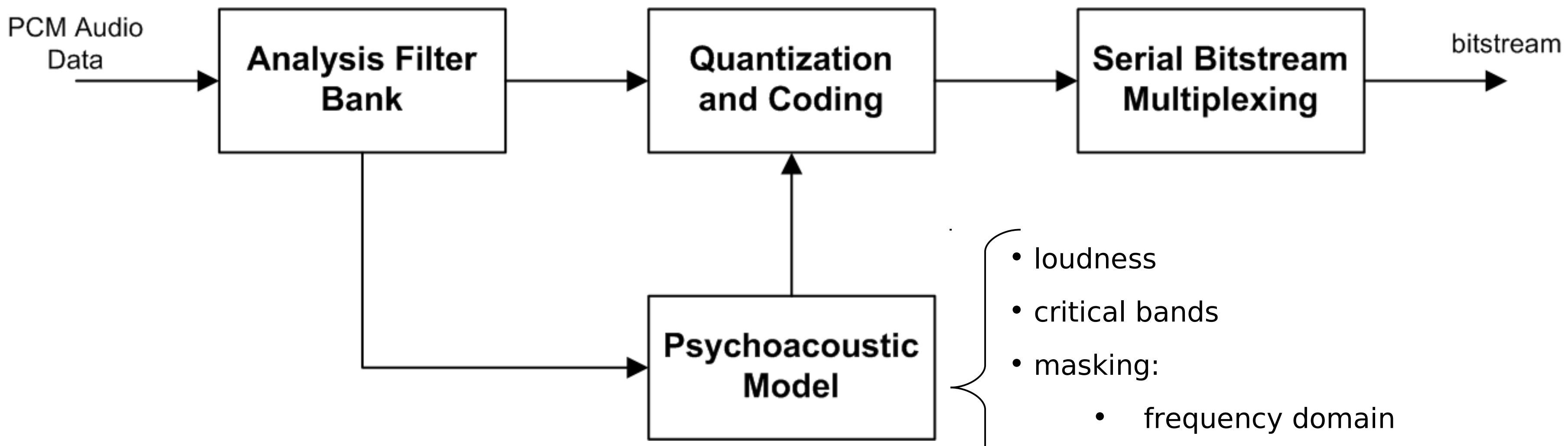
Psychoacoustics Models

Block Diagram of a Perceptual Audio Encoder

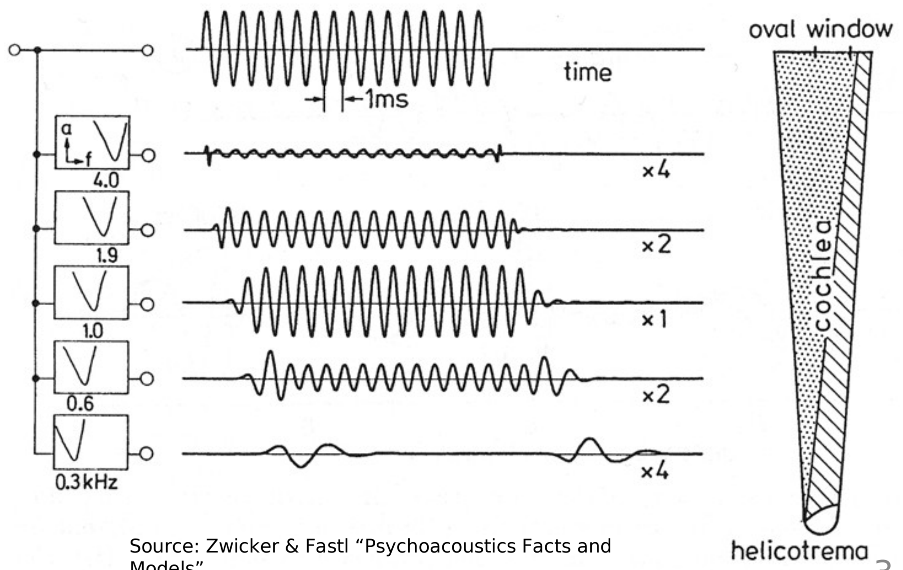


- loudness
- critical bands
- masking:
  - frequency domain
  - time domain
- binaural cues (overview)

Source: Brandenburg, “Vorlesung: Dig. Audiosignalverarbeitung”

Information Processing in the Auditory System

- basilar membrane as a filter bank
- bank of highly overlapping bandpass filters
- the magnitude responses are asymmetric and nonlinear (level dependent)
- non-uniform bandwidth, and the bandwidths increase with increasing frequency



Source: Zwicker & Fastl "Psychoacoustics Facts and Models"

Threshold in Quiet or the Absolute Threshold

- Hearing threshold of 100 persons with normal hearing for sine tones (50% curve is the median)

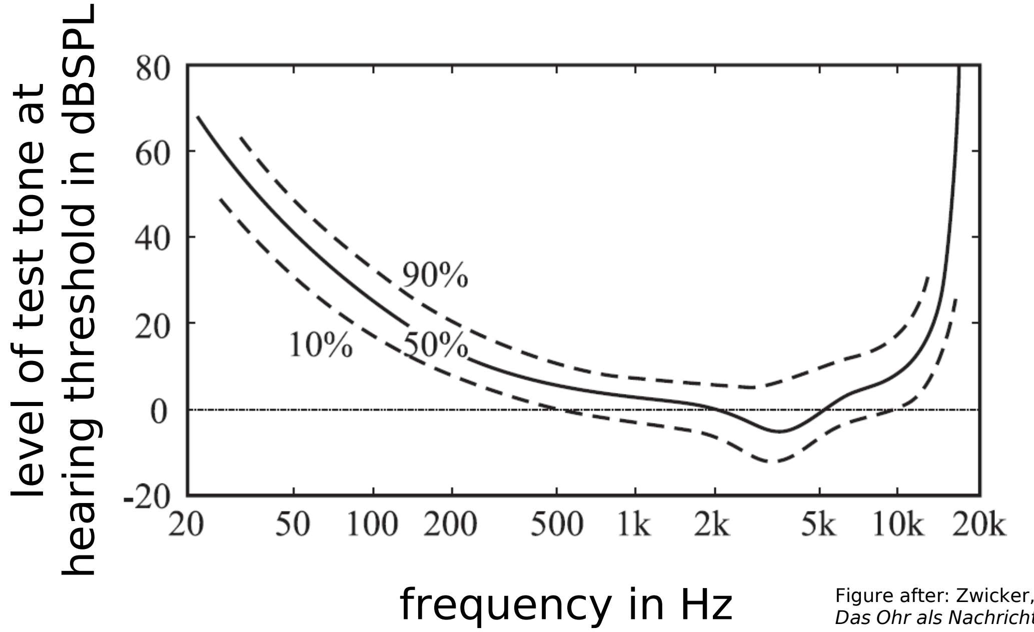


Figure after: Zwicker, E.; Feldtkeller, R. (1967). *Das Ohr als Nachrichtenempfänger*, Hirzel Verlag, Stuttgart.

- Approximations:

$$\frac{L_{Tq}}{dB} = 3.64\left(\frac{f}{kHz}\right)^{-0.8} - exp\left(-0.6\left(\frac{f}{kHz} - 3.3\right)^{2}\right) + 10^{-3}\left(\frac{f}{kHz}\right)^{4}$$

$4\ kHz\ Signal\ with\ Amplitude\ \pm 1\ LSB\ (16\ bit)$

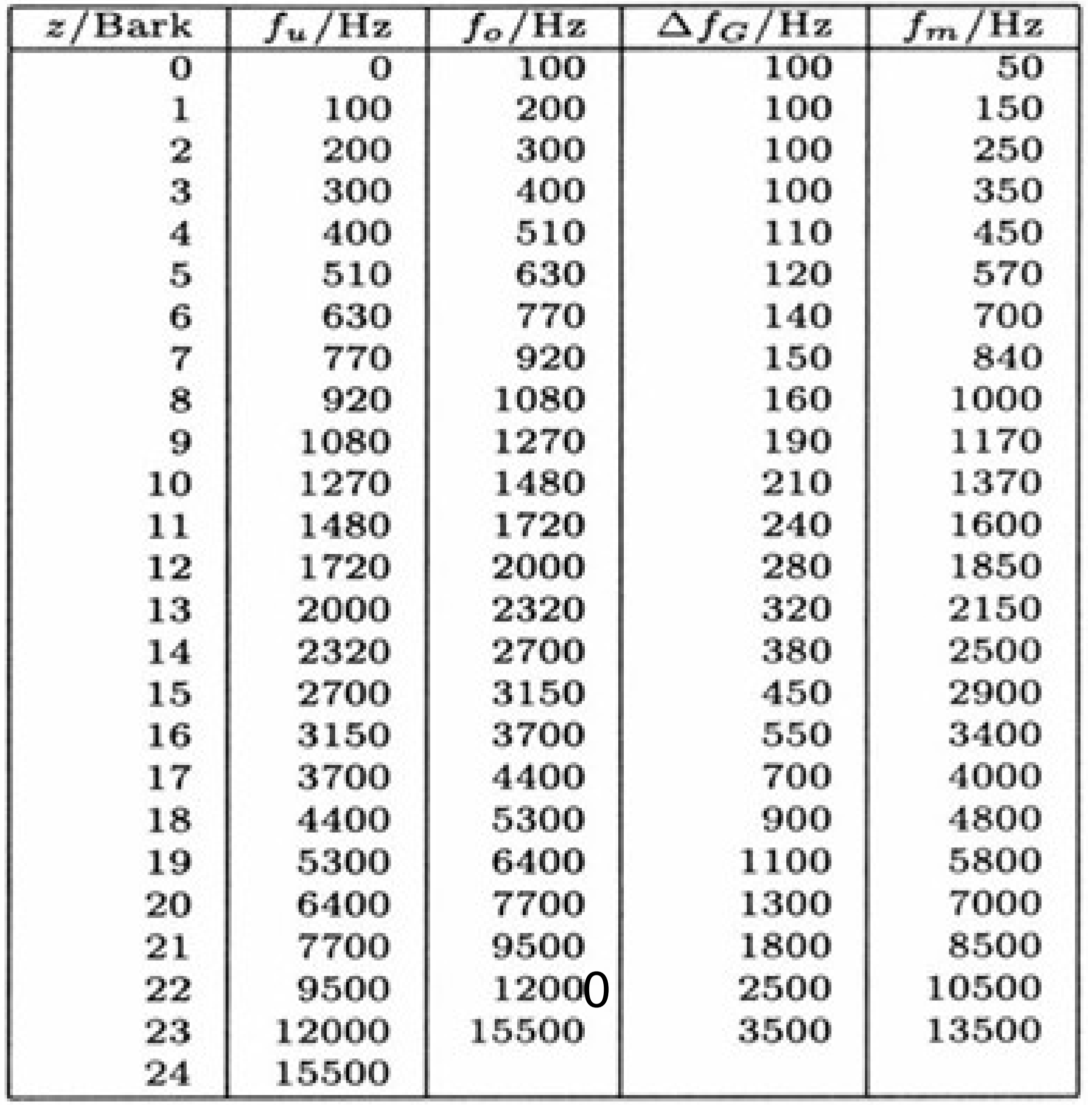
Source: U. Zölzer, “Digital Audio Signal Processing”

