment details the process of analyzing a recorded sound signal and computing the Sound Pressure Level (SPL) in Decibels (dB) accross the narrow, 1/3 Octave, and Octave-bands. The source signal is a recording of a sonic boom contained in the included data file 'Boom_F1B2_6.wav'.

All computations and plotting for this project were performed using Python, and the source code is attached in the Appendix. All of the primary data processing code is contained in the file 'hw1_00_processing.py' and the plotting code is contained in the file 'hw1_01_plotting.py', which is suplimented by the custom plotting package 'lplot.py'.

2 Problem 1 – Signal Processing

The pressure signal input file 'Boom_F1B2_6.wav' was read using the 'PySoundFile' audio library for Python, which can be found at: `https://pypi.python.org/pypi/SoundFile/`. This library was chosen over other Python audio libraries (e.g. 'scipy.io.wavfile') because it normalized the .wav file data between -1.0 and 1.0 identically to MATLAB's 'audioread' function. (Note: Though other libraries did not read in the same values, all libraries returned the same results when normalized by the maximum value of the data).

2.1 Problem 1.1 – Sonic Boom Pressure Signal

After reading the raw data from the .wav file, signal values were converted from units of Volts (V) to Pascals (Pa) using the conversion ratio obtained from Eqn 1:

$$-116Pa = 1V \tag{1}$$

The resulting pressure history of the sonic boom is displayed in Fig 1, which demonstrates the classic "N" shape of the sonic boom, where there is an initial positive pressure discontinutity caused by the supersonic front of the aircraft, followed by an almost linear decrease in pressure over the surface of the aircraft until an abrupt discontinuity back to freestream pressure occurs. This results in the observer hearing a "double-boom" for low-flying supersonic aircraft (at higher altitudes, the two shocks can merge by the time they reach the observer). After the aircraft, the pressure does not immediately return to steady-state freestream flow, but experiences a slight oscillation in pressure as the remaining wake dissipates.

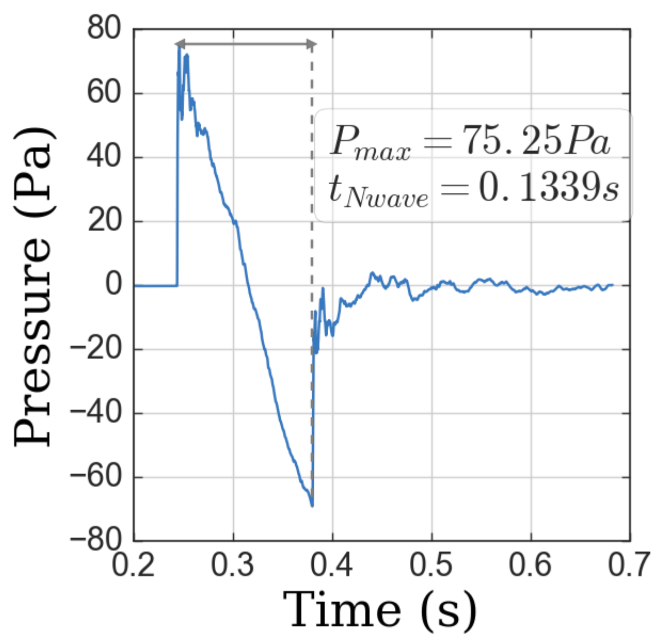

Fig. 1: Recorded sonic boom shockwave pressure time history with peak pressure and N-wave time duration values (Zero-pressure from recording start to initial shock)

Also contained in Fig 1 are the values for the peak sonic boom pressure perturbation:

$$P_{max} = \max(|P|) = \boxed{75.25Pa}$$

and sonic boom N-wave duration:

$$t_{Nwave} = \boxed{133.9ms}$$

3 Problem 1.2 – Power Spectral Density Decomposition

In order to analyze the frequency-dependent nature of the recorded sonic boom, it is nessesary to break the signal into its component frequencies, for which we employ the technique of Fast-Fourer Transform (FFT). This descrete Fourier transform algorithm transforms data from the time domain to the frequency domain, and can be called in Python from 'numpy.fft'.

