

## **Psycho-acoustic-modelDFT\_gs**

**This module offers you to process an audio signal(wave audio) with psycho-acoustic model and also shows live plot of spectrum where the user can vary the Masking Threshold and observe the change in audio output due to change in the value of masking threshold.**

- Please note that this model is similar to module 'psyacmodel' plus it offers additional features to play around with it. But it differs in implementation from programming point of view. Here all the functions are listed inside class which are static(similar static methods in JAVA/static functions in C++),

### **Importing relevant modules.**

```
In [ ]: __author__ = 'S.I. Mimalakis, G. Schuller'
         __copyright__ = 'MacSeNet, TU Ilmenau'
         #Using original DFT for signal and noise, not magnitude and phase

         import sys, os
         current_dir = os.path.dirname(os.path.realpath(__file__))
         print(current_dir + '/../')
         sys.path.insert(0, current_dir + '/../')
         import IOMethods as IO
         #import scipy.io.wavfile as wav
         import numpy as np
         #import numpy.fft
         #from scipy.fftpack import fft, ifft
         from numpy.fft import fft, ifft
         import matplotlib.pyplot as plt
         import pyaudio
         import wave
         import time
         import pygame
```

**Bundling all the functions required for the psycho-acoustic model in a class.**

```
In [ ]: class PsychoacousticModel:  
    """ Class that performs a very basic psychoacoustic model.  
    Bark scaling is based on Perceptual-Coding-In-Python, [Online] :  
        https://github.com/stephencwelch/Perceptual-Coding-In-Python  
    """
```

## Constructor to initialize class variables

```
In [ ]: def __init__(self, N = 4096, fs = 44100, nfilters=24, type = 'rasta', width = 1.0, minfreq=0,
maxfreq=22050):

    self.nfft = N
    from DFT
    self.fs = fs
    self.nfilters = nfilters

    self.width = width
    self.min_freq = minfreq
    self.max_freq = maxfreq
    self.max_freq = fs/2
    self.nfreqs = N/2
s (e.g. from DFT)
    self._LTeq = np.zeros(nfilters, dtype = np.float32) # Initialization for the absolute threshold

    # Type of transformation
    self.type = type

    # Computing the matrix for forward Bark transformation
    self.W = self.mX2Bark(type)

    # Computing the inverse matrix for backward Bark transformation
    self.W_inv = self.bark2mX()

    # Non-linear superposition parameters
    #Set alpha below for suitable exponents for non-linear superposition!
    #After Baumgarten: alpha = 0.3 for power, hence 2*0.3=0.6 for our "voltage":
    self._alpha = 0.8 # Exponent alpha
    self._maxb = 1./self.nfilters # Bark-band normalization
    #self._fa = 1./(10 ** (14.5/20.) * 10 ** (12./20.)) # Simultaneous masking for tones at Bark ban
d 12
    self._fadB= 14.5+12
    self._fa = 1./(10 ** ((self._fadB)/20.)) # shorter...
    self._fbdb=7.5
    self._fb = 1./(10**((self._fbdb/20.))) # Upper slope of spreading function
    self._fbbdb=26.0
    self._fbb = 1./(10**((self._fbbdb/20.))) # Lower slope of spreading function

    self._fd = 1./self._alpha # One over alpha exponent

    self.spreadingfuncmatrix=self.spreadingfunctionmat(self.max_freq)
```

**Computes a matrix with shifted spreading functions in each column, in the Bark scale.  
Including the alpha exponent for non-linear superposition.**

```
In [ ]: def spreadingfunctionmat(self,maxfreq):
    #computes a matrix with shifted spreading functions in each column, in the Bark scale.
    #including the alpha exponent for non-Linear superposition
    maxbark=self.hz2bark(maxfreq)
    spreadingfunctionBarkdB=np.zeros(2*self.nfilts)

    #upper slope, fbdB attenuation per Bark, over maxbark Bark (full frequency range), with fadB dB simultaneous masking:
    spreadingfunctionBarkdB[0:self.nfilts]=np.linspace(-maxbark*self._fbdb,-2.5,self.nfilts)-self._fadB
    #Lower slope fbbdb attenuation per Bark, over maxbark Bark (full frequency range):
    spreadingfunctionBarkdB[self.nfilts:2*self.nfilts]=np.linspace(0,-maxbark*self._fbbdb,self.nfilts)-self._fadB
    #print "spreadingfunctionBarkdB=", spreadingfunctionBarkdB
    #Convert from dB to "voltage" and include alpha exponent
    spreadingfunctionBarkVoltage=10.0**((spreadingfunctionBarkdB/20.0*self._alpha))
    #print "spreadingfunctionBarkVoltage=", spreadingfunctionBarkVoltage
    #Spreading functions for all bark scale bands in a matrix:
    spreadingfuncmatrix=np.zeros((self.nfilts,self.nfilts))

    for k in range(self.nfilts):
        spreadingfuncmatrix[:,k]=spreadingfunctionBarkVoltage[(self.nfilts-k):(2*self.nfilts-k)]
        #print "spreadingfuncmatrix= ", spreadingfuncmatrix
    return spreadingfuncmatrix
```

**Method to perform the transformation.**

```
In [ ]: def mX2Bark(self, type):
        """ Method to perform the transformation.
        Args :
            type : (str)      String denoting the type of transformation. Can be either
                            'rasta' or 'peaq'.
        Returns :
            W   : (ndarray)   The transformation matrix.

        """
        if type == 'rasta':
            W = self.fft2bark_rasta()
        elif type == 'peaq':
            W = self.fft2bark_peaq()
        else:
            assert('Unknown method')

        return W
```

**Method construct the weight matrix.**

```
In [ ]: def fft2bark_peaq(self):
    """ Method construct the weight matrix.
    Returns :
        W : (ndarray) The transformation matrix, used in PEAQ evaluation.
    """

    # Acquire a Local copy of the necessary variables
    nfft = self.nfft
    nfilts = self.nfilts
    fs = self.fs

    # Acquire frequency analysis
    df = float(fs)/nfft

    # Acquire filter responses
    fc, fl, fu = self.CB_filters()

    W = np.zeros((nfilts, nfft))

    for k in range(nfft/2+1):
        for i in range(nfilts):
            temp = (np.amin([fu[i], (k+0.5)*df]) - np.amax([fl[i], (k-0.5)*df])) / df
            W[i,k] = np.amax([0, temp])

    return W
```

