

# 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) across 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 supplemented 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 identical values, all libraries returned the same results when normalized by the maximum absolute 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 contained in Eqn 1:

$$-116\text{Pa} = 1\text{V} \quad (1)$$

The resulting pressure history 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 at the rear of the aircraft. 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, resulting in a single “boom”). Aft of 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.

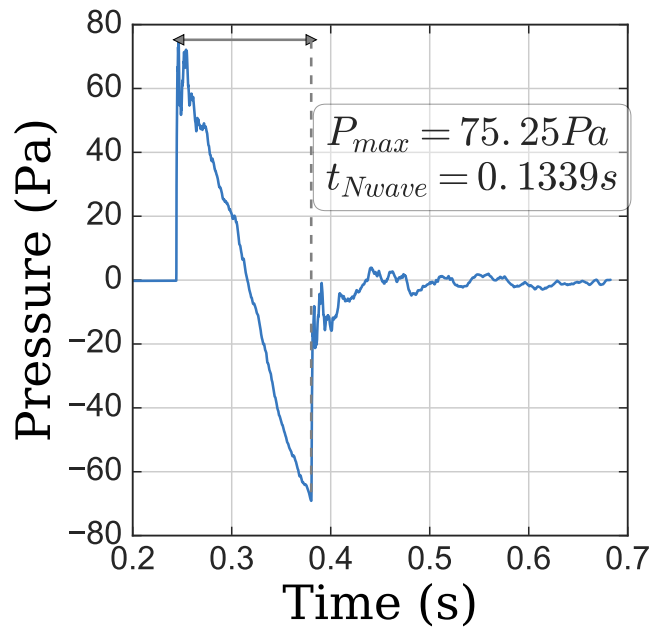


Fig. 1: Recorded sonic boom shockwave pressure time history with peak pressure  $P_{max}$  and N-wave time duration  $t_{Nwave}$  values (Zero-pressure from recording start to initial shock)

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

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

and sonic boom N-wave duration:

$$t_{Nwave} = 133.9 ms$$

### 3 Problem 1.2 – Power Spectral Density Decomposition

In order to analyze the frequency-dependent nature of the recorded sonic boom, it is necessary 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 into the frequency domain, and can be called in Python using ‘numpy.fft’.

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

$$fft = np.fft.fft(pressure) \cdot dt$$

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:

$$S_{xx} = \frac{|fft|^2}{T} \quad (2)$$

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 single-sided power spectrum  $G_{xx}$  can be acquired as twice of the first half of  $S_{xx}$  (Eqn 3):

$$G_{xx} = 2S_{xx}[idx] \quad (3)$$

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 that 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’:

$$\begin{aligned} freqs &= np.fft.fftfreq(pressure.size, dt) \\ freqs &= freqs[idx] \end{aligned}$$

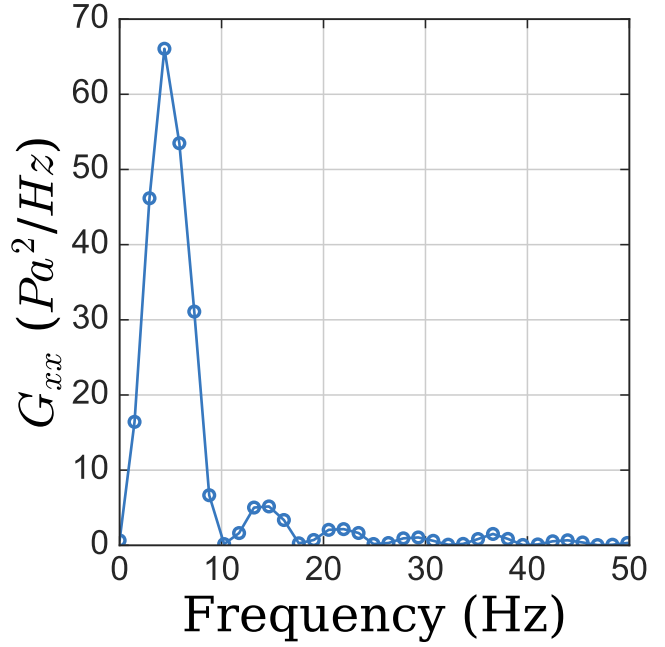


Fig. 2: Shockwave signal power spectral density as a function of frequency (All frequencies above 50Hz are very low power)

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  $66 Pa^2/Hz$  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  $P_{ref} = 20 \mu Pa$ .

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

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.

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

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

In the data processing script for this analysis, center frequencies are calculated specifically to match the narrow-band input. The function ‘OctaveCenterFreqs’ will loop through  $m$  in intervals appropriate to the given octave-band choice, but only return 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.

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

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.

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

where  $f_{l,c}, f_{u,c}$  are the band limits 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  $10\text{Hz}$ , 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.

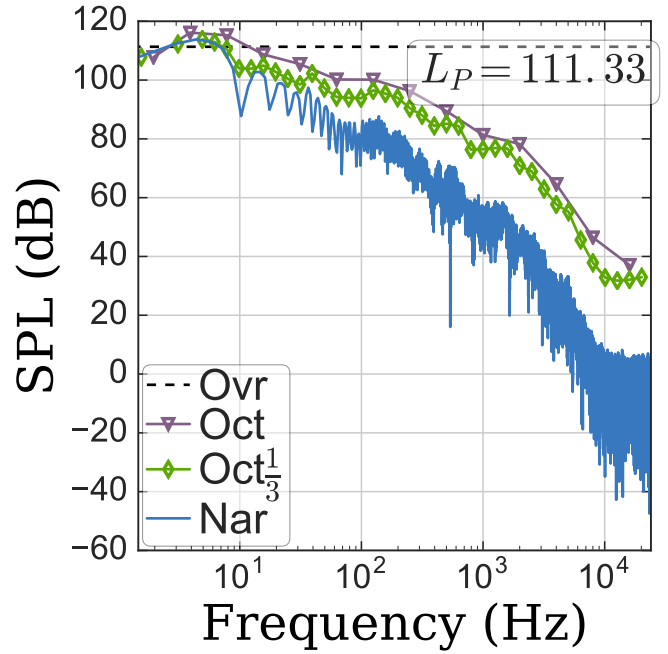


Fig. 3: Shockwave signal narrow-band (Nar), 1/3 octave-band ( $\text{Oct}_{\frac{1}{3}}$ ), and octave-band (Oct), with overall Sound Pressure Level (Ovr) reported in upper right ( $L_p$ )

#### 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 general trends as that of the narrow-band, but with much 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 generally represented with a very small data set.