Prof. Dr.-Ing. K. Brandenburg, bdg@idmt.fraunhofer.de Prof. Dr.-Ing. G. Schuller, shl@idmt.fraunhofer.de

Critical Bands: Bark Scale

- Critical-band concept used in many models and hypothesis,
- unit was defined leading to so-called critical-band rate scale
- z-scale ranging from 0 – 24, unit “Bark”

- relation between *z* and *f* is important for under-standing many character-istics of human ear



z/Bark	f_u/Hz	f_o/Hz	$\Delta f_G/\mathrm{Hz}$	f_m/Hz
0	0	100	100	50
1	100	200	100	150
2	200	300	100	250
3	300	400	100	350
4	400	510	110	450
5	510	630	120	570
6	630	770	140	700
7	770	920	150	840
8	920	1080	160	1000
9	1080	1270	190	1170
10	1270	1480	210	1370
11	1480	1720	240	1600
12	1720	2000	280	1850
13	2000	2320	320	2150
14	2320	2700	380	2500
15	2700	3150	450	2900
16	3150	3700	550	3400
17	3700	4400	700	4000
18	4400	5300	900	4800
19	5300	6400	1100	5800
20	6400	7700	1300	7000
21	7700	9500	1800	8500
22	9500	12000	2500	10500
23	12000	15500	3500	13500
24	15500			

Critical Bands: Bark Scale

- Critical-band concept used in many models and hypotheses
  →unit was defined leading to so-called critical-band rate scale
- scale ranging from 0 – 24, unit "Bark" (after Zwicker)
- One Bark corresponds to one critical band
- Attempt to approximate critical bands with formulas:

Critical Bandrate z:

$$\frac{z}{Bark} = 13\arctan\left(0.76\frac{f}{kHz}\right) + 3.5\arctan\left(\frac{f}{7.5\,kHz}\right)^2$$

Critical Bandwidth:

$$\Delta f_B = 25 + 75\left(1 + 1.4\left(\frac{f}{kHz}\right)^2\right)^{0.69}$$

Tonality (1)

- Tonality index α:
  - noisy signal: $\alpha = 0$
  - tonal signal: $\alpha = 1$

- System theory
  - Sharp spectral lines = Signal is periodic = Signal is predictable
  - Approximation: If the signal is predictable then it should be periodic
  - Therefore we can use prediction to approximate if a signal is tonal (by periodicity)

Tonality (2)

Example:

- calculate the predictability:

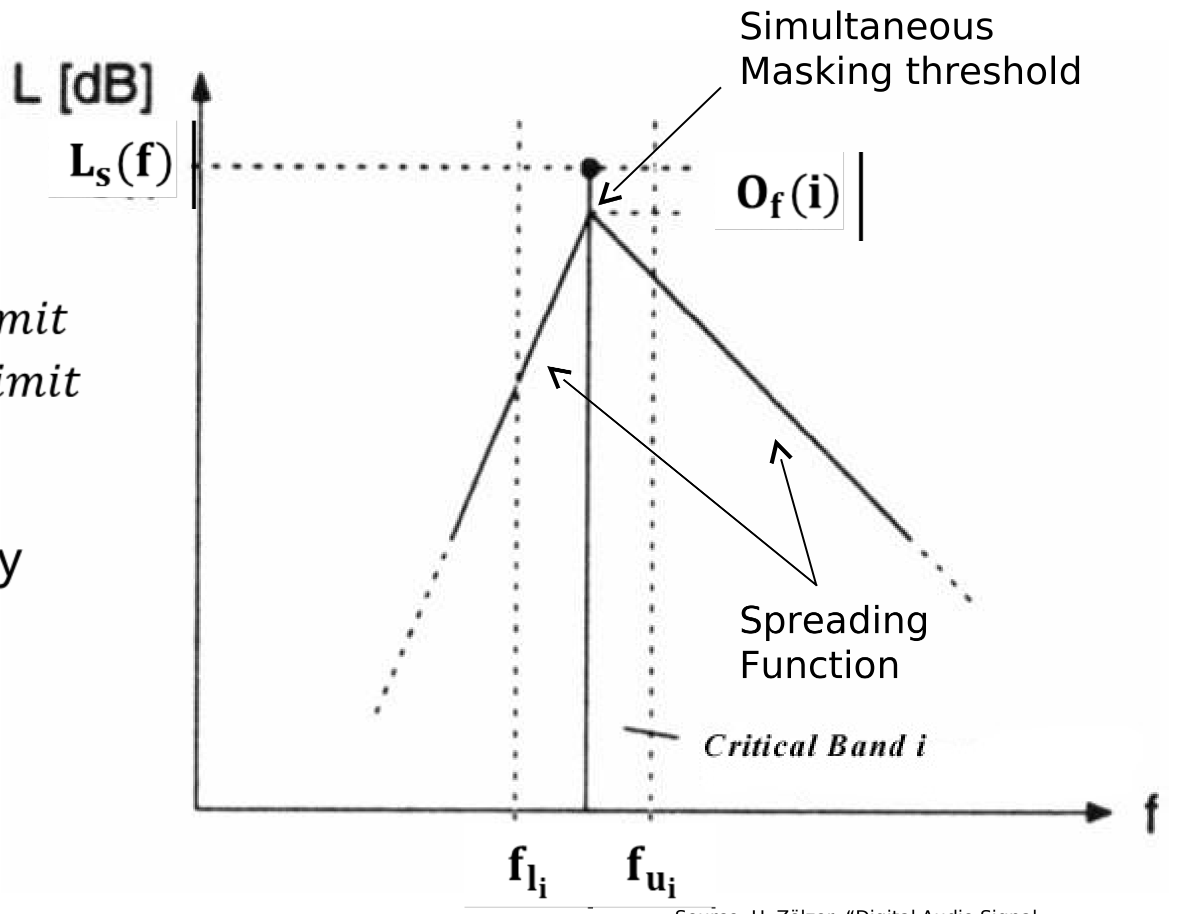
$$c(t,f) = \frac{(\hat{r}(t,f) - r(t,f))^2}{r(t,f)^2} + \frac{\left(\widehat{\Phi}(t,f) - \Phi(t,f)\right)^2}{\Phi(t,f)^2} \qquad \begin{array}{l} c \to 1 : \textit{noisy signal} \\ c \to 0 : \textit{tonal signal} \end{array}$$

- If $c(t,f) > 1$ set it to 1 → $\alpha(t,f) = |c(t,f) - 1|$
- amplitude of a spectral line in time and frequency $r(t,f)$
- phase of a spectral line in time and frequency $\Phi(t,f)$
- predicted values $\hat{r}(l,k)$ for amplitude and $\widehat{\Phi}(l,k)$ for phase

$$\hat{r}(t,f) = r(t-1,f) + \left(r(t-1,f) - r(t-2,f)\right)$$

$$\widehat{\Phi}(t,f) = \Phi(t-1,f) + \left(\Phi(t-1,f) - \Phi(t-2,f)\right)$$

Masking – Spreading Function



f_{l_i} ... *lower frequency limit*
f_{u_i} ... *upper frequency limit*
of Critical Band i

Power Spectral Density of signal x(t)

$$S_p(f) = X_R^2(f) + X_I^2(f)$$

Sound Pressure Level

$$L_S(f) = 10\ \log_{10} S_p(f)$$

Source: U. Zölzer, "Digital Audio Signal Processing"

Calculating the Masking Threshold

Comparison of the signal level to Masking Threshold_

$$\frac{O_f(i)}{dB} = \alpha(14.5 + i) + (1-\alpha)\alpha_v$$

$$\alpha_v = -2 - 2.05\arctan\left(\frac{f}{4\,kHz}\right) - 0.75arctan\left(\frac{f^2}{2{,}56\,kHz^2}\right)$$

