

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 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:

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

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.

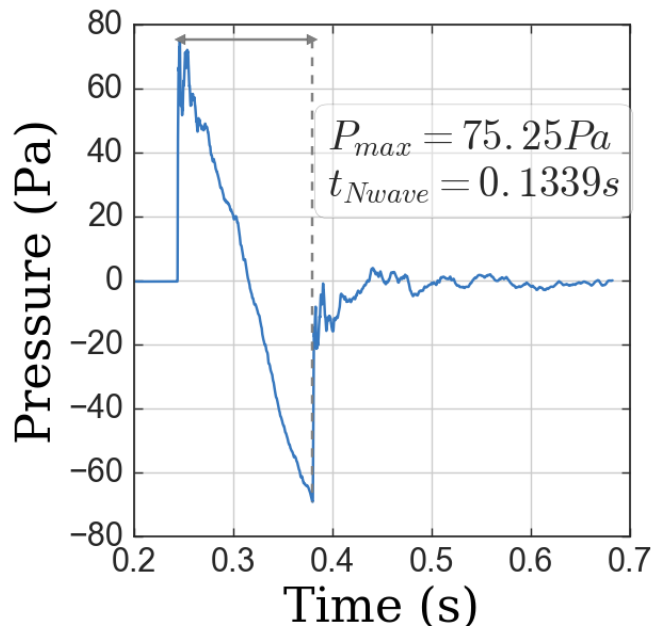


Fig. 1: Recorded sonic boom shockwave pressure time history with peak pressure and N-wave time duration values (Zero-pressure from recording start to initial shock)

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

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

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 to the frequency domain, and can be called in Python from 'numpy.fft'.

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

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

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

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

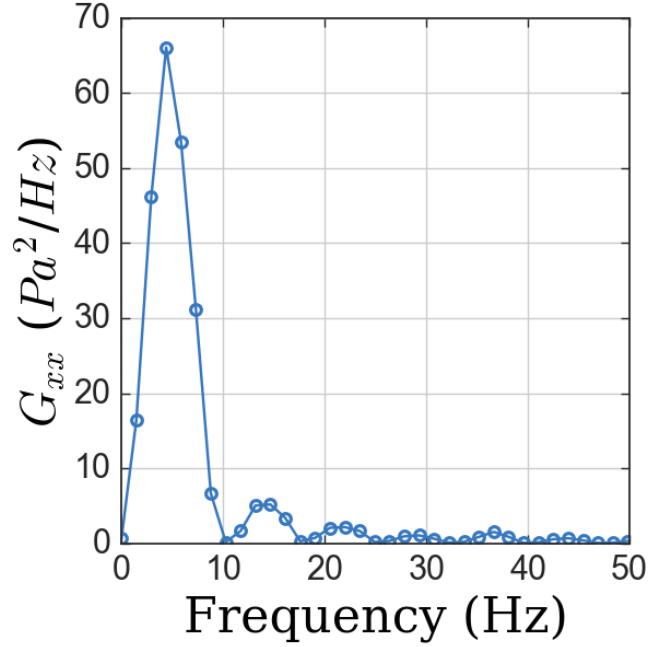


Fig. 2: Shockwave signal power spectral density as a function of frequency (All frequencies above 50Hz very low power)

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 $4.49 Hz$, 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 $P_{ref} = 20 \mu Pa$.

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

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 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.

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

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

results now, discuss/derive below

4.1 Problem 2.1 – $1/3$ Octave Bands

specifics on $1/3$

4.2 Problem 2.2 – Octave Bands

specifics on octave

4.3 Problem 2.3 – Overall Sound Pressure Level

overall

5 Conclusion

conclude

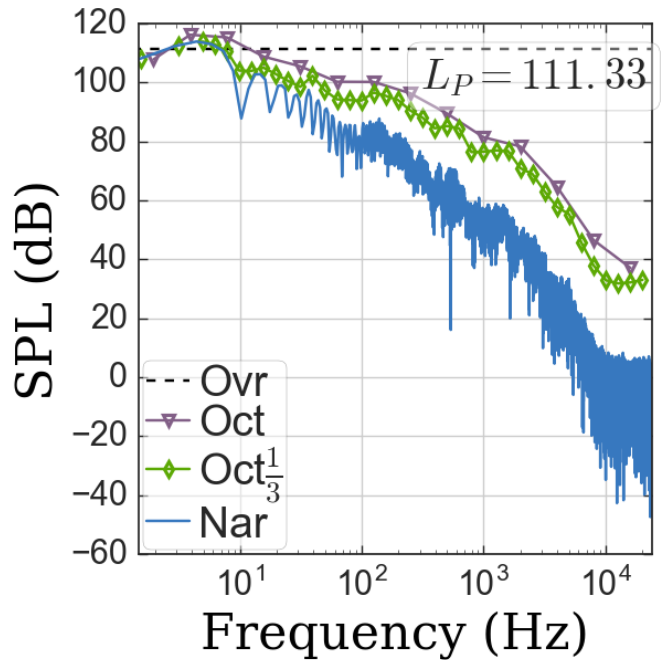


Fig. 3: Shockwave signal narrow-band, $\frac{1}{3}$ octave-band, and octave-band, with overall Sound Pressure Level reported in upper right

Appendix A: 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('Maximum Power Spectrum, frequency', maxima['freq'])
187 print('Maximum Power Spectrum, power', 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 #####
235 ### SAVE DATA #####
236 #####
237
238 #SAVE WAVE SIGNAL DATA
239 df = df[['time', 'Pa', 'SPL', 'V']] #reorder
240 df.to_csv( '{}/timespec.dat'.format(datadir), sep=' ', index=False ) #save
241
242 #SAVE POWER SPECTRUM DATA
243 powspec.to_csv( '{}/freqspec.dat'.format(datadir), sep=' ', index=False )
244
245 #SAVE OCTAVE-BAND DATA
246 octv3rd.to_csv( '{}/octv3rd.dat'.format(datadir), sep=' ', index=False)

```

```

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