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 discontinuity 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.

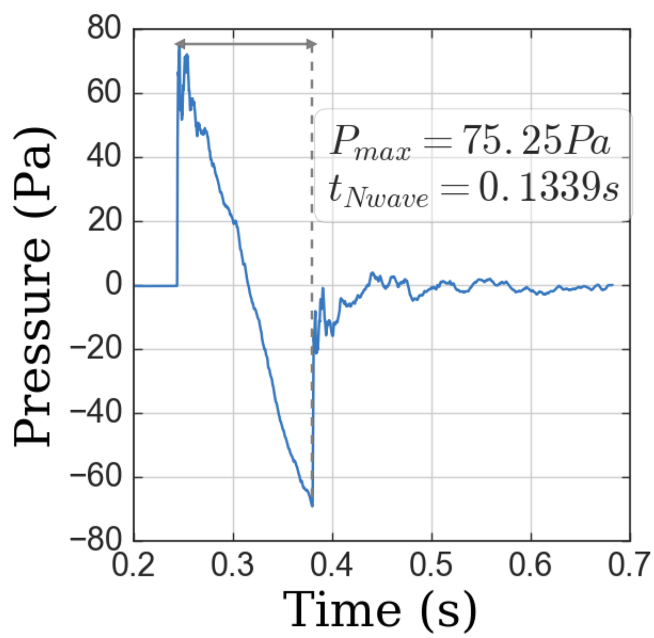

Fig. 1: Recorded sonic boom shockwave pressure time history with peak pressure and N-wave time duration values (Zero-pressure from recording start to initial shock)

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 necesary to break the signal into its component frequencies, for which we employ the technique of Fast-Fourier Transform (FFT). This discrete 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 power spectral density G_{xx} has been computed, the corresponding single-sided frequency spectrum can be calculated with the built in numpy function 'fftfreq':

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

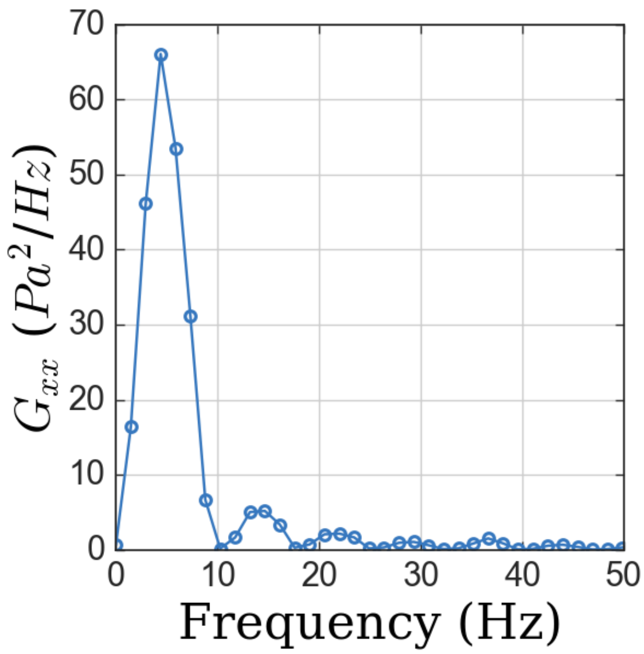

Fig. 2: Shockwave signal power spectral density as a function of frequency (All frequencies above 50Hz very low power)

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) occurring around 15, 23, and 37 Hz, respectively.

3.1 Problem 1.3 – Sound Pressure Level

With the power spectrum density function calculated, it is meaningful to compute the Sound Pressure Level (SPL) in dB. The formula for discrete SPL as a function of frequency is given in Eqn 4, which demonstrates that SPL is a logarithmic ratio of the signal's pressure disturbance 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, discrete sets of frequencies 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 omitted 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.