α ... *Tonality Index*
α_v ... *Noise Coefficient*

Approximation (α=1: tonal):

$$\frac{O_f(i)}{dB} = \alpha(14.5 + i) + (1-\alpha)5.5$$

Different Masking with different maskers:

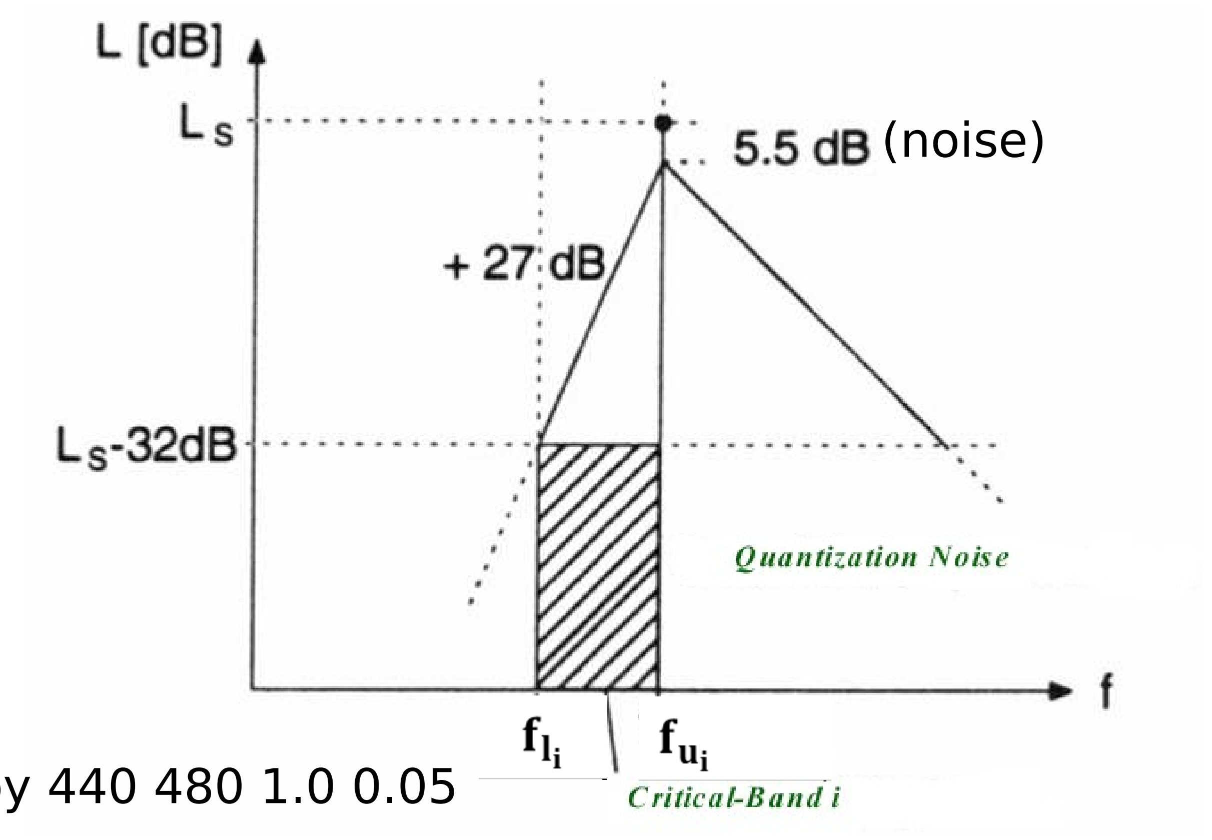
- Tone masking: (14.5 + i) dB, where *i* is the integer number for the critical band
- Noise as a masker: 5.5 dB

Simultaneous Masking Threshold (Power)

$$T(f) = 10^{[L_s(f) - O_f(i)]/10}$$

$L_s(f)$ ... *Sound Pressure Level*
$O_f(i)$ ... *Distance to Masking Threshold*

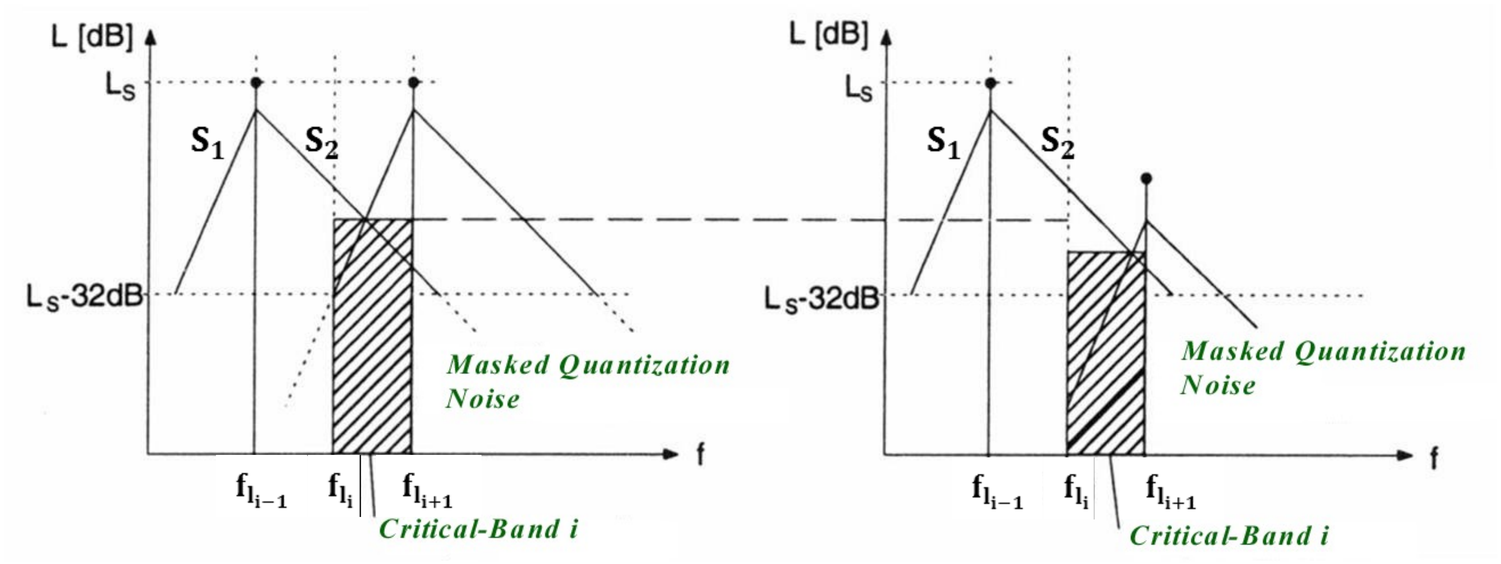
In-Band Masking



Try also:
Python tonarg2tones.py 440 480 1.0 0.05

Masking Neighboring Bands

- spread of masking due to the non-linearity of auditory filters
- resulting masking threshold = sum of power of neighboring spreading functions
- here: value at intersection of neighboring spreading functions taken



$$S_1 = 27 \frac{dB}{Bark}$$

$$S_2 = \left(24 + 0.23 \left(\frac{f}{kHz}\right)^{-1} - 0.2 \frac{L_s(f)}{dB}\right) \frac{dB}{Bark}$$

Masking Neighboring Bands
Non-Linear Superposition

• The total Masking Threshold of the ear results from non-linear superposition.
• resulting masking threshold = sum of **fractional** power of neighbouring spreading functions
•According to Frank Baumgarte, Charalampos Ferekidis, Hendrik Fuchs: “A Nonlinear Psychoacoustic Model Applied to the ISO MPEG Layer 3 Coder”, 99th AES Convention, October 1, 1995.
ftp://mpeg.tnt.uni-hannover.de/pub/papers/1995/AES99-FB.ps.gz
 and
•R. A. Lutfi. ”A Power–Law Transformation Predicting Masking by Sounds with Complex Spectra”. J. Acoust. Soc. Am. 77 (6), June 1985.

• With $I_{T,k}$ the intensity of the k’th speading function (with the “intensity” acting like a power), and a suitable parameter “a” we get the intensity of the total masking threshold as

$$I_T(z_i) = [\sum_k I_{T,k}(z_i)^a]^{1/a}$$

•According to the references, **a=0.3** is in good agreemend with psycho-acoustics

Python Example, Spreading Function

- This Python example shows the non-linear superposition with parameter **2*a=alpha=0.6, in the Bark scale.** We construct a matrix which does the actual superposition in the Bark domain, because that is most efficient:

```
def spreadingfunctionmat(maxfreq,nfilts,alpha):
   #Arguments: maxfreq: half the sampling frequency
   #nfilts: Number of subbands in the Bark domain, for instance 64
   fadB= 14.5+12 # Simultaneous masking for tones at Bark band 12
   fbdb=7.5     # Upper slope of spreading function
   fbbdb=26.0   # Lower slope of spreading function
   maxbark=hz2bark(maxfreq)
   spreadingfunctionBarkdB=np.zeros(2*nfilts)
   #upper slope, fbdB attenuation per Bark, over maxbark Bark (full frequency range), 
with fadB dB simultaneous masking:
   spreadingfunctionBarkdB[0:nfilts]=np.linspace(-maxbark*fbdb,-2.5,nfilts)-fadB
   #lower slope fbbdb attenuation per Bark, over maxbark Bark (full frequency range):
   spreadingfunctionBarkdB[nfilts:2*nfilts]=np.linspace(0,-maxbark*fbbdb,nfilts)-fadB
   #Convert from dB to "voltage" and include alpha exponent
   spreadingfunctionBarkVoltage=10.0**(spreadingfunctionBarkdB/20.0*alpha)
   #Spreading functions for all bark scale bands in a matrix:
   spreadingfuncmatrix=np.zeros((nfilts,nfilts))
   for k in range(nfilts):
      spreadingfuncmatrix[:,k]=spreadingfunctionBarkVoltage[(nfilts-k):(2*nfilts-k)]
   return spreadingfuncmatrix
```

