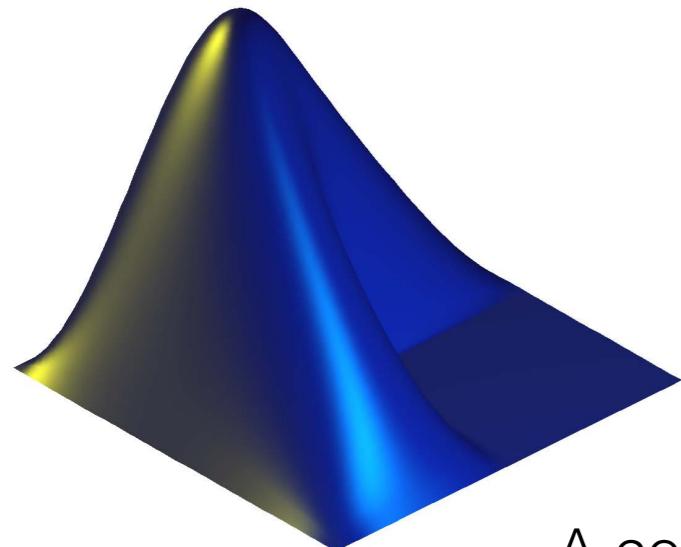


ISMIR 2014 Tutorial, 27.10.2014
Slides version 0.3, 22.10.2014
Current version available at:

<https://miningsuite.googlecode.com/svn/trunk/ismir2014tutorial.pdf>



MiningSuite

A comprehensive framework for music analysis, articulating audio
(MIRtoolbox 2.0) and symbolic approaches

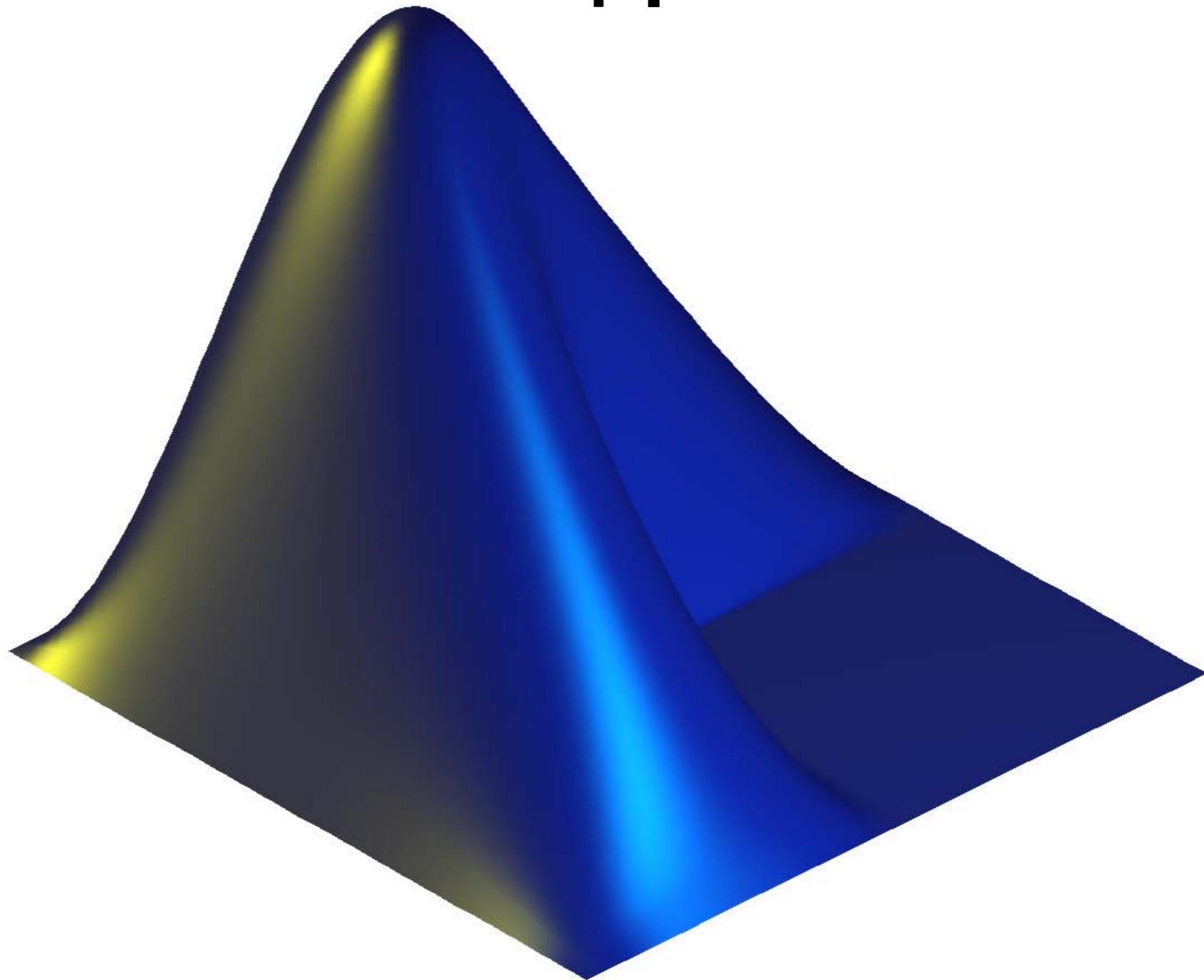
Olivier Lartillot
Aalborg University, Denmark



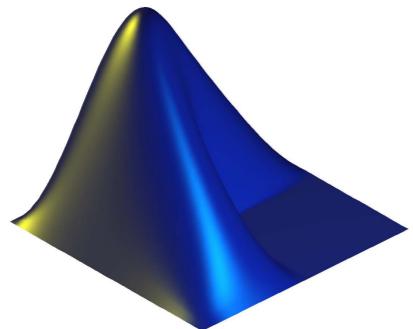
Tutorial overview

1. General description: Aims, architecture, syntax, ...
2. Audio-based approaches (*MIRtoolbox* 2.0)
3. Symbolic approaches: “*MIDItoolbox* 2.0” and new musicological models
4. How it works
5. How you can contribute: open-source community

1.



General description



MiningSuite

- Matlab framework
- Large range of audio and music analysis tools
- Both audio and symbolic representations
- Highly adaptive syntactic layer on top of Matlab
- Syntactic layer within the operators' Matlab code, simplifying and clarifying the code
- Memory management mechanisms



MIRtoolbox

- Matlab framework ✓ (but intricate, non-optimized)
- Large range of audio and music analysis tools ✓
- Both audio and symbolic representations ✗
- Highly adaptive syntactic layer on top of Matlab ✓
- Syntactic layer within the operators' Matlab code, simplifying and clarifying the code ✗
- Memory management mechanisms ✓

What's new in MiningSuite

- Code & architecture entirely rebuilt
 - More efficient, more readable, better organised, better generalisable
- Integration audio / symbolic representations
- Innovative and integrative set of symbolic-based musicological tools and pattern mining
- Syntactic layer within the operators' Matlab code, simplifying and clarifying the code
- Open-source collaborative environment

Aim of this tutorial

- Overview of the audio and symbolic approaches
 - Quick tour of computational audio/music analysis
- Take benefit of the capabilities of the environment
 - User-friendly syntax
 - Understand how the framework works
 - Add new features and new codes in the project

MIRtoolbox history

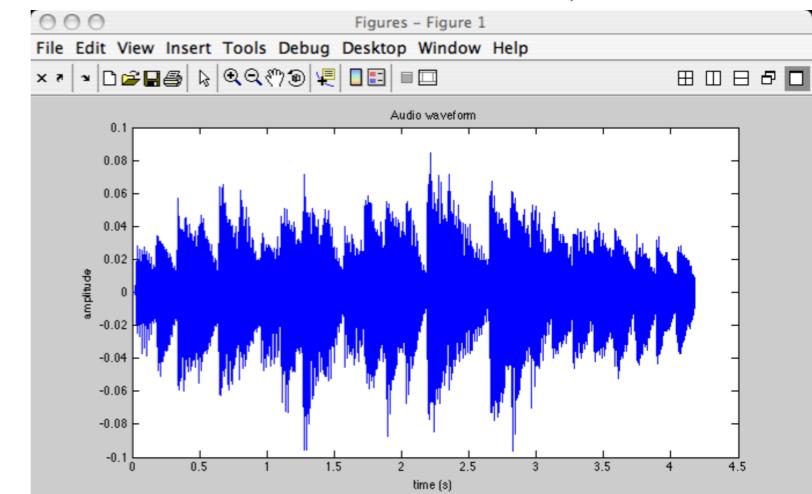
- Matlab toolboxes @ University of Jyväskylä:
 - *MIDItoolbox* (Eerola & Toiviainen, 2004)
 - *Music Therapy Toolbox* (Lartillot, Toiviainen, Erkkilä, 2005)
- European project *Tuning the Brain for Music* (NEST, FP6, 2006–08)
 - **Large range of audio/musical features** extracted from large databases, linked to emotional ratings (ISMIR 2009)
- Master program @ University of Jyväskylä, MIR course (2005–)
 - Toolbox **easy to use**, no Matlab expertise required
- version 1.0 in 2007, current version 1.5.

MIRtoolbox advantages

- Highly modular framework:
 - building blocks can be reused, reordered
 - no need to recode processes, to reinvent the wheel
- Adaptive syntax: users can focus on design, *MIRtoolbox* takes care of technical details
- Free software, open source: Capitalized expertise of the research community, for everybody
- 10000s download, 500+ citations, reference tool in MIR

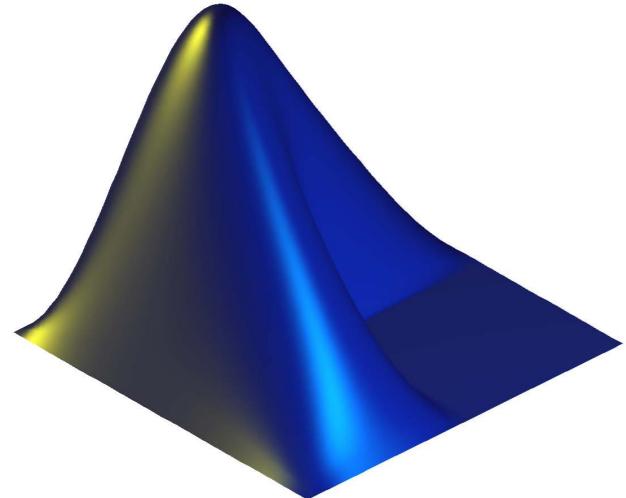
MIRtoolbox syntax

- ***miraudio('ragtime.wav')*** outputs a figure:
- *miraudio('ragtime.wav');* blocks figure display
- ***mirtempo('ragtime.wav')***
- *mirtempo('Folder')* operates on all files in the Current Directory
- ***a = miraudio('ragtime.wav', 'Extract', 0, 60, 's')***
- ***a = miraudio(a, 'Center');***
- ***t = mirtempo(a)***
- ***d = mirgetdata(t)*** returns the result in *Matlab* array
- ***get(t, 'Sampling')*** returns additional data stored in *MIRtoolbox* object



MiningSuite history

- Academy of Finland research fellowship, 2009-14
 - Integrating audio and symbolic into common framework, higher-level music analysis
- *MIRtoolbox*: a “Rube Goldberg machine”
 - Obscure architecture, obscure code, highly inefficient in speed and memory → rewrite
- *MIRtoolbox* innovative framework draws interest outside MIR
 - Reorganize framework into discipline-focused packages
- *MIDItoolbox* did not evolve since 1.0 (2004)



MiningSuite

- **SIGMINR**: signal processing
- **AUDMINR**: auditory modelling
- **MUSMINR**: music analysis
- **PATMINR**: pattern mining
- **SEQMINR**: sequence processing
- **VOCMINR**: voice analysis?

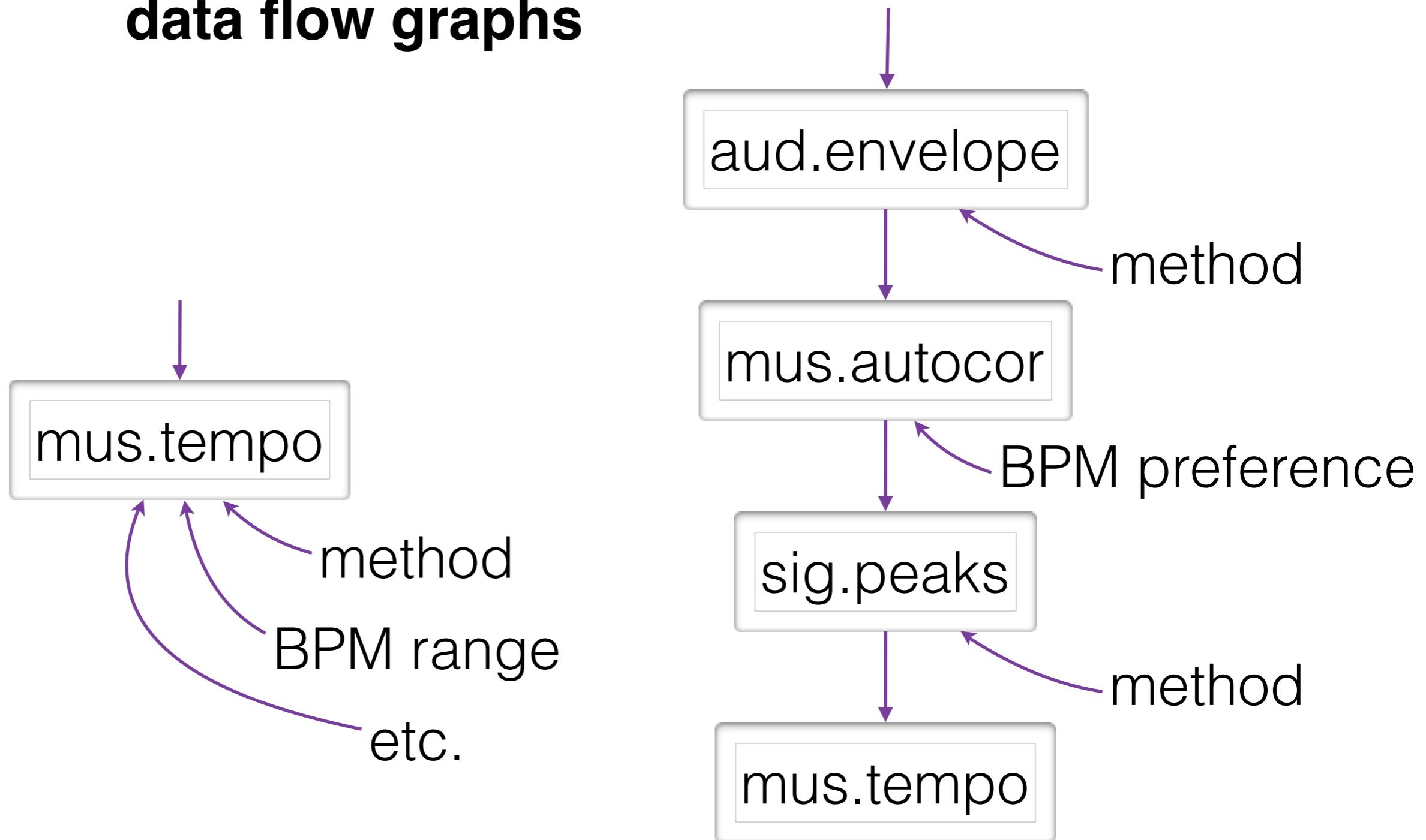
Packages

- In *MIRtoolbox*, all operators start with *mir...* prefix (***miraudio***, ***mirspectrum***, etc.)
 - to avoid conflicts with other Matlab functions
- In *MiningSuite*, each module (*SigMinr*, *AudMinr*, etc.) is a package: its operators are called using a particular prefix (***sig.spectrum***, ***aud.spectrum***, etc.)

Signal domain

- **SIGMINR**
 - sig.input, sig.spectrum, ...
- **AUDMINR**
 - aud.spectrum, ...
 - aud.mfcc, aud.brightness, ...
- **MUSMINR**
 - mus.spectrum, ...
 - mus.tempo, mus.key, ...
- Sets of operators related to signal processing operations, audio and musical features
- Versions specific to particular domains
- Each operator can be tuned with a set of options

Signal domain: Modular data flow graphs



Symbolic domain

- MUSMINR
- ~~Succession of operations applied to input signal?~~
- Several types of analysis applied *together for each successive note* of the symbolic sequence
- One single operator: ***mus.score***
- Types of analysis selected as options of *mus.score*

Symbolic domain

- **AUDMINR**: *aud.score* (auditory scene transcription)
- **MUSMINR**: *mus.score*
- **SEQMINR**: sequence management
- **PATMINR**: sequential pattern mining

Software dependencies

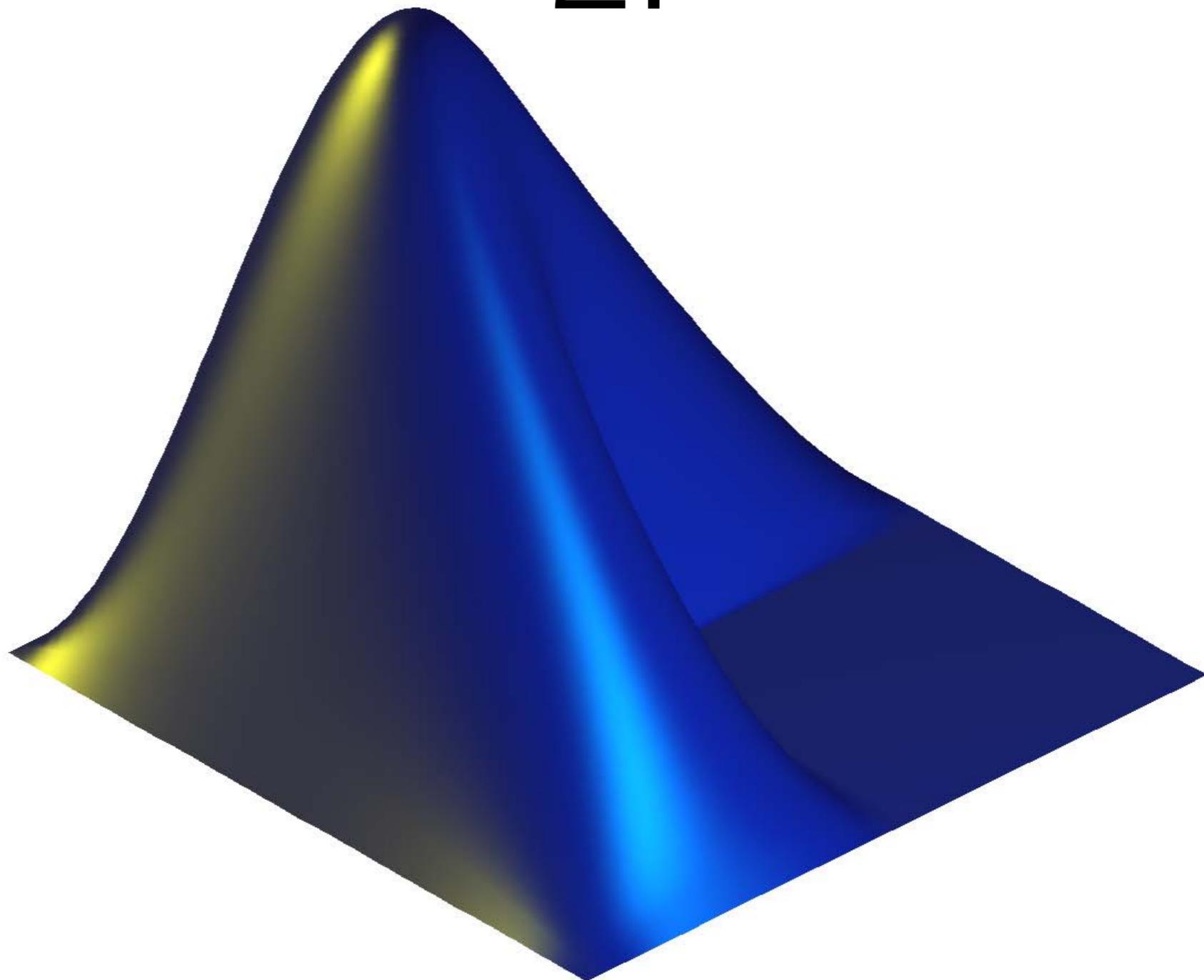
- *MathWorks' Matlab* 7.6 (r2008a) or newer versions.
 - But *Matlab* 8.2 (r2013b) or newer is strongly recommended, because *MiningSuite* easily crashes on previous versions*.
- *MathWorks' Signal Processing Toolbox*
- Included in the distribution: *Auditory Toolbox* (Slaney), *Music Analysis Toolbox* (Pampalk)

* versions 7.6 to 8.2: there exists a workaround based on debug mode.

MiningSuite development

- Started in 2010
- Sneak Peek version 0.6 released in January 2014 for AES Semantic Audio. Proof of concept, basic architecture
- Beta version 0.8 released now for ISMIR 2014. **Focus on the architecture**, not on the exact feature implementation. Analytical results cannot be trusted!
- Version 0.9 will contain most of the *MIRtoolbox* implementation, user's guide (in wiki) and code documentation.
- Version 1.0 with complete bench test. We need you..
- Further versions: We definitely need you!

2.



Audio-based approaches

SIGMINR

signal processing

sig.input	sig.frame	sig.peaks	sig.segment
	sig.flux		
sig.rms	sig.autocor		sig.stat...
sig.zerocross	sig.spectrum	sig.rolloff	sig.export
	sig.cepstrum		
sig.filterbank		sig.simatrix	sig.cluster
sig.envelope			

AUDMINR

auditory modeling

aud.input

aud.spectrum

aud.envelope

aud.play

aud.brightness

aud.filterbank

aud.save

aud.roughness

aud.mfcc

aud.attacktime

aud.segment

aud.attackslope

aud.simatrix

aud.novelty

aud.score

MUSMINR

music theory

mus.spectrum

mus.pitch

mus.chromagram

mus.fluctuation

mus.keystrength

mus.tempo

mus.key

mus.pulseclarity

mus.keysom

mus.metre

mus.mode

mus.score

sig.input

- In *MIRtoolbox*: *miraudio*. But *SigMinr* is not restricted to audio only, but any kind of signal.
- *sig.signal* is the actual signal object (cf. part IV)
- *sig.input* takes care of the transformations of the signal representation before further processing
- *sig.input(v, sr)* converts a *Matlab* vector *v* into a signal of sampling rate *sr*

sig.input

- *sig.input('myfile')**
 - Formats: .wav, .flac, .ogg, .au, .mp3, .mp4, .m4a
 - Using Matlab's *audioread*
- * Indicate the file extension as well (because *audioread* requires that).

(*sig.input* actually calls routine from *AudMinr* for the processing of audio files.)

sig.input: transformations

- By default, multi-tracks are summed into mono.
 - *sig.input('mysong', 'Mix', 'no')* keeps the multitrack decomposition.
- *sig.input(..., 'Center')* centers the signal.
- *sig.input(..., 'Sampling', r)* resamples at rate r (in Hz.), using *resample* from *Signal Processing Toolbox*
- *sig.input(..., 'Normal')* normalizes with respect to RMS

sig.input: extraction

- *sig.input(..., ‘Extract’, 1, 2, ‘s’, ‘Start’)*
 - extracts signal from 1 s to 2 s after the start
- *sig.input(..., ‘Extract’, 44100, 88200, ‘sp’)*
 - from samples #44100 to 88200 after the start
- *sig.input(..., ‘Extract’, -1, +1, ‘Middle’)*
 - from 1 s before to 1 s after the middle of the signal
- *sig.input(..., ‘Extract’, -10000, 0, ‘sp’, ‘End’)*
 - the last 10000 samples in the signal

sig.input: trimming silence

- *sig.input(..., ‘Trim’)* trims (pseudo-)silence at start and end
- *miraudio(..., ‘TrimStart’)* at start only
- *miraudio(..., ‘TrimEnd’)* at end only
- *miraudio(..., ‘TrimThreshold’, thr)* specifies the silence threshold. Default *thr* = *.06*
- Silent frames have RMS energy below *thr* times the medium RMS energy of the whole audio file.

aud.play: sonification

aud.save: save in audio file

Output file names can be specified in different ways:

<i>a = miraudio('mysong.au')</i>	mysong.au
<i>mirsave(a)</i>	mysong.mir.au
<i>mirsave(a,'new')</i>	new.au
<i>mirsave(a,'.wav')</i>	mysong.mir.wav
<i>mirsave(a,'new.wav')</i>	new.wav

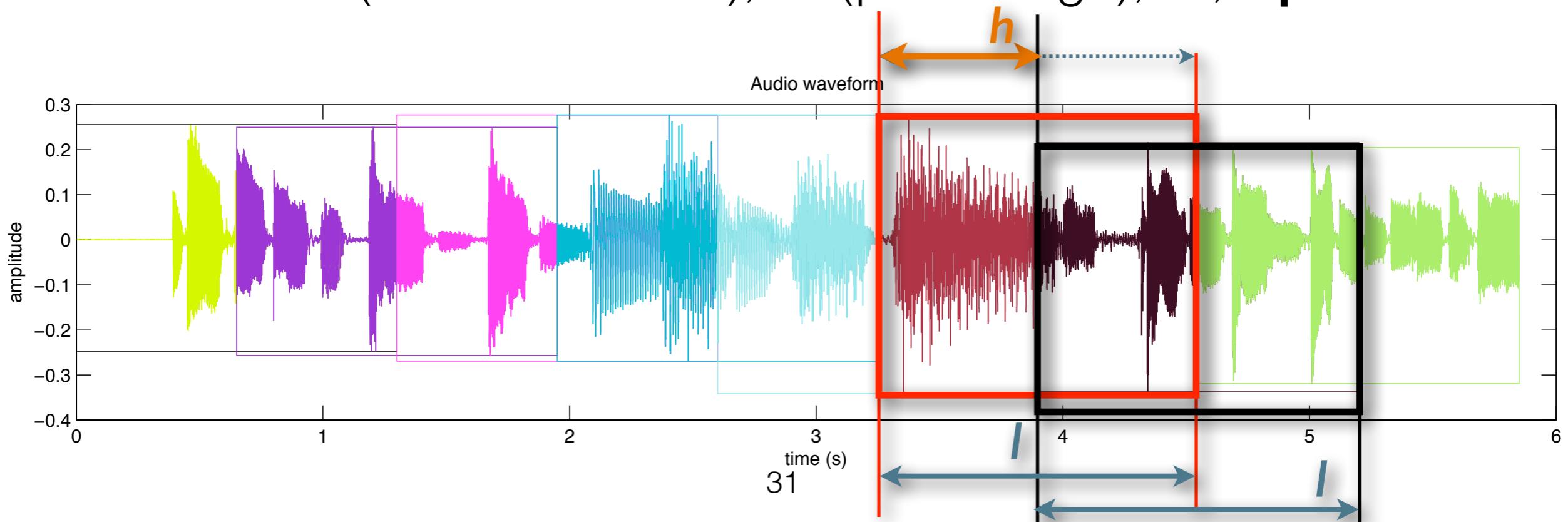
<i>a = miraudio('Folder')</i>	song1.wav	song2.wav	song3.au
<i>mirsave(a)</i>	song1.mir.wav	song2.mir.wav	song3.mir.au
<i>mirsave(a,'new')</i>	song1new.wav	song2new.wav	song3new.au
<i>mirsave(a,'.wav')</i>	song1.mir.wav	song2.mir.wav	song3.mir.wav
<i>mirsave(a,'new.wav')</i>	song1new.wav	song2new.wav	song3new.wav

sig.frame

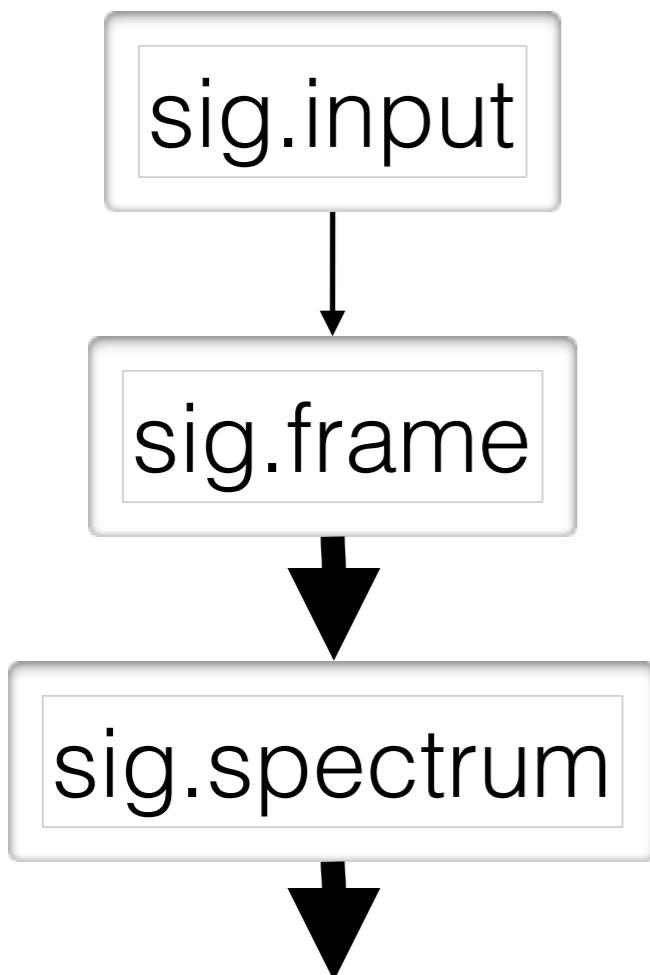
frame decomposition

- $f = \text{sig.frame}(\dots, \text{'FrameSize'}, l, \text{'s'})$
 - unit: '**s**' (seconds), '**sp**' (samples)
- $f = \text{sig.frame}(\dots, \text{'FrameHop'}, h, \text{'/1'})$
 - unit: '**/1**' (ratio from 0 to 1), '**%**' (percentage), '**s**', '**sp**'

*aud.play(f)
aud.save(f)*



sig.frame frame decomposition



- $a = \text{sig.input}(\dots)$
- $f = \text{sig.frame}(a)$
- $s = \text{sig.spectrum}(f)$

Or: $s = \text{sig.spectrum}(\dots, \text{'Frame'})$

‘*Frame*’ option frame decomposition

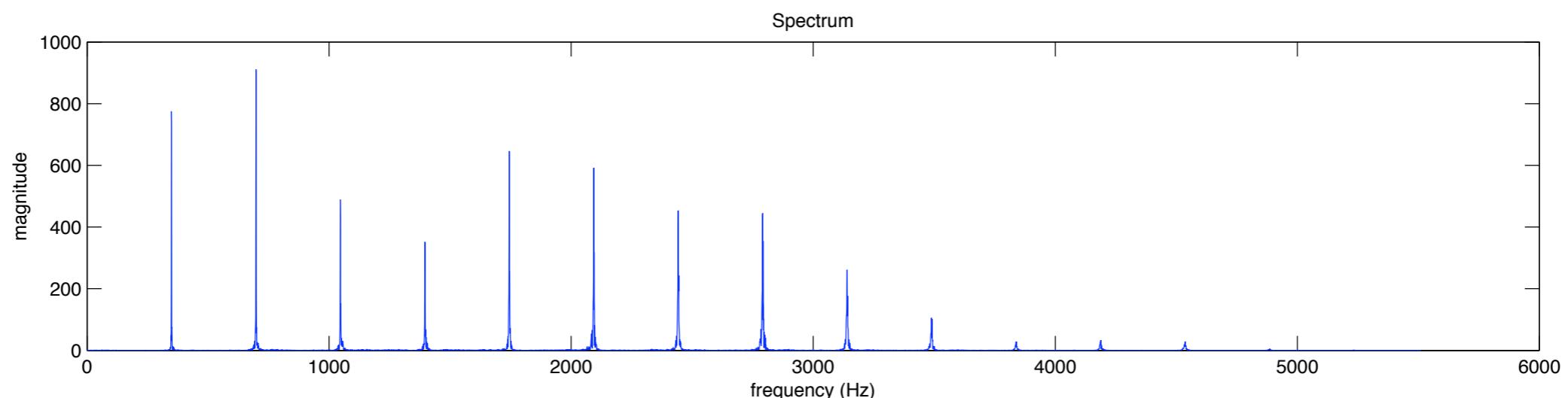
- *sig.input(..., ‘FrameSize’, ..., ‘FrameHop’, ...)*
- *sig.spectrum(..., ‘FrameSize’, ..., ‘FrameHop’, ...)*
- *sig.spectrum(..., ‘Frame’)* (default frame configuration)
- ‘*Frame*’ option available to most operators, each with its own default frame configuration
- Each operator can perform the frame decomposition where it is most suitable.

sig.spectrum frequency spectrum

Discrete Fourier Transform of audio signal x:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N - 1$$

Amplitude spectrum: $s = \text{sig.spectrum}(\dots)$



Phase spectrum: $s.\text{phase}$

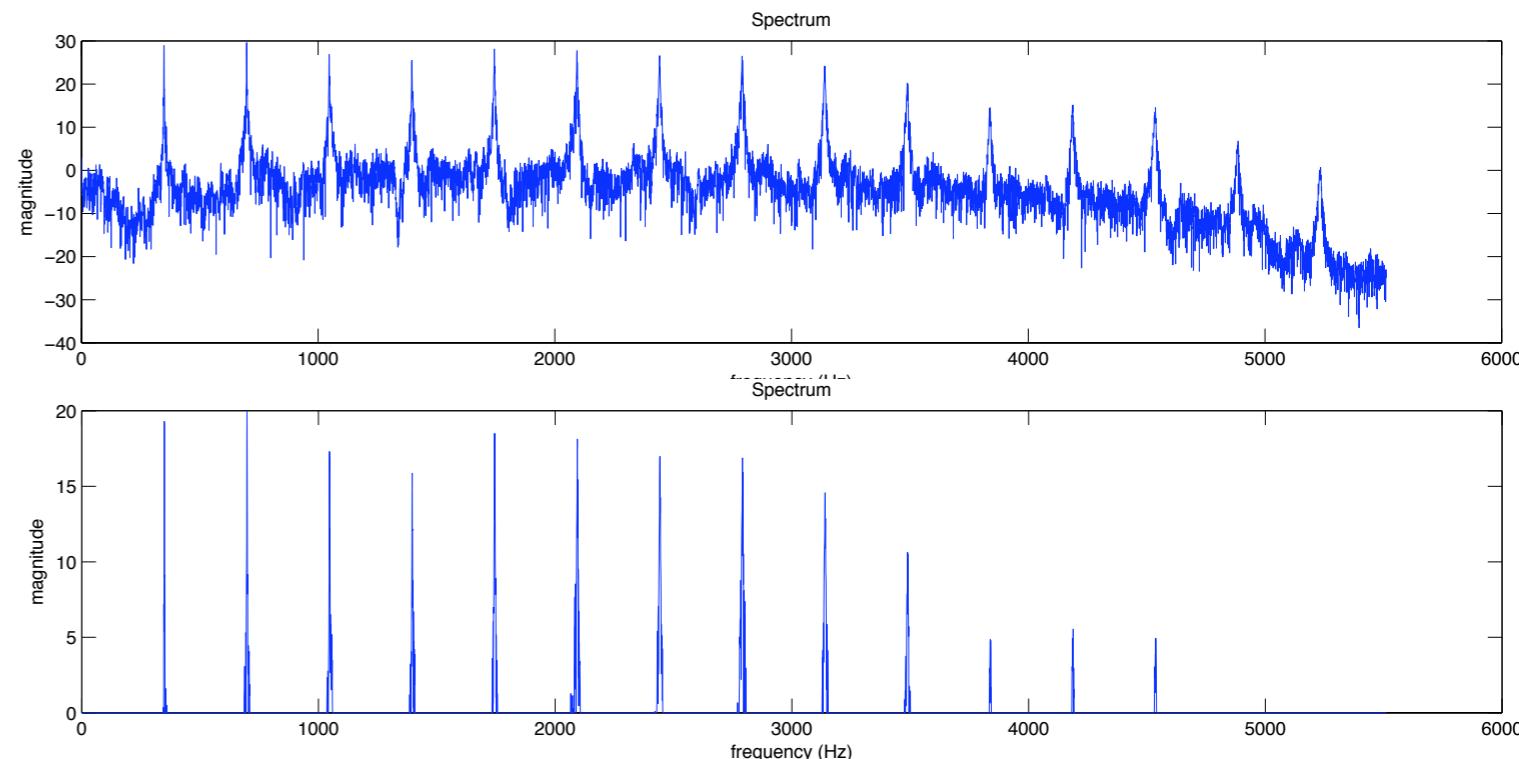
Fast Fourier Transform using Matlab's *fft* function.

sig.spectrum parameter specification

- *sig.spectrum(..., 'Min', 0)* in Hz
- *sig.spectrum(..., 'Max', sampling rate/2)* in Hz
- *sig.spectrum(..., 'Window', 'hamming')*
- frequency resolution r , in Hz:
 - *sig.spectrum(..., 'Res', r)*: exact resolution specification
 - *sig.spectrum(..., 'MinRes', r)*: minimal resolution (less precise constraint, but more efficient)
- *sig.spectrum(..., 'Phase', 'No')*

sig.spectrum post-processing

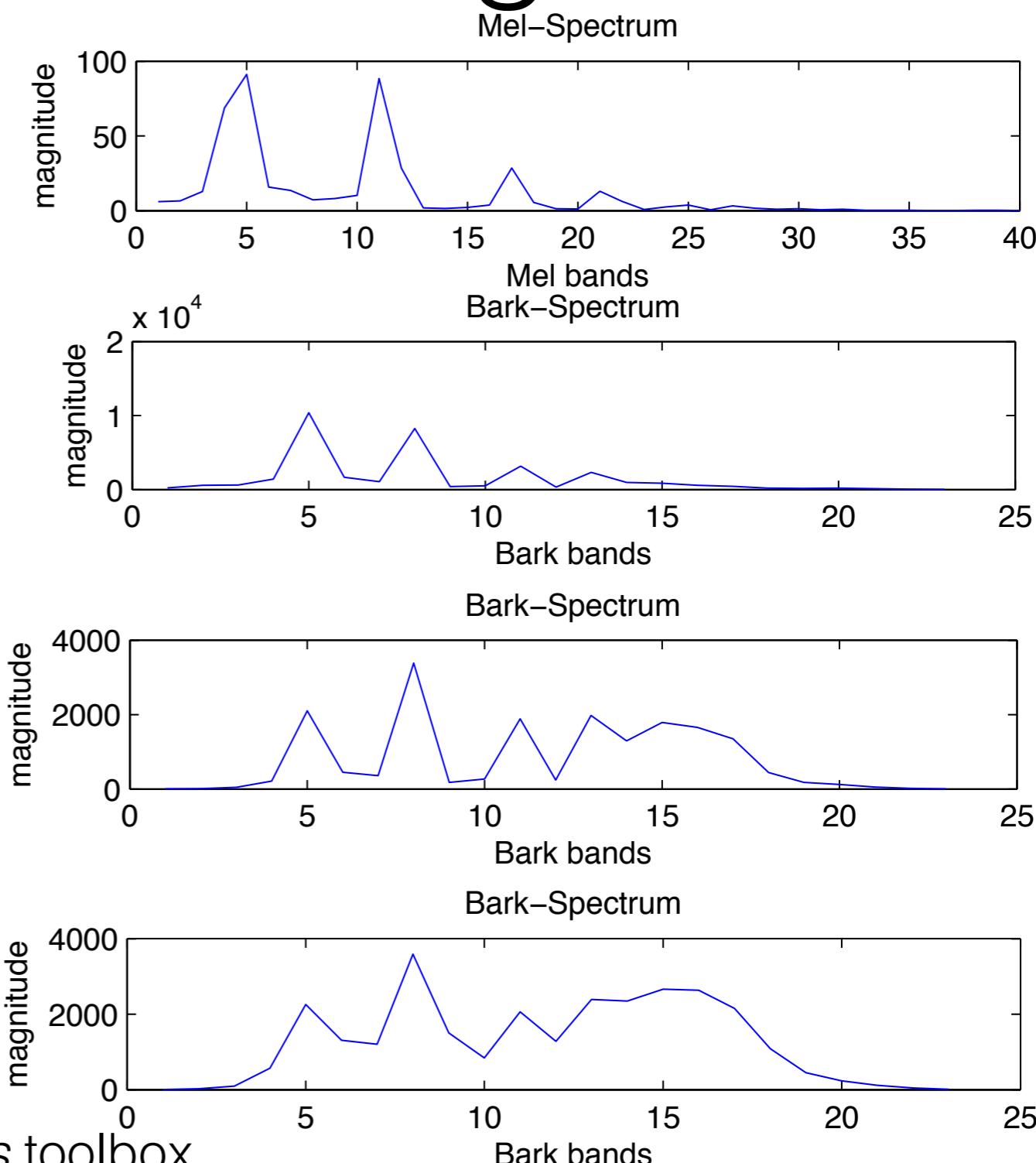
- *mirspectr um(..., 'Normal')* normalizes w.r.t. energy.
- *mirspectr um(..., 'Power')* squares the energy.
- *mirspectr um(..., 'dB')*
 - in dB scale
- *mirspectr um(..., 'dB', th)*
 - th highest dB
- *mirspectr um(..., 'Smooth', o)*
- *mirspectr um(..., 'Gauss', o)*



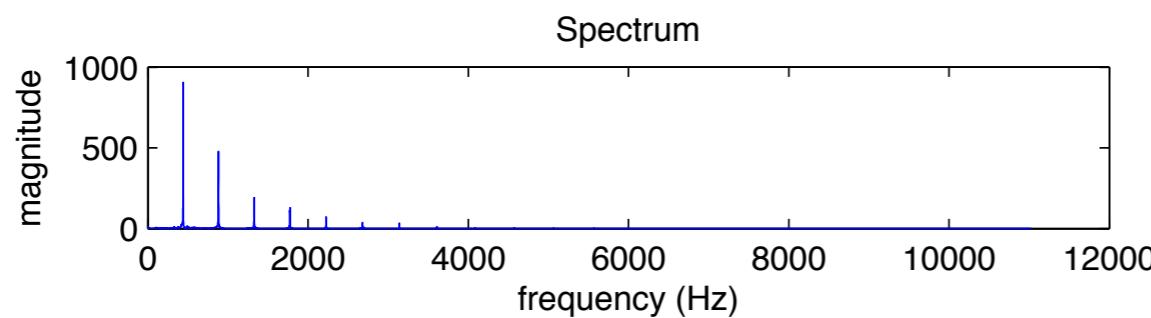
aud.spectrum

auditory modeling

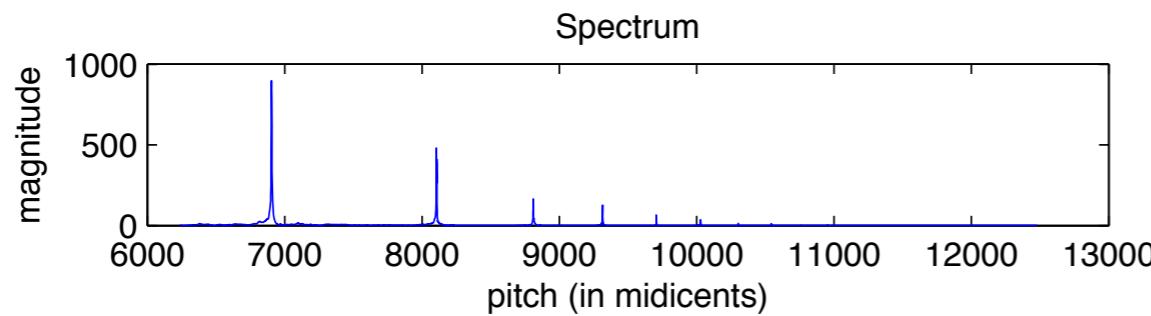
- *aud.spectrum(..., 'Mel')*
 - Mel-band decomposition
- *aud.spectrum(..., 'Bark')*
 - Bark-band decomposition
- *aud.spectrum(..., 'Terhardt')*:
 - Outer-ear modeling
- *aud.spectrum(..., 'Mask')*:
 - Masking effects along bands



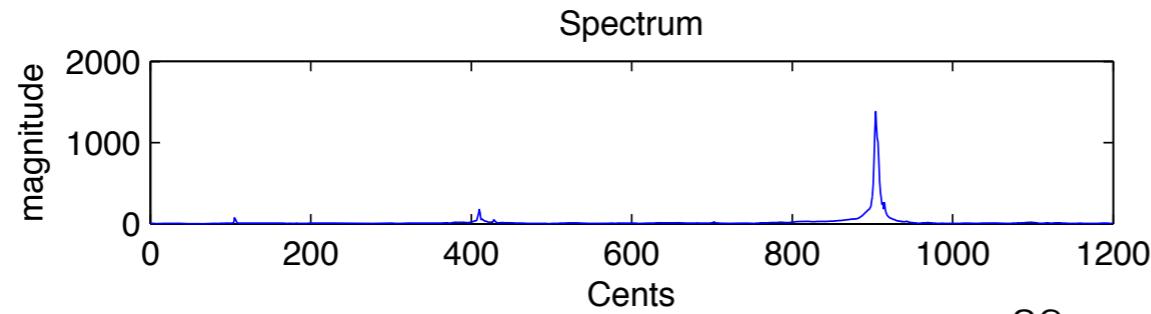
mus.spectrum pitch-based distribution



- *mus.spectrum(..., 'Cents')*



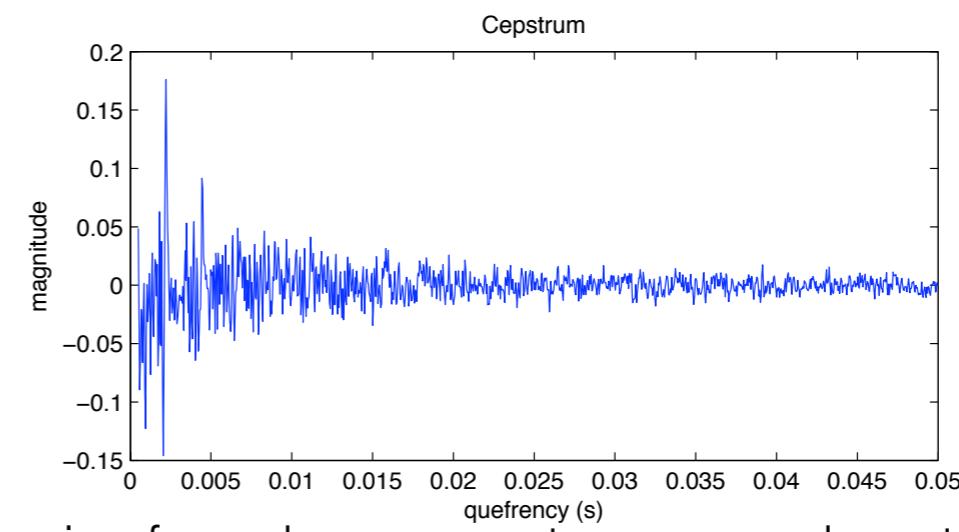
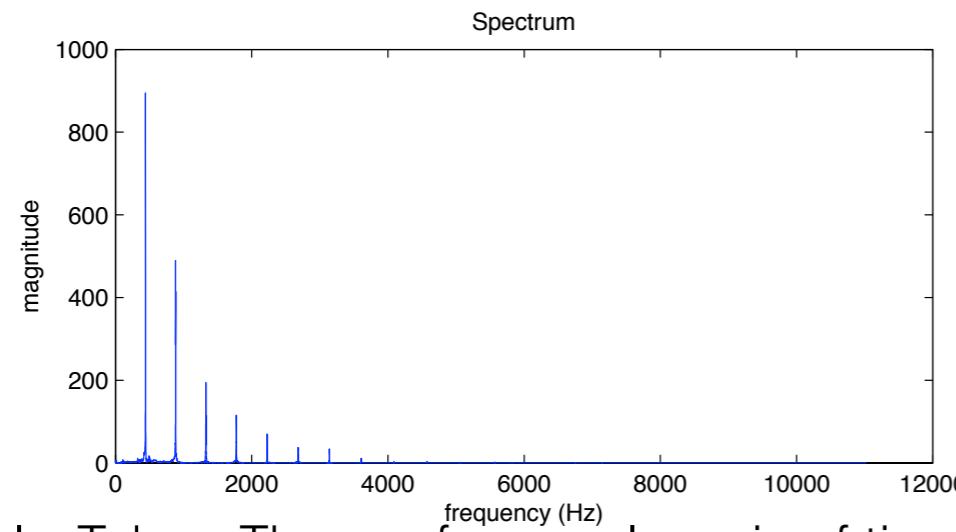
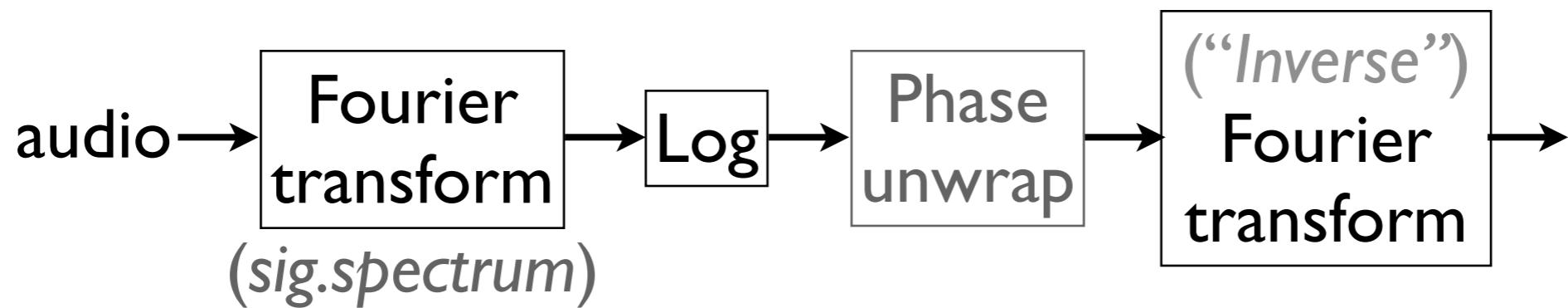
- *aud.spectrum(..., 'Collapsed')*: into one octave, divided into 1200 cents



sig.cepstrum

cepstral analysis

sig.cepstrum(..., ‘Complex’): complex cepstrum

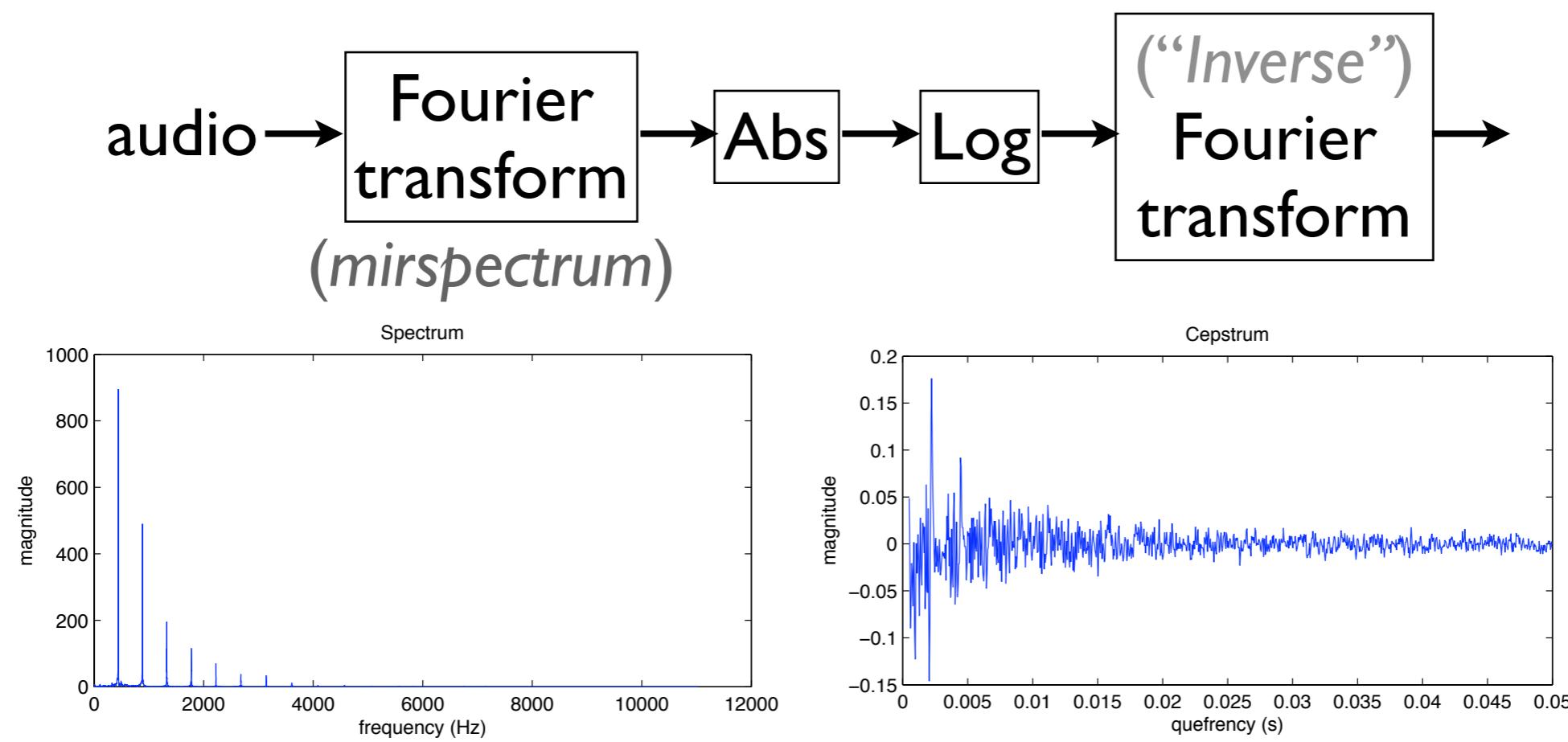


Bogert, Healy, Tukey. The quefrency alalysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. Symposium on Time Series Analysis, Chapter 15, 209-243, Wiley, 1963.

sig.cepstrum

cepstral analysis

sig.cepstrum(..., ‘Real’): real cepstrum



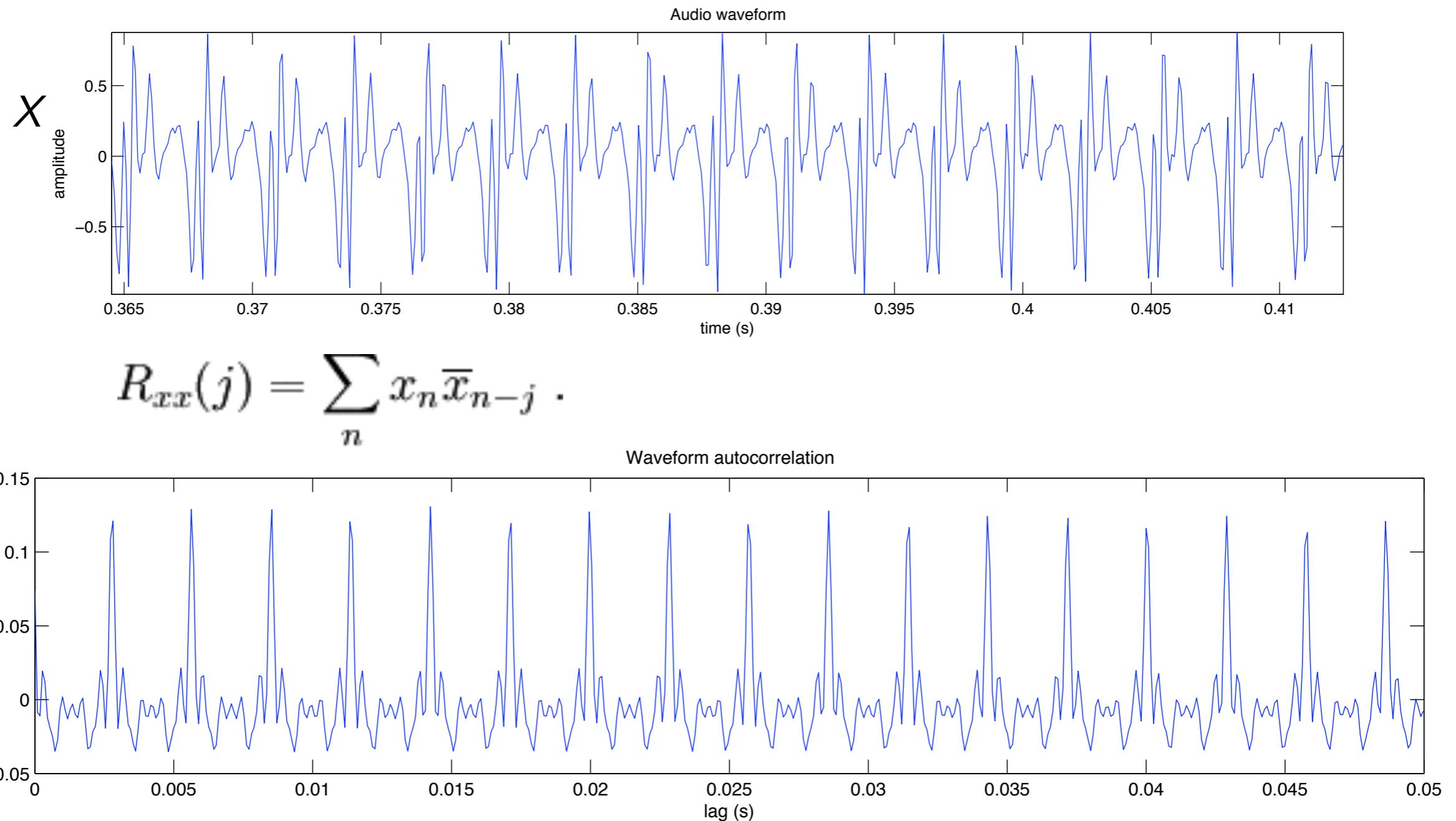
Bogert, Healy, Tukey. The quefrency alalysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. Symposium on Time Series Analysis, Chapter 15, 209-243, Wiley, 1963.

sig.cepstrum cepstral analysis

- *sig.cepstrum(..., 'Freq')*
 - represented in the frequency domain
- *sig.cepstrum(..., 'Min', 0, 's')*
- *sig.cepstrum(..., 'Max', .05, 's')*

sig.autocor

autocorrelation function



sig.autocor

autocorrelation function

- *sig.autocor(..., 'Min', t1, 's')* $t1=0$ s
- *sig.autocor(..., 'Max', t2, 's')* $t2=.05$ s (audio) or $t2=2$ s (envelope)
- *sig.autocor(..., 'Freq')* lags in Hz.
- *sig.autocor(..., 'Window', w)* $w='hanning'$ specifies a windowing method
- *sig.autocor(..., 'NormalWindow', f)* $f='on'='yes'=1$ divides by autocorrelation of the window*
- *sig.autocor(..., 'Halfwave')* half-wave rectification

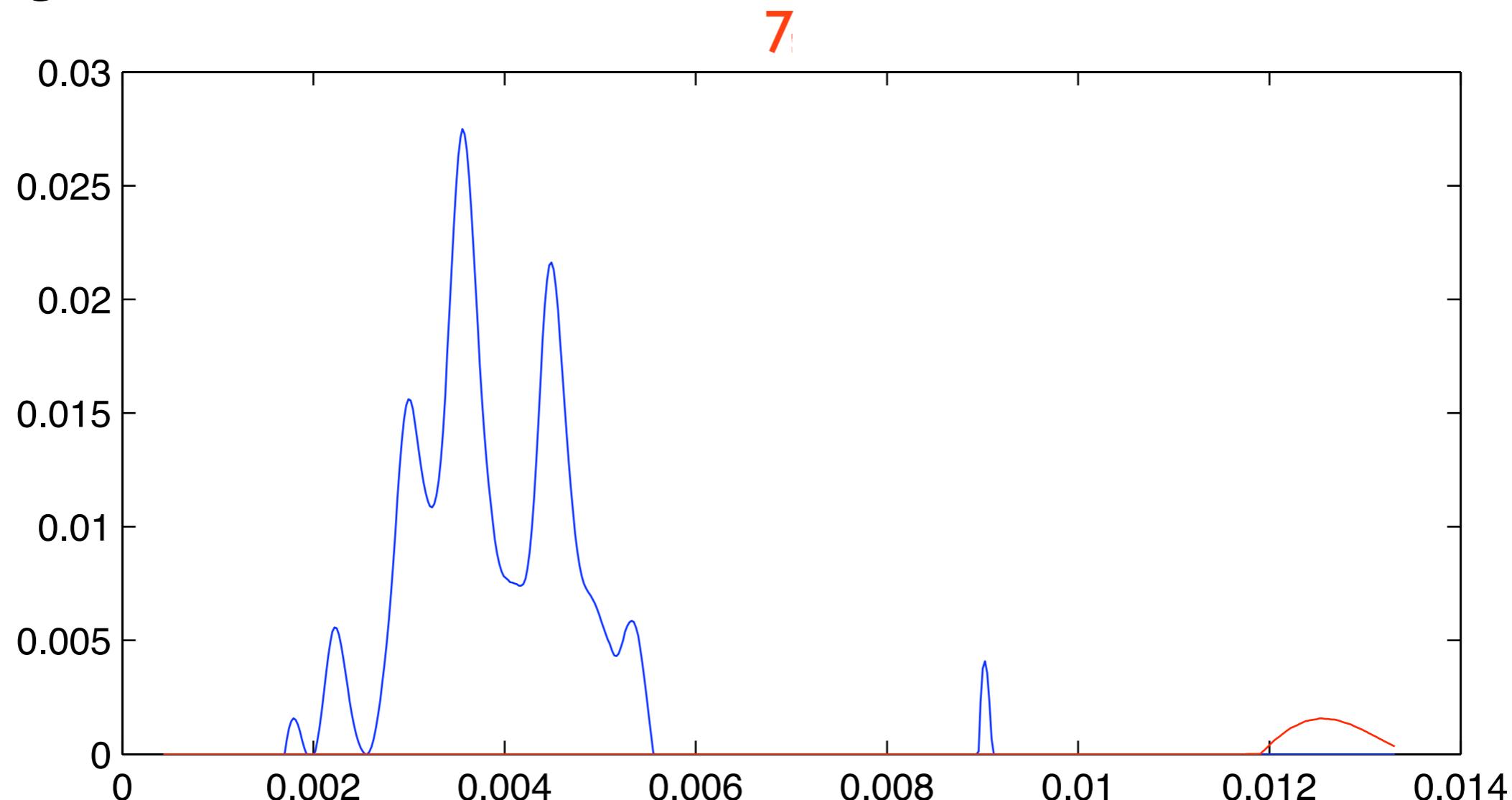
*Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound, IFA Proceedings 17: 97-110, 1993.

sig.autocor generalized autocorrelation

- Autocorrelation (by default):
 - $y = IDFT(|DFT(x)|^2)$
- Generalized autocorrelation:
 - $y = IDFT(|DFT(x)|^k)$
 - $sig.autocor(\dots, \text{'Compress'}, k)$ $k=.67$

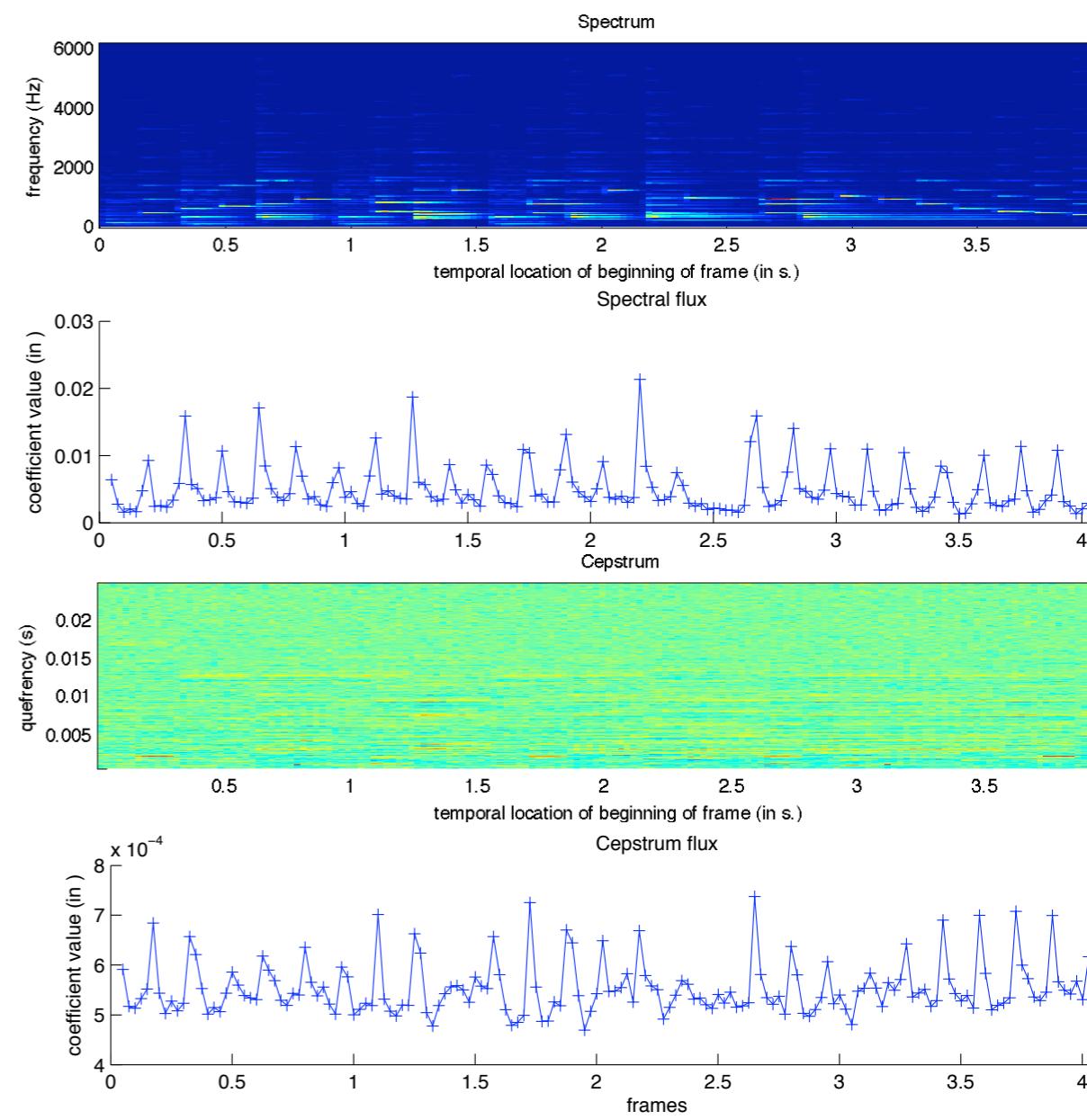
sig.autocor generalized autocorrelation

- *sig.autocor('Amin3', 'Enhanced', 2:10)*



sig.flux: distance between successive frames

- $s = \text{sig.spectrum}(a, \text{'Frame'})$
- $\text{sig.flux}(s) = \text{sig.flux}(a)$
- $c = \text{sig.cepstrum}(a, \text{'Frame'})$
- $\text{sig.flux}(c)$



sig.flux: distance between successive frames

- *sig.flux(..., ‘**Dist**’, d)* *d = ‘Euclidian’, ‘City’, ‘Cosine’* specifies the distance measure.
- *sig.flux(..., ‘**Inc**’)* positive differences
- *sig.flux(..., ‘**Complex**’)* complex flux
- *sig.flux(..., ‘**Median**’, l, C)*
- *sig.flux(..., ‘**Median**’, l, C, ‘**Halfwave**’)*

