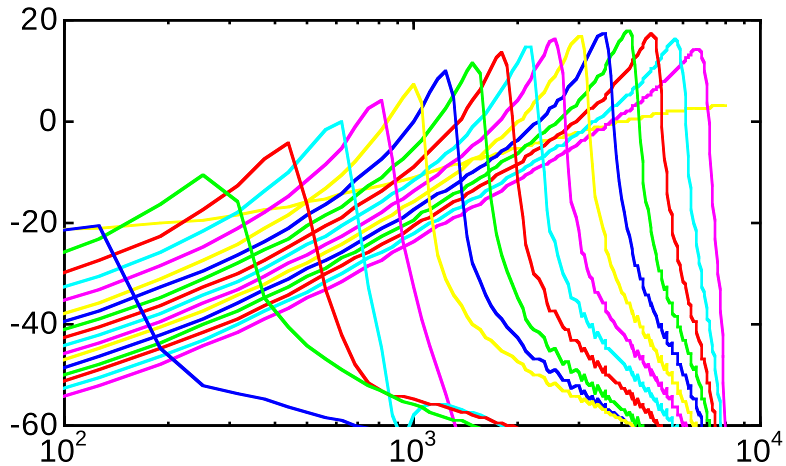
Auditory Toolbox

Malcolm Slaney
Apple Technical Report #45
Apple Computer, Inc.
Advanced Technology Group
malcolm@apple.com



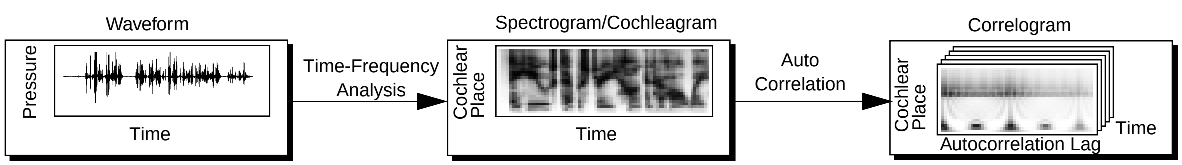
Auditory Toolbox:
A MATLAB Toolbox for Auditory Modeling Work

Apple Computer Technical Report #45

Malcolm Slaney
malcolm@apple.com

This report describes a collection of tools that implement several popular auditory models for a numerical programming environment called MATLAB. This toolbox will be useful to researchers that are interested in how the auditory periphery works and want to compare and test their theories. This toolbox will also be useful to speech and auditory engineers who want to see how the human auditory system represents sounds.

There are many ways to describe and represent sounds. The figure below shows one taxonomy based on signal dimensionality. A simple waveform is at the one-dimensional level. The two-dimensional level describes the acoustic signal as a time-frequency image. This is the typical approach for sound and speech analysis. This toolbox includes the conventional short-time-Fourier-Transform (STFT or Spectrogram) and several cochlear models that estimate auditory nerve firing “probabilities” as a function of time. Finally, the next level of abstraction is to summarize the periodicities of the cochlear output with the correlogram. The correlogram provides a power representation that makes it easier to understand multiple sounds and to perform auditory scene analysis.



Five time-frequency representations are implemented in this toolbox:

1) Conventional FFT analysis is represented using the spectrogram. Both narrow band and wide band spectrograms are possible. See the *spectrogram* command for more information.
2) A common front-end for many speech recognition systems consists of Mel-frequency cepstral coefficients (MFCC). This technique combines an auditory filter-bank with a cosine transform to give a rate representation roughly analogous to the auditory system. See the *mfcc* command for more information.
3) Richard F. Lyon has described an auditory model based on a transmission line model of the basilar membrane and followed by several stages of adaptation. This model can represent sound at either a fine time scale (probabilities of an auditory nerve firing) or at the longer time scales characteristic of the spectrogram or MFCC analysis. The *LyonPassiveEar* command implements this particular ear model.
4) Roy Patterson has proposed a model of psychoacoustic filtering based on critical bands. This auditory front-end combines a Gammatone filter bank with a model of hair cell dynamics proposed by Ray Meddis. This auditory model is implemented using the *MakeERBFilters*, *FilterBank*, and *MeddisHairCell* commands.
5) Finally, Stephanie Seneff has described a cochlear model that combines a critical band filterbank with models of detection and automatic gain control. This toolbox implements stages I and II of her model.

Our work here at Apple has concentrated on how to capture and represent the information in our auditory environment. Towards this goal, we have been investigating the correlogram. The primary goal of the correlogram is to summarize the temporal activity at the output of the cochlea. With most sounds, and especially with voiced speech, much of the information in the waveform and cochlear output is repetitive. The correlogram is an easy way to capture the periodicities and make them visible. This toolbox includes several routines to compute and display correlograms, and to compute pitch estimate from correlograms.

This toolbox has a very simple view of data. Sound waveforms are stored as one-dimensional arrays. The output from cochlear models is stored as a two-dimensional array, with columns of the matrix representing firing probabilities on the auditory nerve at one time. Correlograms can be stored as either movies or as an array. Filter coefficients are either stored as two lists, like the MATLAB filter function, or second-order-sections are stored as a list of five coefficients.

Users of this package might also be interested in the Sound and Image Toolbox. This toolbox includes tools to read and write sound files stored in many common formats. This tools also includes additional MATLAB tools that make it easy to record sounds and work with images on Macintosh computers.

This report is not a detailed description of each auditory model. Most function descriptions include references to more detailed descriptions of each model.

This software has been tested on Macintosh computers running MATLAB 4.1 and on Sun and HP workstations. All of this code is pretty portable so we don't expect any problems when running on any other machine that runs MATLAB.

Finally, a word from our lawyers:

> Warranty Information: Even though Apple has reviewed this software, Apple makes no warranty or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. As a result, this software is provided "as is," and you, its user, are assuming the entire risk as to its quality and accuracy.

A flowchart showing how all the commands in this toolbox fit together is shown in the next section.

Installation

This toolbox is supplied as a collection of MATLAB m-functions and three MEX functions written in C. The three MEX functions, *agc*, *soscascade*, and *sosfilters*, are precompiled for the Macintosh. You will need to compile them yourself for other machines using the Mathworks cmex function. Use the example code, included with the documentation for each function, to test each function.

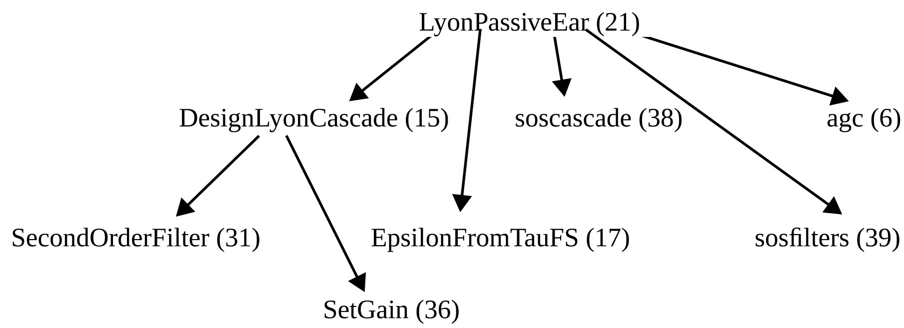
References

Malcolm Slaney, "Sound and Image Toolbox," Apple Computer Technical Report #61, (Apple Corporate Library, Cupertino, CA 95014,) April 1995.

Flow Charts

This section shows which routines are used by each function in this toolbox. This will help readers understand the structure of the cochlear models. Page numbers are shown in parenthesis.

Lyons Passive Long Wave Cochlear Model



Patterson-Holdsworth ERB Filter Bank

MakeERBFilters (23)   FilterBank (18)   MeddisHairCell (26)

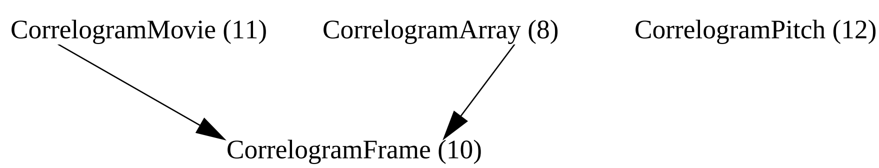
Seneff Auditory Model

SeneffEar (33) → SeneffEarSetup (35)

Alternate Analysis Techniques

mfcc (28)   spectrogram (41)

Correlogram Processing



Miscellaneous

MakeVowel (25)   FMPoints (19)

agc

Purpose

Adaptation process for Lyon's passive longwave cochlear model

Synopsis

output = agc(input, coeffs, output, state)

Description

This function implements multiple stages of the multiplicative adaptive gain used by Lyon's passive longwave cochlear model. The input is a number of channels from a filter bank. An array of state filters, one per channel and per stage, measure a running average of the energy in the channel. These state variables are then used to drive a single multiplicative gain per stage per channel.

The coeffs array is used to parameterize the AGC system. Two parameters must be supplied for each stage, a target output value and an epsilon. The AGC tries to keep the output below the target value. The gain is changed gradually based on the value of epsilon. Smaller values of epsilon allow the AGC process to take longer to adjust the output. Values of epsilon should be between 0 and 1. See the routine *EpsilonFromTauFS (17)* for more information.

The output and states arguments are optional. If present, and they are the right size, then these arrays are used instead of allocating new arrays. If the input has N samples then:

input is C x N
Coeffs is 2 x S (targets;epsilons)
output is C x N
state is C x S

If the output argument is not present then a new array is allocated and returned to MATLAB. If the state argument is not present then a new array is allocated and remembered for the next time this function is called. It will be reallocated if the number of filter channels change.

Note, the implementation of the *agc* function in this toolbox includes an additional limiting term to prevent the system gain from getting to close to zero. This is done (as described on page 19 of "Lyon's Cochlear Model" by not letting the state variable exceed 0.9.

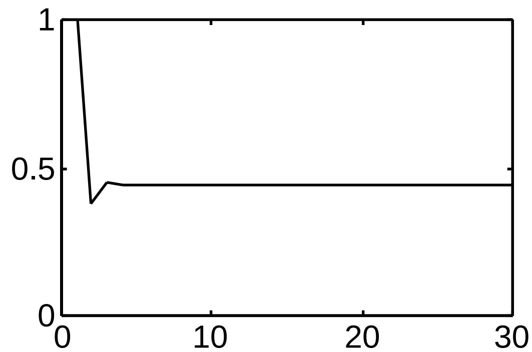
If the first argument, *input*, is the string 'clear' then all internal state is set to zero. It is important to clear the state between runs so that the data at the end of one input array doesn't affect the start of the next run.