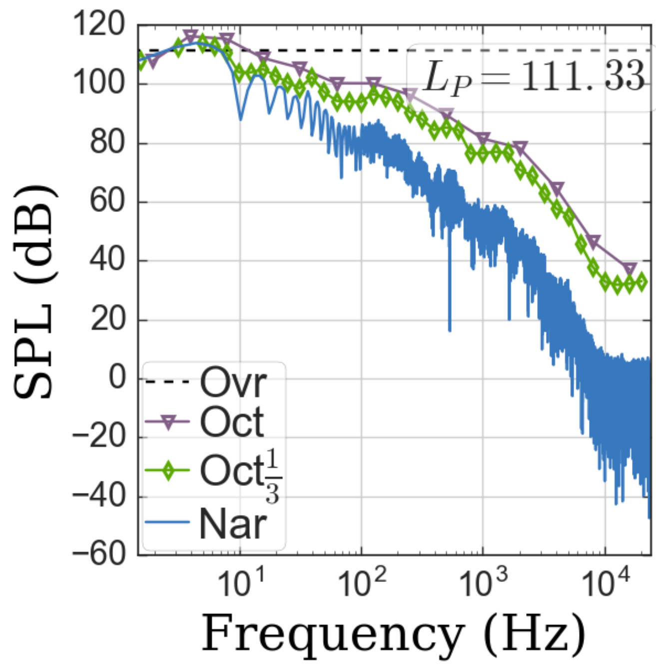

Fig. 3: Shockwave signal narrow-band (Nar), 1/3 octave-band (Oct$\frac{1}{3}$), and octave-band (Oct), with overall Sound Pressure Level (Ovr) reported in upper right (L_p)

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 $10 Hz$.

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 discrete 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: Octave-Band Data

```
freq SPL
1.953125 107.79232221931316
3.90625 116.13832007561234
7.8125 115.24738469733194
15.625 108.73126147737337
31.25 105.43628428981057
62.5 100.20098966455878
125.0 100.13721967833573
250.0 96.27463977997218
500.0 89.42249293814415
1000.0 81.33795412947121
2000.0 78.25181954951658
4000.0 64.60591443609596
8000.0 46.46688865429704
16000.0 37.07099842429648
```

Table 1: *octv.dat* - Octave-band center frequencies and sound pressure levels (space-separated)

```
freq SPL
1.2303916502879626 -inf
1.5501963398126943 107.79232221931316
1.953125 -inf
2.460783300575925 -inf
3.1003926796253887 112.28088546977946
3.90625 -inf
4.92156660115185 113.83657680640881
6.200785359250777 112.92024617089574
7.8125 110.56392376435218
9.8431332023037 103.97986159057722
12.401570718501555 103.86832513214438
15.625 104.95290674684567
19.68626640460739 102.79122106668738
24.803141437003124 100.48148679363506
31.25 98.47652078196909
39.37253280921478 102.24036717858095
49.60628287400625 97.266336131359
62.5 94.2318123086147
78.74506561842956 93.9710795617455
99.2125657480125 93.81727083347613
125.0 96.4857045687097
157.49013123685913 95.3913335241595
198.425131496025 93.97769306668212
250.0 90.44518784661584
314.9802624737184 88.02085164002017
396.85026299204975 84.40136340242056
500.0 85.15527300078429
629.9605249474369 84.34949464954664
793.7005259840995 76.3217254612694
1000.0 76.47066347106642
1259.9210498948737 76.88782990167843
1587.401051968199 76.70936677474515
2000.0 70.88801322015367
2519.8420997897474 68.87567649715912
3174.802103936398 62.933466530779675
4000.0 57.71119190837549
5039.684199579495 55.22014500340504
6349.604207872796 45.598185454496004
8000.0 37.86140116838294
10079.36839915899 32.84445322831763
12699.208415745592 31.758272520564606
16000.0 32.09795512842807
20158.73679831798 32.95422977440326
```

Table 2: *octv3rd.dat* - 1/3 Octave-band center frequencies and sound pressure levels (space-separated)

Appendix B: 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)
```

```
    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

Appendix C: Data Plotting Script

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

DESCRIPTION: Plot processed signal of sonic boom.
narrow-band spectrum
single-side spectral density
SPL
octave bands
overall SPL
"""

#IMPORT GLOBAL VARIABLES
from hw1_98_globalVars import *

import numpy as np
import pandas as pd

import os

#CUSTOM PLOTTING PACKAGE
import matplotlib.pyplot as plt
import sys
sys.path.append('/Users/Logan/lib/python')
from lplot import *
from seaborn import color_palette
import seaborn as sns
UseSeaborn('xkcd') #use seaborn plotting features with custom colors
```

```
colors = sns.color_palette() #color cycle
markers = bigmarkers         #marker cycle