## Method construct the weight matrix.

In [ ]:

```

def fft2bark_rasta(self):
    """ Method construct the weight matrix.
    Returns :
        W : (ndarray)      The transformation matrix, used in PEAQ evaluation.
    """
    minfreq = self.min_freq #default: 0Hz
    maxfreq = self.max_freq #default: 22050 Hz
    nfilts = self.nfilts #default: 24, from call: 64
    nfft = self.nfft #default: 4096, from call: 2048
    fs = self.fs
    width = self.width #default: 1.0

    min_bark = self.hz2bark(minfreq) #default: 0.0
    nyqbark = self.hz2bark(maxfreq) - min_bark #Bark range, for instance 24
    print "nyqbark= ", nyqbark

    if (nfilts == 0):
        nfilts = np.ceil(nyqbark)+1
    #from call: nfilts=64

    W = np.zeros((nfilts, nfft))
    #from call: 64x2048

    # Bark per bark-scale band
    step_barks = nyqbark/(nfilts-1) #from call: 24.0/63

    # Frequency of each FFT bin in Bark, in 1025 frequency bands (from call)
    binbarks = self.hz2bark(np.linspace(0,(nfft/2),(nfft/2)+1)*fs/nfft)

    f_bark_mid=np.zeros(nfilts)

    for i in xrange(nfilts):

        f_bark_mid[i] = min_bark + (i)*step_barks #from call: 0+(i)*24.0/63

        # Compute the absolute threshold from Hz to dB:
        #f=self.bark2hz(f_bark_mid[i] )
        f=min_bark + (i)*step_barks #from call: 0+(i)*24.0/63
        if f<4000.0:
            f=self.bark2hz(min_bark + (i+1)*step_barks )
            #declining slope, minimum ist at boundary of next Bark domain subband:
            self._LTeq[i] = min((3.64*(f/1000.)**-0.8-6.5*np.exp(-0.6*(f/1000.-3.3)**2.))+1e-3*
((f/1000.)**4.)), 60)

```

```

    else:
        #increasing slope, Lower Bark domain boundary is minimum:
        self._LTeq[i] = min((3.64*(f/1000.)**-0.8-6.5*np.exp(-0.6*(f/1000.-3.3)**2.))+1e-3*((f/1000.)**4.)), 60)
        #LTQ=np.clip((3.64*(f/1000.)**-0.8 -6.5*np.exp(-0.6*(f/1000.-3.3)**2.))+1e-3*((f/1000.)**4.)), -20, 60)

        """
        # Linear slopes in Log-space (i.e. dB) intersect to trapezoidal window
        lof = np.add(binbarks, (-1*f_bark_mid - 0.5))
        #print "Lof= ", lof
        hif = np.add(binbarks, (-1*f_bark_mid + 0.5))
        #print "hif=", hif
        #lower half of FFT spectrum is assigned, Looks like spreading functions already applied:
        #W[i,0:(nfft/2)+1] = 10**((np.minimum(0, np.minimum(np.divide(hif,width), np.multiply(lof,-2.5/width))))))

        """
        W[i,0:(nfft/2)+1] = (np.round(binbarks/step_barks)== i)
        print "self.bark2hz(f_bark_mid)= ", self.bark2hz(f_bark_mid)
        plt.plot(self.bark2hz(f_bark_mid), self._LTeq)
        plt.title('Masking Threshold in Quiet')
        plt.show()
        #print "W.shape= ", W.shape
        #print "W= ", W
        #print "self._LTeq", self._LTeq

    return W

```

**Method construct the inverse weight matrix, to map back to FT domain.**

```
In [ ]: def bark2mX(self):
    """ Method construct the inverse weight matrix, to map back to FT domain.
    Returns :
        W      : (ndarray)    The inverse transformation matrix.
    """
    # Fix up the weight matrix by transposing and "normalizing"
    """
    W_short = self.W[:,0:self.nfreqs + 1]
    WW = np.dot(W_short.T,W_short)

    WW_mean_diag = np.maximum(np.mean(np.diag(WW)), sum(WW,1))
    print "WW_mean_diag=", WW_mean_diag
    WW_mean_diag = np.reshape(WW_mean_diag,(WW_mean_diag.shape[0],1))
    W_inv_denom = np.tile(WW_mean_diag,(1,self.nfilts))

    W_inv = np.divide(W_short.T, W_inv_denom)
    """

    W_inv= np.dot(np.diag((1.0/np.sum(self.W,1))**0.5), self.W[:,0:self.nfreqs + 1]).T

    #print "W_inv.shape=", W_inv.shape
    #print "W_inv=", W_inv

    return W_inv
```

## Method to compute Bark from Hz. Based on :

<https://github.com/stephencwelch/Perceptual-Coding-In-Python> (<https://github.com/stephencwelch/Perceptual-Coding-In-Python>)

```
In [ ]: def hz2bark(self, f):
    """ Method to compute Bark from Hz. Based on :
    https://github.com/stephencwelch/Perceptual-Coding-In-Python
    Args :
        f      : (ndarray)    Array containing frequencies in Hz.
    Returns :
        Brk   : (ndarray)    Array containing Bark scaled values.
    """
    Brk = 6. * np.arcsinh(f/600.)                                # From RASTA, Dan Ellies method (for speech quality)
    # An inverse of that can be computed.
    #Brk = 13. * np.arctan(0.76*f/1000.) + 3.5 * np.arctan(f / (1000 * 7.5)) ** 2. # From WS16_17, Psychoacoustic slides (no 5)

    return Brk
```

### Method to compute Hz from Bark scale. Based on :

<https://github.com/stephencwelch/Perceptual-Coding-In-Python> (<https://github.com/stephencwelch/Perceptual-Coding-In-Python>)

```
In [ ]: def bark2hz(self, Brk):
    """ Method to compute Hz from Bark scale. Based on :
    https://github.com/stephencwelch/Perceptual-Coding-In-Python
    Args :
        Brk   : (ndarray)    Array containing Bark scaled values.
    Returns :
        Fhz  : (ndarray)    Array containing frequencies in Hz.
    """
    Fhz = 600. * np.sinh(Brk/6.)

    return Fhz
```

### Method to acquire critical band filters for creation of the PEAQ FFT model.

Based on : <https://github.com/stephencwelch/Perceptual-Coding-In-Python> (<https://github.com/stephencwelch/Perceptual-Coding-In-Python>)