Examples

```
»agc(ones(1,20),[.5;.5])

ans =

  Columns 1 through 7

    1.0000 0.1000 0.4500 0.2750 0.3625 0.3187 0.3406

  Columns 8 through 14

    0.3297 0.3352 0.3324 0.3338 0.3331 0.3334 0.3333

  Columns 15 through 20

    0.3334 0.3333 0.3333 0.3333 0.3333 0.3333
```

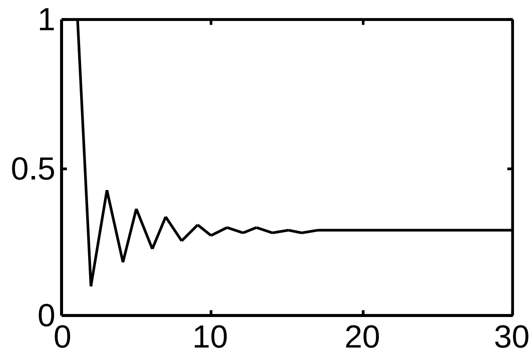
Here are some more examples. First use a target of 0.8 and a relatively short time constant (large value of epsilon or 0.5). Note the final output value will be dependent on both the input value and the AGC target. The oscillatory behaviour is normal when the input signal gets larger than the target faster than the AGC can cut the gain.

```
»agc('clear');plot(agc(ones(1,30),[.8;.5]))
```



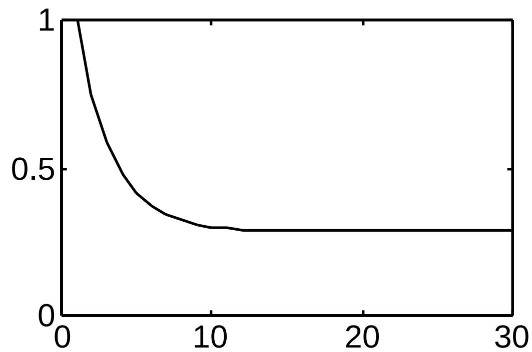
Now switch to a much smaller target value.

```
»agc('clear');plot(agc(ones(1,30),[.4;.5]))
```



Finally, we switch to a much longer time constant (smaller value of epsilon.) This makes the response much less likely to oscillate, but now the AGC takes longer to cut the signal level to the target.

```
»agc('clear');plot(agc(ones(1,30),[.4;.1]))
```



See Also

Malcolm Slaney, *Lyon's Cochlear Model*, Apple Computer Technical Report #13, 1988.

CorrelogramArray

Purpose

Compute a sequence of correlogram frames and store in one large array

Synopsis

movie = CorrelogramArray(input, sr, frameRate, width)

Description

This routine computes multiple frames of a correlogram, storing each frame as one row in a large array. The *input* data can be from any of the cochlear models in this toolbox.

The *input* array should be of size NxL where N is the number of cochlear channels, and each channel has L firing probabilities. The *input* is sampled at a frequency of *sr*.

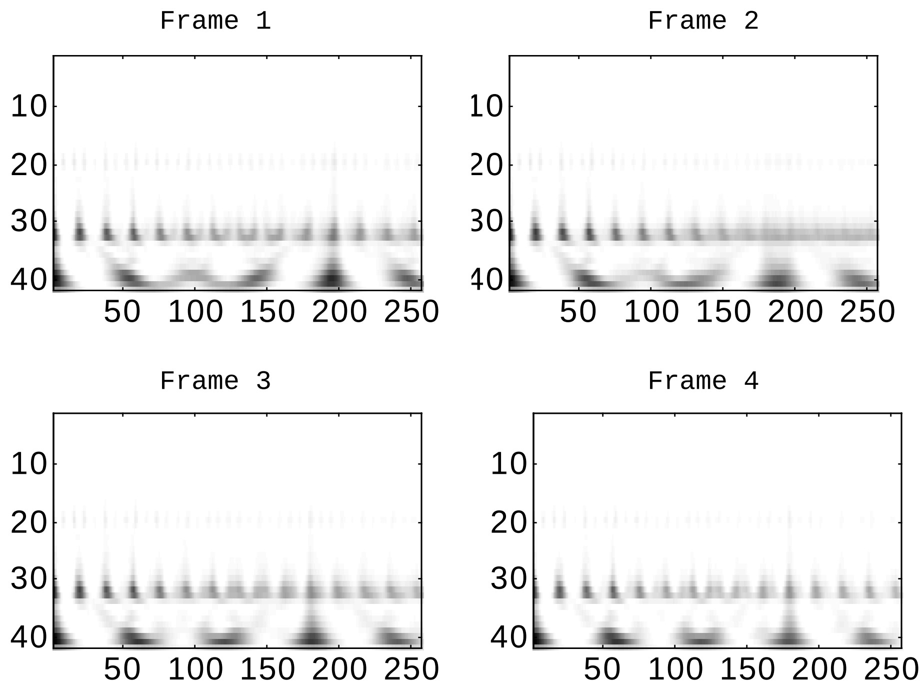
The resulting correlogram array will have one frame stored in each row. These frames will be computed *frameRate* times per second. Each image in the correlogram will be N x*width* in size. The rows in the output array will each have N**width* elements.

Examples

The correlogram of a vowel with vibrato can be calculated, played, and displayed using the following code.

```
»u=MakeVowel(4000,FMPoints(4000,120),16000,'u');
»playsound(u,16000)
»coch=LyonPassiveEar(u,16000,1,4,.25);
»width = 256;
»cor=CorrelogramArray(coch,16000,16,width);
Correlogram spacing is 1000 samples per frame.
CorrelogramFrame fftSize is 8192
CorrelogramFrame fftSize is 8192
CorrelogramFrame fftSize is 8192
CorrelogramFrame fftSize is 8192
»[pixels frames] = size(cor);
»colormap(1-gray);
»for j=1:frames
      corFrame = reshape(cor(:,j),pixels/width,width);
      scale = length(colormap)/max(max(corFrame));
      image(corFrame*scale);
      drawnow;
end
```

This produces the following images. Note how the pitch line moves.



See Also

CorrelogramFrame, CorrelogramMovie

Malcolm Slaney and R. F. Lyon, “On the importance of time—A temporal representation of sound,” in *Visual Representations of Speech Signals*, M. Cooke, S. Beete, and M. Crawford, eds., J. Wiley and Sons, Sussex, England, 1993.

CorrelogramFrame

Purpose

Compute one frame of a correlogram

Synopsis

picture = CorrelogramFrame(data, picWidth, start, winLen)

Description

This routine computes one frame of a correlogram. The *input* data is a two-dimensional array of cochlear data, each row representing firing probabilities from one cochlear channel. The output picture is a two dimensional array with one row for each row of cochlear input data and *picWidth* pixels wide.

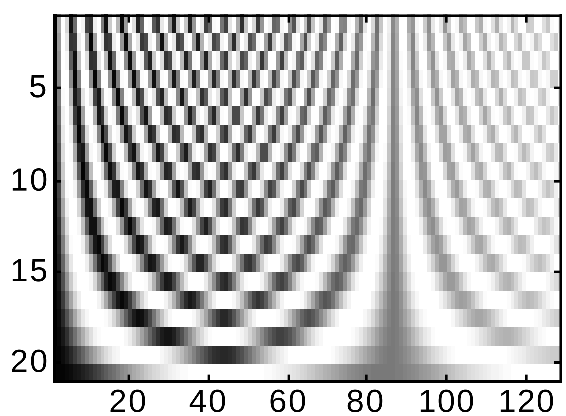
The correlogram is computed with autocorrelation using data from the input array. For each channel, the data from is extracted starting at column *start* and extending for *winLength* time steps.

Examples

A simple correlogram can be calculated from synthetic data using the following code. We use 20 sinusoids (with high frequencies at the top to simulate the cochlea).

```
for j=20:-1:1
      c(j,:) = max(0,sin((1:256)/256*(21-j)*3*2*pi));
end
picture=CorrelogramFrame(c,128,1,256);
image(picture/4*length(colormap))
```

Which produces the following image:



See Also

This routine is used by the CorrelogramArray and CorrelogramMovie routines.

Malcolm Slaney and R. F. Lyon, "On the importance of time—A temporal representation of sound," in *Visual Representations of Speech Signals*, M. Cooke, S. Beete, and M. Crawford, eds., J. Wiley and Sons, Sussex, England, 1993.

CorrelogramMovie

Purpose

Compute a correlogram movie

Synopsis

movie = CorrelogramMovie(data, sr, frameRate, width)

Description

This routine computes multiple frames of a correlogram, storing each image as one frame in a MATLAB movie object. The *input* data can be from any of the cochlear models in this toolbox.

The *input* array should be of size NxL where N is the number of cochlear channels, and each channel has L firing probabilities. The *input* is sampled at a frequency of *sr*.

The resulting correlogram array will have one frame stored in each row. These frames will be computed *frameRate* times per second. Each image in the correlogram will be N x*width* in size. The rows in the output array will each have N**width* elements. Use the MATLAB movie command to play the resulting movie on the screen.

Examples

A correlogram movie can be calculated using the following code. See the *CorrelogramArray (8)* documentation for images of the movie frames.

```
»u=MakeVowel(4000,FMPoints(4000,120),16000,'u');
»playsound(u,16000)
»coch=LyonPassiveEar(u,16000,1,4,.25);
»mov=CorrelogramMovie(coch,16000,16,256);
»movie(mov,-10,16)
```

See Also

CorrelogramFrame, CorrelogramArray

Malcolm Slaney and R. F. Lyon, “On the importance of time—A temporal representation of sound,” in *Visual Representations of Speech Signals*, M. Cooke, S. Beete, and M. Crawford, eds., J. Wiley and Sons, Sussex, England, 1993.

CorrelogramPitch

Purpose

Compute the pitch of a sound using the correlogram

Synopsis

```
[pitch salience]=CorrelogramPitch(correlogram, ...
                    width, sr [, lowPitch, highPitch]);
```

Description

This routine calculates the pitch of a sound using a correlogram model of human pitch perception. Given a correlogram, as computed by *CorrelogramArray*, this routine performs the following operations on each frame of the correlogram.

1) Reshape the data in one row of the correlogram array into a correlogram image. The second function argument, *width*, is needed to reshape the correlogram to the proper size.
2) Summarize the correlogram frame by adding the energy at each time lag across all channels.
3) Remove the peak in the summary correlogram at zero lag by removing all points in the summary correlogram up to the first positive inflection.
4) Remove all points from the summary correlogram outside the valid pitch range.
5) Find the position of the largest peak that remains. This is an estimate of the pitch. The sample rate (sr) argument is used to convert from autocorrelation lag index to pitch frequency.
6) Pitch salience is calculated by dividing the value of the summary correlogram at the pitch peak into the value of the summary correlogram at zero lag. Highly periodic sounds with an easily perceivable pitch will have a salience close to 1, aperiodic sounds will have a salience closer to zero.

The result is a single pitch estimate and a crude estimate of pitch salience at every frame.

Note, this routine uses a very simple, but powerful, algorithm to model human pitch perception. The paper by Slaney and Lyon describes additional algorithmic enhancements that allow more robust pitch estimates.

