Hearing Protection Effectiveness Testing

Goal

Outr goal here is to determine whether a hearing protection device (HPD) is effective and, evaluate it's effectiveness at attenuating sound.

Overview

The algorithms for the testing of HPD are laid out in CSA Z94.2-14, section 9.6. The result is one number—the ***Noise Reduction Rating*** (NRR)—which describes the device's efficacy.

The test procedure is to play pink noise of a known level (100dB) at a dummy head wearing a set of the HPDs, and compare this to the noise level at an unprotected ear. The incoming signals are seperated into octave bands for analysis, and the noise levels are both A- and C-weighted to perform the calculations. The mean and standard deviation of multiple trials is needed to get an accurate result. This project will involve using the Fourier transform, windowing, filtering and separating the FFT into octave bands.

In [1]:
```
import numpy as np
from numpy import fft
import scipy.io.wavfile
import matplotlib.pyplot as plt
from scipy import signal

# Misc Global Funcions
def normalize(sig):
    return sig/np.max(np.abs(sig))
def getRMS(sig):
    return np.sqrt(1/float(len(sig)) * np.sum(sig**2))
def db(x, xref=None):
    x = abs(x)
    if xref:
        return 20 * np.log10(x/xref)
    return 20 * np.log10(x)
```

Octave FIlter

Here we make an octave filter bank, configurable to the fraction of octave desired (i.e. for a third-octave bank, use `B=3`) and with a variable range and centre frequency.

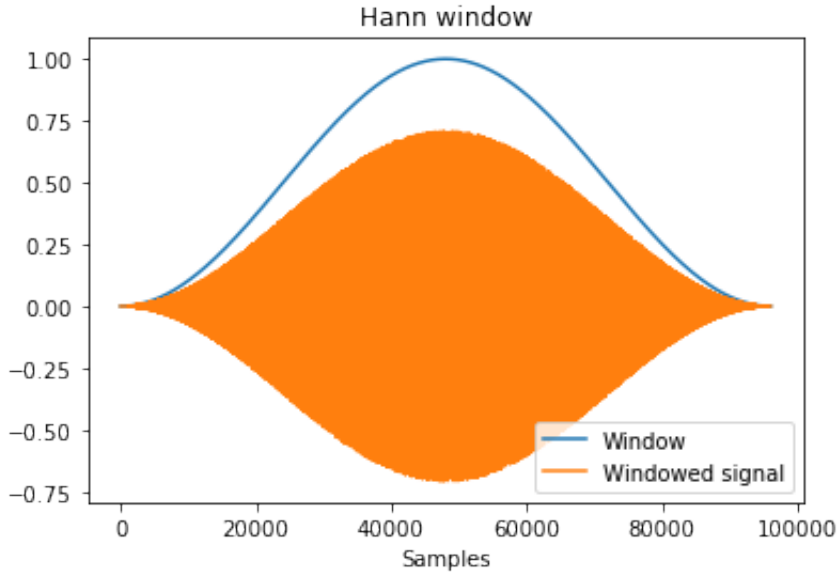
In calculating the NRR, we really only need single-octave filters though.

In [2]:
```
def OctaveBank(B, ny=None, octavesDown=None, octavesUp=None, centerFreq=N
    if not(ny):
        ny = float(20000)
    if (octavesDown and octavesUp):
        K = range(-octavesDown*B, octavesUp*B)
    else:
        K = range(-4*B, 4*B)
    if (centerFreq):
        f0 = centerFreq
    else:
        f0 = 1000
    Bank = []
    Fc = []
    for k in K:
        fc = f0*2**((k)/float(B))
        Fc.append(fc)
        fl = fc/float(2**(1/float(2*B)))
        fu = fc*(2**(1/float(2*B)))
        [b, a] = signal.butter(2, [fl/ny, fu/ny], btype='band')
        Bank.append([b,a])
    return Bank, Fc

def ApplyFilterBank(Bank, x):
    y = [None] * len(Bank)
    for i, filt in enumerate(Bank):
        y[i] = (signal.filtfilt(filt[0], filt[1], x))
    return y
```