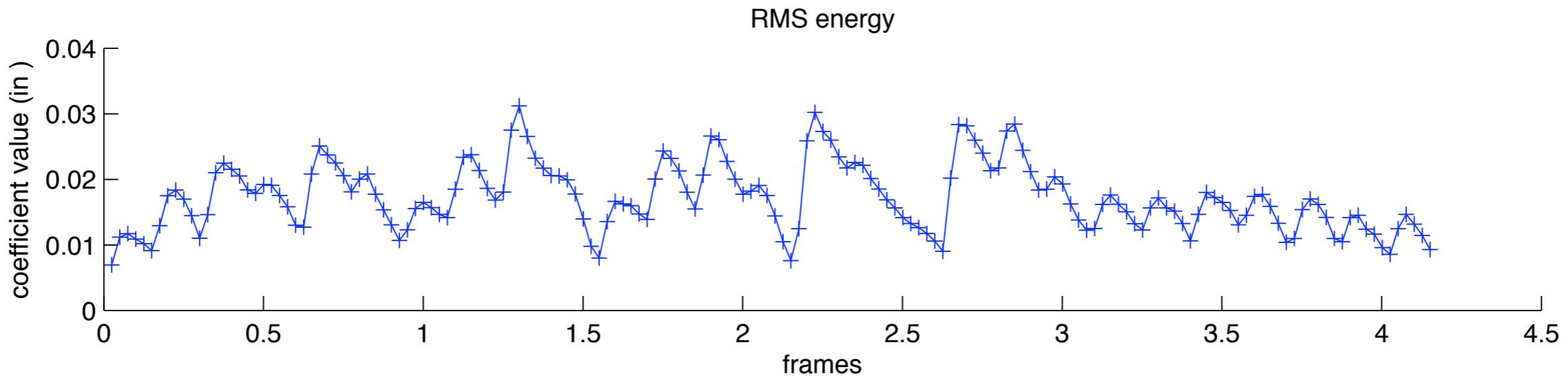
Audio level

Dynamics
Sound 

sig.rms root-mean square

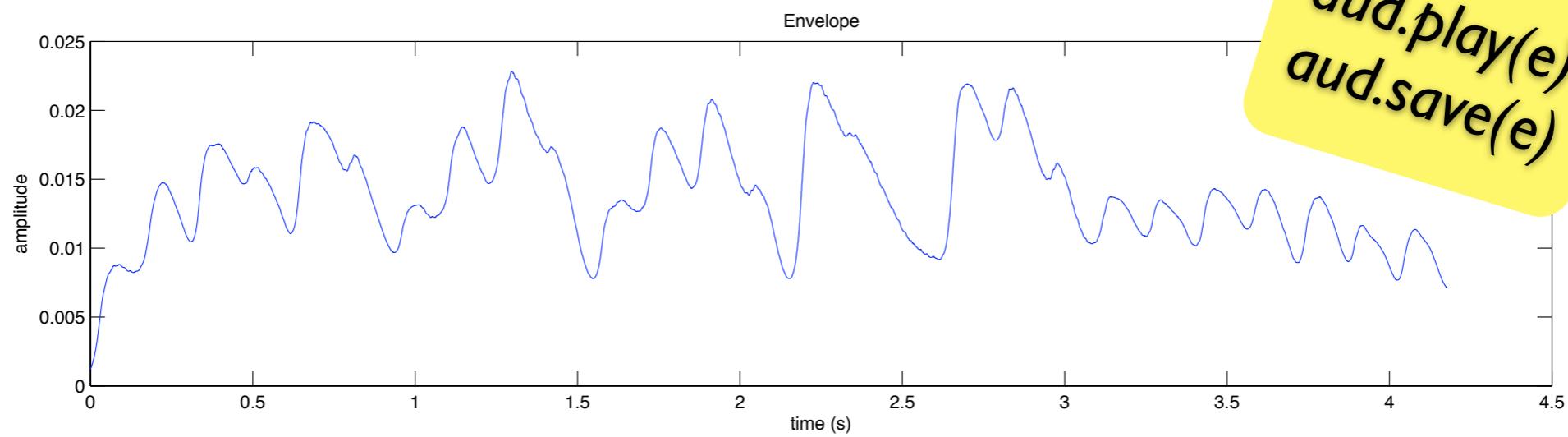
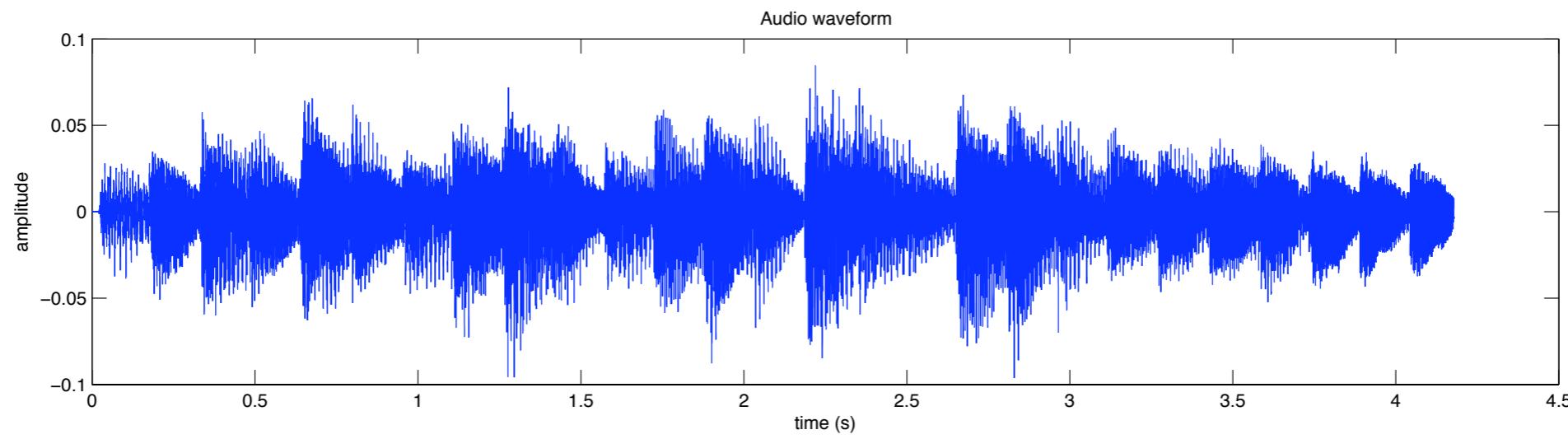
$$x_{\text{rms}} = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \dots + x_n^2}{n}}$$

- *sig.rms('ragtime')*
The RMS energy related to file `ragtime` is 0.017932
- *sig.rms('ragtime', 'Frame')*
Default frame size .05 s, frame hop = .5

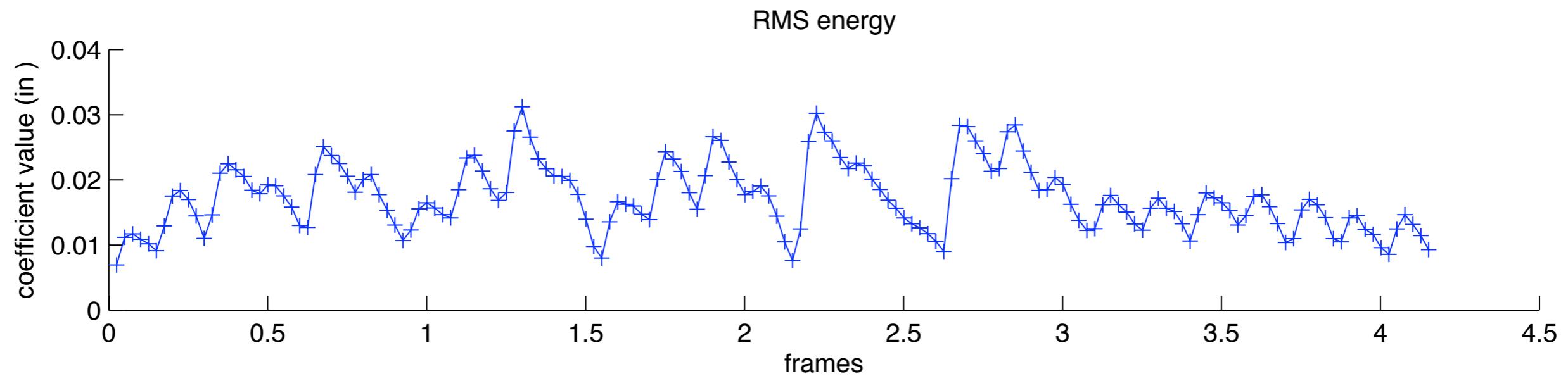


sig.envelope envelope extraction

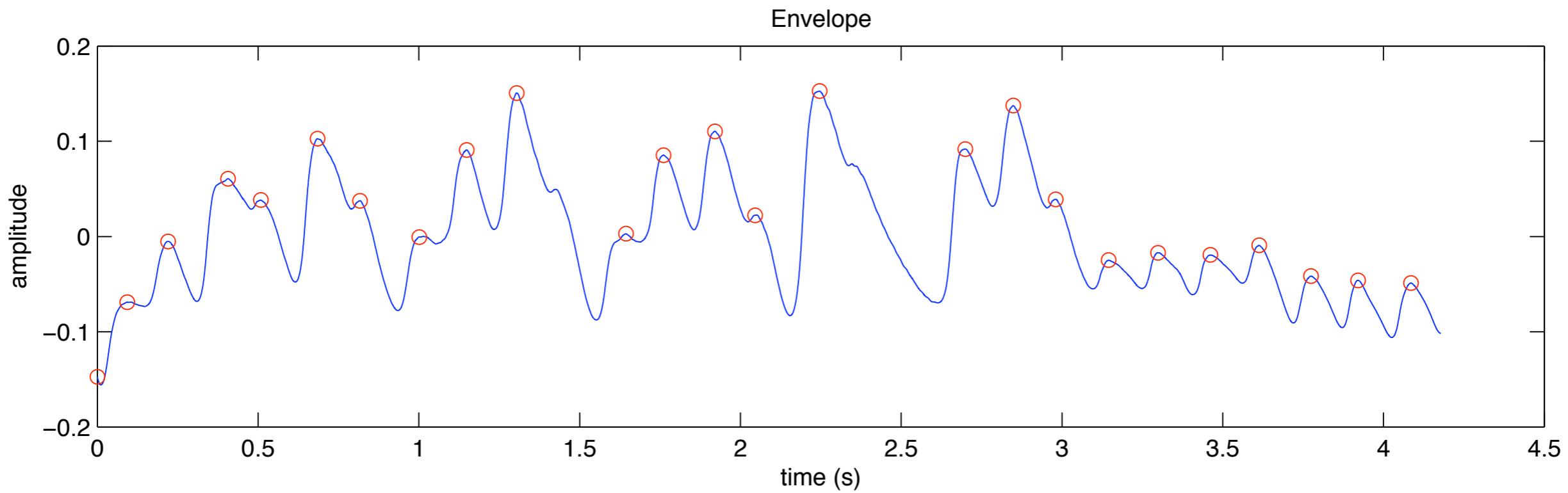
- a :
- $e = \text{sig.envelope}(a)$:



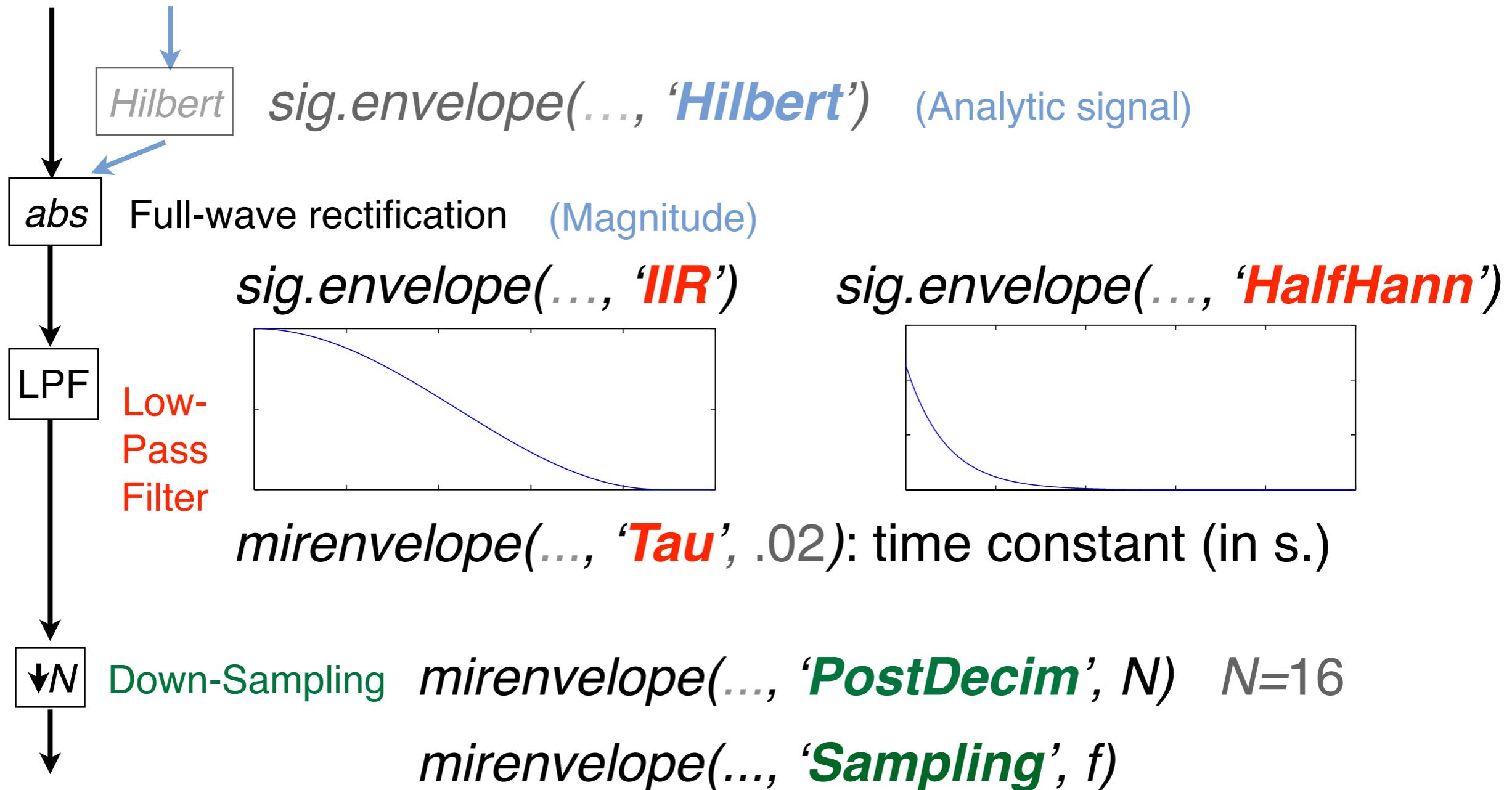
sig.rms



sig.envelope



`sig.envelope(..., 'Filter')`
based on low-pass filtering



sig.envelope(..., 'Spectro')
based on power spectrogram

- Band decomposition:
 - *sig.envelope(..., 'Freq')*: none (default)
 - *aud.envelope(..., 'Mel')*: Mel-band
 - *aud.envelope(..., 'Bark')*: Bark-band
- *sig.envelope(..., 'UpSample', 2)*
- *sig.envelope(..., 'Complex')*

sig.envelope post-processing

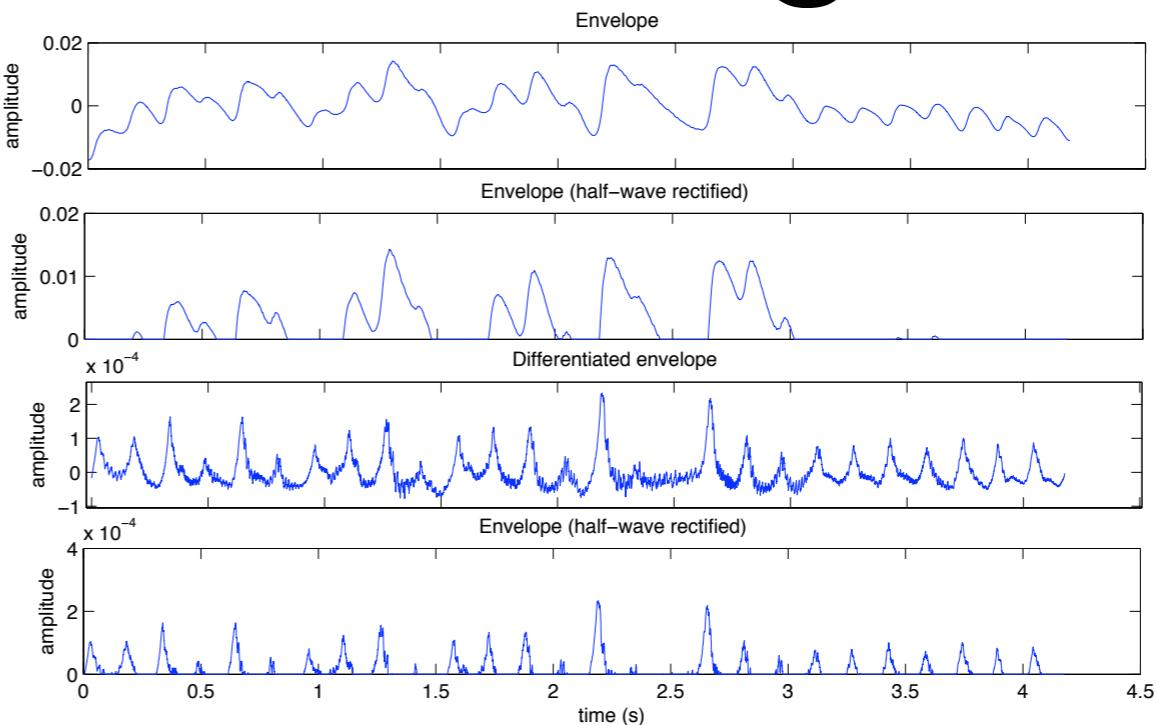
- *sig.envelope(..., 'Center')*

'HalfWaveCenter')

'Diff')

'HalfWaveDiff')

- *sig.envelope(..., 'Power')*
- *sig.envelope(..., 'Normal')*
- *sig.envelope(..., 'Smooth', o)* moving average, order $o = 30$ sp.
- *sig.envelope(..., 'Gauss', o)* gaussian, std deviation $o = 30$ sp.



aud.envelope

auditory modeling

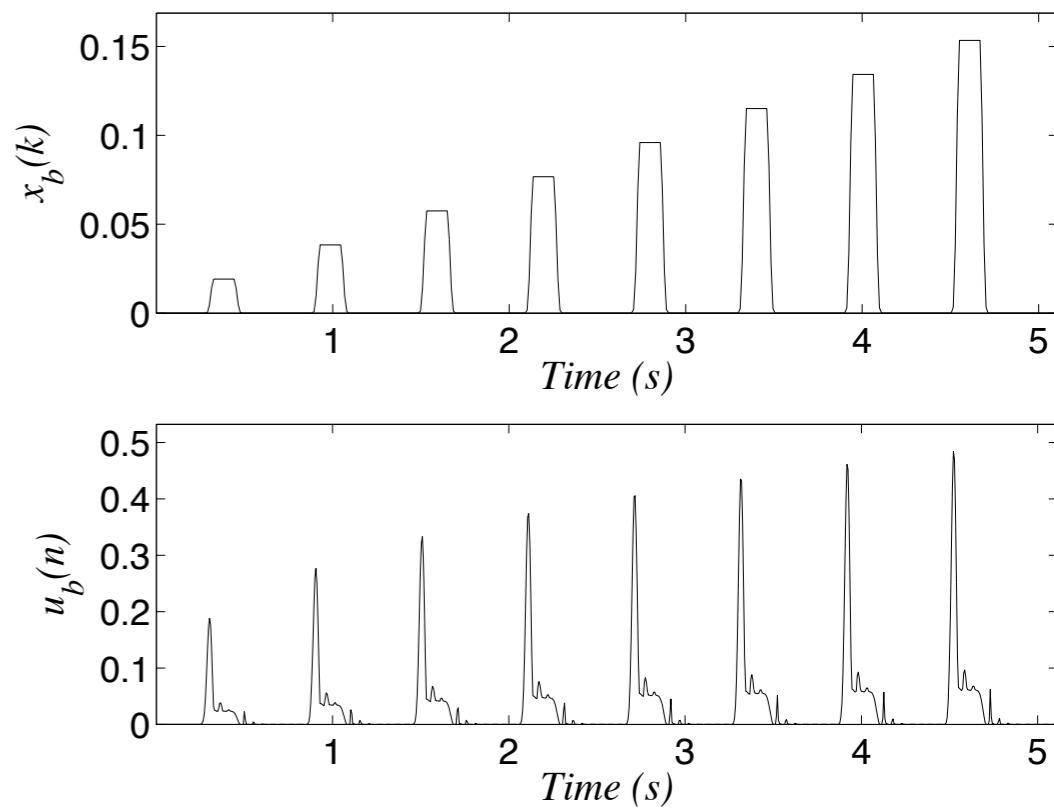
- *aud.envelope(..., 'Log', mu)*,
mu-law compression, mu = 100

$$y_b(k) = \frac{\ln(1 + \mu x_b(k))}{\ln(1 + \mu)}$$

- *aud.envelope(..., 'Lambda', l)* $u_b(n) = (1 - \lambda)z_b(n) + \lambda \frac{f_r}{f_{LP}} z_b'(n)$

- *aud.envelope(..., 'Klapuri06'):*

- *e = aud.envelope(..., 'Spectro',
'UpSample', 'Log',
'HalfwaveDiff', 'Lambda', .8);*
- *sig.sum(e, 'Adjacent', 10)*

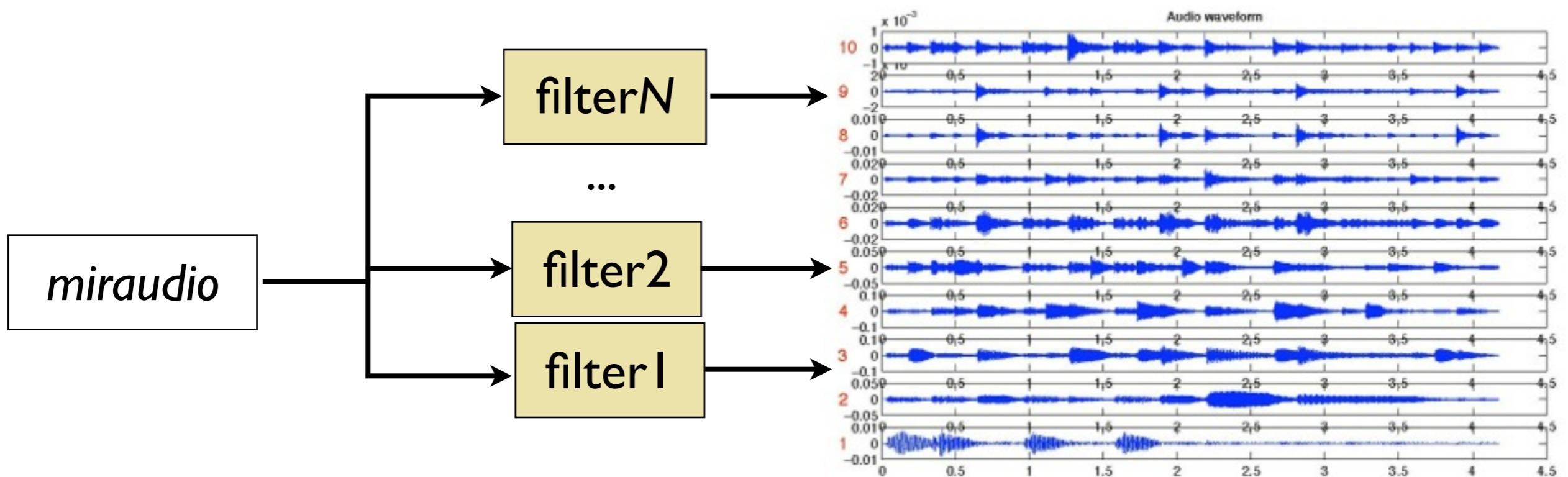


Klapuri, A., A. Eronen and J. Astola. (2006). “Analysis of the meter of acoustic musical signals”, IEEE Transactions on Audio, Speech and Language Processing, 14-1, 342– 355.

sig.filterbank

filterbank decomposition

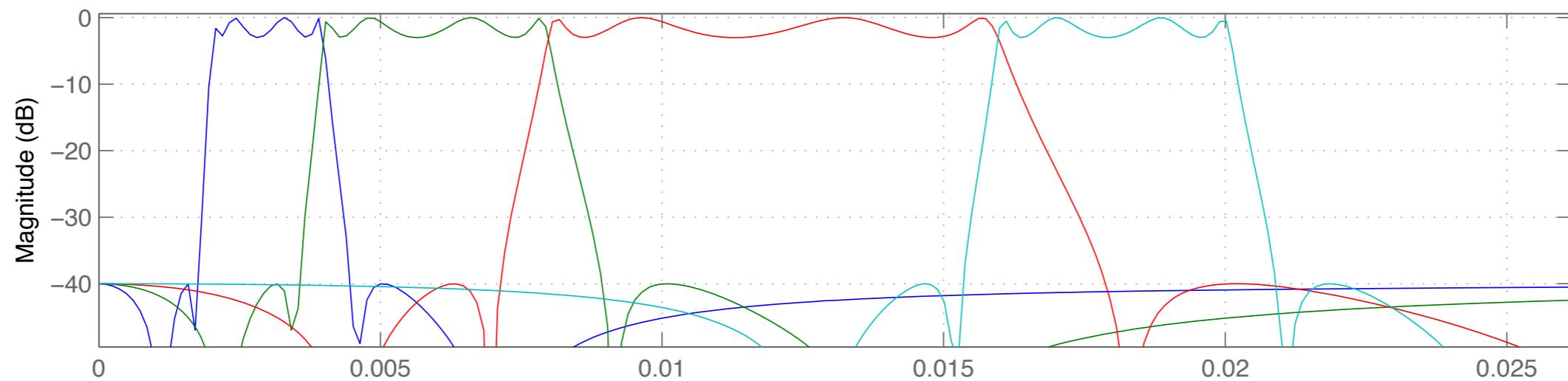
- `f = sig.filterbank(..., 'CutOff', [500, 1000])`
- `aud.play(f)`
- `aud.save(f)`



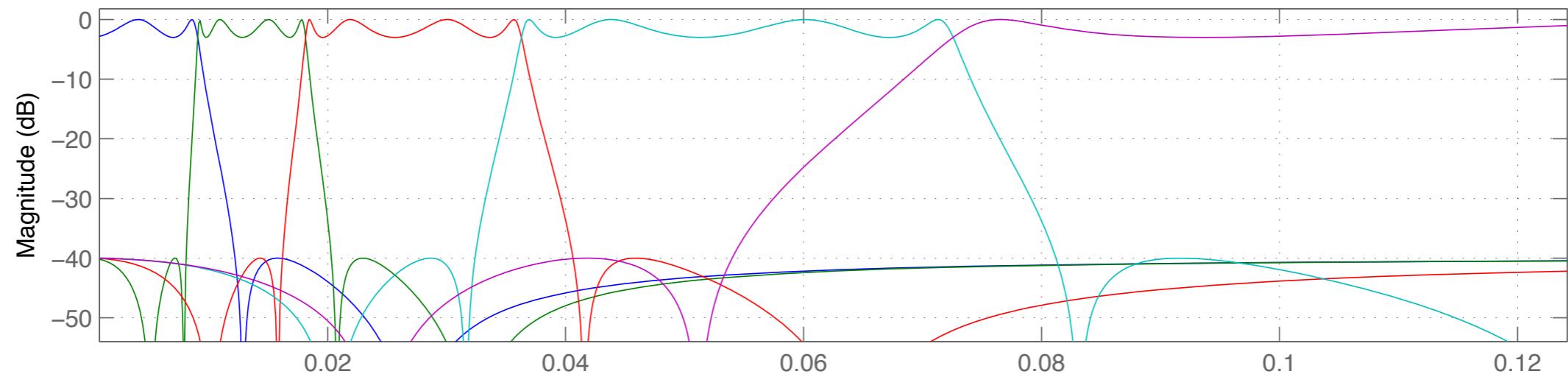
sig.filterbank

filterbank decomposition

`sig.filterbank(..., 'CutOff', [44, 88, 176, 352, 443])`



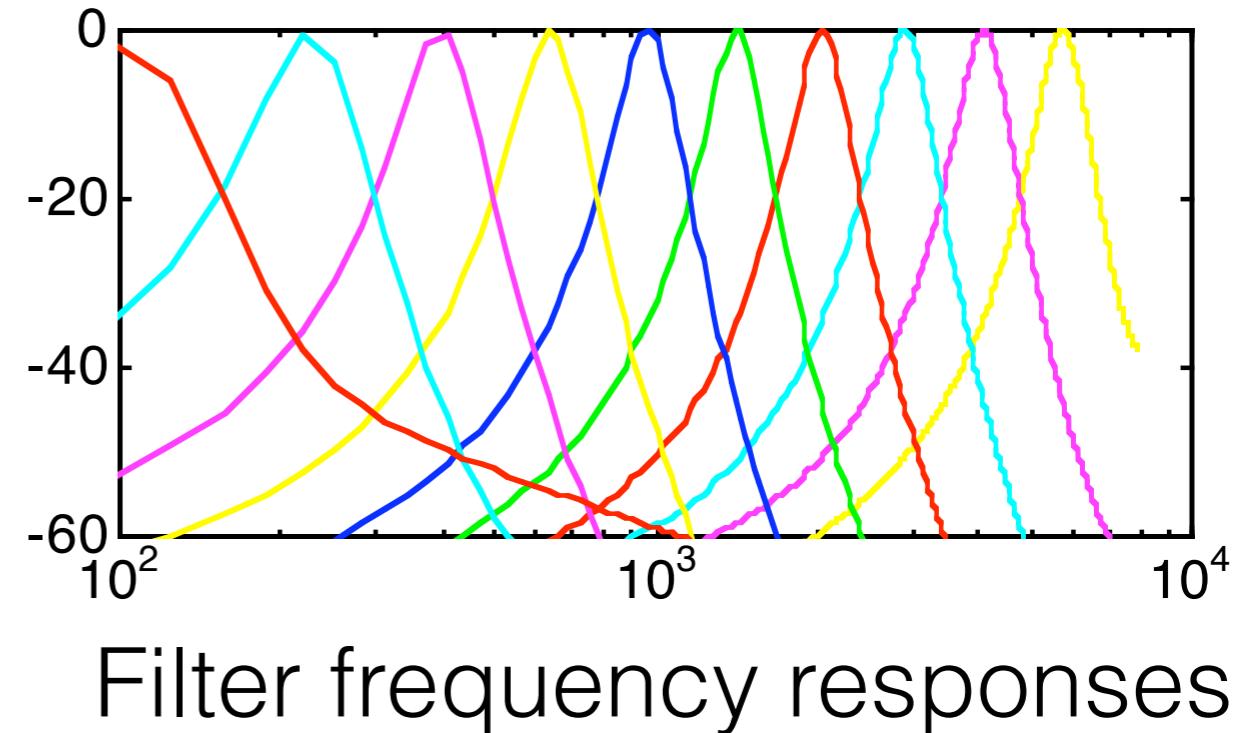
`sig.filterbank(..., 'CutOff', [-Inf 200 400 800 1600 Inf])`



aud.filterbank

auditory modeling

- *aud.filterbank(...)*
- Equivalent Rectangular Bandwidth (ERB)
Gammatone filterbank
- *aud.filterbank(..., 'NbChannels', 10)*
- *aud.filterbank(..., 'Channel', 1:10)*

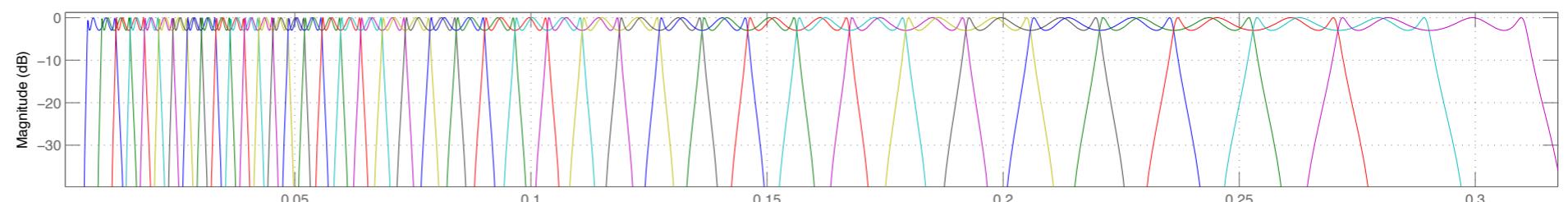


R. D. Patterson et al. “Complex sounds and auditory images,” in Auditory Physiology and Perception, Y. Cazals et al, Oxford, 1992, pp. 429-446

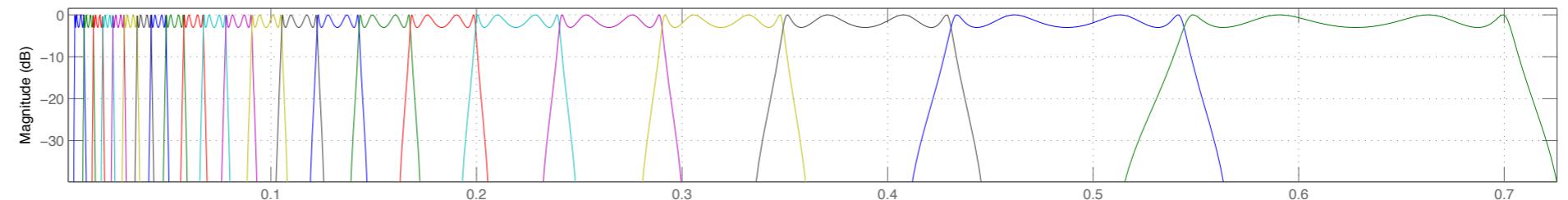
aud.filterbank

auditory modeling

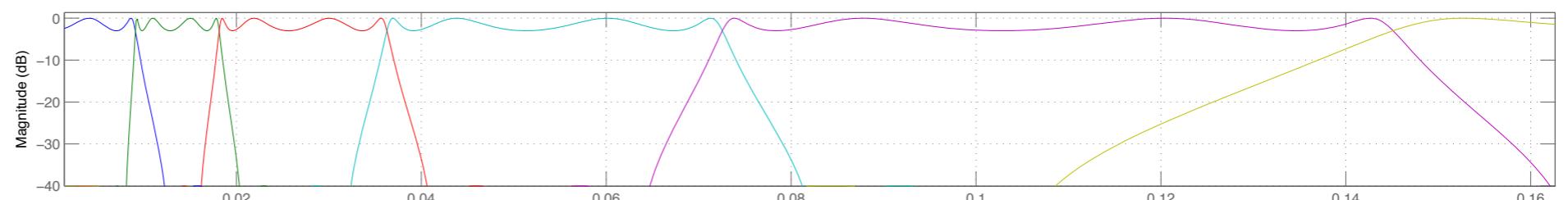
- *aud.filterbank(..., 'Mel')*



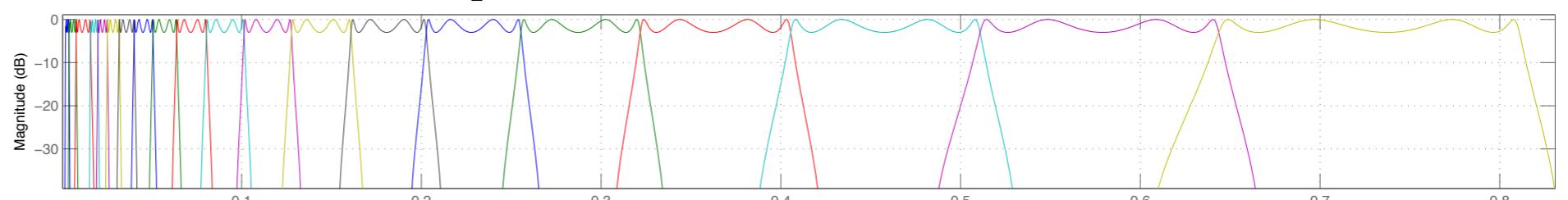
- *aud.filterbank(..., 'Bark')*



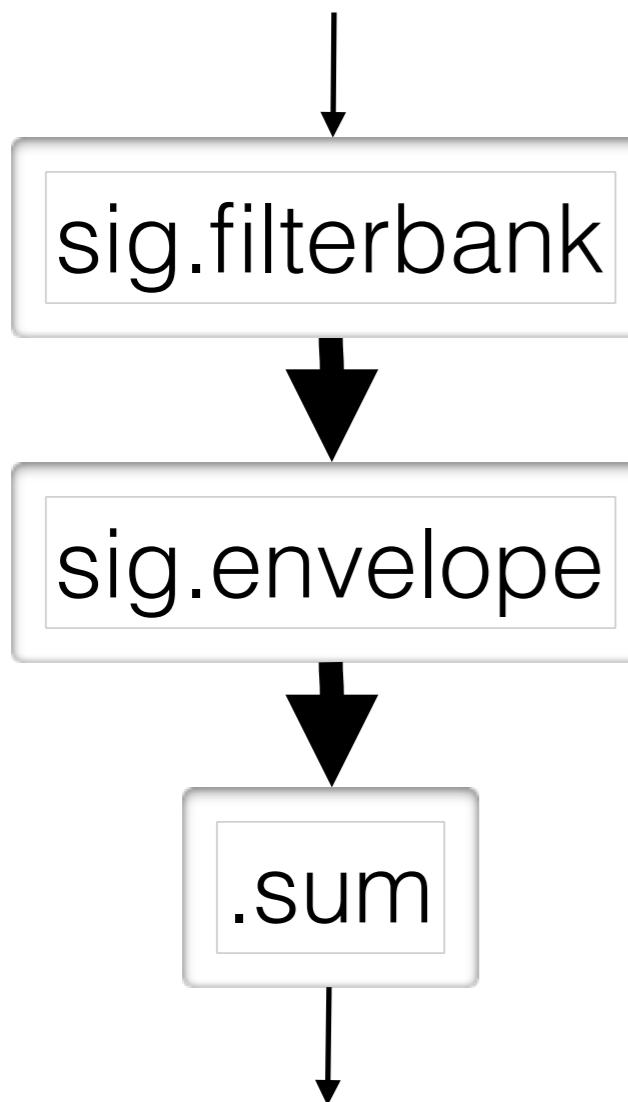
- *aud.filterbank(..., 'Scheirer')*



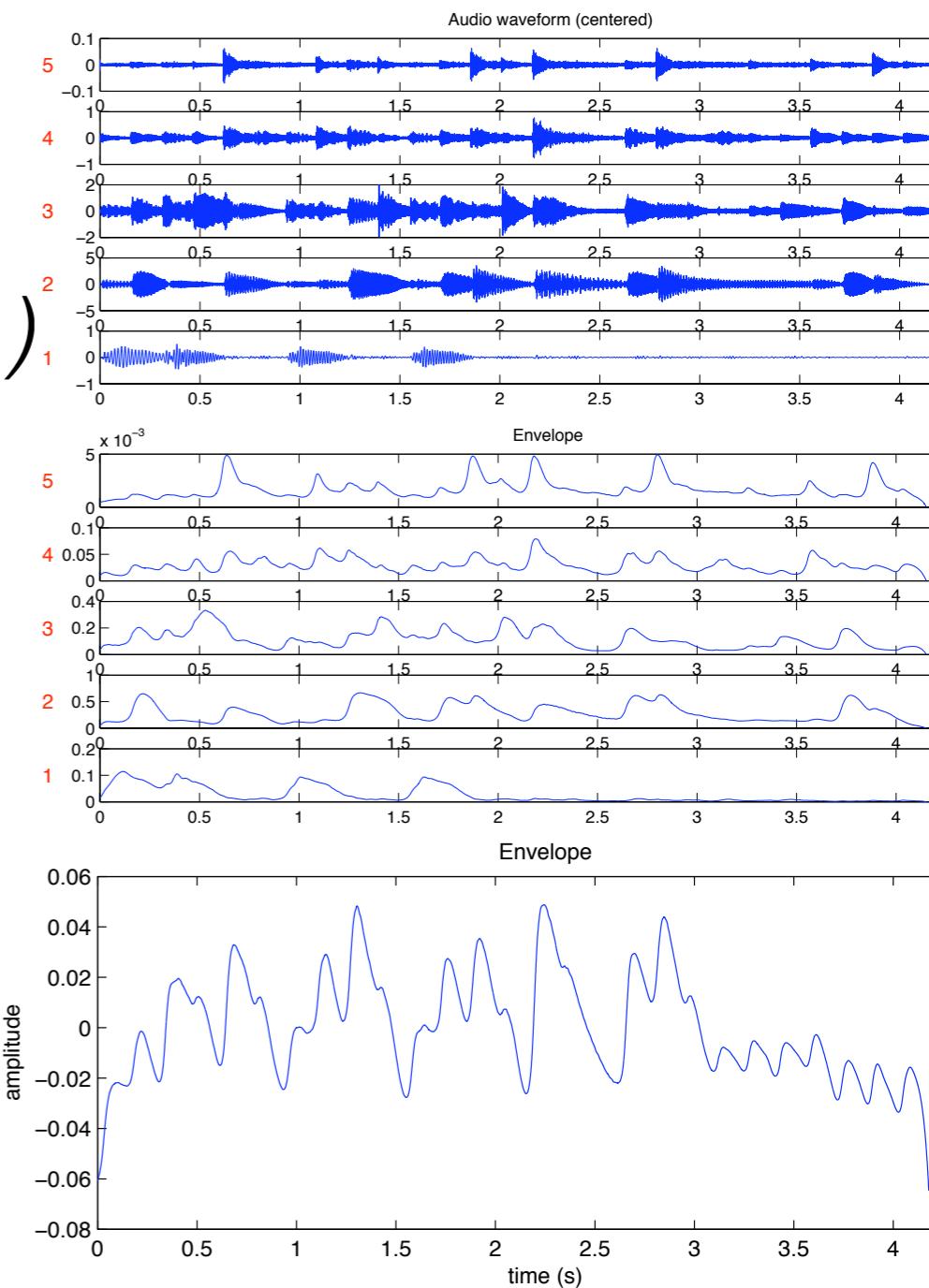
- *aud.filterbank(..., 'Klapuri')*



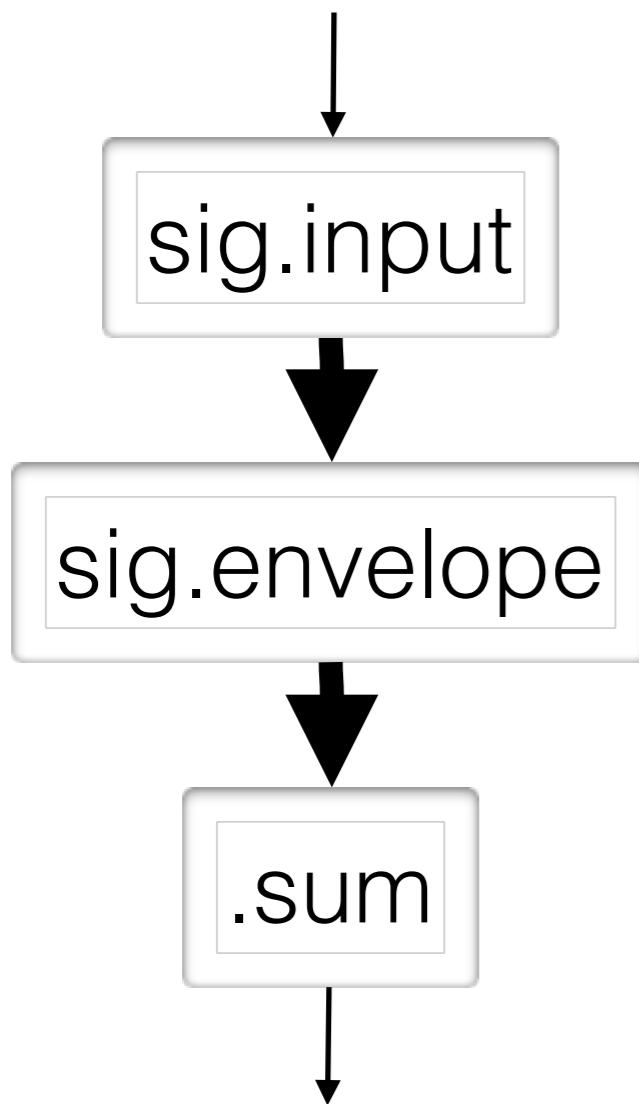
.sum across-channel summation



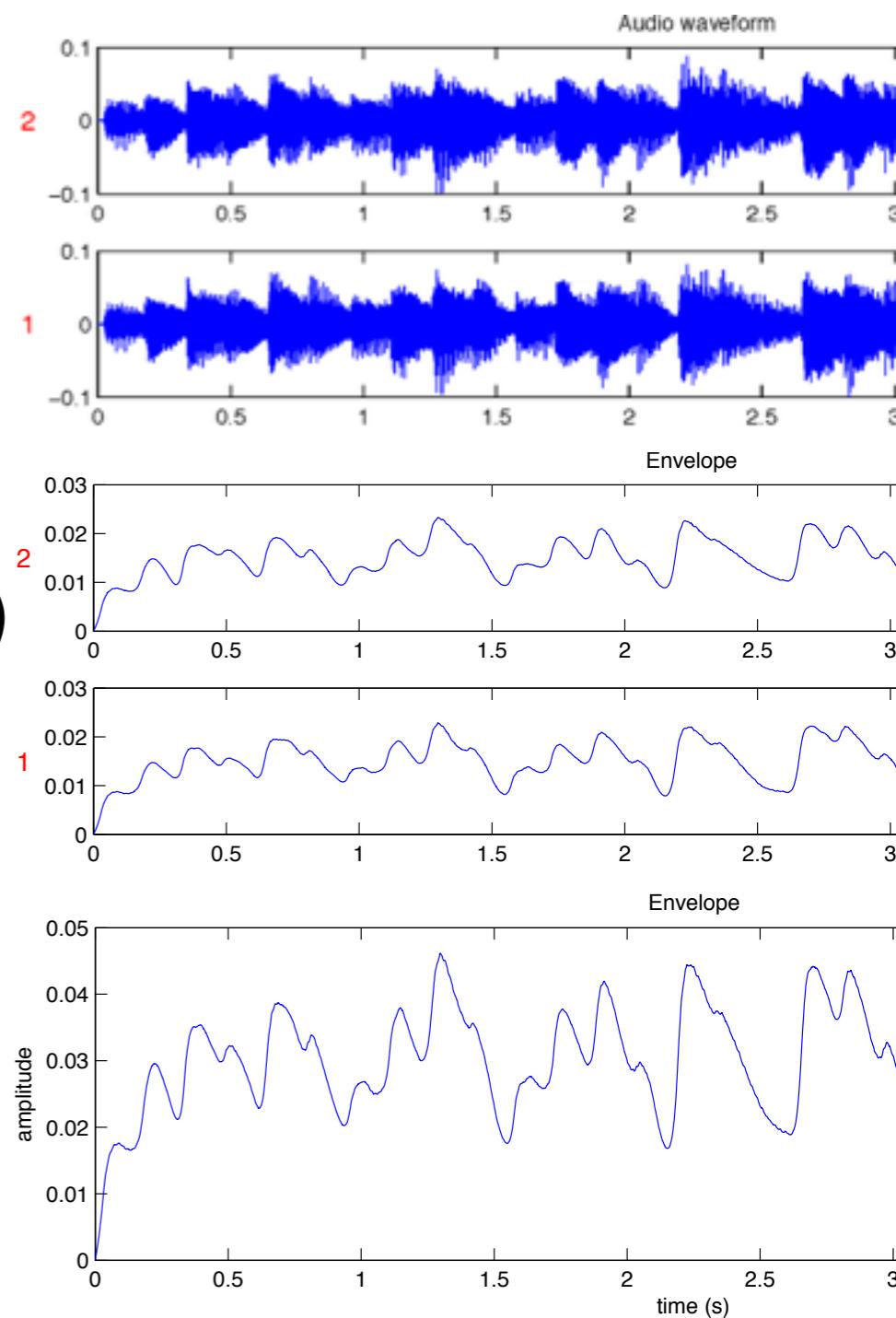
- $f = \text{sig.filterbank}(\dots)$
- $e = \text{sig.envelope}(f)$
- $e.sum$



.sum stereo summation



- $a = \text{sig.input}(\dots, \text{'Mix'}, \text{'No'})$
- $e = \text{sig.envelope}(a)$
- $e.\text{sum}(\text{'channel'})$



.sum across-channel summary

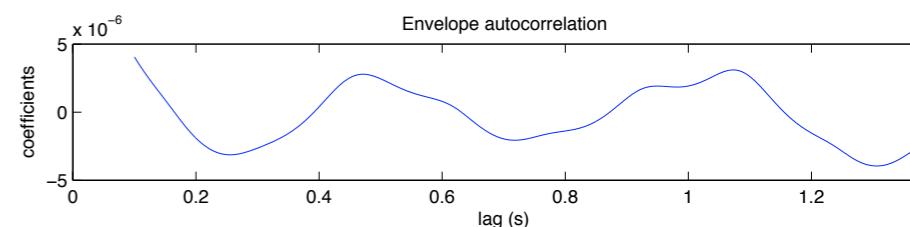
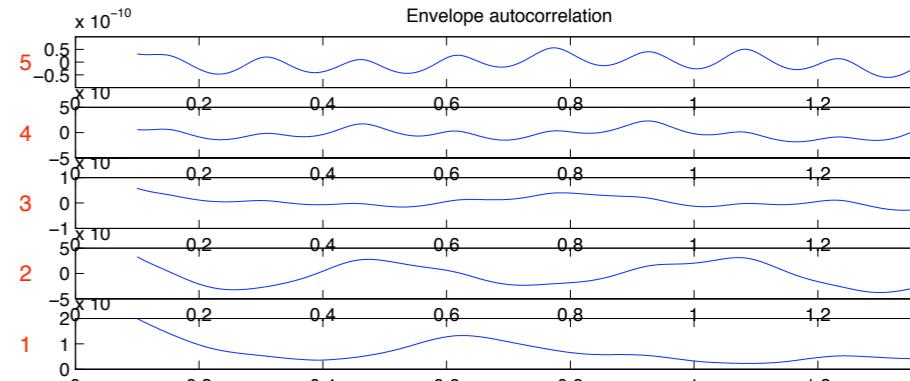
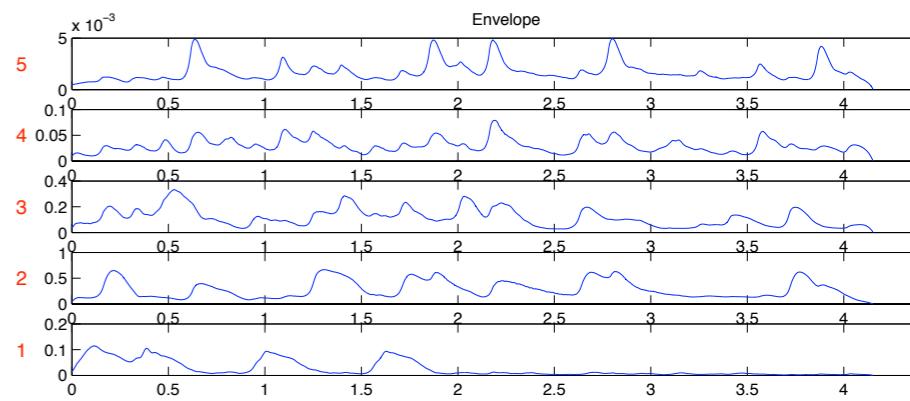
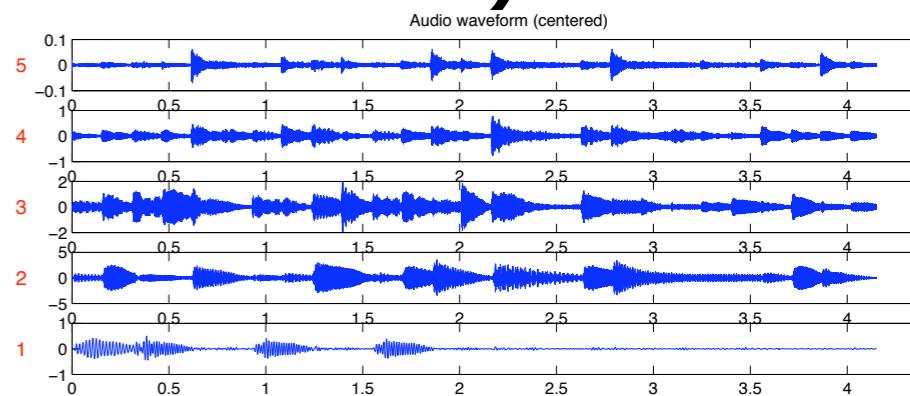
↓
sig.filterbank

↓
sig.envelope

↓
sig.autocor

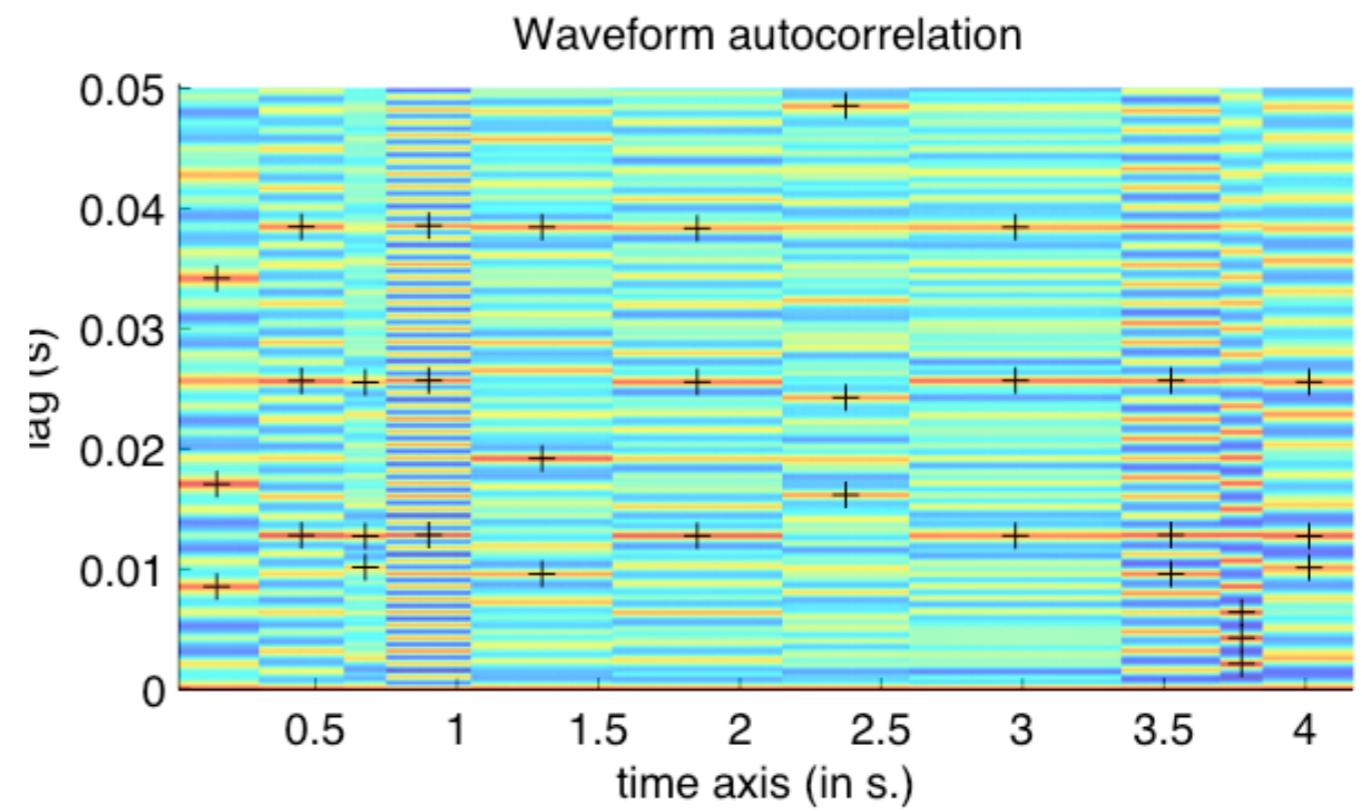
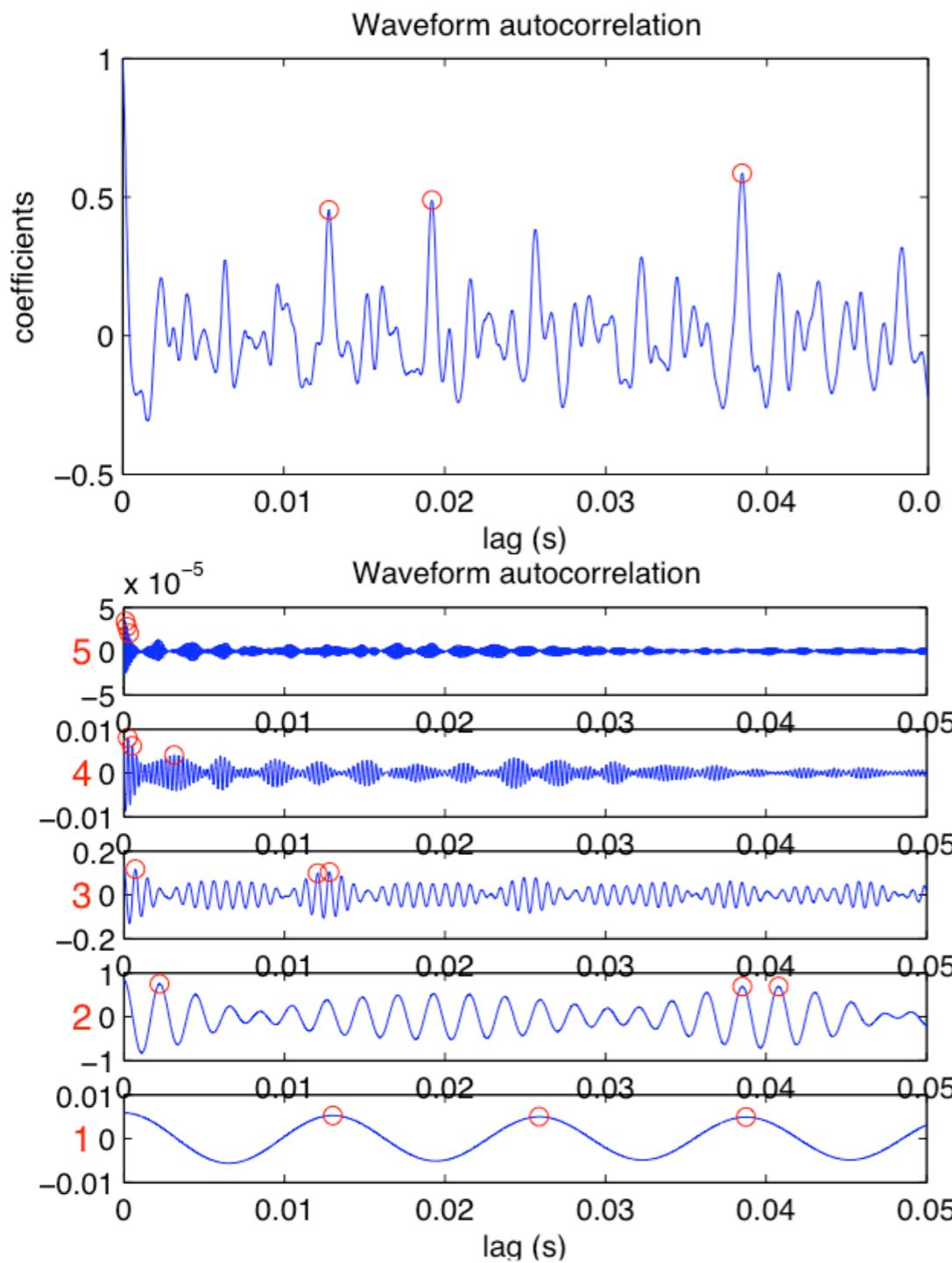
↓
.sum

- $f = \text{sig.filterbank}(\dots)$
- $e = \text{sig.envelope}(f)$
- $a = \text{sig.autocor}(e)$
- $a.sum$



sig.peaks

peak picking

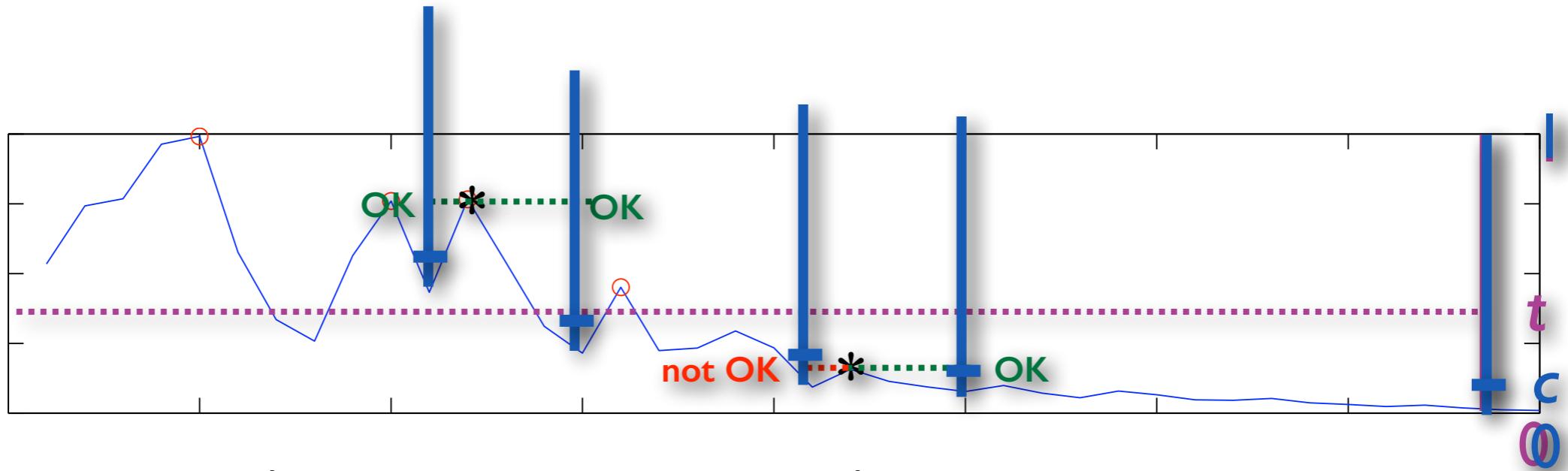


sig.peaks

peak picking

- *sig.peak(..., ‘Total’, Inf)* Number of peaks
- Border effects:
 - *sig.peak(..., ‘NoBegin’)* First sample excluded
 - *sig.peak(..., ‘NoEnd’)* Last sample excluded
- *sig.peak(..., ‘Order’, o)* Ordering of peaks
 - o = ‘**Amplitude**’ From highest to lowest
 - o = ‘**Abscissa**’ Along the abscissa axis
- *sig.peak(..., ‘Valleys’)* Local minima