#### 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, but 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  $SPL_{ovr} = 111.33dB$ , 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 all of the spectra, especially at high frequencies. 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 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

```
0 """HW1 - DATA PROCESSING
1 Logan Halstrom
2 MAE 298 AEROACOUSTICS
3 HOMEWORK 1 - SIGNAL PROCESSING
4 CREATED: 04 OCT 2016
5 MODIFIY: 17 OCT 2016
6
7 DESCRIPTION: Read sound file of sonic boom and convert signal to
8 Narrow-band in Pa.
9 Compute Single-side power spectral density (FFT).
10 1/3 octave and octave band
11
12 NOTE: use 'soundfile' module to read audio data. This normalizes data from
13 -1 to 1, like Matlab's 'audioread'. 'scipy.io.wavfile' does not normalize
14 """
15
16 #IMPORT GLOBAL VARIABLES
17 from hw1_98_globalVars import *
18
19 import numpy as np
20 import pandas as pd
21
22 def ReadWavNorm(filename):
23     """Read a .wav file and return sampling frequency.
24     Use 'soundfile' module, which normalizes audio data between -1 and 1,
25     identically to MATLAB's 'audioread' function
26     """
27     import soundfile as sf
28     #Returns sampled data and sampling frequency
29     data, samplerate = sf.read(filename)
30     return samplerate, data
31
32 def ReadWav(filename):
33     """NOTE: NOT USED IN THIS CODE, DOES NOT NORMALIZE LIKE MATLAB
34     Read a .wav file and return sampling frequency
35     Use 'scipy.io.wavfile' which doesn't normalize data.
36     """
37     from scipy.io import wavfile
38     #Returns sample frequency and sampled data
39     sampFreq, snd = wavfile.read(filename)
40     #snd = Normalize(snd)
41     return sampFreq, snd
42
43 def Normalize(data):
44     """NOTE: NOT USED IN THIS CODE, TRIED BUT FAILED TO NORMALIZE LIKE MATLAB
45     Trying to normalize data between -1 and 1 like matlab audioread
46     """
47     data = np.array(data)
48     return ( 2*(data - min(data)) / (max(data) - min(data)) - 1)
49
50 def SPLt(P, Pref=20e-6):
51     """Sound Pressure Level (SPL) in dB as a function of time.
52     P --> pressure signal (Pa)
53     Pref --> reference pressure
54     """
55     PrmsSq = 0.5 * P ** 2 #RMS pressure squared
56     return 10 * np.log10(PrmsSq / Pref ** 2)
57
58 def SPLf(Gxx, T, Pref=20e-6):
59     """Sound Pressure Level (SPL) in dB as a function of frequency
60     Gxx --> Power spectral density of a pressure signal (after FFT)
```

```

61     T    --> Total time interval of pressure signal
62     Pref --> reference pressure
63     """
64     return 10 * np.log10( (Gxx / T) / Pref ** 2 )
65
66 def OctaveCenterFreqsGen(dx=3, n=39):
67     """NOTE: NOT USED IN THIS CODE. INSTEAD, OCTAVECENTERFREQS
68     Produce general center frequencies for octave-band spectra
69     dx --> frequency interval spacing (3 for octave, 1 for 1/3 octave)
70     n --> number of center freqs to product (starting at dx)
71     """
72     fc30 = 1000 #Preferred center freq for m=30 is 1000Hz
73     m = np.arange(1, n+1) * dx #for n center freqs, multiply 1-->n by dx
74     freqs = fc30 * 2 ** (-10 + m/3) #Formula for center freqs
75
76 def OctaveBounds(fc, octv=1):
77     """Get upper/lower frequency bounds for given octave band.
78     fc --> current center frequency
79     octv --> octave-band (octave-->1, 1/3 octave-->1/3)
80     """
81     upper = 2 ** ( octv / 2 ) * fc
82     lower = 2 ** (-octv / 2) * fc
83     return upper, lower
84
85 def OctaveCenterFreqs(narrow, octv=1):
86     """Calculate center frequencies (fc) for octave or 1/3 octave bands.
87     Provide original narrow-band frequency vector to bound octave-band.
88     Only return center frequencies who's lowest lower band limit or highest
89     upper band limit are within the original data set.
90     narrow --> original narrow-band frequencies (provides bounds for octave)
91     octv --> frequency interval spacing (1 for octave, 1/3 for 1/3 octave)
92     """
93     fc30 = 1000 #Preferred center freq for m=30 is 1000Hz
94     freqs = []
95     for i in range(len(narrow)):
96         #current index
97         m = (3 * octv) * (i + 1) #octave, every 3rd, 1/3 octave, every 1
98         fc = fc30 * 2 ** (-10 + m/3) #Formula for center freq
99         fcu, fcl = OctaveBounds(fc, octv) #upper and lower bounds for fc band
100        if fcu > max(narrow):
101            break #quit if current fc is greater than original range
102        if fcl >= min(narrow):
103            freqs.append(fc) #if current fc is in original range, save
104    return freqs
105
106 def OctaveLp(Lp):
107     """Given a range of SPLs that are contained within a given octave band,
108     perform the appropriate log-sum to determine the octave SPL
109     Lp --> SPL range in octave-band
110     """
111     #Sum 10^(Lp/10) accross current octave-band, take log
112     Lp_octv = 10 * np.log10( np.sum( 10 ** (Lp / 10) ) )
113     return Lp_octv
114
115 def GetOctaveBand(df, octv=1):
116     """Get SPL ( Lp(fc,m) ) for octave-band center frequencies.
117     Returns octave-band center frequencies and corresponding SPLs
118     df --> pandas dataframe containing narrow-band frequencies and SPL
119     octv --> octave-band type (octave-->1, 1/3 octave-->1/3)
120     """
121
122     #Get Center Frequencies