In [ ]:

```

def CB_filters(self):
    """ Method to acquire critical band filters for creation of the PEAQ FFT model.
    Based on : https://github.com/stephencwelch/Perceptual-Coding-In-Python
    Returns :
        fc, fl, fu : (ndarray)    Arrays containing the values in Hz for the
                                    bandwidth and centre frequencies used in creation
                                    of the transformation matrix.
    """
# Lower frequencies Look-up table
fl = np.array([ 80.000,   103.445,   127.023,   150.762,   174.694, \
               198.849,   223.257,   247.950,   272.959,   298.317, \
               324.055,   350.207,   376.805,   403.884,   431.478, \
               459.622,   488.353,   517.707,   547.721,   578.434, \
               609.885,   642.114,   675.161,   709.071,   743.884, \
               779.647,   816.404,   854.203,   893.091,   933.119, \
               974.336,   1016.797,   1060.555,   1105.666,   1152.187, \
               1200.178,   1249.700,   1300.816,   1353.592,   1408.094, \
               1464.392,   1522.559,   1582.668,   1644.795,   1709.021, \
               1775.427,   1844.098,   1915.121,   1988.587,   2064.590, \
               2143.227,   2224.597,   2308.806,   2395.959,   2486.169, \
               2579.551,   2676.223,   2776.309,   2879.937,   2987.238, \
               3098.350,   3213.415,   3332.579,   3455.993,   3583.817, \
               3716.212,   3853.817,   3995.399,   4142.547,   4294.979, \
               4452.890,   4616.482,   4785.962,   4961.548,   5143.463, \
               5331.939,   5527.217,   5729.545,   5939.183,   6156.396, \
               6381.463,   6614.671,   6856.316,   7106.708,   7366.166, \
               7635.020,   7913.614,   8202.302,   8501.454,   8811.450, \
               9132.688,   9465.574,   9810.536,   10168.013,   10538.460, \
               10922.351,   11320.175,   11732.438,   12159.670,   12602.412, \
               13061.229,   13536.710,   14029.458,   14540.103,   15069.295, \
               15617.710,   16186.049,   16775.035,   17385.420 ])
# Centre frequencies Look-up table
fc = np.array([ 91.708,   115.216,   138.870,   162.702,   186.742, \
               211.019,   235.566,   260.413,   285.593,   311.136, \
               337.077,   363.448,   390.282,   417.614,   445.479, \
               473.912,   502.950,   532.629,   562.988,   594.065, \
               625.899,   658.533,   692.006,   726.362,   761.644, \
               797.898,   835.170,   873.508,   912.959,   953.576, \
               995.408,   1038.511,   1082.938,   1128.746,   1175.995, \
               1224.744,   1275.055,   1326.992,   1380.623,   1436.014, \
               1493.237,   1552.366,   1613.474,   1676.641,   1741.946, \
               1809.474,   1879.310,   1951.543,   2026.266,   2103.573, \
               2180.880,   2260.297,   2342.714,   2427.131,   2513.548, \
               2602.965,   2694.382,   2787.799,   2883.216,   2981.633, \
               3082.050,   3184.467,   3288.884,   3395.291,   3503.700, \
               3613.110,   3724.520,   3837.930,   3952.340,   4067.750, \
               4184.160,   4301.570,   4420.980,   4541.400,   4662.820, \
               4785.240,   4908.660,   5033.080,   5158.500,   5284.920, \
               5412.340,   5540.760,   5670.180,   5800.600,   5932.020, \
               6064.440,   6207.860,   6352.280,   6507.700,   6664.120, \
               6821.540,   7000.000,   7180.460,   7362.920,   7546.400, \
               7731.880,   7918.360,   8106.840,   8306.320,   8507.800, \
               8710.280,   8914.760,   9120.240,   9327.720,   9536.200, \
               9745.680,   9956.160,   10167.640,   10379.120,   10591.600, \
               10804.080,   11017.560,   11231.040,   11445.520,   11660.000, \
               11875.480,   12091.960,   12309.440,   12527.920,   12747.400, \
               12967.880,   13189.360,   13412.840,   13637.320,   13862.800, \
               14089.280,   14317.760,   14547.240,   14777.720,   15009.200, \
               15242.680,   15477.160,   15712.640,   15949.120,   16186.600, \
               16425.080,   16663.560,   16903.040,   17143.520,   17385.000, \
               17627.480,   17871.960,   18117.440,   18363.920,   18611.400, \
               18859.880,   19109.360,   19360.840,   19613.320,   19866.800, \
               20121.280,   20377.760,   20635.240,   20893.720,   21153.200, \
               21413.680,   21675.160,   21937.640,   22201.120,   22465.600, \
               22731.080,   23097.560,   23465.040,   23833.520,   24202.000, \
               24571.480,   24941.960,   25313.440,   25685.920,   26059.400, \
               26433.880,   26809.360,   27186.840,   27564.320,   27942.800, \
               28322.280,   28702.760,   29083.240,   29464.720,   29846.200, \
               30228.680,   30612.160,   31095.640,   31579.120,   32063.600, \
               32548.080,   33033.560,   33519.040,   34005.520,   34492.000, \
               34980.480,   35469.960,   35959.440,   36450.920,   36942.400, \
               37434.880,   37928.360,   38422.840,   38917.320,   39412.800, \
               39908.280,   40405.760,   40903.240,   41401.720,   41900.200, \
               42409.680,   42910.160,   43412.640,   43915.120,   44418.600, \
               44923.080,   45428.560,   45934.040,   46440.520,   46947.000, \
               47454.480,   47962.960,   48471.440,   48980.920,   49490.400, \
               49999.880,   50510.360,   51021.840,   51533.320,   52045.800, \
               52558.280,   53071.760,   53585.240,   54108.720,   54632.200, \
               55156.680,   55681.160,   56206.640,   56732.120,   57258.600, \
               57785.080,   58312.560,   58840.040,   59368.520,   59897.000, \
               60425.480,   60954.060,   61483.640,   61913.220,   62443.800, \
               62974.380,   63505.960,   64037.540,   64569.120,   65101.700, \
               65634.280,   66167.860,   66701.440,   67235.020,   67768.600, \
               68302.180,   68836.360,   69370.540,   69904.720,   70438.900, \
               70973.080,   71507.260,   72041.440,   72575.620,   73110.800, \
               73646.080,   74181.360,   74716.640,   75251.920,   75787.200, \
               76322.580,   76858.260,   77394.940,   77931.620,   78468.300, \
               78905.080,   79442.860,   79979.640,   80516.420,   81053.200, \
               81590.080,   82127.960,   82665.840,   83203.720,   83741.600, \
               84279.580,   84817.560,   85355.540,   85893.520,   86431.500, \
               86969.580,   87507.660,   88045.740,   88583.820,   89121.900, \
               89659.980,   90208.060,   90756.140,   91304.220,   91852.300, \
               92400.380,   92948.460,   93496.540,   93944.620,   94392.700, \
               94840.780,   95288.860,   95736.940,   96184.020,   96631.100, \
               97078.180,   97525.260,   97972.340,   98419.420,   98866.500, \
               99313.580,   99760.660,   100207.740,   100654.820,   101101.900, \
               101549.080,   101996.160,   102443.240,   102890.320,   103337.400, \
               103784.480,   104231.560,   104678.640,   105125.720,   105572.800, \
               105919.880,   106366.960,   106814.040,   107261.120,   107708.200, \
               108155.280,   108602.360,   109049.440,   109496.520,   109943.600, \
               110390.680,   110837.760,   111284.840,   111731.920,   112179.000, \
               112626.080,   113073.160,   113520.240,   113967.320,   114414.400, \
               114861.480,   115308.560,   115755.640,   116202.720,   116650.800, \
               117108.880,   117566.960,   118024.040,   118481.120,   118938.200, \
               119395.280,   119852.360,   120310.440,   120768.520,   121226.600, \
               121684.680,   122142.760,   122600.840,   123058.920,   123517.000, \
               123975.080,   124433.160,   124891.240,   125349.320,   125807.400, \
               126265.480,   126723.560,   127181.640,   127639.720,   128097.800, \
               128555.880,   129013.960,   129472.040,   129930.120,   130388.200, \
               130846.280,   131304.360,   131762.440,   132220.520,   132678.600, \
               133136.680,   133594.760,   134052.840,   134510.920,   134969.000, \