Python Example, Spreading Function

- The application ot the spreading function is then a simple matrix multiplication (which avoids slow “for” loops) in the Bark domain, as in the following Python function:

```
def maskingThresholdBark(mXbark,spreadingfuncmatrix,alpha):
  #Computes the masking threshold on the Bark scale with non-linear superposition
  #usage: mTbark=maskingThresholdBark(mXbark,spreadingfuncmatrix,alpha)
  #Arg: mXbark: magnitude of FFT spectrum,
  #spreadingfuncmatrix: spreading function matrix from function spreadingfunctionmat
  #alpha: exponent for non-linear superposition (eg. 0.6)
  #return: masking threshold as "voltage" on Bark scale

  #mXbark: is the magnitude-spectrum mapped to the Bark scale,
  #mTbark: is the resulting Masking Threshold in the Bark scale, whose components are
  #sqrt(I_tk) on page 13.

  mTbark=np.dot(mXbark**alpha, spreadingfuncmatrix)

  #apply the inverse exponent to the result:
  mTbark=mTbark**(1.0/alpha)

  return mTbark
```

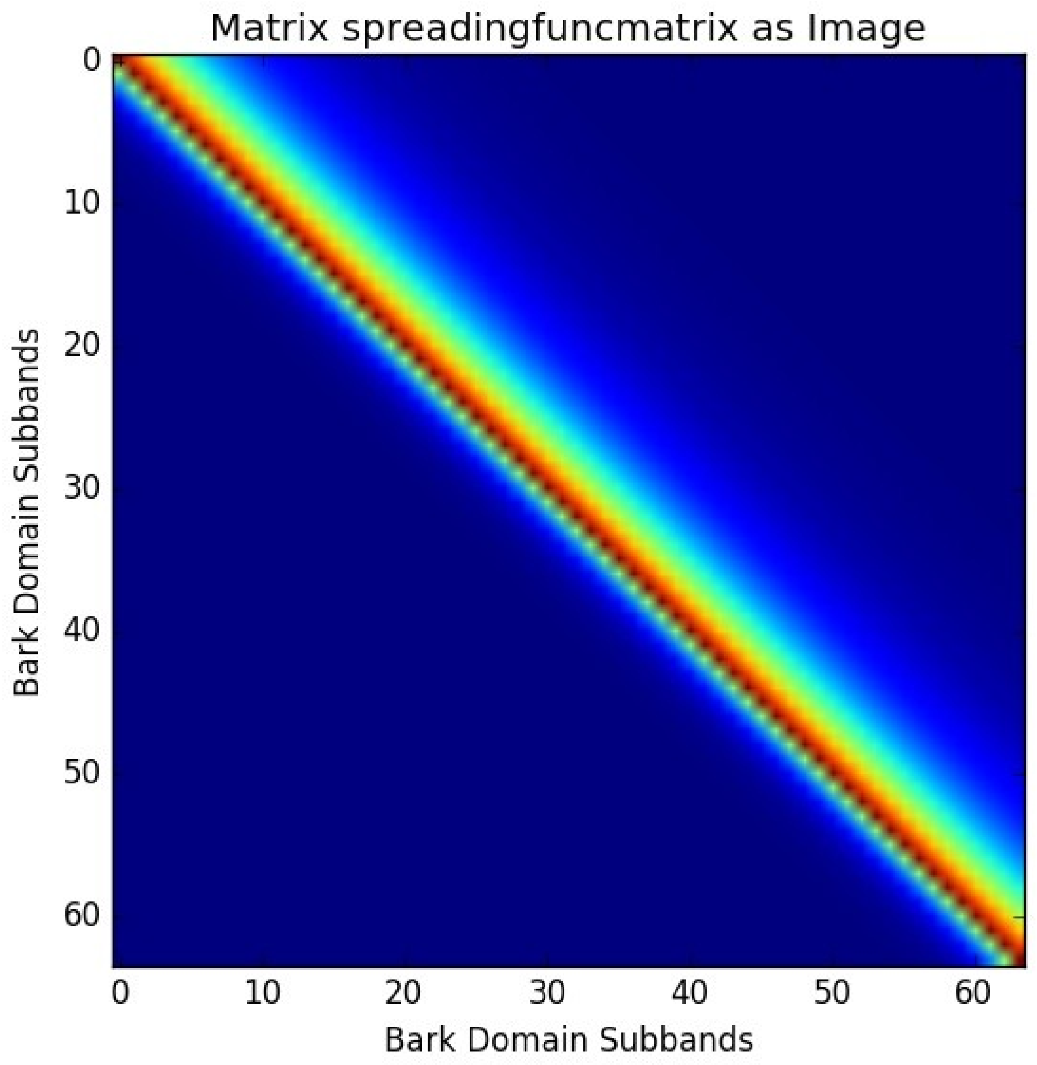
Python Example, Spreading Function

- We can take a look at the resulting spreading function matrix with:

```
from psyacmodel import *
import matplotlib.pyplot as plt
fs=32000  # sampling frequency of audio signal
maxfreq=fs/2
alpha=0.6  #Exponent for non-linear superposition of spreading functions
nfilts=64  #number of subbands in the bark domain

spreadingfuncmatrix=spreadingfunctionmat(maxfreq,nfilts,alpha)
plt.imshow(spreadingfuncmatrix)
plt.title('Matrix spreadingfuncmatrix as Image')
plt.xlabel('Bark Domain Subbands')
plt.ylabel('Bark Domain Subbands')
plt.show()
```

Python Example, Spreading Function



Masking Neighboring Bands Non-Linear Superposition

- Observe that we **don’t need any tonality estimation** for this model!
- Usually our signal from the filter bank is like a “voltage”, not like a power as in this model.
- We obtain a “**power**” if we **square** our signal.
- Hence our exponent is multiplied by a factor of 2.
- We get a → 2*a, hence our exponent becomes 0.6.

- Observe: The frequency index is on the **Bark-scale**, as can be seen in slide 12
- Hence we need a **mapping** from **Hertz to Bark**, from our linear filter bank scale to the bark scale, where we apply masking.
- Then we need an **inverse mapping,** from **Bark to Hertz,** to apply our found masking threshold to the **quantization stepsizes** of our **linearly spaced** subbands.

Bark Scale Approximations

- There are several functional approximations of the Bark scale for this mapping.
- An example of an overview can be seen in https://ccrma.stanford.edu/courses/120-fall-2003/lecture-5.html
- The approximation we previously saw is from:
 Zwicker & Terhardt (1980), "Analytical expressions for critical-band rate and critical bandwidth as a function of frequency", Article in The Journal of the Acoustical Society of America 68(5):1523 · November 1980
- https://www.researchgate.net/publication/209436182_Analytical_expressions_for_critical-band_rate_and_critical_bandwidth_as_a_function_of_frequency
- Also in Wikipedia

- In Python notation, **the approximation** is, with f in Herz and z in Bark:
- z=13*arctan(0.00076*f)+3.5*arctan((f/7500.0)**2)

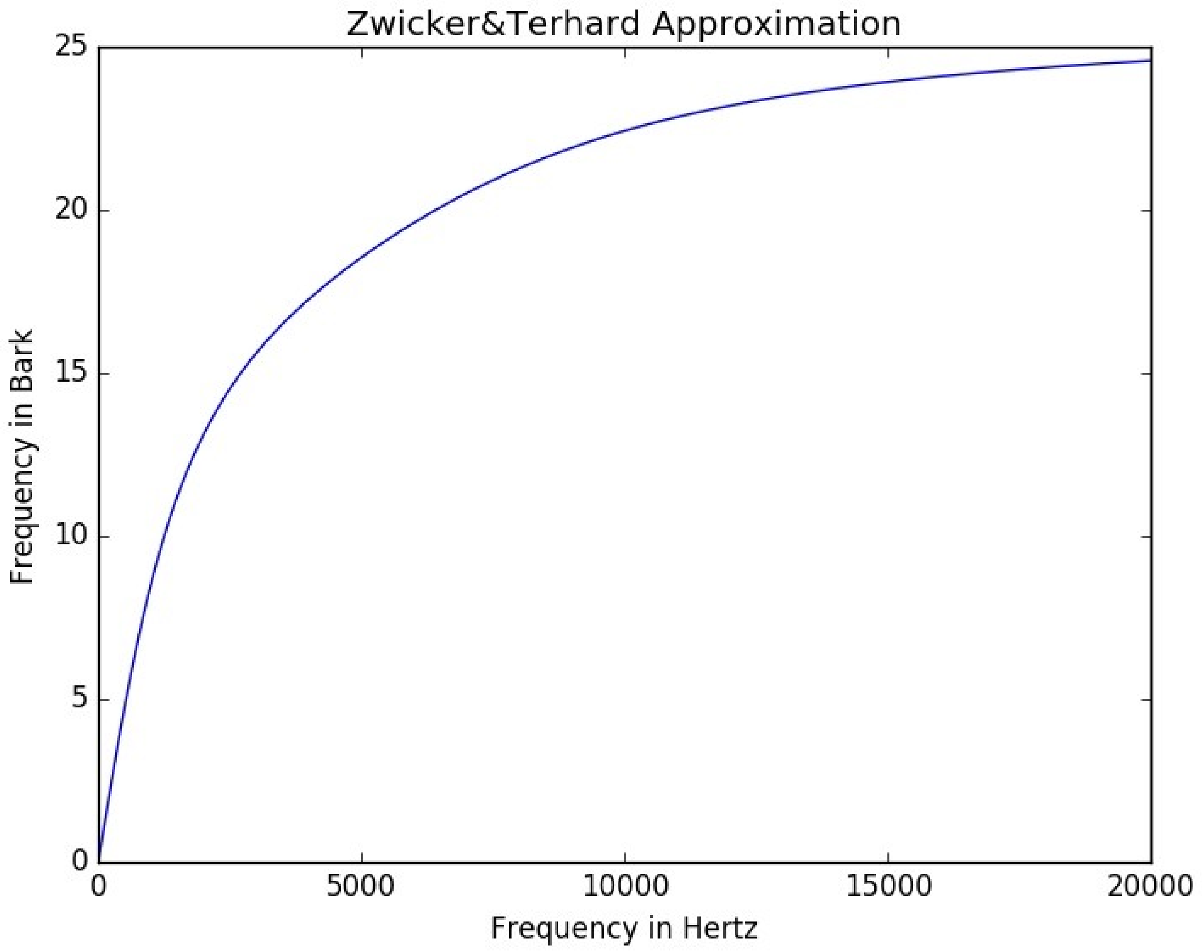
- It only has an **approximate closed form inverse** formula, according to http://www.auditory.org/postings/1995/34.html:
- f= (((exp(0.219*z)/352.0)+0.1)*z-0.032*exp(-0.15*(z-5)**2))*1000