The FFT is performed in Python according to the following pseudocode:

$$fft = \text{np.fft.fft}(pressure) \cdot dt$$

where *pressure* is the time-domain pressure signal and $dt = \frac{1}{f_s}$ is the time step of the discrete frequency domain, which is equal to the inverse of the sampling frequency f_s.
Next, the double-sided power spectrum S_{xx} is obtained according to Eqn 2:

$$S_{xx} = \frac{|fft|^2}{T} \tag{2}$$

where all operations are performed element-wise on the data series fft and $T = \text{len}(fft) \cdot dt$ is the total time interval of the data series.

From the double-sided power spectrum, the singe-sided power spectrum G_{xx} can be acquired as twice of the first half of S_{xx} (Eqn 3):

$$G_{xx} = 2S_{xx}[idx] \tag{3}$$

where all operations are performed element-wise on S_{xx} and *idx* is the first half of all of the indices of S_{xx}.

Now the the power spectral density G_{xx} has been computed, the corresponding single-sided frequency spectrum can be calculated with the built in numpy function 'fftfreq':

$$freqs = \text{np.fft.fftfreq}(pressure.\text{size}, dt)$$
$$freqs = freqs[idx]$$

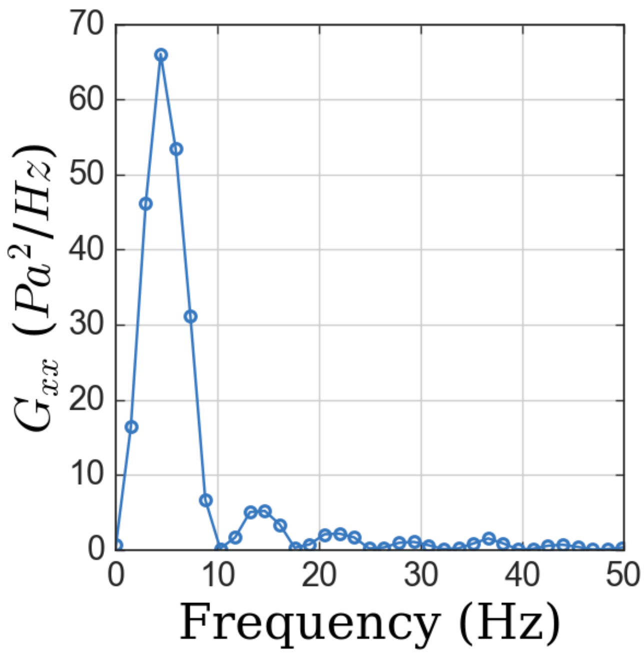

Fig. 2: Shockwave signal power spectral density as a function of frequency (All frequencies above 50Hz very low power)

The resulting power spectrum G_{xx} vs $freq$ is plotted for reference in Fig 2. From the figure, it can be seen that the most dominate frequencies occur between 1 and 50 Hz, with the maximum power spectral density of $66Pa^2/Hz$ corresponding to $4.49Hz$, and the other significant peaks (of significantly lower magnitude) occuring around 15, 23, and 37 Hz, respectively.

3.1 Problem 1.3 – Sound Pressure Level

With the power spectrum density function calculated, it is meaninful to compute the Sound Pressure Level (SPL) in dB. The formula for descrete SPL as a function of freqency is given in Eqn 4, which demonstrates that SPL is a logarithmic ratio of the signal's pressure distrubance and a reference pressure $P_{ref} = 20\mu Pa$.

$$SPL(f) = 10\log_{10}\left(\frac{G_{xx}/T}{P_{ref}^2}\right) \tag{4}$$

SPL of the narrow-band frequencies is plotted with the octave-band frequencies in Fig 3 in Section 4

4 Problem 2 – Octave-Band Spectra

In aeroacoustic analysis, it is common for recorded data sets to contain massive amounts of data that can be redundant and combersome to process. To ease this burden, originally sampled narrow-band frequency spectra can be binned into fewer, pre-selected, descrete sets of freqencies called the Third (1/3) Octave and Octave Bands.

This binning process is accomplished by summing the narrow-band frequencies over intervals defined by specific center frequencies f_c of the octave-band. Center frequencies can be chosen from a pre-selected "preferred" set of "nicer" numbers, or they can be calculated according to Eqn 5, which does not produce the "nice" numbers.

$$f_{c,m} = f_{c,30} \cdot 2^{-10+\frac{m}{3}} \tag{5}$$

where $f_{c,30} = 1000Hz$ is the center frequency corresponding to $m = 30$, and $m = 1,2,3,\ldots$ for 1/3 octave-band and $m = 3,6,9,\ldots$ for octave-band.