MarkerWidth = 2.25

def PlotArrow(ax, x1, y1, x2, y2, label, head1='<', head2='>',
                color='grey', sz=10):
    """Plot an arrow between two given points.  Specify arrowhead type on
    either side (default double-headed arrow).
    ax      --> plot axis object
    x1,y1   --> x,y coordinates of starting point
    x2,y2   --> x,y coordinates of ending point
    label   --> label for legend
    head1,2 --> first and second arrowheads (e.g. '<', '>', 'v', '^')
    color   --> color of arrow
    sz      --> size of arrowheads
    """
    #Plot line connecting two points
    ax.plot([x1, x2], [y1, y2], color=color, label=label)
    ax.plot(x1, y1, color=color, marker=head1, markersize=sz) #1st arrow head
    ax.plot(x2, y2, color=color, marker=head2, markersize=sz) #2nd arrow head
    return ax


def main():
    """input description
    """

    #Make Plot Output Directory
    MakeOutputDir(picdir)

    #LOAD DATA
    df = pd.read_csv('{}/timespec.dat'.format(datadir), sep=' ' )
    powspec = pd.read_csv('{}/freqspec.dat'.format(datadir), sep=' ' )
    params = pd.read_csv('{}/params.dat'.format(datadir), sep=' ' )
    octv3rd = pd.read_csv('{}/octv3rd.dat'.format(datadir), sep=' ' )
    octv = pd.read_csv('{}/octv.dat'.format(datadir), sep=' ' )


    ####################################################################
    ### 1.1 PLOT PRESSURE WAVE #########################################
    ####################################################################

    #PLOT VOLTAGE
    _,ax = PlotStart(None, 'Time (s)', 'Voltage (V)', figsize=[6, 6])
    #Hollow Marker Plot
    ax.plot(df['time'], df['V'],
            #label=lbl, color=clr,
            # linewidth=0,
            marker=markers[0], markevery=500,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    savename = '{}/1_1_Voltage.{}'.format(picdir, pictype)
    SavePlot(savename)

    #PLOT PRESSURE IN PASCALS
    _,ax = PlotStart(None, 'Time (s)', 'Pressure (Pa)', figsize=[6, 6])
    #Hollow Marker Plot
    ax.plot(df['time'], df['Pa'],
            #label=lbl, color=clr,
            # linewidth=0,
```

```
            #marker=markers[0], markevery=500,
            #markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            #markerfacecolor="None",
            )
    #Plot horizontal arrow marking shock duration
    ax = PlotArrow(ax, params['ti'], params['Pi'], params['tf'], params['Pi'],
                    label='test', head1='<', head2='>', color='grey', sz=7)
    #Plot vertical dashed line at beginning of shock
    ax.plot([params['tf'], params['tf']], [params['Pi'], params['Pf']],
                color='grey', linestyle='--')
    #Put maximum absolute pressure and shock duration in text box
    text = '$P_{{max}}={:.2f}Pa$\n$t_{{Nwave}}={:.4f}s$'.format(
                                                    float(params['Pmax']),
                                                    float(params['tNwave']) )
    TextBox(ax, text, x=0.39, y=0.82, alpha=0.4)

    ax.set_xlim([0.2, 0.7])

    savename = '{}/1_1_Pressure.{}'.format(picdir, pictype)
    SavePlot(savename)

    ####################################################################
    ### 1.2 PLOT POWER SPECTRUM DENSITY ################################
    ####################################################################

    _,ax = PlotStart(None, 'Frequency (Hz)', '$G_{xx}$ ($Pa^2$/$Hz$)', figsize=[6, 6])
    ax.plot(powspec['freq'], powspec['Gxx'],
            marker=markers[0], markevery=1,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    plt.xlim([0,50])

    savename = '{}/1_2_PowerSpec.{}'.format(picdir, pictype)
    SavePlot(savename)

    _,ax = PlotStart(None, 'Frequency (Hz)', '$G_{xx}$ ($Pa^2$/$Hz$)', figsize=[6, 6])
    ax.plot(powspec['freq'], powspec['Gxx'],
            marker=markers[0], markevery=1,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    ax.set_xscale('log')
    plt.xlim([0,10000])

    savename = '{}/1_2_PowerSpecLog.{}'.format(picdir, pictype)
    SavePlot(savename)

    ####################################################################
    ### 1.3 PLOT SOUND PRESSURE LEVEL ##################################
    ####################################################################

    _,ax = PlotStart(None, 'Time (s)', 'SPL (dB)', figsize=[6, 6])
    ax.plot(df['time'], df['SPL'],
            marker=markers[0], markevery=500,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )

    savename = '{}/1_3_SPLt.{}'.format(picdir, pictype)
    SavePlot(savename)
```

```
    _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
    ax.plot(powspec['freq'], powspec['SPL'],
            marker=markers[0], markevery=500,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    ax.set_xscale('log')

    savename = '{}/1_3_SPLf.{}'.format(picdir, pictype)
    SavePlot(savename)