               135427.080,   135885.160,   136343.240,   136801.320,   137269.400, \
               137737.480,   138205.560,   138673.640,   139141.720,   139609.800, \
               140077.880,   140545.960,   141014.040,   141482.120,   141950.200, \
               142418.280,   142886.360,   143354.440,   143822.520,   144290.600, \
               144758.680,   145226.760,   145694.840,   146162.920,   146631.000, \
               147099.080,   147567.160,   148035.240,   148503.320,   148971.400, \
               149449.480,   149917.560,   150385.640,   150853.720,   151321.800, \
               151789.880,   152257.960,   152725.040,   153193.120,   153661.200, \
               154139.280,   154617.360,   155095.440,   155573.520,   156051.600, \
               156529.680,   157007.760,   157485.840,   157963.920,   158442.000, \
               158920.080,   159398.160,   159876.240,   160354.320,   160832.400, \
               161310.480,   161788.560,   162266.640,   162744.720,   163222.800, \
               163700.880,   164178.960,   164657.040,   165135.120,   165613.200, \
               166091.280,   166569.360,   167047.440,   167525.520,   168003.600, \
               168481.680,   168959.760,   169437.840,   169915.920,   170394.000, \
               170872.080,   171350.160,   171828.240,   172306.320,   172784.400, \
               173262.480,   173740.560,   174218.640,   174696.720,   175174.800, \
               175652.880,   176130.960,   176609.040,   177087.120,   177565.200, \
               178043.280,   178521.360,   179000.440,   179478.520,   180056.600, \
               180634.680,   181212.760,   181790.840,   182368.920,   182947.000, \
               183525.080,   184103.160,   184681.240,   185259.320,   185837.400, \
               186415.480,   186993.560,   187571.640,   188150.720,   188729.800, \
               189308.880,   189887.960,   190467.040,   191046.120,   191625.200, \
               192204.280,   192783.360,   193362.440,   193941.520,   194520.600, \
               195100.680,   195679.760,   196258.840,   196838.920,   197418.000, \
               198097.080,   198676.160,   199255.240,   199834.320,   200413.400, \
               201092.480,   201671.560,   202250.640,   202830.720,   203410.800, \
               204089.880,   204669.960,   205249.040,   205829.120,   206409.200, \
               207089.280,   207669.360,   208249.440,   208829.520,   209409.600, \
               210089.680,   210669.760,   211250.840,   211830.920,   212411.000, \
               213091.080,   213671.160,   214251.240,   214831.320,   215411.400, \
               216091.480,   216671.560,   217251.640,   217831.720,   218411.800, \
               219091.880,   219671.960,   220252.040,   220832.120,   221412.200, \
               222092.280,   222672.360,   223252.440,   223832.520,   224412.600, \
               225092.680,   225672.760,   226252.840,   226832.920,   227413.000, \
               228093.080,   228673.160,   229253.240,   229833.320,   230413.400, \
               231093.480,   231673.560,   232253.640,   232833.720,   233413.800, \
               234093.880,   234673.960,   235254.040,   235834.120,   236414.200, \
               237094.280,   237674.360,   238254.440,   238834.520,   239414.600, \
               240094.680,   240674.760,   241254.840,   241834.920,   242414.000, \
               243094.080,   243674.160,   244254.240,   244834.320,   245414.400, \
               246094.480,   246674.560,   247254.640,   247834.720,   248414.800, \
               249094.880,   249674.960,   250255.040,   250835.120,   251415.200, \
               252095.280,   252675.360,   253255.440,   253835.520,   254415.600, \
               255095.680,   255675.760,   256255.840,   256835.920,   257415.000, \
               258095.080,   258675.160,   259255.240,   259835.320,   260415.400, \
               261095.480,   261675.560,   262255.640,   262835.720,   263415.800, \
               264095.880,   264675.960,   265256.040,   265836.120,   266416.200, \
               267096.280,   267676.360,   268256.440,   268836.520,   269416.600, \
               270096.680,   270676.760,   271256.840,   271836.920,   272416.000, \
               273096.080,   273676.160,   274256.240,   274836.320,   275416.400, \
               276096.480,   276676.560,   277256.640,   277836.720,   278416.800, \
               279096.880,   279676.960,   280256.040,   280836.120,   281416.200, \
               282096.280,   282676.360,   283256.440,   283836.520,   284416.600, \
               285096.680,   285676.760,   286256.840,   286836.920,   287416.000, \
               288096.080,   288676.160,   289256.240,   289836.320,   290416.400, \
               291096.480,   291676.560,   292256.640,   292836.720,   293416.800, \
               294096.880,   294676.960,   295256.040,   295836.120,   296416.200, \
               297096.280,   297676.360,   298256.440,   298836.520,   299416.600, \
               299996.680,   300676.760,   301256.840,   301836.920,   302416.000, \
               303096.080,   303676.160,   304256.240,   304836.320,   305416.400, \
               306096.480,   306676.560,   307256.640,   307836.720,   308416.800, \
               309096.880,   309676.960,   310256.040,   310836.120,   311416.200, \
               312096.280,   312676.360,   313256.440,   313836.520,   314416.600, \
               315096.680,   315676.760,   316256.840,   316836.920,   317416.000, \
               318096.080,   318676.160,   319256.240,   319836.320,   320416.400, \
               321096.480,   321676.560,   322256.640,   322836.720,   323416.800, \
               324096.880,   324676.960,   325256.040,   325836.120,   326416.200, \
               327096.280,   327676.360,   328256.440,   328836.520,   329416.600, \
               330096.680,   330676.760,   331256.840,   331836.920,   332416.000, \
               333096.080,   333676.160,   334256.240,   334836.320,   335416.400, \
               336096.480,   336676.560,   337256.640,   337836.720,   338416.800, \
               339096.880,   339676.960,   340256.040,   340836.120,   341416.200, \
               342096.280,   342676.360,   343256.440,   343836.520,   344416.600, \
               345096.680,   345676.760,   346256.840,   346836.920,   347416.000, \
               348096.080,   348676.160,   349256.240,   349836.320,   350416.400, \
               351096.480,   351676.560,   352256.640,   
```

```

2183.564, 2266.340, 2352.008, 2440.675, 2532.456, \
2627.468, 2725.832, 2827.672, 2933.120, 3042.309, \
3155.379, 3272.475, 3393.745, 3519.344, 3649.432, \
3784.176, 3923.748, 4068.324, 4218.090, 4373.237, \
4533.963, 4700.473, 4872.978, 5051.700, 5236.866, \
5428.712, 5627.484, 5833.434, 6046.825, 6267.931, \
6497.031, 6734.420, 6980.399, 7235.284, 7499.397, \
7773.077, 8056.673, 8350.547, 8655.072, 8970.639, \
9297.648, 9636.520, 9987.683, 10351.586, 10728.695, \
11119.490, 11524.470, 11944.149, 12379.066, 12829.775, \
13294.850, 13780.887, 14282.503, 14802.338, 15341.057, \
15899.345, 16477.914, 17077.504, 17690.045 ])

# Upper frequencies Look-up table
fu = np.array([ 103.445, 127.023, 150.762, 174.694, 198.849, \
223.257, 247.950, 272.959, 298.317, 324.055, \
350.207, 376.805, 403.884, 431.478, 459.622, \
488.353, 517.707, 547.721, 578.434, 609.885, \
642.114, 675.161, 709.071, 743.884, 779.647, \
816.404, 854.203, 893.091, 933.113, 974.336, \
1016.797, 1060.555, 1105.666, 1152.187, 1200.178, \
1249.700, 1300.816, 1353.592, 1408.094, 1464.392, \
1522.559, 1582.668, 1644.795, 1709.021, 1775.427, \
1844.098, 1915.121, 1988.587, 2064.590, 2143.227, \
2224.597, 2308.806, 2395.959, 2486.169, 2579.551, \
2676.223, 2776.309, 2879.937, 2987.238, 3098.350, \
3213.415, 3332.579, 3455.993, 3583.817, 3716.212, \
3853.348, 3995.399, 4142.547, 4294.979, 4452.890, \
4643.482, 4785.962, 4961.548, 5143.463, 5331.939, \
5527.217, 5729.545, 5939.183, 6156.396, 6381.463, \
6614.671, 6856.316, 7106.708, 7366.166, 7635.020, \
7913.614, 8202.302, 8501.454, 8811.450, 9132.688, \
9465.574, 9810.536, 10168.013, 10538.460, 10922.351, \
11320.175, 11732.438, 12159.670, 12602.412, 13061.229, \
13536.710, 14029.458, 14540.103, 15069.295, 15617.710, \
16186.049, 16775.035, 17385.420, 18000.000 ])

return fc, fl, fu