In the data processing script for this analysis, center frequencies are calculated specifically for the narrow-band input. The function 'OctaveCenterFreqs' will loop through m in intervals appropriate to the given octave-band choice, but only center frequencies whose lower f_l and upper f_u band limits are contained within the original data set. This ensures that no sum will take place over an incomplete band. Upper and lower band limits for a given center frequencies are calculated according to Eqn 6.

$$\begin{aligned} f_u &= 2^{\frac{octv}{2}} \cdot f_c \\ f_l &= 2^{-\frac{octv}{2}} \cdot f_c \end{aligned} \tag{6}$$

where $octv = \frac{1}{3}$ for the 1/3 octave-band and $octv = 1$ for the octave-band.

Once the center frequencies and associated band limits have been calculated, the sum in Eqn 7 must be performed for each band to determine the octave-band SPL associated with each center frequency. To calculate the 1/3 or full octave-bands from the narrow-band, simply select the center frequency bands for the desired octave and apply them in Eqn 7.

$$Lp(f_c) = 10\log_{10}\left(\sum_{f=f_{l,c}}^{f_{u,c}} 10^{\frac{Lp(f)}{10}}\right) \tag{7}$$

where $f_{l,c}, f_{u,c}$ are the bounds corresponding to the given center frequency f_c.

Finally, it can be beneficial to summarize the frequency spectrum with a single representative value called the Overall Sound Pressure Level SPL_{ovr}. This parameter is calculated in the same manner as the 1/3 and full octave-band SPL, but the sum is taken over the entire data set, rather than in bins. It is also convention not to include SPL values corresponding to less than $10Hz$, so these are ommitted in this analysis.

The results for all parts of Problem 2 are summarized below in Fig 3, and they will be individually discussed in the following sections.

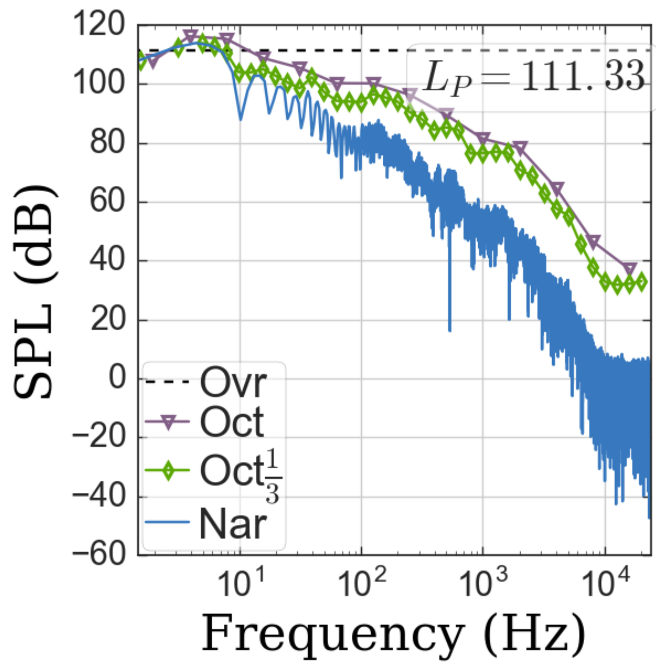

Fig. 3: Shockwave signal narrow-band (Nar), 1/3 octave-band (Oct$\frac{1}{3}$), and octave-band (Oct), with overall Sound Pressure Level (Ovr) reported in upper right (L_p)

4.1 Problem 2.1 – 1/3 Octave Bands

Results for the 1/3 octave-band are plotted as a green line with diamond markers in Fig 3 and are also tabulated in Appendix A. Compared to the narrow-band SPL (blue line), it is apparent that the binned values of the 1/3 octave-band are generally greater in magnitude than their narrow-band counterparts. This is because the octave band is calculated as a binned sum over bands of the narrow-band and therefore *must* be greater in magnitude.

The 1/3 octave-band plot follows the same trends as that of the narrow-band, but with many fewer points (43 vs. 32768, to be exact). This demonstrates the power of using the octave bands, namely in that the general trends of the frequency spectrum can be accurately with a very few amount of points.