    ####################################################################
    ### 2.1 PLOT 1/3 OCTAVE-BAND SPL ###################################
    ####################################################################

    _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
    ax.plot(octv3rd['freq'], octv3rd['SPL'],
            marker=markers[0], markevery=500,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    ax.set_xscale('log')

    savename = '{}/2_1_SPLf_octv3rd.{}'.format(picdir, pictype)
    SavePlot(savename)

    ####################################################################
    ### 2.2 PLOT OCTAVE-BAND SPL #######################################
    ####################################################################

    _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
    ax.plot(octv['freq'], octv['SPL'],
            marker=markers[0], markevery=500,
            markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
    ax.set_xscale('log')

    savename = '{}/2_2_SPLf_octv.{}'.format(picdir, pictype)
    SavePlot(savename)

    ####################################################################
    ### PLOT ALL OCTAVE-BAND SPL #######################################
    ####################################################################

    _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])

    #Plot Overall SPL as Horizontal Line
    xmin = min(powspec['freq'])
    xmax = max(powspec['freq'])
    ax.plot([xmin, xmax], [params['SPL_overall'], params['SPL_overall']],
            label='Ovr', color='black', linestyle='--')

    i = 1
    #Plot Octave-Band
    ax.plot(octv['freq'], octv['SPL'], label='Oct',
            color=colors[i],
            marker=markers[i], markevery=1,
            markeredgecolor=colors[i], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )
```

```python
    i += 1
    #Plot 1/3 Octave-Band
    ax.plot(octv3rd['freq'], octv3rd['SPL'], label='Oct$\\frac{1}{3}$',
            color=colors[i],
            marker=markers[i], markevery=1,
            markeredgecolor=colors[i], markeredgewidth=MarkerWidth,
            markerfacecolor="None",
            )

    #i += 1
    i = 0
    #Plot Narrow-Band
    ax.plot(powspec['freq'], powspec['SPL'], label='Nar',
            color=colors[i],
            )

    ax.set_xscale('log')
    ax.set_xlim([xmin, xmax])
    PlotLegend(ax)

    #Overall SPL text box
    text = '$L_P={0:.2f}$'.format(float(params['SPL_overall']))
    TextBox(ax, text, x=0.55, y=0.94, alpha=0.4)

    savename = '{}/2_SPLf_all.{}'.format(picdir, pictype)
    SavePlot(savename)



if __name__ == "__main__":




    main()
```

Listing 2: *hw1_01_plot.py* – Performs all plotting associated with the project (refer to Appendix F for supplemental custom plotting package 'lplot.py')

Appendix D: Data Processing/Plotting Wrapper Script

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

DESCRIPTION: Run data processing and plotting programs with common inputs.
"""

#IMPORT GLOBAL VARIABLES AND PROGRAMS TO WRAP
from hw1_98_globalVars import *
import hw1_00_process as process
import hw1_01_plot as plot


def main(source):
    """input description
    """
```

```python

    print('\nProcessing Data')
    process.main(source)
    print('\nPlotting Data')
    plot.main()


if __name__ == "__main__":


    Source = 'Boom_F1B2_6.wav'

    main(Source)
```

Listing 3: *hw1_99_wrapper.py* - Wrapper program that runs data processing and plotting scripts simultaneously

Appendix E: Global Variables

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

DESCRIPTION: Provide global variables for all scripts including wrapper.
"""

#DATA OVERWRITE SWITCHES
overwrite = 1

#LOAD/SAVE DIRECTORIES
datadir = 'Data'    #Source and processed data storage directory
savedir = 'Results' #

picdir = 'Plots' #Plot storage directory
pictype = 'png'      #Plot save filetype
pictype = 'pdf'      #Plot save filetype


#CONVERSIONS
volt2pasc = -116.0 #volts to pascals
```

Listing 4: *hw1_98_globalVars.py* - Container for global variables such as conversion factors, default file types and locations, etc.

Appendix F: Custom Plotting Library