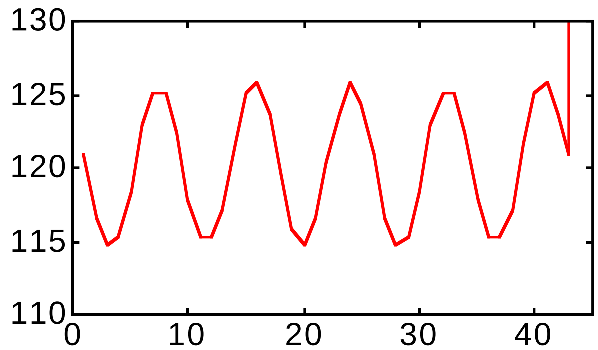
The *CorrelogramPitch* function includes optional *lowPitch* and *highPitch* arguments that can be used to limit the range of legal pitch values.It is important to note that neither *CorrelogramPitch* or the papers referenced below include any other higher-level knowledge about pitch. Notably, this work does not enforce any frame-to-frame continuity in the pitch. Each pitch estimate is independent and there is no restriction preventing the estimate to change instantaneously from frame to frame.

Examples

The simplest possible pitch detector is computed using auto-correlation of the original waveform. This can be done by computing the correlogram of the original waveform (pretending that it is the output of a cochlear model with just one channel). The input is a vowel with it’s pitch centered at 120Hz and a 5Hz vibrato.

```
»u=MakeVowel(20000,FMPoints(20000,120),22254,'u');
»cor=CorrelogramArray(u,22254,50,256);
»p=CorrelogramPitch(cor,256,22254);
»plot(p)
»axis([0 45 110 130])
```

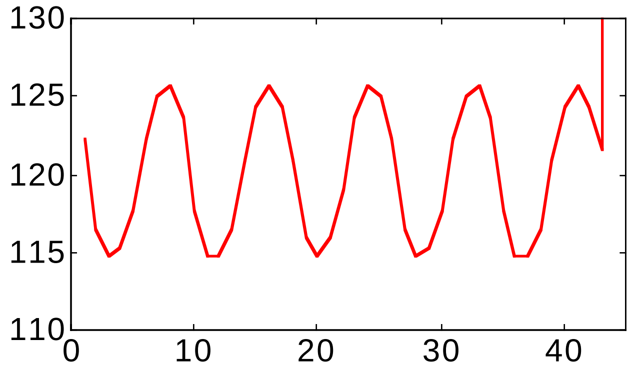
The resulting pitch is shown below. The final pitch value is an artifact of the computation at the end of the signal.



A more robust result is possible if the correlogram is computed on the output of a cochlear model. The example below computes the cochlear output and then the correlogram of the /u/ vowel.

```
»coch=LyonPassiveEar(u,22254,1,4,.5);
»cor=CorrelogramArray(coch,22254,50,256);
»p=CorrelogramPitch(cor,256,22254);
»plot(p)
»axis([0 45110 130])
```

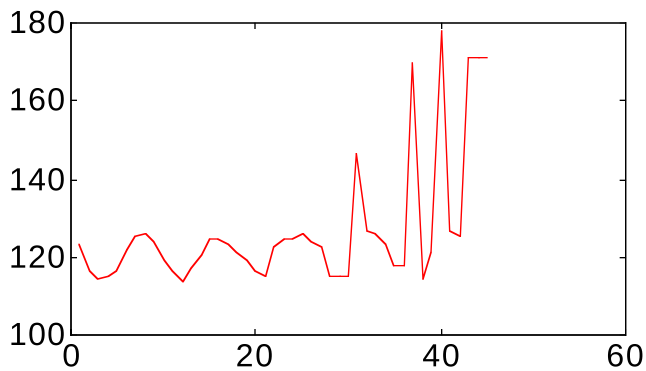
The resulting pitch estimate is



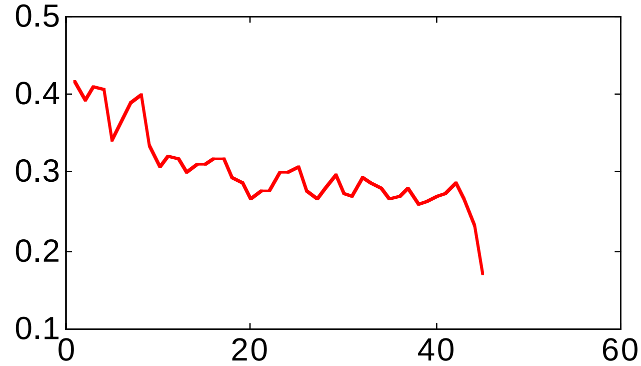
The *lowPitch* and *highPitch* arguments can be used to limit the pitch estimates to a known range. This is a simple way to make the pitch estimate use high-order bodies of knowledge. The example below shows adding steadily increasing gaussian-white noise to the /u/ vowel and then estimating the pitch.

```
»u=MakeVowel(20000,FMPoints(20000,120),22254,'u');
»n=randn([1 20000]).*(1:20000)/20000;
»un=u+n/4;
»coch=LyonPassiveEar(un,22254,1,4,.5);
»cor=CorrelogramArray(coch,22254,50,256);
»[pitch sal]=CorrelogramPitch(cor,256,22254,100,200);
»plot(pitch)
```

The pitch is limited to values between 100 and 200Hz and the resulting estimate is



The salience of this pitch estimate declines as the signal to noise ratio goes down.



See Also

The basic idea behind this model of pitch is described in three papers. The first paper extends the functionality implemented by this function using several techniques to more reliably pick the most robust peak.

> Malcolm Slaney and Richard F. Lyon, “A perceptual pitch detector,” in the *Proceedings of the 1990 International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, NM, IEEE, pp 357-360, 1990.

An extensive comparison of the performance of a correlogram model of pitch and human performance is described in:

> Ray Meddis and Michael J. Hewitt, “Virtual pitch and phase sensitivity of a computer model of the auditory periphery, I. Pitch identification,” *J. Acoustical Society of America*, 89 (6), pp. 2866-2682, 1991.

> Ray Meddis and Michael J. Hewitt, “Virtual pitch and phase sensitivity of a computer model of the auditory periphery, II. Phase sensitivity,” *J. Acoustical Society of America*, 89 (6), pp. 2683-2894, 1991.

DesignLyonFilters

Purpose

Design the filters needed to implement Lyon's passive cochlear model

Synopsis

[filters, freqs] = DesignLyonFilters(fs,EarQ,StepFactor)

Description

Design the cascade of second order filters and the front filters (outer/middle and compensator) needed for Lyon's Passive Short Wave (Second Order Sections) cochlear model. The variables used here come from Apple ATG Technical Report #13 titled *Lyon's Cochlear Model*.

Most of the parameters are hardwired into this m-function. The user settable parameters are the digital sampling rate (fs), the basic Q of the each stage (usually 8 or 4), and the *StepFactor* between channels (usually .25 and .125, respectively.) The *EarQ* and *Stepfactor* parameters default to 8 and 32/EarQ respectively.

The result is returned as rows of second order filters; three coefficients for the numerator and two for the denominator. Using the same convention as the Matlab filter function, the coefficients are [B0 B1 B2 A1 A2].

This function is called automatically by LyonPassiveEar to design the appropriate filterbank.

Examples

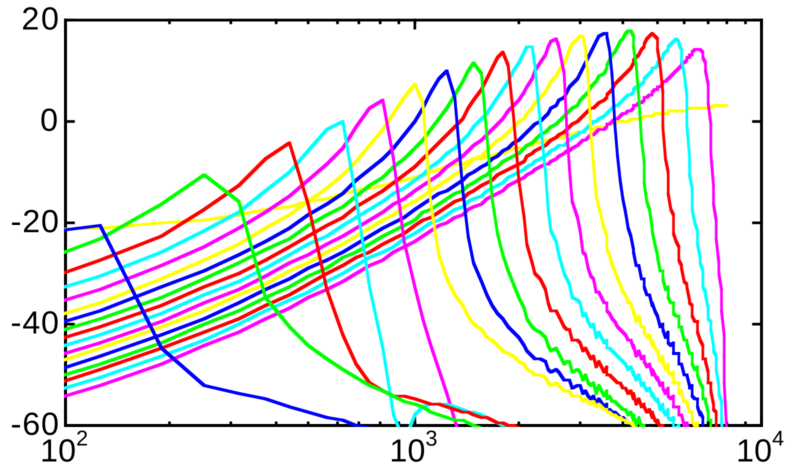
The first five channels of a typical filter bank design are shown below. Note, the first two channels implement the model's outer and middle ear filters.

```
»filts=DesignLyonFilters(16000);
»size(filts)

ans =

    88     5

»filts(1:5,:)

ans =

         0    0.7474   -0.6644         0         0
    0.8373         0   -0.8373    1.6433    0.6772
    0.8899    1.7137    0.8251    1.6369    0.6854
    0.8877    1.7027    0.8250    1.6160    0.6934
    0.8859    1.6774    0.8252    1.5821    0.7011
```

The frequency response of this filter bank can be calculated using soscascade.

```
»resp=soscascade([1 zeros(1,255)],filts);
»freqResp=20*log10(abs(fft(resp(1:5:88,:)')));
»freqScale=(0:255)/256*16000;
»semilogx(freqScale(1:128),freqResp(1:128,:))
»axis([100 10000 -60 20])
```

The resulting response for every fifth channel is shown below.



See Also

Malcolm Slaney, *Lyon's Cochlear Model*, Apple Computer Technical Report #13, 1988.

EpsilonFromTauFs

Purpose

Calculate first order decay coefficient (tau)

Synopsis

epsilon = EpsilonFromTauFS(tau, fs)

Description

Find first order filter coefficient as a function of time constant and sample rate

Examples

The following example shows the design of a first order filter with a sampling interval of 1 and a time constant of 5 (samples). A simple digital filter is formed by adding at each time the current input value and (1-*epsilon*) times the last output value. The resulting impulse response decays exponentially, reaching 37% of it's original value in one time constant, *tau*.

You can verify the relationship between the time constant and the impulse response using the following example code.

```
»eps=EpsilonFromTauFS(5,1)

eps =

    0.1813

»filter(1, [1 eps-1],[1 zeros(1,9)])

ans =

  Columns 1 through 7

    1.0000 0.8187 0.6703 0.5488 0.4493 0.3679 0.3012

  Columns 8 through 10

    0.2466 0.2019 0.1653

»sosfilters([1 zeros(1,9)],[1 0 0 eps-1 0])

ans =

  Columns 1 through 7

   1.0000 0.8187  0.6703  0.5488  0.4493  0.3679 0.3012

  Columns 8 through 10

   0.2466 0.2019  0.1653
```

FilterBank

Purpose

Compute an array of parallel filters

Synopsis

y=FilterBank(forward,feedback,x)

Description

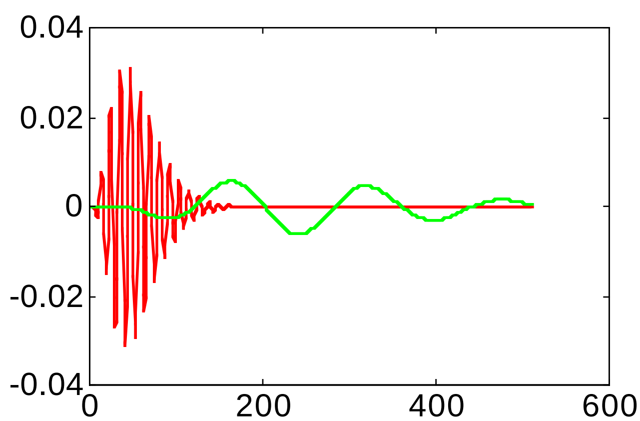
This function filters the waveform *x* with the array of filters specified by the forward and feedback parameters. Each row of the *forward* and *feedback* parameters are the parameters to the Matlab built-in function filter. The output is an array of filter outputs, one waveform per row of the output array.