Bark Scale Approximations, Zwicker&Terhard

- We can test the Zwicker & Terhard approximation in ipython:

```
ipython -pylab
#Frequency array between 0 and 20000 Hz in 1000 steps:
f=linspace(0,20000,1000)
#Computation of Zwickers Bark approximation formula:
z=13*arctan(0.00076*f)+3.5*arctan((f/7500.0)**2)
#plot Bark over Hertz:
plot(f,z)
xlabel('Frequency in Hertz')
ylabel('Frequency in Bark')
title('Zwicker&Terhard Approximation')
```

Bark Scale Approximations, Zwicker&Terhard

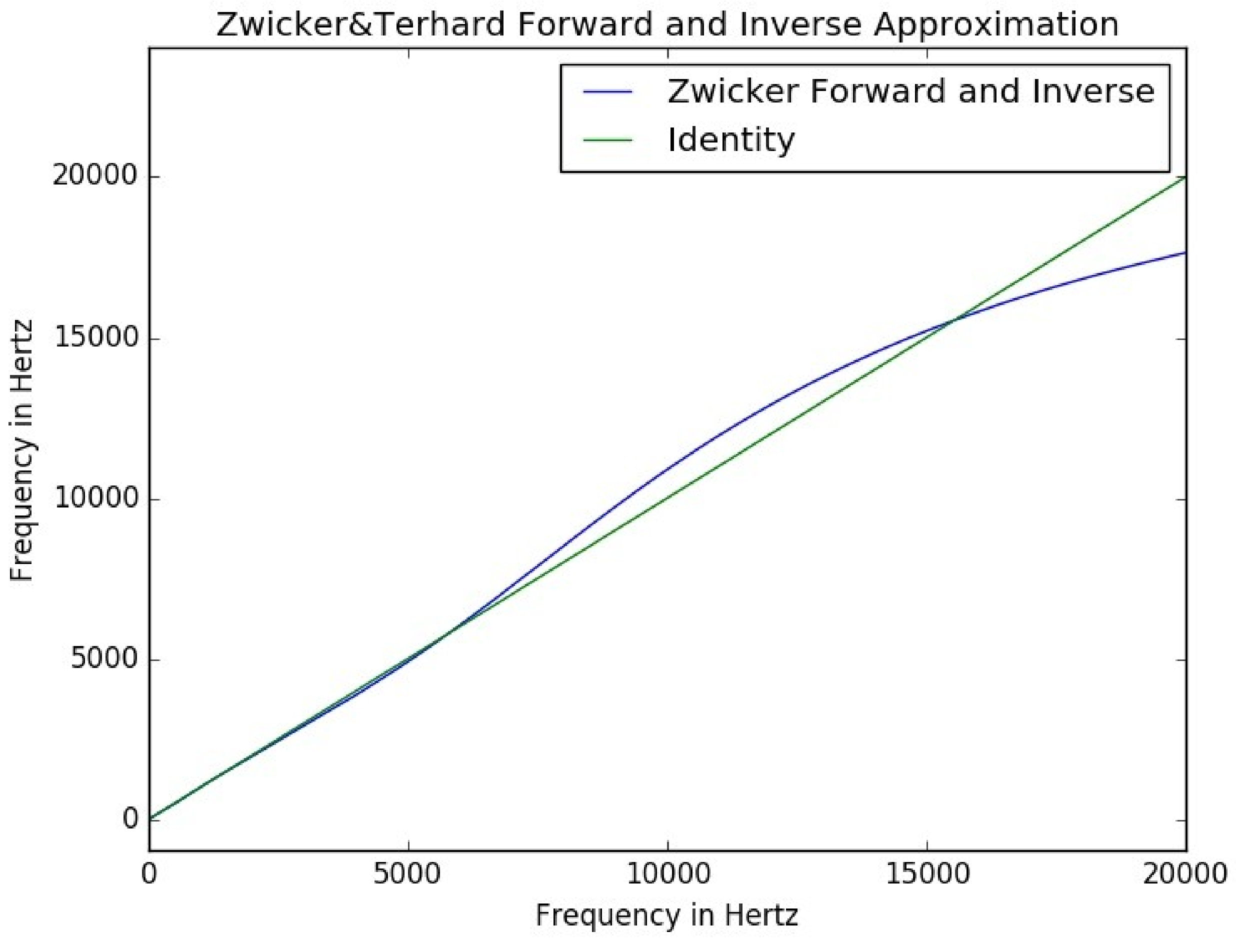


Bark Scale Approximations, Zwicker&Terhard, Inverse

- We can test the Zwicker & Terhard **inverse** approximation in ipython:

```
ipython -pylab
#Frequency array between 0 and 20000 Hz in 1000 steps:
f=linspace(0,20000,1000)
#Computation of Zwickers Bark approximation formula:
z=13*arctan(0.00076*f)+3.5*arctan((f/7500.0)**2)
#computation of the approximate inverse, frec: reconstructed freq.:
frec= (((exp(0.219*z)/352.0)+0.1)*z-0.032*exp(-0.15*(z-5)**2))*1000
#plot reconstructed freq. Over original freq:
plot(f,frec)
#comparison: identity:
plot(f,f)
xlabel('Frequency in Hertz')
ylabel('Frequency in Hertz')
title('Zwicker&Terhard Forward and Inverse Approximation')
legend(('Zwicker Forward and Inverse','Identity'))
```

Bark Scale Approximations, Zwicker&Terhard Inverse



Bark Scale Approximations, Traunmueller

- Traunmueller-formula, 1990, from:
- Traunmüller, H. (1990). "Analytical expressions for the tonotopic sensory scale". The Journal of the Acoustical Society of America.
- Also in Wikipedia:
- In Python notation, **the approximation** is, with f in Herz and z in Bark:
- for **above 200 Hz**:
  z=26.81*f/(1960.0+f)-0.53
- **below 200Hz**:
  z= f/102.9

- It has an **exact inverse**:
- **Above 200 Hz**:
  f=1960.0/(26.81/(z+0.53)-1)
- **Below 200 Hz**:
  f=z*102.9

Bark Scale Approximations, Schröder

- -Schroeder, M. R. (1977). Recognition of Complex, Acoustic
- Signal & Life Sciences Research Report 5, edited by T. H. Bullock (Abakon Verlag, Berlin), p. 324.
- See also: "Perceptual linear predictive (PLP) analysis of speech" by Hynek Hermansky, J. AcousL Soc. Am. 87 (4). April 1990, (http://seed.ucsd.edu/mediawiki/images/5/5c/PLP.pdf)
- It is eq. (3), for angular frequency, which is in turn from Schroeder above
- Also used in the PEAQ standard for objective quality estimation (eq. (2) in the paper:
- https://www.ee.columbia.edu/~dpwe/papers/Thiede00-PEAQ.pdf
- "PEAQ--The ITU Standard for Objective Measurement of Perceived Audio Quality", THILO THIEDE et al., J. Audio Eng. Soc., Vol. 48, No.1/2, 2000 January/February

- **It is the simplest Approximation**:
  z= 6*arcsinh(f/600.0)

- It has an **exact inverse**, Bark to Hertz:
  f=600 * sinh(z/6.0)