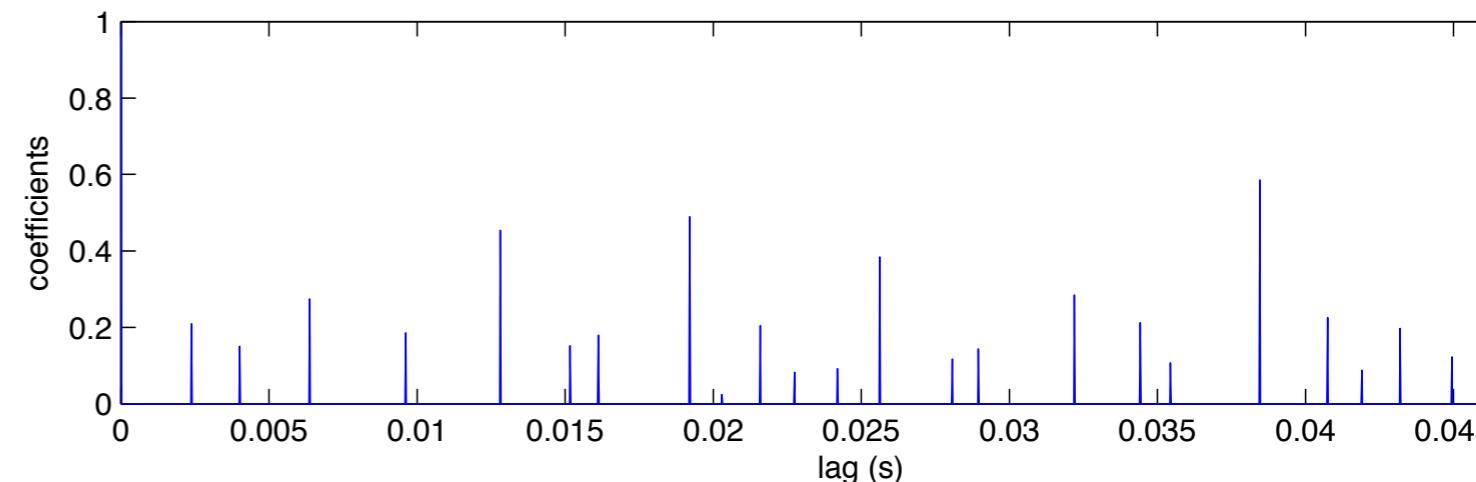
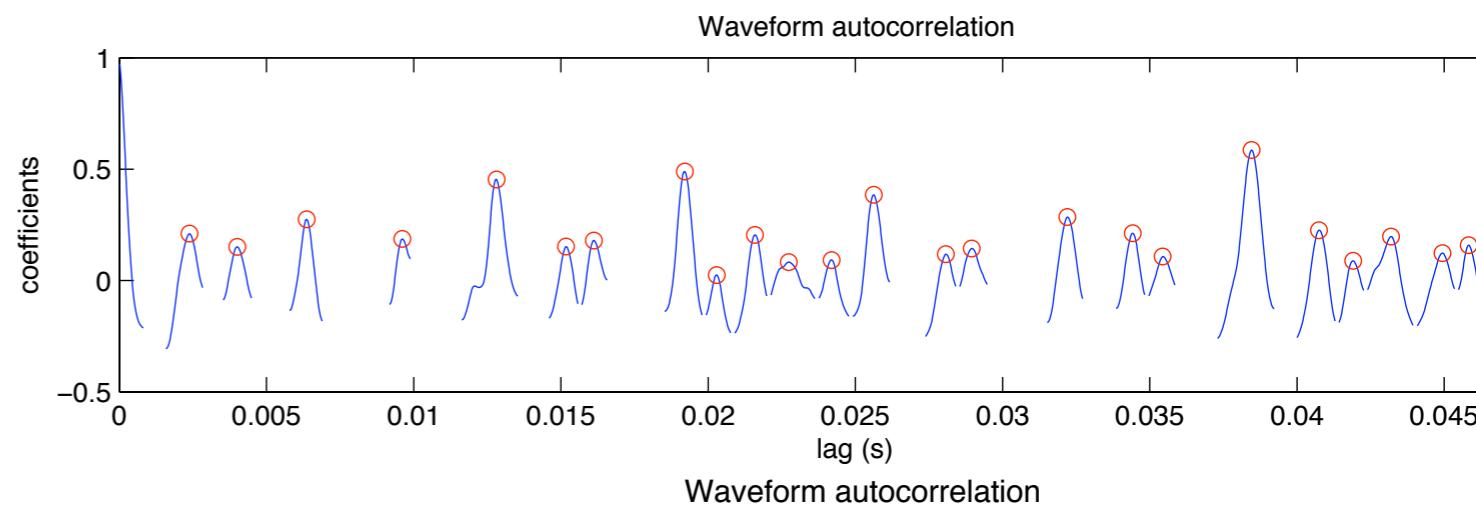
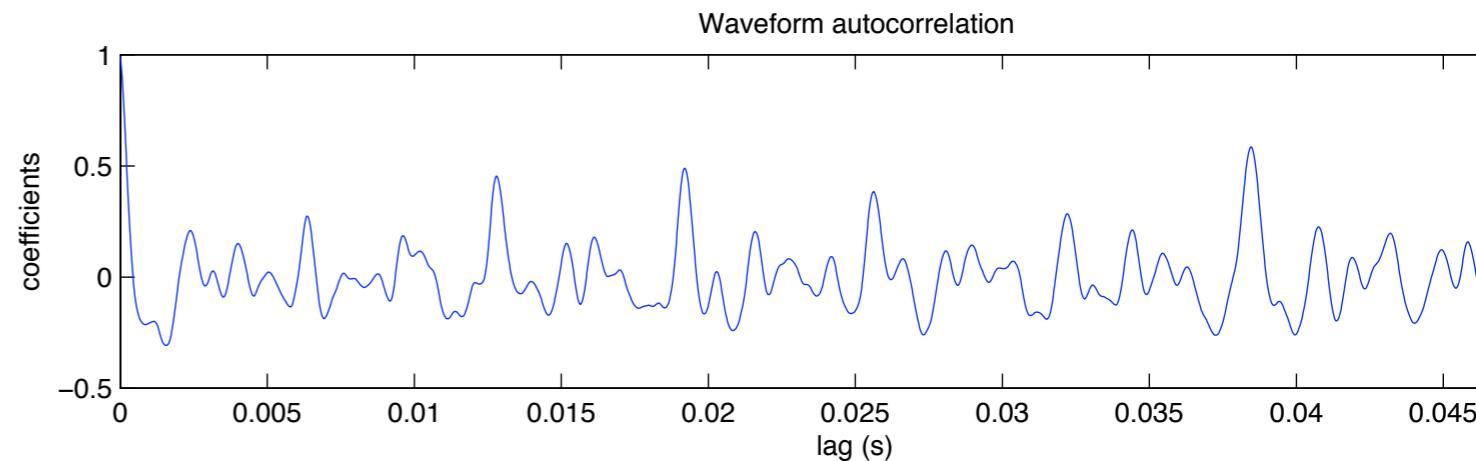
sig.peaks peak picking



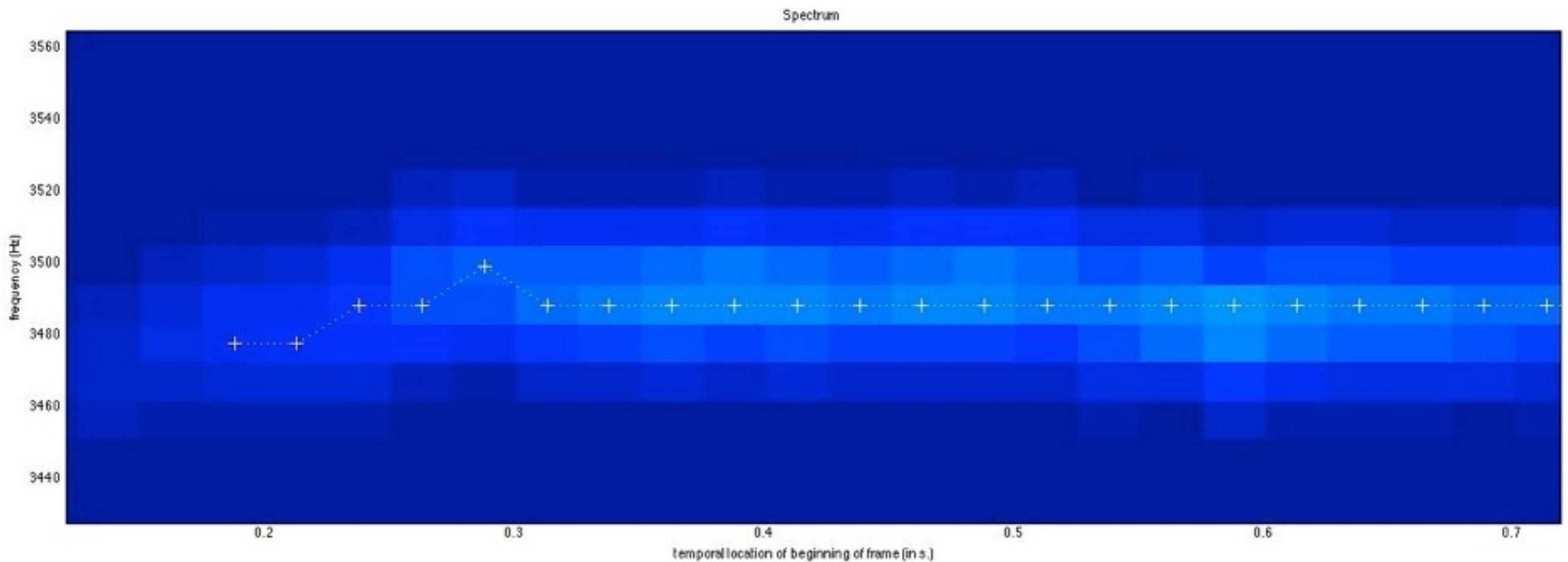
- *sig.peak(..., 'Threshold', 0)*
- *sig.peak(..., 'Contrast', .1)*
- *sig.peak(..., 'SelectFirst', .05)*
- *mus.peak(..., 'Reso', 'SemiTone')*

sig.peaks peak picking

- *sig.peak(...)*
- *sig.peak(..., 'Extract')*
- *sig.peak(..., 'Only')*



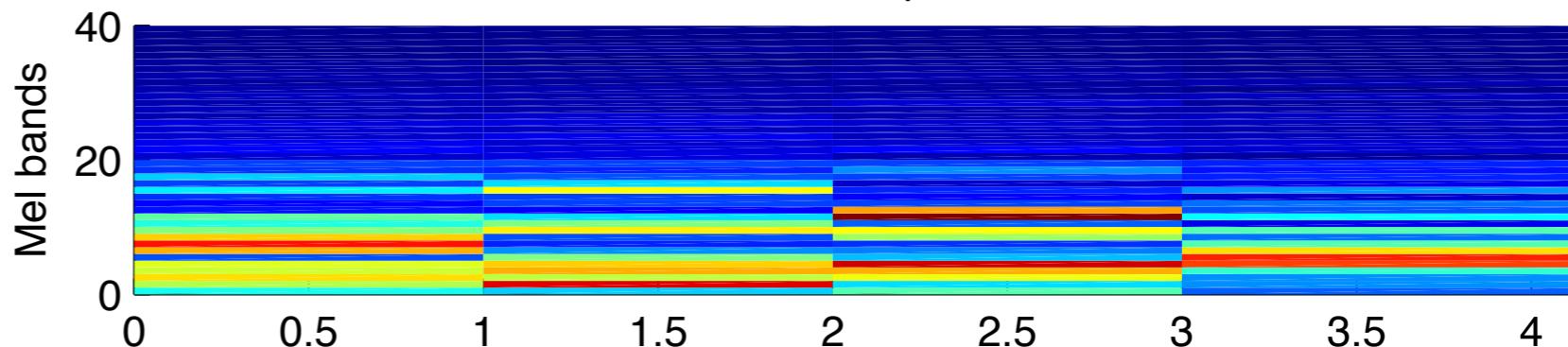
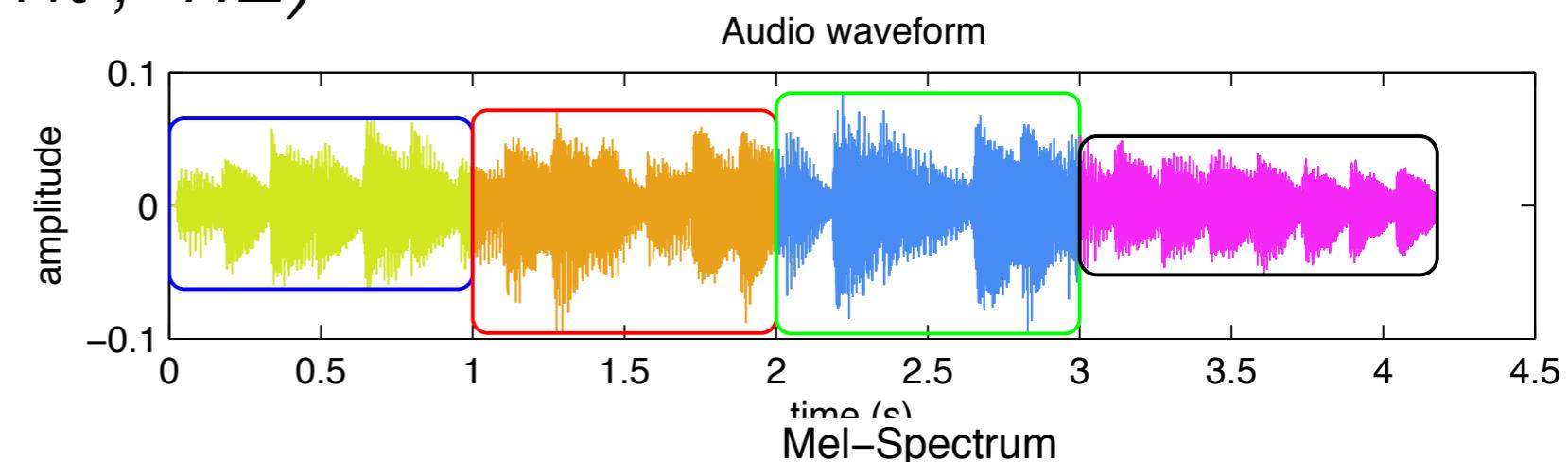
`sig.peaks(..., 'Track')` peak tracking



McAulay, R.; Quatieri, T. (1996). “Speech analysis/Synthesis based on a sinusoidal representation”, IEEE Transactions on Acoustics, Speech and Signal Processing, 34:4, 744–754.

sig.segment segmentation

- `s = sig.segment('myfile', [1 2 3])`
- `aud.play(s, 'Segment', 1:2)`
- `aud.save(s)`
- `sig.spectrum(s)`



sig.segment segmentation

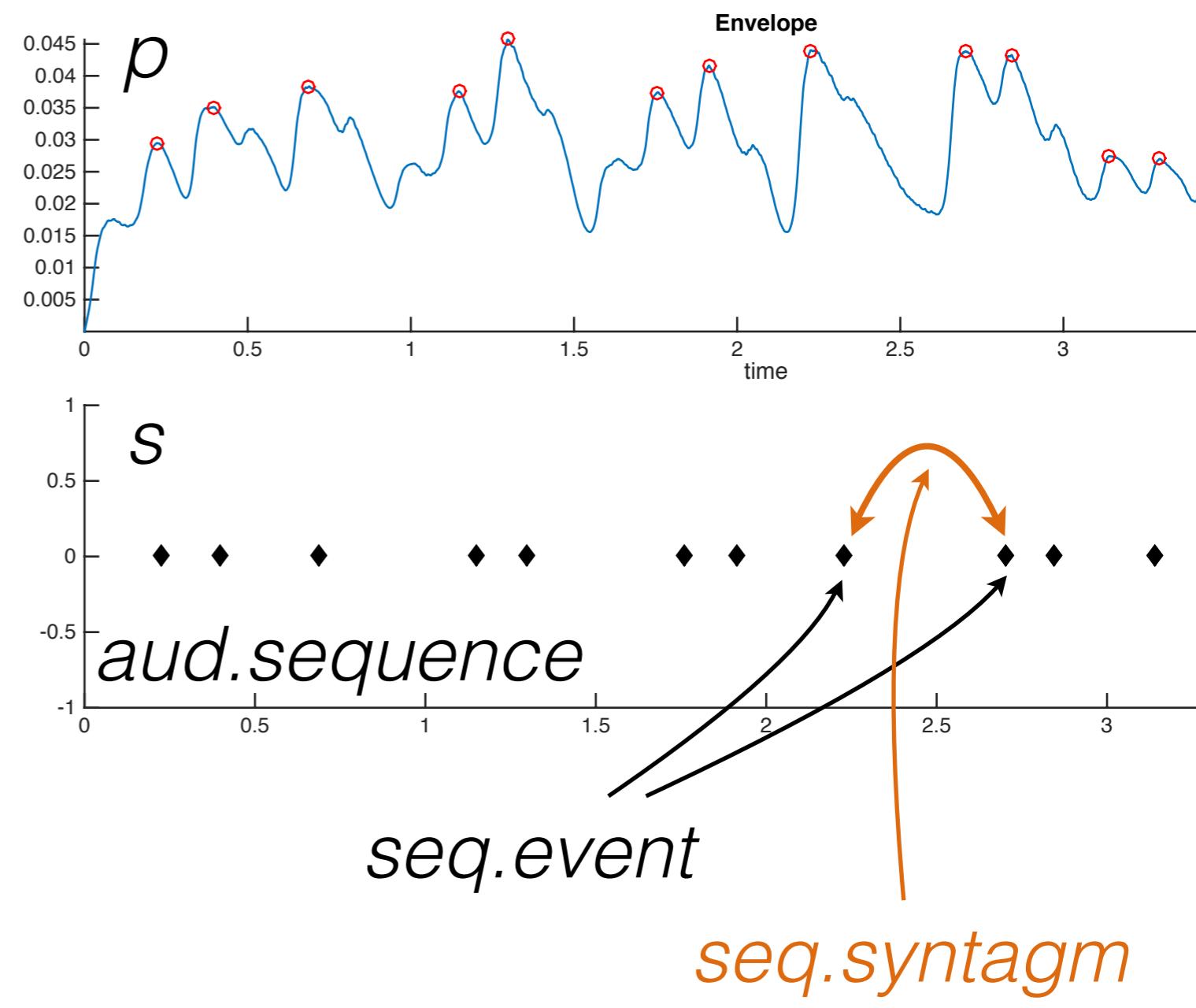
- $e = \text{sig.envelope}(\text{'myfile'})$
- $p = \text{sig.peaks}(e)$
- $s = \text{sig.segment}(\text{'myfile'}, p)$

aud.score onset detection

- $e = \text{sig.envelope}(\text{'myfile'})$
- $p = \text{sig.peaks}(e, \text{'Contrast'}, .1)$
- $s = \text{aud.score}(p)$

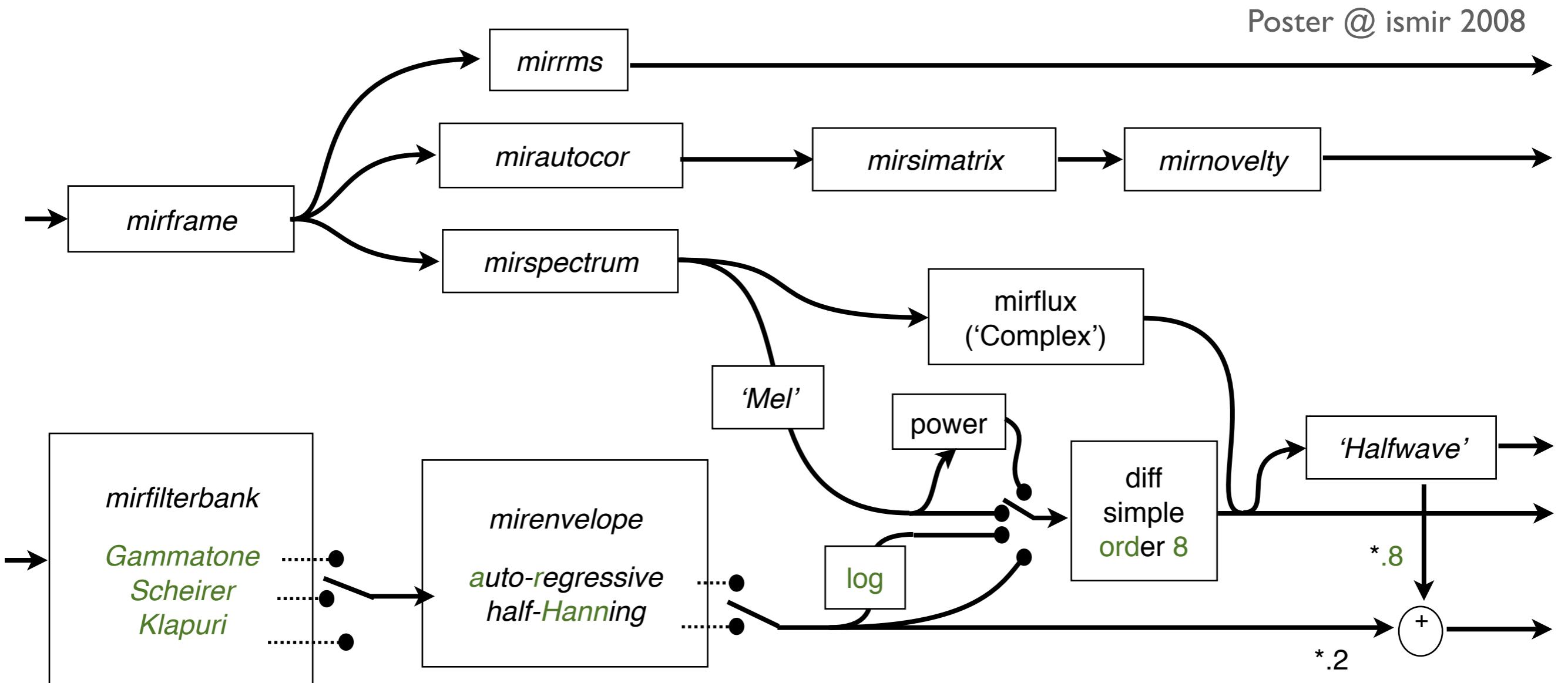
In short:

- $[s p] = \text{aud.score}(\text{'myfile'}, \textbf{Contrast}, .1)$



mironsets

onset detection

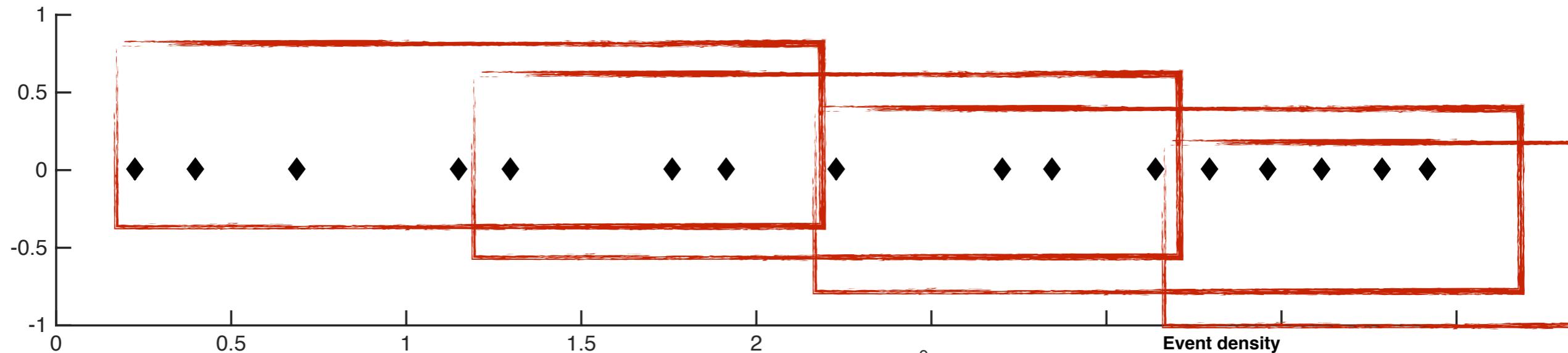


aud.score(..., 'Scheirer')

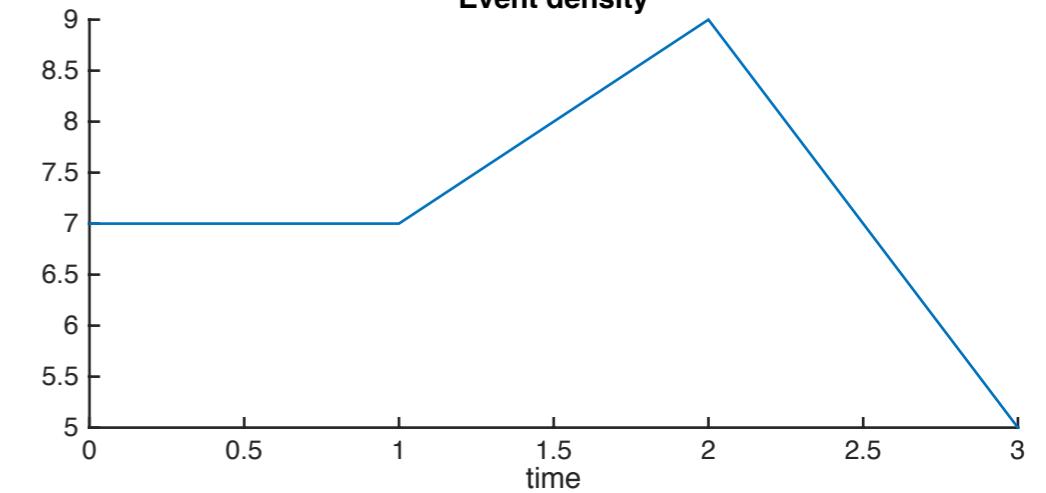
aud.score(..., 'Klapuri')

aud.eventdensity

temporal density of notes



- `a = aud.score('audio.wav')`
- `aud.eventdensity(a)`
- `aud.eventdensity('audio.wav', 'Frame')`
- `aud.eventdensity('score.mid')`



Audio level

Timbre

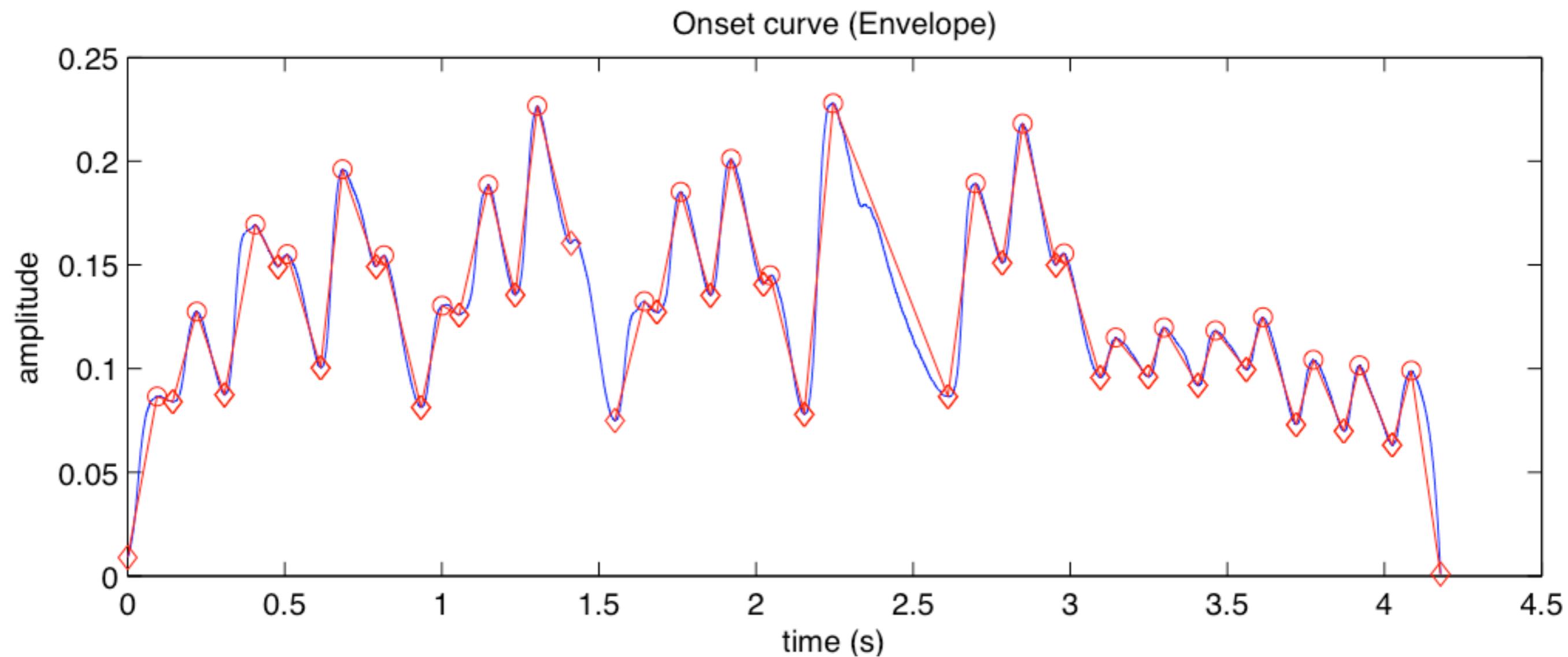


Sound

Dynamics

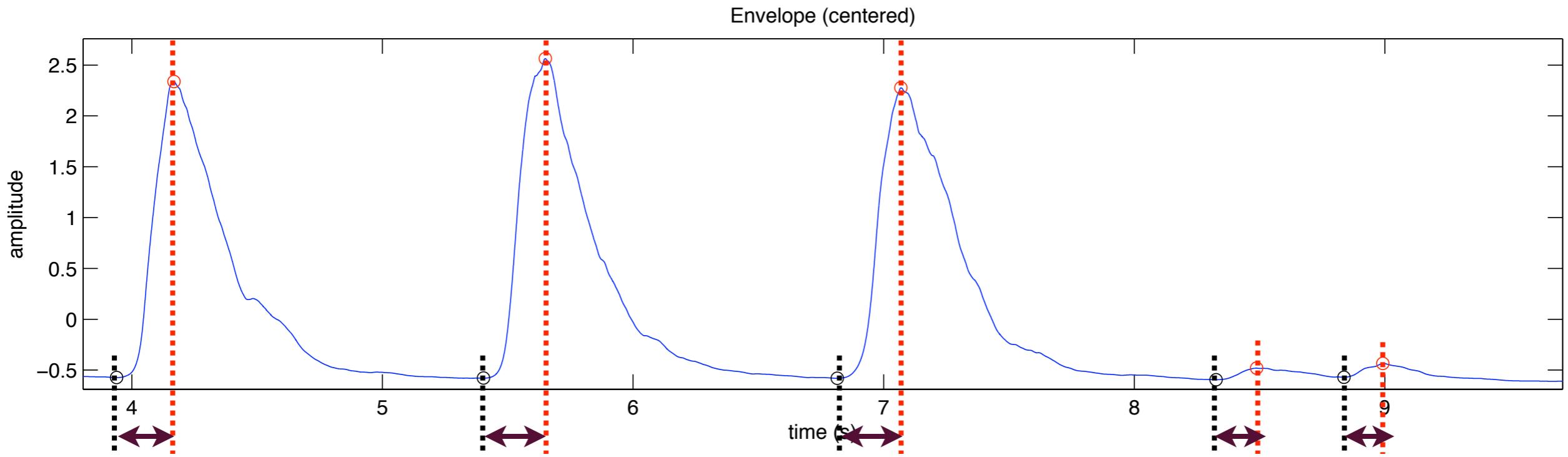


*aud.score(...,
'Attack', 'Release')*



aud.attacktime

duration of note attacks

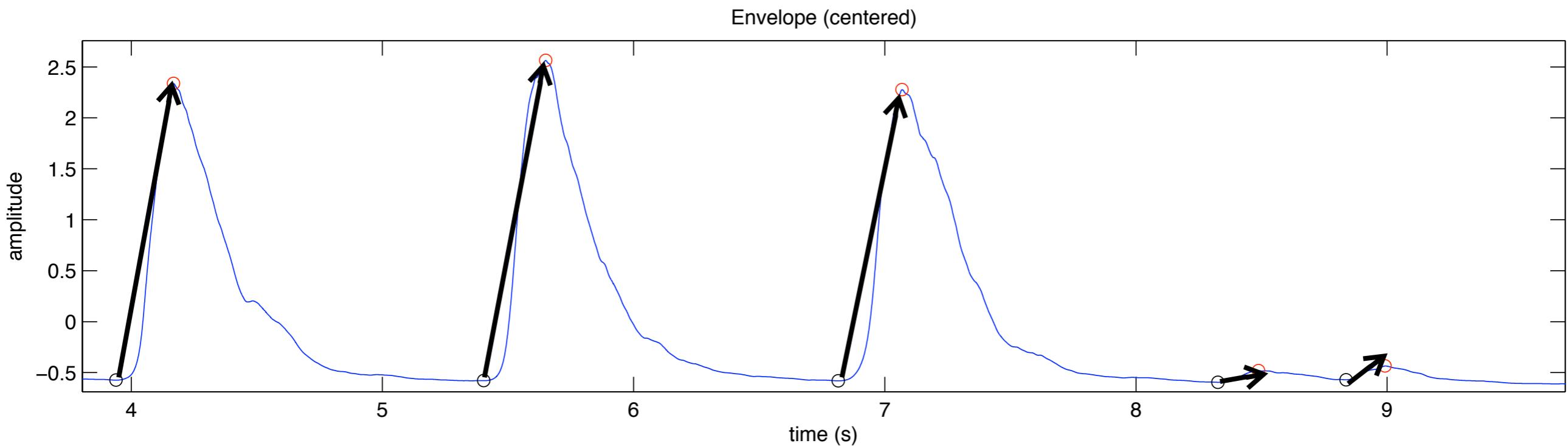


- *aud.attacktime(..., ‘Lin’)*: duration in seconds
- *aud.attacktime(..., ‘Log’)*: duration in log scale

Krimphoff, J., McAdams, S. & Winsberg, S. (1994), Caractérisation du timbre des sons complexes. II : Analyses acoustiques et quantification psychophysique. Journal de Physique, 4(C5), 625-628.

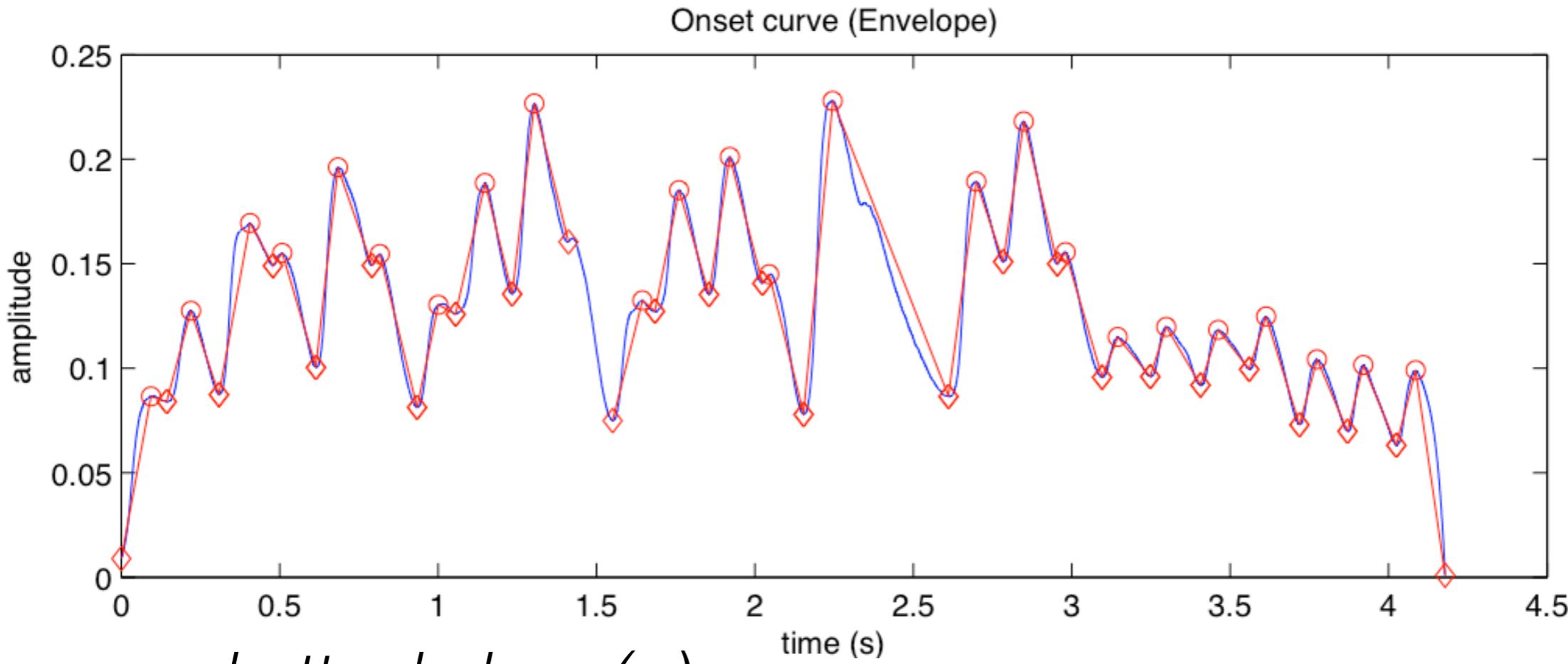
aud.attackslope

average slope of note attacks

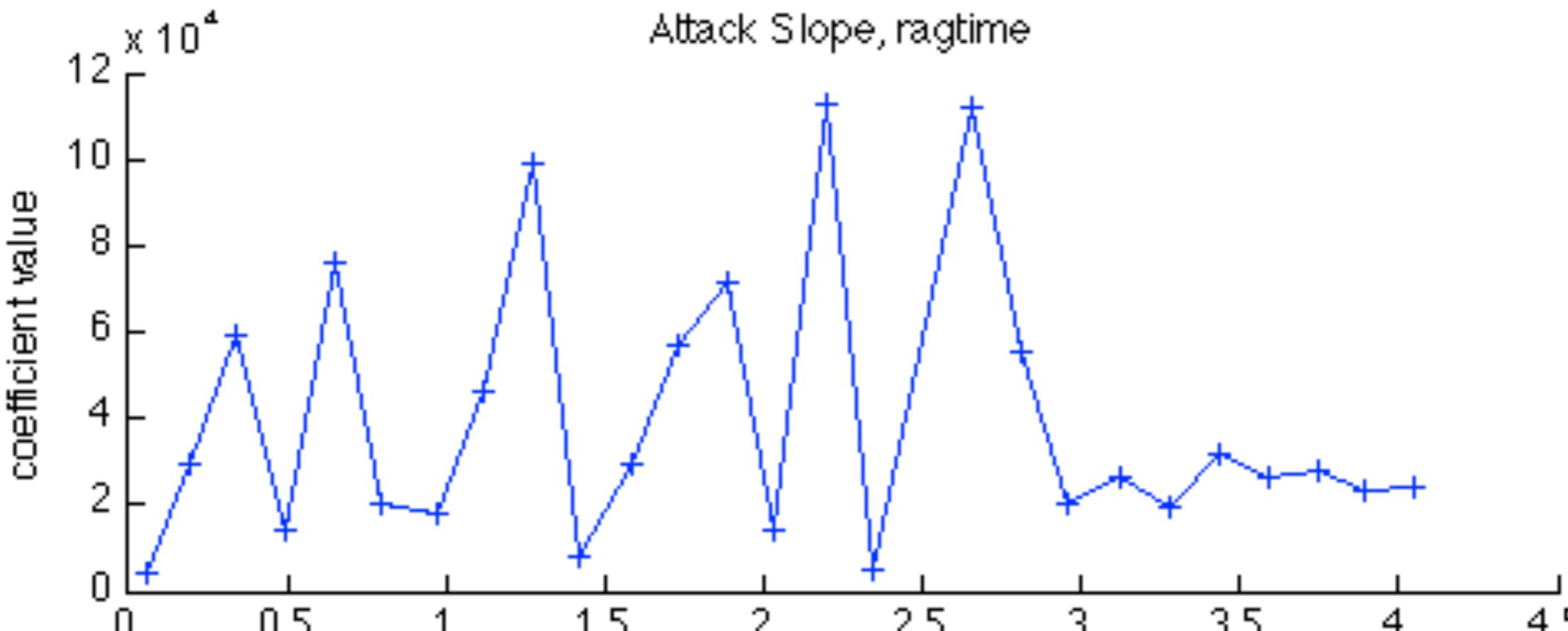


- *mirattackslope(..., 'Diff')*: average slope
- *mirattackslope(..., 'Gauss')*: gaussian average, highlighting the middle of the attack phase

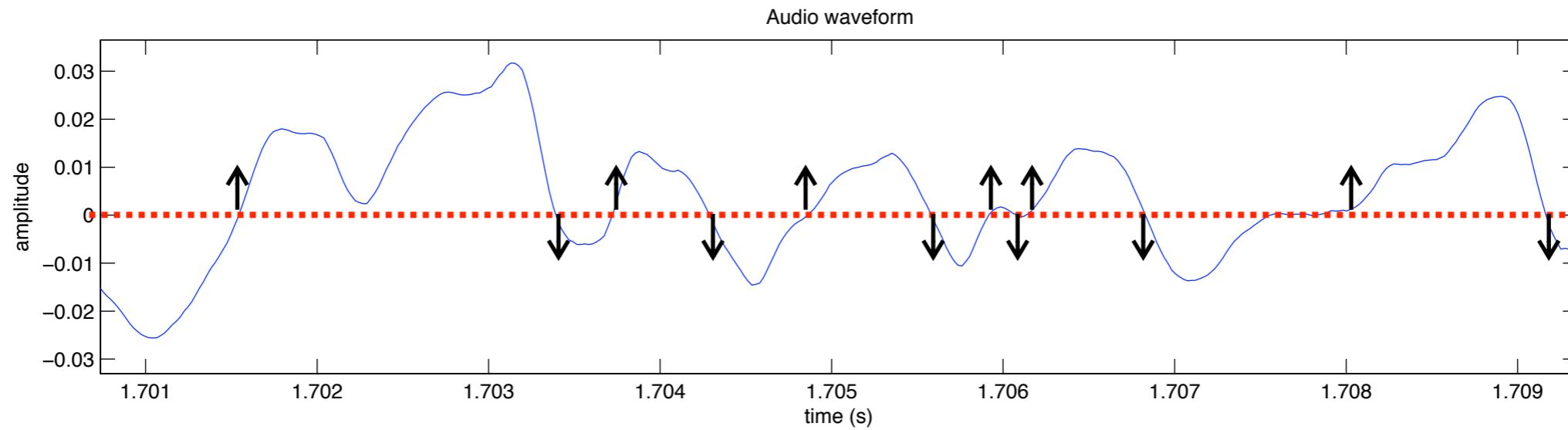
- $o = \text{aud.onsets}(\text{'audiofile'}, \text{'Attack'}, \text{'Release'})$



- $\text{aud.attackslope}(o)$

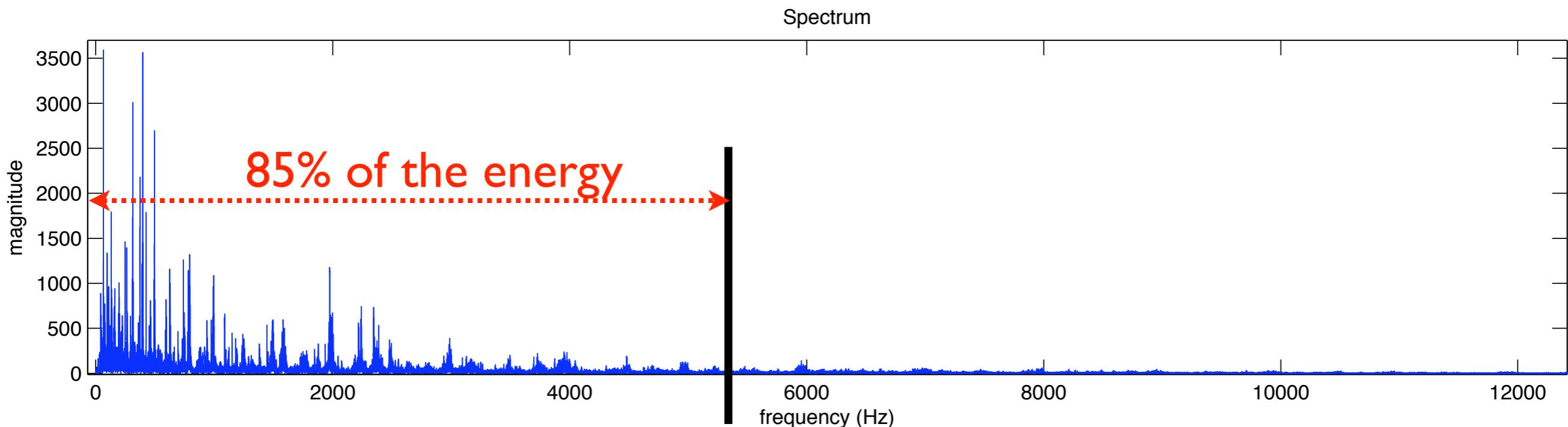


sig.zerocross waveform sign-change rate



- Indicator of nosiness
- `sig.zerocross(..., 'Per', 'Second')`: rate per second
- `sig.zerocross(..., 'Per', 'Sample')`: rate per sample
- `sig.zerocross(..., 'Dir', 'One')`: only ↑ or ↓
- `sig.zerocross(..., 'Dir', 'Both')`: both ↑ and ↓

sig.rolloff high-frequency energy (I)

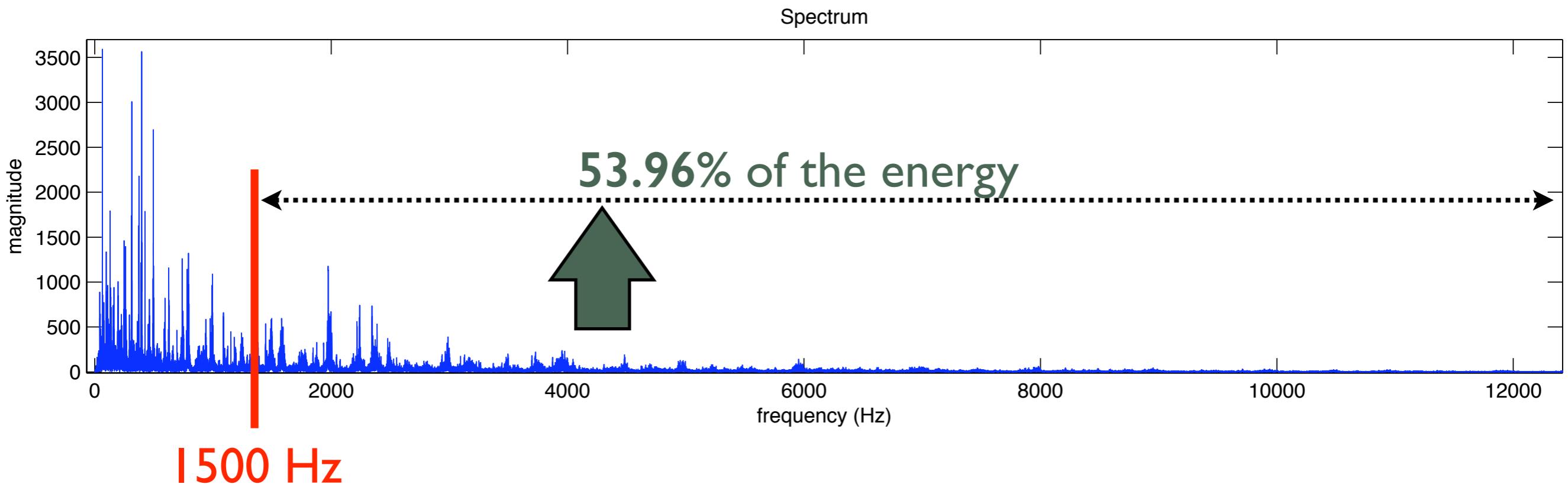


mirrolloff(..., 'Threshold', .85)

th=.85 in Tzanetakis, Cook. Musical genre classification of audio signals. IEEE Tr. Speech and Audio Processing, 10(5),293-302, 2002.

th=.95 in Pohle, Pampalk, Widmer. Evaluation of Frequently Used Audio Features for Classification of Music Into Perceptual Categories, ?

aud.brightness high-frequency energy (II)

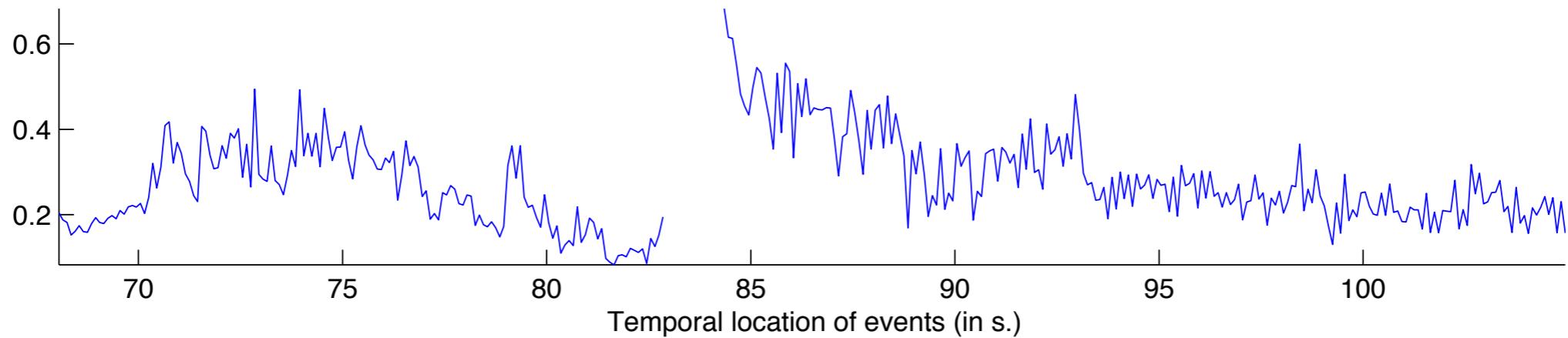


- *aud.brightness(..., 'CutOff', 1500)* (in Hz)
- *aud.brightness(..., 'Unit', u)* u = '/1' or '%'
 - 3000 Hz in Juslin 2001, p. 1802.
 - 1500 Hz and 1000 Hz in Laukka, Juslin and Bresin 2005.

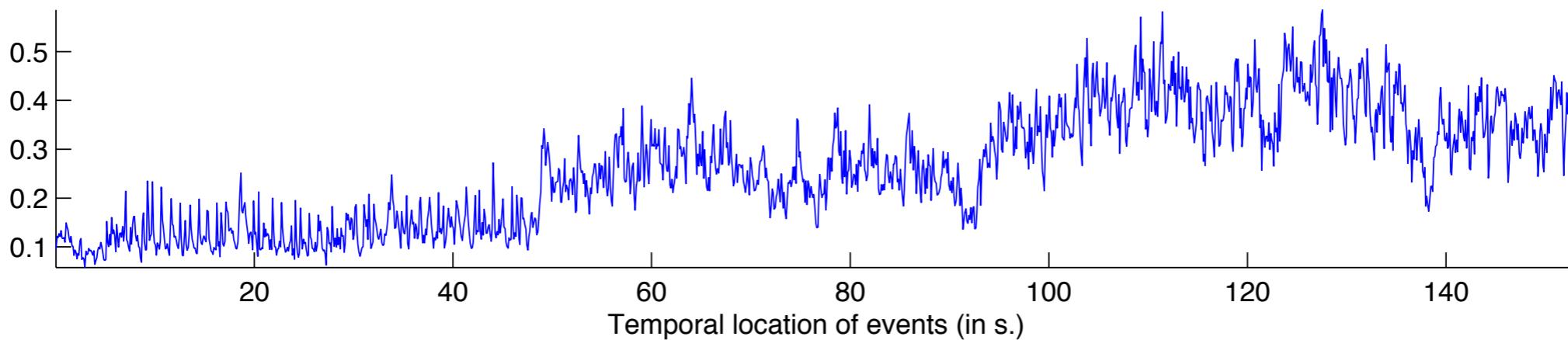
aud.brightness high-frequency energy (II)

- *aud.brightness(..., 'Frame')*

frame length = .05 s
frame hop = 50%



Beethoven, 9th Symphony, *Scherzo*

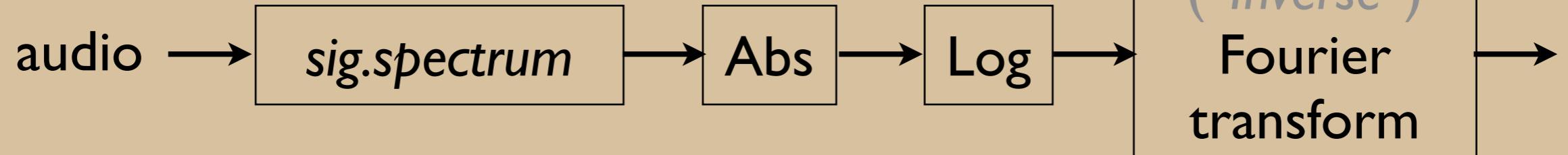


Beethoven, 7th Symphony, *Allegretto*

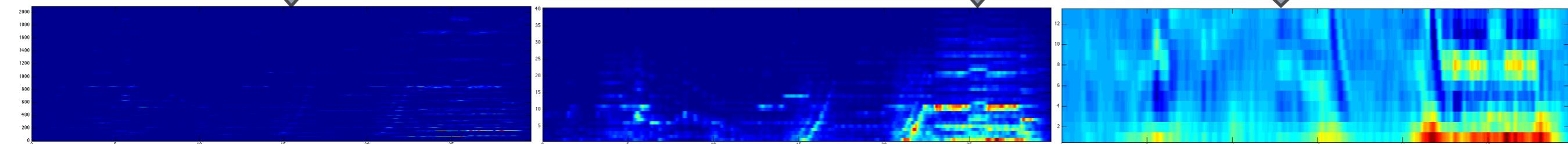
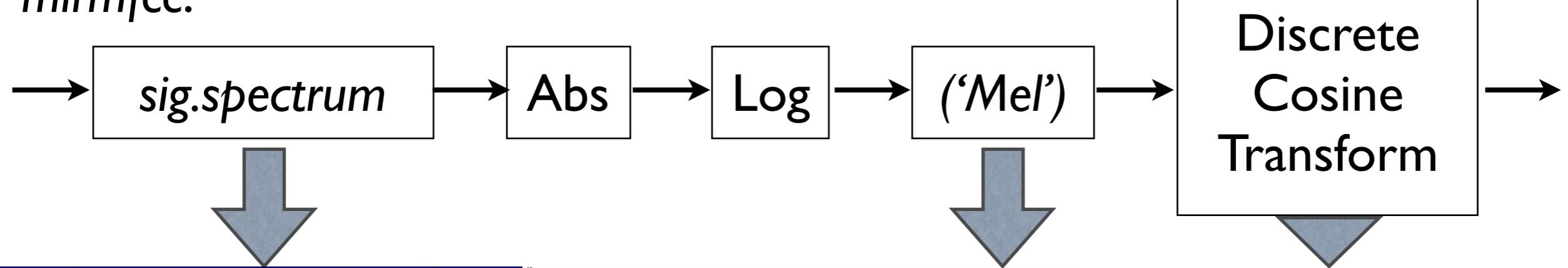
aud.mfcc

mel-frequency cepstral coefficients

mircepstrum:



mirmfcc:

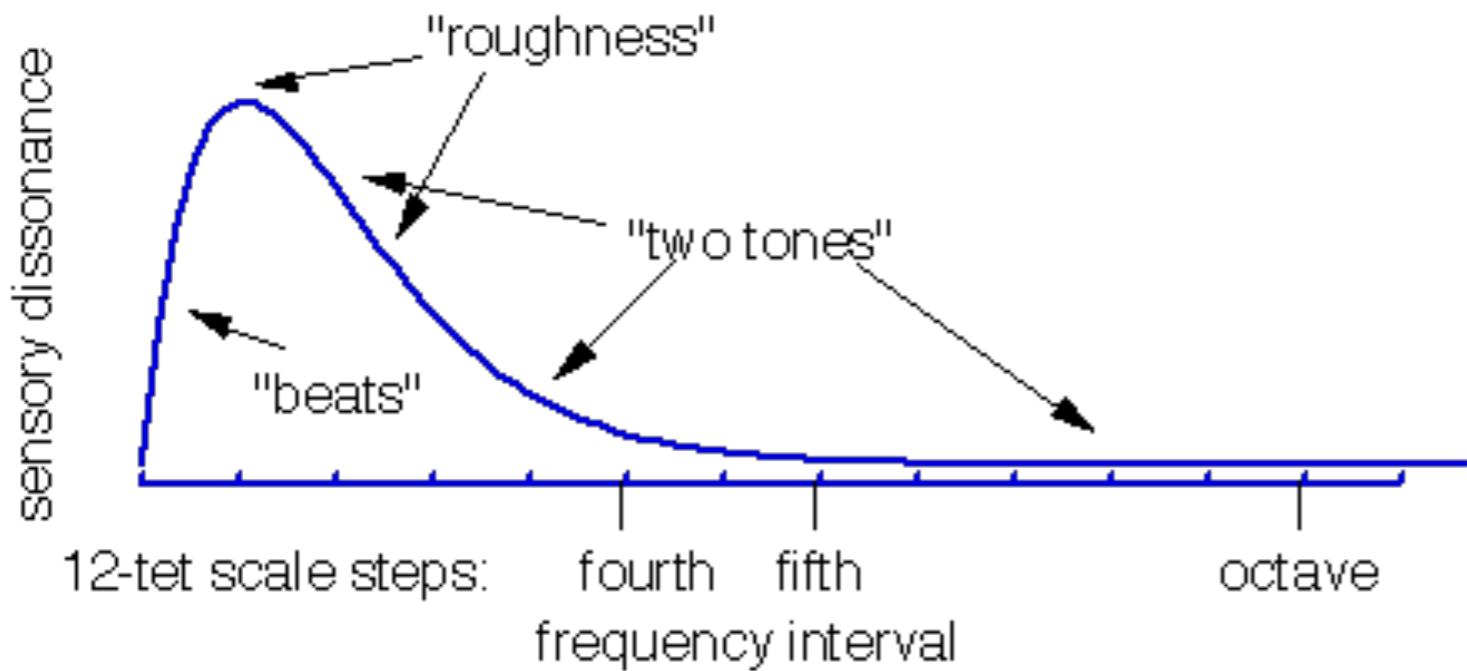


- Description of spectral shape

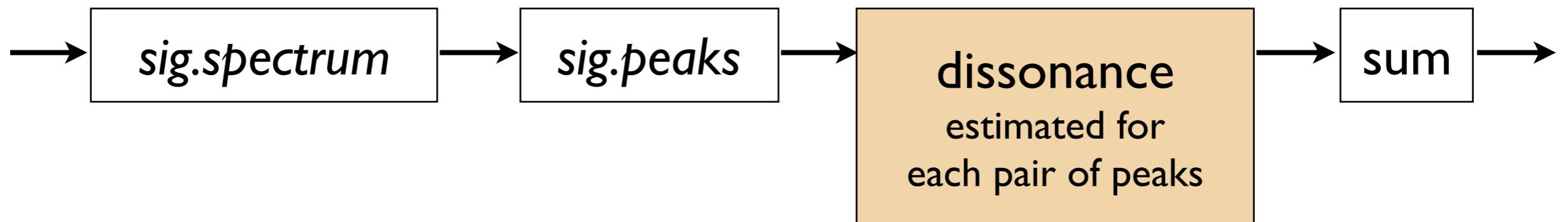
aud.roughness

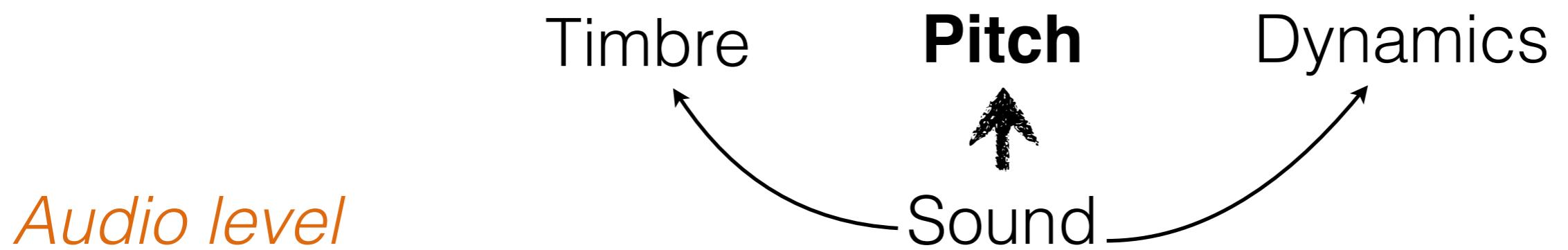
sensory dissonance

- *aud.roughness(..., 'Sethares')*

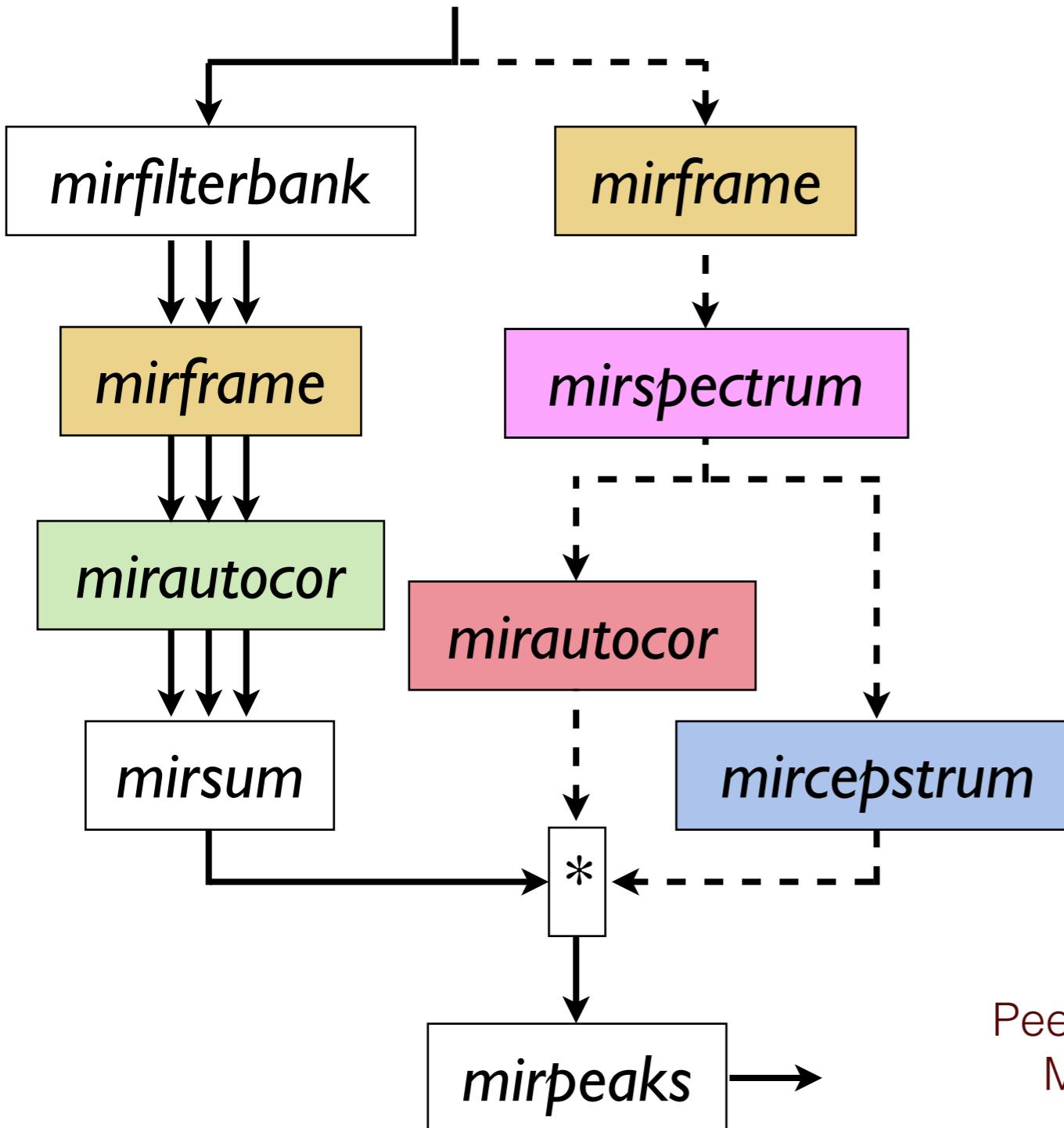


Dissonance produced by two sinusoids depending on their frequency ratio





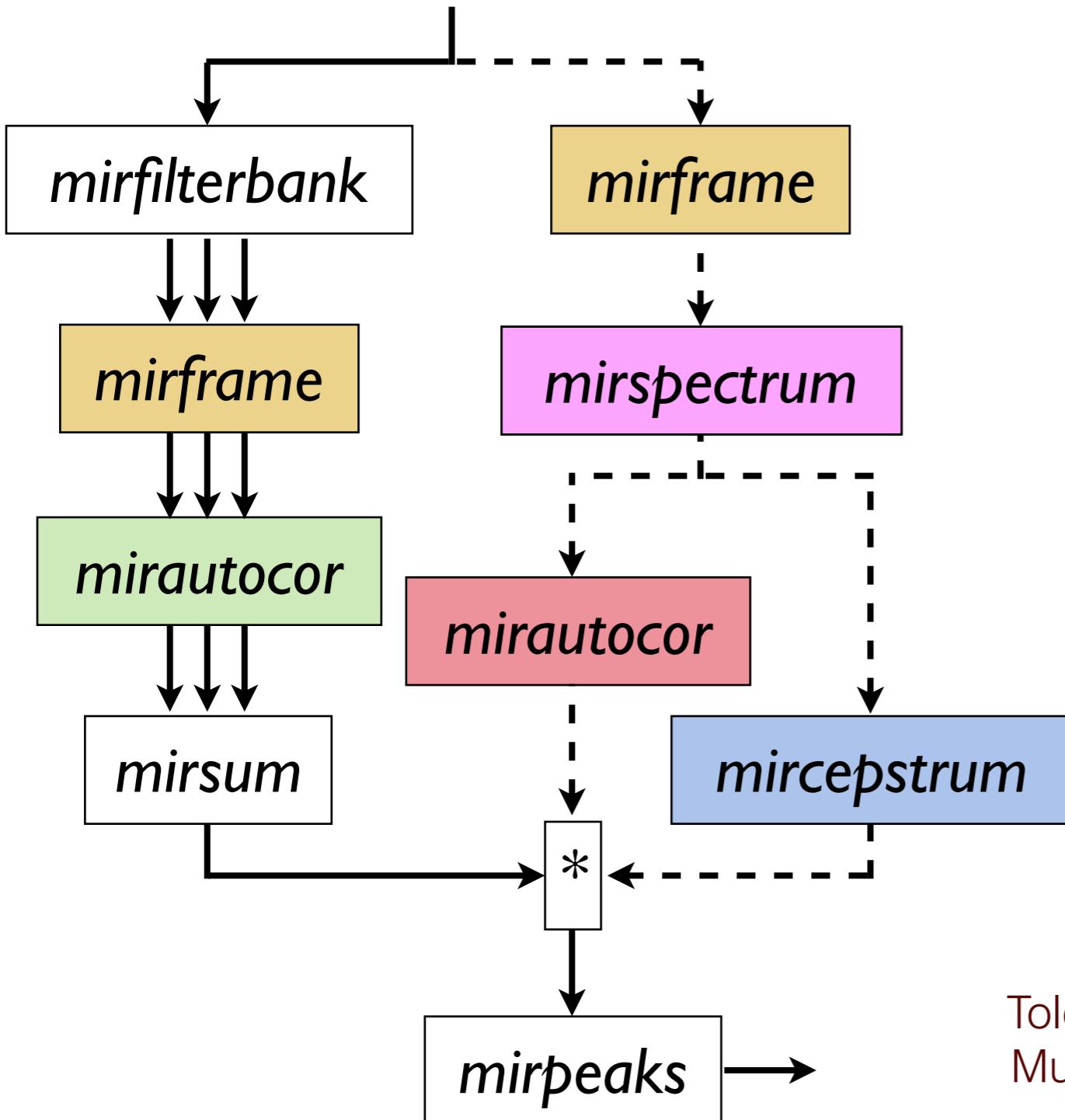
mus.pitch f0 estimation



- $p = \text{mirpitch}(\dots, \text{'Autocor'})$
- $\text{mirpitch}(\dots, \text{'AutocorSpectrum'})$
- $\text{mirpitch}(\dots, \text{'Cepstrum'})$
- $\text{mirpitch}(\dots, \text{'Frame'}, \dots)$