```

```

123 fcs = OctaveCenterFreqs(df['freq'], octv)
124 Lp_octv = np.zeros(len(fcs))
125 for i, fc in enumerate(fcs):
126     #Get Upper/Lower center frequency band bounds
127     fcu, fcl = OctaveBounds(fc, octv)
128
129     band = df[df['freq'] >= fcl]
130     band = band[band['freq'] <= fcu]
131
132     #SPLs in current octave-band
133     Lp = np.array(band['SPL'])
134     #Sum 10^(Lp/10) accross current octave-band, take log
135     Lp_octv[i] = OctaveLp(Lp)
136
137 return fcs, Lp_octv
138
139
140
141
142 def main(source):
143     """Perform calculations for frequency data processing
144     source --> file name of source sound file
145     """
146
147     #####
148     ### READ SOUND FILE #####
149     #####
150
151     df = pd.DataFrame() #Stores signal data
152
153     #Read source frequency (fs) and signal in volts normalized between -1&1
154     fs, df['V'] = ReadWavNorm( '{}/{}'.format(datadir, source) ) #Like matlab
155
156     #Convert to pascals
157     df['Pa'] = df['V'] * volt2pasc
158
159     #####
160     ### POWER SPECTRAL DENSITY #####
161     #####
162
163     #TIME
164     #calculate time of each signal, in seconds, from source frequency
165     N = len(df['Pa']) #Number of data points in signal
166     dt = 1 / fs #time step
167     T = N * dt #total time interval of signal (s)
168     df['time'] = np.arange(N) * dt #individual sample times
169     idx = range(int(N/2)) #Indices of single-sided power spectrum (first half)
170
171     #POWER SPECTRUM
172     fft = np.fft.fft(df['Pa']) * dt #Fast-Fourier Transform
173     Sxx = np.abs(fft) ** 2 / T #Two-sided power spectrum
174     #Gxx = Sxx[idx] #Single-sided power spectrum
175     Gxx = 2 * Sxx[idx] #Single-sided power spectrum
176
177     #FREQUENCY
178     freqs = np.fft.fftfreq(df['Pa'].size, dt) #Frequencies
179     #freqs = np.arange(N) / T #Frequencies
180     freqs = freqs[idx] #single-sided frequencies
181
182     #COMBINE POWER SPECTRUM DATA INTO DATAFRAME
183     powspec = pd.DataFrame({'freq' : freqs, 'Gxx' : Gxx})
184

```

```

185 maxima = powspec[powspec['Gxx'] == max(powspec['Gxx'])]
186 print('\nMaximum Power Spectrum, frequency:t', float(maxima['freq']))
187 print( 'Maximum Power Spectrum, power:', float(maxima['Gxx' ]))
188
189 #####
190 ### FIND SOUND PRESSURE LEVEL IN dB #####
191 #####
192
193 #SPL VS TIME
194 df['SPL'] = SPLt(df['Pa'])
195 #SPL VS FREQUENCY
196 powspec['SPL'] = SPLf(Gxx, T)
197
198 #####
199 ### SONIC BOOM N-WAVE PEAK AND DURATION #####
200 #####
201
202 #SONIC BOOM PRESSURE PEAK
203 Pmax = max(abs(df['Pa']))
204
205 #SONIC BOOM N-WAVE DURATION
206 #Get shock starting and ending times and pressures
207 shocki = df[df['Pa'] == max(df['Pa'])] #Shock start
208 ti = float(shocki['time']) #start time
209 Pi = float(shocki['Pa']) #start (max) pressure
210 shockf = df[df['Pa'] == min(df['Pa'])] #Shock end
211 tf = float(shockf['time']) #start time
212 Pf = float(shockf['Pa']) #start (max) pressure
213 #Shockwave time duration
214 dt_Nwave = tf - ti
215
216 #####
217 ### OCTAVE-BAND CONVERSION #####
218 #####
219
220 #1/3 OCTAVE-BAND
221 octv3rd = pd.DataFrame()
222 octv3rd['freq'], octv3rd['SPL'] = GetOctaveBand(powspec, octv=1/3)
223
224 #OCTAVE-BAND
225 octv = pd.DataFrame()
226 octv['freq'], octv['SPL'] = GetOctaveBand(powspec, octv=1)
227
228 #OVERALL SOUND PRESSURE LEVEL
229 #Single SPL value for entire series
230 #Sum over either octave or 1/3 octave bands (identical)
231 #but exclude frequencies below 10Hz
232 Lp_overall = OctaveLp(octv[octv['freq'] >= 10.0]['SPL'])
233
234 print('\nNum. of Points, Narrow-band:' , len(df))
235 print( 'Num. of Points, 1/3 Octave-band:', len(octv3rd))
236 print( 'Num. of Points, Octave-band:' , len(octv))
237
238 #####
239 ### SAVE DATA #####
240 #####
241
242 #SAVE WAVE SIGNAL DATA
243 df = df[['time', 'Pa', 'SPL', 'V']] #reorder
244 df.to_csv( '{}/timespec.dat'.format(datadir), sep=' ', index=False ) #save
245
246 #SAVE POWER SPECTRUM DATA

```



```

247     powspec.to_csv( '{}/freqspec.dat'.format(datadir), sep=' ', index=False )
248
249     #SAVE OCTAVE-BAND DATA
250     octv3rd.to_csv( '{}/octv3rd.dat'.format(datadir), sep=' ', index=False)
251     octv.to_csv( '{}/octv.dat'.format(datadir), sep=' ', index=False)
252
253     #SAVE SINGLE PARAMETERS
254     params = pd.DataFrame()
255     params = params.append(pd.Series(
256         {'fs' : fs, 'SPL_overall' : Lp_overall,
257          'Pmax' : Pmax, 'tNwave' : dt_Nwave,
258          'ti' : ti, 'Pi' : Pi, 'tf' : tf, 'Pf' : Pf}
259         ), ignore_index=True)
260     params.to_csv( '{}/params.dat'.format(datadir), sep=' ', index=False)
261
262
263 if __name__ == "__main__":
264
265     Source = 'Boom_F1B2_6.wav'
266
267     main(Source)
268