```

## Method to transform FT domain to Bark.

```
In [ ]: def forward(self, spc):
    """ Method to transform FT domain to Bark.
    Args      :
        spc      : (ndarray)    2D Array containing the magnitude spectra.
    Returns   :
        Brk_spc : (ndarray)    2D Array containing the Bark scaled magnitude spectra.
    """
    W_short = self.W[:, 0:self.nfreqs]
    Brk_spc = np.dot(W_short,spc)
    return Brk_spc
```

## Method to reconstruct FT domain from Bark.

```
In [ ]: def backward(self, Brk_spc):
    """ Method to reconstruct FT domain from Bark.
    Args      :
        Brk_spc : (ndarray)    2D Array containing the Bark scaled magnitude spectra.
    Returns   :
        Xhat     : (ndarray)    2D Array containing the reconstructed magnitude spectra.
    """
    Xhat = np.dot(self.W_inv,Brk_spc)
    return Xhat
```

## Method to "correct" the middle outer ear transfer function.

As appears in :

- A. Härmä, and K. Palomäki, ''HUTear - a free Matlab toolbox for modeling of human hearing'', in Proceedings of the Matlab DSP Conference, pp 96-99, Espoo, Finland 1999.

In [ ]:

```

def OutMidCorrection(self, correctionType, firOrd, fs):
    """
        Method to "correct" the middle outer ear transfer function.
        As appears in :
            - A. Härmä, and K. Palomäki, ''HUTear - a free Matlab toolbox for modeling of human hearing'',
            in Proceedings of the Matlab DSP Conference, pp 96-99, Espoo, Finland 1999.
    """
    # Lookup tables for correction
    # Frequency table 1
    f1 = np.array([20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100, \
125, 150, 177, 200, 250, 300, 350, 400, 450, 500, 550, \
600, 700, 800, 900, 1000, 1500, 2000, 2500, 2828, 3000, \
3500, 4000, 4500, 5000, 5500, 6000, 7000, 8000, 9000, 10000, \
12748, 15000])

    # ELC correction in dBs
    ELC = np.array([ 31.8, 26.0, 21.7, 18.8, 17.2, 15.4, 14.0, 12.6, 11.6, 10.6, \
9.2, 8.2, 7.7, 6.7, 5.3, 4.6, 3.9, 2.9, 2.7, 2.3, \
2.2, 2.3, 2.5, 2.7, 2.9, 3.4, 3.9, 3.9, 3.9, 2.7, \
0.9, -1.3, -2.5, -3.2, -4.4, -4.1, -2.5, -0.5, 2.0, 5.0, \
10.2, 15.0, 17.0, 15.5, 11.0, 22.0])

    # MAF correction in dBs
    MAF = np.array([ 73.4, 65.2, 57.9, 52.7, 48.0, 45.0, 41.9, 39.3, 36.8, 33.0, \
29.7, 27.1, 25.0, 22.0, 18.2, 16.0, 14.0, 11.4, 9.2, 8.0, \
6.9, 6.2, 5.7, 5.1, 5.0, 5.0, 4.4, 4.3, 3.9, 2.7, \
0.9, -1.3, -2.5, -3.2, -4.4, -4.1, -2.5, -0.5, 2.0, 5.0, \
10.2, 15.0, 17.0, 15.5, 11.0, 22.0])

    # Frequency table 2 for MAP
    f2 = np.array([ 125, 250, 500, 1000, 1500, 2000, 3000, \
4000, 6000, 8000, 10000, 12000, 14000, 16000])

    # MAP correction in dBs
    MAP = np.array([ 30.0, 19.0, 12.0, 9.0, 11.0, 16.0, 16.0, \
14.0, 14.0, 9.9, 24.7, 32.7, 44.1, 63.7])

    if correctionType == 'ELC':
        freqTable = f1
        CorrectionTable = ELC
    elif correctionType == 'MAF':
        freqTable = f1
        CorrectionTable = MAF