Examples

The impulse response of two filters can be calculated using the following code. These filters are designed by the MakeERBFilters function.

```
»[frwd,fdbk]=MakeERBFilters(16000,2,100);
»y=FilterBank(frwd,fdbk,[1 zeros(1,511)]);
»plot(y')
```

This generates the following plot of the two channel's impulse responses.



FMPoints

Purpose

Compute glottal pulses for voice with vibrato

Synopsis

points=FMPoints(len, freq, fmFreq, fmAmp, fs)

Description

This routine generates (fractional) sample locations for frequency-modulated impulses.

len	number of samples
freq	pitch frequency (Hz)
fmFreq	vibrato frequency (Hz) (defaults to 6 Hz)
fmAmp	max change in pitch (defaults to 5% of freq)
fs	sample frequency (defaults to 22254.545454 samples/s)

The basic formula for the phase angle is

$$\theta = 2\pi \cdot \mathrm{freq} \cdot t + \mathrm{fmAmp}/\mathrm{fmFreq} \cdot \sin(2\pi \mathrm{fmFreq} \cdot t)$$

The output of this routine is a list of glottal pulse locations that can be used by the *MakeVowel* routine.

Examples

The *MakeVowel* routine is used to synthesize simple vowels. More realistic sounding vowels are possible with a bit of vibrato added to them with the *FMPoints* routine.

```
»u=MakeVowel(20000,FMPoints(20000,120),22254,'u');
»playsound(u/max(u))
```

Acknowledgments

This routine was written by Richard O. Duda of San Jose State University.

FreqResp

Purpose

Evaluate frequency response of a filter

Synopsis

mag = FreqResp(filter, f, fs)

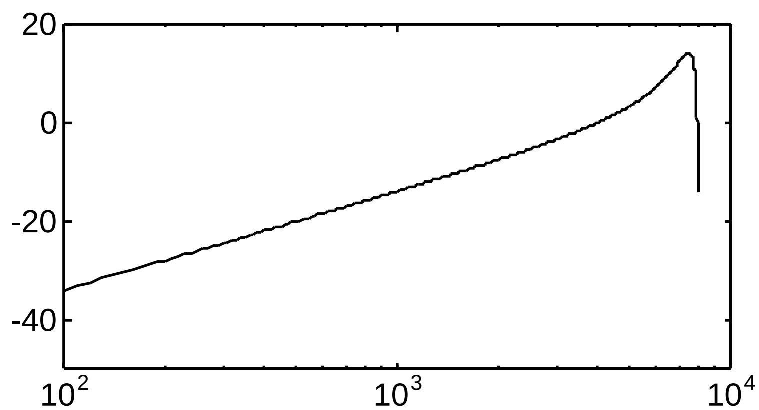
Description

Find the frequency response (in dB) of a filter (1x5 vector) at frequency f assuming a sampling rate fs. A vector of frequencies can be used as input.

Examples

The *FreqResp* function can be used to evaluate the second filter in the default filter bank shown in the *DesignSosFilters* example.

```
»f=10:10:7990;
»resp=FreqResp([0.8373 0 -0.8373 1.6433 .6772], ...
                  f, 16000);
»semilogx(f,resp);
```



LyonPassiveEar

Purpose

Calculate auditory nerve responses using Lyon's passive cochlear model

Synopsis

y=LyonPassiveEar(x,sr,df,earQ,stepfactor,diff,agcf,tau)

Description

This mex-function calculates the probability of firing along the auditory nerve due to a sound input, *x*, with a sample rate, *sr*. The rest of the arguments are optional parameters of the cochlear model implementation and are described below.

df(1)	Decimation Factor - How much to decimate the model's output. Normally the cochlear model produces one output per channel at each sample time. This parameters allows the output to be decimated in time (using a filter to reduce aliasing. See the tau parameters.)
earQ(8)	Quality Factor - The quality factor of a filter is a measure of its bandwidth. In this case it measures the ratio of the width of each bandpass filter at a point 3dB down from the maximum. Normally, critical band filters have a Q of about 8. Smaller values of earQ mean broader cochlear filters.
stepfactor	Filter stepping factor - Each filter in a filter bank is overlapped by a fixed fraction given by this parameter. The default value is given by earQ/32. Thus normally filters (q=8) are overlapped by 25%.
differ(1)	Channel Difference Flag - Adjacent filter channels can be subtracted to further improve the model's frequency response. This parameter is a flag; non-zero values indicate the channel differences should be computed.
agcf(1)	Automatic Gain Control Flag - An automatic gain control is used to model neural adaptation. This flag turns the adaptation mechanism on and off.
taufactor(3)	Filter Decimation Tau Factor - When the output of the cochlear model is decimated, a low pass filter is applied to each channel to reduce the high frequency content and minimize aliasing. The filter's time constant (tau) is set to the decimation factor multiplied by this argument. Larger values for the taufactor mean less high frequency information is passed.

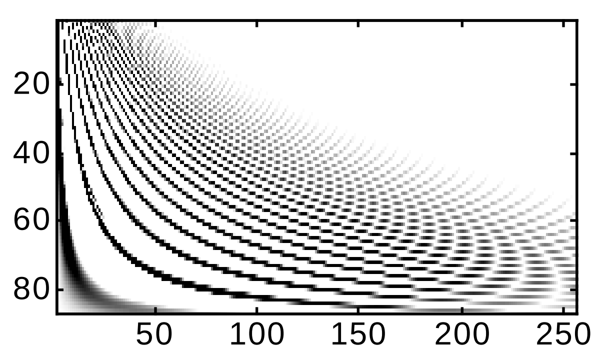
Note this function resets the filter state each time it is run. The default state of the AGC filters is zero, so the cochlear model is very sensitive to initial sounds.

Examples

Calculate the impulse response with

```
»is=LyonPassiveEar([1 zeros(1,255)],16000,1);
»image(min(is/.0004*length(colormap),64))
```

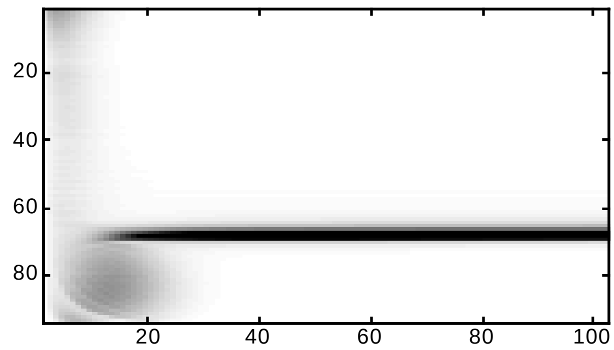
The response looks like



A sine wave (1kHz) is generated and run through the standard cochlear model using the following code.

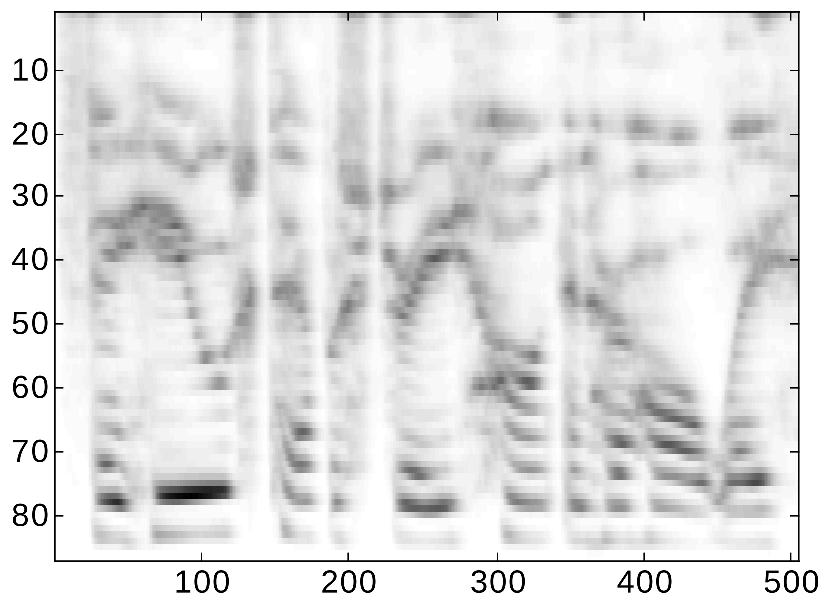
```
»s=sin((0:2041)/20000*2*pi*1000);
»ys=LyonPassiveEar(s,20000,20);
»image(ys/max(max(ys))*length(colormap));
```

The resulting image looks like this:



Finally, a cochleagram of the ‘A huge tapestry hung in her hallway’ utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC) is shown below. It was computed using the following command line.

```
»coch=LyonPassiveEar(tap,16000,100);
»image(coch/max(max(coch))*length(colormap));
```



See Also

“Lyon’s Cochlear Model”, by Malcolm Slaney and published as Apple Technical Report #13 (1988) describes the implementation of this particular cochlear model.

DesignLyonFilters, soscascade, agc

MakeERBFilters

Purpose

Design the filters needed to implement an ERB cochlear model.

Synopsis

[frwd, fdbk]=MakeERBFilters(fs,numChannels,lowFreq)

Description

The *MakeERBFilters* function computes the filter coefficients for a bank of Gammatone filters. These filters were defined by Patterson and Holdsworth for simulating the cochlea. The results are returned as arrays of filter coefficients. Each row of the filter arrays (forward and feedback) can be passed to the MatLab filter function, or you can do all the filtering at once with the *FilterBank*() function. Each filter in the filter bank is a fifth four order IIR filter with both poles and zeros.

The filter bank contains *numChannels* channels that extend from half the sampling rate (*fs*) to *lowFreq*.

