Python Open Acoustics

Jason Krasavage

May 9, 2018

1 Introduction

Python Open Acoustics is a module created for Python 3 that is tailored towards not only acousticians that need to do quick and accurate calculations, but the average, tech-savvy individual who may need specific acoustic calculations to help themselves legally in certain situations. This module aims to make doing these calculations as easy as possible. With a little coding experience (in Python nonetheless, one of the easiest to learn high-level programming languages out there today), anyone can successfully use this module.

2 Components

Python Open Acoustics is comprised of functions. These functions are divided into the following categories.

1. Acoustic Units
2. Inverse Laws
3. Decibel Calculations
4. Decibel Weightings
5. Spectrum Calculations
6. Noise Descriptors

3 Examples

3.1 A-Weighted dB Calculation

Given a scenario where you are able to find the unweighted dB levels of a sound source for each band of an octave-band spectrum, using *Python Open Acoustics* you can easily find both the unweighted, a-weighted, and c-weighted dB value of this spectrum. For the purpose of this example we will look at the a-weighted sum.

The first step is to use a Python list to store your 10 band values. To do so, assign a new variable to a list of the 10 values.

```
values = [50, 60, 63, 64, 66, 66, 74, 73, 69, 65]
```

Next, use the `a_weighting()` function and pass in your new list as an argument, while assigning this to a new variable.

```
a_weighted_values = a_weighting(values)
```

Lastly, use the module's db_add() function to sum the levels of this spectrum into a single, now a-weighted, dB value. If you'd like you can assign this to a new variable, then print that variable's value. Additionally, you can just call the method and the value will be displayed for you.

```
a_weighted_value = db_add(a_weighted_values)
print(a_weighted_value)
>>> 78.5
```

3.2 Inverse Distance Law

Often times it is useful to know the dB SPL of a sound source from a certain distance away. In this type of situation there would already be a dB SPL value that is measure from a certain known distance. Using the inverse_distance_law() function, this new dB SPL value can easily be calculated.

Let's say a tornado siren was measured to be 108 dB SPL at a distance of 20 meters, and you wanted to know the dB SPL at 150 meters. You could execute this calculation very easily.

```
new_dBSPL = inverse_distance_law(108, 20, 150)
print(new_dBSPL)
>>> 90.498774732166
```

Additionally, this function works in both directions. If you were to measure the same siren at 150 meters, as 90.5 dB SPL, and you wanted to know the dB SPL value at 20 meters, you can do that as well. The arguments for this function, in order, are *dB SPL, old distance*, and *new distance*.

```
new_dBSPL = inverse_distance_law(90.5, 150, 20)
print(new_dBSPL)
>>> 108.001225267834
```

3.3 Time Weighted Average

A Time Weighted Average (TWA) is a common noise descriptor used to calculate a worker's daily exposure to noise during an average 8-hour work day. The TWA() function in Python Open Acoustics calculates this descriptor using two Python lists of the exposure times (in minutes) and their corresponding levels (in dBA). Each index of the lists must correspond to one another (e.g. the first index in both must be the exposure time and it's matching dBA level). For example, if you measure the exposure times to be 10, 20, and 45, and then measures their corresponding dBA values to be 86, 90, 93, you would simply do the following.

```
time_weighted_average = TWA([10, 20, 45],[86, 90, 93])
print(time_weighted_average)
>>> 83.77260264417602
```

The TWA() function uses the daily noise dose function(dose()) which in turn, uses the recommended exposure time function (recExposureTime()). The recExposureTime() function returns the NIOSH recommendation for exposure time (in minutes) for a given dBA value. The dose() function returns the daily noise dose (which is a percentage) for a given pair of lists for the exposure times and corresponding levels (in that order), and it is necessary for calculating the TWA (but the TWA() function does this for you automatically).