Bark Scale Approximations, Comparisons

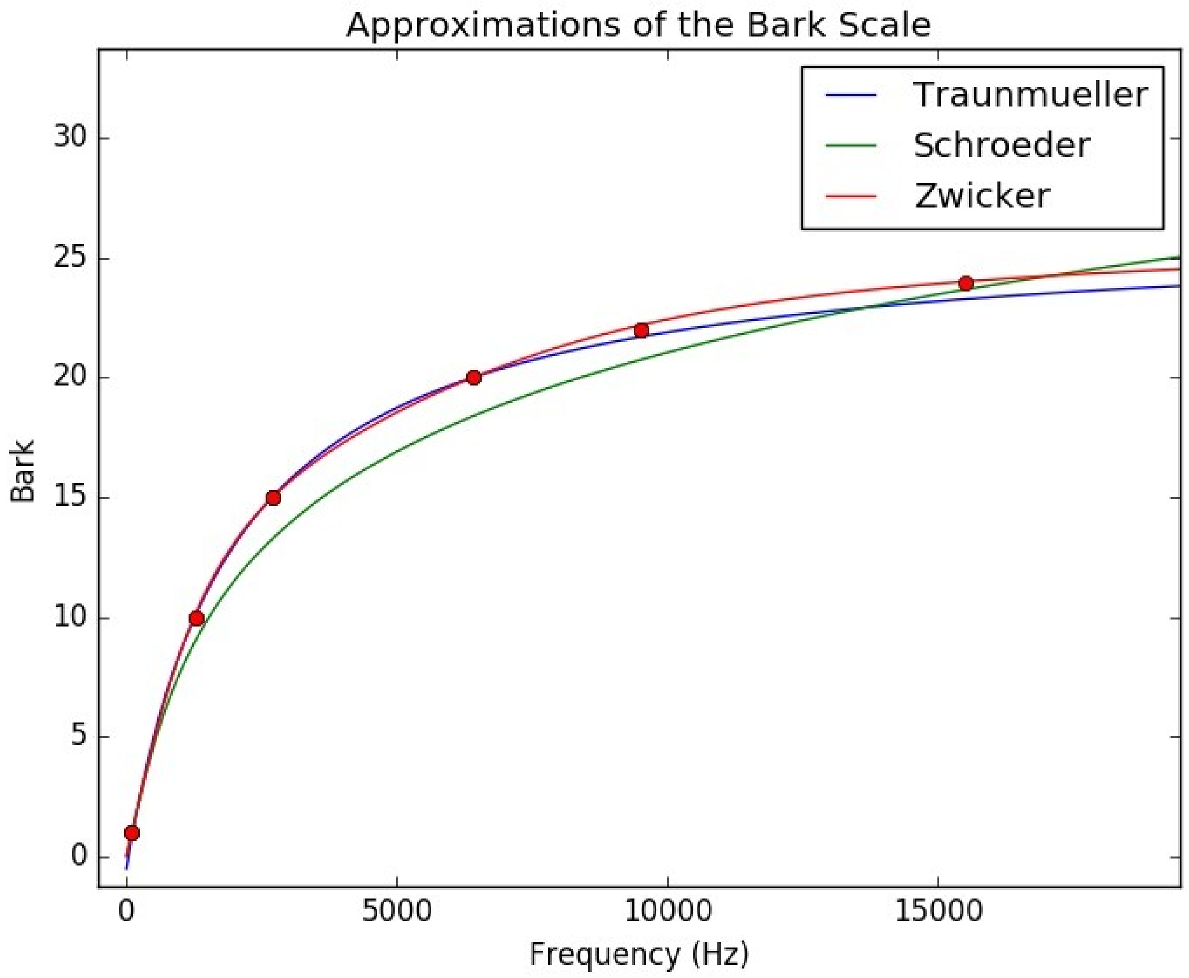
- Comparison of our functional approximation with our Bark-Table.
- The approximation formulas also give fractional Bark numbers, and the integer Bark numbers correspond to unique frequencies, which are a band limit.
- Tables name bands after an integer Bark number, but they differ in if the band above or below is named after that number.
- In the lecture table this integer Bark number corresponds to the lower limit of the band, hence it starts with index 0, in other literature (CCRMA Webpage) and Wikipedia to the upper limit, starting with index 1!
- We use these pairs out of the table for our comparison:
- 1 bark - 100Hz
- 10 Bark - 1270Hz
- 15 - 2700 Hz
- 20 - 6400 Hz
- 22 - 9500 Hz

Bark Scale Approximations, Comparisons

- Use ipython for the comparison:

```
ipython --pylab
f=arange(0,20000,10)
z=26.81*f/(1960.0+f)-0.53 #Traunmueller
plot(f,z)
z= 6*arcsinh(f/600.0)  #Schroeder
plot(f,z)
z=13*arctan(0.00076*f)+3.5*arctan((f/7500.0)**2) #Zwicker
plot(f,z)
legend(('Traunmueller','Schroeder','Zwicker'))
#plot single comparison points:
plot([100,1270,2700,6400,9500,15500],[1,10,15,20,22,24],'ro')
xlabel('Frequency (Hz)')
ylabel('Bark')
title('Approximations of the Bark Scale')
```

Bark Scale Approximations, Comparisons



Bark Scale Approximations

- **Observe:** The Zwicker approximation is the most precise, it hits our test points, but it has no closed form inverse.

- The Schroeder approximation is the least accurate, but it is the simplest and it has an exact inverse in closed form, hence it is used most often, and we will also use it.
-
- In Python we use the function:

```
def hz2bark(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.)

        return Brk
```

Bark Scale Approximations

- The inverse function in Python is:

```
def bark2hz(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
```

Bark Scale Mapping

•We choose 64 subbands in the Bark scale, hence each about 1/3 Bark wide.

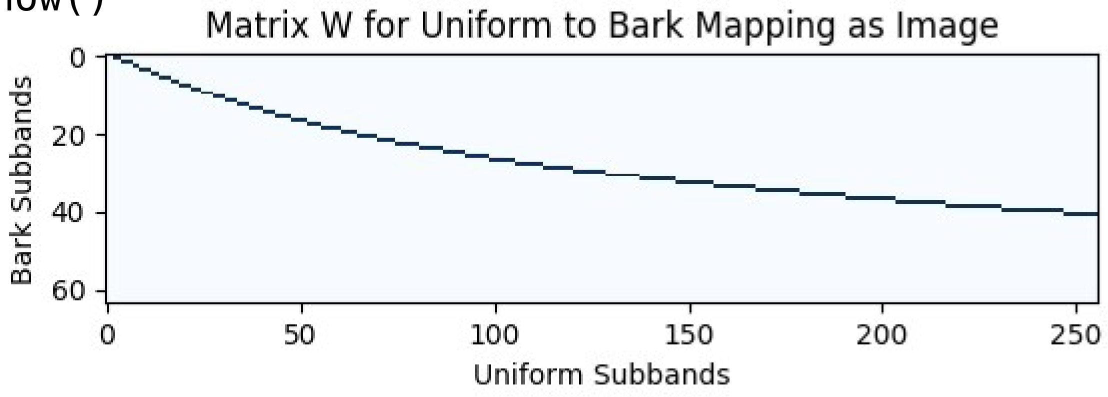
•In Python we construct a matrix W for this mapping (again, to avoid slow “for” loops), which has 1’s at the position of each such 1/3 Bark band:

```
def mapping2barkmat(fs, nfilts,nfft):
  #Constructing matrix W which has 1’s for each Bark subband, and 0’s else:
  #nfft=2048; nfilts=64;
  nfreqs=nfft/2
  #the linspace produces an array with  the fft band edges:
  binbarks = hz2bark(np.linspace(0,(nfft/2),(nfft/2)+1)*fs/nfft)
  W = np.zeros((nfilts, nfft))
  for i in xrange(nfilts):
     W[i,0:(nfft/2)+1] = (np.round(binbarks/step_barks)== i)
  return W
```

Bark Scale Mapping

- Matrix W as image in Python:

```
fs=32000
W=mapping2barkmat(fs,nfilts=64,nfft=2048)
plt.imshow(W[:,:256],cmap='Blues')
plt.title('Matrix W as Image')
plt.xlabel('Uniform Subbands')
plt.ylabel('Bark Subbands')
plt.show()
```



Bark Scale Mapping

- For each such 1/3 bark subband we add the signal powers from the corresponding DFT bands.

- Then we take the square root to obtain a “voltage” again.

- As Python function:

```
def mapping2bark(mX,W,nfft):
    #Maps (warps) magnitude spectrum vector mX from DFT to the Bark scale
    #returns: mXbark, magnitude mapped to the Bark scale
    #Frequency of each FFT bin in Bark, in 1025 frequency bands (from call)
    #nfft=2048; nfilts=64;
    nfreqs=nfft//2
    #Frequencies of each FFT band, up to Nyquits frequency, converted to Bark:
    #Here is the actual mapping, suming up powers and conv. back to Voltages:
    mXbark = (np.dot( np.abs(mX[:nfreqs])**2.0, W[:, :nfreqs].T))**(0.5)
    return mXbark
```

Mapping from Bark scale back to Linear

•After having computed the masking threshold in the Bark scale, we need to map it back to the linear scale of our filter bank
•For that we need to “distribute” the corresponding power of each of our 1/3 Bark bands into the corresponding filter bank bands on the linear frequency scale

•Then we take the square root to obtain a “voltage” again.