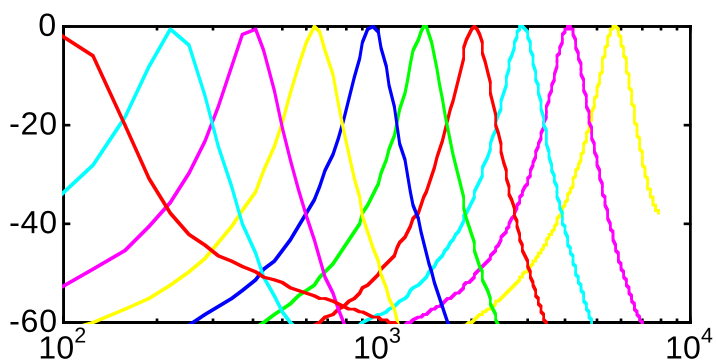
Examples

The ten ERB filters between 100 and 8000Hz are computed using

```
»[frwd,fdbk]=MakeERBFilters(16000,10,100);
```

The resulting frequency response is given by

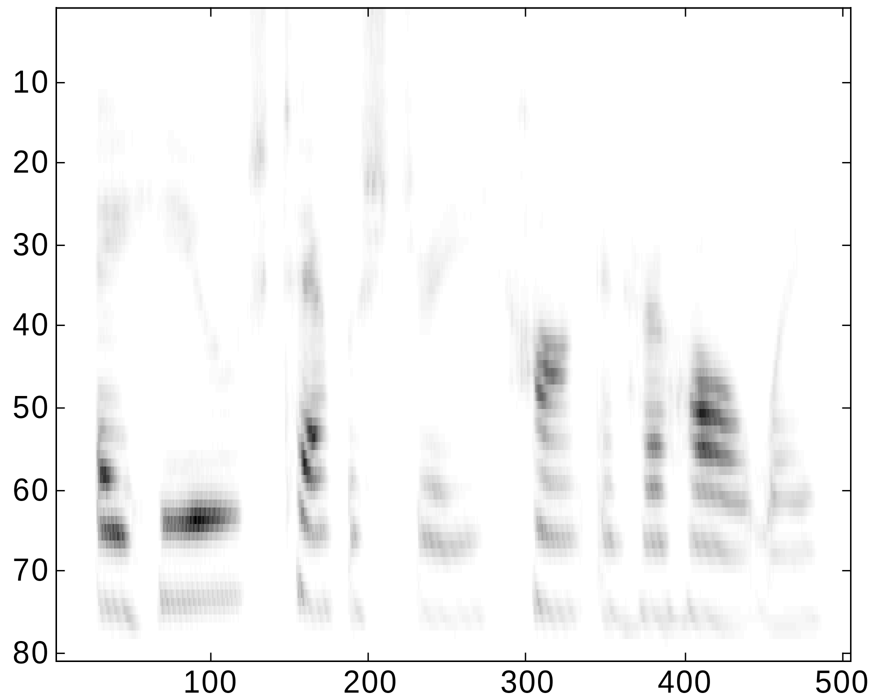
```
»y=FilterBank(frwd,fdbk,[1 zeros(1,511)]);
»resp=20*log10(abs(fft(y')));
»freqScale=(0:511)/512*16000;
»semilogx(freqScale(1:255),resp(1:255,:))
»axis([100 10000 -60 0])
```



A simple cochlear model can be formed by filtering an utterance with these filters. To convert this data into an image we pass each row of the cochleagram through a half-wave-rectifier, a low-pass filter, and then decimate by a factor of 100. A cochleagram of the ‘A huge tapestry hung in her hallway’ utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC) is shown below. It was computed using the following commands.

```
»[frwd,fdbk]=MakeERBFilters(16000,40,100);
»coch=FilterBank(frwd,fdbk,tap);
»for j=1:20
      c=max(coch(j,:),0);
      c=filter([1],[1 -.99],c);
      coch(j,:)=c;
end
»image(c/max(max(c))*length(colormap));
```

Unlike the other cochlear models in this report, there is no adaptation or automatic gain control to equalize the formant frequencies and enhance the onsets. The *MeddisHairCell (26)* function can be used to do this.



See Also

Malcolm Slaney, *An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank,* Apple Computer Technical Report #35, 1993.

MakeVowel

Purpose

Simple vowel synthesis

Synopsis

y = MakeVowel(len, pitch, sampleRate, f1, f2, f3)

Description

Make a vowel with *length* samples and the given *pitch*. The vowel's sample rate is given by *sampleRate*. The formant frequencies are *f1*, *f2* & *f3*. The formant frequencies for these English vowels are given by:

Vowel	*f1*	*f2*	*f3*
/a/	730	1090	2440
/i/	270	2290	3010
/u/	300	870	2240

The *pitch* variable can either be a scalar indicating the actual pitch frequency, or an array of impulse locations. Using an array of impulses allows this routine to compute vowels with varying pitch.

Alternatively, *f1* can be replaced with one of the following strings 'a', 'i', 'u' and the appropriate formant frequencies are automatically selected.

Examples

A sequence of three vowels can be computed using

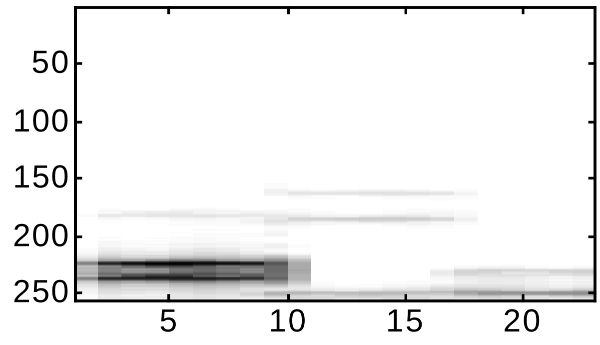
```
»vowels=[MakeVowel(10000,100,16000,'a') ...
            MakeVowel(10000,100,16000,'i') ...
            MakeVowel(10000,100,16000,'u')];
```

and then played with

```
»PlaySound(vowels/max(vowels),16000);
```

Similarly, a shorter sequence of vowels can be displayed as a spectrogram.

```
»vowels=[MakeVowel(1000,100,16000,'a')...
            MakeVowel(1000,100,16000,'i') ...
            MakeVowel(1000,100,16000,'u')];
»s=spectrogram(vowels,256,2,2);
»image(s/256*length(colormap))
```



Acknowledgments

The first version of this routine was written by Richard O. Duda (San Jose State University). Additional debugging was provided by Professor. Duda.

MeddisHairCell

Purpose

Implement Meddis' Inner Hair Cell Model

Synopsis

y = MeddisHairCell(data,sampleRate[,subtractSpont])

Description

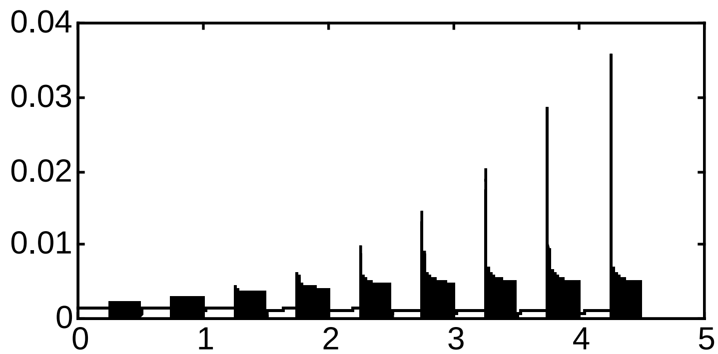
This function calculates Ray Meddis' hair cell model for a number of channels. Data is arrayed as one channel per row. All channels are done in parallel (but each time step is sequential) so it is much more efficient to process lots of channels at once.

The *subtractSpont* argument is optional. If this argument is positive then the hair cell's spontaneous rate is subtracted before the result is returned.

Examples

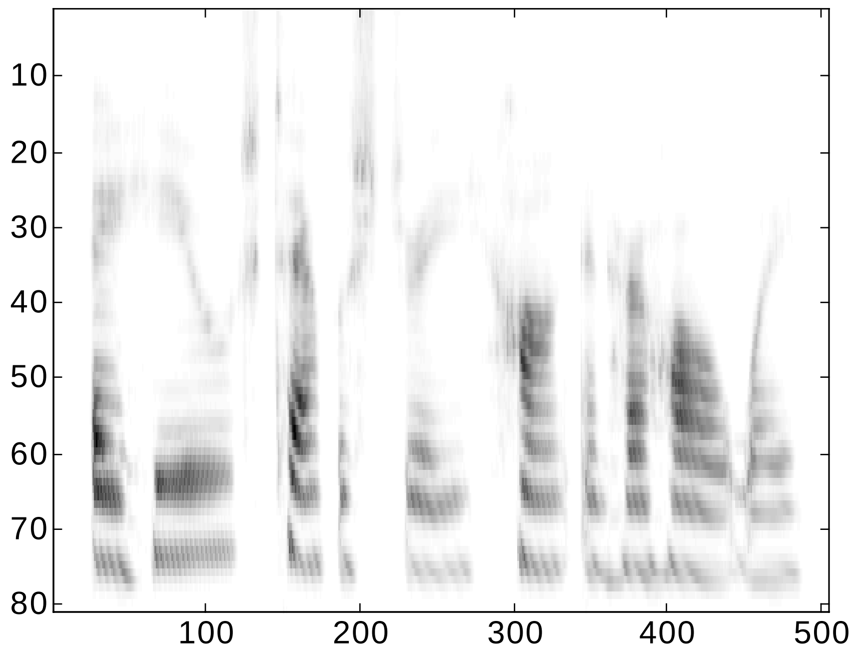
This MEX function can be checked by comparing the results to those published in Ray Meddis' 1986 JASA paper. The first two statements generate a sequence of tone pips, each 250 ms long, ranging in amplitude from 40dB to 80 dB in 5dB steps. Note the amplitude scale is arbitrary. In this case it was chosen to agree with the examples shown in his 1990 paper.

```
tone=sin((0:4999)/20000*2*pi*1000);
s=[zeros(1,5000) ...
 tone*10^(40/20-1.35) zeros(1,5000) ...
 tone*10^(45/20-1.35) zeros(1,5000) ...
 tone*10^(50/20-1.35) zeros(1,5000) ...
 tone*10^(55/20-1.35) zeros(1,5000) ...
 tone*10^(60/20-1.35) zeros(1,5000) ...
 tone*10^(65/20-1.35) zeros(1,5000) ...
 tone*10^(70/20-1.35) zeros(1,5000) ...
 tone*10^(75/20-1.35) zeros(1,5000) ...
 tone*10^(80/20-1.35)];
y=MeddisHairCell(s,20000);
plot((1:90000)/20000,y(1:90000))
```



This hair cell model can be added to the back-end of a Gammatone filter bank to form a "complete" auditory model. A cochleagram of the 'A huge tapestry hung in her hallway' utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC) is shown below. The output of the filter-bank is scaled by a factor of 80 to put this particular result in the correct range for the hair cell model. The filtering operation in the loop and the decimation are performed to reduce the amount of data to display. The cochleagram was computed using the following commands.

```
»[frwd,fdbk]=MakeERBFilters(16000,80,100);
»coch=ERBFilterBank(frwd,fdbk,tap);
»hc=MeddisHairCell(coch/max(max(coch))*80,16000,1);
»for j=1:80
      c=hc(j,:);
      c=filter([1],[1, -.99],c);
      h(j,:)=c(1:100:50381);
 end
»image(h/max(max(h))*length(colormap));
```



See Also

R. Meddis, “Simulation of mechanical to neural transduction in the auditory recepter,” *Journal of the Acoustical Society of America*, vol.79, no.3, p. 702-711, March 1986.