```python
"""PYTHON PLOTTING UTILITIES
Logan Halstrom
07 OCTOBER 2015

DESCRIPTION:  File manipulation, matplotlib plotting and saving.  A subset of
lutil.py simply for plotting.
"""

import subprocess
```

```python
import os
import re
import matplotlib.pyplot as plt
import numpy as np

def MakeOutputDir(savedir):
    """make results output directory if it does not already exist.
    instring --> directory path from script containing folder
    """
    #split individual directories
    splitstring = savedir.split('/')
    prestring = ''
    for string in splitstring:
        prestring += string + '/'
        try:
            os.mkdir(prestring)
        except Exception:
            pass

def GetParentDir(savename):
    """Get parent directory from path of file"""
    #split individual directories
    splitstring = savename.split('/')
    parent = ''
    #concatenate all dirs except bottommost
    for string in splitstring[:-1]:
        parent += string + '/'
    return parent

def GetFilename(path):
    """Get filename from path of file"""
    parent = GetParentDir(path)
    filename = FindBetween(path, parent)
    return filename

def NoWhitespace(str):
    """Return given string with all whitespace removed"""
    return str.replace(' ', '')

def FindBetween(str, before, after=None):
    """Returns search for characters between 'before' and 'after characters
    If after=None, return everything after 'before'"""
    # value_regex = re.compile('(?<=' + before + ')(?P<value>.*?)(?='
    #                                   + after + ')')
    if after==None:
        match = re.search(before + '(.*)$', str)
        if match != None: return match.group(1)
        else: return 'No Match'
    else:
        match = re.search('(?<=' + before + ')(?P<value>.*?)(?='
                                     + after + ')', str)
        if match != None: return match.group('value')
        else: return 'No Match'


##########################################################################
### PLOTTING #############################################################
##########################################################################

xkcdcolors = ["windows blue", "dusty purple", "leaf green", "macaroni and cheese",  "cherry"   "greyish
xkcdhex =    ['#3778bf',      '#825f87',      '#5ca904',    '#efb435',              '#cf0234'  '#a8a495
```

```python
def UseSeaborn(palette='deep'):
    """Call to use seaborn plotting package
    """
    import seaborn as sns
    #No Background fill, legend font scale, frame on legend
    sns.set(style='whitegrid', font_scale=1.5, rc={'legend.frameon': True})
    #Mark ticks with border on all four sides (overrides 'whitegrid')
    sns.set_style('ticks')
    #ticks point in
    sns.set_style({"xtick.direction": "in","ytick.direction": "in"})

    # sns.choose_colorbrewer_palette('q')

    #Nice Blue,green,Red
    # sns.set_palette('colorblind')
    if palette == 'xkcd':
        #Nice blue, purple, green
        sns.set_palette(sns.xkcd_palette(xkcdcolors))
    else:
        sns.set_palette(palette)
    #Nice blue, green red
    # sns.set_palette('deep')

    # sns.set_palette('Accent_r')
    # sns.set_palette('Set2')
    # sns.set_palette('Spectral_r')
    # sns.set_palette('spectral')

    #FIX INVISIBLE MARKER BUG
    sns.set_context(rc={'lines.markeredgewidth': 0.1})

#PLOT FORMATTING
# Configure figures for production
WIDTH = 495.0  # width of one column
FACTOR = 1.0   # the fraction of the width the figure should occupy
fig_width_pt  = WIDTH * FACTOR

inches_per_pt = 1.0 / 72.27
golden_ratio  = (np.sqrt(5) - 1.0) / 2.0      # because it looks good
fig_width_in  = fig_width_pt * inches_per_pt  # figure width in inches
fig_height_in = fig_width_in * golden_ratio   # figure height in inches
fig_dims      = [fig_width_in, fig_height_in] # fig dims as a list

#Line Styles
mark = 5
minimark = 0.75
line = 1.5

#dot, start, x, tri-line, plus
smallmarkers = ['.', '*', 'd', '1', '+']
bigmarkers = ['o', 'v', 'd', 's', '*', 'D', 'p', '>', 'H', '8']
scattermarkers = ['o', 'v', 'd', 's', 'p']

#GLOBAL INITIAL FONT SIZES
Ttl = 32
Lbl = 32
Box = 28
Leg = 28
Tck = 22

#MAKE FONT DICT GLOBAL SO IT CAN BE MADE AND USED IN DIFFERENT FUNCTIONS
global font_ttl, font_lbl, font_box, font_tck, font_leg
```

```python

def SetFontDictSize(ttl=None, lbl=None, box=None, tck=None, leg=None):
    """Set font size in styling dictionaries.  Global dictionaries to use for
    font styles in all functions.
    To change a single parameter, call function like: SetFontDictSize(lbl=18)
    ttl --> title, lbl --> axis label, box --> textbox,
    tck --> axis tick labels, leg --> legend text
    """
    global font_ttl, font_lbl, font_box, font_tck, font_leg