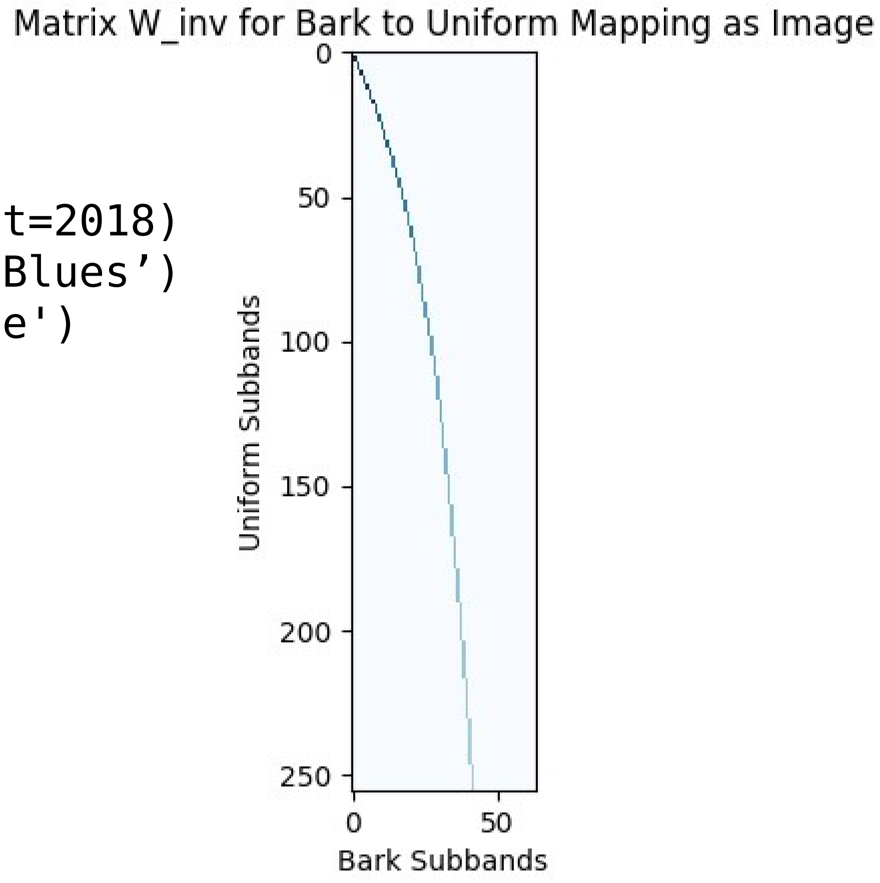
•We again contruct a matrix to do that in Python. When there is 1 subband in the 1/3 bark scale, it gets a factor 1, if there are 2 subbands, they get a factor of sqrt(2), and so on, using a diagonal matrix multiplication for those factors. It is an 64x1024 matrix:

```
def mappingfrombarkmat(W,nfft):
      #Constructing matrix W_inv from matrix W for mapping back from bark scale
      #nfft=2048;
      nfreqs=nfft//2
      W_inv= np.dot(np.diag((1.0/np.sum(W,1))**0.5), W[:,0:nfreqs + 1]).T
      return W_inv
```

Mapping from Bark scale back to Linear

- Matrix W_inv as image in python:

```
W_inv=mappingfrombarkmat(W, nfft=2018)
plt.imshow(W_inv[:256,:],cmap=’Blues’)
plt.title('Matrix W_inv as Image')
plt.xlabel('Bark Subbands')
plt.ylabel('Uniform Subbands')
plt.show()
```



Prof. Dr.-Ing. K. Brandenburg, bdg@idmt.fraunhofer.de Prof. Dr.-Ing. G. Schuller, shl@idmt.fraunhofer.de

Mapping from Bark scale back to Linear

- The function for mapping the masking threshold from Bark scale to linear scale is

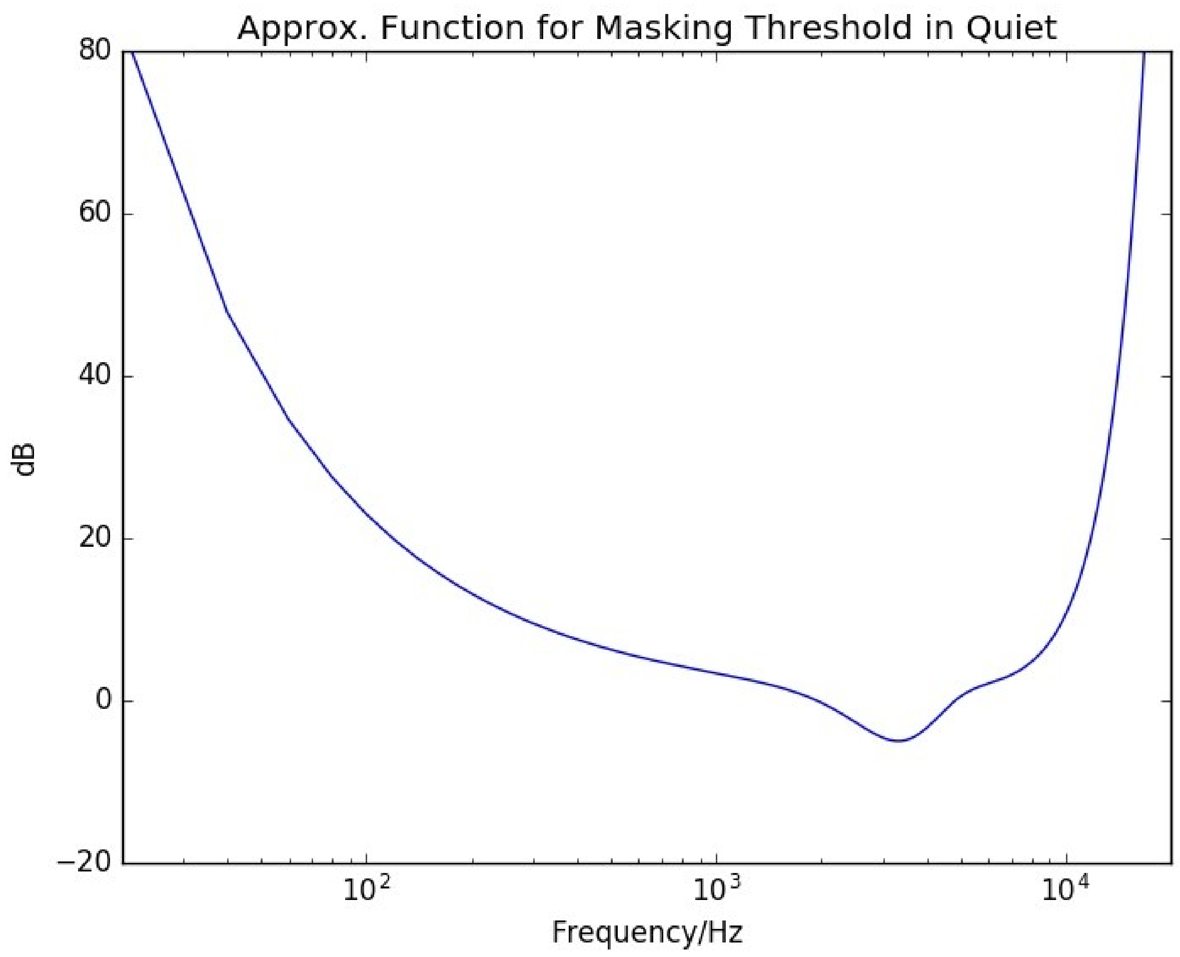
```
def mappingfrombark(mTbark,W_inv, nfft=2048):
  #Maps (warps) magnitude spectrum vector mTbark in the Bark scale
  # back to the linear scale
  #returns: mT, masking threshold in the linear scale
  nfreqs=nfft/2
  mT = np.dot(mTbark, W_inv[:, :nfreqs].T)
  return mT
```

Hearing Threshold in Quiet

•On top of our signal adaptive masking threshold, we have the threshold in quiet.
•We have an approximation formula from Zoelzer: “Digital Audio Signal Processing”
• For the case of quiet and only a barely audible test tone.
• The approximation for this Level of the **Threshold in Quiet**, LTQ, **in dB** and in Python notation is:
LTQ=3.64 * (f/1000.) **(-0.8) - 6.5*np.exp(-0.6 * (f/1000. - 3.3) ** 2.) + 1e-3*((f/1000.) ** 4.)
•Plot it with ipython:

```
 ipython --pylab
 f=linspace(20,20000,1000)
 LTQ=3.64*(f/1000.)**-0.8 -6.5*np.exp(-0.6*(f/1000.-3.3)**2.)+1e-
3*((f/1000.)**4.)
 semilogx(f,LTQ)
 axis([20,20000, -20,80])
 xlabel('Frequency/Hz')
 ylabel('dB')
 title('Approx. Function for Masking Threshold in Quiet')
```

Hearing Threshold in Quiet



Prof. Dr.-Ing. K. Brandenburg, bdg@idmt.fraunhofer.de Prof. Dr.-Ing. G. Schuller, shl@idmt.fraunhofer.de

Hearing Threshold in Quiet

•The dB of the formula is for sound pressure. Our internal representation has +-1 as a full scale, which corresponds to 0 dB. Assume we play back our audio signal such that full scale appears at a sound level of speech, which is about 60 dB. Hence to convert the sound level to our internal representation, we need to **reduce** the threshold of quiet by **60 dB**.

•Even with an audio signal the masking threshold in quiet still matters at the lowest and highest frequencies.
•We **combine** the signal dependent masking threshold and the threshold in quiet by taking the **maximum** of the two at each frequency.
•In Python we clip the result to avoid overloading and numerical problems, and correct our masking threshold with:

```
LTQ=np.clip(LTQ,-20,60)
#Shift dB according to our internal representation:
LTQ=LTQ-60
```

Prof. Dr.-Ing. K. Brandenburg, bdg@idmt.fraunhofer.de Prof. Dr.-Ing. G. Schuller, shl@idmt.fraunhofer.de

Hearing Threshold in Quiet, Testing

• We can test our approximation formula for our hearing threshold in quiet by producing noise below this spectral threshold, and then listen to it. If we **don't hear the noise** it works!
 We can use the Python function in our program `maskinginquietdemo.py` (see our Moodle page):

```
  noisefromdBSpectrum(spec,fs)
```

•With: spec: spectral shape of the produced noise in dB, fs: sampling rate

•Then we can listen to the sound corresponing to our threshold approximation with with:

```
   f=np.linspace(0,fs/2,N)
   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)
   #Shift dB according to our internal representation:
   LTQ=LTQ-60
   #Play back noise shaped like the masking theshold in quoet:
   noisefromdBSpectrum(LTQ,fs)
```

Hearing Threshold in Quiet, Testing

- We can start the complete demo with:
- `Python maskinginquietdemo.py`

- Observe: White noise (flat spectrum) is clearly audible
- Noise shaped according to our threshold approximation should be inaudible!