```

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

```

0  """HW1 - DATA PLOTTING
1  Logan Halstrom
2  MAE 298 AEROACOUSTICS
3  HOMEWORK 1 - SIGNAL PROCESSING
4  CREATED: 04 OCT 2016
5  MODIFIY: 17 OCT 2016
6
7  DESCRIPTION: Plot processed signal of sonic boom.
8  narrow-band spectrum
9  single-side spectral density
10 SPL
11 octave bands
12 overall SPL
13 """
14
15 #IMPORT GLOBAL VARIABLES
16 from hw1_98_globalVars import *
17
18 import numpy as np
19 import pandas as pd
20
21 import os
22
23 #CUSTOM PLOTTING PACKAGE
24 import matplotlib.pyplot as plt
25 import sys
26 sys.path.append('/Users/Logan/lib/python')
27 from lplot import *
28 from seaborn import color_palette
29 import seaborn as sns
30 UseSeaborn('xkcd') #use seaborn plotting features with custom colors

```

```

31 colors = sns.color_palette() #color cycle
32 markers = bigmarkers        #marker cycle
33
34 MarkerWidth = 2.25
35
36 def PlotArrow(ax, x1, y1, x2, y2, label, head1='<', head2='>',
37             color='grey', sz=10):
38     """Plot an arrow between two given points. Specify arrowhead type on
39     either side (default double-headed arrow).
40     ax --> plot axis object
41     x1,y1 --> x,y coordinates of starting point
42     x2,y2 --> x,y coordinates of ending point
43     label --> label for legend
44     head1,2 --> first and second arrowheads (e.g. '<', '>', 'v', '^')
45     color --> color of arrow
46     sz --> size of arrowheads
47     """
48     #Plot line connecting two points
49     ax.plot([x1, x2], [y1, y2], color=color, label=label)
50     ax.plot(x1, y1, color=color, marker=head1, markersize=sz) #1st arrow head
51     ax.plot(x2, y2, color=color, marker=head2, markersize=sz) #2nd arrow head
52     return ax
53
54
55 def main():
56     """input description
57     """
58
59     #Make Plot Output Directory
60     MakeOutputDir(picdir)
61
62     #LOAD DATA
63     df = pd.read_csv('{}timespec.dat'.format(datadir), sep=' ')
64     powspec = pd.read_csv('{}freqspec.dat'.format(datadir), sep=' ')
65     params = pd.read_csv('{}params.dat'.format(datadir), sep=' ')
66     octv3rd = pd.read_csv('{}octv3rd.dat'.format(datadir), sep=' ')
67     octv = pd.read_csv('{}octv.dat'.format(datadir), sep=' ')
68
69
70     #####
71     ### 1.1 PLOT PRESSURE WAVE #####
72     #####
73
74     #PLOT VOLTAGE
75     _,ax = PlotStart(None, 'Time (s)', 'Voltage (V)', figsize=[6, 6])
76     #Hollow Marker Plot
77     ax.plot(df['time'], df['V'],
78           #label=lbl, color=clr,
79           # linewidth=0,
80           marker=markers[0], markevery=500,
81           markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
82           markerfacecolor="None",
83           )
84     savename = '{}1_1_Voltage.{}'.format(picdir, pictype)
85     SavePlot(savename)
86
87     #PLOT PRESSURE IN PASCALS
88     _,ax = PlotStart(None, 'Time (s)', 'Pressure (Pa)', figsize=[6, 6])
89     #Hollow Marker Plot
90     ax.plot(df['time'], df['Pa'],
91           #label=lbl, color=clr,
92           # linewidth=0,

```

```

93         #marker=markers[0], markevery=500,
94         #markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
95         #markerfacecolor="None",
96     )
97     #Plot horizontal arrow marking shock duration
98     ax = PlotArrow(ax, params['ti'], params['Pi'], params['tf'], params['Pi'],
99                   label='test', head1='<', head2='>', color='grey', sz=7)
100    #Plot vertical dashed line at beginning of shock
101    ax.plot([params['tf'], params['tf']], [params['Pi'], params['Pf']],
102            color='grey', linestyle='--')
103    #Put maximum absolute pressure and shock duration in text box
104    text = '$P_{\max}={:.2f}$Pa$\n$t_{Nwave}={:.4f}$s'.format(
105                                                float(params['Pmax']),
106                                                float(params['tNwave']) )
107    TextBox(ax, text, x=0.39, y=0.82, alpha=0.4)
108
109    ax.set_xlim([0.2, 0.7])
110
111    savename = '{}/1_1_Pressure.{}'.format(picdir, pictype)
112    SavePlot(savename)
113
114    #####
115    ### 1.2 PLOT POWER SPECTRUM DENSITY #####
116    #####
117
118    _,ax = PlotStart(None, 'Frequency (Hz)', '$G_{xx}$ ($Pa^2/$Hz$)', figsize=[6, 6])
119    ax.plot(powspec['freq'], powspec['Gxx'],
120           marker=markers[0], markevery=1,
121           markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
122           markerfacecolor="None",
123           )
124    plt.xlim([0,50])
125
126    savename = '{}/1_2_PowerSpec.{}'.format(picdir, pictype)
127    SavePlot(savename)
128
129    _,ax = PlotStart(None, 'Frequency (Hz)', '$G_{xx}$ ($Pa^2/$Hz$)', figsize=[6, 6])
130    ax.plot(powspec['freq'], powspec['Gxx'],
131           marker=markers[0], markevery=1,
132           markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
133           markerfacecolor="None",
134           )
135    ax.set_xscale('log')
136    plt.xlim([0,10000])
137
138    savename = '{}/1_2_PowerSpecLog.{}'.format(picdir, pictype)
139    SavePlot(savename)
140
141    #####
142    ### 1.3 PLOT SOUND PRESSURE LEVEL #####
143    #####
144
145    _,ax = PlotStart(None, 'Time (s)', 'SPL (dB)', figsize=[6, 6])
146    ax.plot(df['time'], df['SPL'],
147           marker=markers[0], markevery=500,
148           markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
149           markerfacecolor="None",
150           )
151
152    savename = '{}/1_3_SPLt.{}'.format(picdir, pictype)
153    SavePlot(savename)
154