Peeters. Music Pitch Representation by Periodicity
Measures Based on Combined Temporal and
Spectral Representations. ICASSP 2006.

mus.pitch f0 estimation

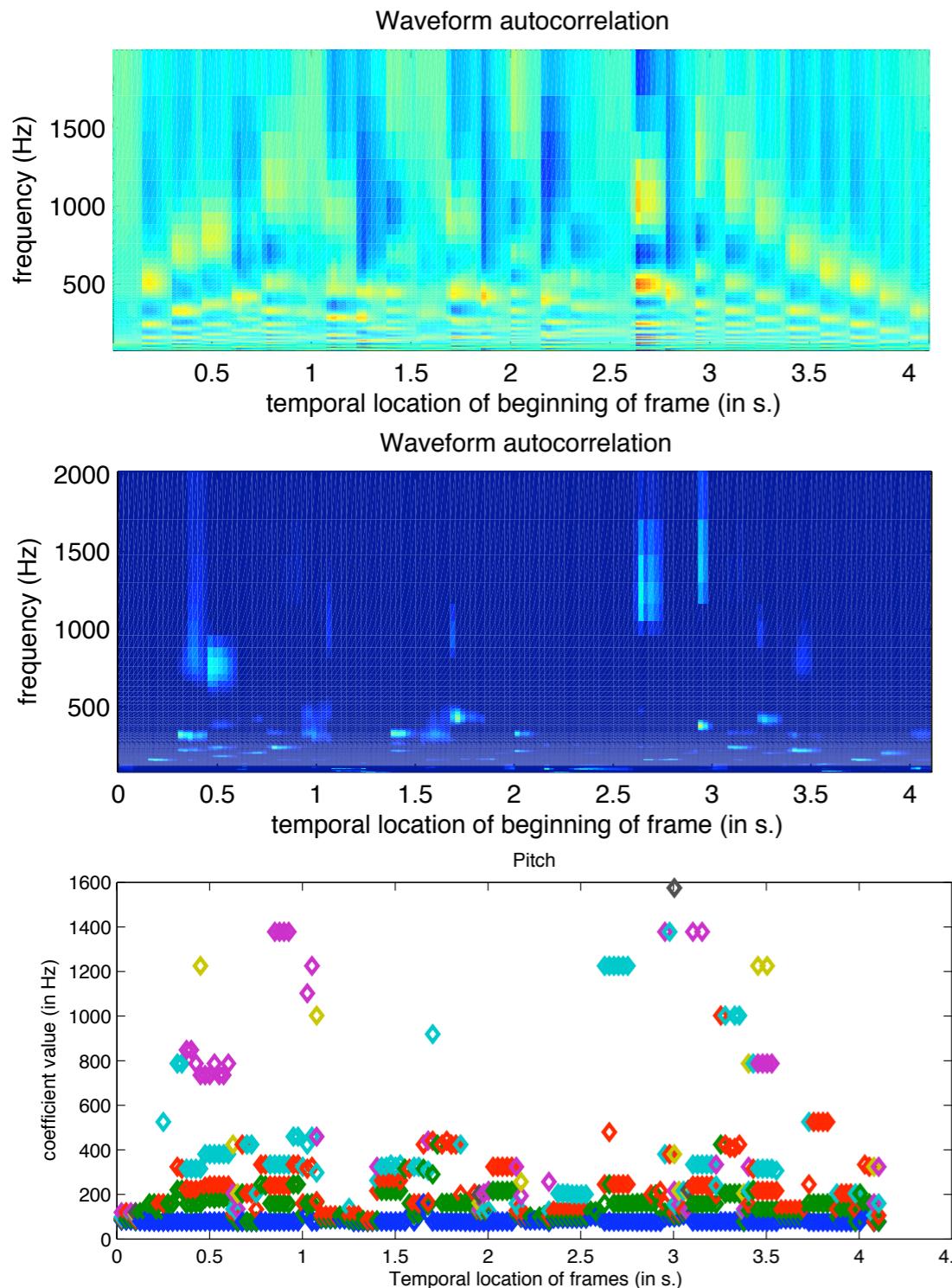


- *mirpitch(..., 'Tolonen')*
- *mirpitch(..., '2Channels', 'Enhanced', 2:10, 'Generalized', .67)*

Tolonen, Karjalainen. A Computationally Efficient Multipitch Analysis Model. IEEE Transactions on Speech and Audio Processing, 8(6), 2000.

mus.pitch

f0 estimation



- $ac = \text{mirautocor}(\text{'ragtime'}, \text{'Frame'})$

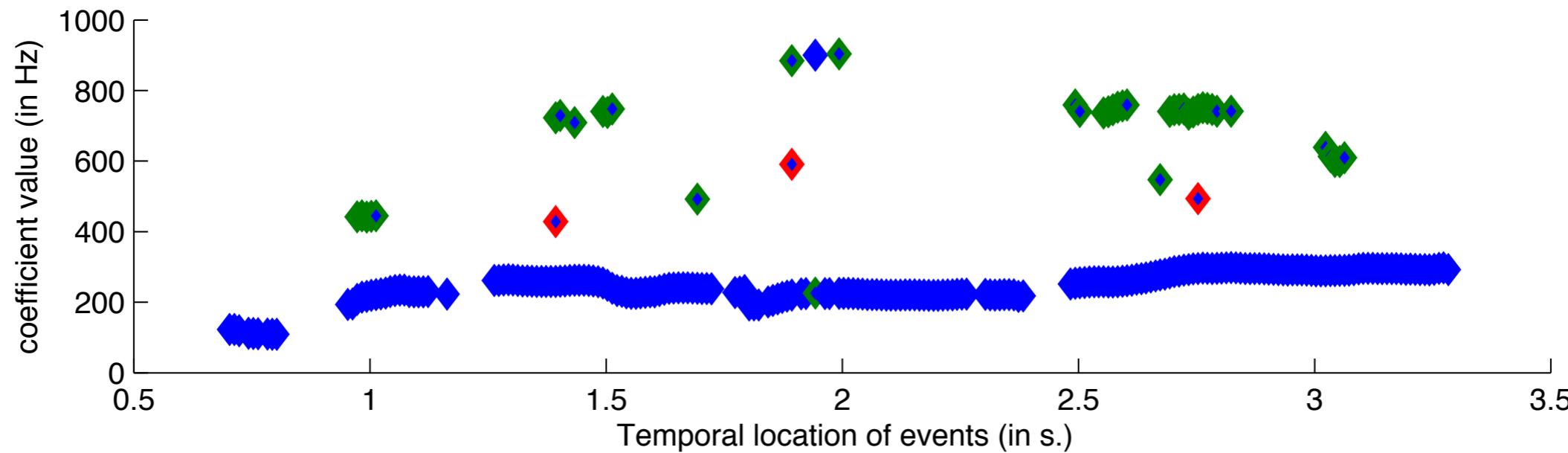
- $ac = \text{mirautocor}(ac, \text{'Enhanced'}, 2:10)$

- $\text{mirpitch}(ac)$

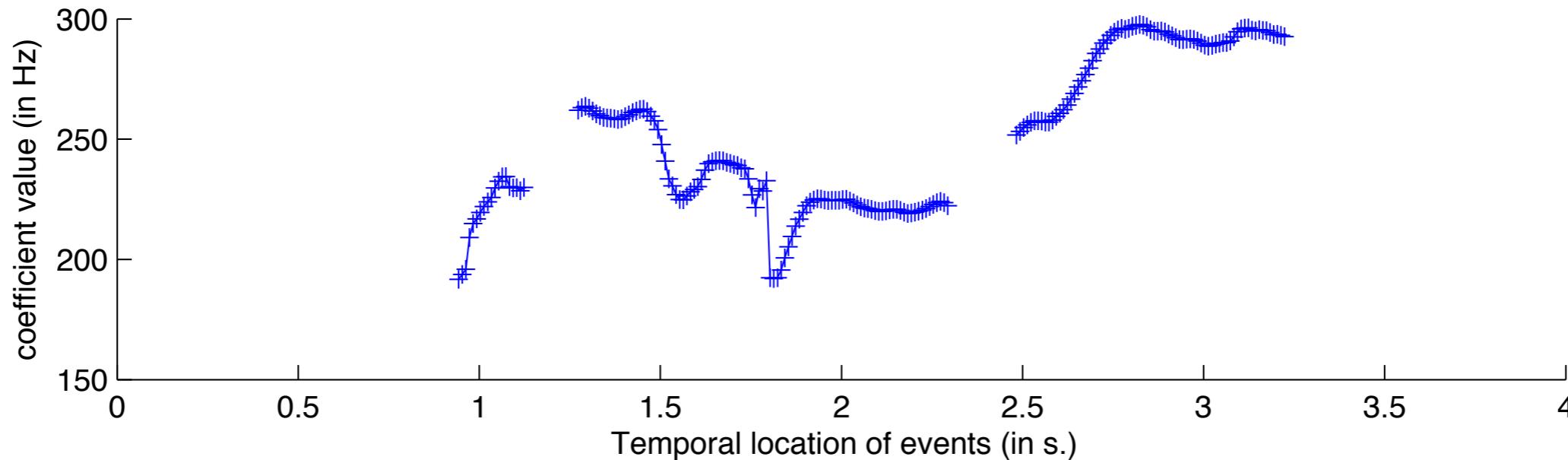
Tolonen, Karjalainen. A Computationally Efficient Multipitch Analysis Model. IEEE Transactions on Speech and Audio Processing, 8(6), 2000.

mus.pitch f0 estimation

- `mus.pitch(..., 'Frame')`

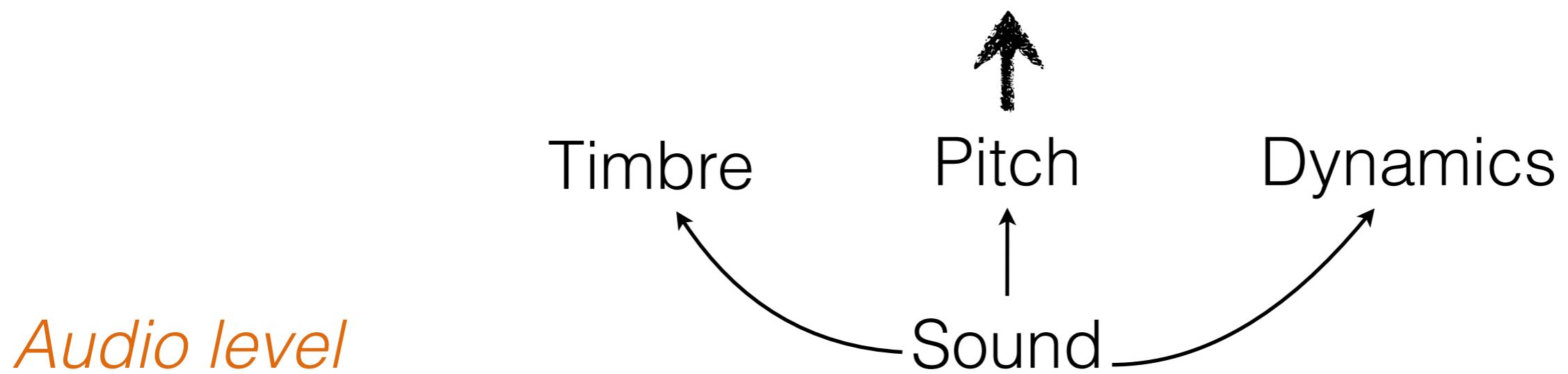


- `mus.pitch(..., 'Frame', 'Mono', 'Max', 400)`



Symbolic level

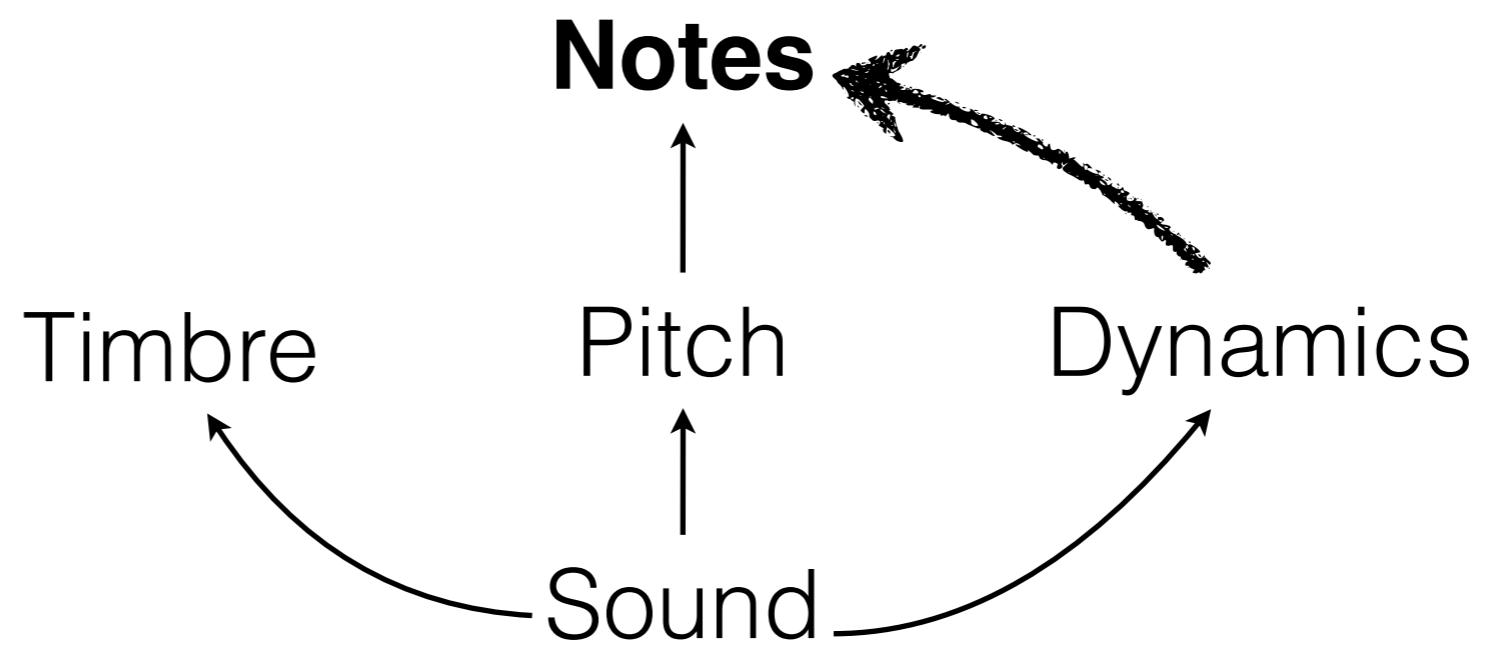
Notes



mus.pitch(..., 'Segment')
pitch-based segmentation

Symbolic level

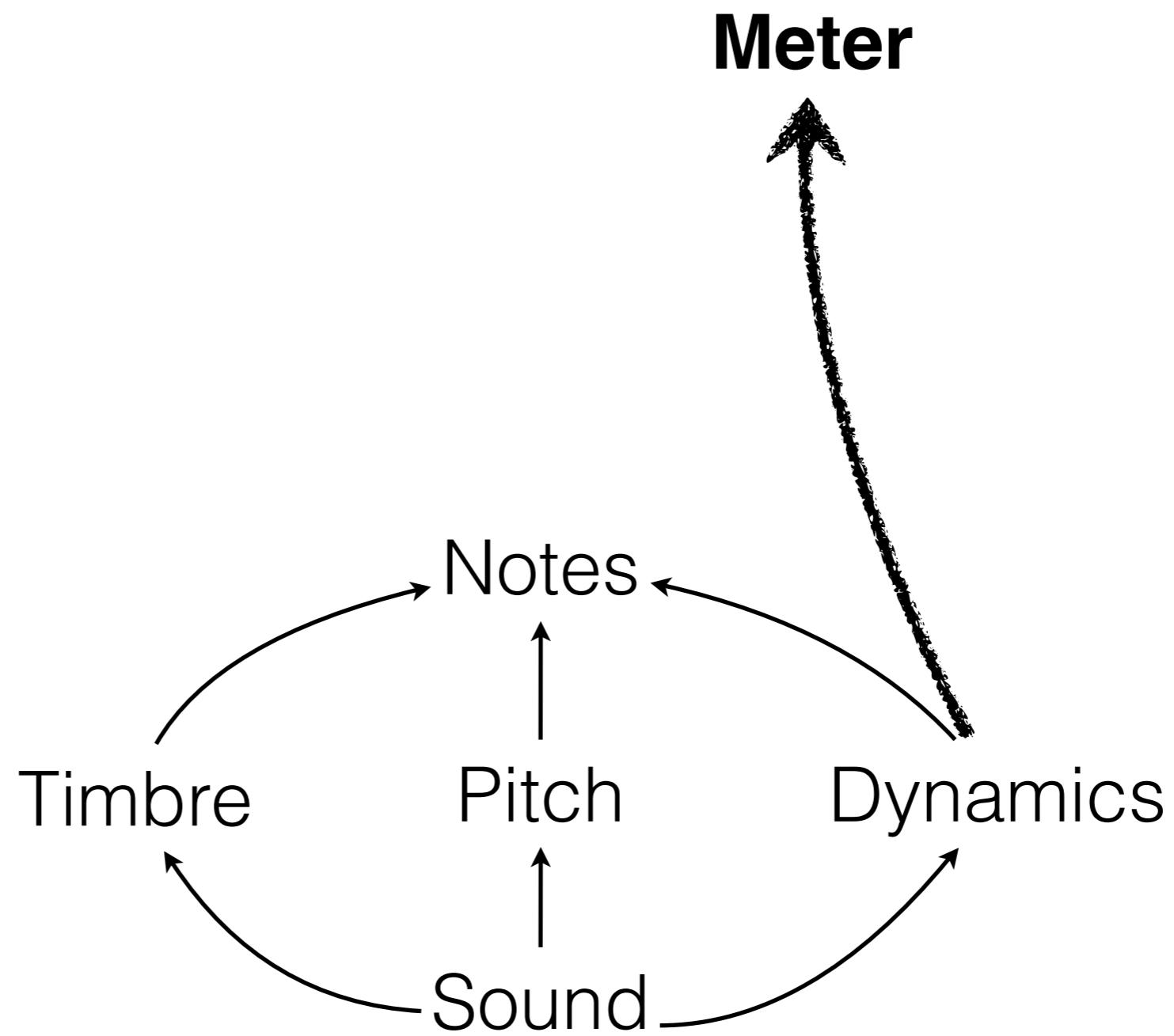
Audio level



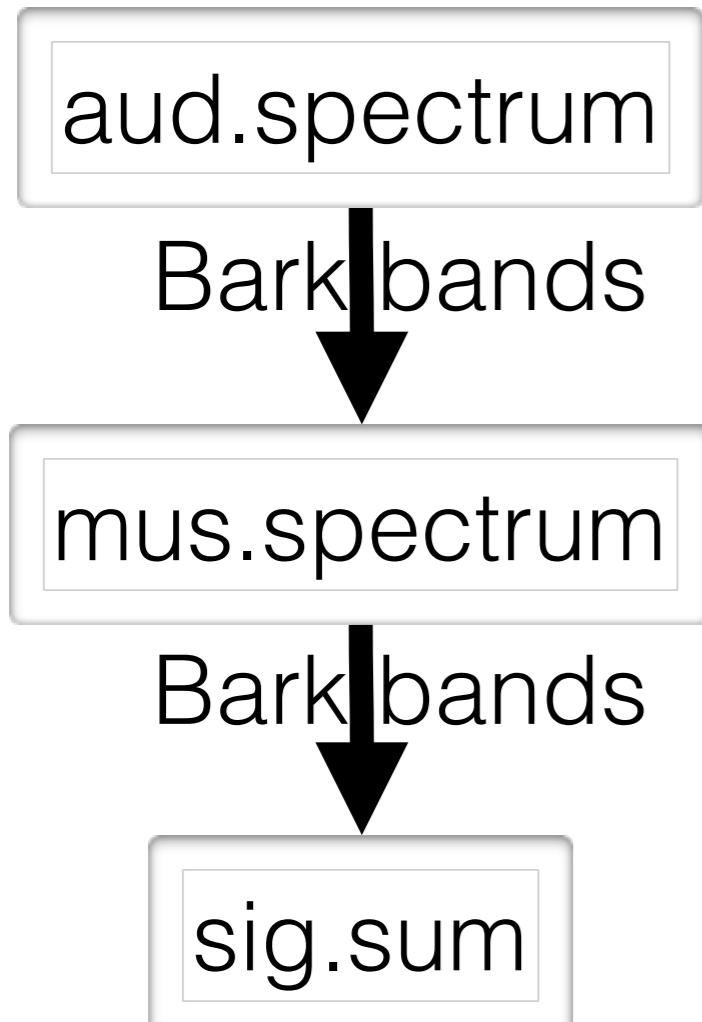
*Structural
levels*

Symbolic level

Audio level

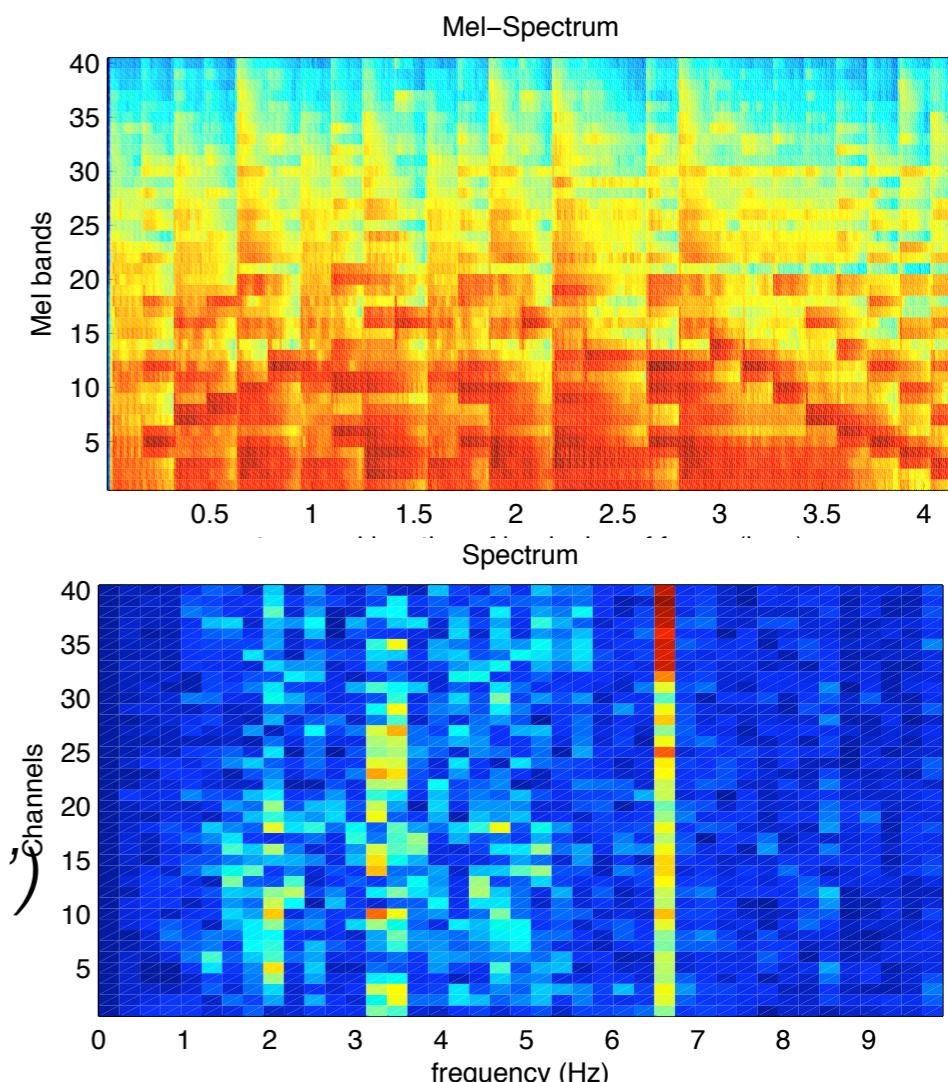
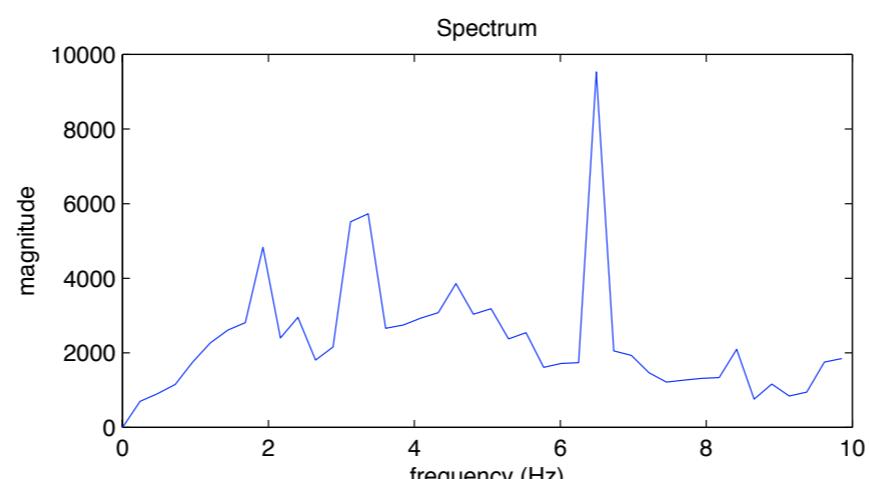


mus.fluctuation rhythmic periodicity along auditory channels



('Frame',.023,.5, 'Terhardt',
'Bark', 'Mask', 'dB')

(**'AlongBands'**, 'Max', 10,
'Window', 'No',
'Resonance', **'Fluctuation'**)



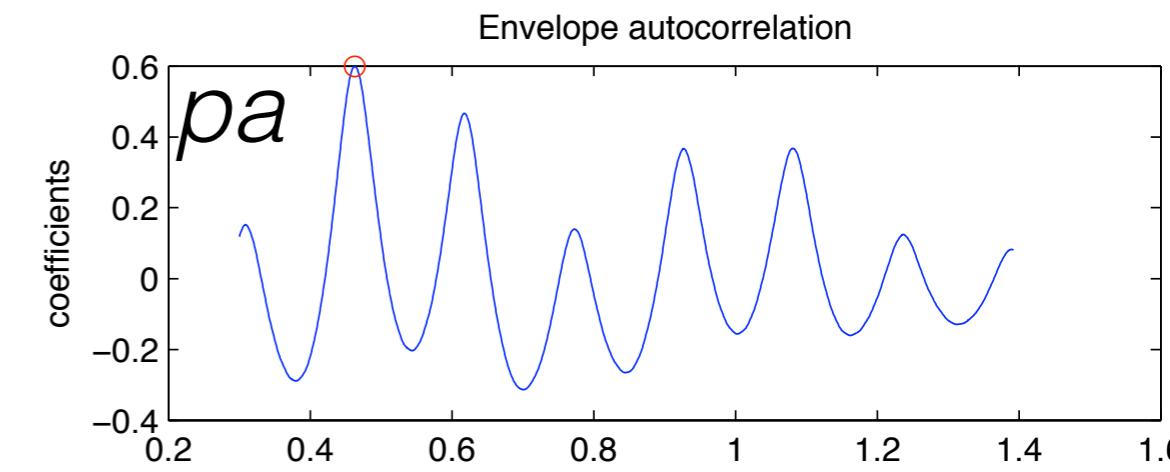
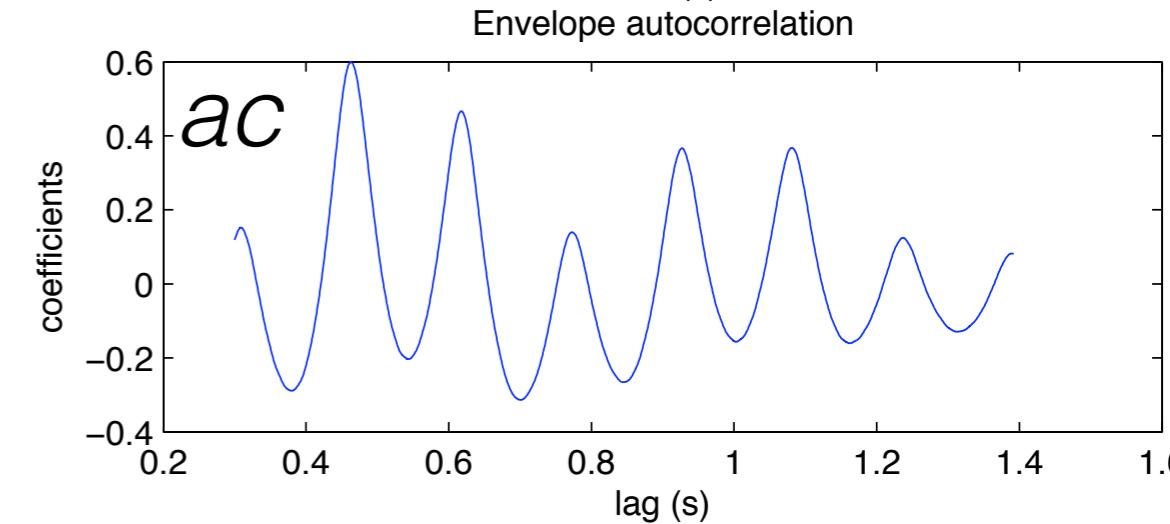
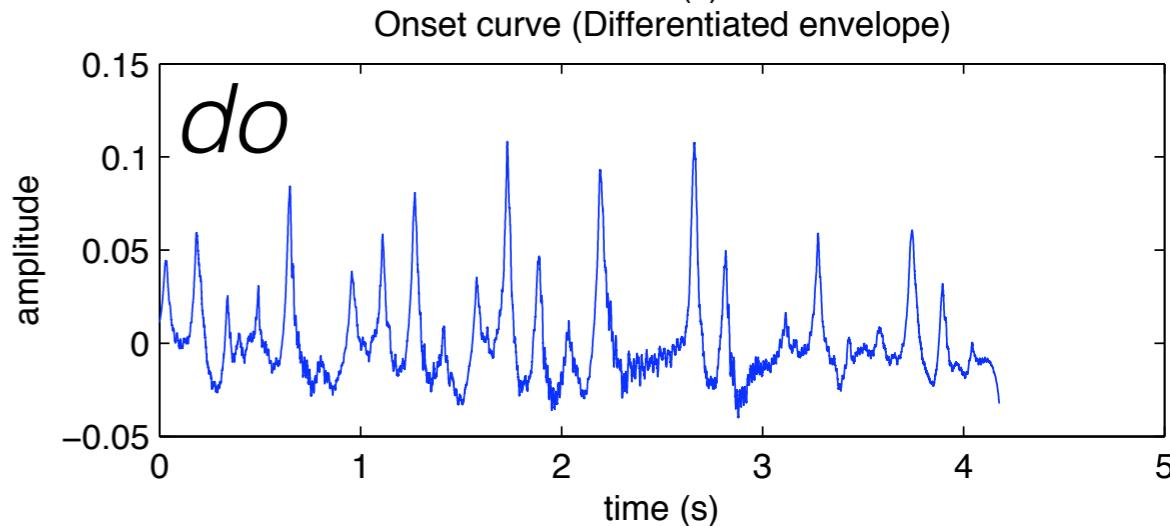
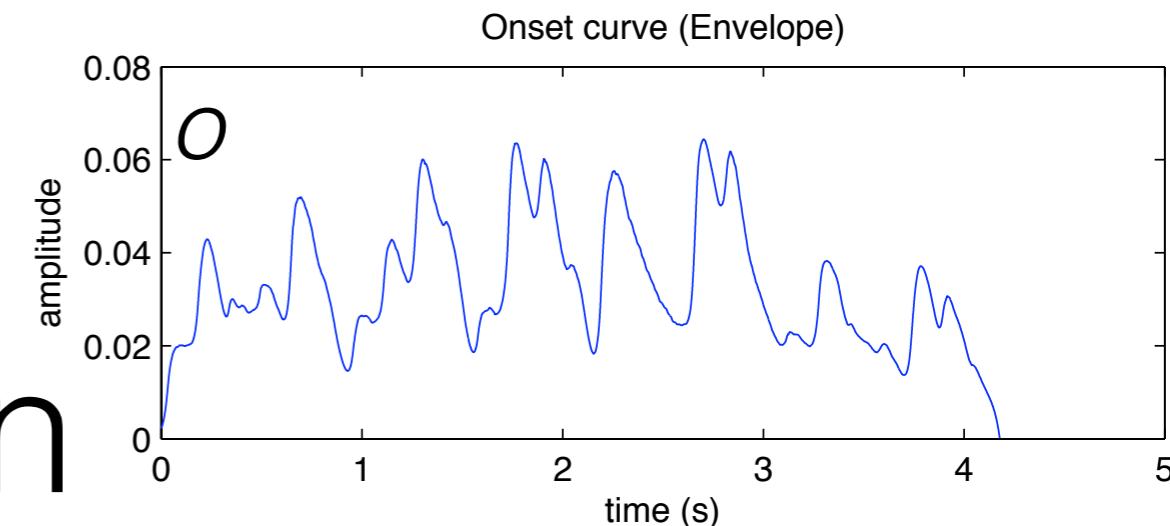
mus.tempo tempo estimation

- $o = \text{aud.onsets}(\text{'mysong'}, \text{'Detect'}, \text{'No'})$
- $do = \text{aud.onsets}(o, \text{'Diff'})$
- $ac = \text{sig.autocor}(do)$
- $pa = \text{sig.peaks}(ac, \text{'Total'}, 1)$

In short:

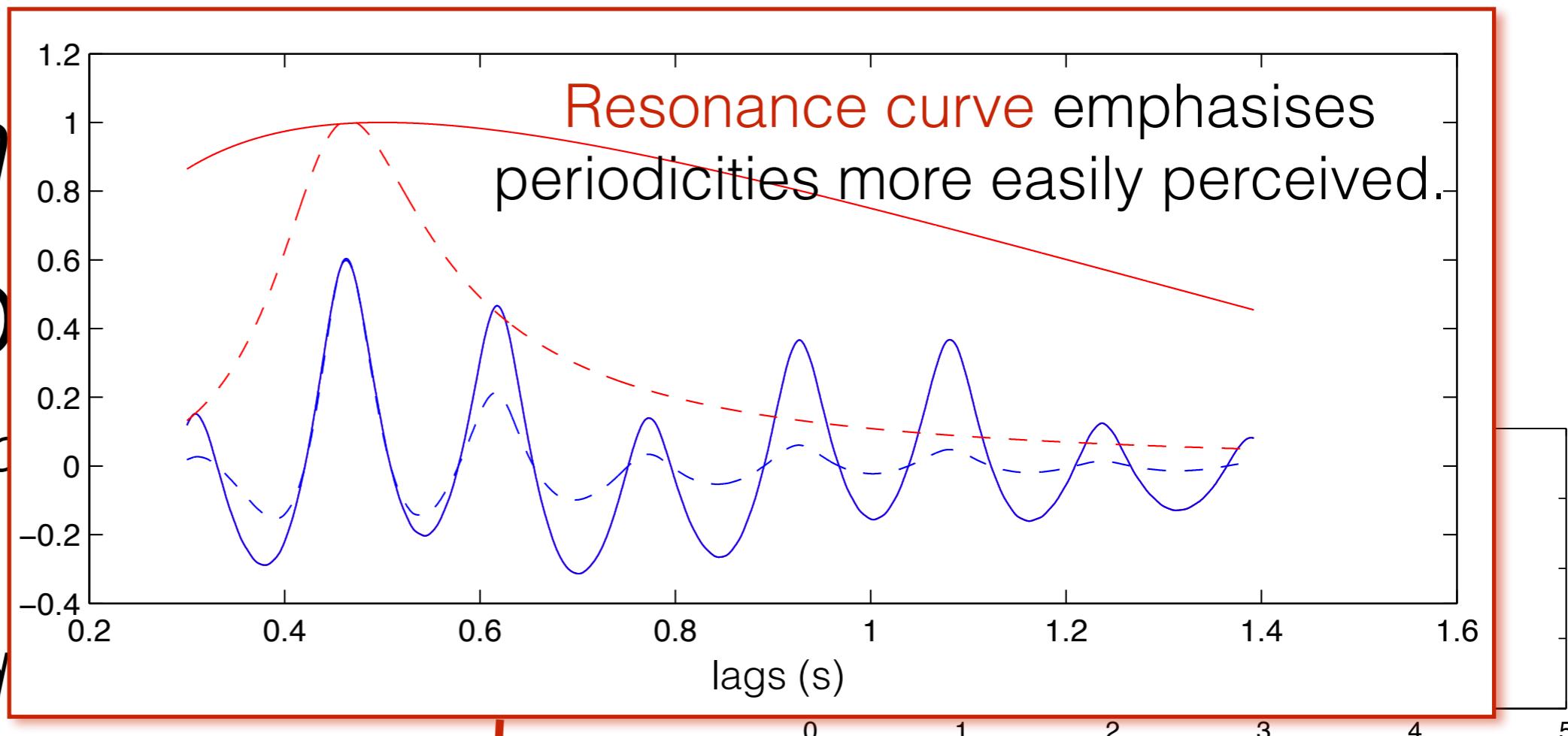
- $[t, pa] = \text{mus.tempo}(\text{'mysong'})$

$t = 129.6333$ bpm



m
temp

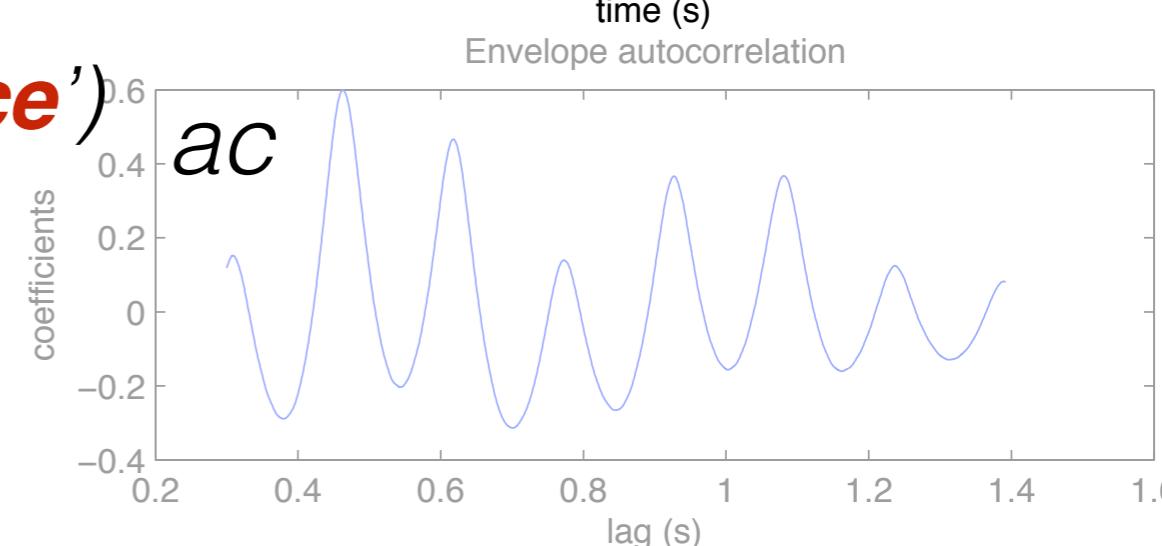
- $o = \text{aud.}$
(‘No’)
- $do = \text{aud.}$



- $ac = \text{aud.autocor}(do, \text{'Resonance'})$
- $pa = \text{sig.peaks}(ac, \text{'Total'}, 1)$

In short:

- $[t, pa] = \text{mus.tempo('mysong')}$



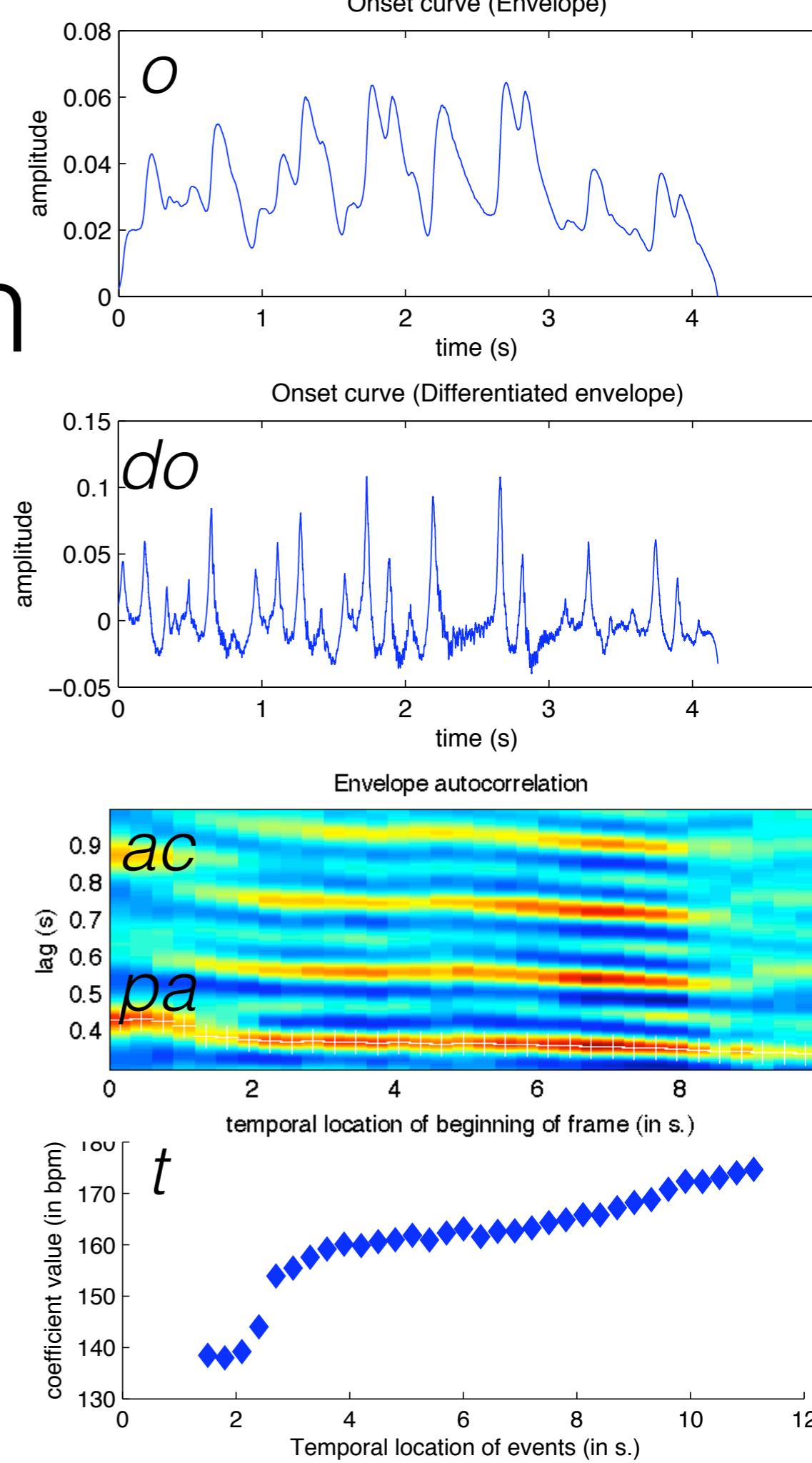
$$t = 129.6333 \text{ bpm}$$

mus.tempo tempo estimation

- $o = \text{aud.onsets}(\text{'mysong'}, \text{'Detect'}, \text{'No'})$
- $do = \text{aud.onsets}(o, \text{'Diff'})$
- $ac = \text{aud.autocor}(do, \text{'Frame'}, \text{'Resonance'})$
- $pa = \text{sig.peaks}(ac, \text{'Total'}, 1)$

In short:

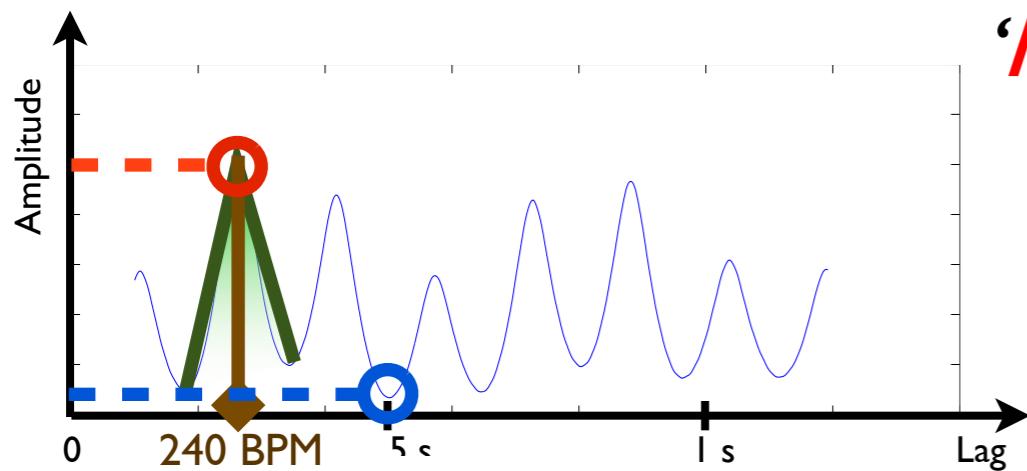
- $[t, pa] = \text{mus.tempo}(\text{'mysong'}, \text{'Frame'})$



mus.pulseclarity

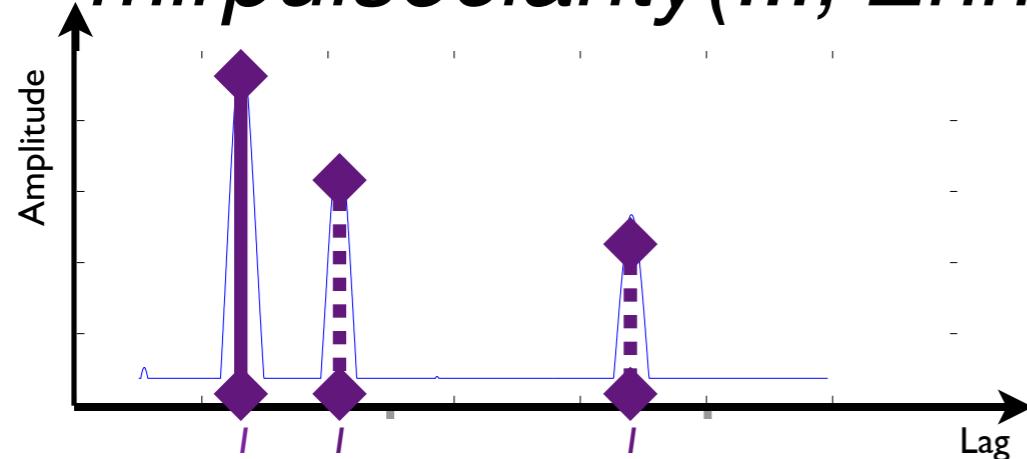
rhythmic clarity, beat strength

mirpulseclarity(..., 'Enhanced', 'No')



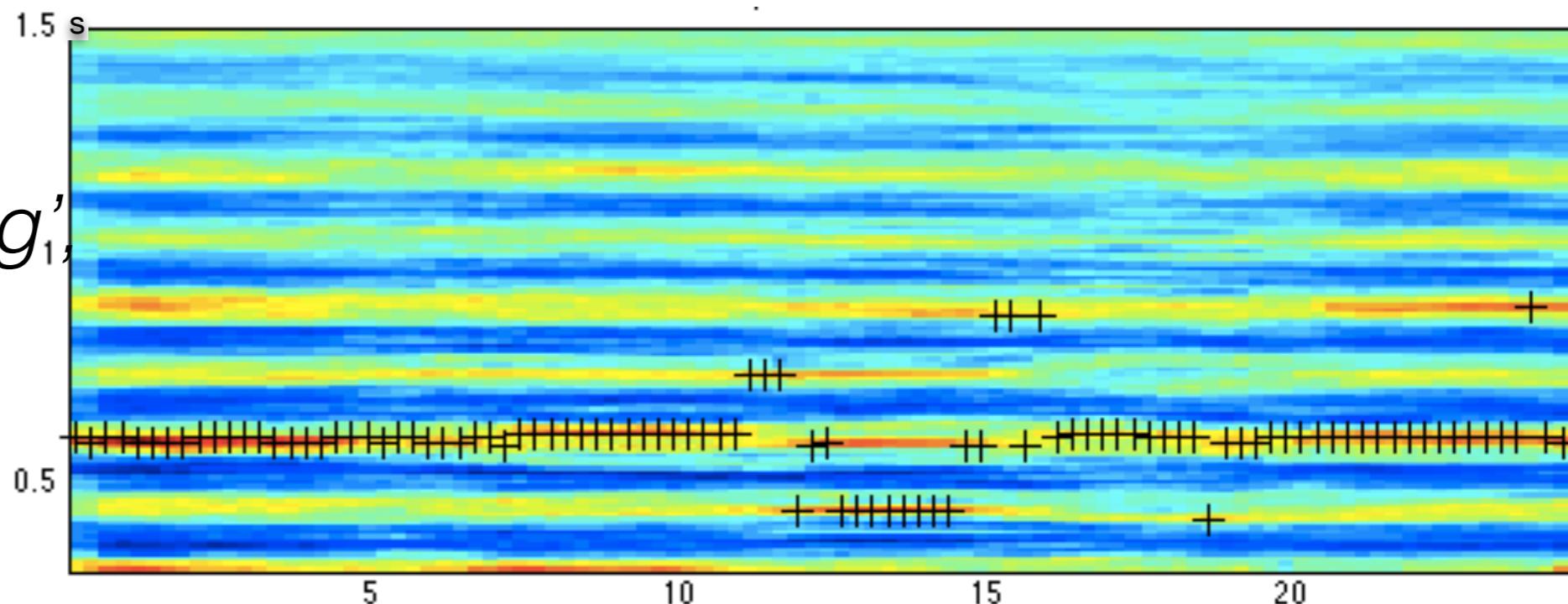
'MaxAutocor' 'MinAutocor'
'KurtosisAutocor'
'TempoAutocor'
'EntropyAutocor'
'InterfAutocor'

mirpulseclarity(..., 'Enhanced', 'Yes')



'EntropyAutocor'
'InterfAutocor'

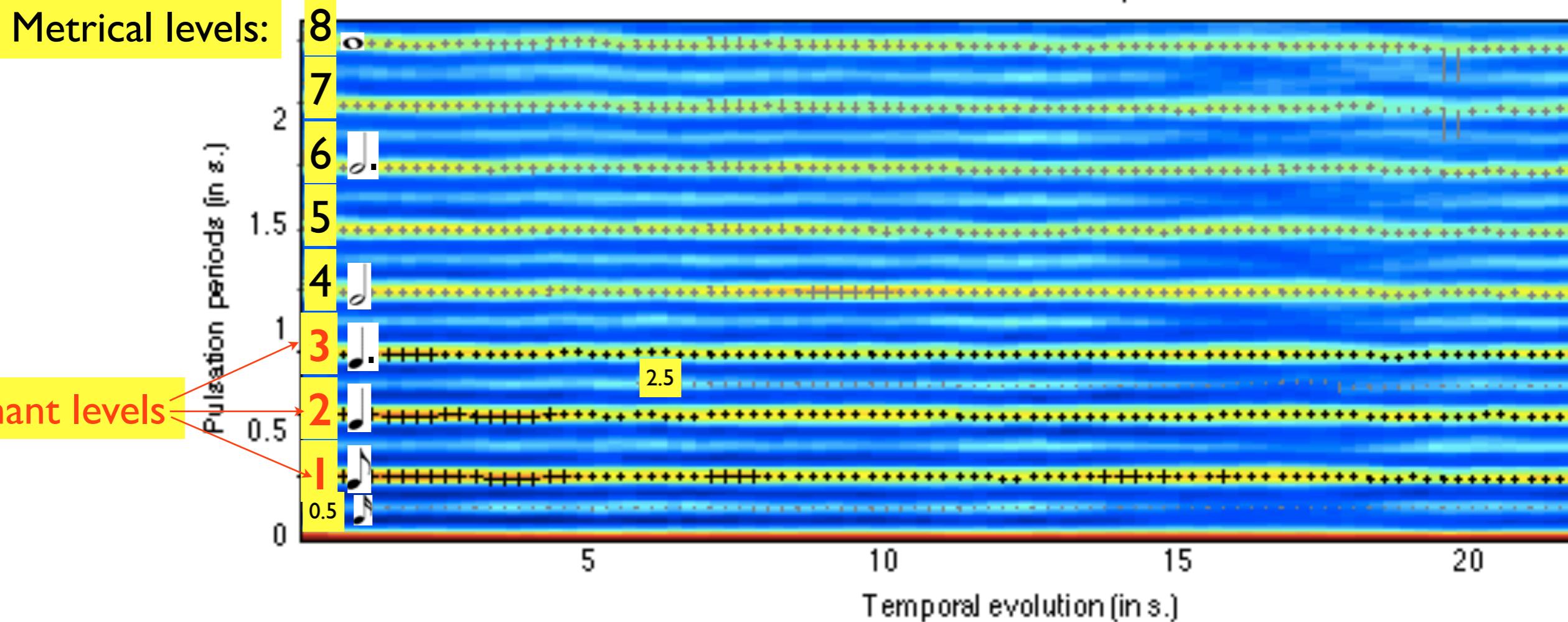
```
[t, pa] =  
mus.tempo('mysong',  
'Frame')
```



Pulsation does not always focus on one single metrical level.

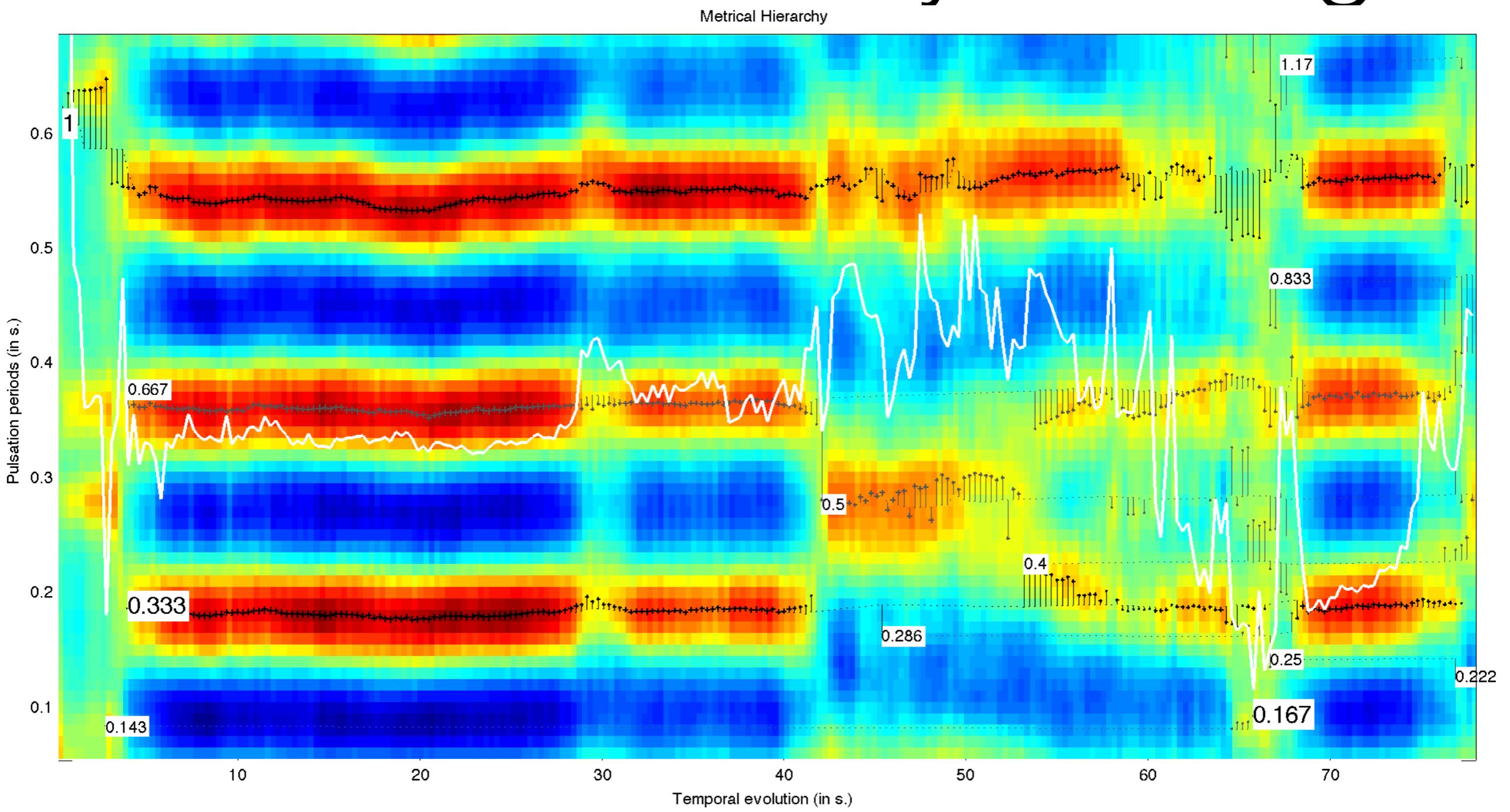
mus.metre tracks all metrical levels in parallel.