```

```
elif correctionType == 'MAP':
    freqTable = f2
    CorrectionTable = MAP
else :
    print('Unrecongised operation: ELC will be used instead...')
    freqTable = f1
    CorrectionTable = ELC

# Filter desing through spline interpolation
freqN = np.arange(0, firOrd) * fs/2. / (firOrd-1)
spline = uspline(freqTable, CorrectionTable)
crc = spline(freqN)
crclin = 10. ** (-crc/ 10.)
return crclin, freqN, crc
```

**Method to approximate middle-outer ear transfer function for linearly scaled frequency representations, using an FIR approximation of order 600 taps.**

As appears in :

- A. Härmä, and K. Palomäki, ''HUTear - a free Matlab toolbox for modeling of human hearing'', in Proceedings of the Matlab DSP Conference, pp 96-99, Espoo, Finland 1999.

```
In [ ]: def MOEar(self, correctionType = 'ELC'):
    """ Method to approximate middle-outer ear transfer function for linearly scaled
    frequency representations, using an FIR approximation of order 600 taps.
    As appears in :
        - A. Härmä, and K. Palomäki, ''HUTear - a free Matlab toolbox for modeling of human hearing'',
          in Proceedings of the Matlab DSP Conference, pp 96-99, Espoo, Finland 1999.

    Arguments      :
        correctionType : (string)      String which specifies the type of correction :
                                         'ELC' - Equal Loudness Curves at 60 dB (default)
                                         'MAP' - Minimum Audible Pressure at ear canal
                                         'MAF' - Minimum Audible Field

    Returns        :
        LTq       : (ndarray)      1D Array containing the transfer function, without the DC sub-band.

    """
    # Parameters
    firOrd = self.nfft                                     # FIR order
    Cr, fr, Crdb = self.OutMidCorrection(correctionType, firOrd, self.fs)      # Acquire frequency/magnitude tables
    Cr[self.nfft - 1] = 0.

    # FIR Design
    A = firwin2(firOrd, fr, Cr, nyq = self.fs/2)
    B = 1
    _, LTq = freqz(A, B, firOrd, self.fs)

    LTq = 20. * np.log10(np.abs(LTq))
    LTq -= max(LTq)
    return LTq[:self.nfft/2 + 1]
```

## **Method to compute the masking threshold by non-linear superposition.**

As used in :

- G. Schuller, B. Yu, D. Huang and B. Edler, "Perceptual Audio Coding Using Adaptive Pre and Post-filters and Lossless Compression", in IEEE Transactions on Speech and Audio Processing, vol. 10, n. 6, pp. 379-390, September, 2002.

As appears in :

- F. Baumgarte, C. Ferekidis, H Fuchs, "A Nonlinear Psychoacoustic Model Applied to ISO/MPEG Layer 3 Coder", in Proceedings of the 99th Audio Engineering Society Convention, October, 1995.

In [ ]:

```

def maskingThreshold(self, mX):
    """ Method to compute the masking threshold by non-Linear superposition.
    As used in :
    - G. Schuller, B. Yu, D. Huang and B. Edler, "Perceptual Audio Coding Using Adaptive
    Pre and Post-filters and Lossless Compression", in IEEE Transactions on Speech and Audio Processing,
    vol. 10, n. 6, pp. 379-390, September, 2002.
    As appears in :
    - F. Baumgarte, C. Ferekidis, H Fuchs, "A Nonlinear Psychoacoustic Model Applied to ISO/MPEG Layer 3
    Coder",
    in Proceedings of the 99th Audio Engineering Society Convention, October, 1995.

    Args          :
        mX       : (ndarray)    2D Array containing the magnitude spectra (1 time frame x frequency subbands)
    Returns        :
        mT       : (ndarray)    2D Array containing the masking threshold.
    Set alpha in __init__ for suitable exponents for non-Linear superposition!
    Authors       : Gerald Schuller('shl'), S.I. MimiLakis ('mis')
    """
    # Bark Scaling with the initialized, from the class, matrix W.
    if mX.shape[1] % 2 == 0:
        #print "mX.shape even"
        nyq = False
        mX = (np.dot( np.abs(mX)**2.0, self.W[:, :self.nfreqs].T))**(0.5)
    else :
        nyq = True
        mX = np.dot(np.abs(mX), self.W[:, :self.nfreqs + 1].T)

    # Parameters
    Numsubbands = mX.shape[1] #Number of bark bands
    #print "Numsubbands= ", Numsubbands, "self.nfilts=", self.nfilts
    timeFrames = mX.shape[0]
    maxbark=self.hz2bark(self.max_freq)
    fc = self._maxb*Numsubbands #Numsubbands/nfilts, after call: Numsubbands=self.nfilts
    fc= maxbark/Numsubbands

    #self._fa = 1./(10 ** ((14.5+12)/20.))           # simultaneous masking, 26.5dB
    #self._fb = 1./(10**((7.5/20.)))                 # Upper slope of spreading function, 7.5dB/B
ark
    #self._fbb = 1./(10**((26./20.)))                # Lower slope of spreading function, 26dB/ba
rk
    #self._fd = 1./self._alpha

```

```

# Initialization of the matrix containing the masking threshold
maskingThreshold = np.zeros((timeFrames, Numsubbands))

"""

spreadingfunctionBarkdB=np.zeros(2*Numsubbands)
#Lower slope in dB: 26 dB attenuation per Bark, over 24 Bark (full frequency range), with 26.5 dB simultaneous masking:
spreadingfunctionBarkdB[0:Numsubbands]=np.linspace(-24*26,0,Numsubbands)-26.5
#upper slope in dB: 7.5 dB attenuation per Bark, over 24 Bark (full frequency range):
spreadingfunctionBarkdB[Numsubbands,2*Numsubbands]=np.linspace(0,-24*7.5,Numsubbands)-26.5
spreadingfunctionBarkVoltage=10.0** (spreadingfunctionBarkdB/20.0)
#Spreading functions for all bark scale bands in a matrix:
self.spreadingfuncmatrix=np.zeros((Numsubbands,Numsubbands))
for k in range(Numsubbands):
    self.spreadingfuncmatrix[k,:]=spreadingfunctionBarkVoltage[(Numsubbands-k):(2*Numsubbands-k)] 
"""

#print "spreadingfuncmatrix= ", self.spreadingfuncmatrix

for frameindx in xrange(mX.shape[0]) :

    """

    mT = np.zeros((Numsubbands))
    for n in xrange(Numsubbands):
        for m in xrange(0, n):
            mT[n] += (mX[frameindx, m]*self._fa * (self._fb ** ((n - m) * fc))) ** self._alpha
            #print "factors Loops: ",self._fa * (self._fb ** ((n - m) * fc)) ** self._alpha

        for m in xrange(n, Numsubbands):
            mT[n] += (mX[frameindx, m]*self._fa * (self._fb ** ((m - n) * fc))) ** self._alpha
            #if n==0:
            #print "(m - n)=", (m - n), "fc=", fc, "self._fa", self._fa, "self._fb", self._fb,
            "self._alpha",
            self._alpha
            #print "factors Loops for band 0: ", (self._fa * (self._fb ** ((m - n) * fc))) ** self._alpha

        mT[n] = mT[n] ** (self._fd)
        #print "mX[frameindx, 0] ", mX[frameindx, 0]
        #print "mT for Loops: ", mT[0]
    """