```

```

155 _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
156 ax.plot(powspec['freq'], powspec['SPL'],
157         marker=markers[0], markevery=500,
158         markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
159         markerfacecolor="None",
160         )
161 ax.set_xscale('log')
162
163 savename = '{}/1_3_SPLf.{}'.format(picdir, pictype)
164 SavePlot(savename)
165
166 #####
167 ### 2.1 PLOT 1/3 OCTAVE-BAND SPL #####
168 #####
169
170 _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
171 ax.plot(octv3rd['freq'], octv3rd['SPL'],
172         marker=markers[0], markevery=500,
173         markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
174         markerfacecolor="None",
175         )
176 ax.set_xscale('log')
177
178 savename = '{}/2_1_SPLf_octv3rd.{}'.format(picdir, pictype)
179 SavePlot(savename)
180
181 #####
182 ### 2.2 PLOT OCTAVE-BAND SPL #####
183 #####
184
185 _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
186 ax.plot(octv['freq'], octv['SPL'],
187         marker=markers[0], markevery=500,
188         markeredgecolor=colors[0], markeredgewidth=MarkerWidth,
189         markerfacecolor="None",
190         )
191 ax.set_xscale('log')
192
193 savename = '{}/2_2_SPLf_octv.{}'.format(picdir, pictype)
194 SavePlot(savename)
195
196 #####
197 ### PLOT ALL OCTAVE-BAND SPL #####
198 #####
199
200 _,ax = PlotStart(None, 'Frequency (Hz)', 'SPL (dB)', figsize=[6, 6])
201
202 #Plot Overall SPL as Horizontal Line
203 xmin = min(powspec['freq'])
204 xmax = max(powspec['freq'])
205 ax.plot([xmin, xmax], [params['SPL_overall'], params['SPL_overall']],
206         label='Ovr', color='black', linestyle='--')
207
208 i = 1
209 #Plot Octave-Band
210 ax.plot(octv['freq'], octv['SPL'], label='Oct',
211         color=colors[i],
212         marker=markers[i], markevery=1,
213         markeredgecolor=colors[i], markeredgewidth=MarkerWidth,
214         markerfacecolor="None",
215         )
216

```

```

217     i += 1
218     #Plot 1/3 Octave-Band
219     ax.plot(octv3rd['freq'], octv3rd['SPL'], label='Oct$\frac{1}{3}$',
220            color=colors[i],
221            marker=markers[i], markevery=1,
222            markeredgewidth=MarkerWidth,
223            markerfacecolor="None",
224            )
225
226     #i += 1
227     i = 0
228     #Plot Narrow-Band
229     ax.plot(powspec['freq'], powspec['SPL'], label='Nar',
230            color=colors[i],
231            )
232
233     ax.set_xscale('log')
234     ax.set_xlim([xmin, xmax])
235     PlotLegend(ax)
236
237     #Overall SPL text box
238     text = '$L_P={0:.2f}$'.format(float(params['SPL_overall']))
239     TextBox(ax, text, x=0.55, y=0.94, alpha=0.4)
240
241     savename = '{} / 2_SPLf_all.{}'.format(picdir, pictype)
242     SavePlot(savename)
243
244
245
246 if __name__ == "__main__":
247
248
249
250
251     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

```

0  """HW1 - WRAPPER
1  Logan Halstrom
2  MAE 298 AEROACOUSTICS
3  HOMEWORK 1 - SIGNAL PROCESSING
4  CREATED: 04 OCT 2016
5  MODIFIY: 04 OCT 2016
6
7  DESCRIPTION: Run data processing and plotting programs with common inputs.
8  """
9
10 #IMPORT GLOBAL VARIABLES AND PROGRAMS TO WRAP
11 from hw1_98_globalVars import *
12 import hw1_00_process as process
13 import hw1_01_plot as plot
14
15
16 def main(source):
17     """input description
18     """

```

```

19
20     print('\nProcessing Data')
21     process.main(source)
22     print('\nPlotting Data')
23     plot.main()
24
25
26 if __name__ == "__main__":
27
28     Source = 'Boom_F1B2_6.wav'
29
30     main(Source)
31

```

Listing 3: *hw1\_99\_wrapper.py* - Wrapper program that runs data processing and plotting scripts simultaneously

## Appendix E: Global Variables

```

0  """HW1 - GLOBAL VARIABLES
1  Logan Halstrom
2  MAE 298 AEROACOUSTICS
3  HOMEWORK 1 - SIGNAL PROCESSING
4  CREATED: 04 OCT 2016
5  MODIFIY: 17 OCT 2016
6
7  DESCRIPTION: Provide global variables for all scripts including wrapper.
8  """
9
10 #DATA OVERWRITE SWITCHES
11 overwrite = 1
12
13 #LOAD/SAVE DIRECTORIES
14 datadir = 'Data'      #Source and processed data storage directory
15 savedir = 'Results' #
16
17 picdir = 'Plots' #Plot storage directory
18 pictype = 'png'      #Plot save filetype
19 pictype = 'pdf'      #Plot save filetype
20
21
22 #CONVERSIONS
23 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

```

0  """PYTHON PLOTTING UTILITIES
1  Logan Halstrom
2  07 OCTOBER 2015
3
4  DESCRIPTION: File manipulation, matplotlib plotting and saving. A subset of
5  lutil.py simply for plotting.
6  """
7
8  import subprocess