    #DEFAULT FONT SIZES
    # if ttl == None: ttl = 18
    # if lbl == None: lbl = 18
    # if box == None: box = 12
    # if tck == None: tck = 16
    # if leg == None: leg = 16

    if ttl == None: ttl = Ttl
    if lbl == None: lbl = Lbl
    if box == None: box = Box
    if tck == None: tck = Tck
    if leg == None: leg = Leg

    #Font Styles
    font_ttl = {'family' : 'serif',
                'color'  : 'black',
                'weight' : 'normal',
                'size'   : ttl,
                }
    font_lbl = {'family' : 'serif',
                'color'  : 'black',
                'weight' : 'normal',
                'size'   : lbl,
                }
    font_box = {'family' : 'arial',
                'color'  : 'black',
                'weight' : 'normal',
                'size'   : box,
                }
    font_tck = tck
    font_leg = leg


#INITIAL FONT DICT SETTINGS
SetFontDictSize()

#Textbox Properties
textbox_props = dict(boxstyle='round', facecolor='white', alpha=0.5)


params = {
            # 'backend': 'ps',
          # 'text.latex.preamble': ['\usepackage{gensymb}'],
          'axes.labelsize' : Lbl,
          'axes.titlesize' : Ttl,
          'text.fontsize' : Box,
          'legend.fontsize': Leg,
          'xtick.labelsize': Tck,
          'ytick.labelsize': Tck,
          # 'text.usetex': True,
          # 'figure.figsize': [fig_width,fig_height],
          'font.family': 'helvetica'
```

```
}
import matplotlib
matplotlib.rcParams.update(params)




def PlotStart(title, xlbl, ylbl, horzy='vertical', figsize='square',
                ttl=None, lbl=None, tck=None, leg=None, box=None,
                grid=True, rc=False):
    """Begin plot with title and axis labels.  Space title above plot.
    horzy --> vertical or horizontal y axis label
    figsize --> set figure size. None for autosizing, 'tex' for latex
                    formatting, or 2D list for user specification.
    ttl,lbl,tck --> title, label, and axis font sizes
    """

    #SET FIGURE SIZE
    if figsize == None:
        #Plot with automatic figure sizing
        fig = plt.figure()
    else:
        if figsize == 'tex':
            #Plot with latex 2-column figure sizing
            figsize = fig_dims
        elif figsize == 'square':
            figsize = [6, 6]
        #Otherwise, plot with user-specificed dimensions (i.e. [width, height])
        fig = plt.figure(figsize=figsize)

    #PLOT FIGURE
    ax = fig.add_subplot(1, 1, 1)

    if rc:
        #USE MATPLOTLIB RC PARAMS SETTINGS
        if title != None:
            plt.title(title)
        plt.xlabel(xlbl)
        plt.ylabel(ylbl)
    else:
        #USE FONT DICT SETTINGS

        #Set font sizes
        if ttl != None or lbl != None or tck != None or leg != None or box != None:
            #Set any given font sizes
            SetFontDictSize(ttl=ttl, lbl=lbl, tck=tck, leg=leg, box=box)
        else:
            #Reset default font dictionaries
            SetFontDictSize()

        if title != None:
            plt.title(title, fontdict=font_ttl)
        plt.xlabel(xlbl, fontdict=font_lbl)
        plt.xticks(fontsize=font_tck)
        plt.ylabel(ylbl, fontdict=font_lbl, rotation=horzy)
        plt.yticks(fontsize=font_tck)

    #INCREASE TITLE SPACING
    ttl = ax.title
    ttl.set_position([.5, 1.025])
    # ax.xaxis.set_label_coords( .5, -1.025*10 )
```

```python
    # ax.yaxis.labelpad = 20

    #TURN GRID ON
    if grid:
        ax.grid(True)

    return fig, ax

def MakeTwinx(ax, ylbl, horzy='vertical'):
    """Make separate y-axis with label"""
    ax2 = ax.twinx()
    ax2.set_ylabel(ylbl, fontdict=font_lbl, rotation=horzy)
    plt.yticks(fontsize=font_tck)
    return ax2

def ZeroAxis(ax, dir='x'):
    """Set axis lower bound to zero, keep upper bound
    """
    if dir == 'x':
        ax.set_xlim([0, ax.get_xlim()[1]])
    elif dir == 'y':
        ax.set_ylim([0, ax.get_ylim()[1]])

def ZeroAxes(ax):
    """Set both axes lower bound to zero, keep upper bound
    """
    ax.set_xlim([0, ax.get_xlim()[1]])
    ax.set_ylim([0, ax.get_ylim()[1]])