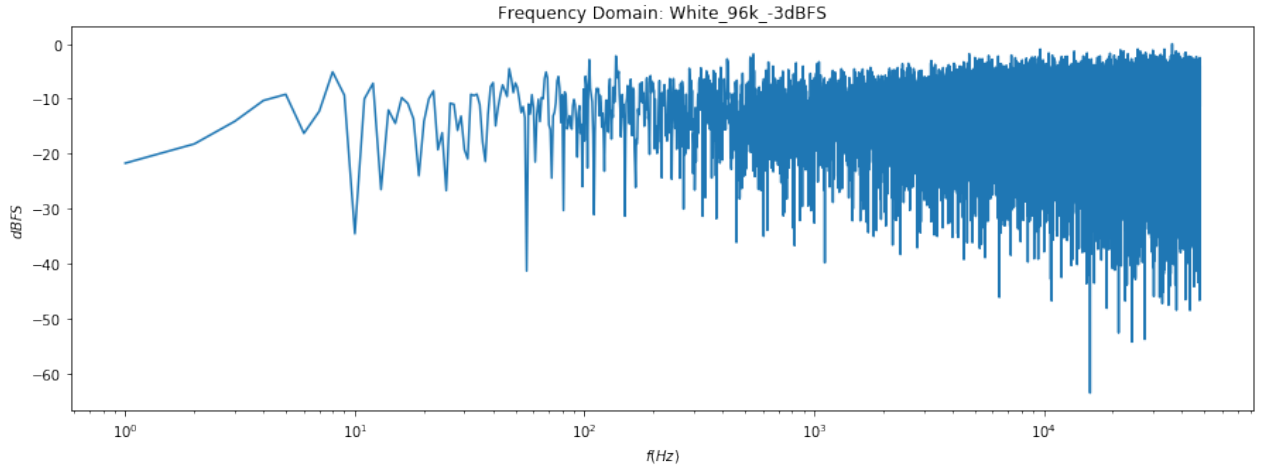
Testing with White Noise

To show that the Octave filter bank is working, we'll find the octave power spectrum of a white noise signal. The expected result is a linearly decreacing function. I'll also plot the FFT of the test signal, which outputs a fairly messy result, since no windowing has been performed.

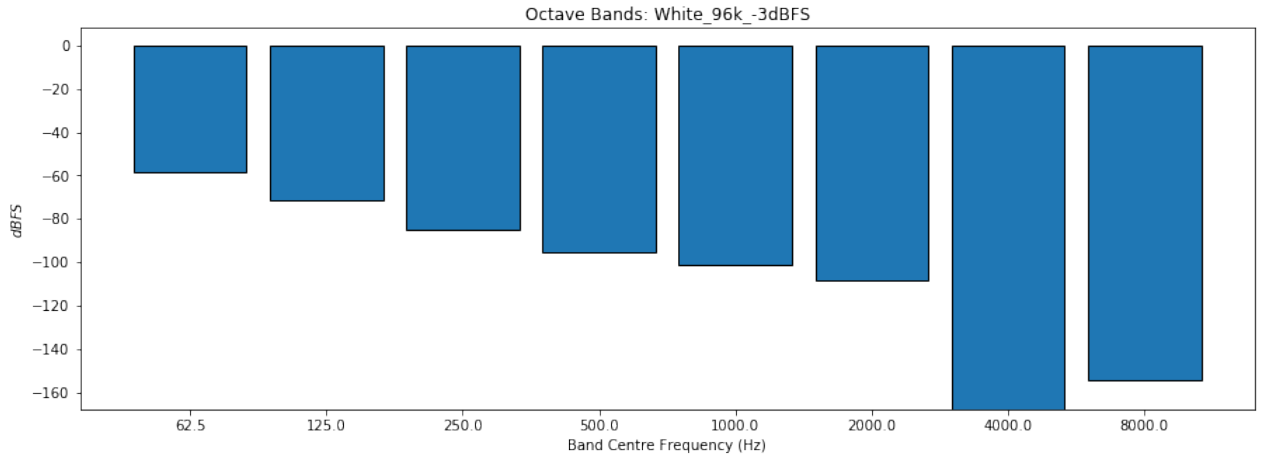
In [22]:
```
# Load the white noise file
filename = 'White_96k_-3dBFS'
[fs, whitenoise] = scipy.io.wavfile.read('recordings/' + filename + '.wav
Ny = fs/2
L = int(len(whitenoise))
whitenoise = whitenoise/float(2**15)

# Now we'll apply a hann window to the
frez = 1 # let's set our frequency resolution in this FFT
hann = signal.hanning(int(fs/frez), sym=False)
Lh = len(hann)

# Apply a hann window in the middle of the test file.
whitenoise = whitenoise[L/2 - Lh/2 : L/2 + Lh/2]
whitenoiseHann = whitenoise * hann


plt.figure(figsize=(6,4))
plt.plot(hann, label="Window")
plt.plot(whitenoiseHann, label="Windowed signal")
plt.title("Hann window")
plt.xlabel("Samples")
plt.legend(loc="best")
plt.show();
```



In [29]:
```
WHITENOISE = np.fft.rfft(whitenoiseHann)
freq = np.linspace(0, Ny, Lh/2+1)

plt.figure(figsize=(15,5))
plt.title("Frequency Domain: " + filename)
plt.semilogx(freq, db(WHITENOISE/np.max(np.abs(WHITENOISE))))
plt.xlabel("$f (Hz)$")
plt.ylabel('$dBFS$')
plt.show();
```



In [24]:
```
Bank, Fc = OctaveBank(1, ny=Ny)    # Create the 1-octave filter bank

y = ApplyFilterBank(Bank, whitenoise)
P = np.zeros(len(Bank))
for i in range(0,len(Bank)):
    P[i] = (np.sum(y[i])**2)/L

plt.figure(figsize=(15,5))
plt.title("Octave Bands: " + filename)
plt.bar(np.log10(Fc), db(P), width=0.25, edgecolor="k", tick_label=Fc)
plt.xlabel("Band Centre Frequency (Hz)")
plt.ylabel('$dBFS$')
plt.show();
```



With the exception of the highest octave, it looks like our octave bank works.