M. J. Hewitt, R. Meddis, “Implementation details of a computation model of the inner hair-cell/auditory-nerve synapse,” *Journal of the Acoustical Society of America*, vol.87, no.4, p. 1813-1816, April 1990.

mfcc

Purpose

Mel-frequency cepstral coefficient transform of an audio signal

Synopsis

```
[ceps,freqresp,fb,recon] = mfcc(input, samplingRate)
```

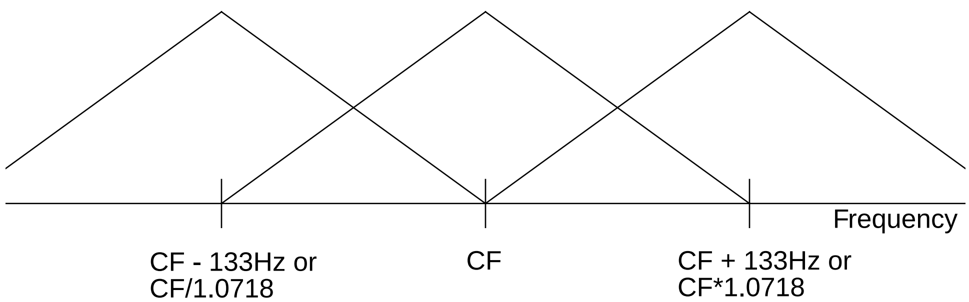
Description

Find the cepstral coefficients (*ceps*) corresponding to the input. Three other quantities are optionally returned that represent the detailed FFT magnitude (*freqresp*), the $\log_{10}$ mel-scale filter bank output (*fb*), and the reconstruction of the filter bank output by inverting the cosine transform.

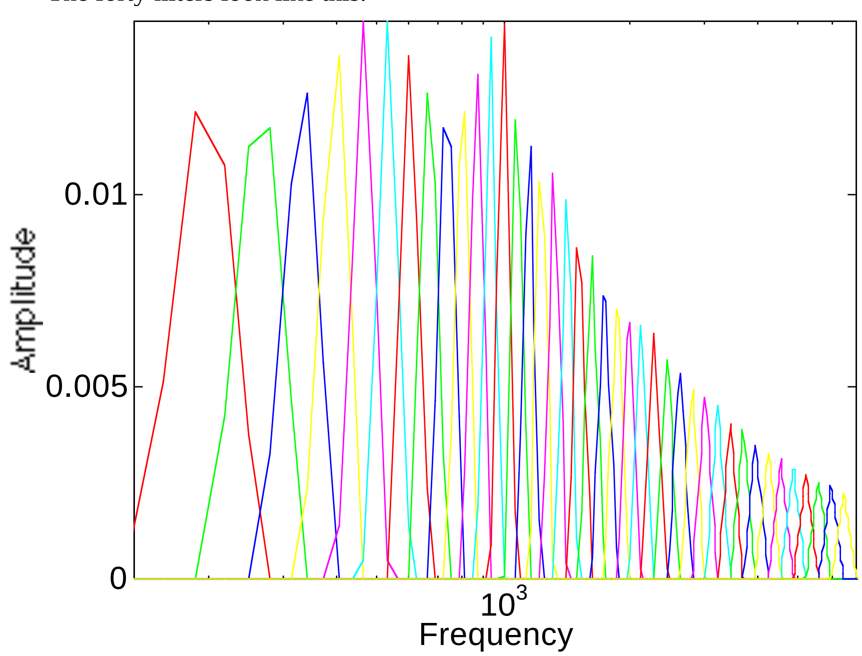
The sequence of processing includes for each chunk of data:

Window the data with a hamming window,
Shift it into FFT order,
Find the magnitude of the FFT,
Convert the FFT data into filter bank outputs,
Find the log base 10,
Find the cosine transform to reduce dimensionality.

The filter bank is constructed using 13 linearly-spaced filters (133.33Hz between center frequencies,) followed by 27 log-spaced filters (separated by a factor of 1.0711703 in frequency.) Each filter is constructed by combining the amplitude of FFT bin as shown in the figure below.

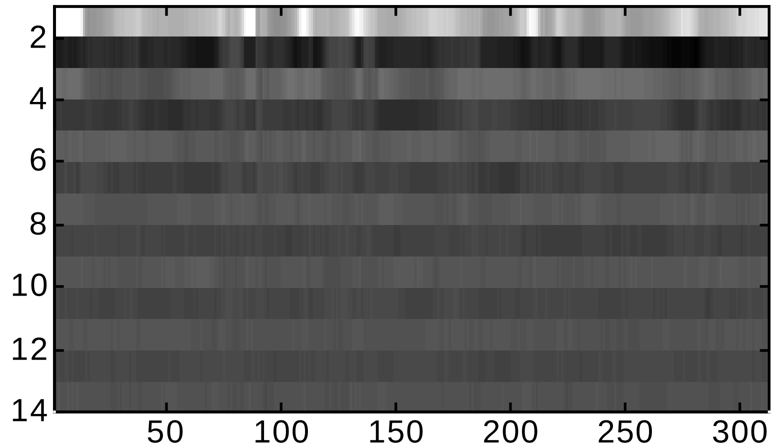


The forty filters look like this.

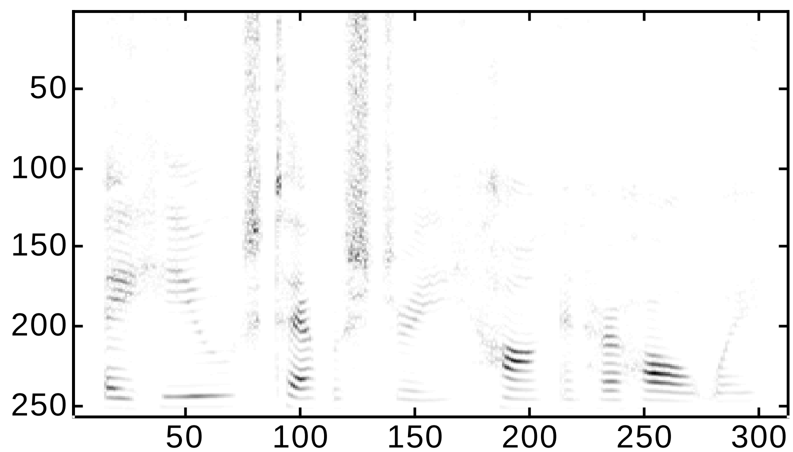


Examples

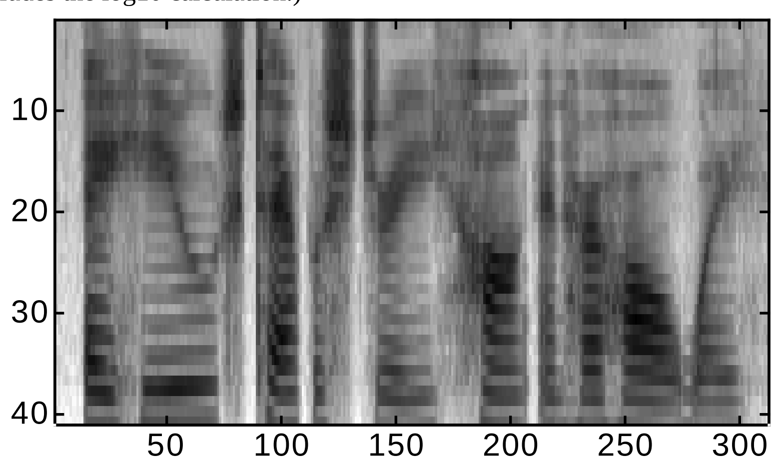
Here is the result of calculating the cepstral coefficients of the ‘A huge tapestry hung in her hallway’ utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC). The utterance is 50189 samples long at 16kHz, and all pictures are sampled at 100Hz and there are 312 frames. Note, the top row of the mfcc-cepstrum, ceps(1,:), is known as C_0 and is a function of the power in the signal.Since the waveform in our work is normalized to be between -1 and 1, the C_0 coefficients are all negative. The other coefficients, C_1-C_{12}, are generally zero-mean.



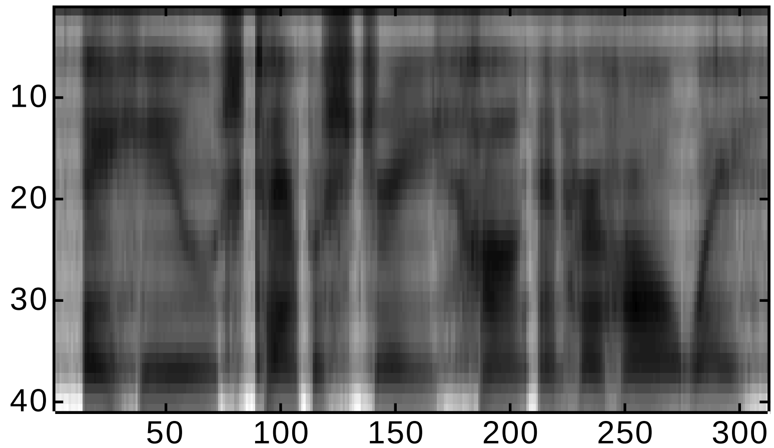
Several intermediate results are also generated that can be used to investigate the performance of the algorithm. The uncompressed FFT spectrogram (freqresp) is shown below (it’s been flipped so that high frequencies are at the top.)



After combining several FFT channels into a single mel-scale channel, the result is the filter bank output. This is shown below (the *fb* output of the mfcc command includes the log10 calculation.)



Finally, the conversion into the cepstral domain uses the discrete cosine transform to reduce the dimensionality of the output. We can invert the cosine transform to get back into the filter bank domain and see how much information we have lost. The *recon* output is shown below (flipped again so that the high frequency channel is at the top). Note the result is much smoother then the original filter bank output.



See Also

An MFCC-like algorithm was proposed by M. J. Hunt, M. Lennig, and P. Mermelstein, “Experiments in syllable-based recognition of continuous speech,” *Proceedings of the 1980 ICASSP*, Denver, CO, pp. 880-883, 1980.

SecondOrderFilter

Purpose

Design a second order filter section

Synopsis

filter = SecondOrderFilter(f, q, fs)

Description

Design a second order digital filter with a center frequency of f, filter quality of q, and digital sampling rate of fs (Hz).