4.2 Problem 2.2 – Octave Bands

Results for the octave-band are plotted as a purple line with triangle markers in Fig 3 and are also tabulated in Appendix A. Trends are similar to those of the 1/3 octave-band, but are slightly greater in magnitude since the summing bands are wider. Thus, the octave-band trends are less accurate at modeling the narrow-band spectrum than the 1/3 octave-band, with the advantage of having even fewer points (14 vs. 43 vs. 32768 for the octave, 1/3 octave, and narrow-bands, respectively).

4.3 Problem 2.3 – Overall Sound Pressure Level

Finally, the overall SPL of the sonic boom signal was found to be $\boxed{SPL_{ovr} = 111.33dB}$, and is represented in Fig 3 as a dashed, black line. This value is generally greater in magnitude than the majority of the points in any of the spectra, especially at high frequency. Each of the narrow, 1/3, and octave-bands peak at a higher value than SPL_{ovr}, however, because SPL_{ovr} does not include information for frequencies below $10Hz$.

5 Conclusion

In summary, this analysis has demonstrated basic signal processing techniques required to analyze aeroacoustical data. Reading raw pressure time histories and transformation into the descrete frequency domain was demonstrated in Problem 1, and conversion into the octave-bands and overall SPL was performed in Problem 2. These techniques will be required for future, more advanced aeroacoustic analyses.