Prof. Dr.-Ing. K. Brandenburg, bdg@idmt.fraunhofer.de Prof. Dr.-Ing. G. Schuller, shl@idmt.fraunhofer.de

The Complete Psycho-Acoustic Model

•Now our complete psycho-acoustic model for the computation of our masking threshold is:

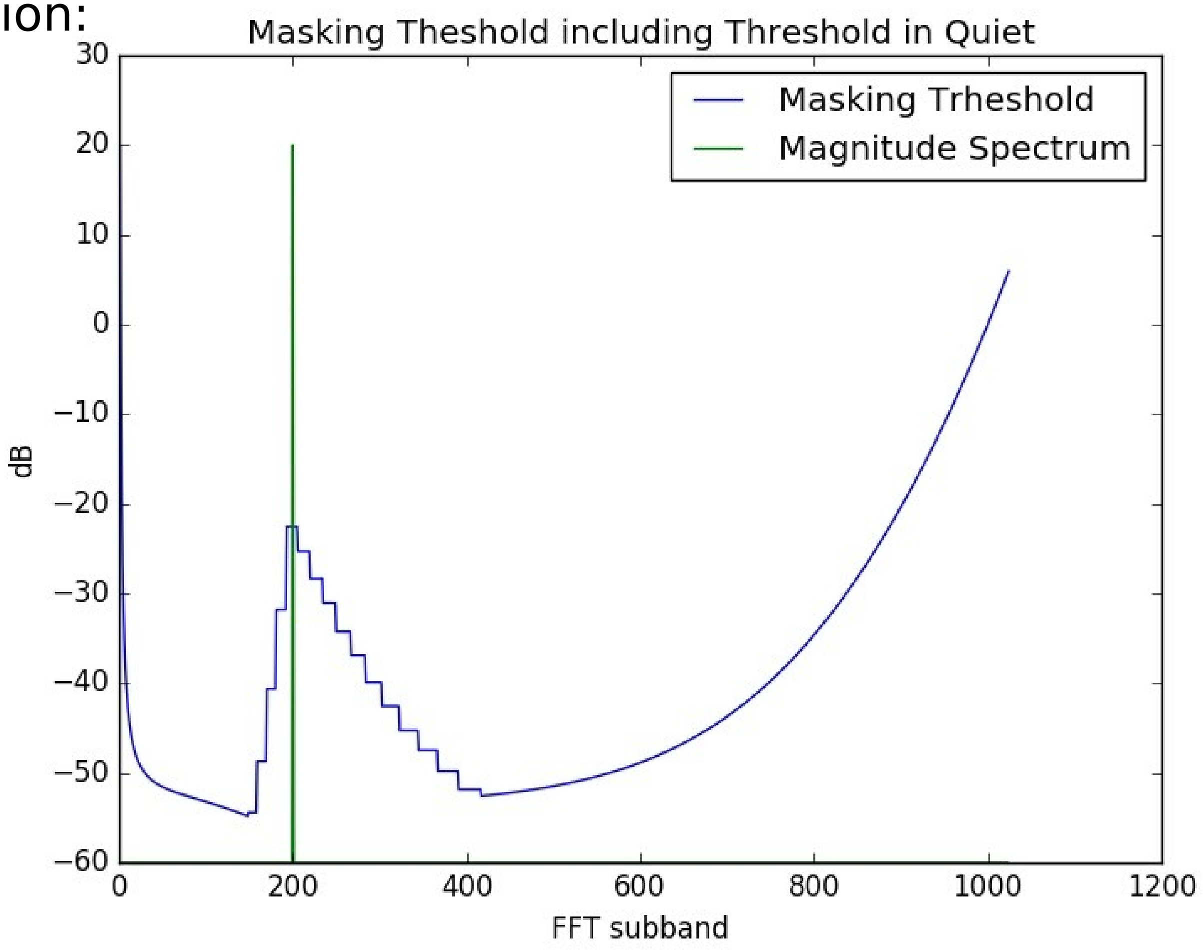
```
fs=32000
W=mapping2barkmat(fs)
W_inv=mappingfrombarkmat(W)

def maskingThreshold(mX, W, W_inv,fs,spreadingfuncmatrix,alpha,nfft):
   #Input: magnitude spectrum of a DFT of size 2048
   #Returns: masking threshold (as voltage) for its first 1025 subbands

   #Map magnitude spectrum to 1/3 Bark bands:
   mXbark=mapping2bark(mX,W, nfft)
   #Compute the masking threshold in the Bark domain:
   mTbark=maskingThresholdBark(mXbark,spreadingfuncmatrix,alpha)
   #Map back from the Bark domain,
   #Result is the masking threshold in the linear domain:
   mT=mappingfrombark(mTbark,W_inv,nfft)
   #Threshold in quiet:
   f=np.linspace(0,fs/2,1025)
   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.)),-20,80)
   mT=np.max((mT, 10.0**((LTQ-60)/20)),0)
   return mT
```

The Complete Psycho-Acoustic Model

This example is an idealized tone in one subband, and its resulting masking threshold, which is mostly its spreading function:



The Complete Psycho-Acoustic Model

•**Example,** complete demo:

```
python3 psyacmodel.py
```

•Demo with according quantization of the subband values:
python3 psyac_quantization.py

•Real-Time Audio Demo:

```
python3 psycho-acoustic-modelDFT_gs.py
```

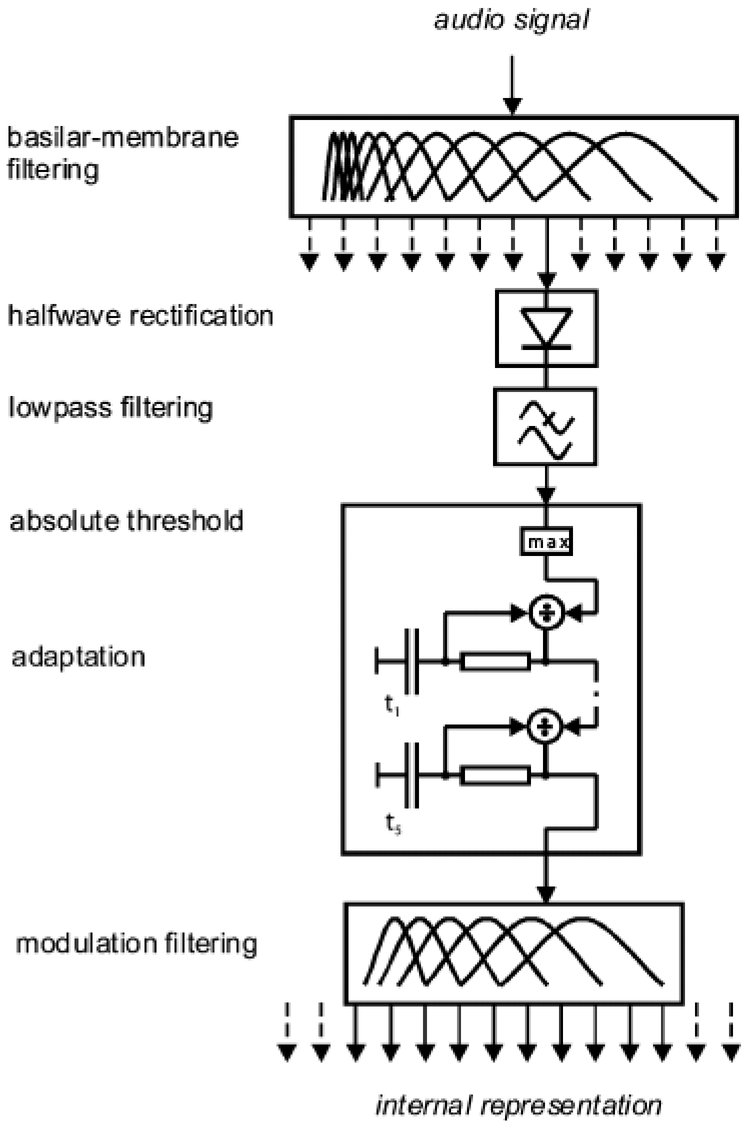
•Try different inputs:
- •Silence, to see the threshold in quiet.
- •A tone, to see its spreading function.
- •A complex music signal, to see its masking threshold.

•**Observe**: here we can use music or a sinusoidal tone of 1 kHz frequency as input, and shift th masking threshold in the dB domain to find the precise threshold at which the added noise becomes inauddible.

Physical Models of Hearing

•Physical models doen't model the effects of hearing, but the **physical functioning** of the **inner ear** instead.
•As a result, their output is an **internal representation**, not a masking threshold
•But they can still be used to compute a **similarity measure** of 2 different sounds, as perceived by the human ear, by **comparing their internal representations.**
•An example is the "**PEMO-Q**" measure, to estimate the "quality" of a sound compared to an original.
• [1]: http://ieeexplore.ieee.org/Xplore/login.jsp?url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F10376%2F36074%2F01709880.pdf&authDecision=-203
•It is used as part of "**PEASS**" toolkit.
•(https://hal.inria.fr/inria-00567152/PDF/emiya2011.pdf)
•This is used for instance for measuring the quality of **audio source separtion**.

Physical Models of Hearing, PEMO Model



From [1] on previous slide