The filter is a biquadratic section with a transfer function equal to

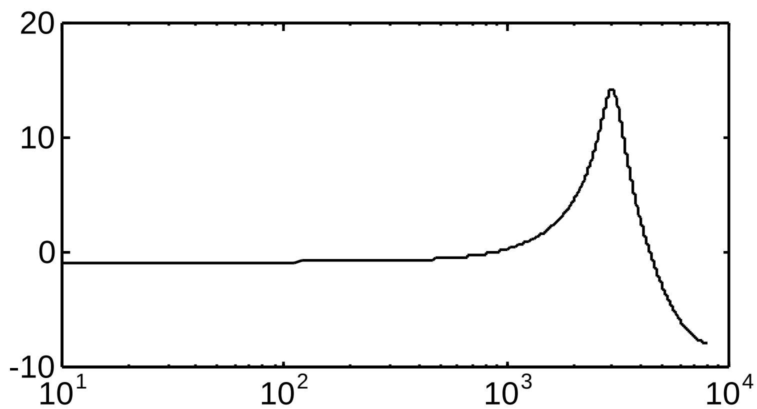
$$\frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + A_1 z^{-1} + A_2 z^{-2}} .$$

The filter is described by a five element row vector with the filter's coefficients equal to [B0 B1 B2 A1 A2].

Examples

A simple bandpass filter is formed by putting the filter's poles near the desired center frequency. This is shown below.

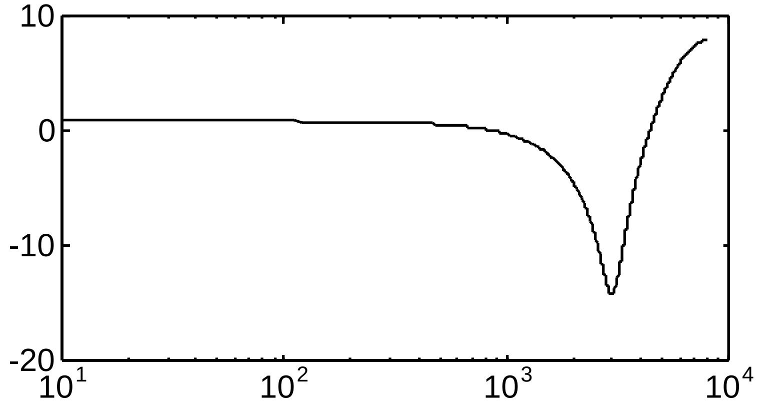
```
»f=10:10:7990;
»sos=SecondOrderFilter(3000,5,16000)

sos =

    1.0000   -0.6900    0.7901

»filt=[1 0 0 sos(2:3)]

filt =

    1.0000         0         0   -0.6900    0.7901

»semilogx(f,FreqResp(filt,f,16000))
```



Likewise, a simple band-reject filter is formed by using a pair of zeros. The resulting filter looks like this.

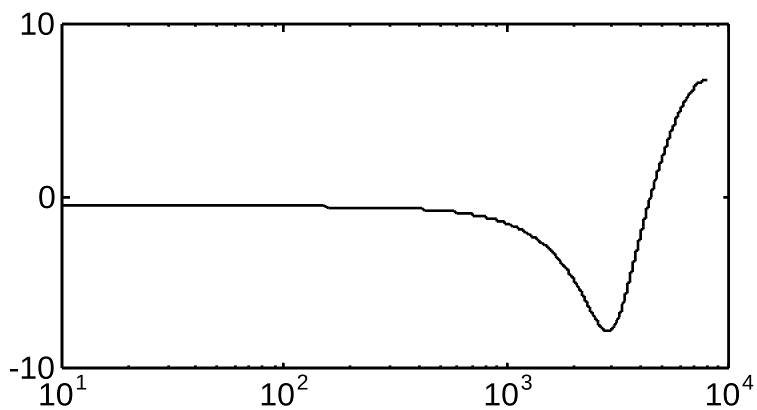
```
»filt=[sos 0 0]

filt =

    1.0000   -0.6900    0.7901         0         0
```

```
»semilogx(f,FreqResp(filt,f,16000))
```



A broader filter is formed by using a lower q, in this case 2.

```
»sos=SecondOrderFilter(3000,2,16000)

sos =

    1.0000   -0.6212    0.5549

»filt=[sos 0 0]

filt =

    1.0000   -0.6212    0.5549         0         0

»semilogx(f,FreqResp(filt,f,16000))
```



SeneffEar

Purpose

Implement the Stages I and II of Seneff's Auditory Model

Synopsis

y = SeneffEar(x, fs [, plotChannel])

Description

This function implements Stage I (Critical Band Filter Bank) and Stage II (Hair Cell Synapse Model) of Seneff's Auditory model. This routine converts an input signal, x, into an array of

> "detailed waveshapes of the probabilistic response to individual cycles of the input stimulus."

This model is

> "based on properties of the human auditory system. A bank of critical-band filters defines the initial spectral analysis. Filter outputs are processed by a model of the nonlinear transduction stages in the cochlea, which accounts for such features as saturation, adaptation and forward masking. The parameters of the model were adjusted to match existing experimental results of the physiology of the auditory periphery."

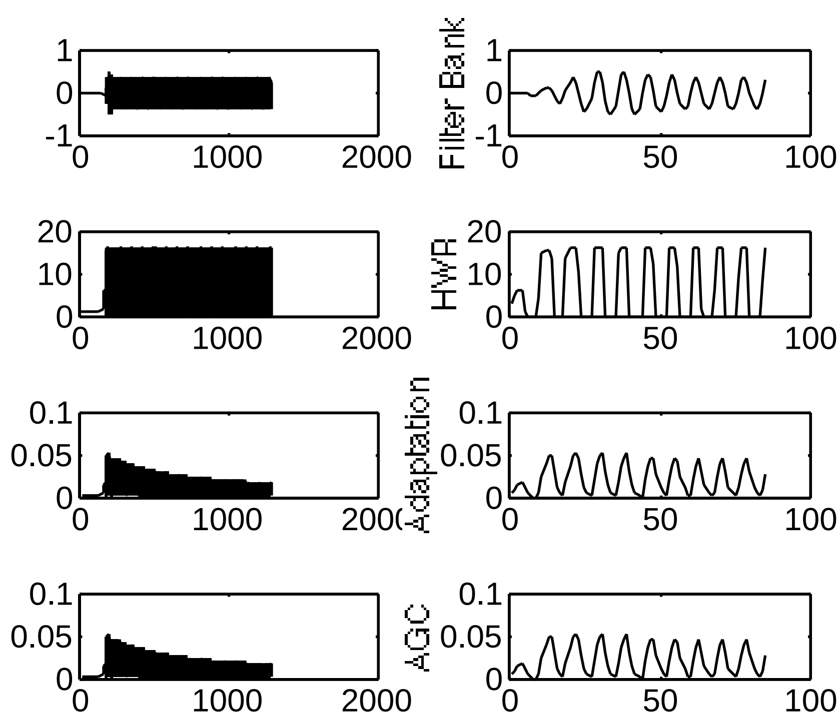
The input data (*x*) is a one-dimensional array with a sampling rate of *fs*. The optional parameter *plotChannel* is used to indicate a channel to plot for debugging (see example below.)

Examples

Figure 3 of Seneff's paper (1988) can be duplicated with the following statements

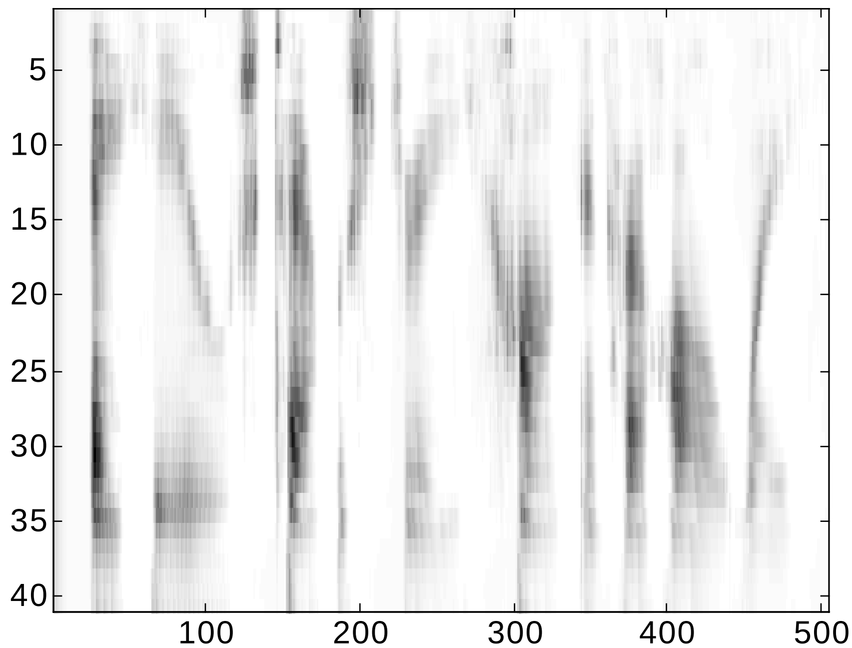
```
»s=[zeros(1,160) sin(2000*2*pi/16000*(1:1120))];
»y=SeneffEar(s,16000,15);
```

which produces the following plot.



A cochleagram using Seneff's ear model of the 'A huge tapestry hung in her hallway' utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC) is shown below. The filtering operation in the loop and the decimation are performed to reduce the amount of data to display. The cochleagram was computed using the following commands.

```
   »hc=SeneffEar(tap,16000);
   »for j=1:40
         c=hc(j,:);
         c=filter([1],[1, -.99],c);
         h(j,:)=c(1:100:50381);
    end
»image(h/max(max(h))*length(colormap));
```



Acknowledgments

This routine is based on work described by Benjamin D. Bryant and John D. Gowdy, "Simulation of Stages I and II of Seneff's Auditory Model (SAM) Using Matlab," and published in the *Proceedings of the 1993 Matlab User's Group Conference*.

The detailed description of this model can be found in Stephanie Seneff, "A joint synchrony/mean-rate model of auditory speech processing," *Journal of Phonetics*, Vol. 16, pp. 55-76, 1988.

SeneffEarSetup

Purpose

Design the filters for Seneff's Auditory Model

Synopsis

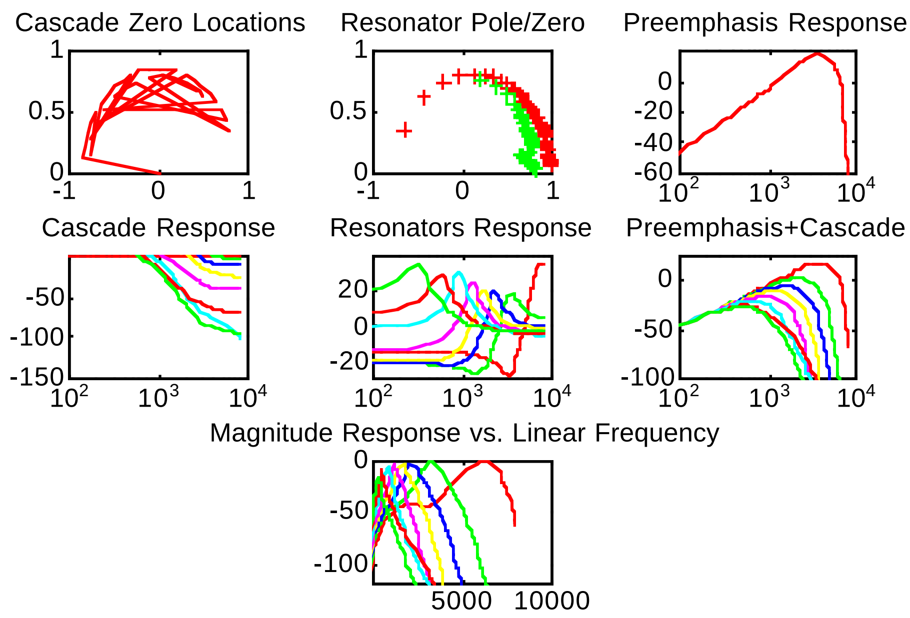
[SeneffPreemphasis, SeneffFilterBank, SeneffForward, SeneffBackward] ...
= SeneffEarSetup(fs)

Description

This function designs the preemphasis and filterbank filters for Seneff's Auditory Model. The only parameter to this function, *fs*, is the desired sampling rate of the digital system. See the SeneffEar command for more details.

