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 Introduction

Give overview of homework and background concepts

2 Read Data

list functions used to read data, how python/matlab compare.

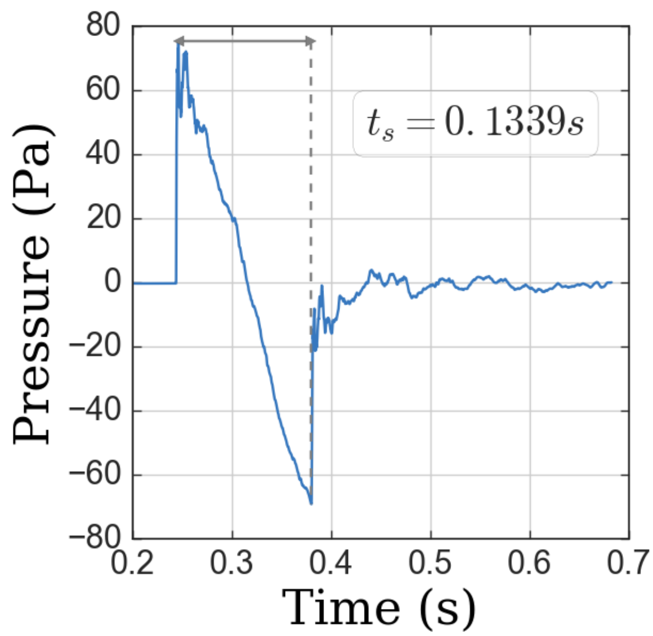

Fig. 1: Recorded sonic boom shockwave pressure time history in characteristic high-low pressure N-wave shape (Zero-pressure from recording start to initial shock)

3 Frequency Domain

decompose into frequency domain with FFT

3.1 Power Spectral Density Decomposition

power spectrum density decomposition stuff

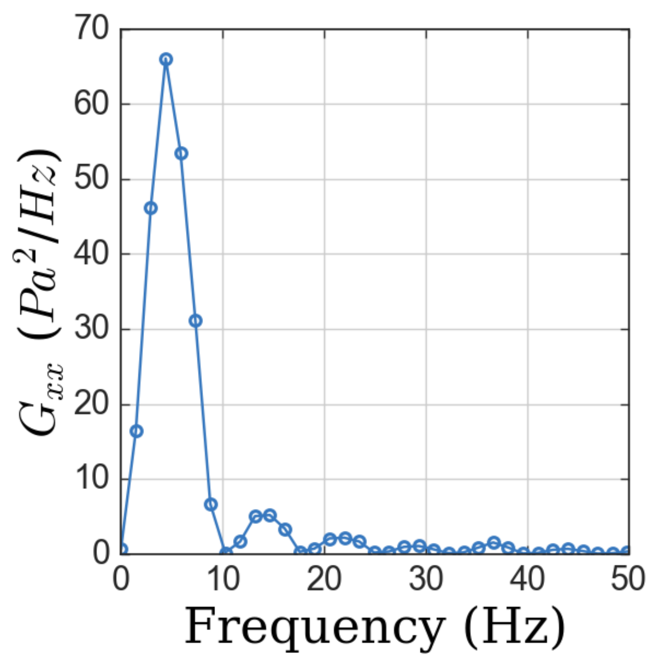

Fig. 2: Shockwave signal power spectral density as a function of frequency (All frequencies above 50Hz very low power)

3.2 Sound Pressure Level

this is actually in the plot in the next section

4 Octave-Band Spectra

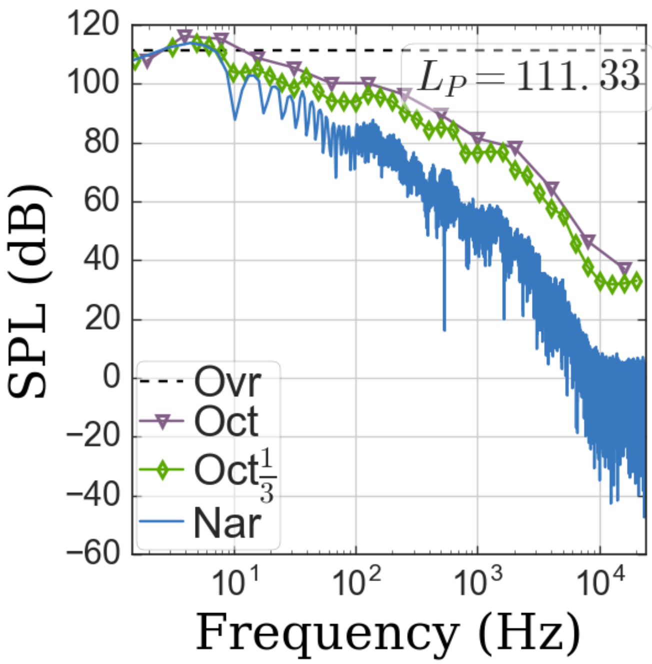

Fig. 3: Shockwave signal narrow-band, $\frac{1}{3}$ octave-band, and octave-band, with overall Sound Pressure Level reported in upper right

5 Conclusion

conclude

Appendix A: Data Processing Script

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

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

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

#IMPORT GLOBAL VARIABLES
from hw1_98_globalVars import *

import numpy as np
import pandas as pd

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

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

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

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

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

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

def 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 OctaveCenterFreqsGen(dx=3, n=39):
    """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 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 frequency.
    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
    fcs = OctaveCenterFreqs(df['freq'], octv)
```

```python
    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
    ### 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 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'])

    ####################################################################
    ### 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
    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, 'tNwave' : dt_Nwave,
         'ti' : ti, 'Pi' : Pi, 'tf' : tf, 'Pf' : Pf}
        ), ignore_index=True)
    params.to_csv( '{}/params.dat'.format(datadir), sep=' ', index=False)
```

```python


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