Appendix A: Data Processing Script

```
"""HW1 - DATA PROCESSING
Logan Halstrom
MAE 298 AEROACOUSTICS
HOMEWORK 1 - SIGNAL PROCESSING
CREATED: 04 OCT 2016
MODIFIY: 17 OCT 2016

DESCRIPTION: Read sound file of sonic boom and convert signal to
Narrow-band in Pa.
Compute Single-side power spectral density (FFT).
1/3 octave and octave band

NOTE: use 'soundfile' module to read audio data.  This normalizes data from
-1 to 1, like Matlab's 'audioread'.  'scipy.io.wavfile' does not normalize
"""

#IMPORT GLOBAL VARIABLES
from hw1_98_globalVars import *

import numpy as np
import pandas as pd

def ReadWavNorm(filename):
    """Read a .wav file and return sampling frequency.
    Use 'soundfile' module, which normalizes audio data between -1 and 1,
    identically to MATLAB's 'audioread' function
    """
    import soundfile as sf
    #Returns sampled data and sampling frequency
    data, samplerate = sf.read(filename)
    return samplerate, data

def ReadWav(filename):
    """NOTE: NOT USED IN THIS CODE, DOES NOT NORMALIZE LIKE MATLAB
    Read a .wav file and return sampling frequency
    Use 'scipy.io.wavfile' which doesn't normalize data.
    """
    from scipy.io import wavfile
    #Returns sample frequency and sampled data
    sampFreq, snd = wavfile.read(filename)
    #snd = Normalize(snd)
    return sampFreq, snd

def Normalize(data):
    """NOTE: NOT USED IN THIS CODE, TRIED BUT FAILED TO NORMALIZE LIKE MATLAB
    Trying to normalize data between -1 and 1 like matlab audioread
    """
    data = np.array(data)
    return ( 2*(data - min(data)) / (max(data) - min(data)) - 1)

def SPLt(P, Pref=20e-6):
    """Sound Pressure Level (SPL) in dB as a function of time.
    P    --> pressure signal (Pa)
    Pref --> reference pressure
    """
    PrmsSq = 0.5 * P ** 2 #RMS pressure squared
    return 10 * np.log10(PrmsSq / Pref ** 2)

def SPLf(Gxx, T, Pref=20e-6):
    """Sound Pressure Level (SPL) in dB as a function of frequency
    Gxx  --> Power spectral density of a pressure signal (after FFT)
```

```python
    T    --> Total time interval of pressure signal
    Pref --> reference pressure
    """
    return 10 * np.log10( (Gxx / T) / Pref ** 2 )

def OctaveCenterFreqsGen(dx=3, n=39):
    """NOTE: NOT USED IN THIS CODE. INSTEAD, OCTAVECENTERFREQS
    Produce general center frequencies for octave-band spectra
    dx --> frequency interval spacing (3 for octave, 1 for 1/3 octave)
    n --> number of center freqs to product (starting at dx)
    """
    fc30 = 1000 #Preferred center freq for m=30 is 1000Hz
    m = np.arange(1, n+1) * dx #for n center freqs, multiply 1-->n by dx
    freqs = fc30 * 2 ** (-10 + m/3) #Formula for center freqs

def OctaveBounds(fc, octv=1):
    """Get upper/lower frequency bounds for given octave band.
    fc --> current center frequency
    octv --> octave-band (octave-->1, 1/3 octave-->1/3)
    """
    upper = 2 ** ( octv / 2) * fc
    lower = 2 ** (-octv / 2) * fc
    return upper, lower

def OctaveCenterFreqs(narrow, octv=1):
    """Calculate center frequencies (fc) for octave or 1/3 octave bands.
    Provide original narrow-band frequency vector to bound octave-band.
    Only return center frequencies who's lowest lower band limit or highest
    upper band limit are within the original data set.
    narrow --> original narrow-band frequencies (provides bounds for octave)
    octv --> frequency interval spacing (1 for octave, 1/3 for 1/3 octave)
    """
    fc30 = 1000 #Preferred center freq for m=30 is 1000Hz
    freqs = []
    for i in range(len(narrow)):
        #current index
        m = (3 * octv) * (i + 1) #octave, every 3rd, 1/3 octave, every 1
        fc = fc30 * 2 ** (-10 + m/3) #Formula for center freq
        fcu, fcl = OctaveBounds(fc, octv) #upper and lower bounds for fc band
        if fcu > max(narrow):
            break #quit if current fc is greater than original range
        if fcl >= min(narrow):
            freqs.append(fc) #if current fc is in original range, save
    return freqs

def OctaveLp(Lp):
    """Given a range of SPLs that are contained within a given octave band,
    perform the appropriate log-sum to determine the octave SPL
    Lp --> SPL range in octave-band
    """
    #Sum 10^(Lp/10) accross current octave-band, take log
    Lp_octv = 10 * np.log10( np.sum( 10 ** (Lp / 10) ) )
    return Lp_octv

def GetOctaveBand(df, octv=1):
    """Get SPL ( Lp(fc,m) ) for octave-band center frequencies.
    Returns octave-band center frequencies and corresponding SPLs
    df --> pandas dataframe containing narrow-band frequencies and SPL
    octv --> octave-band type (octave-->1, 1/3 octave-->1/3)
    """

    #Get Center Frequencies
```

```python
    fcs = OctaveCenterFreqs(df['freq'], octv)
    Lp_octv = np.zeros(len(fcs))
    for i, fc in enumerate(fcs):
        #Get Upper/Lower center freqency band bounds
        fcu, fcl = OctaveBounds(fc, octv)

        band = df[df['freq'] >= fcl]
        band = band[band['freq'] <= fcu]

        #SPLs in current octave-band
        Lp = np.array(band['SPL'])
        #Sum 10^(Lp/10) accross current octave-band, take log
        Lp_octv[i] = OctaveLp(Lp)

    return fcs, Lp_octv




def main(source):
    """Perform calculations for frequency data processing
    source --> file name of source sound file
    """

    ####################################################################
    ### READ SOUND FILE ################################################
    ####################################################################

    df = pd.DataFrame() #Stores signal data

    #Read source frequency (fs) and signal in volts normallized between -1&1
    fs, df['V'] = ReadWavNorm( '{}/{}'.format(datadir, source) ) #Like matlab

    #Convert to pascals
    df['Pa'] = df['V'] * volt2pasc

    ####################################################################
    ### POWER SPECTRAL DENSITY #########################################
    ####################################################################

    #TIME
    #calculate time of each signal, in seconds, from source frequency
    N = len(df['Pa']) #Number of data points in signal
    dt = 1 / fs #time step
    T = N * dt #total time interval of signal (s)
    df['time'] = np.arange(N) * dt #individual sample times
    idx = range(int(N/2)) #Indices of single-sided power spectrum (first half)

    #POWER SPECTRUM
    fft = np.fft.fft(df['Pa']) * dt #Fast-Fourier Transform
    Sxx = np.abs(fft) ** 2 / T #Two-sided power spectrum
    #Gxx = Sxx[idx] #Single-sided power spectrum
    Gxx = 2 * Sxx[idx] #Single-sided power spectrum

    #FREQUENCY
    freqs = np.fft.fftfreq(df['Pa'].size, dt) #Frequencies
    #freqs = np.arange(N) / T #Frequencies
    freqs = freqs[idx] #single-sided frequencies

    #COMBINE POWER SPECTRUM DATA INTO DATAFRAME
    powspec = pd.DataFrame({'freq' : freqs, 'Gxx' : Gxx})