Metrical Hierarchy



mus.metre

metrical hierarchy tracking

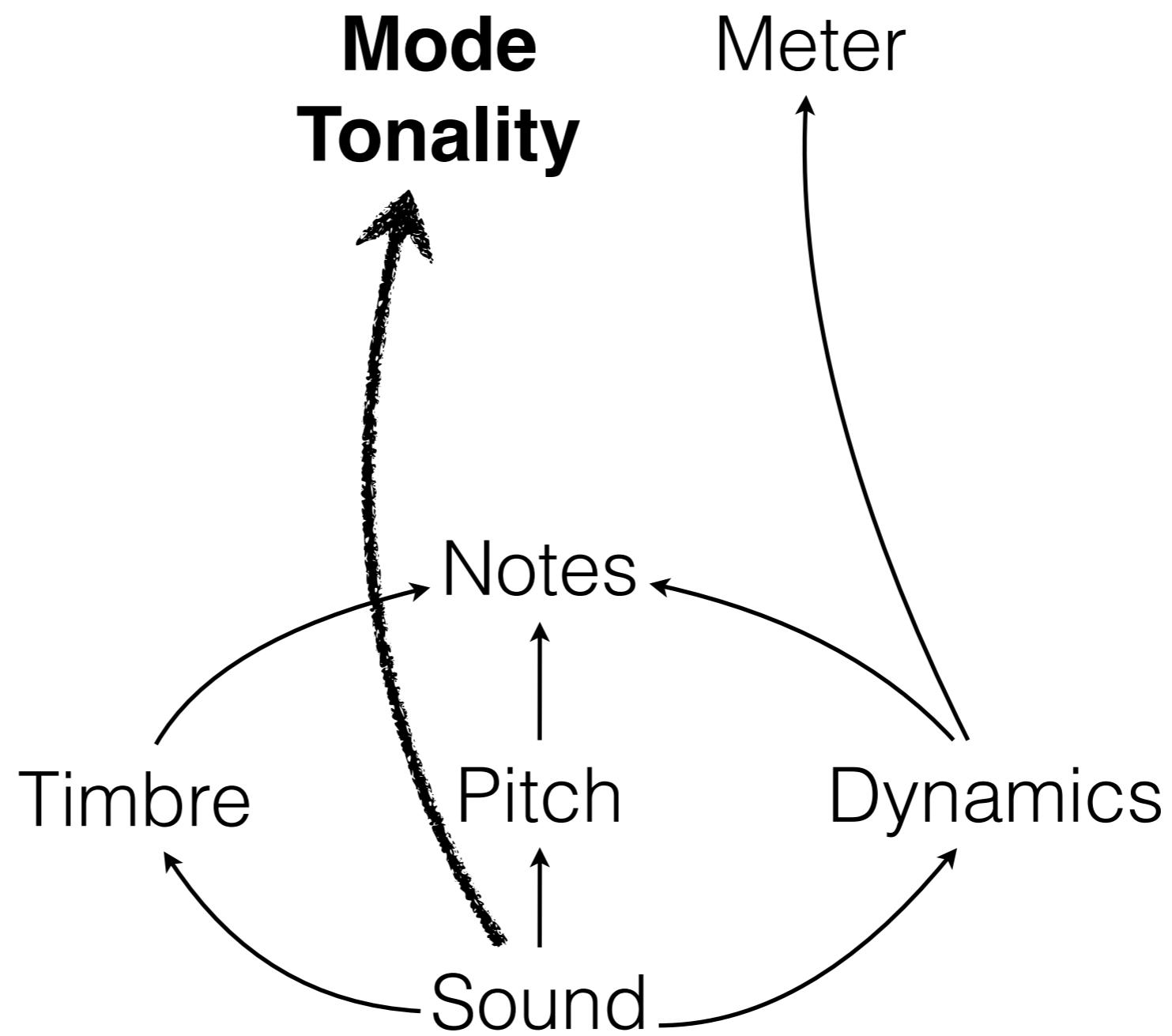


Beethoven, 9th Symphony, Scherzo

*Structural
levels*

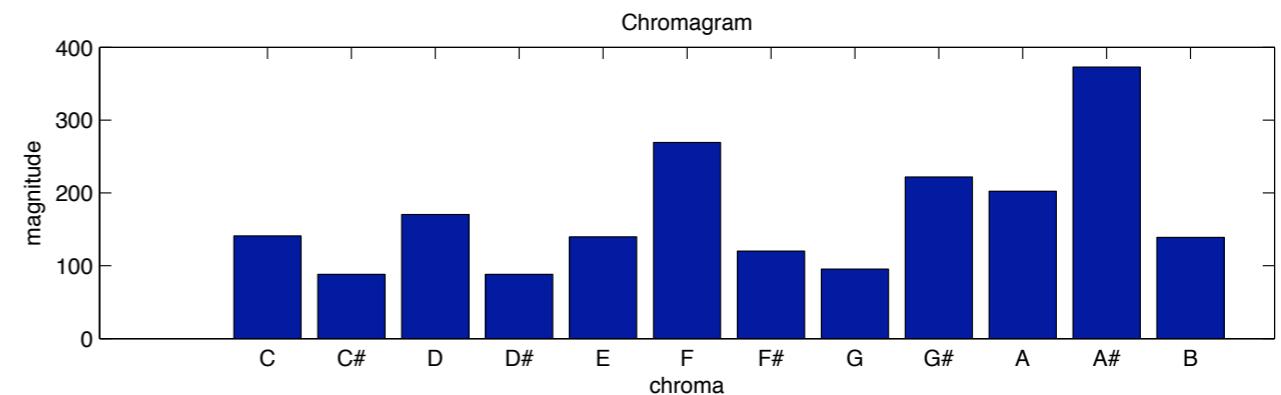
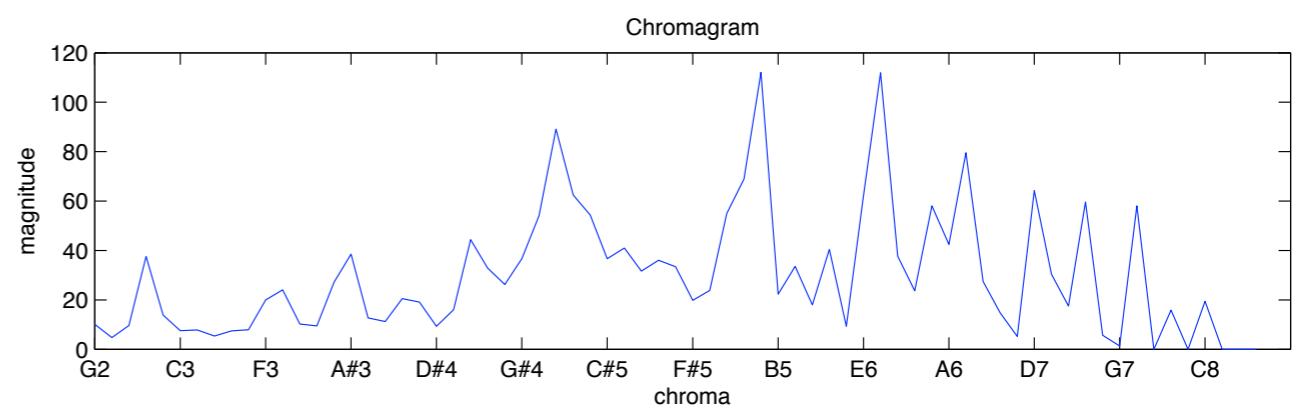
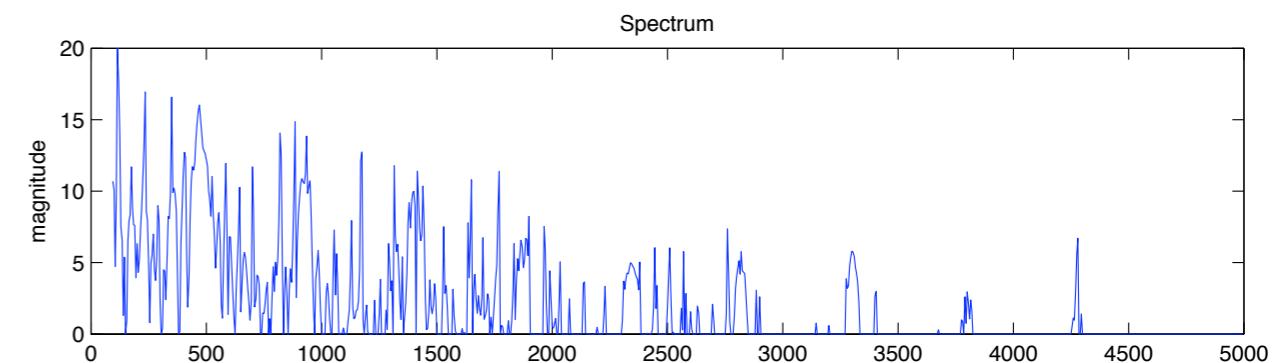
Symbolic level

Audio level



mus.chromagram energy distribution along pitches

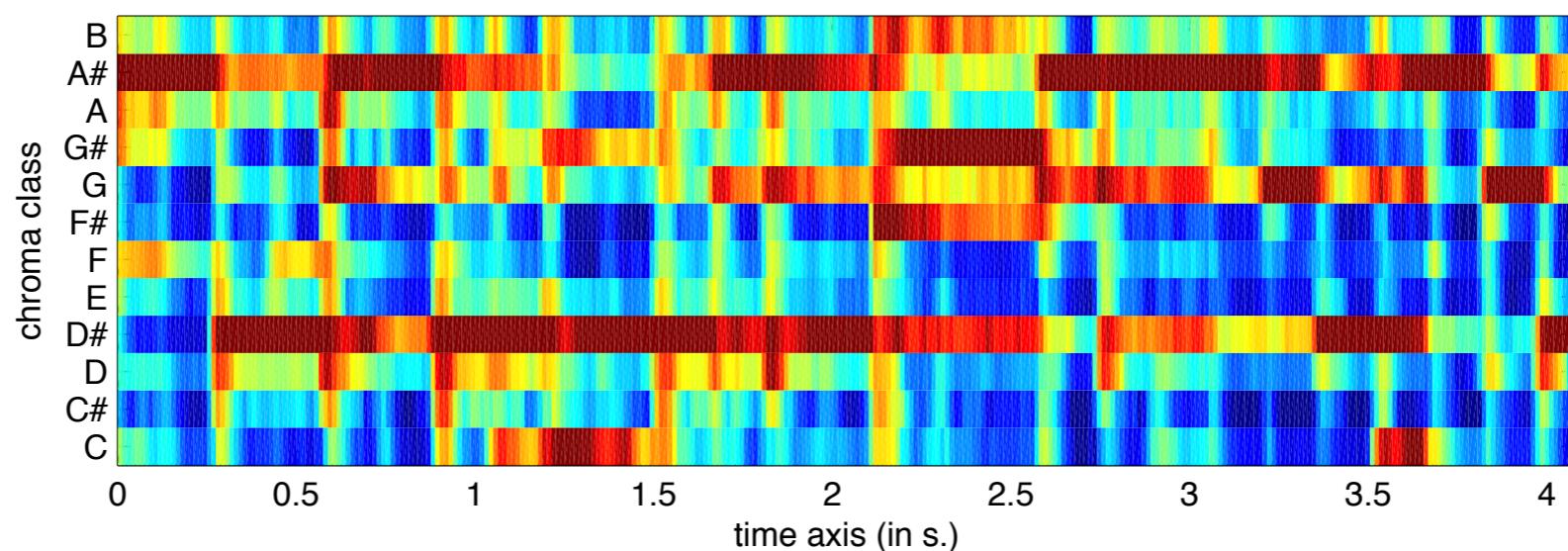
- $s = \text{mirspectrum}(a, \text{'dB'}, 20, \text{'Min'}, 100, \text{'Max'}, 6400)$
- $c = \text{mirchromagram}(s, \text{'Wrap'}, \text{'no'})$
- $c = \text{mirchromagram}(c, \text{'Wrap'}, \text{'yes'})$



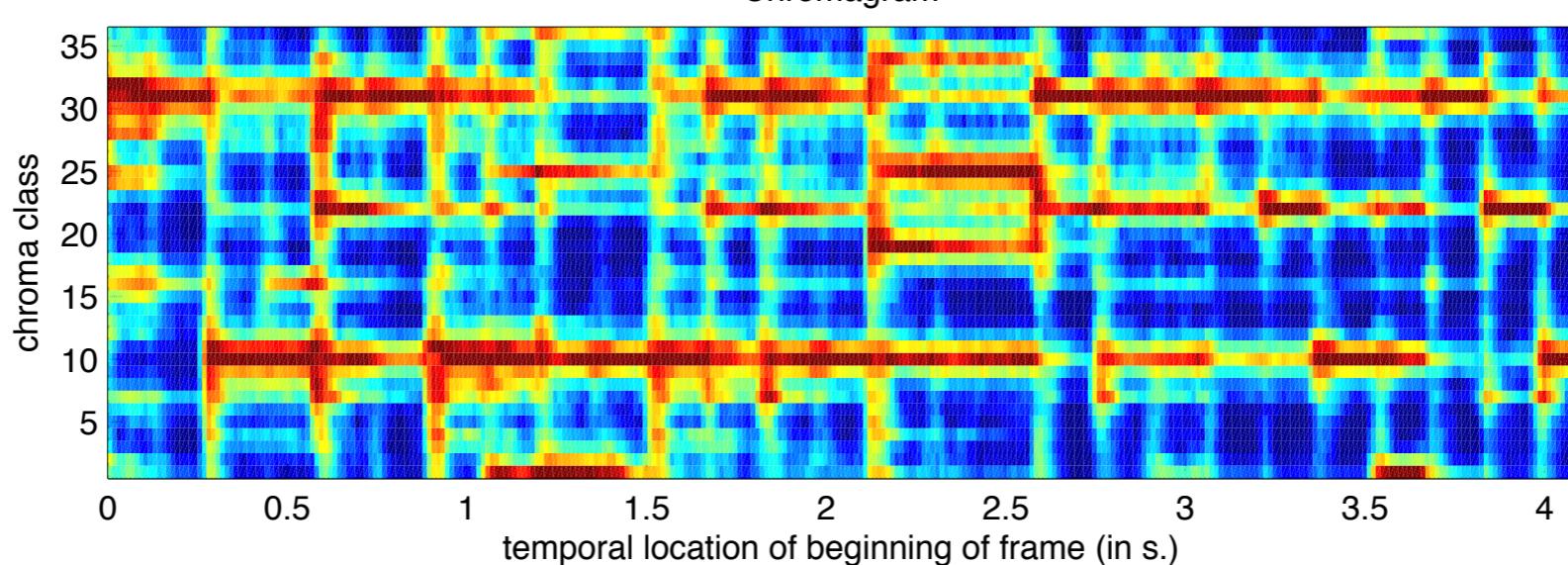
mus.chromagram

chroma resolution

- *mirchromagram(..., 'Res', 12)*



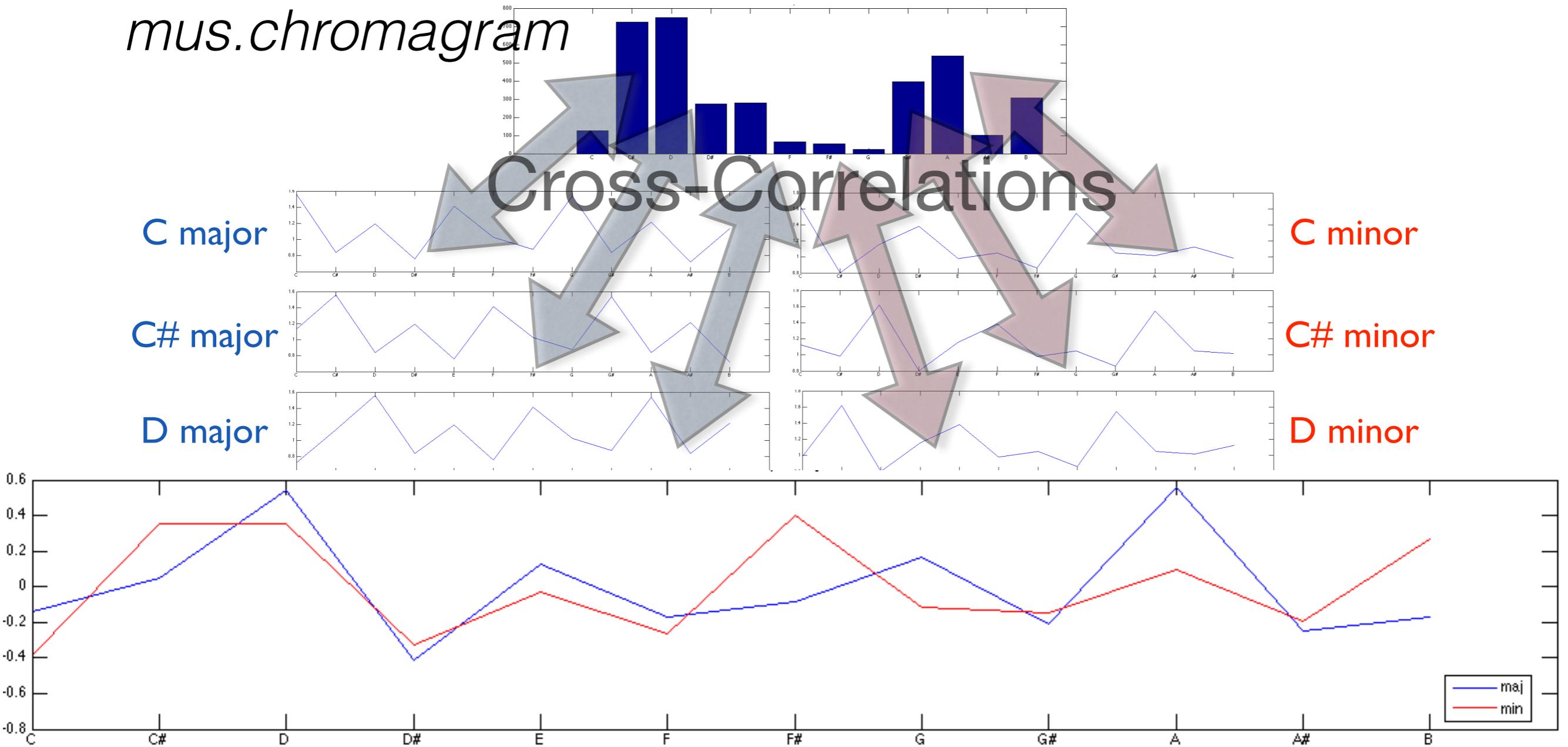
- *mirchromagram(..., 'Res', 36)*



mus.keystrength

probability of key candidates

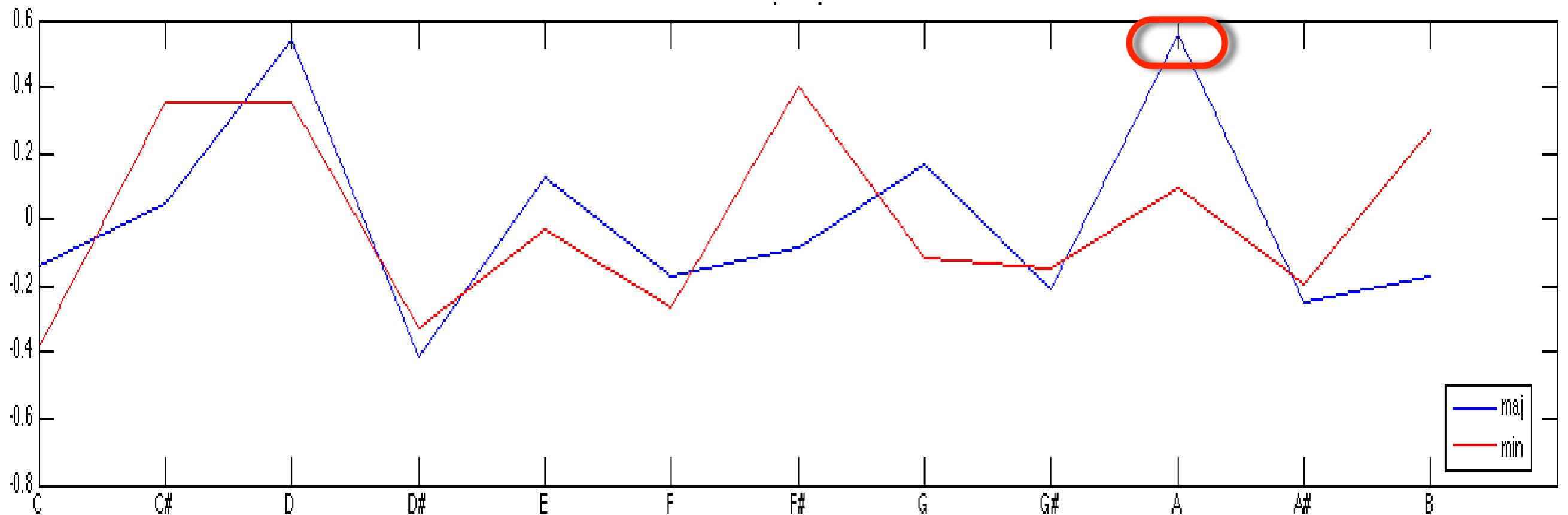
mus.chromagram



Krumhansl, Cognitive foundations of musical pitch. Oxford UP, 1990.

Gomez, "Tonal description of polyphonic audio for music content processing," INFORMS Journal on Computing, 18-3, pp. 294–304, 2006.

mus.key key estimation



sig.peaks(mus.keystrength(...))

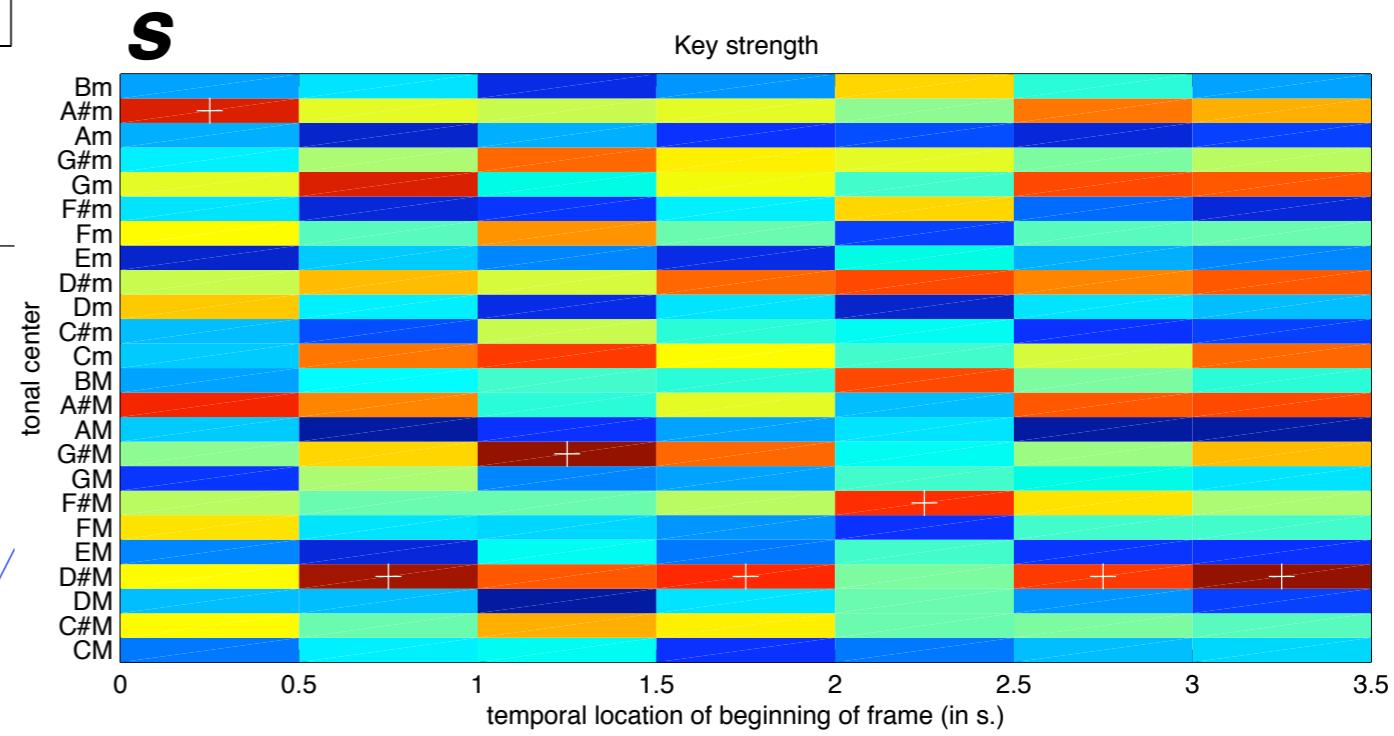
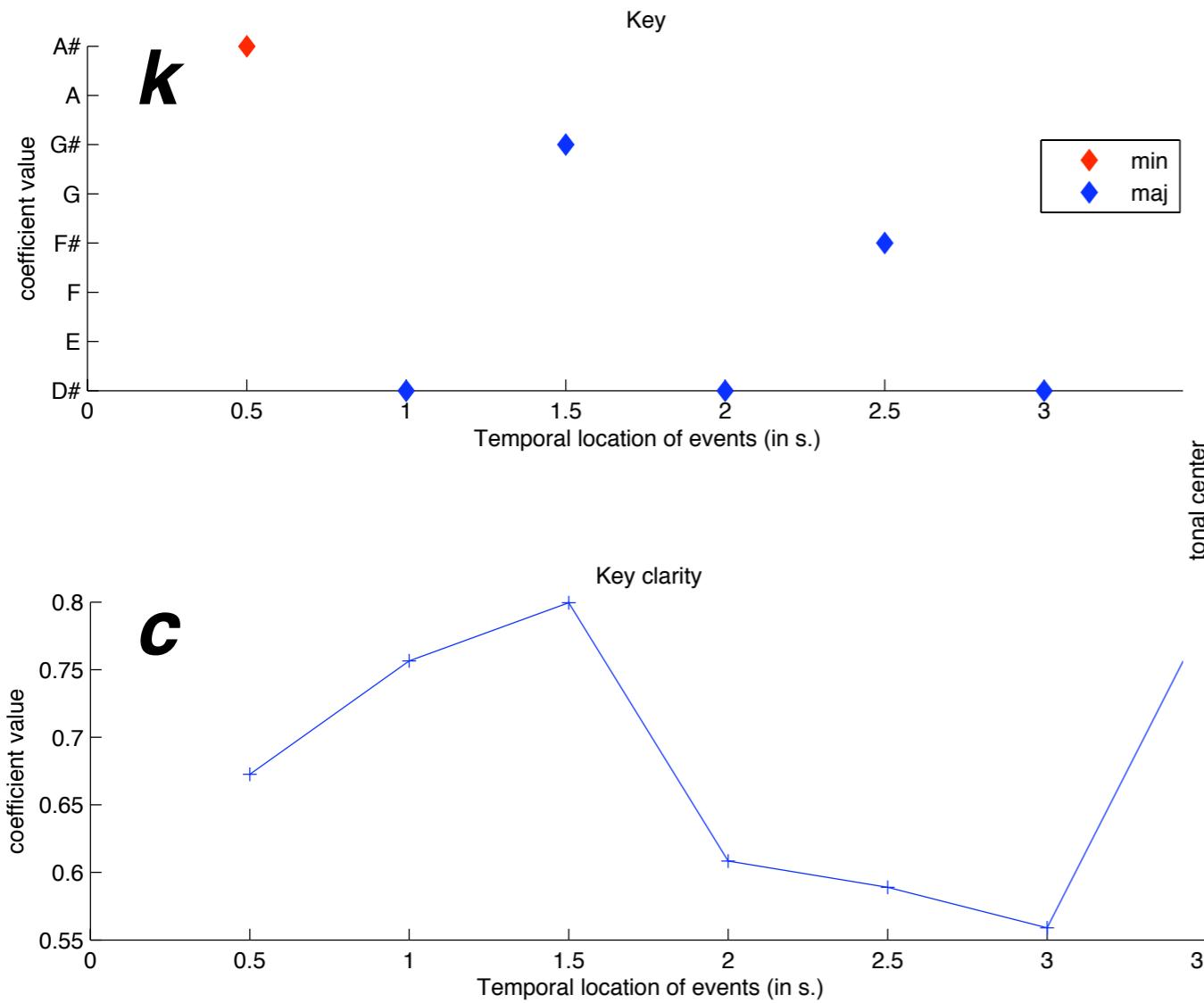
- *mus.key(..., 'Total', 1)*

Krumhansl, Cognitive foundations of musical pitch. Oxford UP, 1990.
Gomez, "Tonal description of polyphonic audio for music content processing," INFORMS Journal on Computing, 18-3, pp. 294–304, 2006.

mus.key

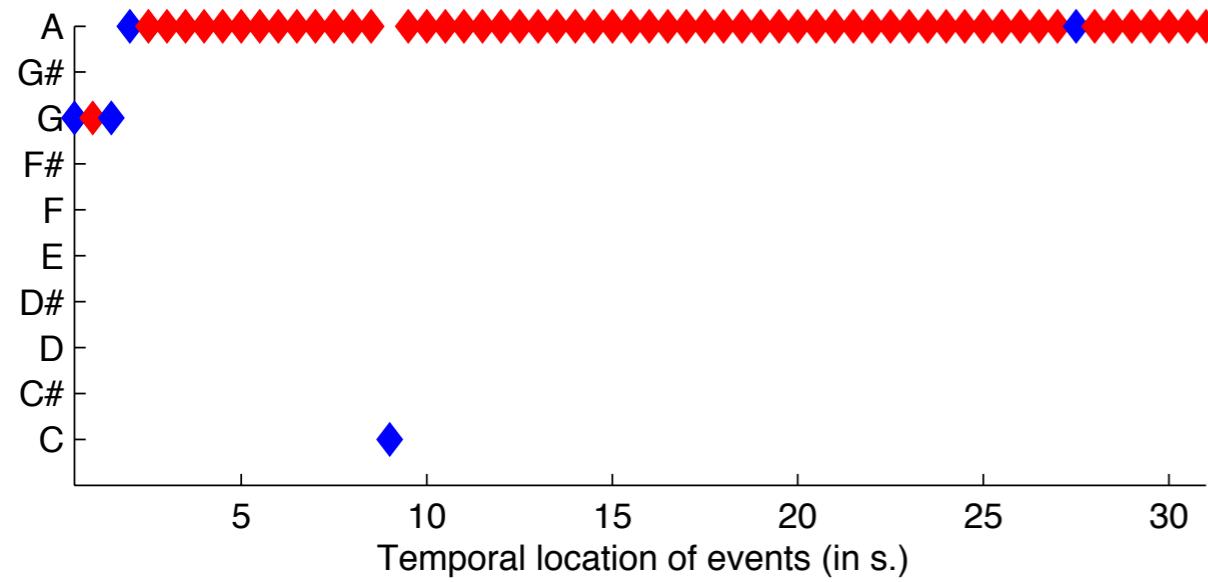
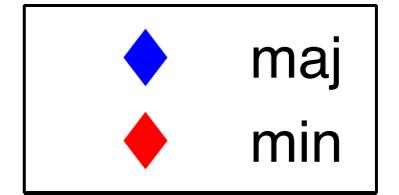
key estimation

- $[k \ c \ s] = \text{mus.key}(\dots, \text{'Frame'})$

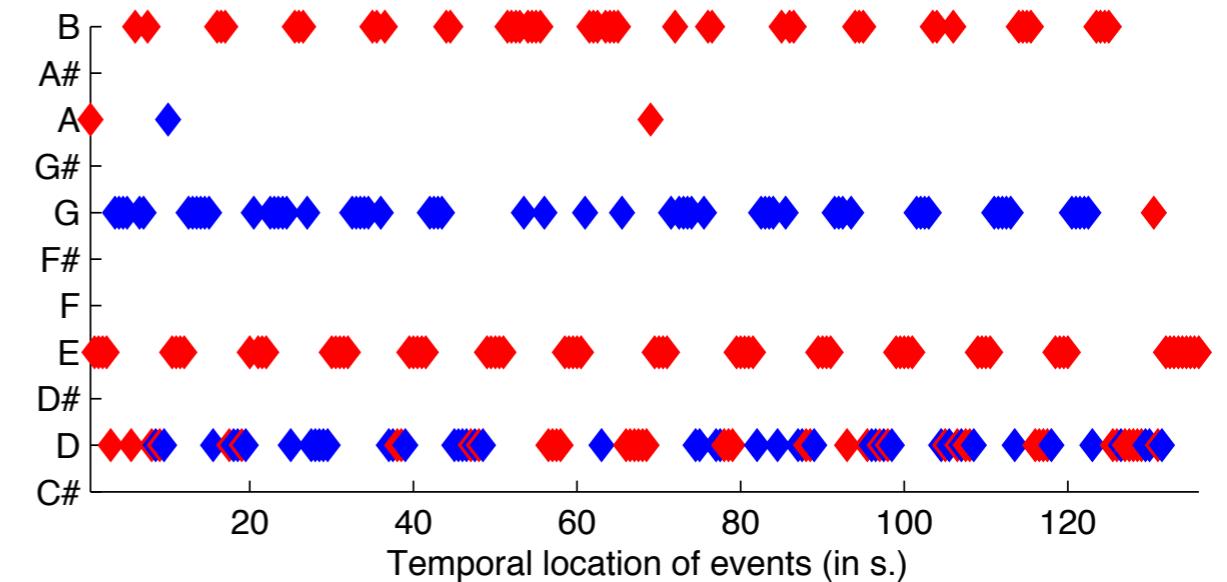


mus.key

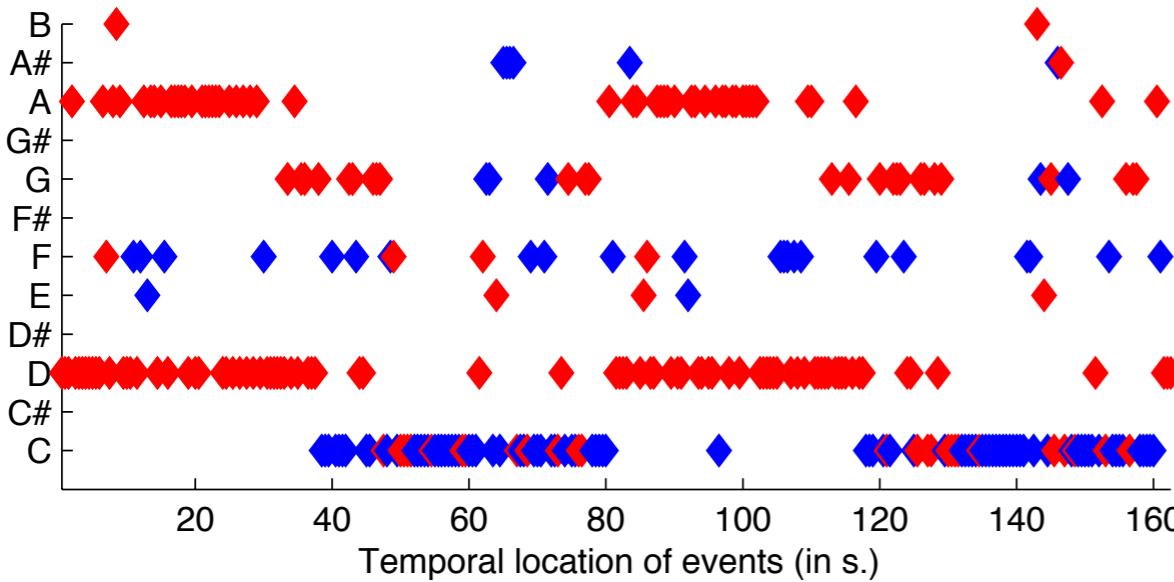
key estimation



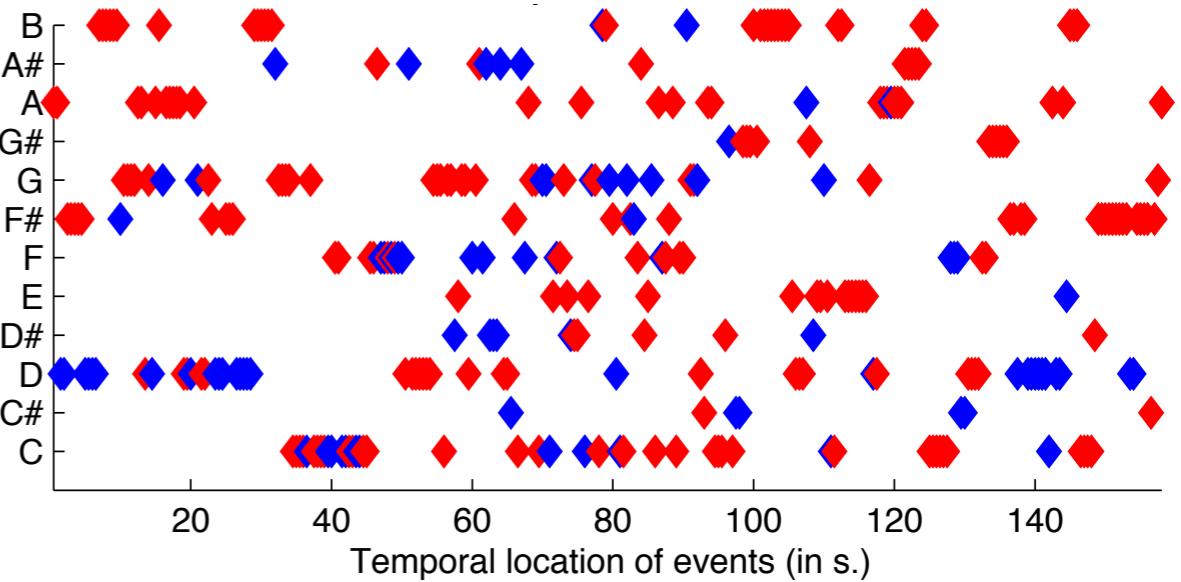
Monteverdi, *Hor che'l ciel e la terra*, 1st part



Tiersen, *Comptine d'un autre été : L'après-midi*



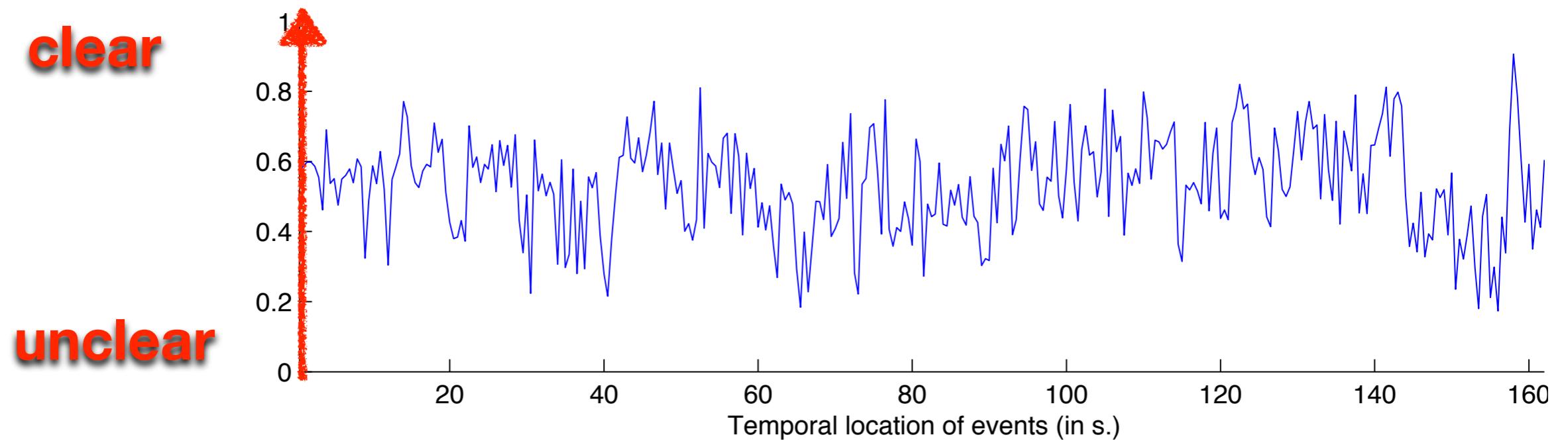
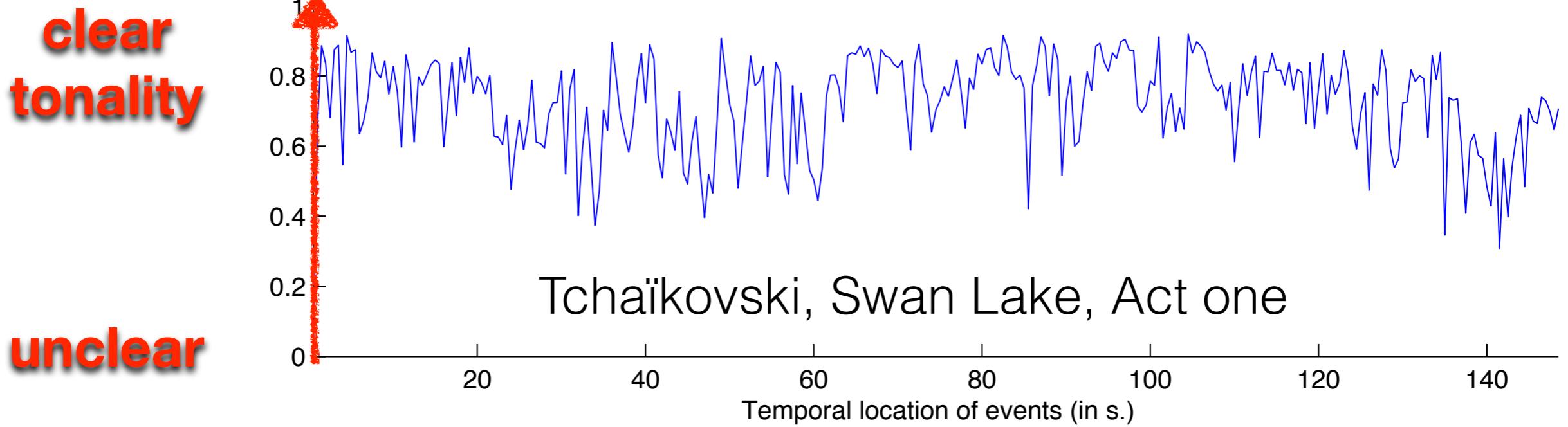
Beethoven, 9th Symphony, Scherzo



Schönberg, *Verklärte Nacht, Sehr Ruhig*

$[k \ c] = mus.key$

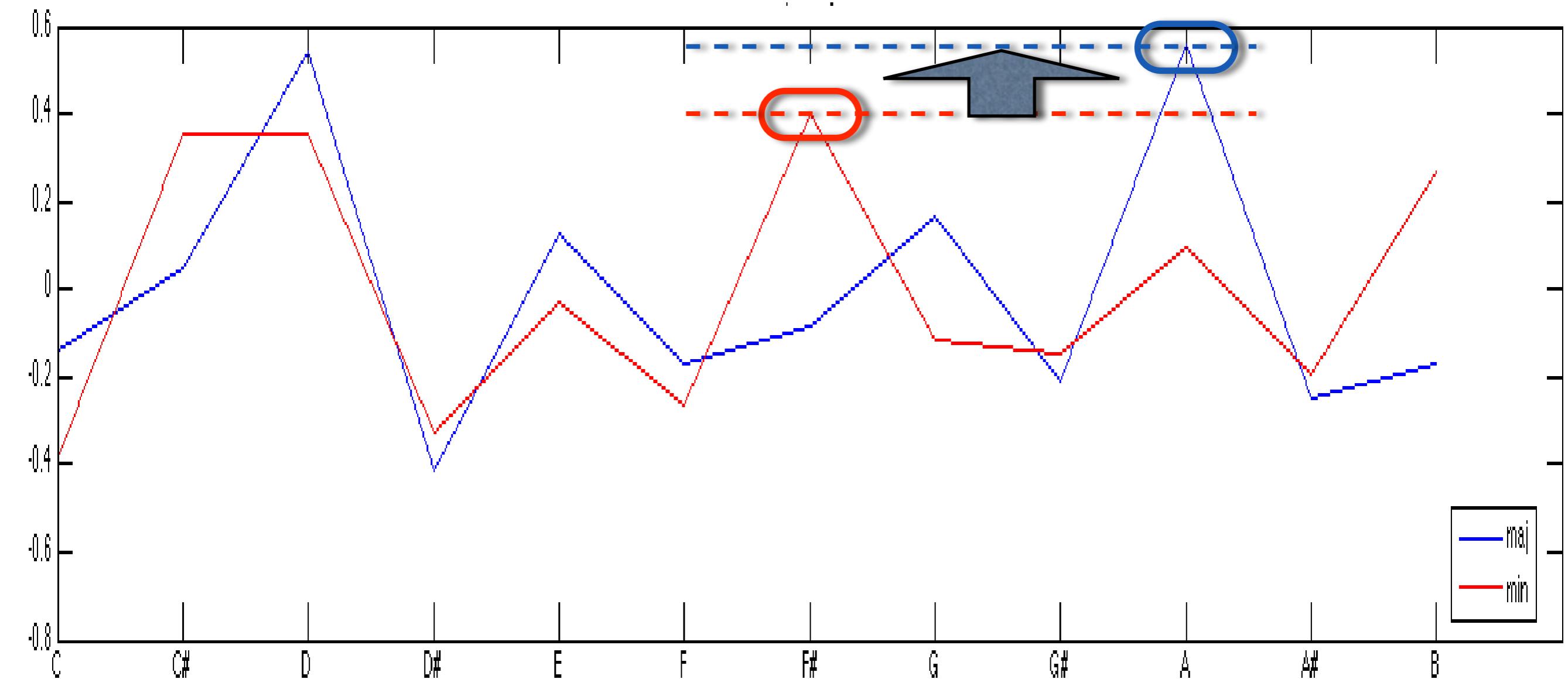
key clarity



Prokofiev, Violon concerto No. in D major, Scherzo: Vivacissimo

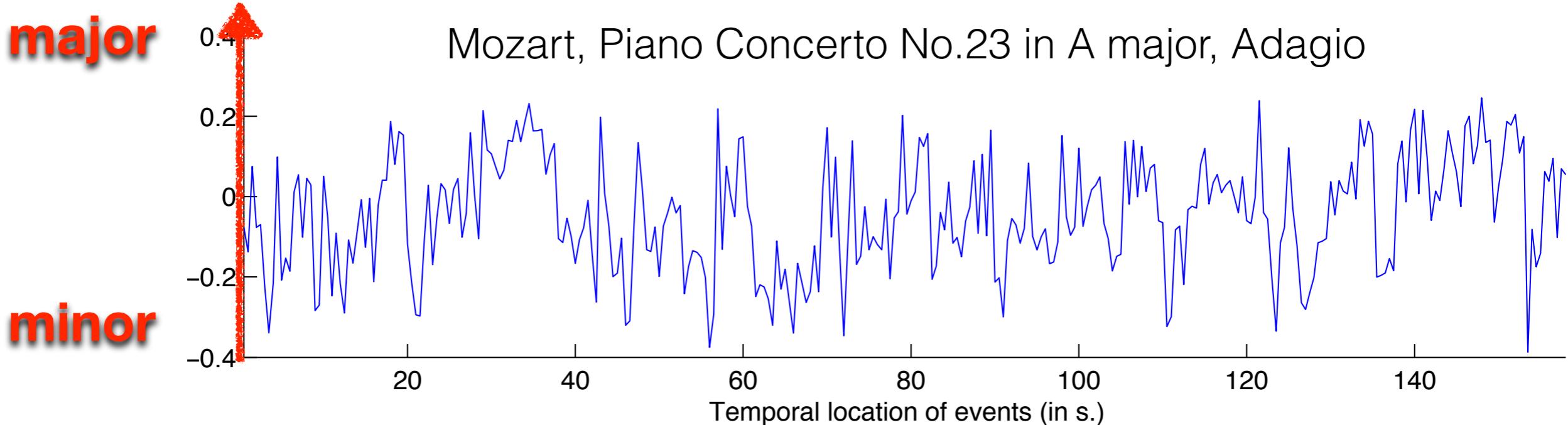
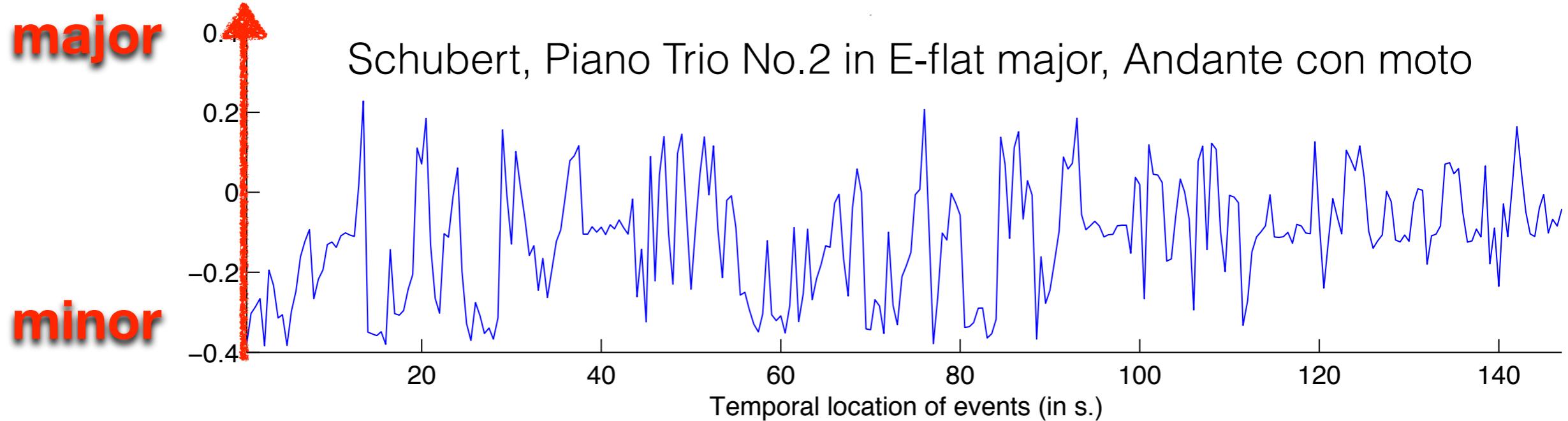
mus.mode

mode estimation



mus.mode

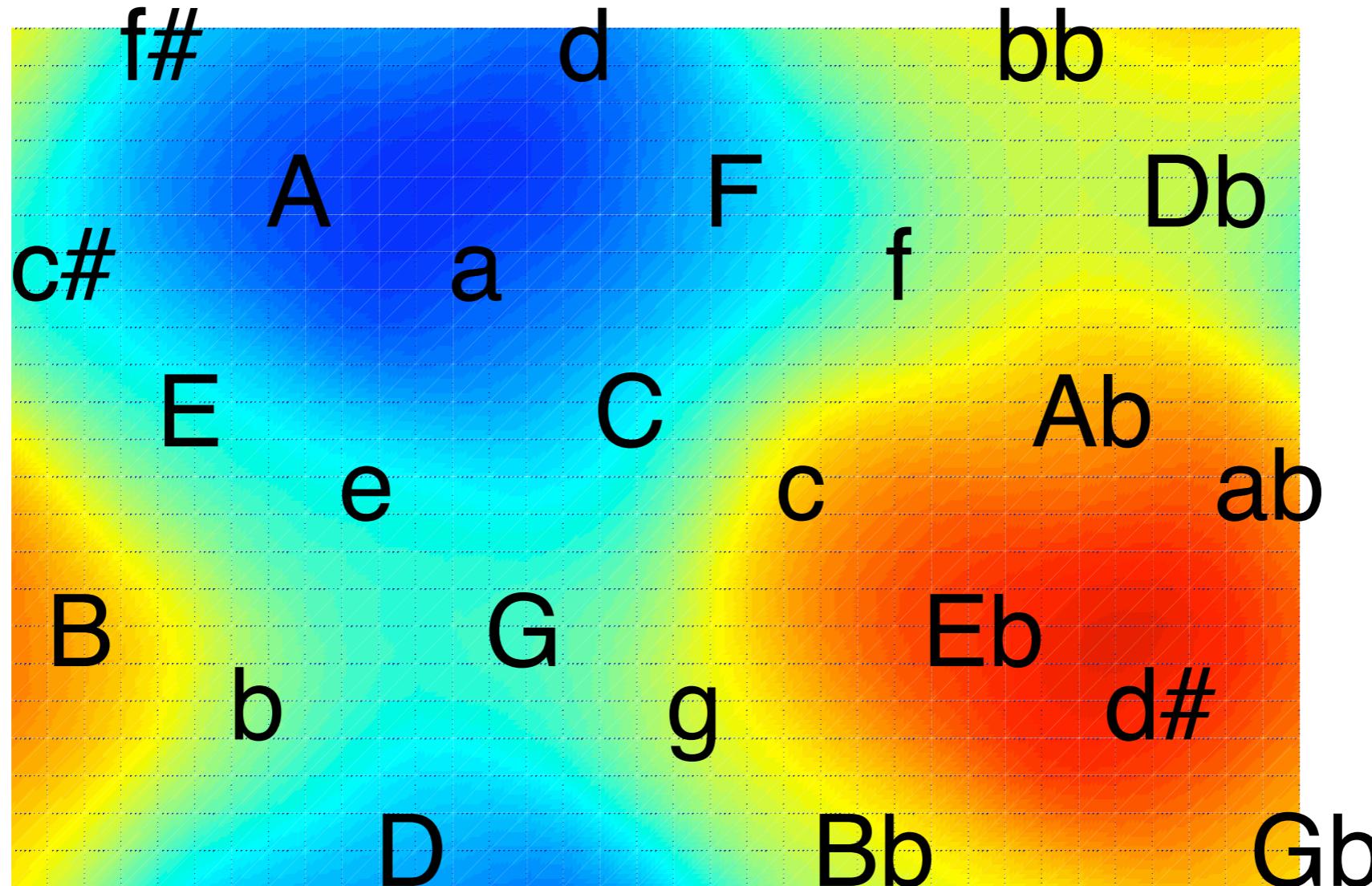
mode estimation



mus.keysom

self-organizing map

Self-organizing map projection of chromagram

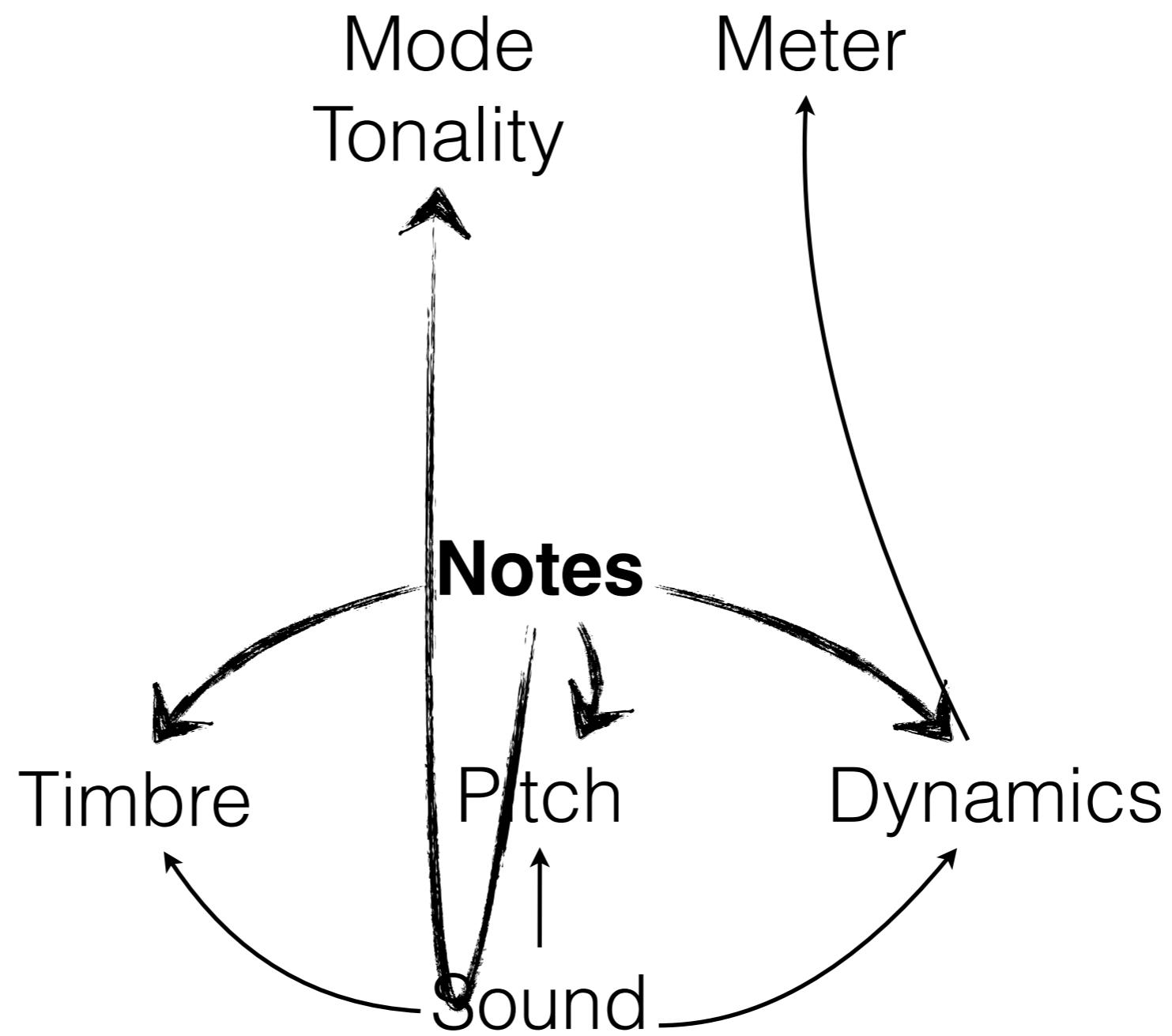


Toiviainen & Krumhansl, “Measuring and modeling real-time responses to music: The dynamics of tonality induction”, Perception 32-6, pp. 741–766, 2003.