```

```

9 import os
10 import re
11 import matplotlib.pyplot as plt
12 import numpy as np
13
14 def MakeOutputDir(savedir):
15     """make results output directory if it does not already exist.
16     inststring --> directory path from script containing folder
17     """
18     #split individual directories
19     splitstring = savedir.split('/')
20     prestring = ''
21     for string in splitstring:
22         prestring += string + '/'
23         try:
24             os.mkdir(prestring)
25         except Exception:
26             pass
27
28 def GetParentDir(savename):
29     """Get parent directory from path of file"""
30     #split individual directories
31     splitstring = savename.split('/')
32     parent = ''
33     #concatenate all dirs except bottommost
34     for string in splitstring[:-1]:
35         parent += string + '/'
36     return parent
37
38 def GetFilename(path):
39     """Get filename from path of file"""
40     parent = GetParentDir(path)
41     filename = FindBetween(path, parent)
42     return filename
43
44 def NoWhitespace(str):
45     """Return given string with all whitespace removed"""
46     return str.replace(' ', '')
47
48 def FindBetween(str, before, after=None):
49     """Returns search for characters between 'before' and 'after' characters
50     If after=None, return everything after 'before'"""
51     # value_regex = re.compile('(?'<= ' + before + ')(?P<value>.*?)(?'>= '
52     #                                     + after + ')' )
53     if after==None:
54         match = re.search(before + '(.*)$', str)
55         if match != None: return match.group(1)
56         else: return 'No Match'
57     else:
58         match = re.search('(?'<= ' + before + ')(?P<value>.*?)(?'>= '
59                                     + after + ')', str)
60         if match != None: return match.group('value')
61         else: return 'No Match'
62
63
64 #####
65 ### PLOTTING #####
66 #####
67
68 xkcdcolors = ["windows blue", "dusty purple", "leaf green", "macaroni and cheese", "cherry", "greyish
69 xkcdhex =    ['#3778bf',          '#825f87',          '#5ca904',          '#efb435',          '#cf0234',          '#a8a495
70

```

```

71 def UseSeaborn(palette='deep'):
72     """Call to use seaborn plotting package
73     """
74     import seaborn as sns
75     #No Background fill, legend font scale, frame on legend
76     sns.set(style='whitegrid', font_scale=1.5, rc={'legend.frameon': True})
77     #Mark ticks with border on all four sides (overrides 'whitegrid')
78     sns.set_style('ticks')
79     #ticks point in
80     sns.set_style({"xtick.direction": "in", "ytick.direction": "in"})
81
82     # sns.choose_colorbrewer_palette('q')
83
84     #Nice Blue, green, Red
85     # sns.set_palette('colorblind')
86     if palette == 'xkcd':
87         #Nice blue, purple, green
88         sns.set_palette(sns.xkcd_palette(xkcdcolors))
89     else:
90         sns.set_palette(palette)
91     #Nice blue, green red
92     # sns.set_palette('deep')
93
94     # sns.set_palette('Accent_r')
95     # sns.set_palette('Set2')
96     # sns.set_palette('Spectral_r')
97     # sns.set_palette('spectral')
98
99     #FIX INVISIBLE MARKER BUG
100    sns.set_context(rc={'lines.markeredgewidth': 0.1})
101
102    #PLOT FORMATTING
103    # Configure figures for production
104    WIDTH = 495.0 # width of one column
105    FACTOR = 1.0 # the fraction of the width the figure should occupy
106    fig_width_pt = WIDTH * FACTOR
107
108    inches_per_pt = 1.0 / 72.27
109    golden_ratio = (np.sqrt(5) - 1.0) / 2.0 # because it looks good
110    fig_width_in = fig_width_pt * inches_per_pt # figure width in inches
111    fig_height_in = fig_width_in * golden_ratio # figure height in inches
112    fig_dims = [fig_width_in, fig_height_in] # fig dims as a list
113
114    #Line Styles
115    mark = 5
116    minimark = 0.75
117    line = 1.5
118
119    #dot, start, x, tri-line, plus
120    smallmarkers = ['.', '*', 'd', '1', '+']
121    bigmarkers = ['o', 'v', 'd', 's', '*', 'D', 'p', '>', 'H', '8']
122    scattermarkers = ['o', 'v', 'd', 's', 'p']
123
124    #GLOBAL INITIAL FONT SIZES
125    Ttl = 32
126    Lbl = 32
127    Box = 28
128    Leg = 28
129    Tck = 22
130
131    #MAKE FONT DICT GLOBAL SO IT CAN BE MADE AND USED IN DIFFERENT FUNCTIONS
132    global font_ttl, font_lbl, font_box, font_tck, font_leg

```



```

133
134 def SetFontDictSize(ttl=None, lbl=None, box=None, tck=None, leg=None):
135     """Set font size in styling dictionaries. Global dictionaries to use for
136     font styles in all functions.
137     To change a single parameter, call function like: SetFontDictSize(lbl=18)
138     ttl --> title, lbl --> axis label, box --> textbox,
139     tck --> axis tick labels, leg --> legend text
140     """
141     global font_ttl, font_lbl, font_box, font_tck, font_leg
142
143     #DEFAULT FONT SIZES
144     # if ttl == None: ttl = 18
145     # if lbl == None: lbl = 18
146     # if box == None: box = 12
147     # if tck == None: tck = 16
148     # if leg == None: leg = 16
149
150     if ttl == None: ttl = Ttl
151     if lbl == None: lbl = Lbl
152     if box == None: box = Box
153     if tck == None: tck = Tck
154     if leg == None: leg = Leg
155
156     #Font Styles
157     font_ttl = {'family' : 'serif',
158                'color' : 'black',
159                'weight' : 'normal',
160                'size' : ttl,
161                }
162     font_lbl = {'family' : 'serif',
163                'color' : 'black',
164                'weight' : 'normal',
165                'size' : lbl,
166                }
167     font_box = {'family' : 'arial',
168                'color' : 'black',
169                'weight' : 'normal',
170                'size' : box,
171                }
172     font_tck = tck
173     font_leg = leg
174
175     #INITIAL FONT DICT SETTINGS
176     SetFontDictSize()
177
178     #Textbox Properties
179     textbox_props = dict(boxstyle='round', facecolor='white', alpha=0.5)
180
181
182
183     params = {
184         # 'backend': 'ps',
185         # 'text.latex.preamble': ['\usepackage{gensymb}'],
186         'axes.labelsize' : Lbl,
187         'axes.titlesize' : Ttl,
188         'text.fontsize' : Box,
189         'legend.fontsize': Leg,
190         'xtick.labelsize': Tck,
191         'ytick.labelsize': Tck,
192         # 'text.usetex': True,
193         # 'figure.figsize': [fig_width,fig_height],
194         'font.family': 'helvetica'