```

```

#Application of spreading function, including exponent alpha, using matrix multiplication:
    """
    mT=np.dot(mX[frameindx, :]**self._alpha, self.spreadingfuncmatrix)
    #print "factors matrix, band0: ", self.spreadingfuncmatrix[:,0]
    #apply the inverse exponent to the result:
    mT=mT**(1.0/self._alpha)
    #print "mX[frameindx, 0] ", mX[frameindx, 0]
    #print "mT matrix mult: ", mT[0]
    """
    #Adding masking threshold in quiet:
    #print "mT.shape=", mT.shape, "self._LTeq.shape", mT +( 10.0**((self._LTeq-20)/20))
maskingThreshold[frameindx, :] = np.max((mT, 10.0**((self._LTeq-60)/20)),0)

# Inverse the bark scaling with the initialized, from the class, matrix W_inv.
if nyq == False:
    maskingThreshold = np.dot(maskingThreshold, self.W_inv[:-1, :self.nfreqs].T)
else :
    maskingThreshold = np.dot(maskingThreshold, self.W_inv[:, :self.nfreqs].T)

return maskingThreshold

```

## Setting up the parameters and testing the model

Take a wave file and read it using 'x, fs = IO.AudioIO.wavRead('track.wav', mono = True)' and test it out.



```

In [ ]: # Parameters
wsz = 1024
N = 2048
hop = 512
#hop = 1024
gain = 0.

# Reading
# File Reading
#x, fs = IO.AudioIO.wavRead('mixed.wav', mono = True)
#x, fs = IO.AudioIO.wavRead('sc03_16m.wav', mono = True)
fs, x= wav.read('sc03_16m.wav')
#x, fs = IO.AudioIO.wavRead('1khz16khz.wav', mono = True)
#x, fs = IO.AudioIO.wavRead('1khz44100hz.wav', mono = True)
#x, fs = IO.AudioIO.wavRead('silence44100Hz.wav', mono = True)
print "fs=", fs
x *= 1.0
print "max= ", np.max(x)
# Cosine testq
#x = np.cos(np.arange(88200) * (1000.0 * (3.1415926 * 2.0) / 44100)) * 0.1
#fs = 44100
# Generate noise. Scaling was found experimentally to perfectly match the masking threshold magnitude.
#should be according to equal energy, with quantization interval Delta, DFT and mask give "voltages",
#squared voltages are power.
#Delta is the quantization step size in a coder,
# Delta= max noise - min noise,, Delta=1 means uniform dist from -0.5 to +0.5,
#noise Energy E=Delta^2/12=mask^2, Delta/sqrt(12)=mask,
# hence Delta=mask*sqrt(12)
# We only have real valued noise, not imaginary: We need factor 2 in power to make up for
# missing imag. part
#We only use half of the DFT of the spectrum, hence we need another factor of 2 in power to
#compensate for it, hence a factor of sqrt(4)=2 in "voltage":
#noise = np.random.uniform(-30., 30., (len(x), 1))
noise = np.random.uniform(-.5, .5, (len(x), 1))*np.sqrt(12)*2

# STFT/iSTFT Test
w = np.bartlett(wsz)
# Normalise windowing function
w = w / sum(w)

#Sine window: choose this for sine windowing for energy conservation (Parseval Theorem):
w=np.sin(np.pi/wsz*np.arange(0.5,wsz))

```

```

print("w.shape = ", w.shape)

# Initialize psychoacoustic mode
pm = PsychoacousticModel(N = N, fs = fs, nfilters = 64)

# Visual stuff
option = 'pygame'
#option = 'matplotlib'

# Pygame visual handles
pygame.init()
pygame.display.set_caption("Masking Threshold Visualization")
color = pygame.Color(255, 0, 0, 0)
color2 = pygame.Color(0, 255, 0, 0)
color3 = pygame.Color(0, 110, 0, 0)
background_color = pygame.Color(0, 0, 0, 0)
screen = pygame.display.set_mode((wsz + 40, 480))
#screenbg = pygame.display.set_mode((wsz + 40, 480))
screenbg = pygame.Surface((wsz + 40, 480))
#screenbg.set_alpha(100)
screenbg.fill(background_color)

dBscales = np.linspace(-120, 0, 25)
dBpos = (dBscales/-120 * 480)

# Draw Labels and help text on background surface "screenbg":
# Done here to get it out of the runtime Loop.
font = pygame.font.Font(None, 24)
xlabel = font.render("Frequency sub-bands", 1, (100, 100, 250))
ylabel = font.render("Magnitude in dB FS", 1, (100, 100, 250))
legendA = font.render("Magnitude Spectrum", 1, (250, 0, 0))
legendB = font.render("Estimated masking threshold", 1, (0, 250, 0))
ylabel = pygame.transform.rotate(ylabel, 90)
maxfrequ= font.render(str(fs/2)+"Hz", 1, (100, 100, 250))
screen.blit(maxfrequ, (895, 460))
screenbg.blit(xlabel, (895, 460))

screenbg.blit(ylabel, (0, 5))
screenbg.blit(legendA, (800, 0))
screenbg.blit(legendB, (800, 15))
#offset = font.render("Masking Threshold Offset (NMR) in dB: " + str(gain), 1, (0, 250, 0))
#bpc = font.render("Est. average bits per sample: " + str(bc), 1, (190, 160, 110))
helptext = font.render("(Adjust the threshold by pressing 'Up' & 'Down' Arrow keys)", 1, (0, 250, 0))