*Structural
levels*

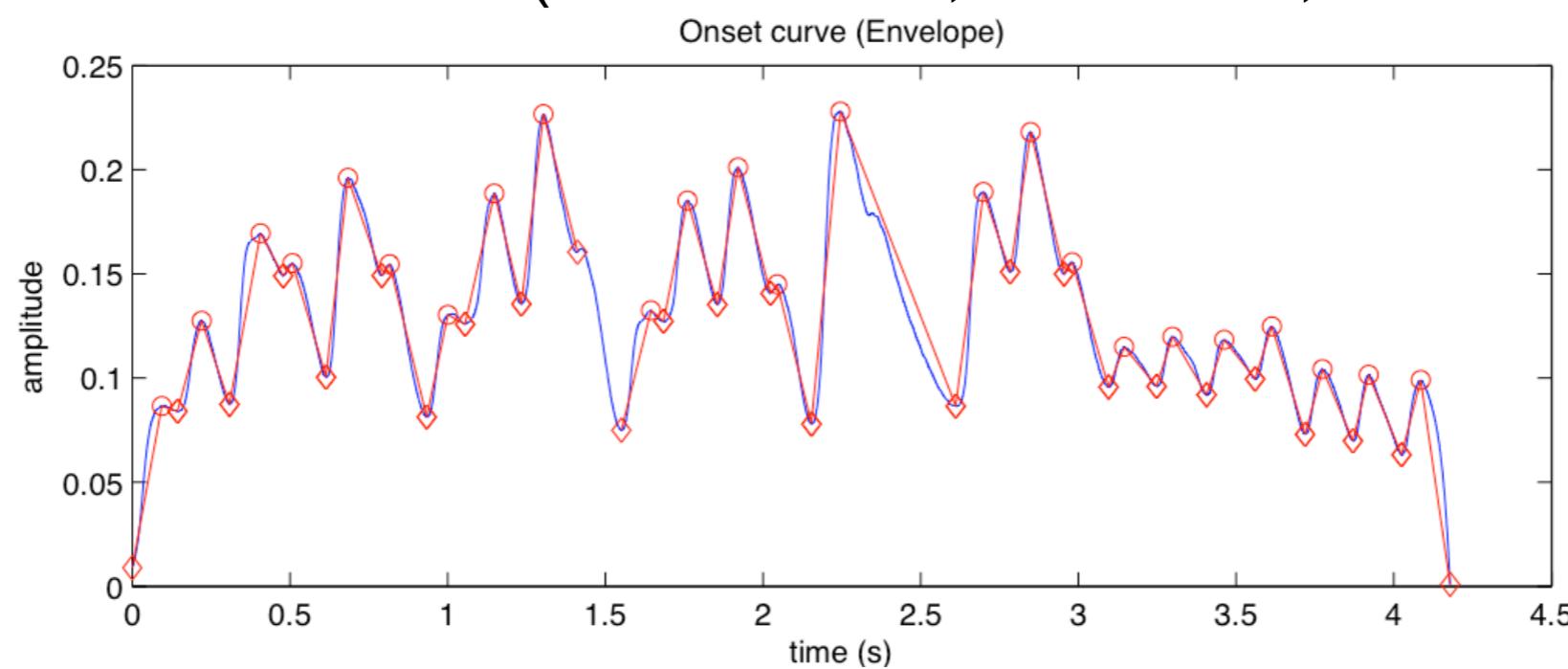
Symbolic level

Audio level

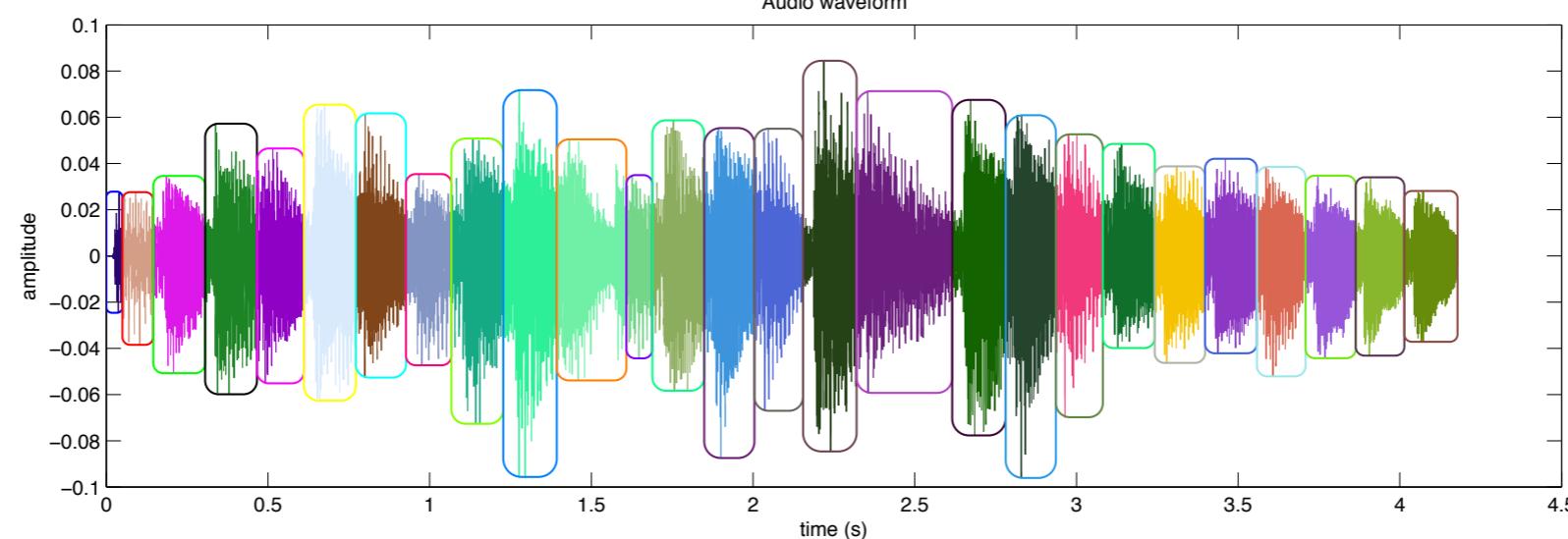


onset-based segmentation

`o = aud.onsets('audiofile', 'Attack', 'Release')`

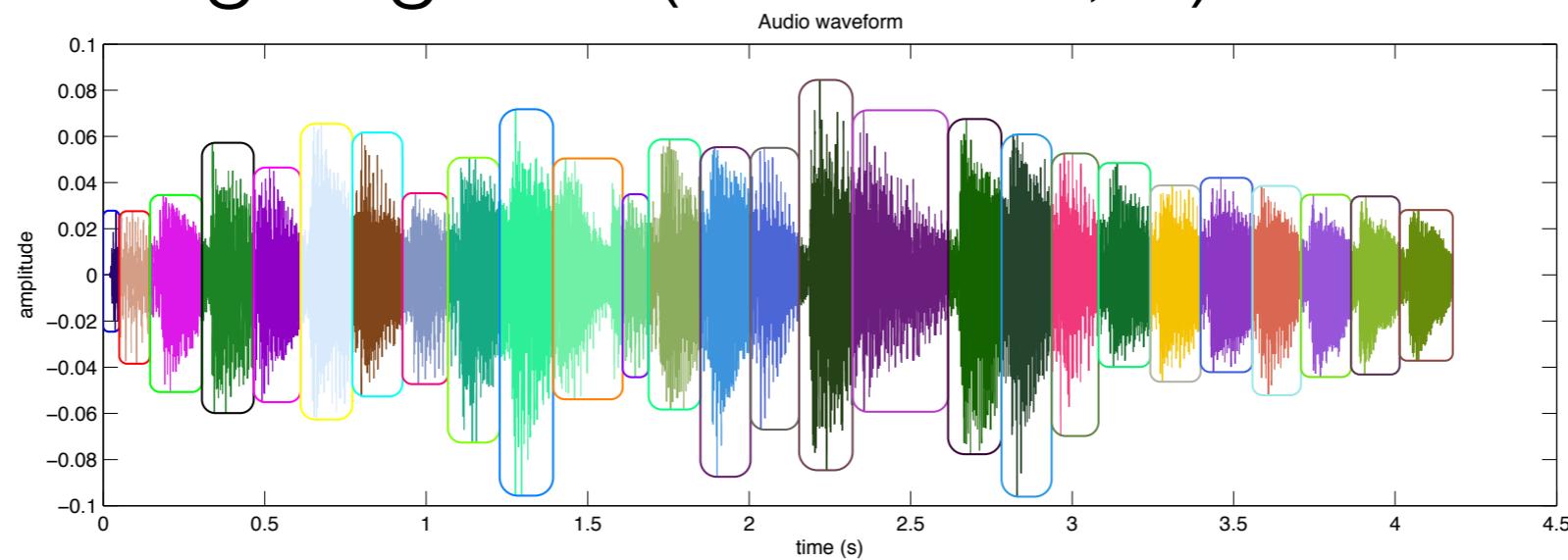


`sig.segment('audiofile', o)`



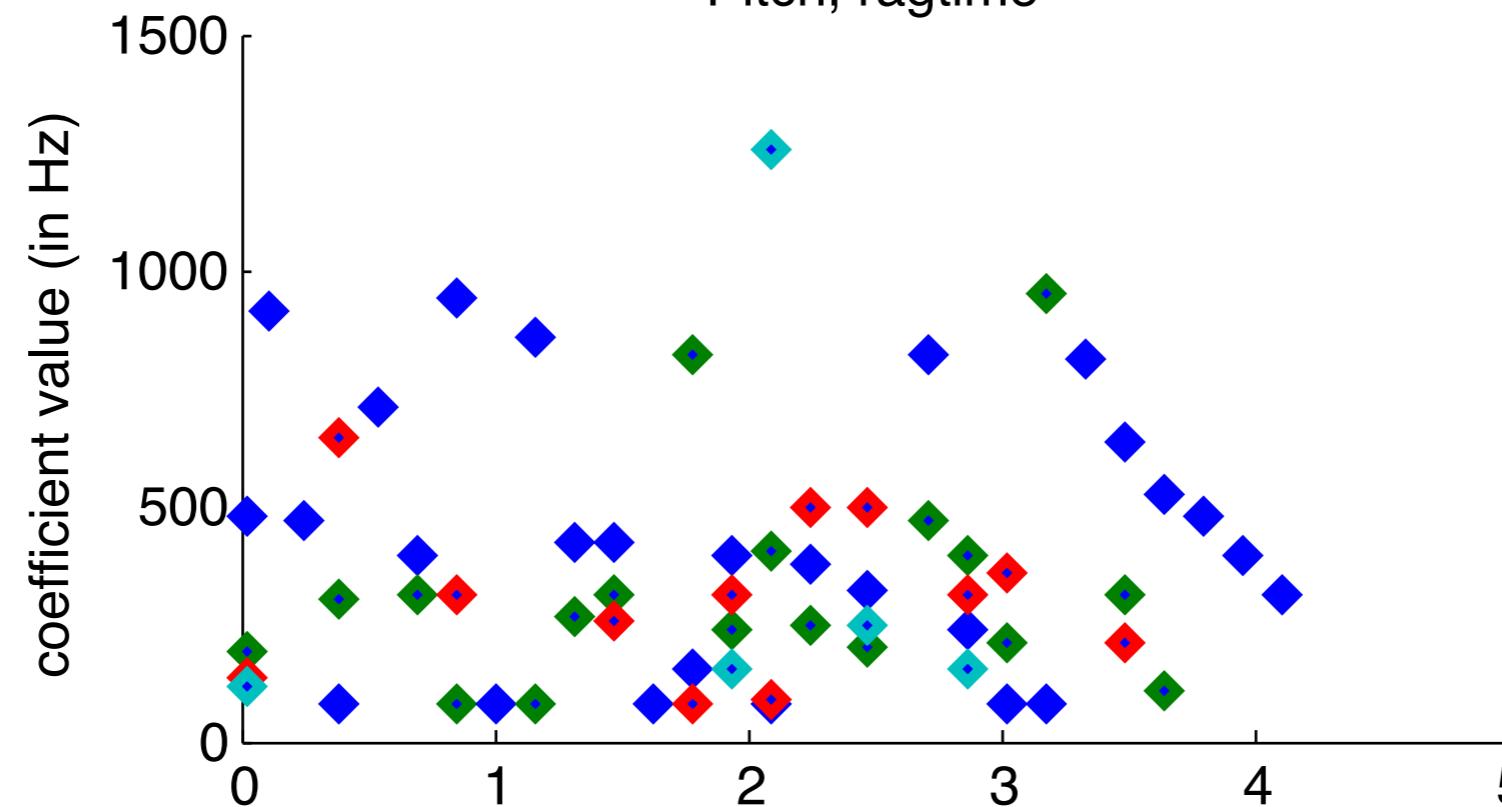
onset-based segmentation

`s = sig.segment('audiofile', o)`



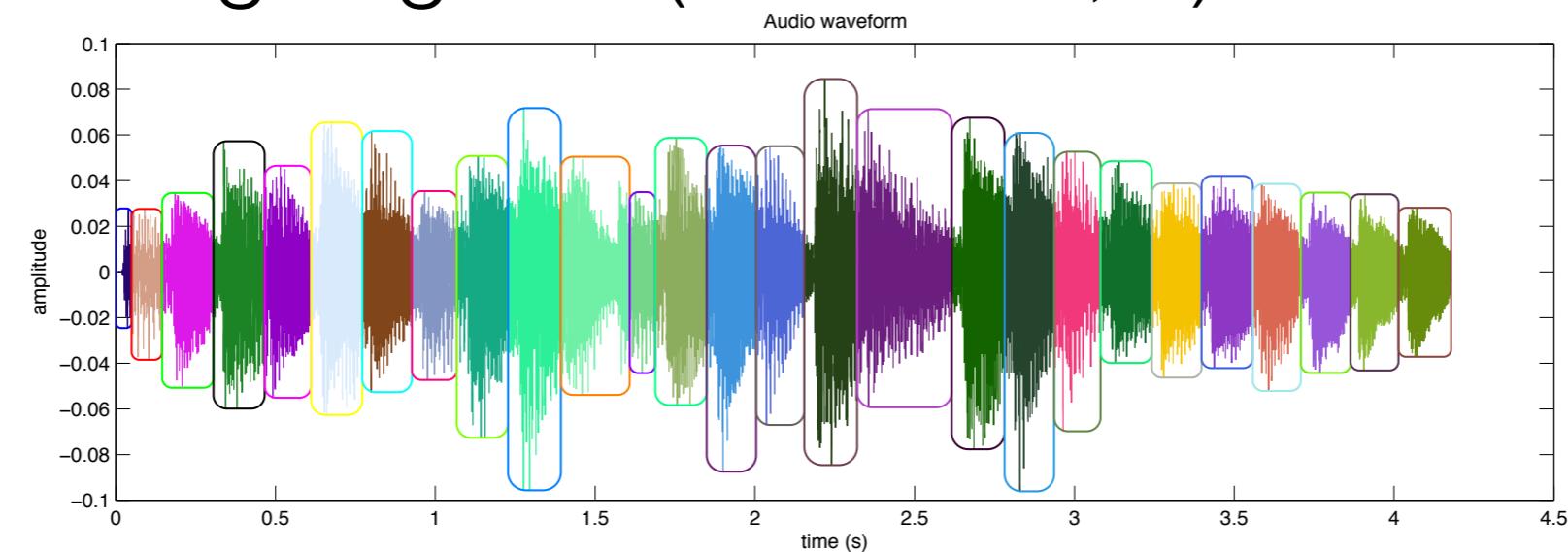
`mus.pitch(s)`

Pitch, ragtime



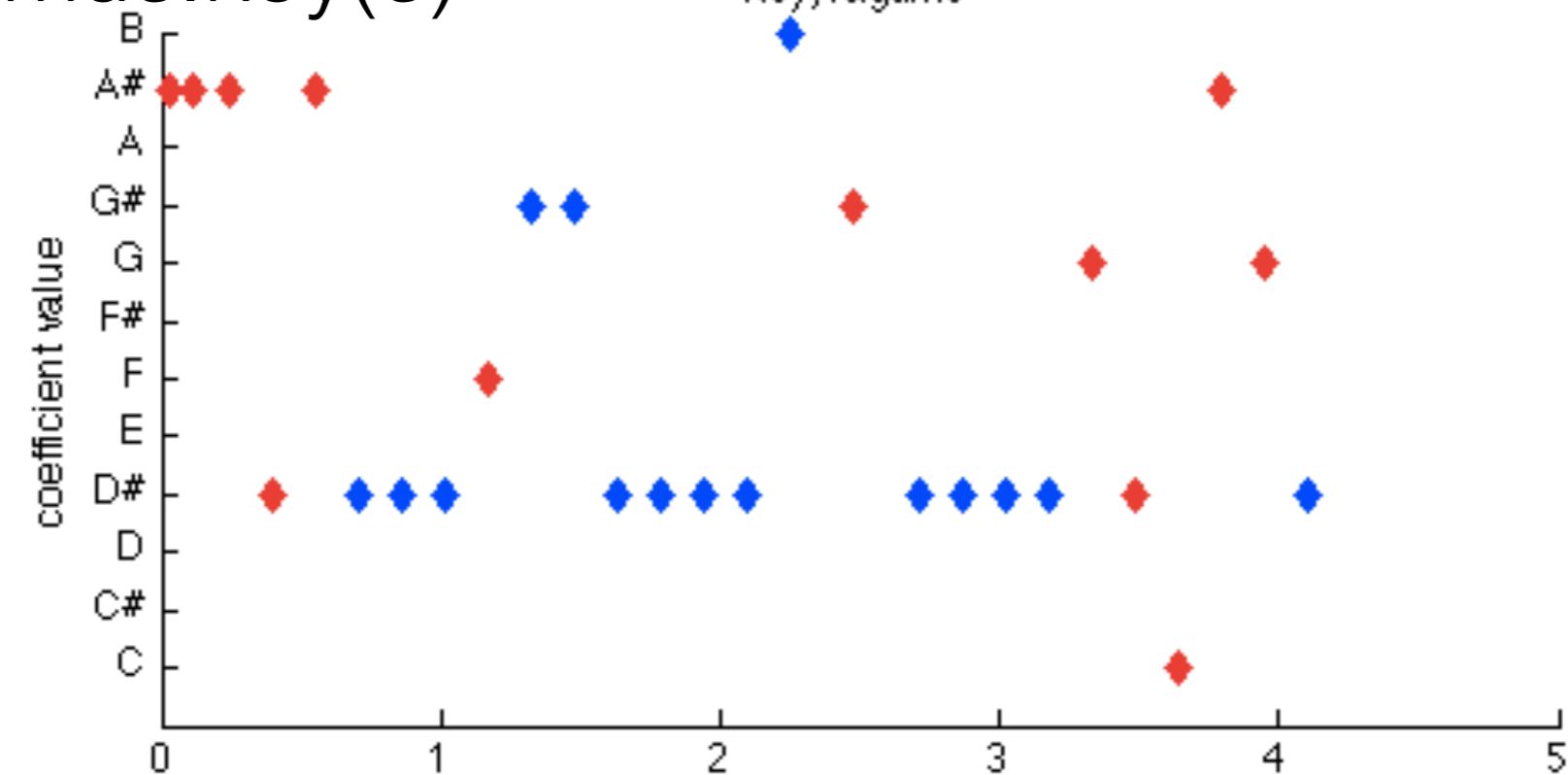
onset-based segmentation

`s = sig.segment('audiofile', o)`



`mus.key(s)`

Key, ragtime



*Structural
levels*

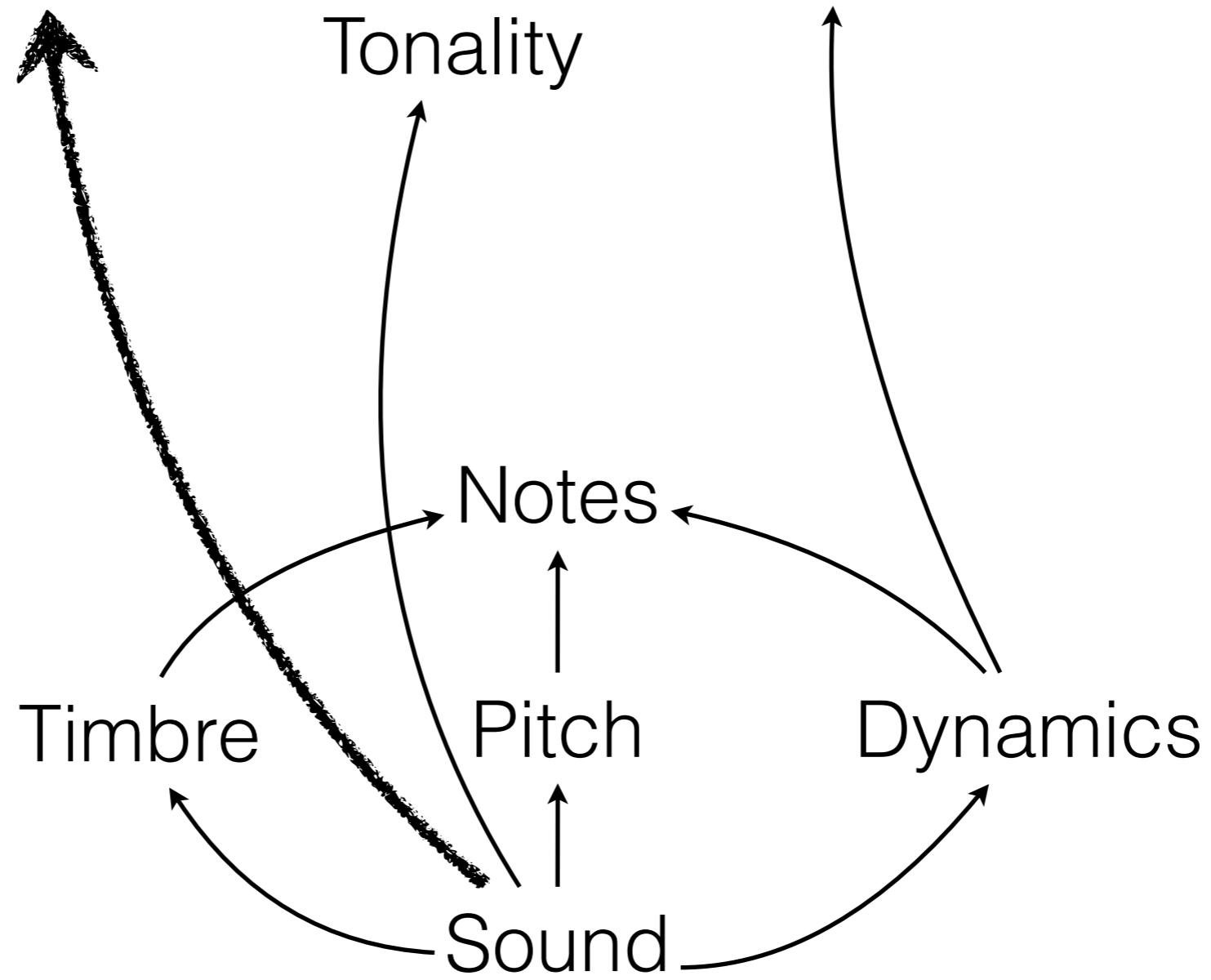
Groups

Mode
Tonality

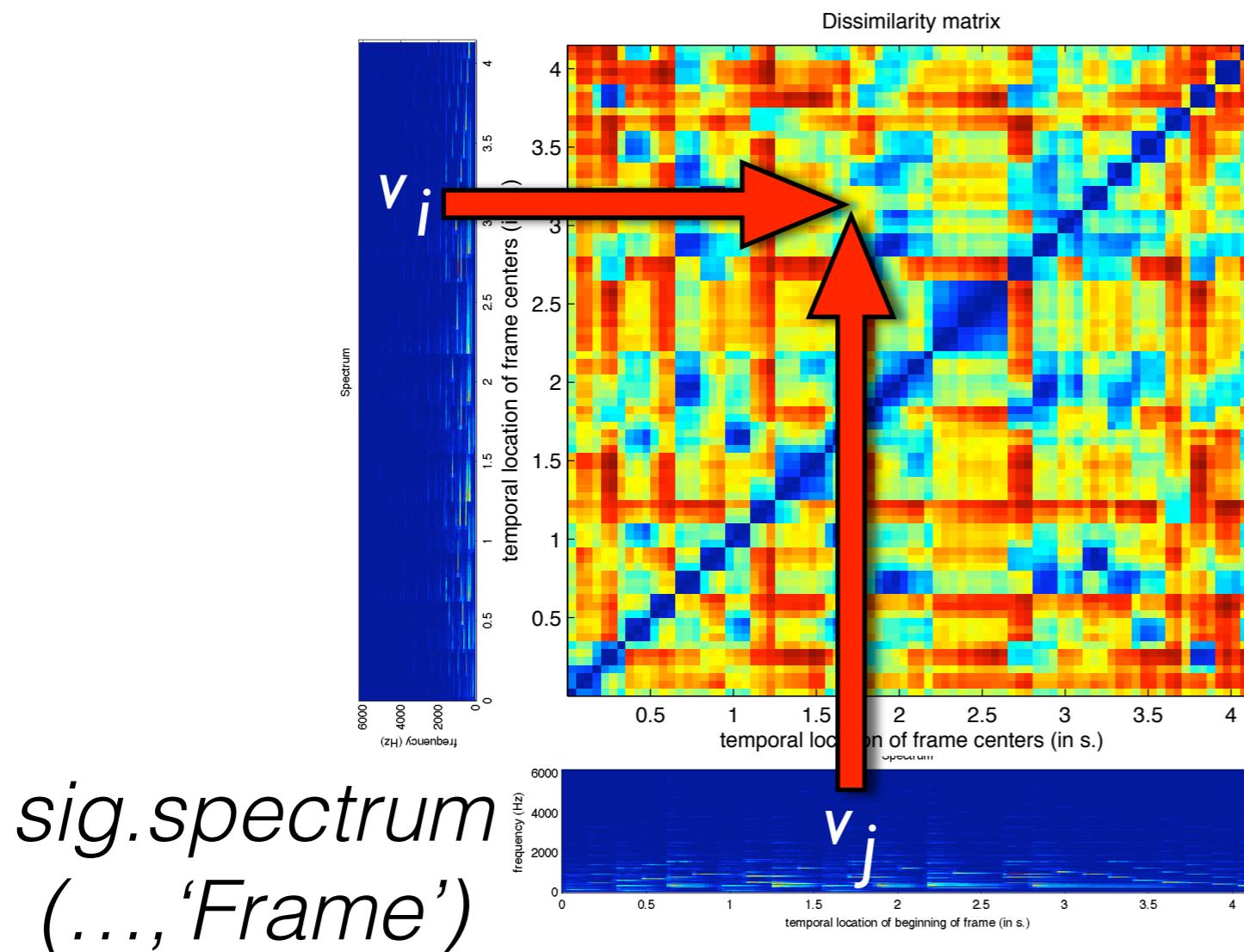
Meter

Symbolic level

Audio level



sig.simatrix dissimilarity matrix



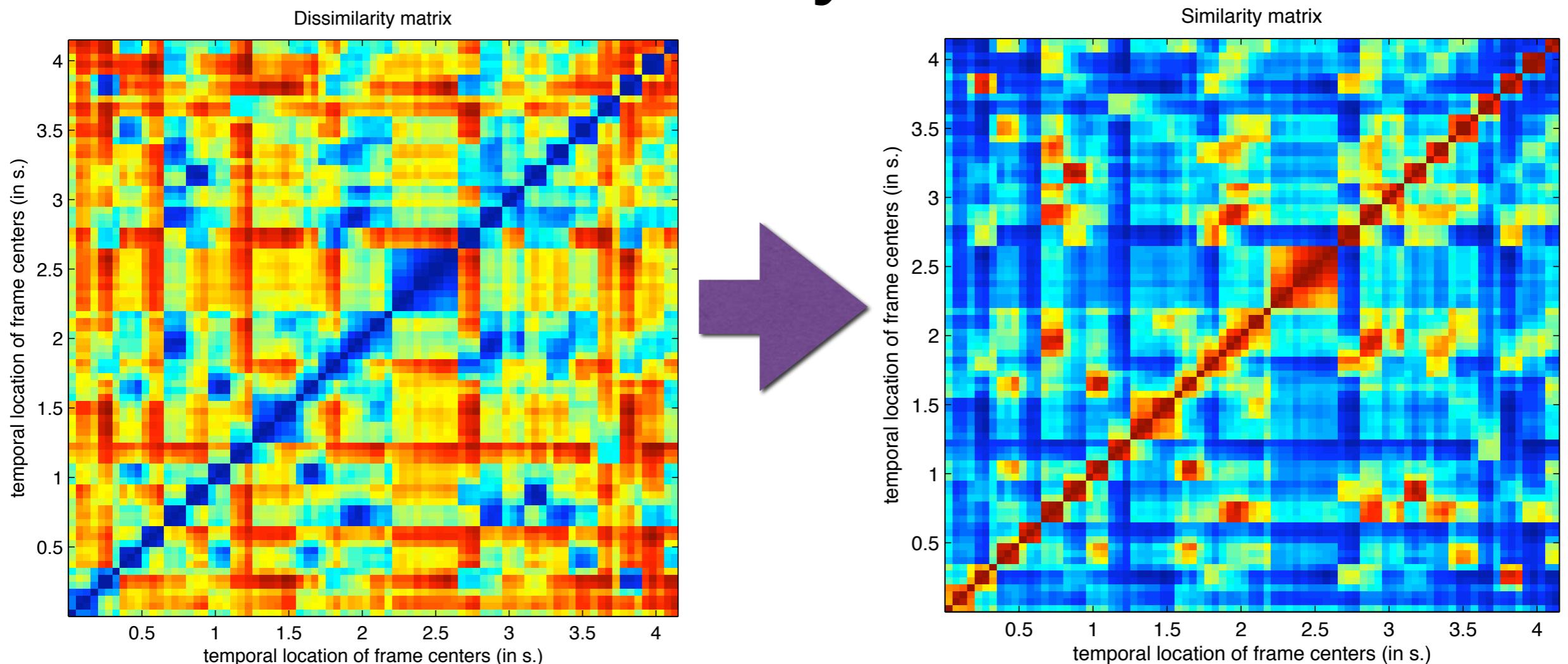
*sig.spectrum
(..., 'Frame')*

- *sig.simatrix(a,
'Dissimilarity')*
- *sig.simatrix(...,
'Distance', '**cosine**')*

$$d_{cos}(v_i, v_j) = \frac{\langle v_i, v_j \rangle}{|v_i||v_j|}$$

sig.simatrix

similarity matrix

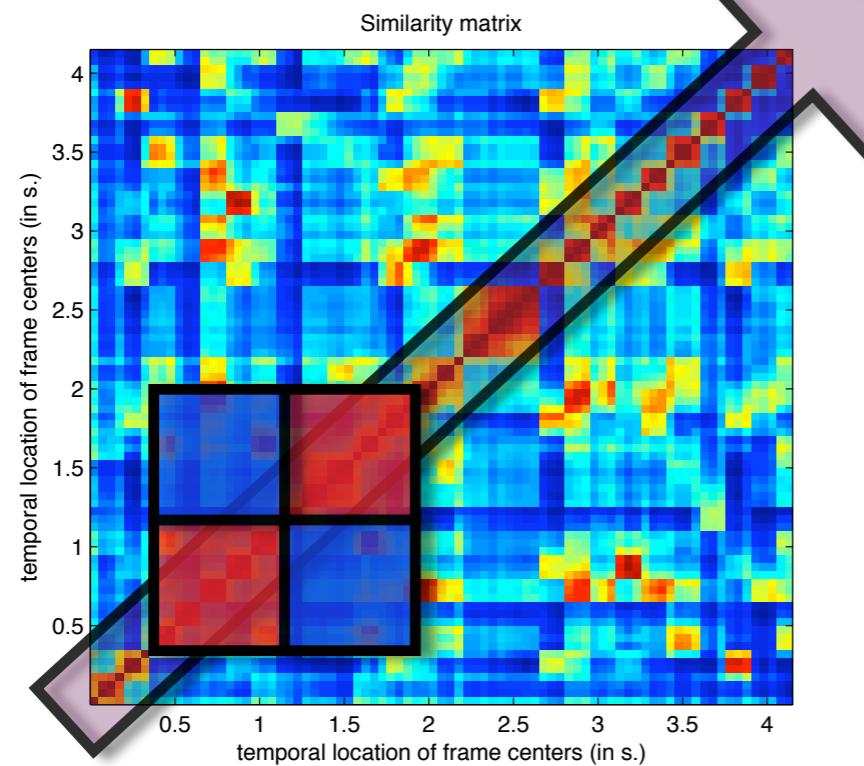


- *sig.simatrix(a, ‘**Similarity**’, ‘*exponential*’)*
 $d_{exp}(v_i, v_j) = \exp(-d_{cos}(v_i, v_j))$

Foote, Cooper. “Media Segmentation using Self-Similarity Decomposition”, SPIE Storage and Retrieval for Multimedia Databases, 5021, 167-75.

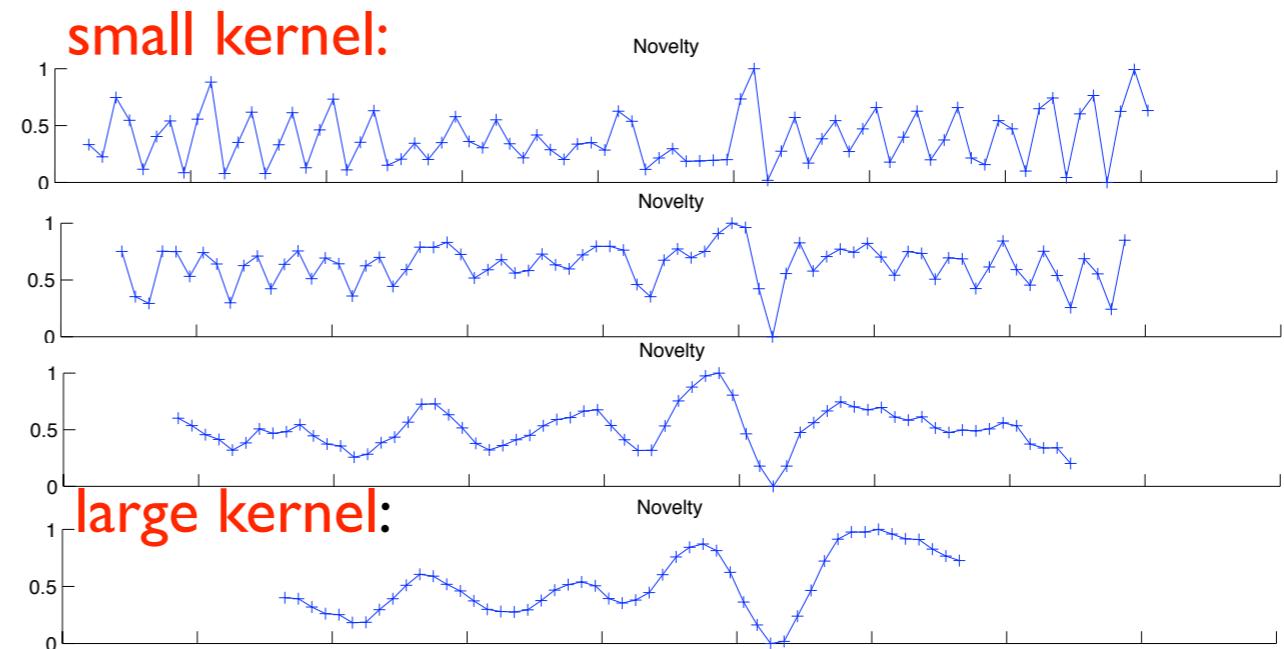
aud.novelty novelty curve

sig.simatrix(a)



aud.novelty(a, 'KernelSize', s)

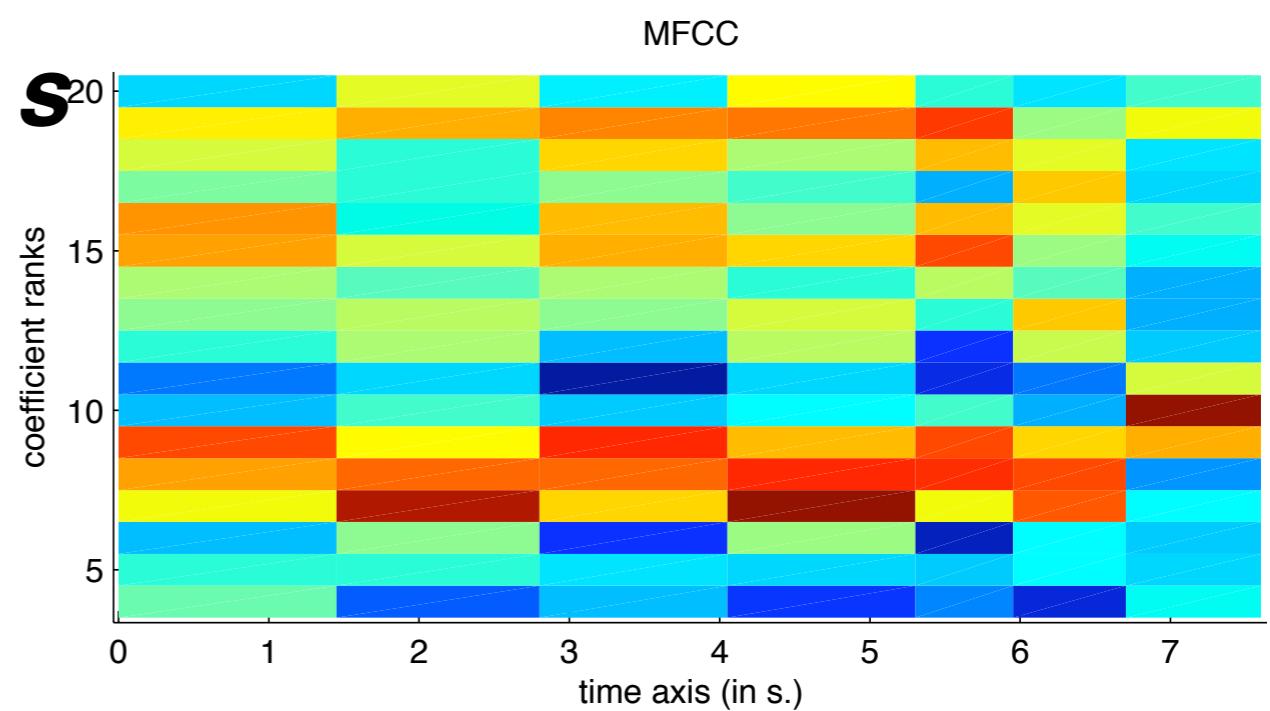
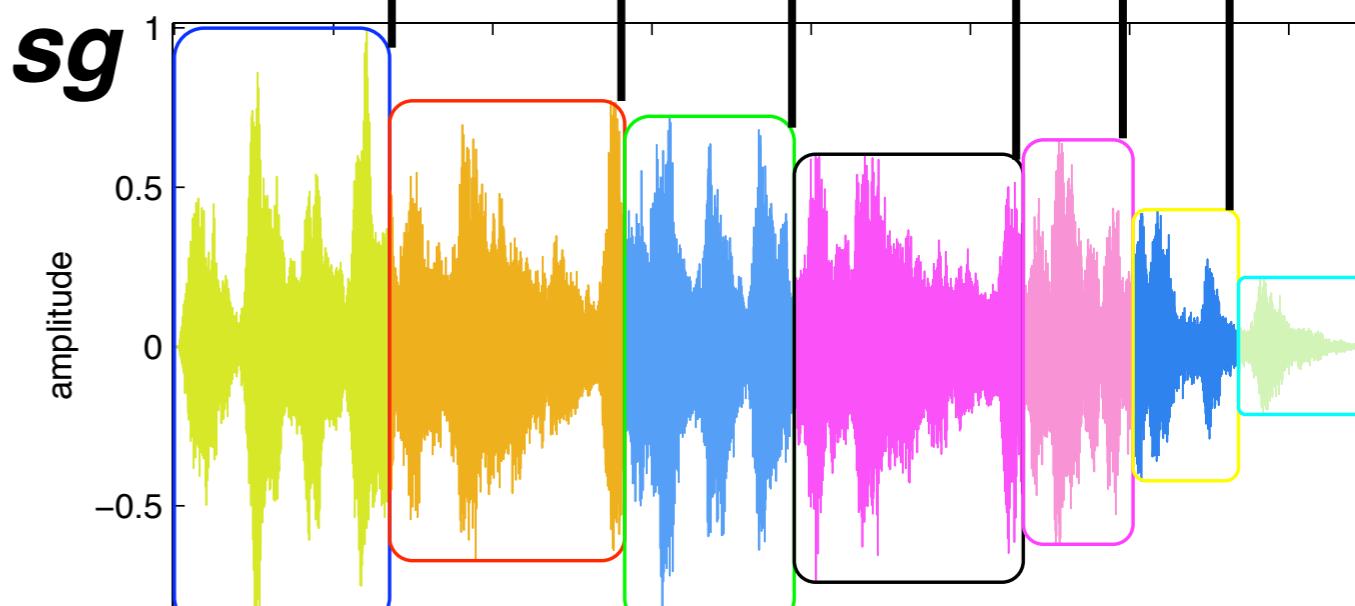
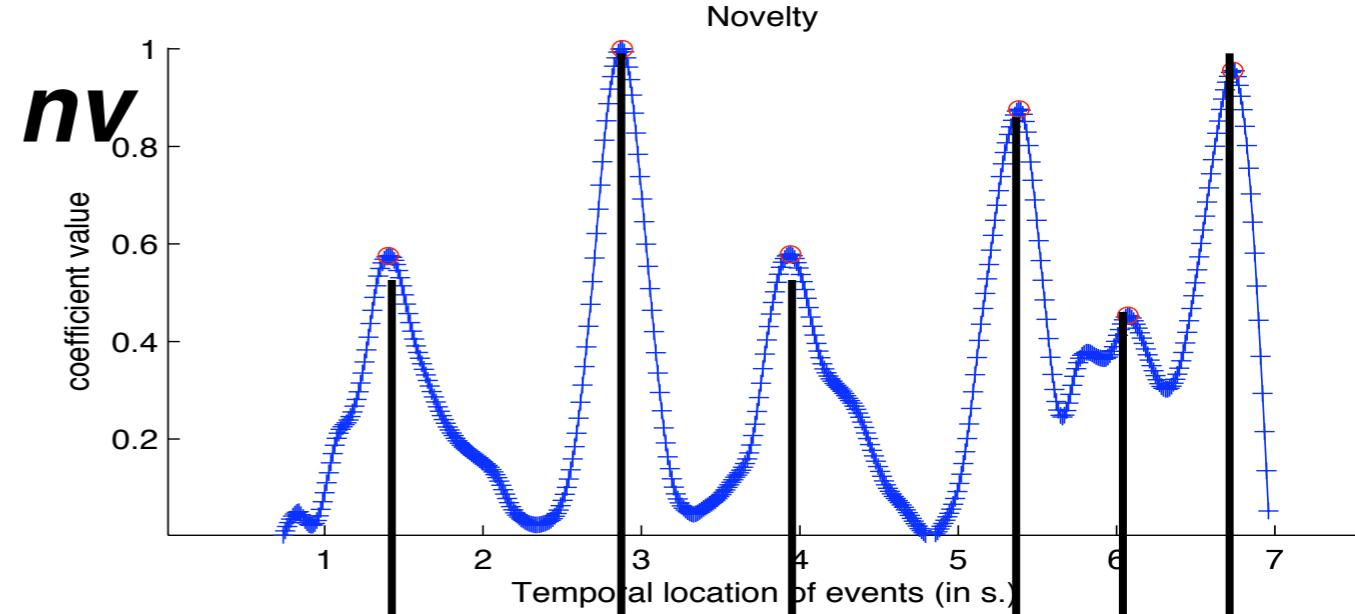
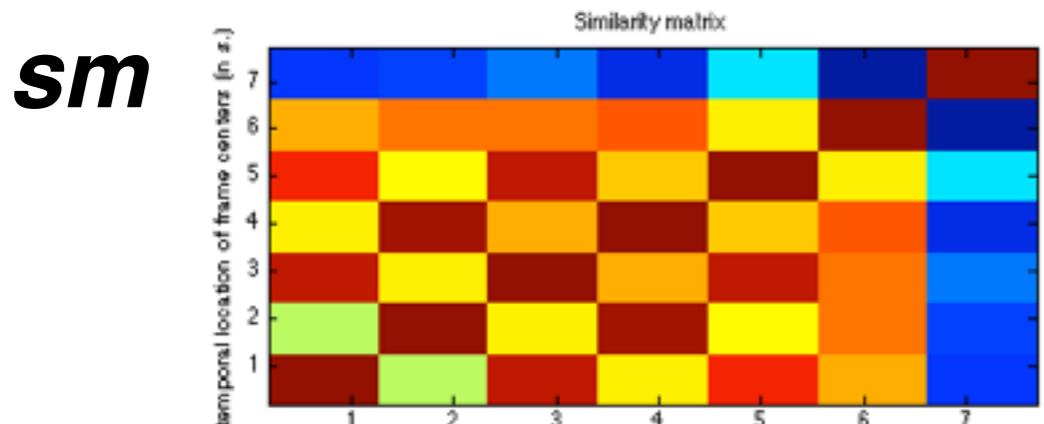
$s = 128$ samples



Convolution with Gaussian
checkerboard kernel

aud.segment novelty-based segmentation

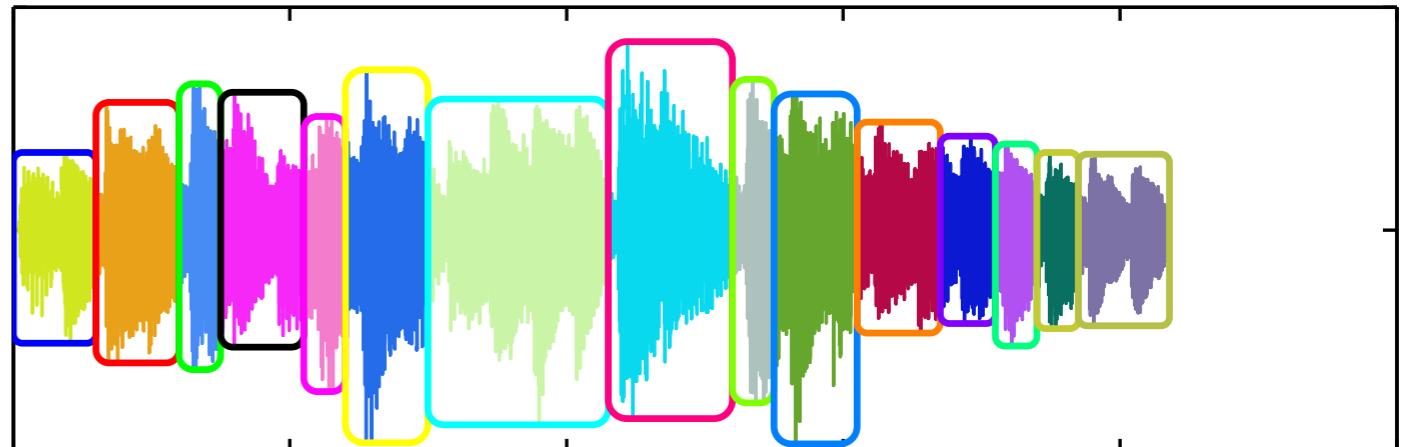
- $nv = aud.novelty(sm)$
- $sg = sig.segment('mysong', nv)$
- $sg = sig.segment('mysong')$
- $aud.play(sg)$
- $s = aud.mfcc(sg)$
- $sm = sig.simatrix(s)$



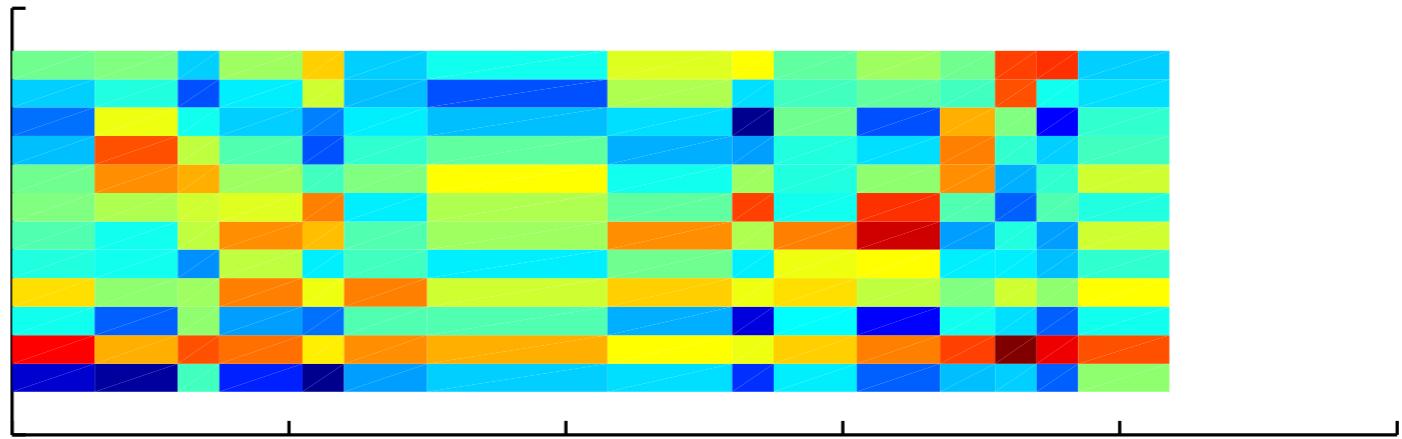
sig.cluster

clustering of audio segments

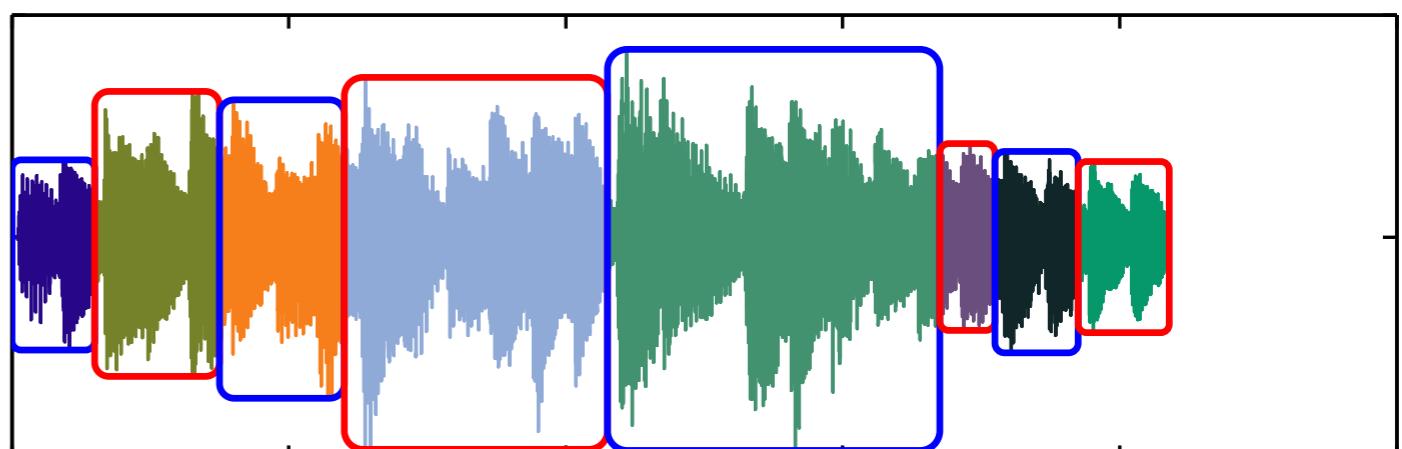
- $sg = aud.segment(a)$



- $cc = aud.mfcc(sg)$



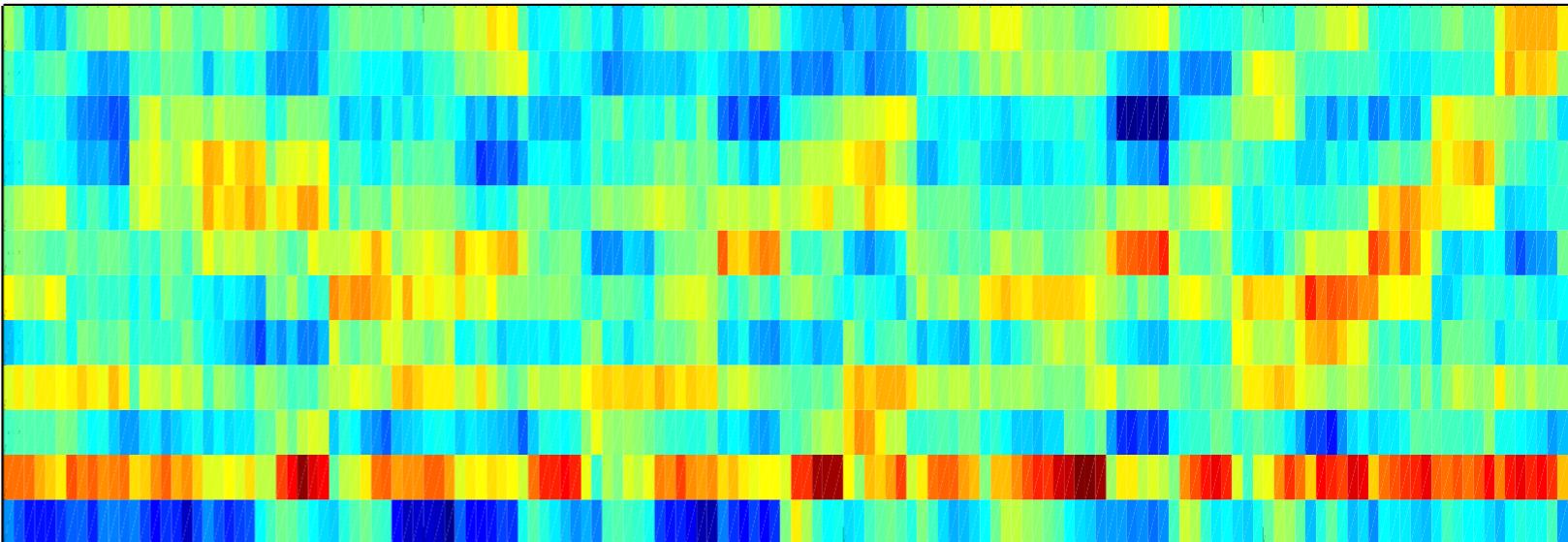
- $sig.cluster(sg, cc)$



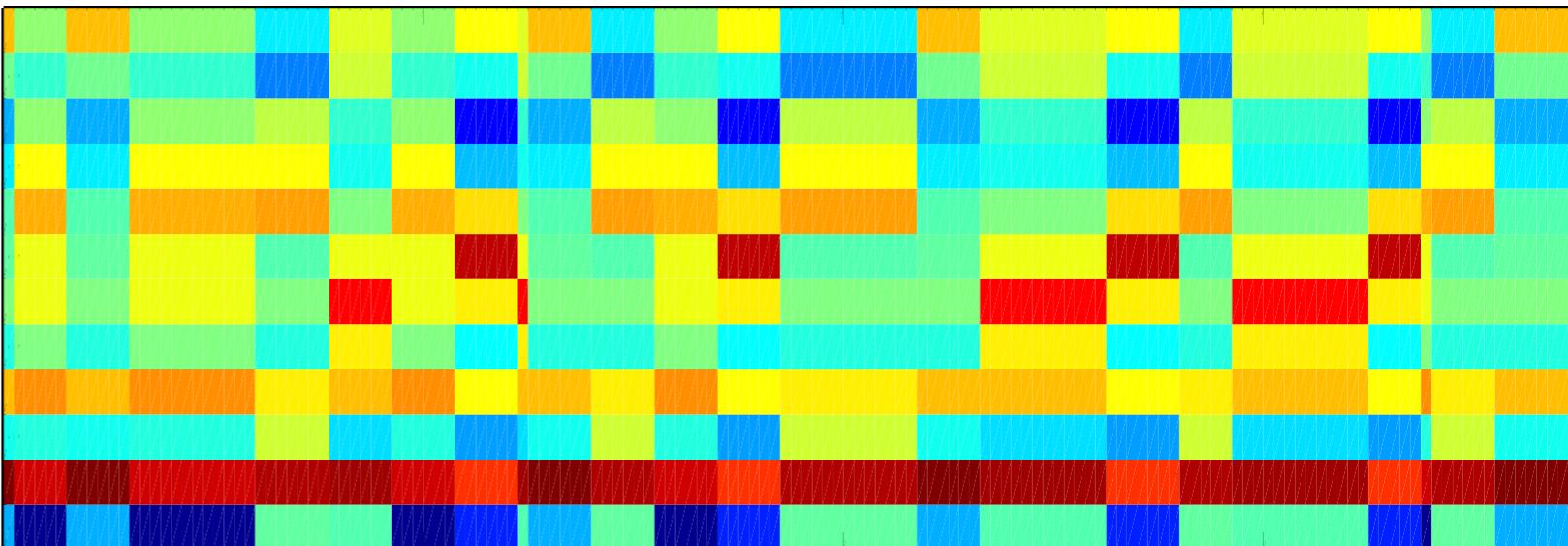
sig.cluster

clustering of frame-decomposed feature

- $cc = aud.mfcc(\dots, 'Frame')$



- $sig.cluster(cc, 4)$

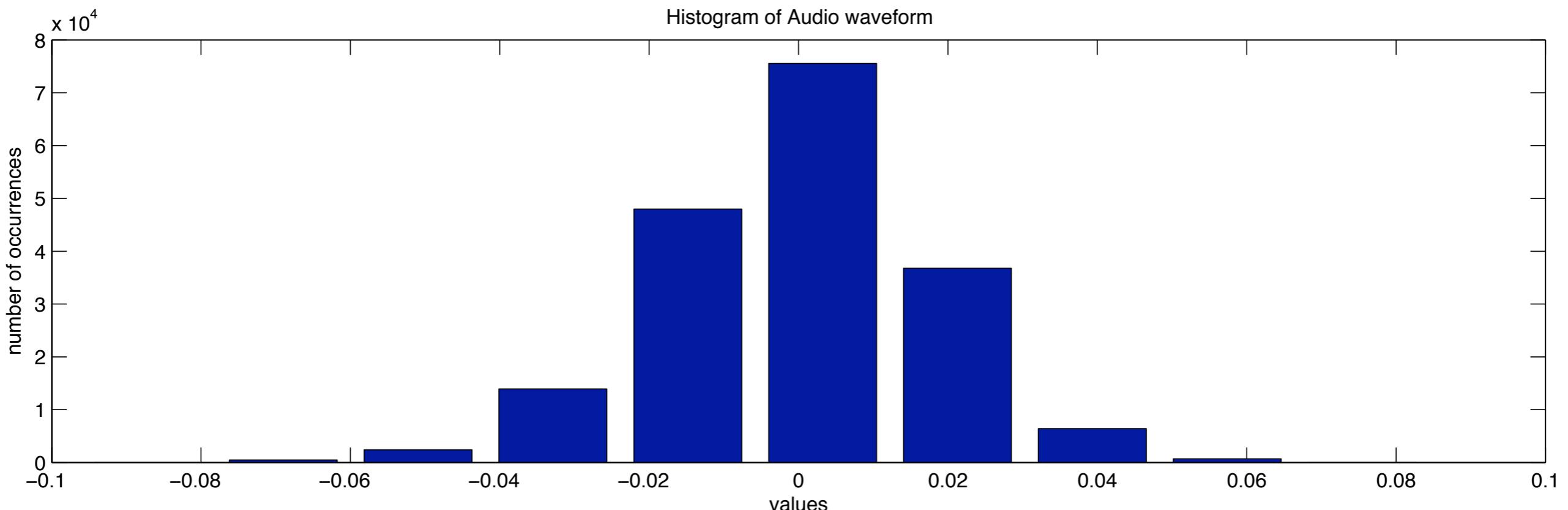


sig.stat

basic statistics

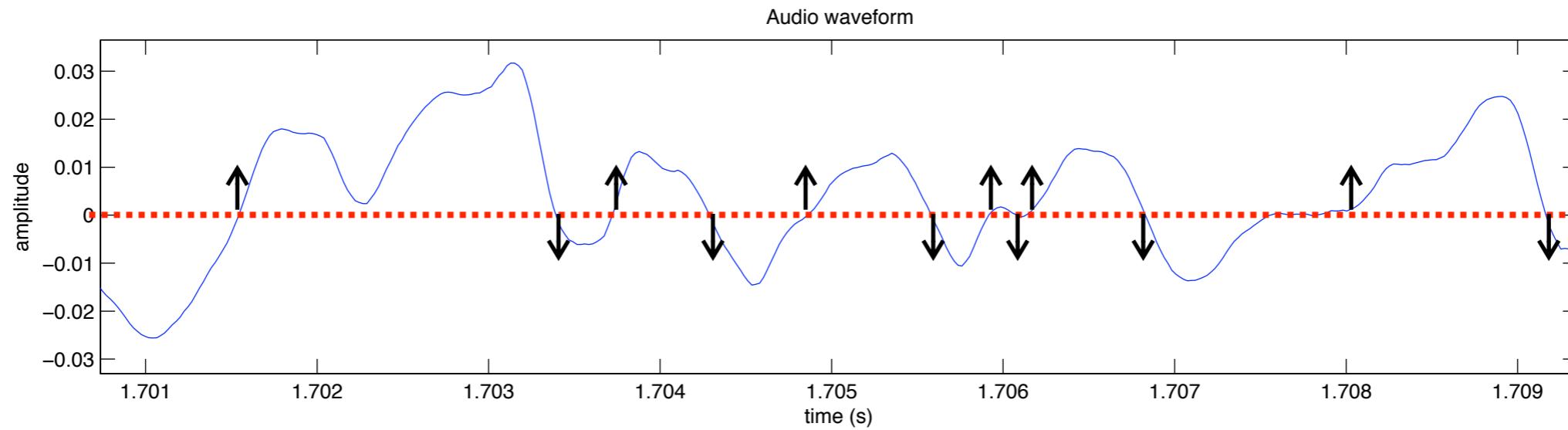
- mean
- standard deviation
- temporal slope
- main periodicity:
 - frequency
 - amplitude
 - periodicity entropy

sig.histo histogram



sig.histo(..., 'Number', 10)

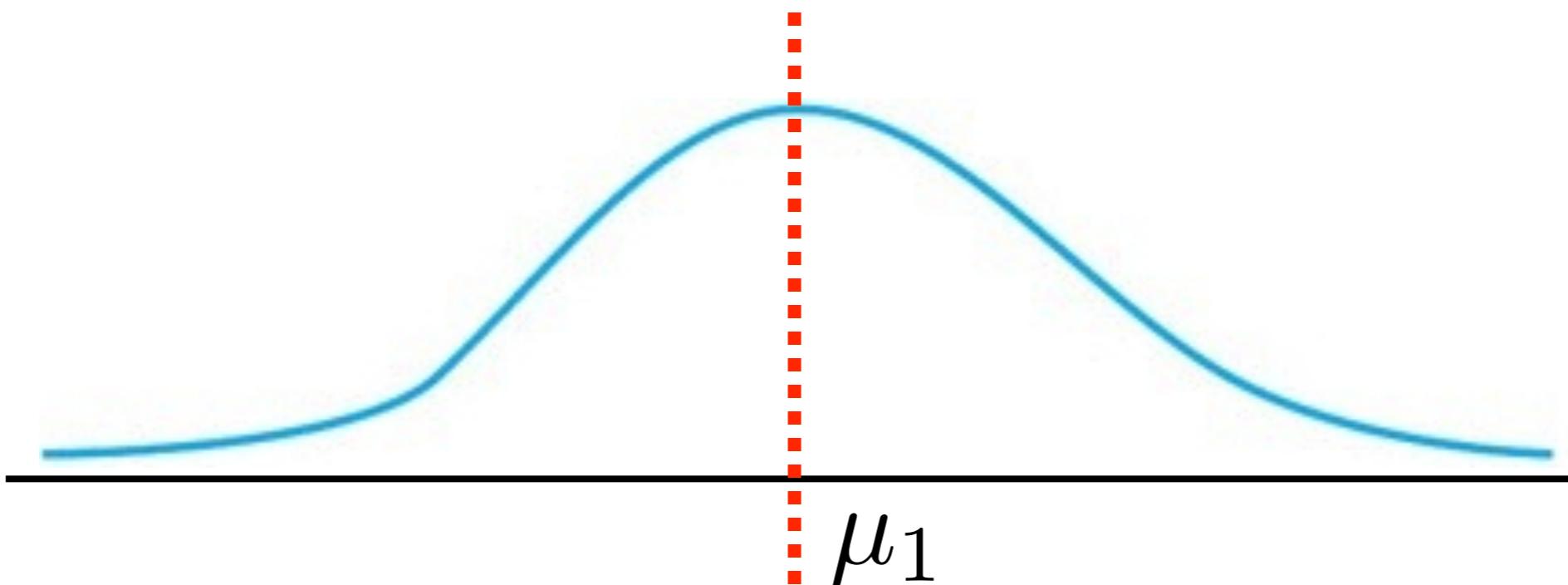
sig.zerocross waveform sign-change rate



- on any curve
- `sig.zerocross(..., 'Per', 'Second')`: rate per second
- `sig.zerocross(..., 'Per', 'Sample')`: rate per sample
- `sig.zerocross(..., 'Dir', 'One')`: only ↑ or ↓
- `sig.zerocross(..., 'Dir', 'Both')`: both ↑ and ↓

sig.centroid geometric center

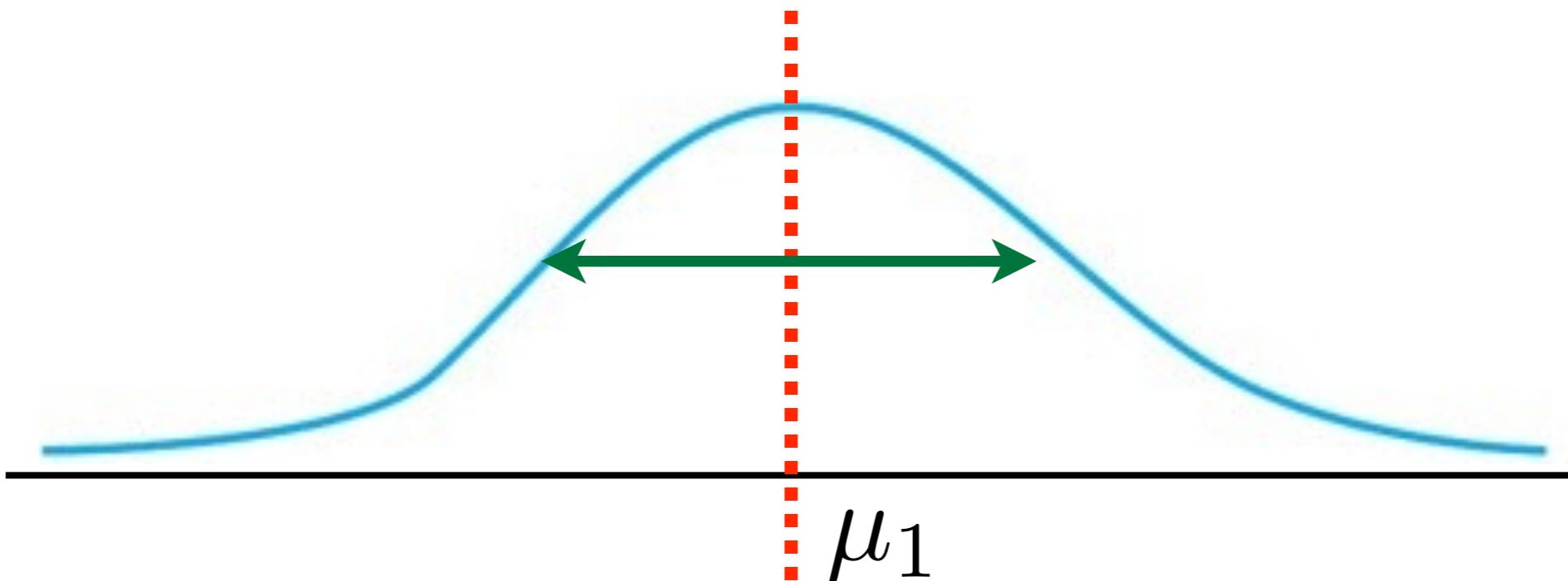
first moment: $\mu_1 = \int x f(x) dx$



sig.spread variance, dispersion

second moment: $\sigma^2 = \mu_2 = \int (x - \mu_1)^2 f(x) dx$

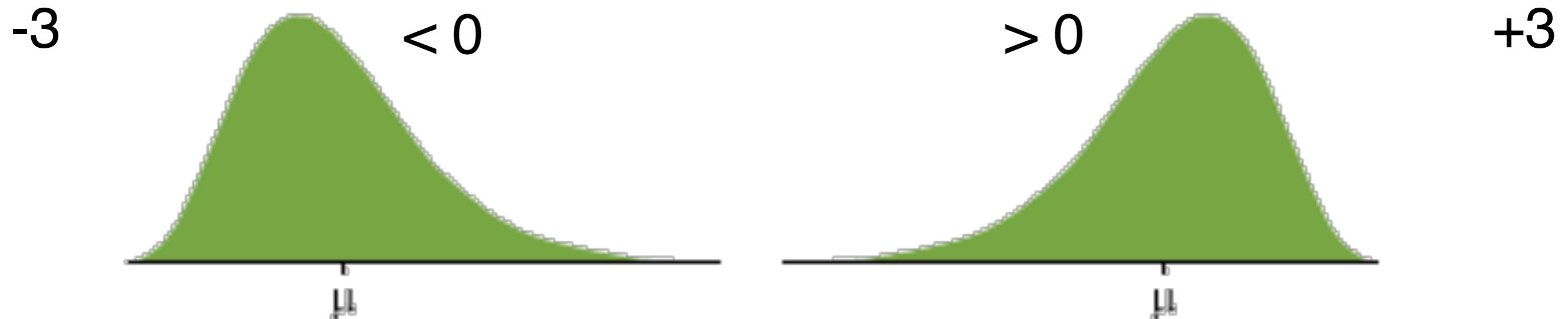
standard deviation: σ



sig.skewness non-symmetry

third moment: $\mu_3 = \int (x - \mu_1)^3 f(x) dx$

third standardized moment: $\frac{\mu_3}{\sigma^3}$

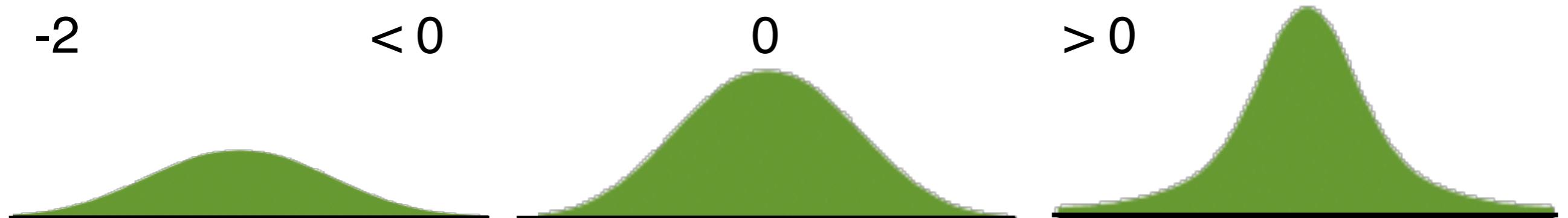


sig.kurtosis pickiness

fourth moment: $\mu_4 = \int (x - \mu_1)^4 f(x) dx$

fourth standardized moment: $\frac{\mu_4}{\sigma^4}$

excess kurtosis: $\frac{\mu_4}{\sigma^4} - 3$



sig.flatness

smooth vs. spiky

geometric mean

arithmetic mean

$$\frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\left(\frac{\sum_{n=0}^{N-1} x(n)}{N} \right)}$$

sig.entropy

relative Shannon entropy

- Data considered as probability distribution:
 - $p \geq 0$: half-wave rectification
 - $\sum p = 1$: normalization
- Shannon entropy: $H(p) = -\sum(p \log(p))$
- Relative entropy, independent on the sequence length:
$$H(p) = -\sum(p \log(p)) / \log(\text{length}(p))$$
- Entropy gives an indication of the curve:
 - High entropy \approx uncertainty \approx flat curve
 - Low entropy \approx certainty \approx peak(s)

sig.export

exportation of statistical data to files

- *sig.export(filename, ...)* adding one or several data from *MiningSuite* operators.
- *sig.export('result.txt', ...)* saved in a text file.
- *sig.export('result.arff', ...)* exported to WEKA for data-mining.
- *sig.export('Workspace', ...)* saved in a *Matlab* variable.

aud.play feature-based playlist

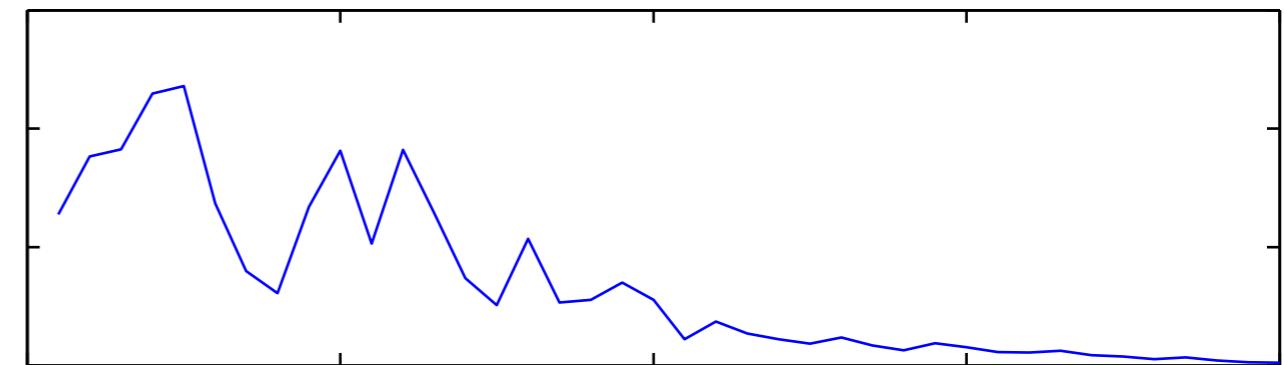
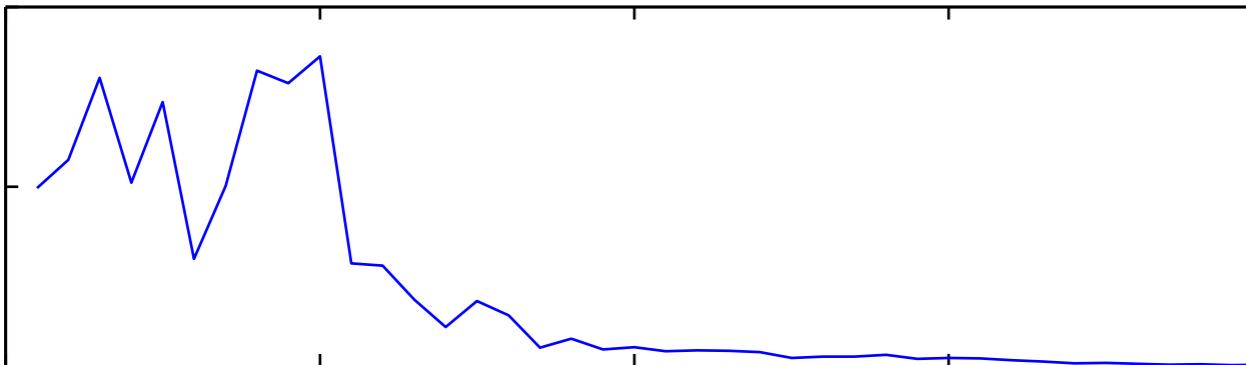
- `zc = sig.zerocross('Folder');`
- `aud.play('Folder', 'Increasing', zc)`
 - Plays the folder of audio files in increasing order of zero-crossing rate.
- `aud.play('Folder', 'Increasing', zc, 'Every', 5)`
 - Plays one out of five audio files.

sig.classify
classification

sig.dist

distance between features

- *s1 = aud.spectrum('a', 'Mel')*
- *s2 = aud.spectrum('b', 'Mel')*



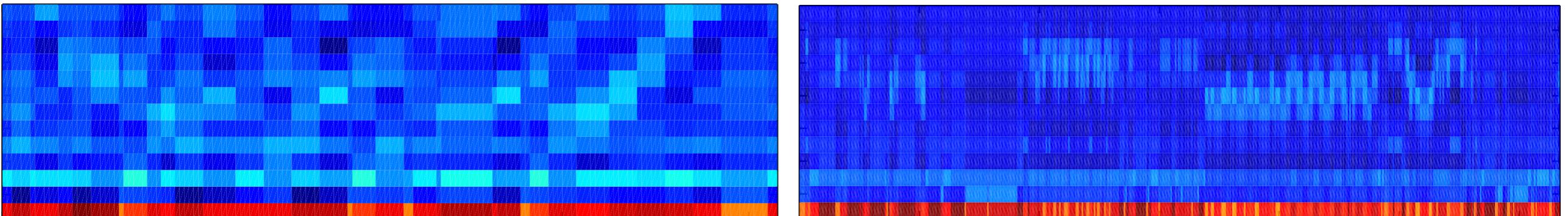
- *sig.dist(s1, s2, 'Cosine')*

The Mel-Spectrum Distance between files a and b is 0.1218

sig.dist

distance between clusters

- $c1 = aud.mfcc('a', 'Frame')$
- $c1 = sig.cluster(c1, 16)$
- $c2 = aud.mfcc('b', 'Frame')$
- $c2 = sig.cluster(c2, 16)$



Earth Mover's Distance between clusters

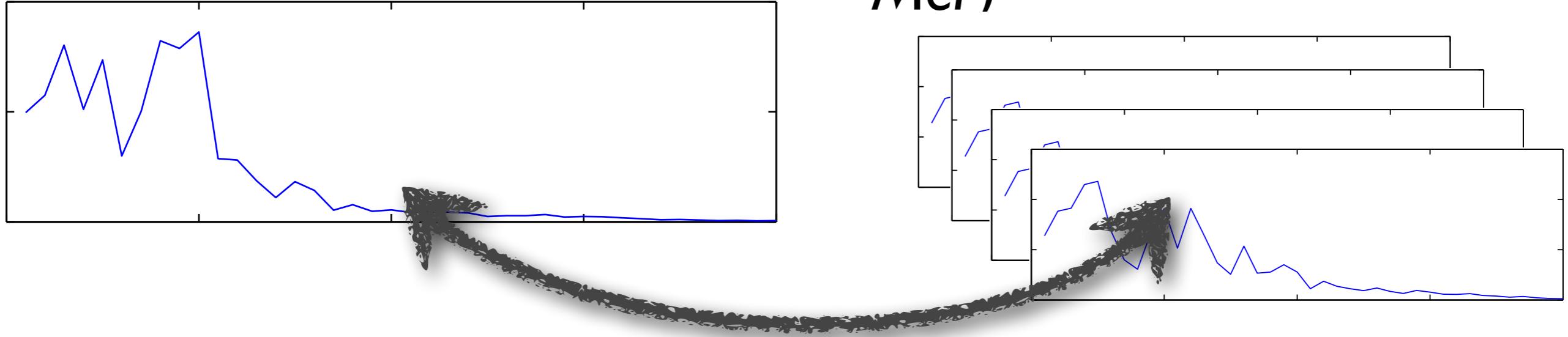
- $sig.dist(c1, c2)$

The MFCC Distance between files a and b is 19.3907

sig.dist

distance between features

- `s1 = aud.spectrum('a', 'Mel')`
- `s2 = aud.spectrum('Folder', 'Mel')`



- `sig.dist(s1, s2)`

The Mel-Spectrum Distance between files a and b1 is 0.2386

The Mel-Spectrum Distance between files a and b2 is 0.45729

The Mel-Spectrum Distance between files a and b3 is 0.6338

The Mel-Spectrum Distance between files a and b4 is 0.20082

.getdata

returns data in Matlab format

s = *sig.spectrum*('file');—————→

Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

s.getdata

vector



s

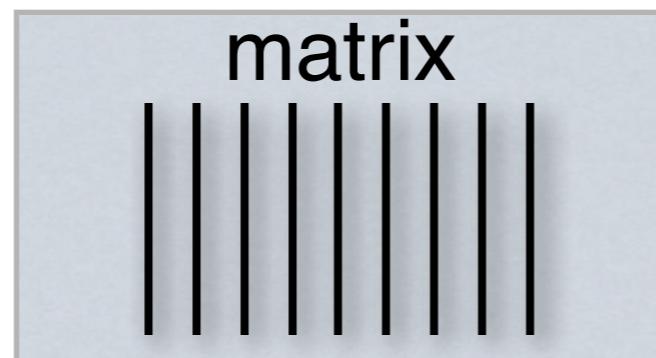
.getdata

returns data in Matlab format

s = *sig.spectrum*('file',
'Frame');

Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

s.getdata



s

.getdata

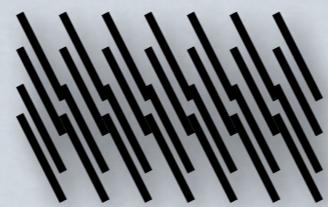
returns data in Matlab format

f = *sig.filterbank*('file',
'Frame');

Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

f.getdata

3D-matrix

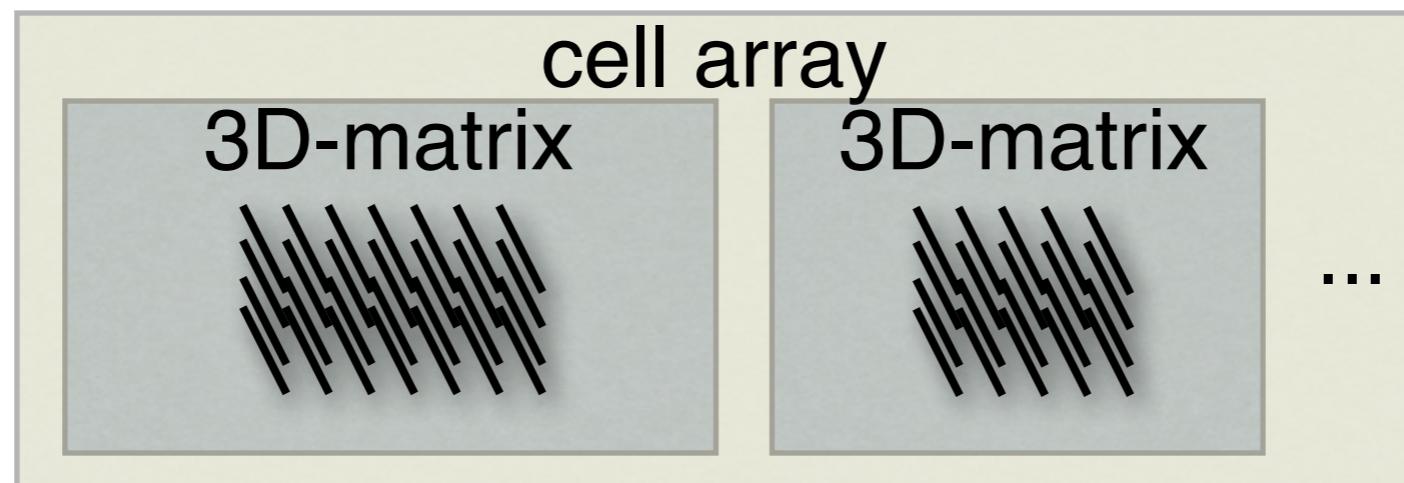


.getdata
returns data in Matlab format

```
sg = sig.segment('file')  
f = sig.filterbank(sg, ——————  
'Frame');
```

Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

f.getdata



.getdata

returns data in Matlab format

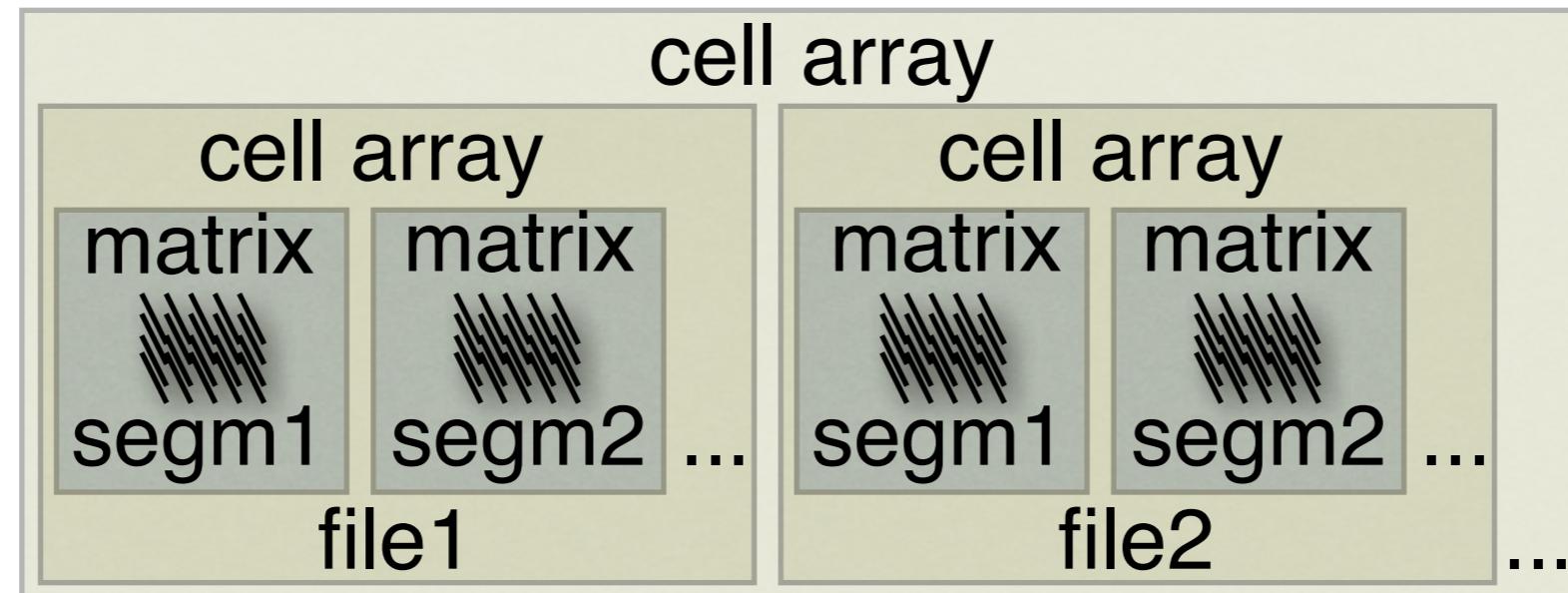
```
sg =  
sig.segment('Folder')  
f = sig.filterbank(sg,  
'Frame');
```



Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

f

f.getdata



.getpeakpos, .getpeakval
returns data in Matlab format

p = sig.peaks...