def Plot(ax, x, y, color, label, linestyle='-',
            marker='None', line=1.5, mark=5):
    """Enter 'Default' to keep default value if entering values for later
    variables"""
    return ax.plot(x, y, color=color, label=label, linestyle=linestyle,
                    linewidth=line, marker=marker, markersize=mark)

def PlotLegend(ax, loc='best', alpha=0.5, title=None, fontsize=None):
    """General legend command.  Use given handles and labels in plot
    commands.  Curved edges, semi-transparent, given title, single markers
    """
    if fontsize == None:
        fontsize = font_leg
    leg = ax.legend(loc=loc, title=title,
                    fancybox=True, frameon=True, framealpha=alpha,
                    numpoints=1, scatterpoints=1, prop={'size':fontsize},
                    borderpad=0.1, borderaxespad=0.1, handletextpad=0.2,
                    handlelength=1.0, labelspacing=0)
    return leg

def PlotLegendLabels(ax, handles, labels, loc='best', title=None, alpha=0.5,
                        fontsize=None):
    """Plot legend specifying labels.
    Curved edges, semi-transparent, given title, single markers
    """
    if fontsize == None:
        fontsize = font_leg
    leg = ax.legend(handles, labels, loc=loc, title=title,
                    fancybox=True, frameon=True, framealpha=alpha,
                    numpoints=1, scatterpoints=1, prop={'size':fontsize},
                    borderpad=0.1, borderaxespad=0.1, handletextpad=0.2,
                    handlelength=1.0, labelspacing=0)
    # leg.get_frame().set_alpha(alpha)
```

```python
    # for label in leg.get_texts():
    #     label.set_fontsize('large')
    return leg

def ColorMap(ncolors, colormap='jet'):
    """return array of colors given number of plots and colormap name
    colormaps: jet, brg, Accent, rainbow
    """
    cmap = plt.get_cmap(colormap)
    colors = [cmap(i) for i in np.linspace(0, 1, ncolors)]
    return colors

def ColorBar(label, horzy='horizontal', ticks=None, colorby=None, pad=25):
    """Add colorbar with label.
    ax --> matplotlib axis object
    label --> colorbar label
    horzy --> label orientation
    ticks --> manually provide colorbar tick location (default is automatic)
    colorby --> data to base colorbar on.  Default is whatever was plotted last
    pad --> spacing of label from colorbar. positive is further away
    """
    if colorby == None:
        #MAKE COLORBAR
        cb = plt.colorbar()
    else:
        #MAKE COLORBAR WITH GIVEN DATA
        cb = plt.colorbar(colorby)
    #SET LABEL
    cb.set_label(label, rotation=horzy, fontdict=font_lbl, labelpad=pad)
    return cb

def SavePlot(savename, overwrite=1, trans=False):
    """Save file given save path.  Do not save if file exists
    or if variable overwrite is 1"""
    if os.path.isfile(savename):
        if overwrite == 0:
            print('     Overwrite is off')
            return
        else: os.remove(savename)
    MakeOutputDir(GetParentDir(savename))
    plt.savefig(savename, bbox_inches='tight', transparent=trans)
    # plt.close()

def ShowPlot(showplot=1):
    """Show plot if variable showplot is 1"""
    if showplot == 1:
        plt.show()
    else:
        plt.close()

def GridLines(ax, linestyle='--', color='k', which='major'):
    """Plot grid lines for given axis.
    Default dashed line, blach, major ticks
    (use: 'color = p1.get_color()' to get color of a line 'p1')
    """
    ax.grid(True, which=which, linestyle=linestyle, color=color)

def TextBox(ax, boxtext, x=0.005, y=0.95, fontsize=font_box['size'],
                                                    alpha=0.5, props=None):
    if props == None:
        props = dict(boxstyle='round', facecolor='white', alpha=alpha)
    ax.text(x, y, boxtext, transform=ax.transAxes, fontsize=fontsize,
```