```

```python
    maxima = powspec[powspec['Gxx'] == max(powspec['Gxx'])]
    print('\nMaximum Power Spectrum, frequency:t', float(maxima['freq']))
    print(  'Maximum Power Spectrum, power:',     float(maxima['Gxx' ]))

    #####################################################################
    ### FIND SOUND PRESSURE LEVEL IN dB #################################
    #####################################################################

    #SPL VS TIME
    df['SPL'] = SPLt(df['Pa'])
    #SPL VS FREQUENCY
    powspec['SPL'] = SPLf(Gxx, T)

    #####################################################################
    ### SONIC BOOM N-WAVE PEAK AND DURATION #############################
    #####################################################################

    #SONIC BOOM PRESSURE PEAK
    Pmax = max(abs(df['Pa']))

    #SONIC BOOM N-WAVE DURATION
    #Get shock starting and ending times and pressures
    shocki = df[df['Pa'] == max(df['Pa'])] #Shock start
    ti = float(shocki['time']) #start time
    Pi = float(shocki['Pa'])   #start (max) pressure
    shockf = df[df['Pa'] == min(df['Pa'])] #Shock end
    tf = float(shockf['time']) #start time
    Pf = float(shockf['Pa'])   #start (max) pressure
    #Shockwave time duration
    dt_Nwave = tf - ti

    #####################################################################
    ### OCTAVE-BAND CONVERSION ##########################################
    #####################################################################

    #1/3 OCTAVE-BAND
    octv3rd = pd.DataFrame()
    octv3rd['freq'], octv3rd['SPL'] = GetOctaveBand(powspec, octv=1/3)

    #OCTAVE-BAND
    octv = pd.DataFrame()
    octv['freq'], octv['SPL'] = GetOctaveBand(powspec, octv=1)

    #OVERALL SOUND PRESSURE LEVEL
    #Single SPL value for entire series
    #Sum over either octave or 1/3 octave bands (identical)
    #but exclude freqencies below 10Hz
    Lp_overall = OctaveLp(octv[octv['freq'] >= 10.0]['SPL'])

    print('\nNum. of Points, Narrow-band:'    , len(df))
    print(  'Num. of Points, 1/3 Octave-band:', len(octv3rd))
    print(  'Num. of Points, Octave-band:'    , len(octv))

    #####################################################################
    ### SAVE DATA #######################################################
    #####################################################################

    #SAVE WAVE SIGNAL DATA
    df = df[['time', 'Pa', 'SPL', 'V']] #reorder
    df.to_csv( '{}/timespec.dat'.format(datadir), sep=' ', index=False ) #save

    #SAVE POWER SPECTRUM DATA
```

```python
    powspec.to_csv( '{}/freqspec.dat'.format(datadir), sep=' ', index=False )

    #SAVE OCTAVE-BAND DATA
    octv3rd.to_csv( '{}/octv3rd.dat'.format(datadir), sep=' ', index=False)
    octv.to_csv( '{}/octv.dat'.format(datadir), sep=' ', index=False)

    #SAVE SINGLE PARAMETERS
    params = pd.DataFrame()
    params = params.append(pd.Series(
        {'fs' : fs, 'SPL_overall' : Lp_overall,
         'Pmax' : Pmax, 'tNwave' : dt_Nwave,
         'ti' : ti, 'Pi' : Pi, 'tf' : tf, 'Pf' : Pf}
        ), ignore_index=True)
    params.to_csv( '{}/params.dat'.format(datadir), sep=' ', index=False)


if __name__ == "__main__":


    Source = 'Boom_F1B2_6.wav'

    main(Source)
```

Listing 1: *hw1_00_process.py* - Performs all primary data processing such as pressure signal input, power spectral density decomposition, and octave-band conversion and saves data to text files