```

```

195 }
196 import matplotlib
197 matplotlib.rcParams.update(params)
198
199
200
201
202
203 def PlotStart(title, xlabel, ylabel, horzy='vertical', figsize='square',
204               ttl=None, lbl=None, tck=None, leg=None, box=None,
205               grid=True, rc=False):
206     """Begin plot with title and axis labels. Space title above plot.
207     horzy --> vertical or horizontal y axis label
208     figsize --> set figure size. None for autosizing, 'tex' for latex
209                 formatting, or 2D list for user specification.
210     ttl, lbl, tck --> title, label, and axis font sizes
211     """
212
213     #SET FIGURE SIZE
214     if figsize == None:
215         #Plot with automatic figure sizing
216         fig = plt.figure()
217     else:
218         if figsize == 'tex':
219             #Plot with latex 2-column figure sizing
220             figsize = fig_dims
221         elif figsize == 'square':
222             figsize = [6, 6]
223         #Otherwise, plot with user-specified dimensions (i.e. [width, height])
224         fig = plt.figure(figsize=figsize)
225
226     #PLOT FIGURE
227     ax = fig.add_subplot(1, 1, 1)
228
229     if rc:
230         #USE MATPLOTLIB RC PARAMS SETTINGS
231         if title != None:
232             plt.title(title)
233         plt.xlabel(xlabel)
234         plt.ylabel(ylabel)
235     else:
236         #USE FONT DICT SETTINGS
237
238         #Set font sizes
239         if ttl != None or lbl != None or tck != None or leg != None or box != None:
240             #Set any given font sizes
241             SetFontDictSize(ttl=ttl, lbl=lbl, tck=tck, leg=leg, box=box)
242         else:
243             #Reset default font dictionaries
244             SetFontDictSize()
245
246         if title != None:
247             plt.title(title, fontdict=font_ttl)
248         plt.xlabel(xlabel, fontdict=font_lbl)
249         plt.xticks(fontsize=font_tck)
250         plt.ylabel(ylabel, fontdict=font_lbl, rotation=horzy)
251         plt.yticks(fontsize=font_tck)
252
253     #INCREASE TITLE SPACING
254     ttl = ax.title
255     ttl.set_position([.5, 1.025])
256     # ax.xaxis.set_label_coords( .5, -1.025*10 )

```

```

257     # ax.yaxis.labelpad = 20
258
259     #TURN GRID ON
260     if grid:
261         ax.grid(True)
262
263     return fig, ax
264
265 def MakeTwinx(ax, ylbl, horzy='vertical'):
266     """Make separate y-axis with label"""
267     ax2 = ax.twinx()
268     ax2.set_ylabel(ylbl, fontdict=font_lbl, rotation=horzy)
269     plt.yticks(fontsize=font_tck)
270     return ax2
271
272 def ZeroAxis(ax, dir='x'):
273     """Set axis lower bound to zero, keep upper bound
274     """
275     if dir == 'x':
276         ax.set_xlim([0, ax.get_xlim()[1]])
277     elif dir == 'y':
278         ax.set_ylim([0, ax.get_ylim()[1]])
279
280 def ZeroAxes(ax):
281     """Set both axes lower bound to zero, keep upper bound
282     """
283     ax.set_xlim([0, ax.get_xlim()[1]])
284     ax.set_ylim([0, ax.get_ylim()[1]])
285
286 def Plot(ax, x, y, color, label, linestyle='-',
287         marker='None', line=1.5, mark=5):
288     """Enter 'Default' to keep default value if entering values for later
289     variables"""
290     return ax.plot(x, y, color=color, label=label, linestyle=linestyle,
291                  linewidth=line, marker=marker, markersize=mark)
292
293 def PlotLegend(ax, loc='best', alpha=0.5, title=None, fontsize=None):
294     """General legend command. Use given handles and labels in plot
295     commands. Curved edges, semi-transparent, given title, single markers
296     """
297     if fontsize == None:
298         fontsize = font_leg
299     leg = ax.legend(loc=loc, title=title,
300                   fancybox=True, frameon=True, framealpha=alpha,
301                   numpoints=1, scatterpoints=1, prop={'size':fontsize},
302                   borderpad=0.1, borderaxespas=0.1, handletextpad=0.2,
303                   handlelength=1.0, labelspacing=0)
304     return leg
305
306 def PlotLegendLabels(ax, handles, labels, loc='best', title=None, alpha=0.5,
307                    fontsize=None):
308     """Plot legend specifying labels.
309     Curved edges, semi-transparent, given title, single markers
310     """
311     if fontsize == None:
312         fontsize = font_leg
313     leg = ax.legend(handles, labels, loc=loc, title=title,
314                   fancybox=True, frameon=True, framealpha=alpha,
315                   numpoints=1, scatterpoints=1, prop={'size':1},
316                   borderpad=0.1, borderaxespas=0.1, handletextpad=0.2,
317                   handlelength=1.0, labelspacing=0)
318     # leg.get_frame().set_alpha(alpha)

```

```

319     # for label in leg.get_texts():
320     #     label.set_fontsize('large')
321     return leg
322
323 def ColorMap(ncolors, colormap='jet'):
324     """return array of colors given number of plots and colormap name
325     colormaps: jet, brg, Accent, rainbow
326     """
327     cmap = plt.get_cmap(colormap)
328     colors = [cmap(i) for i in np.linspace(0, 1, ncolors)]
329     return colors
330
331 def ColorBar(label, horzy='horizontal', ticks=None, colorby=None, pad=25):
332     """Add colorbar with label.
333     ax --> matplotlib axis object
334     label --> colorbar label
335     horzy --> label orientation
336     ticks --> manually provide colorbar tick location (default is automatic)
337     colorby --> data to base colorbar on. Default is whatever was plotted last
338     pad --> spacing of label from colorbar. positive is further away
339     """
340     if colorby == None:
341         #MAKE COLORBAR
342         cb = plt.colorbar()
343     else:
344         #MAKE COLORBAR WITH GIVEN DATA
345         cb = plt.colorbar(colorby)
346     #SET LABEL
347     cb.set_label(label, rotation=horzy, fontdict=font_lbl, labelpad=pad)
348     return cb
349
350 def SavePlot(savename, overwrite=1, trans=False):
351     """Save file given save path. Do not save if file exists
352     or if variable overwrite is 1"""
353     if os.path.isfile(savename):
354         if overwrite == 0:
355             print('    Overwrite is off')
356             return
357         else: os.remove(savename)
358     MakeOutputDir(GetParentDir(savename))
359     plt.savefig(savename, bbox_inches='tight', transparent=trans)
360     # plt.close()
361
362 def ShowPlot(showplot=1):
363     """Show plot if variable showplot is 1"""
364     if showplot == 1:
365         plt.show()
366     else:
367         plt.close()
368
369 def GridLines(ax, linestyle='--', color='k', which='major'):
370     """Plot grid lines for given axis.
371     Default dashed line, black, major ticks
372     (use: 'color = pl.get_color()' to get color of a line 'pl')
373     """
374     ax.grid(True, which=which, linestyle=linestyle, color=color)
375
376 def TextBox(ax, boxtext, x=0.005, y=0.95, fontsize=font_box['size'],
377            alpha=0.5, props=None):
378     if props == None:
379         props = dict(boxstyle='round', facecolor='white', alpha=alpha)
380     ax.text(x, y, boxtext, transform=ax.transAxes, fontsize=fontsize,