4 Python Source Code

```
""" OPEN ACOUSTICS """
```

```
'''Imports'''

#import math
from numpy import sqrt #for speed of sound
from numpy import pi #for angular wavenumber and angular frequency
from numpy import log10 #for decibel sound intensity, Leq, and SEL
from numpy import asarray #for Leq/SEL
from numpy import array #for a weighting
import numpy as np


''''Units'''

def period(frequency):
    return float(1/frequency)
    #unit is seconds

def frequency1(period):
    return float(1/period)
    #unit is hertz

def frequency2(wavelength, degreesCelsius = 15):
    return speed_of_sound(degreesCelsius)/wavelength
    #unit is hertz

def speed_of_sound(degreesCelsius):
    return 20.05 * sqrt(273.75 + degreesCelsius)
    #unit is m/s, and medium is air (do one for gas)

def wavelength(frequency, degreesCelsius = 15):
    return speed_of_sound(degreesCelsius)/frequency
    #unit is meters

def wavenumber(wavelength):
    return 1/wavelength
    #unit is cycles per meter

def angular_wavenumber(wavelength):
    return (2*pi)/wavelength
    #unit is radians per meter

def angular_frequency1(period):
    return (2*pi)/period

def angular_frequency2(frequency):
    return (2*pi)*frequency


'''Inverse Square & Distance Laws'''

#returns the dB SPL value at the new distance for the given sound source, distance is in
↪  meters
```

```
def inverse_distance_law(dBSPL,oldDistance, newDistance):
    return dBSPL+20*log10(oldDistance/newDistance)


'''Decibel'''

def sound_intensity_level(intensity, I0 = 10**(-12)):
    return 10*log10(intensity/I0)
    #unit is dB intensity (SIL)

def sound_pressure_level(pressure, P0 = 2*(10**(-5))):
    return 20*log10(pressure/P0)
    #unit is dB pressure (SPL)

def sound_power_level(watts, W0 = 10**(-12)):
    return 10*log10(watts/W0)
    #unit is dB power (SWL)

def db_add(levels):
    L = asarray(levels)
    return (10*log10(sum(10**(L/10))))


'''Decibel Weightings'''
#convert between dbA and db


def a_weighting(spectrum):
    spectrum_array = asarray(spectrum)
    single_corrections = array([-39.4, -26.2, -16.1, -8.6,
                                -3.2, 0, 1.2, 1, -1.1, -6.6])
    third_corrections = array([-56.7, -50.5, -44.7, -39.4,
                               -34.6, -30.2, -26.2, -22.5,
                               -19.1, -16.1, -13.4, -10.9,
                               -8.6, -6.6, -4.8, -3.2, -1.9,
                               -0.8, 0, 0.6, 1.0, 1.2, 1.3,
                               1.2, 1.0, 0.5, -0.1, -1.1,
                               -2.5, -4.3, -6.6])
    if len(spectrum) > 10:
        #1/3 octave band
        return (spectrum_array + third_corrections)
    else:
        #1/1 octave band
        return (spectrum_array + single_corrections)

def c_weighting(spectrum):
    spectrum_array = asarray(spectrum)
    single_corrections = array([-3.0, -0.8, -0.2, 0.0, 0.0,
                                0.0, -0.2, -0.8, -3.0, -8.5])
    third_corrections = array([-8.5, -6.2, -4.4, -3.0, -2.0
                               -1.3, -0.8, -0.5, -0.3, -0.2
                               -0.1, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
                                0.0, 0.0, 0.0, 0.0, -0.1, -0.2,
                                -0.3, -0.5, -0.8, -1.3, -2.0,
                                -3.0, -4.4, -6.2, -8.5])
    if len(spectrum) > 10:
        #1/3 octave band
        return (spectrum_array + third_corrections)
    else:
        #1/1 octave band
        return (spectrum_array + single_corrections)


'''Spectrum'''

#returns a list of the center frequencies in the range of human hearing (20hz - 20kHz)
def octave_band_centers(low_fc = 31.250):
    centers, center = [low_fc], low_fc
    while centers[-1] < 10000:
        center *= 2
        centers.append(round(center))
    return centers

def third_octave_band_centers(low_fc = 15.625):
    centers, center = [low_fc], low_fc
    while centers[-1] < 16000:
        center *= 2**(1/3)
        centers.append(round(center, 2))
    return centers


'''Descriptors'''

def recExposureTime(level): #returns the recommended exposure times for an array of dBA
↪   values (minutes)
    return 480/(2**((level-85)/3))

def dose(exposureTimes, levels): #returns the daily noise dose for arrays of both actual
↪   exposure times and their corresponding dBA levels)
    C, L = asarray(exposureTimes), asarray(levels)
    return 100*np.sum(C/recExposureTime(L))

def TWA(exposureTimes, levels): #returns the time weighted average for two corresponding
↪   lists of exposure times and levels
    C, L = asarray(exposureTimes), asarray(levels)
    return 10*log10((dose(C,L))/100)+85

def Leq(times, levels):
    T, L = asarray(times), asarray(levels)
    return 10*log10(sum(T*10**(L/10))/sum(T))

def SEL(times, levels):
    T, L = asarray(times), asarray(levels)
    return 10*log10(sum(T*10**(L/10)))
```

```
def testsel(times, levels):
    T, L = asarray(times), asarray(levels)
    return Leq(T, L) + (10 * log10(sum(T)))

def Ldn(daylevels, daytimes, nightlevels, nighttimes):
    day, night = Leq(daylevels, daytimes), Leq(nightlevels, nighttimes)
    return 10*log10((16*(10**(day/10))+8*(10**((night+10)/10)))/24)

def Lden(daylevels, daytimes, eveninglevels, eveningtimes, nightlevels, nighttimes):
    day, evening, night = Leq(daylevels, daytimes), Leq(eveninglevels, eveningtimes),
    ↪  Leq(nightlevels, nighttimes)
    return
    ↪  10*log10((12*(10**(day/10))+4*(10**((evening+5)/10))+8*(10**((night+10)/10)))/24)
```