Testing

This routine includes test code which can be turned on by setting the *plotTests* variable to a positive value. This produces the following plot showing the filter-bank's response.



Acknowledgments

This routine is based on work described by Benjamin D. Bryant and John D. Gowdy, "Simulation of Stages I and II of Seneff's Auditory Model (SAM) Using Matlab," and published in the *Proceedings of the 1993 Matlab User's Group Conference*.

The detailed description of this model can be found in Stephanie Seneff, "A joint synchrony/mean-rate model of auditory speech processing," *Journal of Phonetics*, Vol. 16, pp. 55-76, 1988.

SetGain

Purpose

Set the gain of a second order section

Synopsis

filter = SetGain(filter, desired, f, fs)

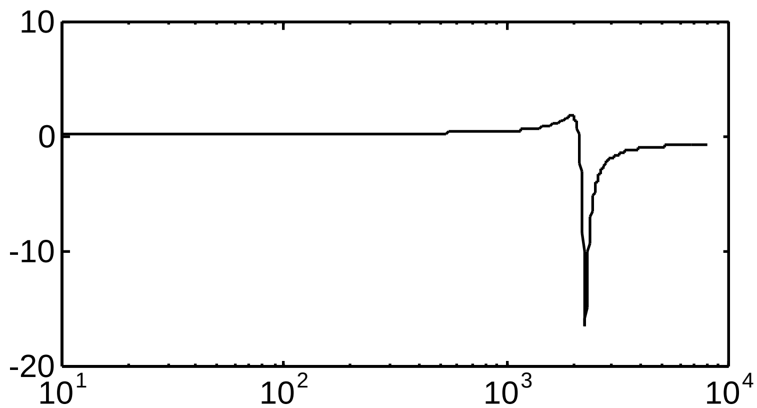
Description

Set the gain of a second order biquadratic filter section to any desired gain at any desired frequency, f, assuming a sampling rate of fs. The filter section is a 1x5 element vector as produced by the *SecondOrderFilter* function.

Examples

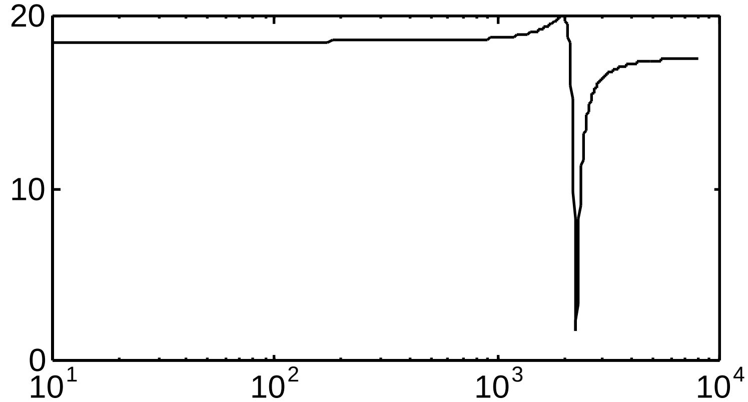
This example shows a second order section designed for Lyon's passive long-wave cochlear model. The frequency response is first plotted for the normal filter.

```
»filts=DesignSosFilters(16000);
»filt=filts(42,:)

filt =

    0.8993   -1.1193    0.8786   -1.2535    0.8899

»f=10:10:7990;
»semilogx(f,FreqResp(filt,f,16000));
```



Then the gain of the filter section is set to 10 (20dB) near the filter's best frequency (1960 Hz). The new frequency response is plotted below.

```
newFilt = SetGain(filt, 10, 1960, 16000);
semilogx(f, FreqResp(newFilt, f, 16000));
```


20
10
0
10^1
10^2
10^3
10^4

soscascade

Purpose

Implement a cascade of second order filters.

Synopsis

output = soscascade(input, coeffs, output, states)

Description

This routine implements a cascade of second order filters. A cascade of filters means that each filter's output is used as input to the next filter. The number of filters defines the number of channels in the filter bank and each channel of the filter bank has its own output waveform. This block is a basic building block for Lyon's passive long-wave cochlear model.

The filter implemented at each stage is a biquadratic section with a transfer function equal to

$$\frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + A_1 z^{-1} + A_2 z^{-2}} .$$

Each filter is described by a five element row vector. The number of filter channels is equal to the number of rows in the coeffs argument to soscascade. Within each row the filter's coefficients are equal to [B0 B1 B2 A1 A2].

The output and states arguments are optional. If present, and they are the right size, then these arrays are used instead of allocating new arrays. If the input has N samples then:

coeffs is C x 5 where C is the number of channels
output is C x N
state is C x 2

If the output argument is not present then a new array is allocated and returned to MATLAB. If the state argument is not present then a new array is allocated and remembered for the next time this function is called. It will be reallocated if the number of filter channels change.

If the first argument, *input*, is the string 'clear' then all internal states are set to zero. It is important to clear the state between runs so that the data at the end of one input array doesn't affect the start of the next run.

Examples

Test this command by trying the following command. The correct results are shown below. The first filter is a simple exponential decay. The second filter sums the last two outputs from the first filter.

```
>>soscascade([1 0 0 0 0],[1 0 0 -.9 0;1 1 0 0 0])

ans =

        1.0000    0.9000    0.8100    0.7290    0.6561
        1.0000    1.9000    1.7100    1.5390    1.3851
```

See Also

LyonPassiveEar

sosfilters

Purpose

Implement a bank of second order filters.

Synopsis

output = sosfilters(input, coeffs, output, states)

Description

This routine implements a bank of second order filters. Each channel of the filter bank is independent of the other filters. The number of filters defines the number of channels in the filter bank and each channel of the filter bank has its own output waveform. This block is a basic building block for the Patterson-Holdsworth ERB cochlear model.

The filter implemented at each stage is a biquadratic section with a transfer function equal to

$$\frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 + A_1 z^{-1} + A_2 z^{-2}} .$$

Each filter is described by a five element row vector. The number of filter channels is equal to the number of rows in the coeffs argument to soscascade. Within each row the filter's coefficients are equal to [B0 B1 B2 A1 A2].

The output and states arguments are optional. If present, and they are the right size, then these arrays are used instead of allocating new arrays. If the input has N samples then:

coeffs is C x 5 where C is the number of channels
output is C x N
state is C x 2

If the output argument is not present then a new array is allocated and returned to MATLAB. If the state argument is not present then a new array is allocated and remembered for the next time this function is called. It will be reallocated if the number of filter channels change.

If the first argument, *input*, is the string ‘clear’ then all internal state is set to zero. It is important to clear the state between runs so that the data at the end of one input array doesn't affect the start of the next run.

Examples

Test this command by trying the following commands. The correct results are shown below. The first example filters an impulse with two low pass filters.

```
»sosfilters([1 0 0 0 0 0],[1 0 0 -.9 0;1 0 0 -.8 0])

ans =

    1.0000   0.9000   0.8100   0.7290   0.6561    0.5905
    1.0000   0.8000   0.6400   0.5120   0.4096    0.3277
```

The next example shows a variant; multiple input arrays, one per filter channel.

```
»sosfilters([1 0 0 0 0 0;2 0 0 0 0 0], ...
            [1 0 0 -.9 0;1 0 0 -.8 0])

ans =

    1.0000   0.9000   0.8100   0.7290   0.6561    0.5905
    2.0000   1.6000   1.2800   1.0240   0.8192    0.6554
```

Finally, multiple input arrays can be filtered by a single filter.

```
»sosfilters([1 0 0 0 0 0;2 0 0 0 0 0],[1 0 0 -.9 0])

ans =

    1.0000   0.9000   0.8100   0.7290   0.6561    0.5905
    2.0000   1.8000   1.6200   1.4580   1.3122    1.1810
```

See Also

soscascade

spectrogram

Purpose

Compute the spectrogram of a signal

Synopsis

```
array = spectrogram(wave,segsize,nlap,ntrans)
```

Description

Compute a short-time Fourier transform (STFT) or spectrogram of a one-dimensional signal. The following optional arguments are used to control the window size, overlap, and thus the image quality.

segsize(128) Size of a segment of data used to calculate each frame. This determines the basic frequency resolution of the spectrogram. Smaller segment sizes give more detailed time resolution, but at the expense of frequency resolution.

nlap (8) Number of hamming windows overlapping a point. Larger overlaps give better resolution in the time domain.

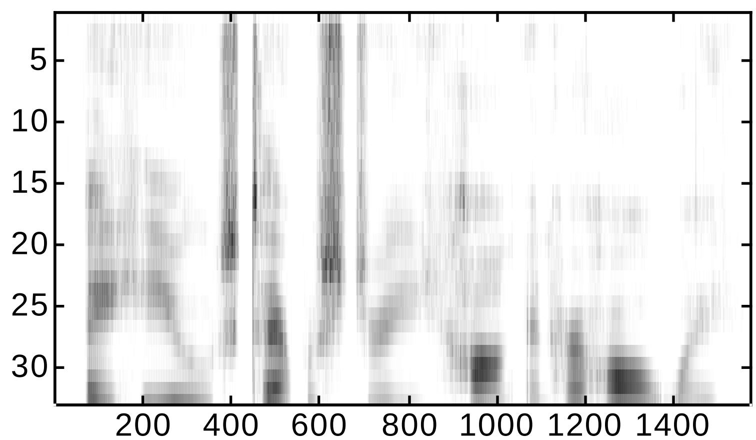
ntrans (4) Factor by which transform is bigger than segment, larger sizes magnify the frequency axis, but don't really give any better resolution.

This function returns a spectrogram 'array' compressed with the square root of the maximum amplitude (fourth root of power)

Examples

A spectrogram of the 'A huge tapestry hung in her hallway' utterance from the TIMIT database (TRAIN/DR5/FCDR1/SX106/SX106.ADC) is shown below. It was computed using the following command line.

```
i=spectrogram(tap,64,2,1);
```



Author

Richard F. Lyon wrote the stabilization code and the smoothing algorithm.