```
            verticalalignment='top', bbox=props)

def TightLims(ax, tol=0.0):
    """Return axis limits for tight bounding of data set in ax.
    NOTE: doesn't work for scatter plots.
    ax  --> plot axes to bound
    tol --> whitespace tolerance
    """
    xmin = xmax = ymin = ymax = None
    for line in ax.get_lines():
        data = line.get_data()
        curxmin = min(data[0])
        curxmax = max(data[0])
        curymin = min(data[1])
        curymax = max(data[1])
        if curxmin < xmin or xmin == None:
            xmin = curxmin
        if curxmax > xmax or xmax == None:
            xmax = curxmax
        if curymin < ymin or ymin == None:
            ymin = curymin
        if curymax > ymax or ymax == None:
            ymax = curymax

    xlim = [xmin-tol, xmax+tol]
    ylim = [ymin-tol, ymax+tol]

    return xlim, ylim

def PadBounds(axes, tol=0):
    """Add tolerance to axes bounds to pad with whitespace"""

    xtol = (axes[1] - axes[0]) * tol
    ytol = (axes[3] - axes[2]) * tol
    tols = [-xtol, xtol, -ytol, ytol]
    for i, (x, t) in enumerate(zip(axes,tols)):
        axes[i] += t
    return axes

def VectorMark(ax, x, y, nmark, color='k'):
    """Mark line with arrow pointing in direction of x+.
    Show nmark arrows
    """
    n = len(y)
    dm = int(len(y) / nmark)
    # indicies = np.linspace(1, n-2, nmark)
    indicies = [1]
    while indicies[-1]+dm < len(y)-1:
        indicies.append(indicies[-1] + dm)

    for ind in indicies:
        #entries are x, y, dx, dy
        xbase, ybase = x[ind], y[ind]
        dx, dy = x[ind+1] - x[ind], y[ind+1] - y[ind]
        ax.quiver(xbase, ybase, dx, dy ,angles='xy',scale_units='xy',scale=1)

def PlotArrow(ax, x1, y1, x2, y2, label, head1='<', head2='>',
                color='grey', sz=10):
    """Plot an arrow between two given points.  Specify arrowhead type on
    either side (default double-headed arrow).
    ax      --> plot axis object
    x1,y1   --> x,y coordinates of starting point
```

```
    x2,y2   --> x,y coordinates of ending point
    label   --> label for legend
    head1,2 --> first and second arrowheads (e.g. '<', '>', 'v', '^')
    color   --> color of arrow
    sz      --> size of arrowheads
    """
    #Plot line connecting two points
    ax.plot([x1, x2], [y1, y2], color=color, label=label)
    ax.plot(x1, y1, color=color, marker=head1, markersize=sz) #1st arrow head
    ax.plot(x2, y2, color=color, marker=head2, markersize=sz) #2nd arrow head
    return ax

def PlotVelProfile(ax, y, u, color='green', narrow=4):
    """Plot velocity profile as y vs y
    y --> non-dim. vetical grid within BL (y/delta)
    u --> non-dim. x-velocity withing BL (u/u_e)
    color --> sting, color of plot
    narrow --> number of points between arrows
    """
    vertlinex = np.zeros(len(y))
    ax.plot(vertlinex, y, color=color, linewidth=line)
    ax.fill_betweenx(y, vertlinex, u, facecolor=color, alpha=0.2)
    wd, ln = 0.03, 0.03
    for i in range(0, len(y), narrow):
        if abs(u[i]) < ln:
            ax.plot([0, u[i]], [y[i], y[i]], color=color, linewidth=line)
        else:
            ax.arrow(0, y[i], u[i]-ln, 0, head_width=wd, head_length=ln,
                fc=color, ec=color, linewidth=line)
    ax.plot(u, y, color=color, linewidth=line)
    ax.axis([min(u), max(u), min(y), max(y)])

def PolyFit(x, y, order, n, showplot=0):
    """Polynomial fit xdata vs ydata points
    x --> independent variable data points vector
    y --> dependent variable data points vector
    order --> order of polynomial fit
    n --> number of points in polynomial fit
    showplot --> '1' to show plot of data fit
    Returns:
    function of polynomial fit
    """
    #New independent variable vector:
    xmin, xmax = x[0], x[-1]
    x_poly = np.linspace(xmin, xmax, n)
    fit = np.polyfit(x, y, order)
    polyfit = np.poly1d(fit)
    y_poly = polyfit(x_poly)
    #Plot Poly Fit
    plt.figure()
    plt.title(str(order) + '-Order Polynomial Fit', fontsize=14)
    plt.xlabel('x', fontsize=14)
    plt.ylabel('y', fontsize=14)
    plt.plot(x, y, 'rx', label='Data')
    plt.plot(x_poly, y_poly, 'b', label='Fit')
    plt.legend(loc='best')
    if showplot == 1:
        plt.show()
    return polyfit
```

Listing 5: *lplot.py* - Custom plotting library developed by Logan Halstrom providing default plotting parameters and enhanced plotting functions