```

```

#screenbg.blit(offset, (300, 0))
screenbg.blit(helpText, (300, 15))
#screenbg.blit(bpc, (300, 30))

for n2 in xrange(len(dBpos)):
    dB = font.render(str(np.int(dBScales[n2])), 1, (0,120,120))
    screenbg.blit(dB, (20, int(dBpos[n2])))

pygame.display.flip()

if option == 'matplotlib':
    # Using matplotlib
    plt.ion()
    ax = plt.axes(xlim=(0, wsz), ylim=(-120, 0))
    line, = plt.plot(np.bartlett(wsz), label = 'Signal')
    line2, = plt.plot(np.bartlett(wsz), label = 'Masking Threshold')
    plt.xlabel('Frequency sub-bands')
    plt.ylabel('dB FS')
    plt.legend(handles=[line, line2])
    plt.show()

# Main Loop
run = True
while run == True:
    # Initialize sound pointers and buffers
    pin = 0
    pend = x.size - wsz - hop
    #b_mX = np.zeros((1, wsz + 1), dtype = np.float32)
    #b_nX = np.zeros((1, wsz + 1), dtype = np.float32)
    b_mX = np.zeros((1, wsz ), dtype = np.float32)
    b_nX = np.zeros((1, wsz), dtype = np.float32)
    #mt = np.zeros((1, wsz + 1), dtype = np.float32)
    mt = np.zeros((1, wsz), dtype = np.float32)
    output_buffer = np.zeros((1, wsz), dtype = np.float32)
    ola_buffer = np.zeros((1, wsz+hop), dtype = np.float32)
    prv_seg = np.zeros((1, 1024), dtype = np.float32)
    noiseMagnitude = np.ones((1, 1025), dtype = np.float32)

    # For less frequent plotting to avoide buffer underruns
    indx = 0

    # Reshaping the signal
    x = x.reshape(len(x), 1)

```

```

# Streaming
p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paFloat32, channels=1, rate=fs, output=True, frames_per_buffer = wsz)

while pin <= pend:

    # Key events for the arrows. Up Arrow : + 1 dB Down Arrow : -1 dB
    cevent = pygame.event.get()
    if cevent != [] :
        if cevent[0].type == pygame.KEYDOWN:
            if cevent[0].key == pygame.K_q:
                run = False
                pygame.display.quit()
                pygame.quit()
                break
            elif cevent[0].key == 273 :
                gain += 1.
            elif cevent[0].key == 274 :
                gain -= 1.

    # Visual stuff
    if indx % 10 == 0:
        # Maximum quantization rate computation
        bc = (np.log2( (b_mX[0, :] + 1e-16)/(mt[0, :] + 1e-16)))
        bc = (np.mean(bc[bc >= 0 ]))

        if option == 'matplotlib' :
            # Matplotlib
            line.set_ydata(20. * np.log10(b_mX[0, :-1] + 1e-16))
            # Check for scaling the noise!
            line2.set_ydata(20. * np.log10(mt[0, :-1] + 1e-16))
            plt.draw()
            plt.pause(0.00001)

    else :
        # Pygame
        #screen.fill(background_color)
        screen.blit(screenbg,(0,0))
        prv_pos = (60, 480)
        prv_pos2 = (60, 480)
        prv_pos3 = (60, 480)
        for n in xrange(0, wsz):

```

```

        val = 20. * np.log10(b_mX[0, n] + 1e-16)
        val2 = 20. * np.log10(mt[0, n] + 1e-16)
        val3 = 20. * np.log10(b_nX[0, n] * mt[0, n] + 1e-16)
        val3 /= -120
        val /= -120
        val2/= -120
        val *= 480
        val2 *= 480
        val3 *= 480
        position = (n + 60, int(val))
        position2 = (n + 60, int(val2))
        position3 = (n + 60, int(val3))
        pygame.draw.line(screen, color, prv_pos, position)
        pygame.draw.line(screen, color2, prv_pos2, position2)
        pygame.draw.line(screen, color3, prv_pos3, position3)
        prv_pos = position
        prv_pos2 = position2
        prv_pos3 = position3

    # Print the surface
    """
        screen.blit(xLabel, (895, 460))
        screen.blit(yLabel, (0, 5))
        screen.blit(LegendA, (800, 0))
        screen.blit(LegendB, (800, 15))
    """
    offset = font.render("Masking Threshold Offset (NMR) in dB: " + str(gain), 1, (0, 250, 0))
    bpc = font.render("Est. average bits per subband: " + str(bc), 1, (190, 160, 110))
    screen.blit(bpc, (300, 30))
    screen.blit(offset, (300, 0))
    """
    helptext = font.render("(Adjust the threshold by pressing 'Up' & 'Down' Arrow keys)", 1, (0,
250, 0))

    screen.blit(helptext, (300, 15))
    for n2 in xrange(len(dbpos)):
        dB = font.render(str(np.int(dbScales[n2])), 1, (0,120,120))
        screen.blit(dB, (20, int(dbpos[n2])))
    """
# Display

pygame.display.flip()

```

```

# Acquire Segment
xSeg = x[pin:pin+wsz, 0]
nSeg = noise[pin:pin+wsz, 0]

# Perform DFT on segment
#mX, pX = TF.TimeFrequencyDecomposition.DFT(xSeg, w, N)
X=fft(xSeg*w,n=N, norm='ortho')[:N/2]
#print "size xSeg: ", xSeg.shape
mX=np.abs(X)
#nX, npX = TF.TimeFrequencyDecomposition.DFT(nSeg, w, N)

# Set it to buffer
b_mX[0, :] = mX
#b_nX[0, :] = nX
b_nX[0, :] = np.abs(nSeg)
# Masking threshold
mt = pm.maskingThreshold(b_mX) * (10**((gain/20.)))
#remove last entry (Nyquist freq)
#mt=mt[:, :]
#print "mt.shape= ", mt.shape
# Resynthesize
#nSeg = TF.TimeFrequencyDecomposition.iDFT(nX * mt[0, :], npX, wsz)
#Add noise according to masking threshold:
mixf=X+mt[0,:]*nSeg
#xSeg = TF.TimeFrequencyDecomposition.iDFT(mX, pX, wsz)
#choose this for Bartlett window:
#mix=np.real(ifft(mixf, n=N, norm='ortho')[:N/2])*hop
#Choose this for sine windowing:
mix=np.real(ifft(mixf, n=N, norm='ortho')[:N/2])*w
#mix = (xSeg + nSeg) * hop

ola_buffer[0, 0:wsz] = prv_seg
ola_buffer[0, hop:hop+wsz] += mix

# Place it to output buffer
output_buffer = ola_buffer[:, :wsz]

# Playback
writedata = output_buffer[0, :].astype(np.float32).tostring()
stream.write(writedata, num_frames = wsz/2, exception_on_underflow = False)

# Store previous frame samples
prv_seg = ola_buffer[:, hop:wsz+hop]

```

```
# Clear the overlap
ola_buffer = np.zeros((1, wsz+hop), dtype = np.float32)

# Update pointer and index
pin += hop
indx += 1

stream.stop_stream()
stream.close()
p.terminate()
pygame.display.quit()
pygame.quit()
```