```

```

381         verticalalignment='top', bbox=props)
382
383 def TightLims(ax, tol=0.0):
384     """Return axis limits for tight bounding of data set in ax.
385     NOTE: doesn't work for scatter plots.
386     ax --> plot axes to bound
387     tol --> whitespace tolerance
388     """
389     xmin = xmax = ymin = ymax = None
390     for line in ax.get_lines():
391         data = line.get_data()
392         curxmin = min(data[0])
393         curxmax = max(data[0])
394         curymin = min(data[1])
395         curymax = max(data[1])
396         if curxmin < xmin or xmin == None:
397             xmin = curxmin
398         if curxmax > xmax or xmax == None:
399             xmax = curxmax
400         if curymin < ymin or ymin == None:
401             ymin = curymin
402         if curymax > ymax or ymax == None:
403             ymax = curymax
404
405     xlim = [xmin-tol, xmax+tol]
406     ylim = [ymin-tol, ymax+tol]
407
408     return xlim, ylim
409
410 def PadBounds(axes, tol=0):
411     """Add tolerance to axes bounds to pad with whitespace"""
412
413     xtol = (axes[1] - axes[0]) * tol
414     ytol = (axes[3] - axes[2]) * tol
415     tols = [-xtol, xtol, -ytol, ytol]
416     for i, (x, t) in enumerate(zip(axes, tols)):
417         axes[i] += t
418     return axes
419
420 def VectorMark(ax, x, y, nmark, color='k'):
421     """Mark line with arrow pointing in direction of x+.
422     Show nmark arrows
423     """
424     n = len(y)
425     dm = int(len(y) / nmark)
426     # indices = np.linspace(1, n-2, nmark)
427     indices = [1]
428     while indices[-1]+dm < len(y)-1:
429         indices.append(indices[-1] + dm)
430
431     for ind in indices:
432         #entries are x, y, dx, dy
433         xbase, ybase = x[ind], y[ind]
434         dx, dy = x[ind+1] - x[ind], y[ind+1] - y[ind]
435         ax.quiver(xbase, ybase, dx, dy, angles='xy', scale_units='xy', scale=1)
436
437 def PlotArrow(ax, x1, y1, x2, y2, label, head1='<', head2='>',
438              color='grey', sz=10):
439     """Plot an arrow between two given points. Specify arrowhead type on
440     either side (default double-headed arrow).
441     ax --> plot axis object
442     x1,y1 --> x,y coordinates of starting point

```

```

443     x2,y2    --> x,y coordinates of ending point
444     label    --> label for legend
445     head1,2  --> first and second arrowheads (e.g. '<', '>', 'v', '^')
446     color    --> color of arrow
447     sz       --> size of arrowheads
448     """
449     #Plot line connecting two points
450     ax.plot([x1, x2], [y1, y2], color=color, label=label)
451     ax.plot(x1, y1, color=color, marker=head1, markersize=sz) #1st arrow head
452     ax.plot(x2, y2, color=color, marker=head2, markersize=sz) #2nd arrow head
453     return ax
454
455 def PlotVelProfile(ax, y, u, color='green', narrow=4):
456     """Plot velocity profile as y vs y
457     y --> non-dim. vertical grid within BL (y/delta)
458     u --> non-dim. x-velocity within BL (u/u_e)
459     color --> string, color of plot
460     narrow --> number of points between arrows
461     """
462     vertline = np.zeros(len(y))
463     ax.plot(vertline, y, color=color, linewidth=line)
464     ax.fill_betweenx(y, vertline, u, facecolor=color, alpha=0.2)
465     wd, ln = 0.03, 0.03
466     for i in range(0, len(y), narrow):
467         if abs(u[i]) < ln:
468             ax.plot([0, u[i]], [y[i], y[i]], color=color, linewidth=line)
469         else:
470             ax.arrow(0, y[i], u[i]-ln, 0, head_width=wd, head_length=ln,
471                     fc=color, ec=color, linewidth=line)
472     ax.plot(u, y, color=color, linewidth=line)
473     ax.axis([min(u), max(u), min(y), max(y)])
474
475 def PolyFit(x, y, order, n, showplot=0):
476     """Polynomial fit xdata vs ydata points
477     x --> independent variable data points vector
478     y --> dependent variable data points vector
479     order --> order of polynomial fit
480     n --> number of points in polynomial fit
481     showplot --> '1' to show plot of data fit
482     Returns:
483     function of polynomial fit
484     """
485     #New independent variable vector:
486     xmin, xmax = x[0], x[-1]
487     x_poly = np.linspace(xmin, xmax, n)
488     fit = np.polyfit(x, y, order)
489     polyfit = np.poly1d(fit)
490     y_poly = polyfit(x_poly)
491     #Plot Poly Fit
492     plt.figure()
493     plt.title(str(order) + '-Order Polynomial Fit', fontsize=14)
494     plt.xlabel('x', fontsize=14)
495     plt.ylabel('y', fontsize=14)
496     plt.plot(x, y, 'rx', label='Data')
497     plt.plot(x_poly, y_poly, 'b', label='Fit')
498     plt.legend(loc='best')
499     if showplot == 1:
500         plt.show()
501     return polyfit

```

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