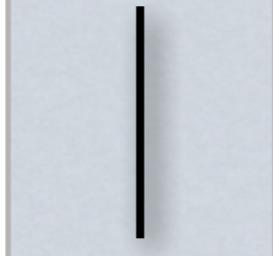


Encapsulated data
numerical data,
related sampling rates,
related file name,
etc.

p

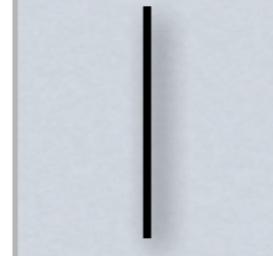
p.getpeakpos

vector



p.getpeakval

vector



sig.signal

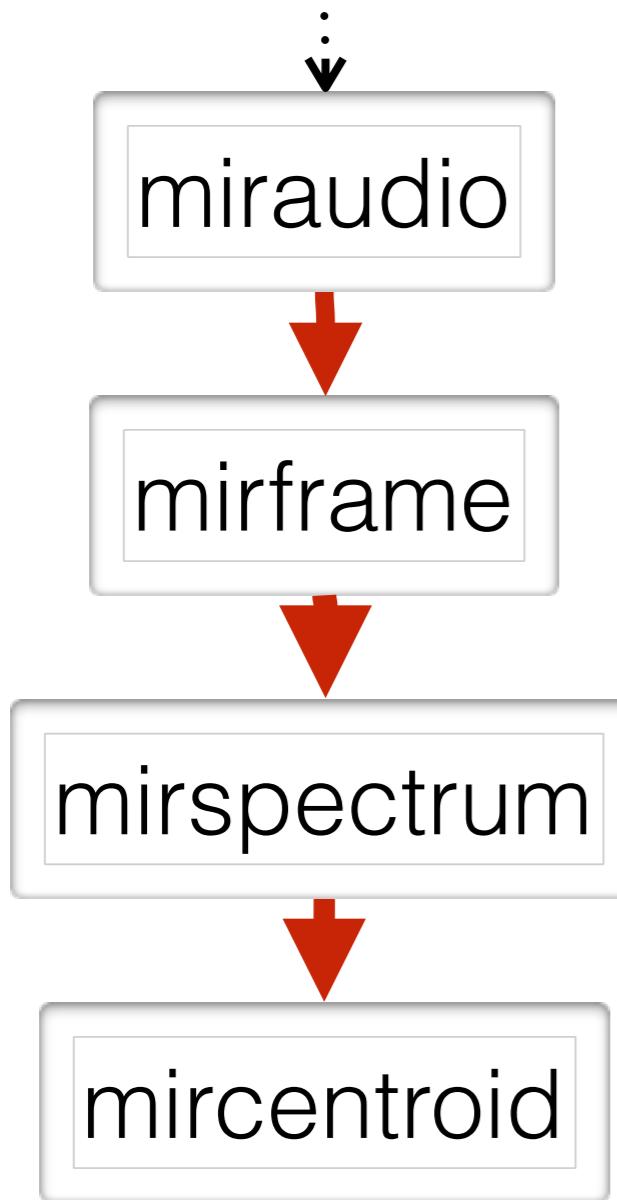
Accessing fields

- a** = *sig.spectrum*(..., ‘Frame’) → *sig.signal* object **a**
- a.**xname** = ‘Frequency’
 - a.**xunit** = ‘Hz’
 - a.**xdata**: frequency bins, recomputed on the fly
 - a.**xstart** = 0
 - a.**xsampling** = 10.766
 - a.**sdata** = frame positions, recomputed on the fly
 - a.**Sstart** = 0
 - a.**Srate** = 40
 - a.**yname** = ‘Spectrum’
 - a.**Ydata** = *sig.data* object
 - a.Ydata.**dims** = {‘element’, ‘sample’}
 - a.Ydata.**content**: Matlab matrix
 - rows are frequencies (‘element’)
 - columns are frames (‘sample’)



Limitations of data flow in *MIRtoolbox*

long audio file,
batch of files

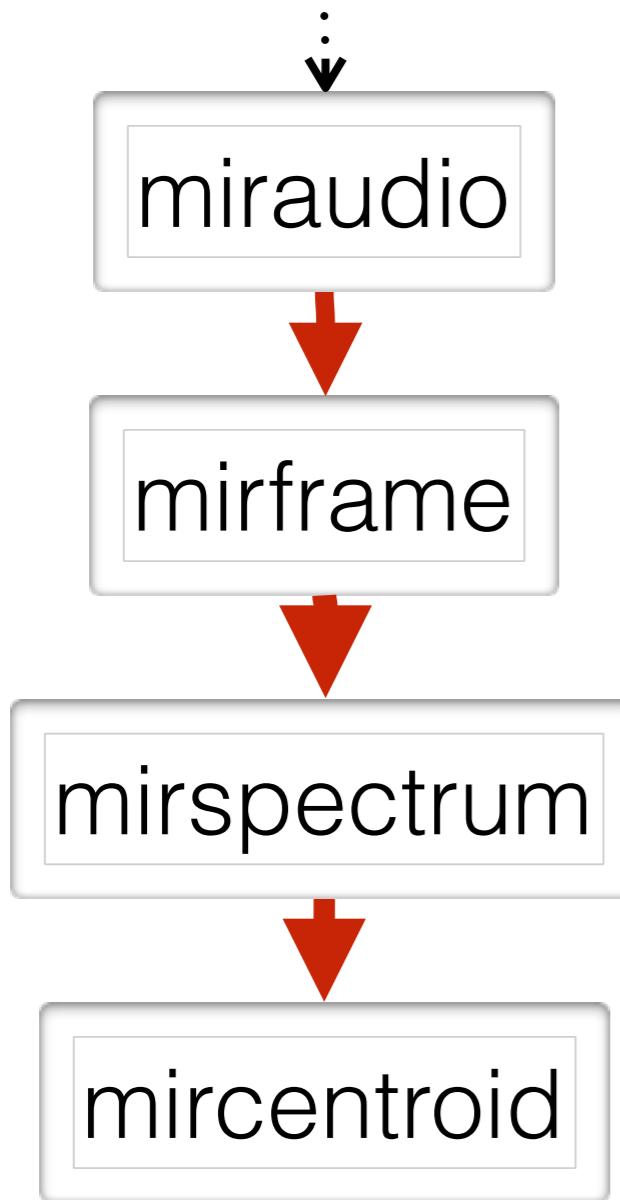


- `a = miraudio('bigfile')`
- `f = mirframe(a)`
- `s = mirspectrum(f)`
- `mircentroid(s)`
 - `mircentroid('bigfile', 'Frame')`

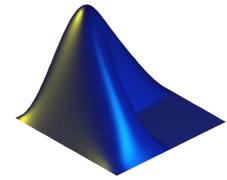


Data flow graph design & evaluation

long audio file,
batch of files

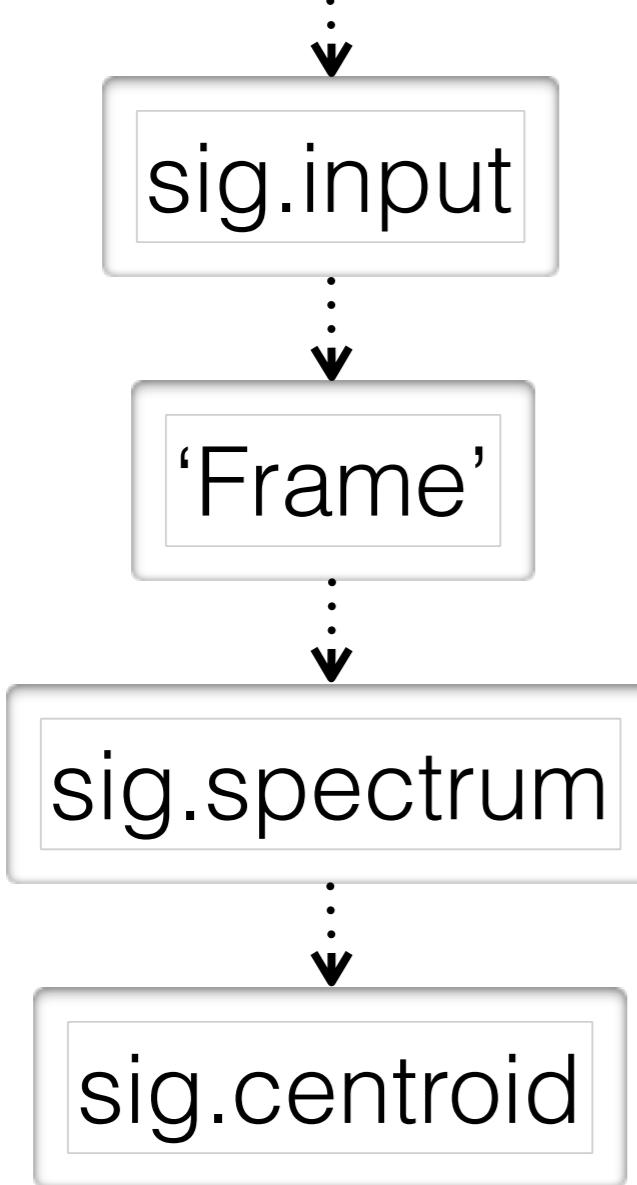


- `a = miraudio('Design', ...)`
- `s = mirspectrum(f, 'Frame', ...)`
- `c = mircentroid(s)`
- **`mireval(c, 'bigfile')`**



Data flow graph in MiningSuite

long audio file,
batch of files



- `a = sig.input('bigfile', ...);`
 - `s = sig.spectrum(f, 'Frame', ...);`
 - `c = sig.centroid(s)`
- `;` → No operation is performed.
(The data flow graph is constructed without actual computation.)

sig.design

data flow graph design

- `a = sig.input('bigfile', ...);` → *sig.design* objects, storing only the data flow graph
- `s = sig.spectrum(a);` → *sig.design* objects, storing only the data flow graph
- `c = sig.centroid(s)` → Design now evaluated in order to display the results.
- `c` → But results was not stored in **c**, so displaying again **c** triggers another evaluation of the design.
- `d = sig.ans` → The last evaluation is stored in **sig.ans**.
- `d = c.eval` → Evaluate and store in a variable.
- `d = c.getdata` → Evaluate and store in a variable.

sig.design.show data flow graph display

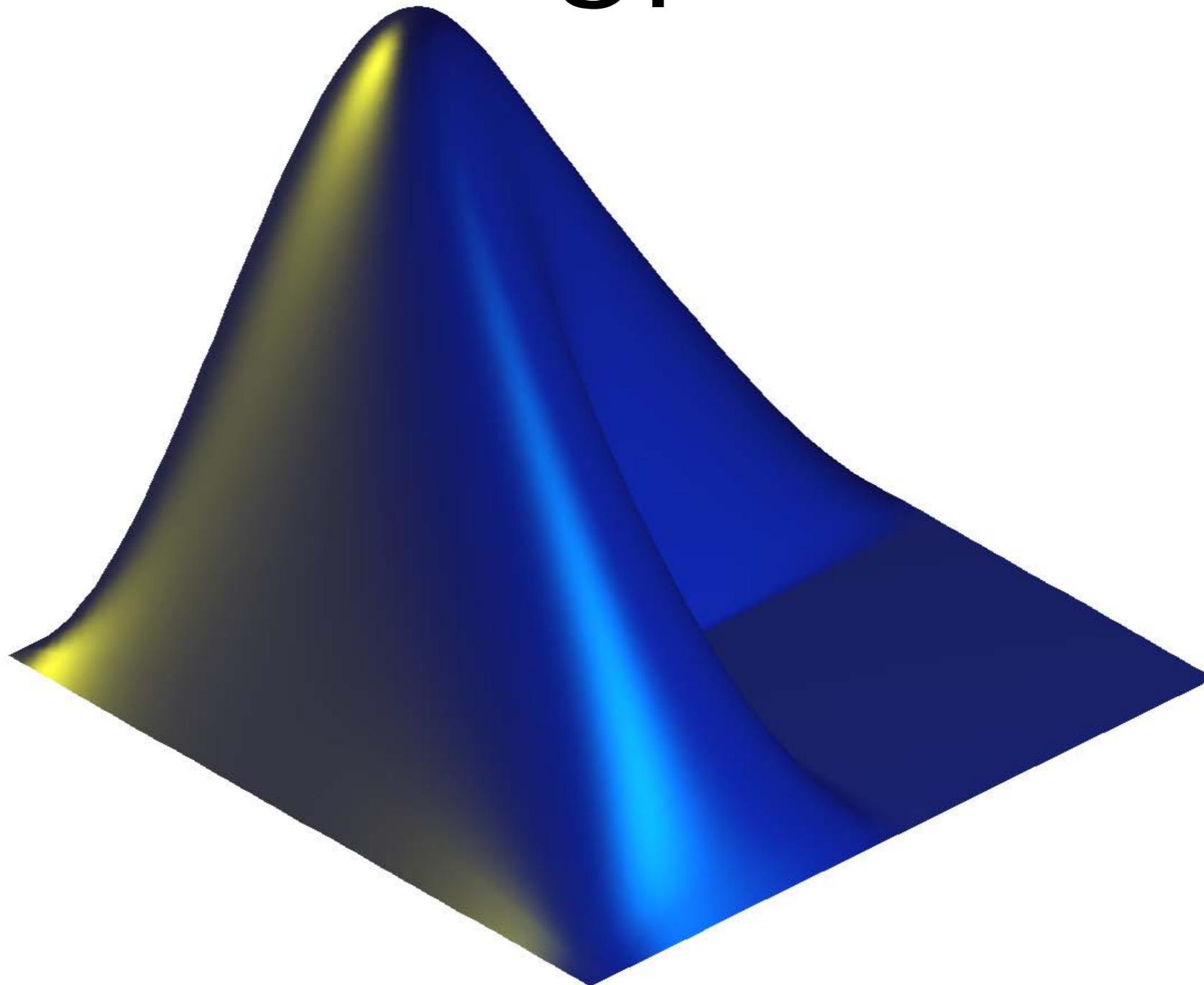
- **a** = *sig.input(...);*
- **s** = *sig.spectrum(a);*
- **c** = *sig.centroid(s)*
- **c.show**

sig.signal.design

design stored in the results

- $a = \text{sig.input}(\dots);$
 - $s = \text{sig.spectrum}(a);$
 - $c = \text{sig.centroid}(s);$
 - $d = c.eval$
 - $d.\textbf{design}$
 - save *result.mat* d
- 1 year later:
- load *result.mat*
 - d → the results
 - $d.\textbf{design}$
- the data flow graph design
is automatically stored

3.



Symbolic approaches

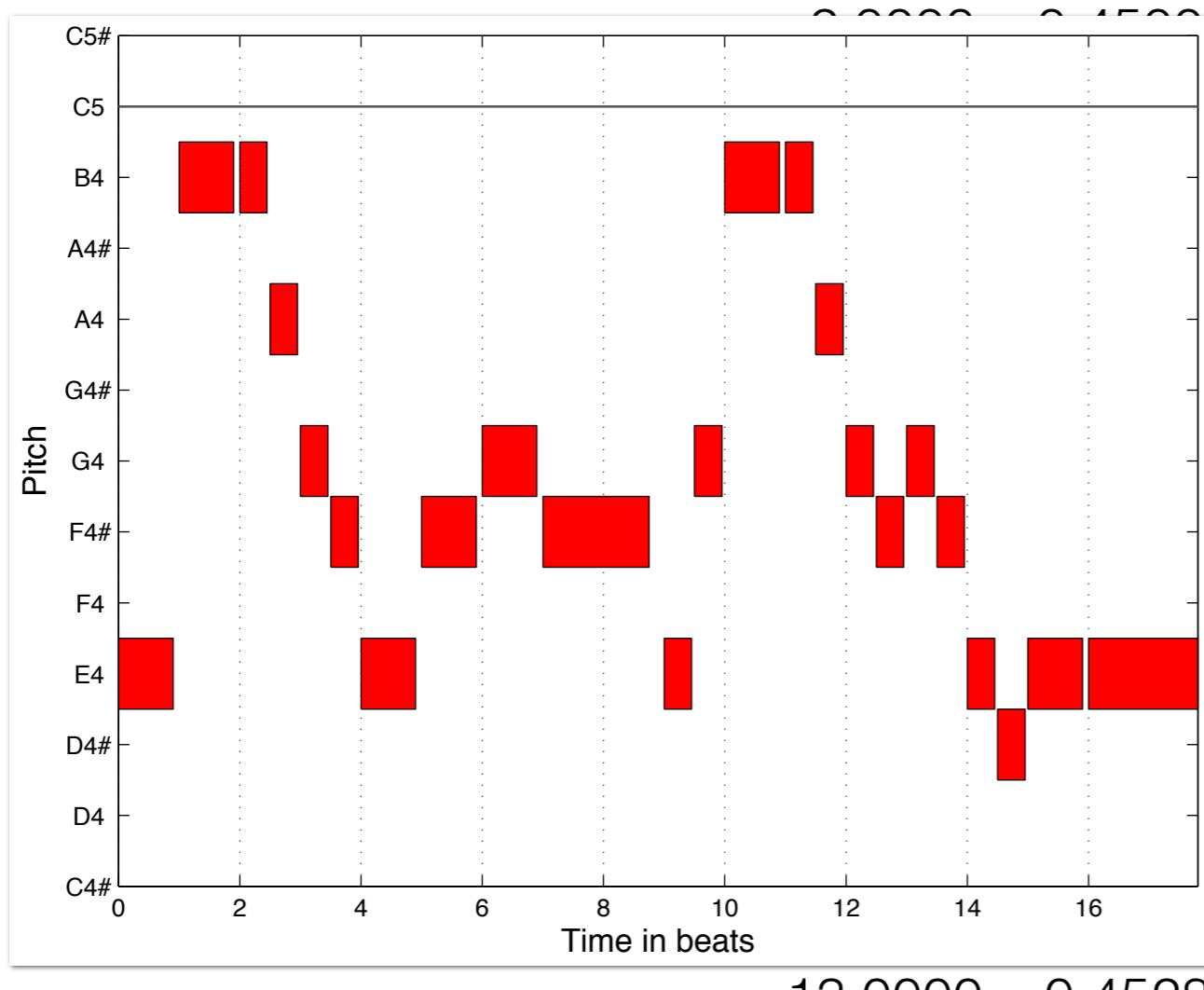
MIDI Toolbox

- nmat = readmidi('laksin.mid')

nmat =

MIDI data

- pianoroll(nmat)



0	64.0000	82.0000	0	0.5510
0	71.0000	89.0000	0.6122	0.5510
0	71.0000	82.0000	1.2245	0.2755
0	69.0000	70.0000	1.5306	0.2755
0	67.0000	72.0000	1.8367	0.2772
0	66.0000	72.0000	2.1429	0.2772
0	64.0000	70.0000	2.4490	0.5510
0	66.0000	79.0000	3.0612	0.5510
0	67.0000	85.0000	3.6735	0.5510
0	66.0000	72.0000	4.2857	1.0714
0	64.0000	74.0000	5.5102	0.2772
0	67.0000	81.0000	5.8163	0.2772
0	71.0000	83.0000	6.1224	0.5510
0	71.0000	78.0000	6.7347	0.2772
0	69.0000	73.0000	7.0408	0.2772
0	67.0000	71.0000	7.3469	0.2772
0	66.0000	69.0000	7.6531	0.2772
0	67.0000	82.0000	7.9502	0.2772

mus.score

score excerpt selection

- *mus.score(..., ‘Notes’, 10:20)*
- *mus.score(..., ‘StartTime’, 30, ‘EndTime’, 60)*
- *mus.score(..., ‘Channel’, 1)*
- *mus.score(..., ‘Trim’)*
 - *mus.score(..., ‘TrimStart’, ‘TrimEnd’)*

mus.score

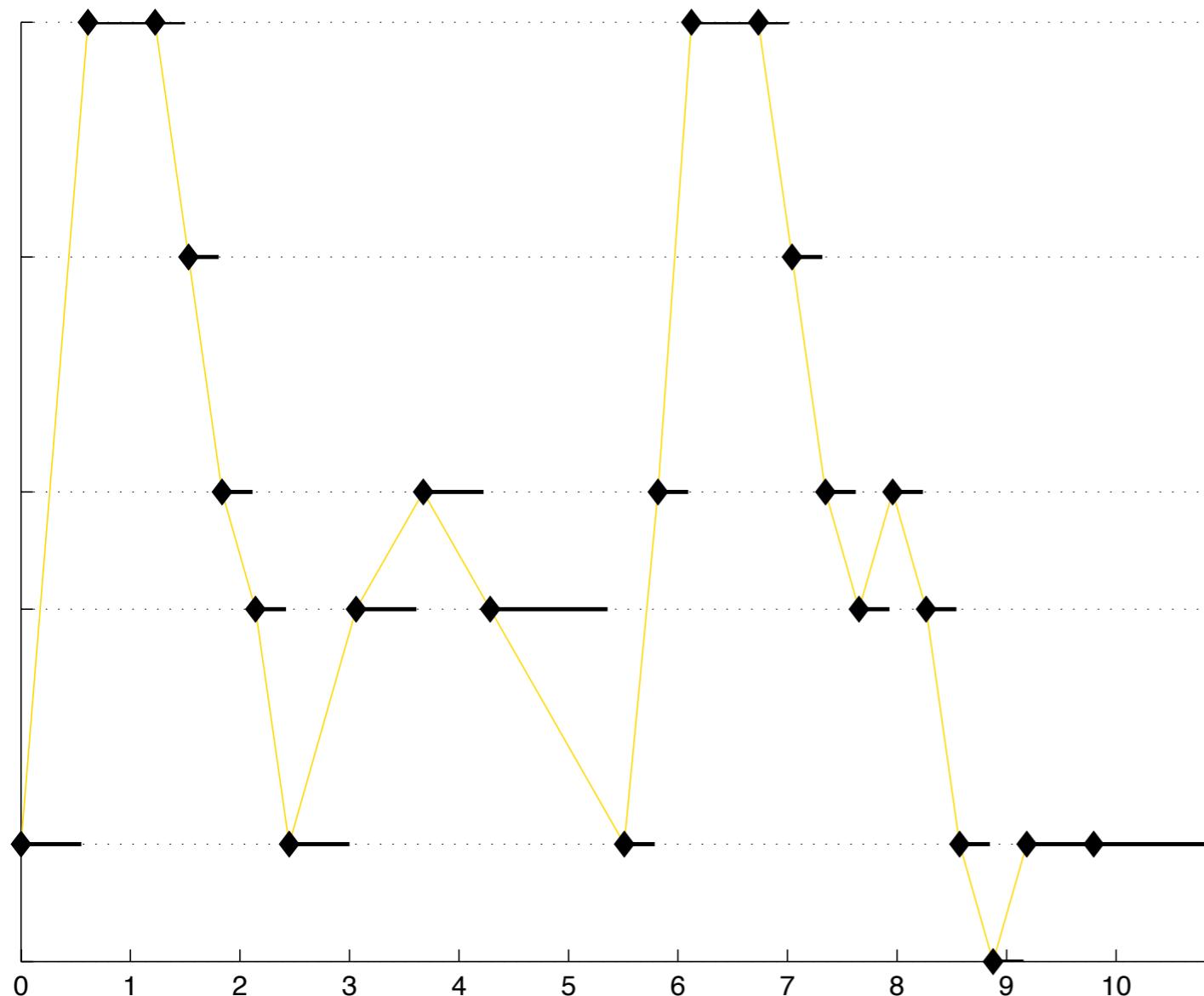
score information

- **metrical grid**: hierarchical construction of pulsations over multiple metrical levels
- **modal and tonal spaces**: mapping pitch values on scale patterns (on delimited temporal regions)
- **syntagmatic chains**: successive notes forming voices, enabling to express relative distance between successive notes (rhythmic values)

mus.score

score information

mus.score('laksin.mid')



mus.save
mus.play

mus.pitch

pitch contour

- $m = \text{mus.score}(\text{'myfile'})$

- $s = \textbf{\textit{mus.pitch}}(m)$

- **$\textbf{\textit{mus.histo}}(s, \text{'Class'})$**

- $i = \text{mus.pitch}(s, \text{'Inter'})$

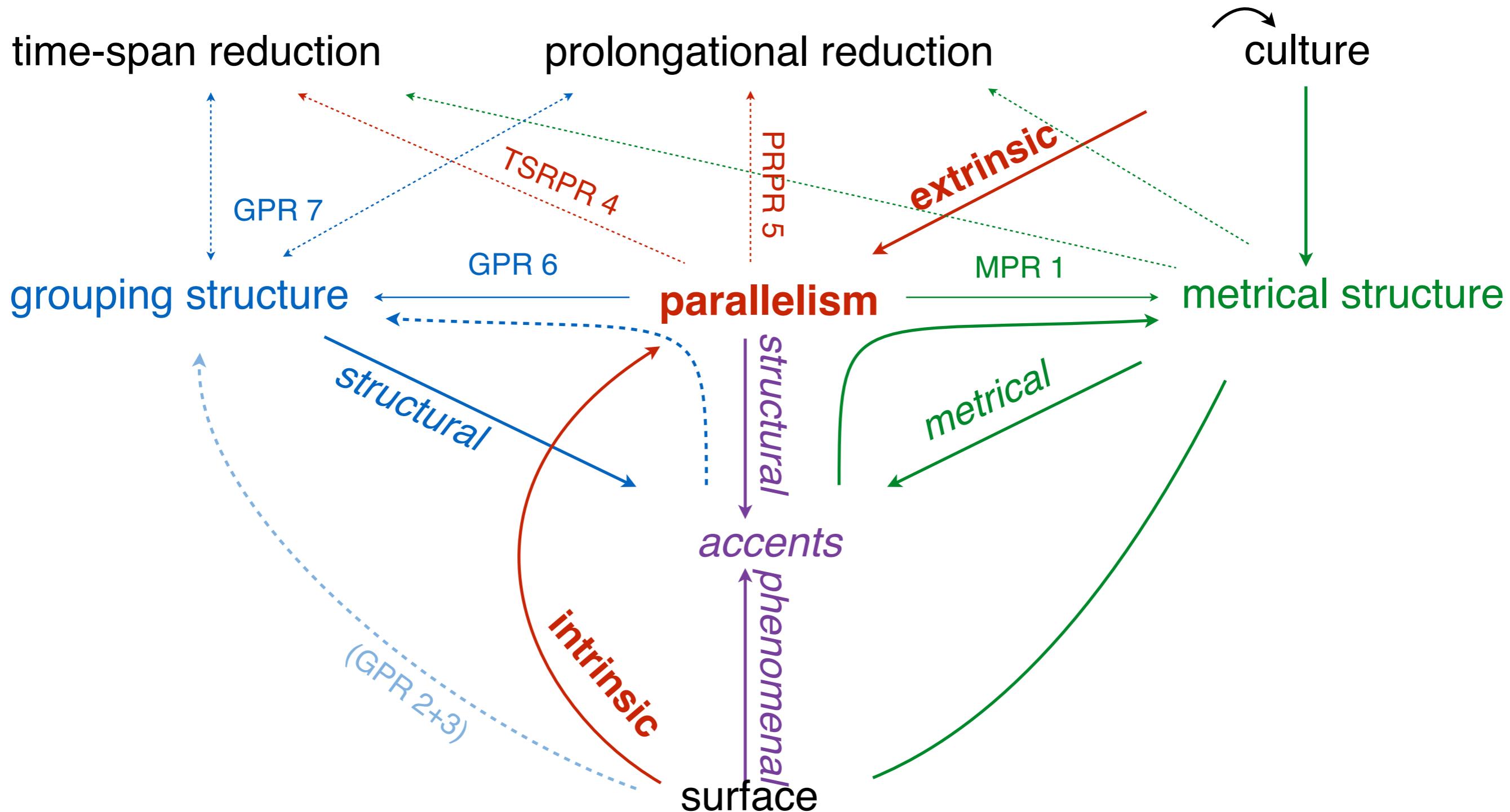
- $\text{mus.histo}(i)$

- $\text{mus.histo}(i, \textbf{\textit{Sign}}, 0)$

mus.pitch pitch contour

- *m = mus.score('myfile')*
- *c = mus.pitch(m, 'Sampling', .25)*
- *mus.autocor(c)*
- *sig.dist(c1, c2)*
- *MIDI toolbox* functions related to melodic expectation and complexity could be integrated easily into *MiningSuite*, if necessary.

F. Lerdahl, R. Jackendoff, A generative theory of tonal music, MIT Press, 1983



O. Lartillot, “Reflexions towards a generative theory of musical parallelism”,
Musicae Scientiae, DF 5, 2010.

*Structural
levels*

Groups

Mode
Tonality

Meter

Symbolic level

Timbre

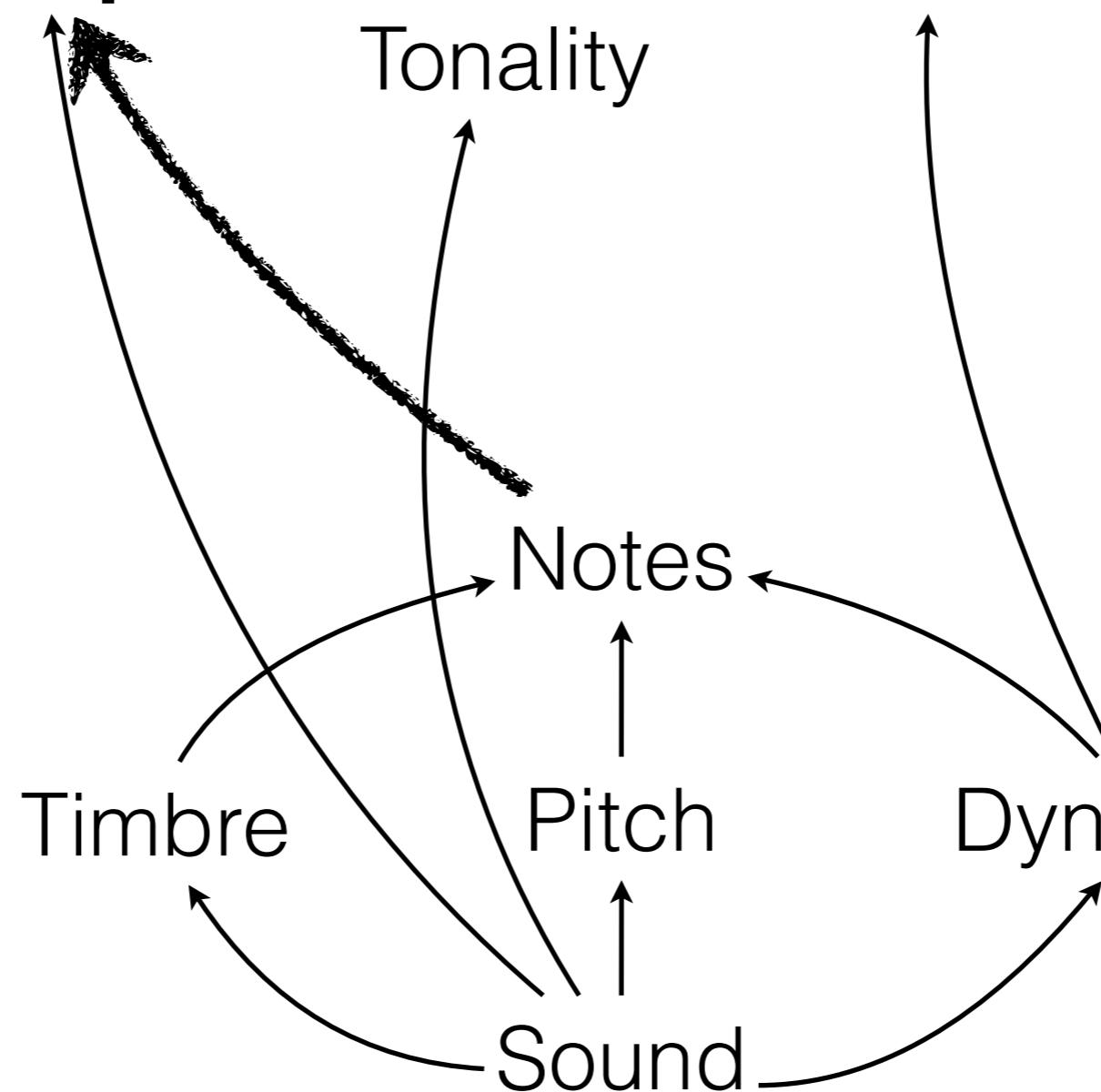
Notes

Dynamics

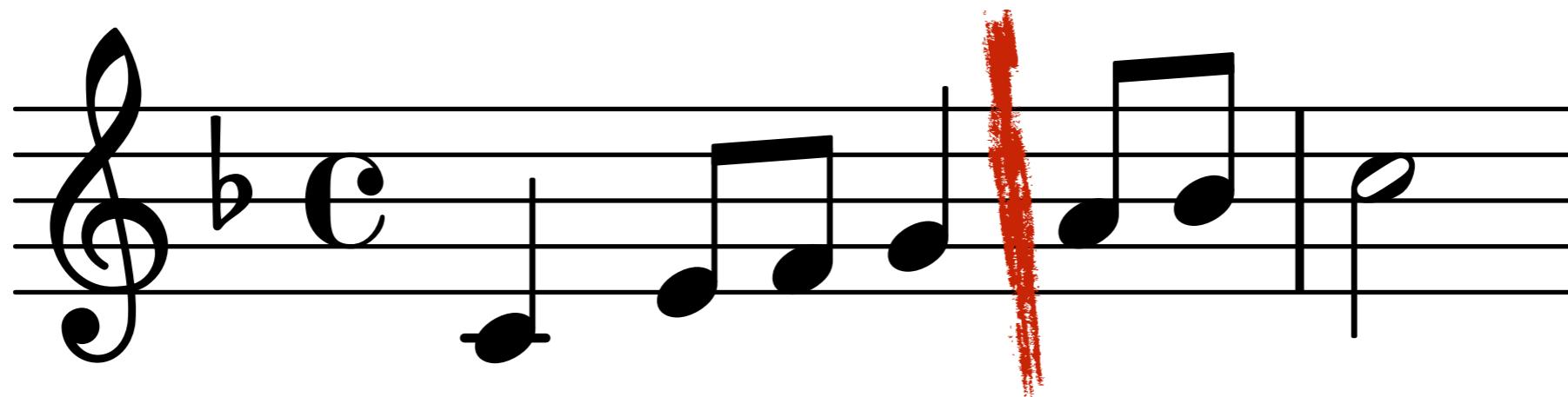
Pitch

Sound

Audio level

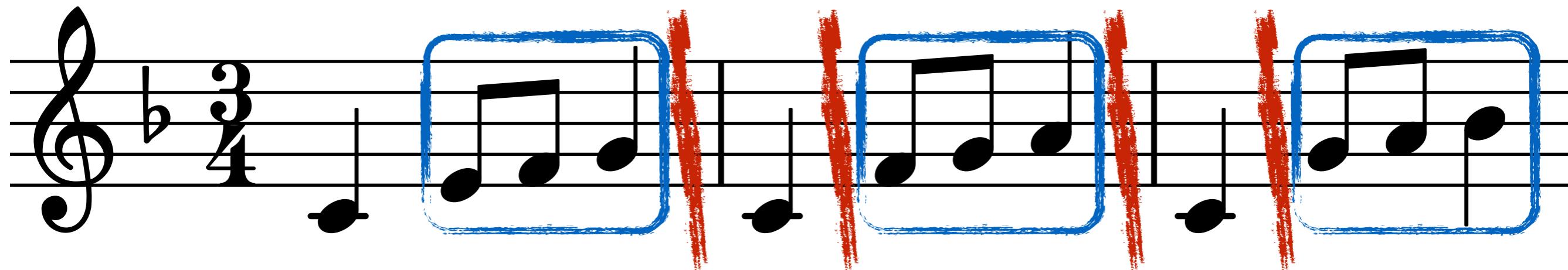


mus.score(..., 'Segment')
local segmentation

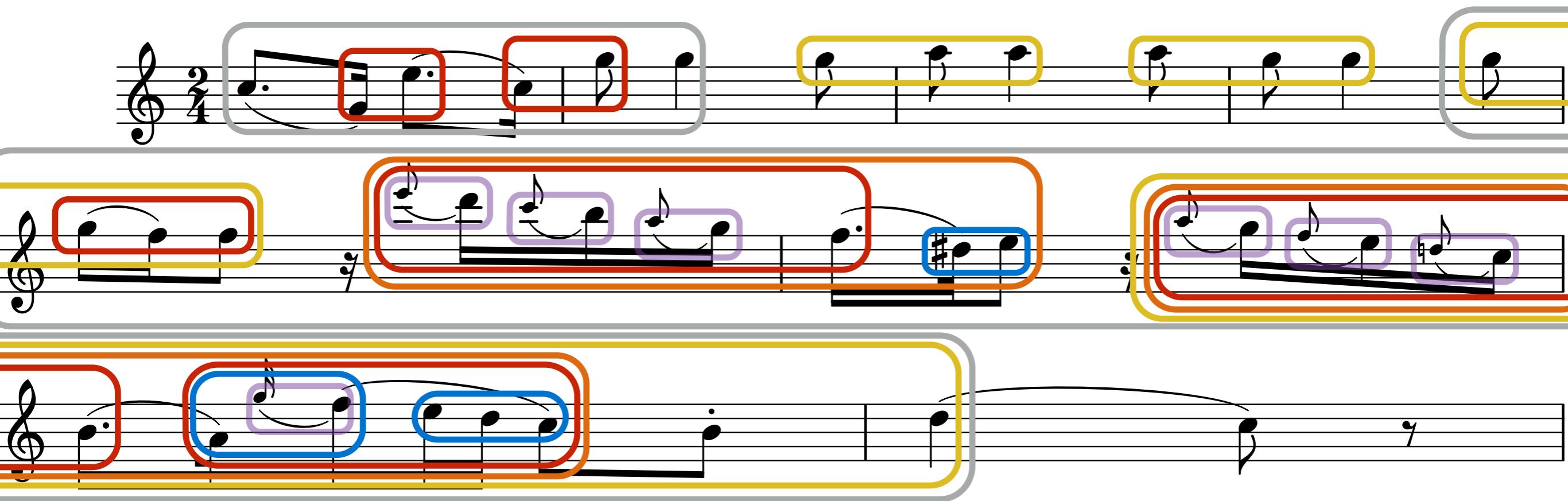


- Tenney & Polansky
- Bod
- LBDM (Cambouropoulos)

`mus.score(..., 'Segment')` vs.
`mus.score(..., 'Group')`



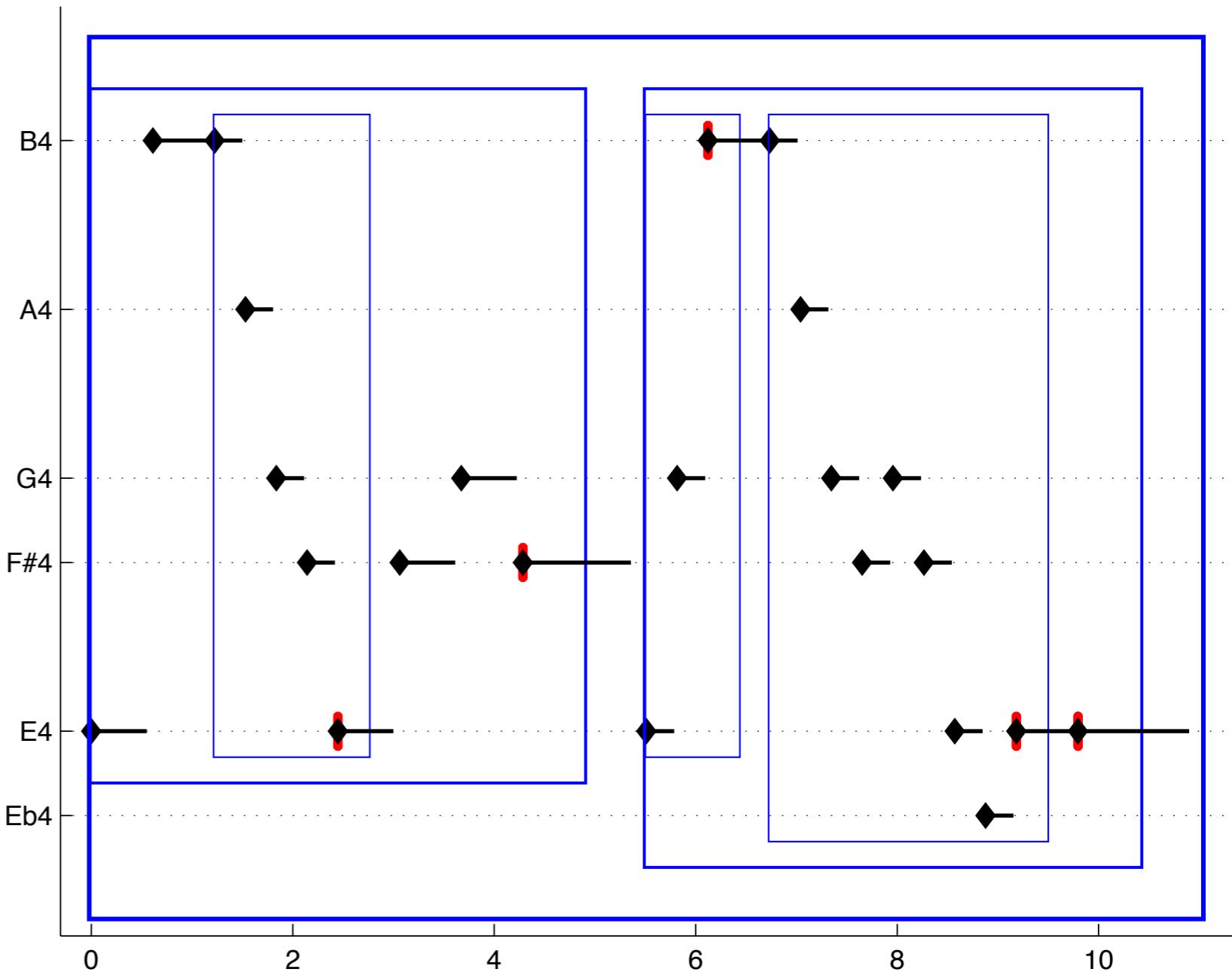
mus.score(..., 'Group') hierarchical grouping



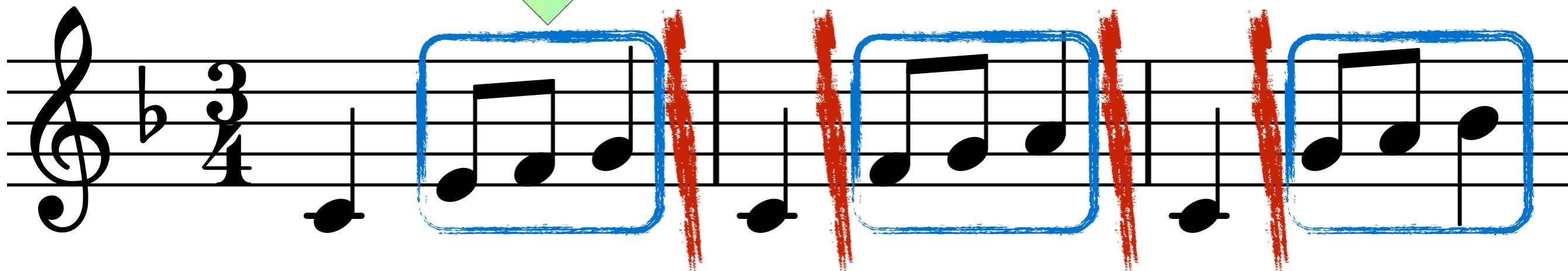
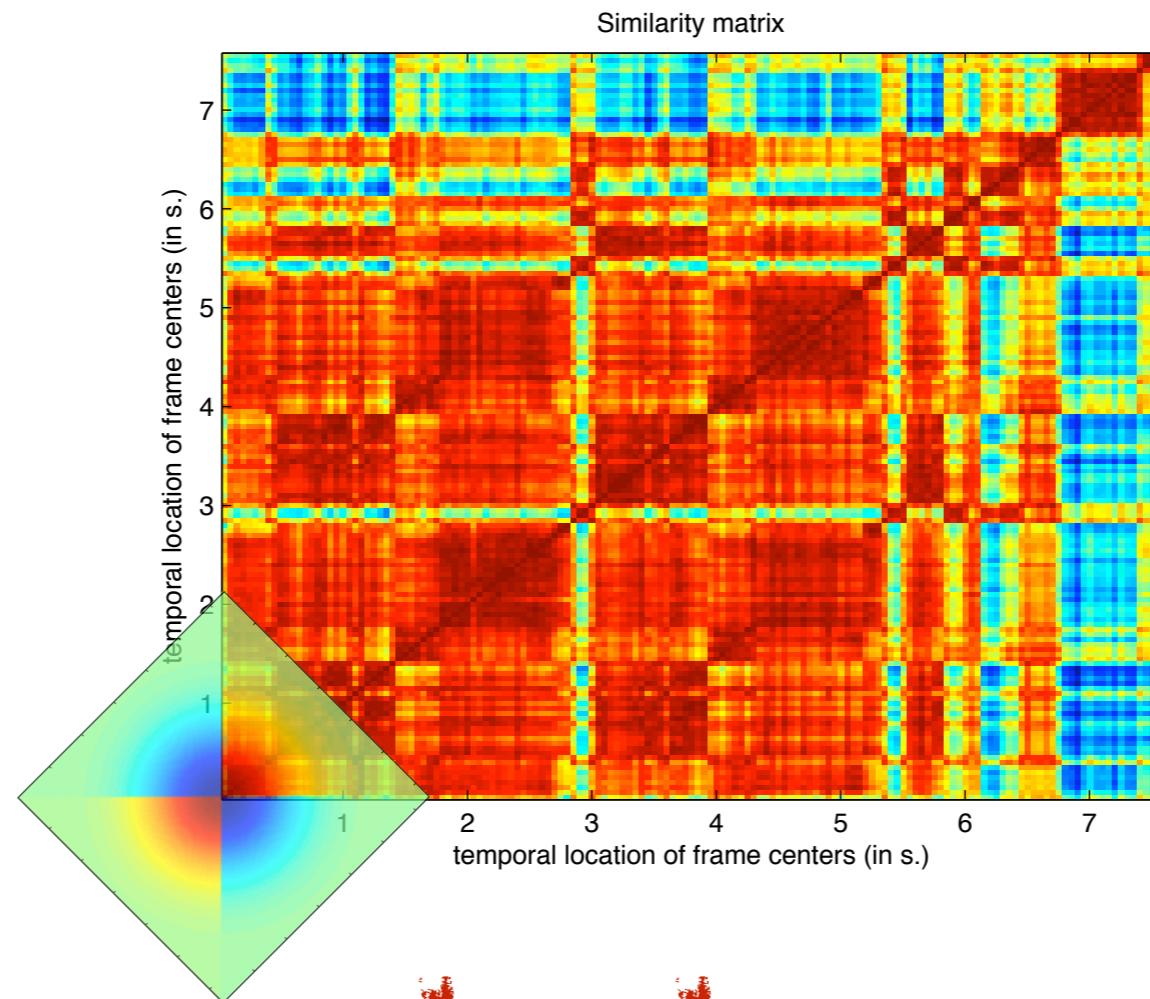
Mozart, Variation XI on “Ah, vous dirai-je maman”, K.265/300e

mus.score(..., 'Group')
hierarchical grouping

*mus.score('laksin.mid',
'Group')*



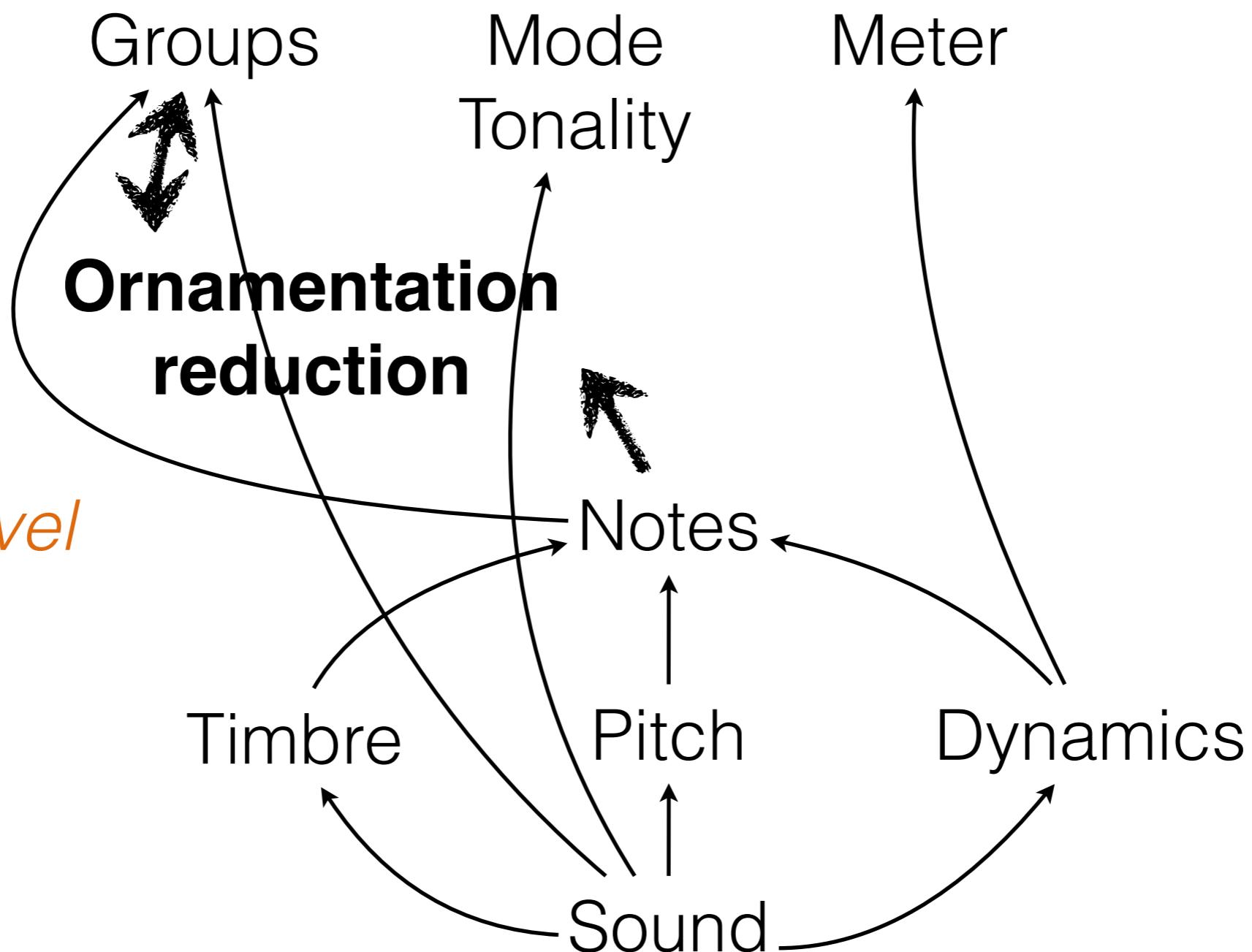
*aud.novelty vs.
mus.score(..., 'Group')*



*Structural
levels*

Symbolic level

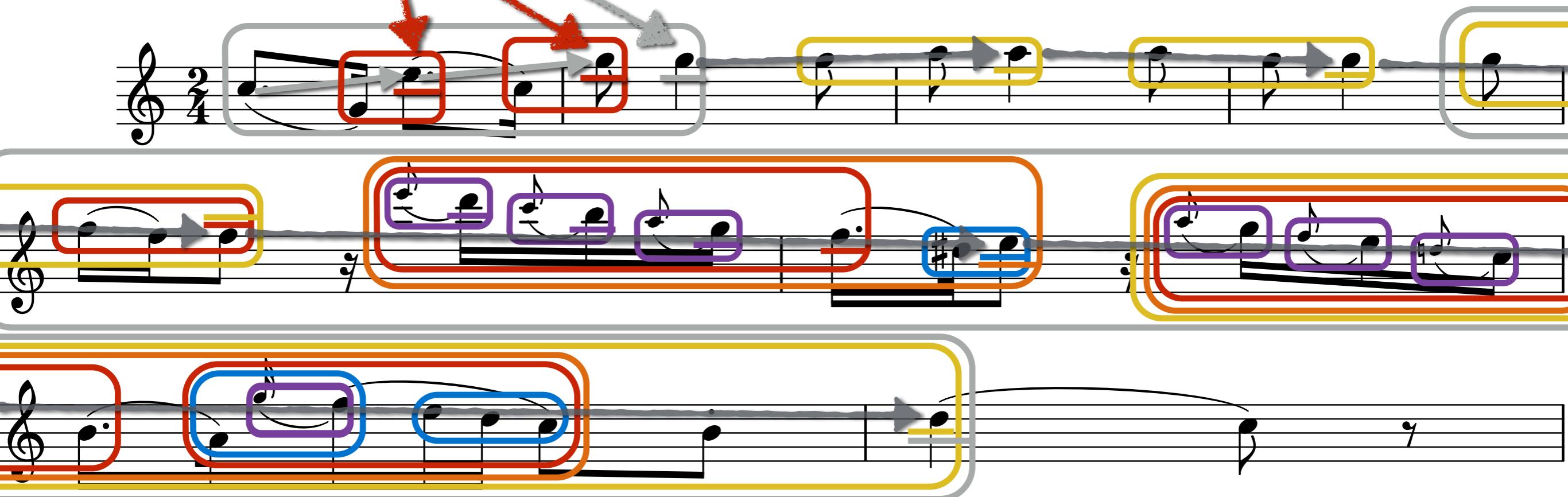
Audio level



```
mus.score(..., 'Reduce')  
ornamentation reduction
```

head

(Lerdahl & Jackendoff)

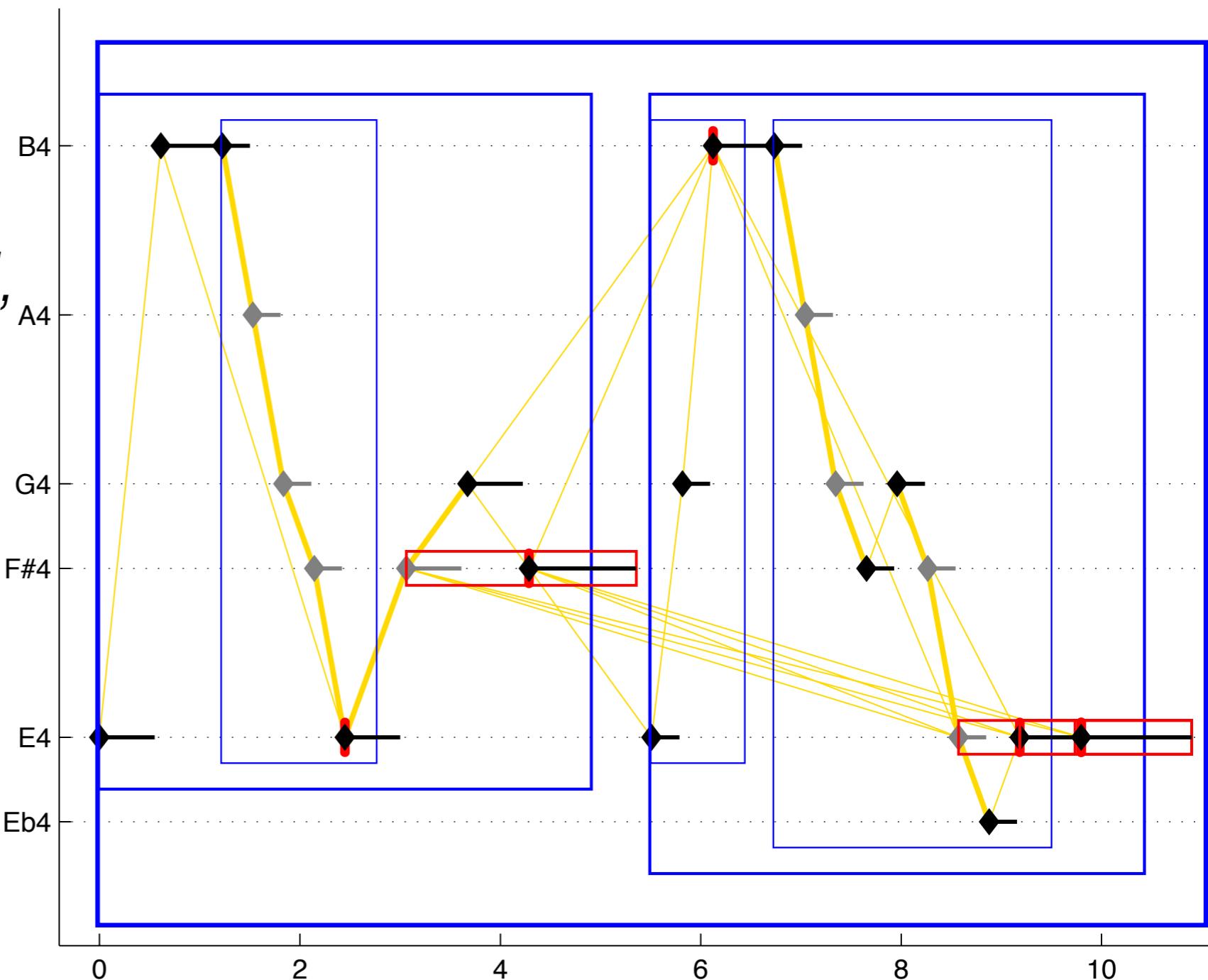


Mozart, Variation XI on “Ah, vous dirai-je maman”, K.265/300e

mus.score(..., 'Reduce')
ornamentation reduction

*mus.minr('laksin.mid',
'Group', 'Reduce')*

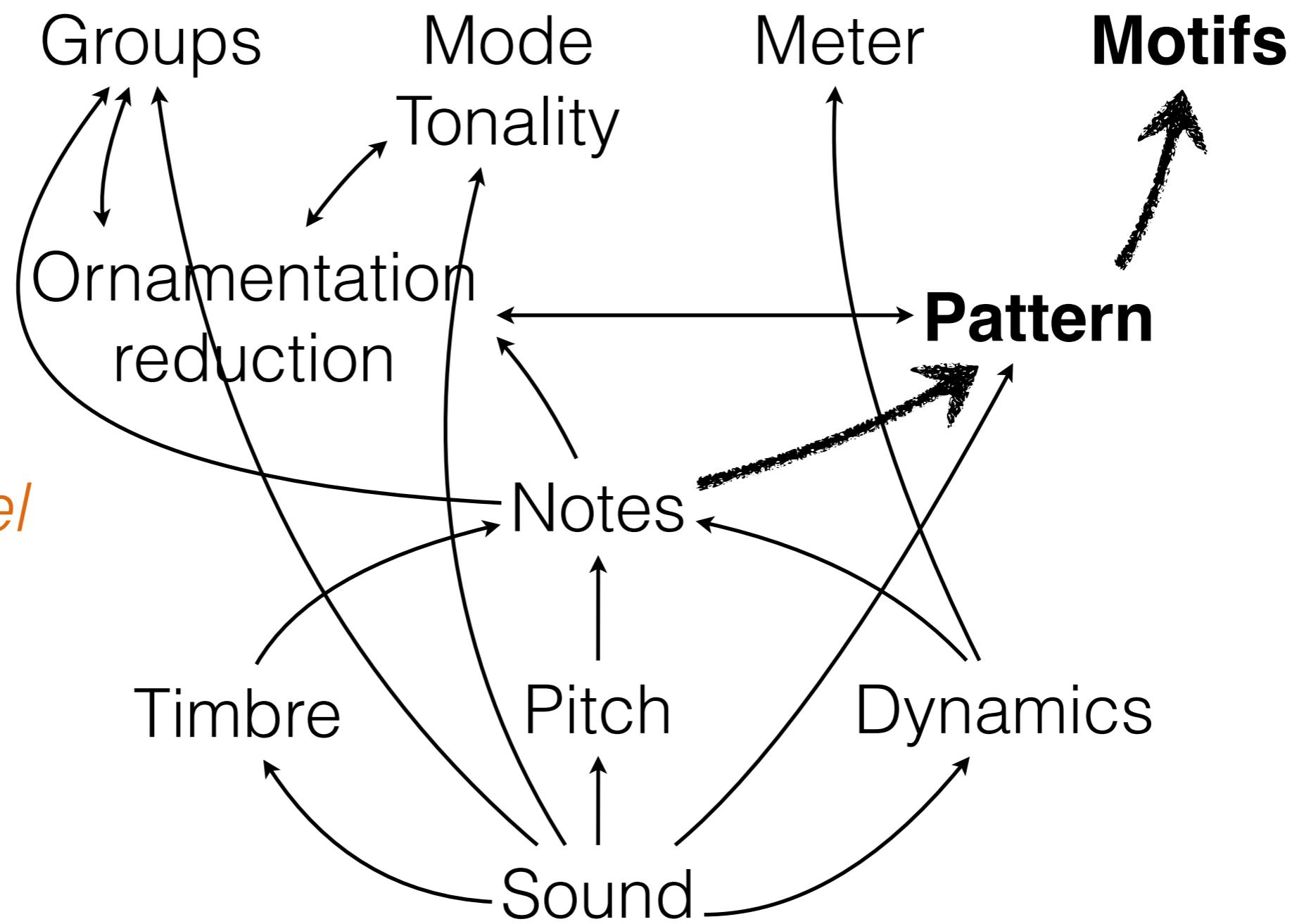
Construction of a
syntagmatic network



*Structural
levels*

Symbolic level

Audio level



motivic pattern analysis

Bach Invention in D minor

Kresky (1977) analysis

Name	Kresky's interpretation
A	Opening motive
A'	First sequence unit
B	Accompanying figure
a	Motive shape
a1	Reversed motive shape
a2=c	Triadic structure
a3=a3'	Retarded scalar climb
a4=C	Bass line
a5	Seconde sequence unit
a6	(not considered)
b	Three-note scale
b'	Identifying a5 with a3
b''	Same, transposed
b'''	Three shape in bass line

motivic pattern analysis

Bach Invention in D minor

Computer analysis

Musical score for Bach's Invention in D minor, measures 1-2. The score consists of two staves: treble and bass. The treble staff starts with a key signature of one flat. Measure 1 begins with a sixteenth-note pattern: A (horizontal bar), a (horizontal bar), b (horizontal bar). Measure 2 continues with a similar pattern: B (horizontal bar), c (horizontal bar), c (horizontal bar). The bass staff has rests in both measures.

