Psycho-acoustic-modelDFT_gs

This module offers you to process an audio signal(wave audio) with psycho-aoustic 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. Mimilakis, 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, nfilts=24, type = 'rasta', width
= 1.0, minfreq=0, maxfreq=22050):

        self.nfft = N                                          # Number of sub-ba
nds in the prior analysis (e.g. from DFT)
        self.fs = fs                                           # Sampling Frequen
cy
        self.nfilts = nfilts                                   # Number of filter
s considered for the scaling,
                                                               # default: 24, fro
m call: 64
        self.width = width                                     # Width of the fil
ters for RASTA method
        self.min_freq = minfreq                                # Minimum consider
ed frequency
        self.max_freq = maxfreq                                # Maximum consider
ed frequency
        self.max_freq = fs/2                                   # Sanity check for
 Nyquist frequency
        self.nfreqs = N/2                                      # Number of freque
ncy points in the prior analysis (e.g. from DFT)
        self._LTeq = np.zeros(nfilts, dtype = np.float32)   # Initialization f
or 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 "vol
tage":
        self._alpha = 0.8                                        # Exponent alph
a
        self._maxb = 1./self.nfilts                               # Bark-band no
rmalization
        #self._fa = 1./(10 ** (14.5/20.) * 10 ** (12./20.))      # Simultaneou
s masking for tones at Bark band 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.))                          # Lowe
r slope of spreading function

        self._fd = 1./self._alpha                                 # One over alp
ha 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 frequ
ency 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 frequ
ency range):
        spreadingfunctionBarkdB[self.nfilts:2*self.nfilts]=np.linspace(0,-maxb
ark*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 transofrmation.
        Args :
            type : (str)        String denoting the type of transformation. Ca
n 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 evalua
tion.
        """

        # 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 evalua
tion.
        """
        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 2
4
        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 cal
l)
        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 subb
and:
            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 slobe, lower Bark domain boundary is minimum:
              self._LTeq[i] = min((3.64*(f/1000.)**-0.8-6.5*np.exp(-0.6*(f/100
0.-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 wind
ow
```

```
            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,wid
th), 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 doma
in.
        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.nf
reqs + 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 Ellis method (for speech quallity)

         # 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 F
FT model.
        Based on : https://github.com/stephencwelch/Perceptual-Coding-In-Pytho
n
        Returns         :
            fc, fl, fu  : (ndarray)    Arrays containing the values in Hz for
 the
                                       bandwidth and centre frequencies used i
n 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, \
              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 magnit
ude 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 magnit
ude spectra.
        Returns      :
            Xhat     : (ndarray)    2D Array containing the reconstructed magn
itude 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, Finl
and 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 linea
rly 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, Finl
and 1999.
        Arguments           :
            correctionType : (string)     String which specifies the type of c
orrection :
                                          '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 fun
ction, 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 Ada
ptive
    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 Appli
ed 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 superpositio
n.
        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 o
n 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 Mo
del Applied to ISO/MPEG Layer 3 Coder",
        in Proceedings of the 99th Audio Engineering Society Convention, Octob
er, 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 superposit
ion!
        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: Numsubban
ds=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/Bark
        #self._fbb = 1./(10**(26./20.))                         # Lower slope
 of spreading function, 26dB/bark
        #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 fre
```

```
quency range), with 26.5 dB simultaneous masking:
        spreadingfunctionBarkdB[0:Numsubbands]=np.linspace(-24*26,0,Numsubband
s)-26.5
        #upper slope in dB: 7.5 dB attenuation per Bark, over 24 Bark (full fr
equency 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[(Numsubb
ands-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._fbb ** ((m -
 n) * fc))) ** self._alpha
                    #if n==0:
                       #print "(m - n)=", (m - n), "fc=", fc, "self._fa", sel
f._fa, "self._fbb", self._fbb, "self._alpha",

                                  self._alpha
                       #print "factors loops for band 0: ", (self._fa * (self.
_fbb ** ((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 ma
trix 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**((se
lf._LTeq-20)/20))
        maskingThreshold[frameindx, :] =  np.max((mT, 10.0**((self._LTeq-60)/2
```

```
0)),0)


        # Inverse the bark scaling with the initialized, from the class, matri
x 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.nf
reqs].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 mask
ing threshold magnitude.
#should be according to equal energy, with quantization interval Delta, DFT an
d 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, nfilts = 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, 1
10))
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()
```