Computer analysis of Bach's Invention in D minor, measures 5-6. The analysis highlights motivic patterns across both staves. The treble staff shows a sixteenth-note pattern starting at measure 5: A (horizontal bar), c (horizontal bar), a (horizontal bar), b (horizontal bar). In measure 6, it transitions to a different pattern: A' (horizontal bar), a³' (horizontal bar), b' (horizontal bar). The bass staff shows a sustained note C (horizontal bar) throughout these measures. The analysis also identifies a new motif E (horizontal bar) in the bass staff of measure 6.

Computer analysis of Bach's Invention in D minor, measures 10-11. The analysis continues to track motivic patterns. In the treble staff of measure 10, it identifies a sixteenth-note pattern: a5 (horizontal bar), b' (horizontal bar), b' (horizontal bar), b' (horizontal bar). It also identifies a sustained note D (horizontal bar). In the bass staff, it identifies a sustained note C (horizontal bar). In measure 11, it identifies a sixteenth-note pattern: a5 (horizontal bar), b'' (horizontal bar), b'' (horizontal bar), b'' (horizontal bar). It also identifies a sustained note D (horizontal bar).

Lartillot, Toivainen, "Motivic matching strategies for automated pattern extraction", Musicæ Scientiae, DF4A, 281-314, 2007.

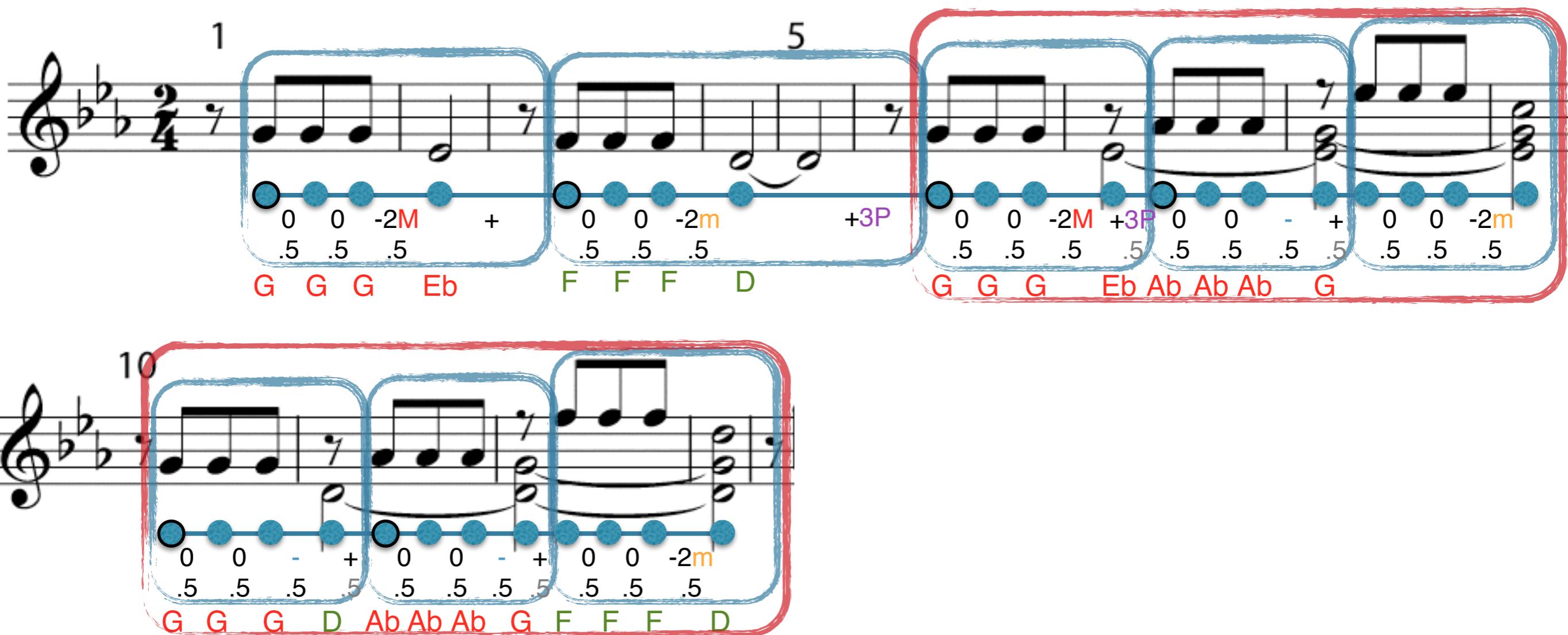
motivic pattern analysis

Bach Invention in D minor

Name	Kresky's interpretation	Computational description	
		Melodic	Rhythmic
A	Opening motive	(D,E,F,G,A,Bb,C#,Bb,A,G,F,E,F)	16th notes
A'	First sequence unit	(-2,+1,+1,+1,+1,-6,+6,-1,-1,-1,+1)	16th notes
B	Accompanying figure	(F,A,D+,-,+,+,-)	8th notes
a	Motive shape	(prefix of A)	
a1	Reversed motive shape	(not detected)	
a2=c	Triadic structure	(+,+,-)	8th notes
a3=a3'	Retarded scalar climb	(-2,+1,+1,+1,+1,-6)	16th notes
a4=c	Bass line	(+7,-5,+1,+1,+1,-6)	8th notes
a5	Seconde sequence unit	(+1,+1,-2,+1,+1,-6)	16th notes
a6	(not considered)	(-,+1,+1,+1)	16th notes
b	Three-note scale	(+1,+1)	16th notes
b'	Identifying a5 with a3	(D5,E5,F5)	16th notes
b''	Same, transposed	(C5,D5,E5)	16th notes
b'''	Three shape in bass line	(not detected)	
D	(not considered)	(-1,+1,+1,+1)	16th except 1st
E	(not considered)	(E5,F5,D5,E5,F5)	16th start. offbeat

Lartillot, Toiviainen, "Motivic matching strategies for automated pattern extraction", *Musicae Scientiae*, DF4A, 281-314, 2007.

motivic pattern analysis



Beethoven, 5th Symphony, *Allegro con brio*

motivic pattern analysis

J.S. Bach, Well-Tempered Clavier, Book II, Fugue XX
Detected subject entries

The image displays six musical staves, each representing a detected subject entry. The staves are arranged vertically, each with a unique label:

- L1: Bass clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- M1: Treble clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- U1: Treble clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- L2: Bass clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- U2: Treble clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- M2: Treble clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- U3: Treble clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.
- L3: Bass clef, common time, key signature of C major. The melody consists of eighth and sixteenth notes.

Each staff features a horizontal bracket above the notes, indicating the span of the detected motif. The music is set against a background of vertical bar lines and rests.

mus.score(..., 'Motif') motivic pattern analysis

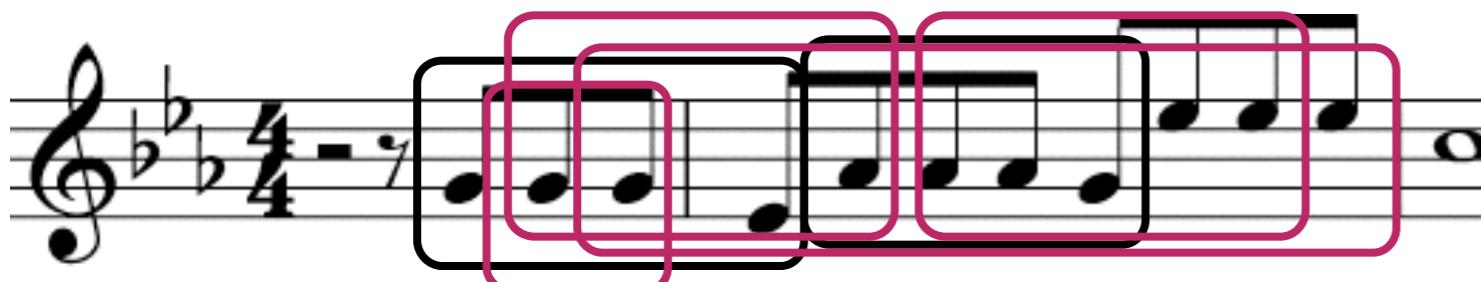
mus.minr('beethoven', 'Motif')



structure complexity



Pattern extraction



Pattern selection (longest, frequent, ...)



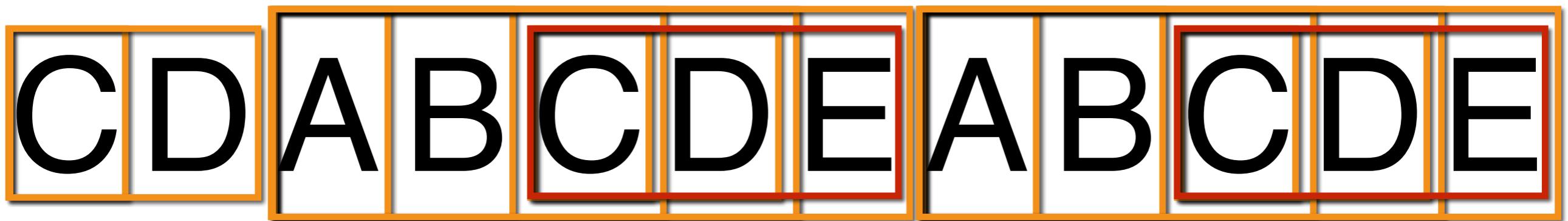
- Large set of irrelevant structures during extraction phase
 - cannot be computed extensively
 - Imperfections of the selection phase:
 - expected patterns accidentally deleted
 - insufficiently selective
 - Improvement of the extraction process

closed patterns



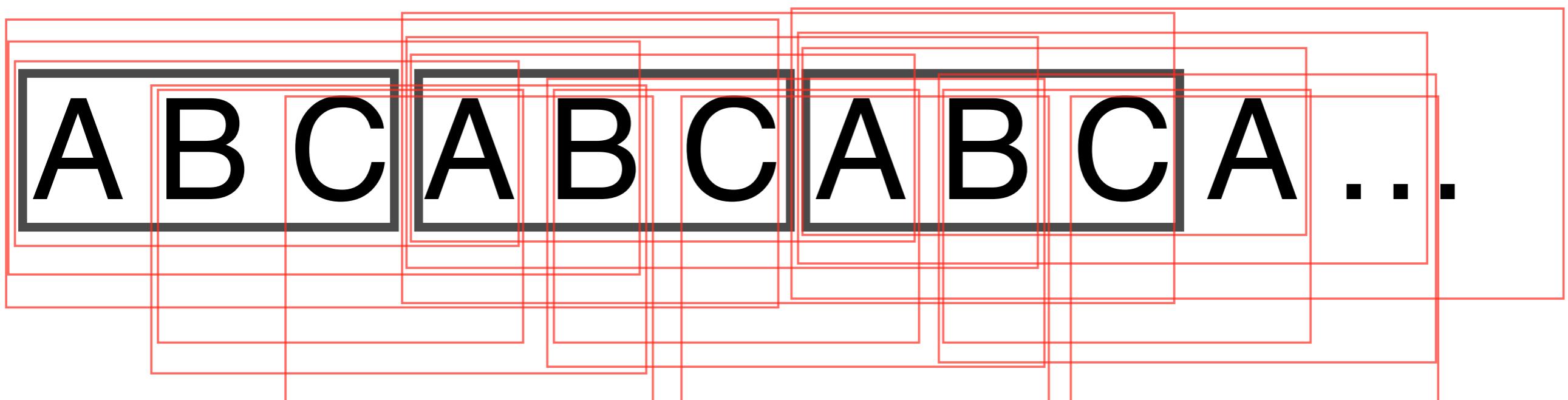
Complete and compact description of pattern repetition

incremental approach



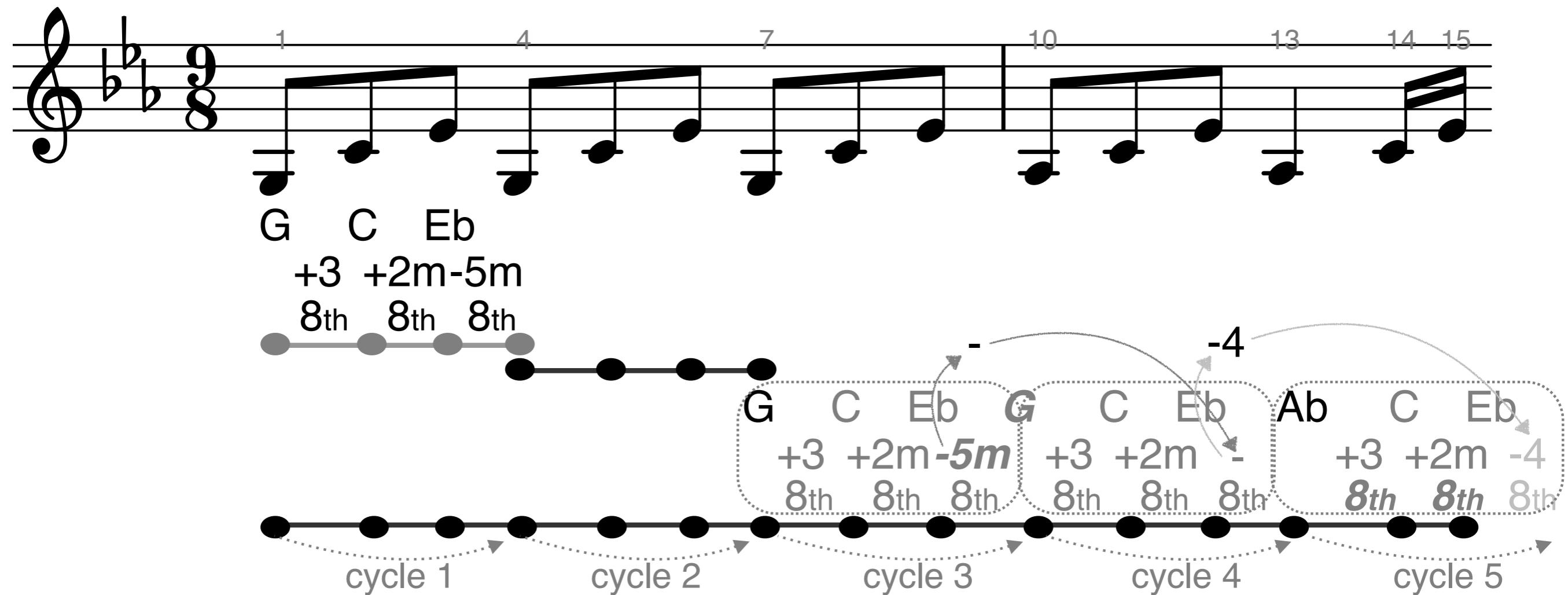
- Progressively analysing music through one single pass
- Controls structural complexity in a way similar to the way listeners perceive music.

pattern cyclicity



Cambouropoulos, 1998

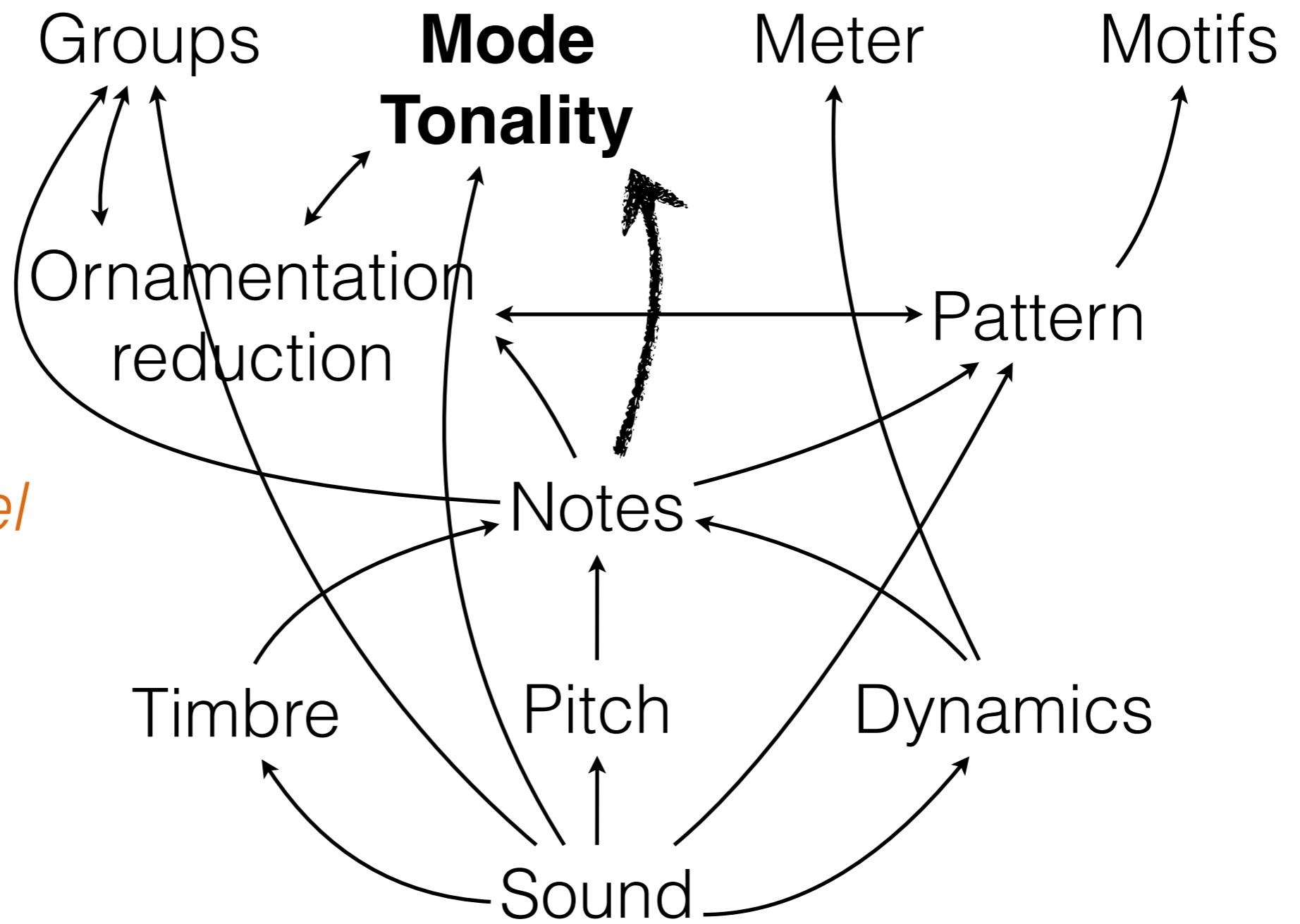
heterogeneous pattern cycles



*Structural
levels*

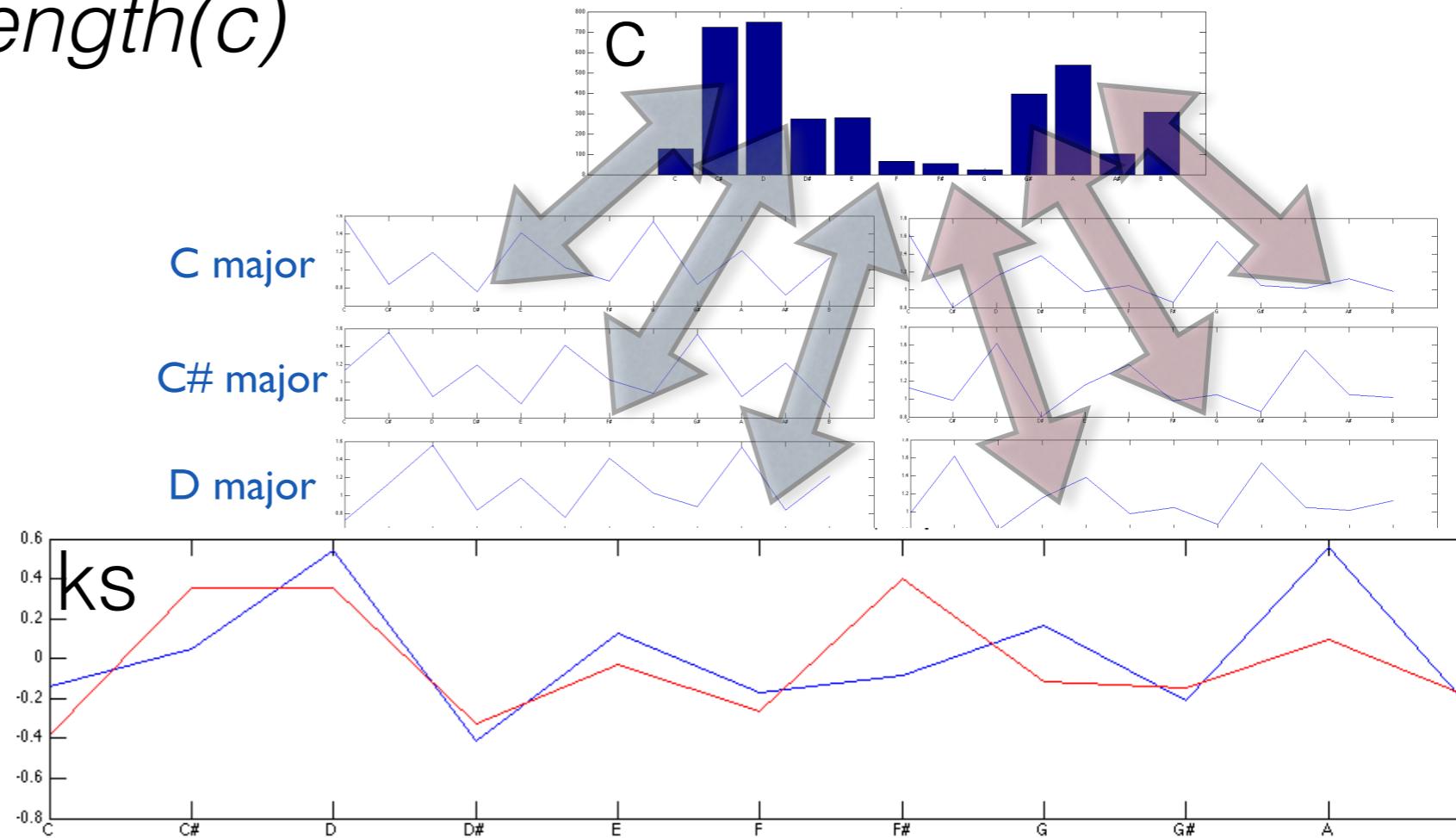
Symbolic level

Audio level



Statistical m/tonal analysis

- $c = \text{mus.chromagram}(\text{'score.mid'})$
- $ks = \text{mus.keystrength}(c)$
- $\text{mus.key}(ks)$
- $\text{mus.mode}(ks)$
- $\text{mus.keysom}(ks)$



Krumhansl, Cognitive foundations of musical pitch. Oxford UP, 1990.

Score-level m/tonal analysis

- Statistical tonal analysis: pitch distribution in frames
 - What if key transition within one single frame?
- Tonality is more than such statistical description:
 - Succession of chords rooted on the scale degrees
 - Standard chord sequences, cadenza formulae
 - Patterns, grouping help emphasize chord changes
 - etc.

Score-level m/tonal analysis

CM

CM

p

legato.

Dm

II

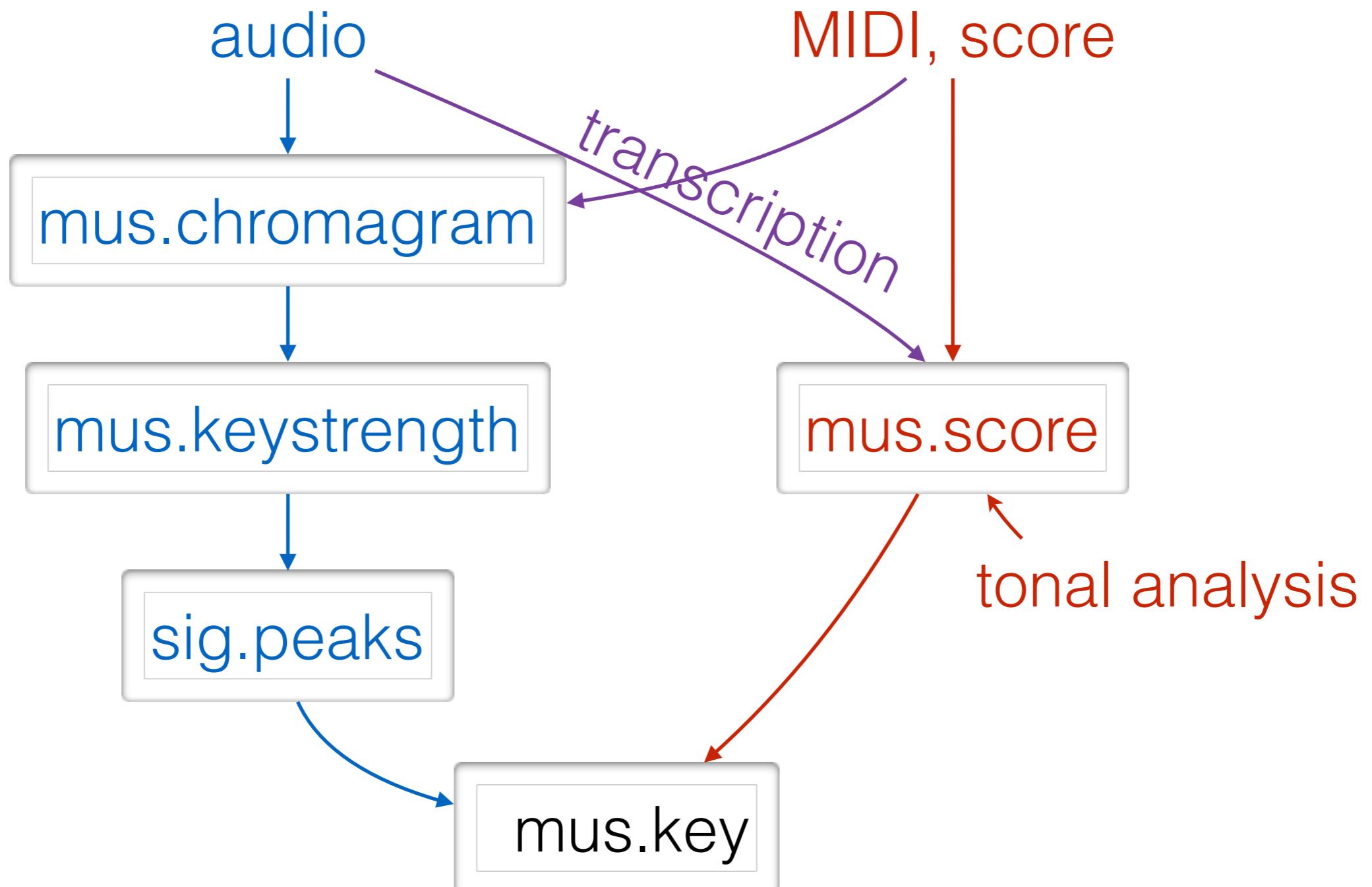
1 2 4

1 3 5

A musical score consisting of three measures. The first measure is labeled GM with a yellow box containing V. The second measure is labeled CM with a yellow box containing I. The third measure is labeled Am with a yellow box containing VI. The bass line consists of eighth notes on the B and A strings. The melody consists of sixteenth-note patterns. The first measure includes a crescendo dynamic.

A musical score consisting of three measures. The first measure is in D major (indicated by a green circle), starting with a forte dynamic (f) and a bass note. The second measure is in G major (indicated by a red circle), starting with a piano dynamic (p) and a bass note. The third measure is in C major (indicated by a red circle), starting with a forte dynamic (f) and a bass note.

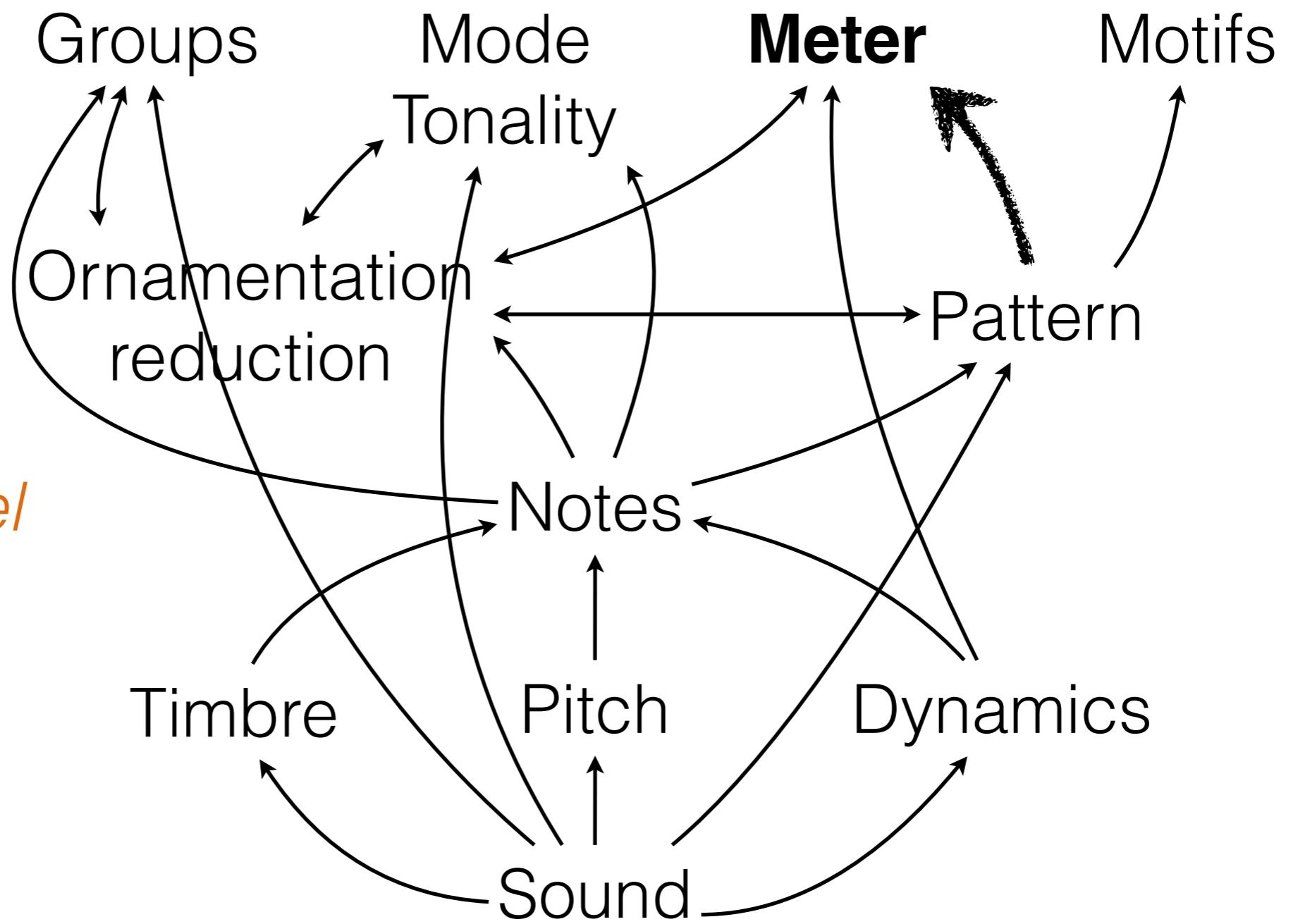
Audio / symbolic



*Structural
levels*

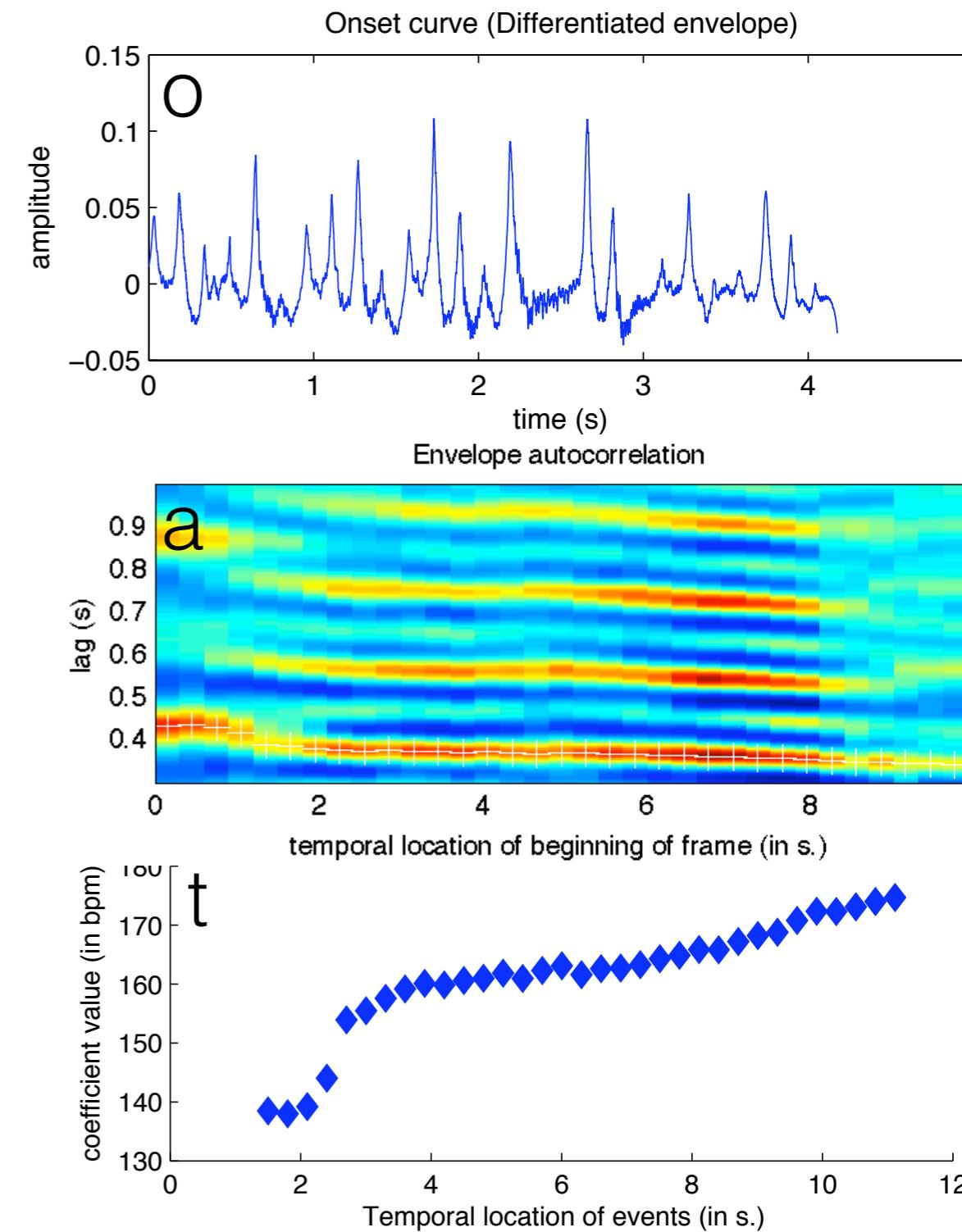
Symbolic level

Audio level

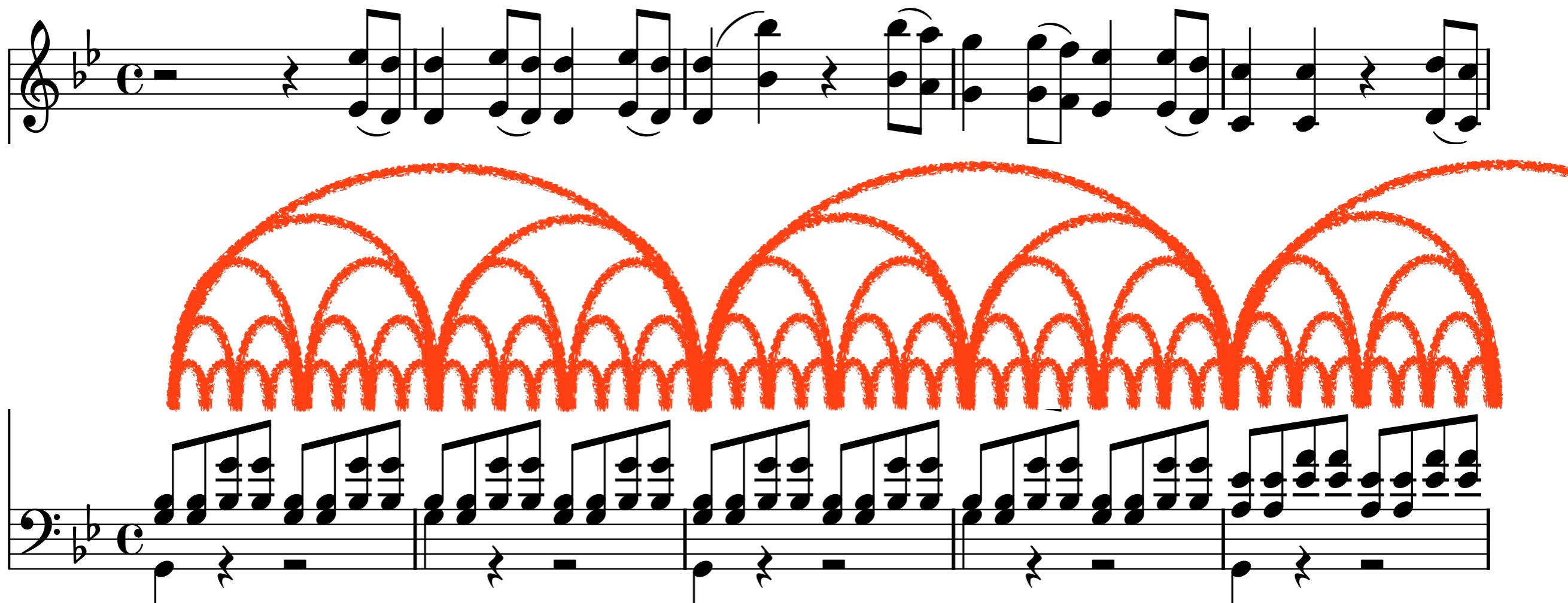


Statistical metrical analysis

- $o = \text{aud.envelope}(\text{'score.mid'})$
- $a = \text{mus.autocor}(o)$
- $t = \text{mus.tempo}(a)$
- $\text{mus.pulseclarity}(a)$
- $\text{mus.metre}(a)$

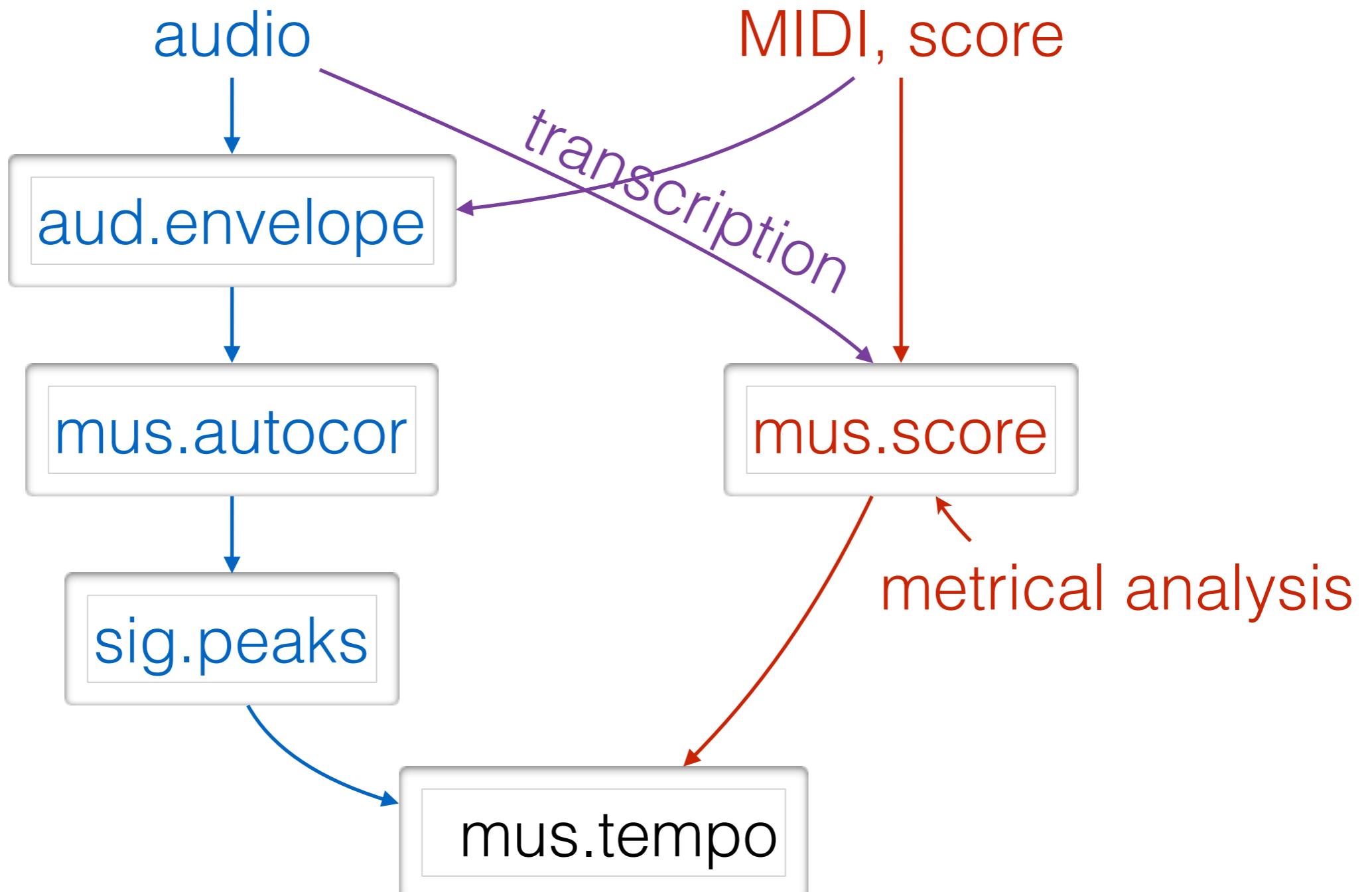


Score-level metrical analysis

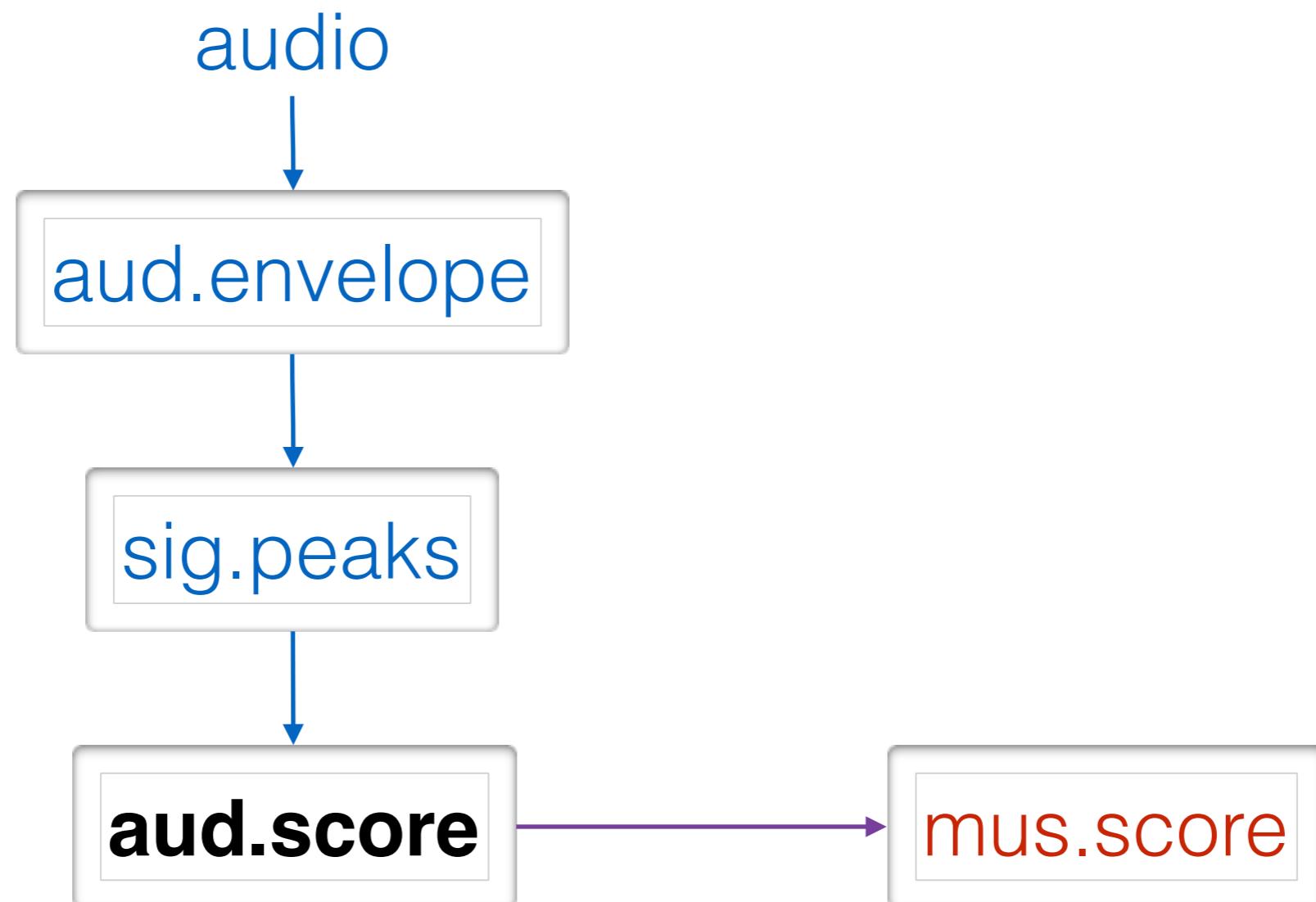


O. Lartillot, “Reflexions towards a generative theory of musical parallelism”, *Musicae Scientiae*, DF 5, 2010.

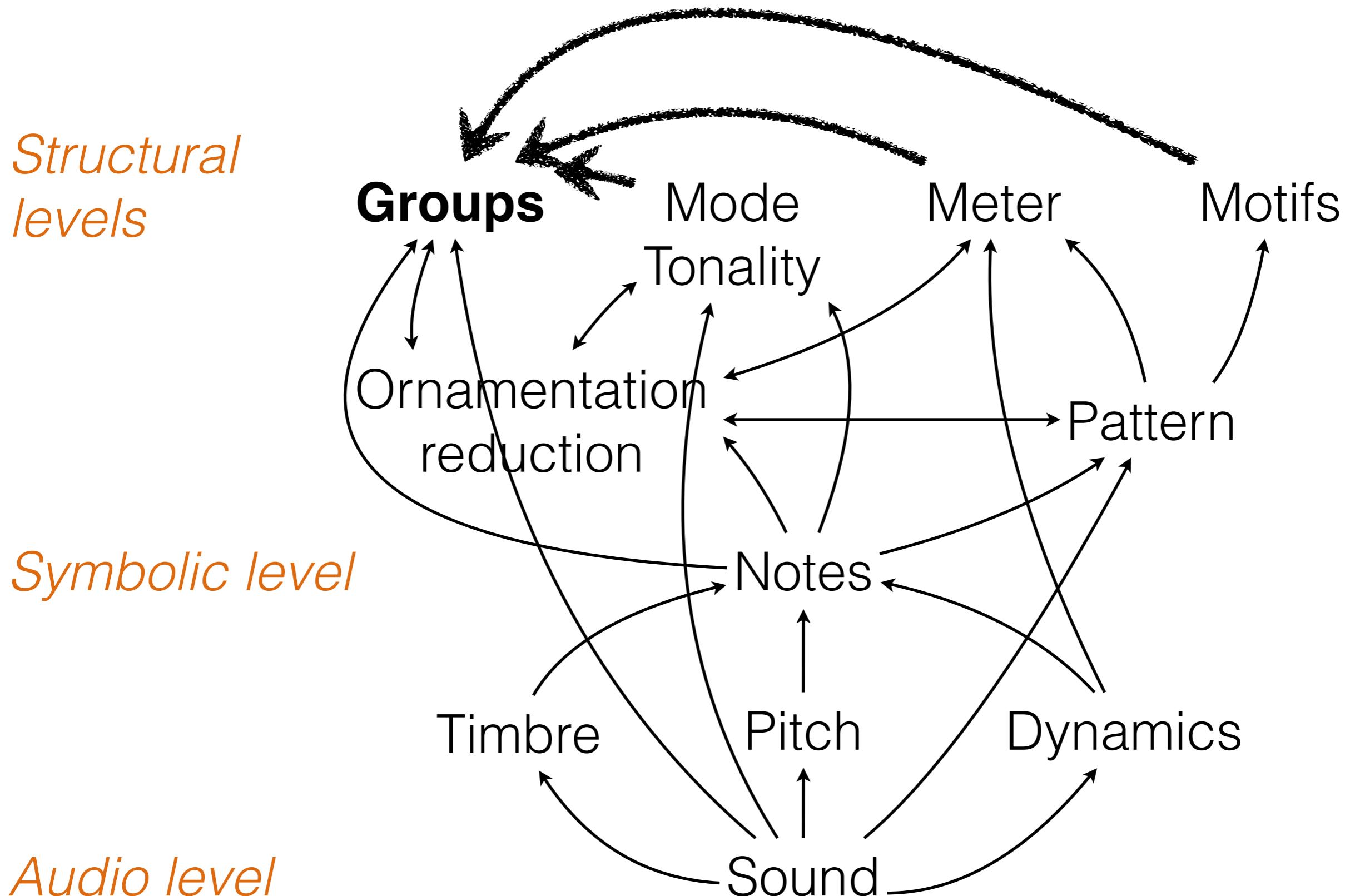
Audio / symbolic



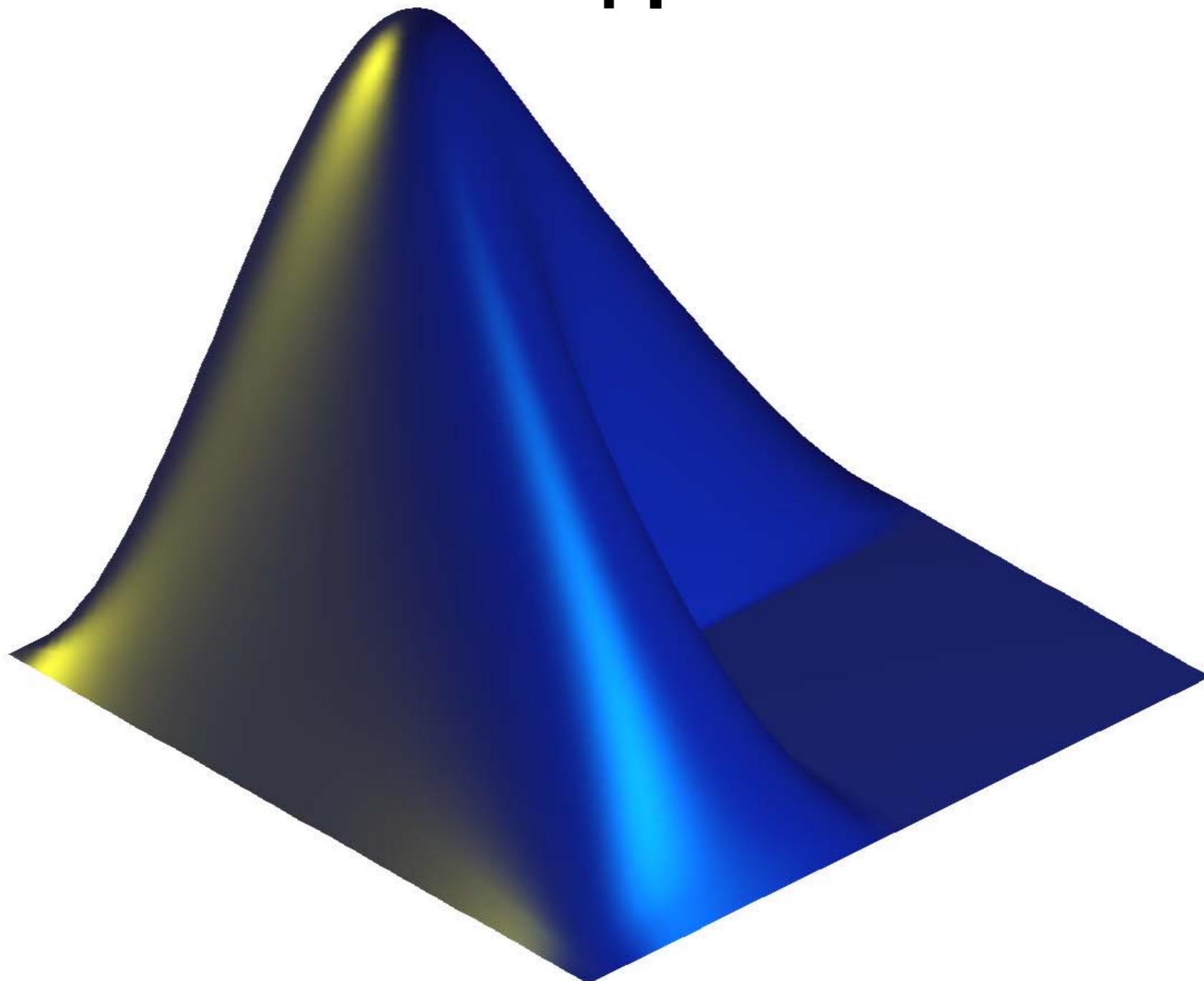
Audio / symbolic



- In future works, symbolic-based segmentation based on combination of different musical factors



4.



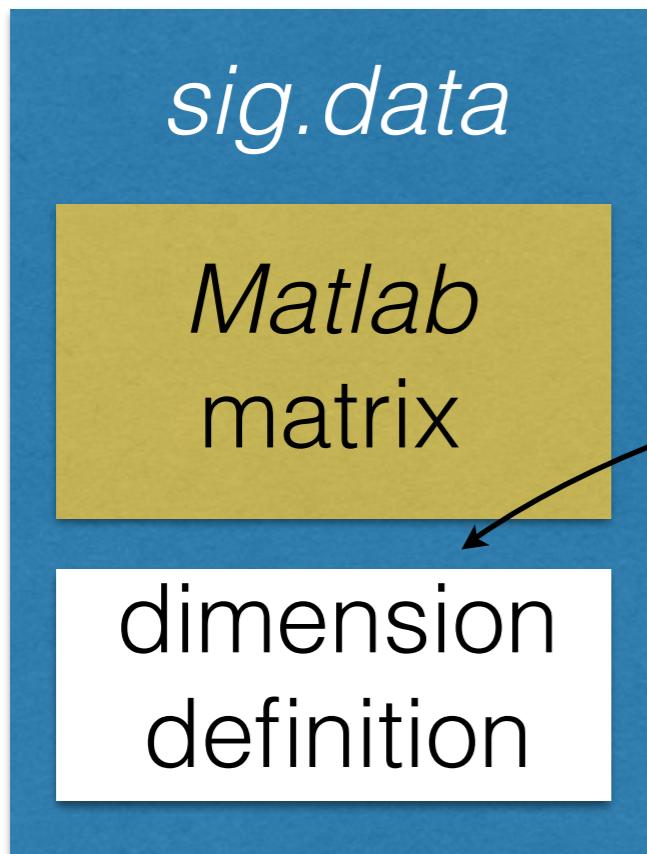
How it works

Matrix convention in *MIRtoolbox*

- In standard *Matlab* code, processing of matrix dimensions makes the code somewhat obscure.
- In *MIRtoolbox*, one single matrix convention:
 - each frame forms a column, 3d dim for channels
 - This convention should be followed everywhere in the code, and is not explicitly explained to the readers.

sig.data

- New syntactic layer on top of *Matlab* that makes operators' code simpler.



For instance:
{element,sample,channel}

- 1st dim: signal (for instance, spectrum)
- 2nd dim: successive frames
- 3rd dim: channels

sig.data

- $x.\textbf{size}(\text{'sample'})$ gives the number of samples.
- $x.\textbf{sum}(\text{'channel'})$ sums the matrix along the channel dimension.
- $x.\textbf{times}(y)$ multiplies two *sig.data* objects, respecting dimension type congruency.
- $x.\textbf{apply}(@\text{xcorr}, \{\}, \{\text{'sample'}\}, 2)$ applies *xcorr* along the sample dimension. The last argument notifies that *xcorr* does not work for matrices with more than 2 dimensions. The extra dimensions are automatically covered via loops.

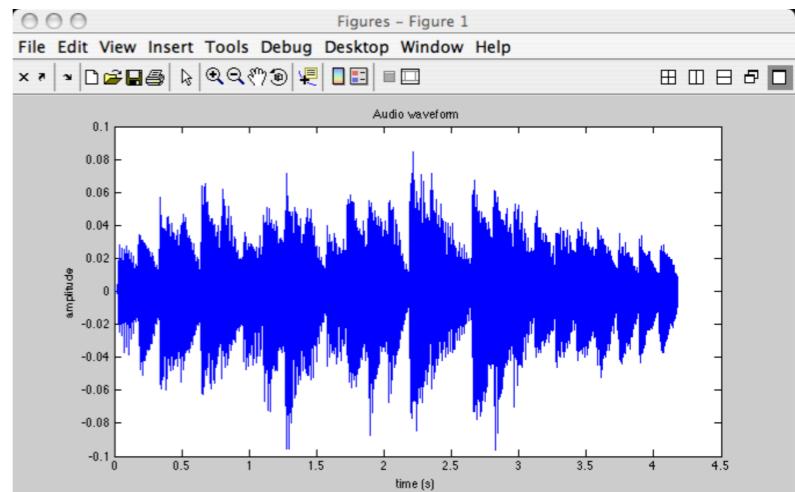
sig.design.display → *sig.design.eval*
→ *sig.signal.display*

sig.design object, storing only the data flow graph

- ***a*** = *sig.input('bigfile', ...)* → ***sig.design.display*** = ***sig.design.eval***, which outputs a ***sig.signal*** and calls ***sig.signal.display***
 - *d* = ***sig.ans***
 - *d* = *a.eval*; → the outputted ***sig.signal***
 - *d* → ***sig.design.eval*** outputs a ***sig.signal***
- sig.signal.display***

- `a = sig.input('myfile', 'Center')`

`sig.signal.display`



sig.signal

Ydata
yname
yunit
Xaxis
xsampling
Sstart
Srate
Ssize
date
ver
design
...

`sig.data`

'audio'

0

0

44100

2205

15.10.2014

R2014b,
MiningSuite 0.8

`sig.axis`

xname

xstart

xunit

1

`sig.design`

`a = sig.input('myfile', 'Center')`

Information such as time positions
are regenerated on the fly.

`sig.unit`

xname

origin

rate

generator

finder

0

0

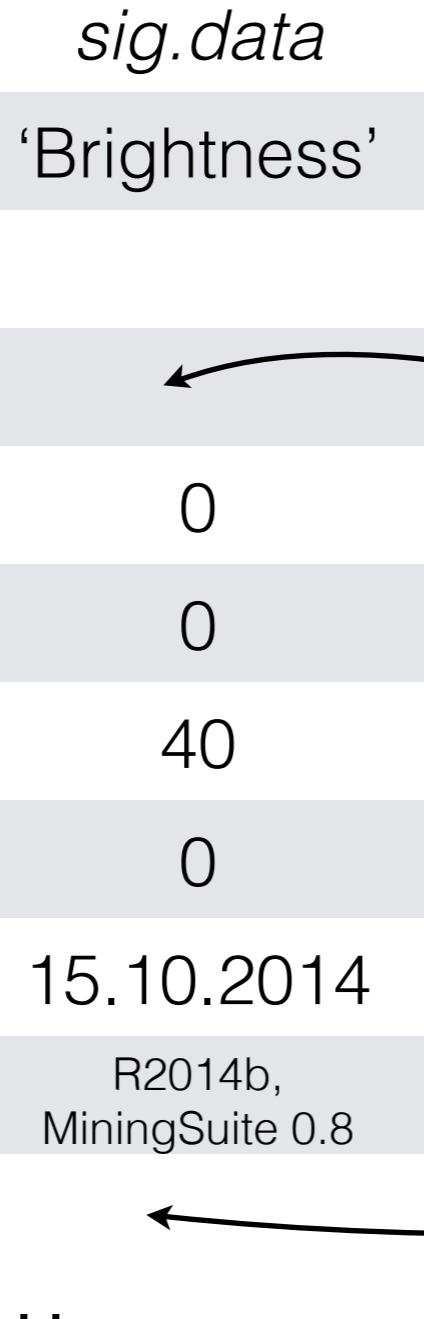
@..

@..

- `b = sig.brightness(a, 'Frame')`

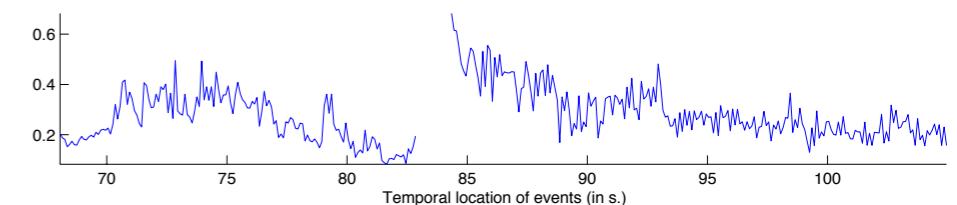
sig.signal

Ydata	<i>sig.data</i>
yname	'Brightness'
yunit	
Xaxis	
xsampling	0
Sstart	0
Srate	40
Ssize	0
date	15.10.2014
ver	R2014b, MiningSuite 0.8
design	...



xname	
xstart	1
xunit	

xname	0
origin	0
rate	0
generator	@..
finder	@..



- `b = sig.spectrum(a, 'Frame')`

sig.signal.display

sig.spectrum

sig.signal

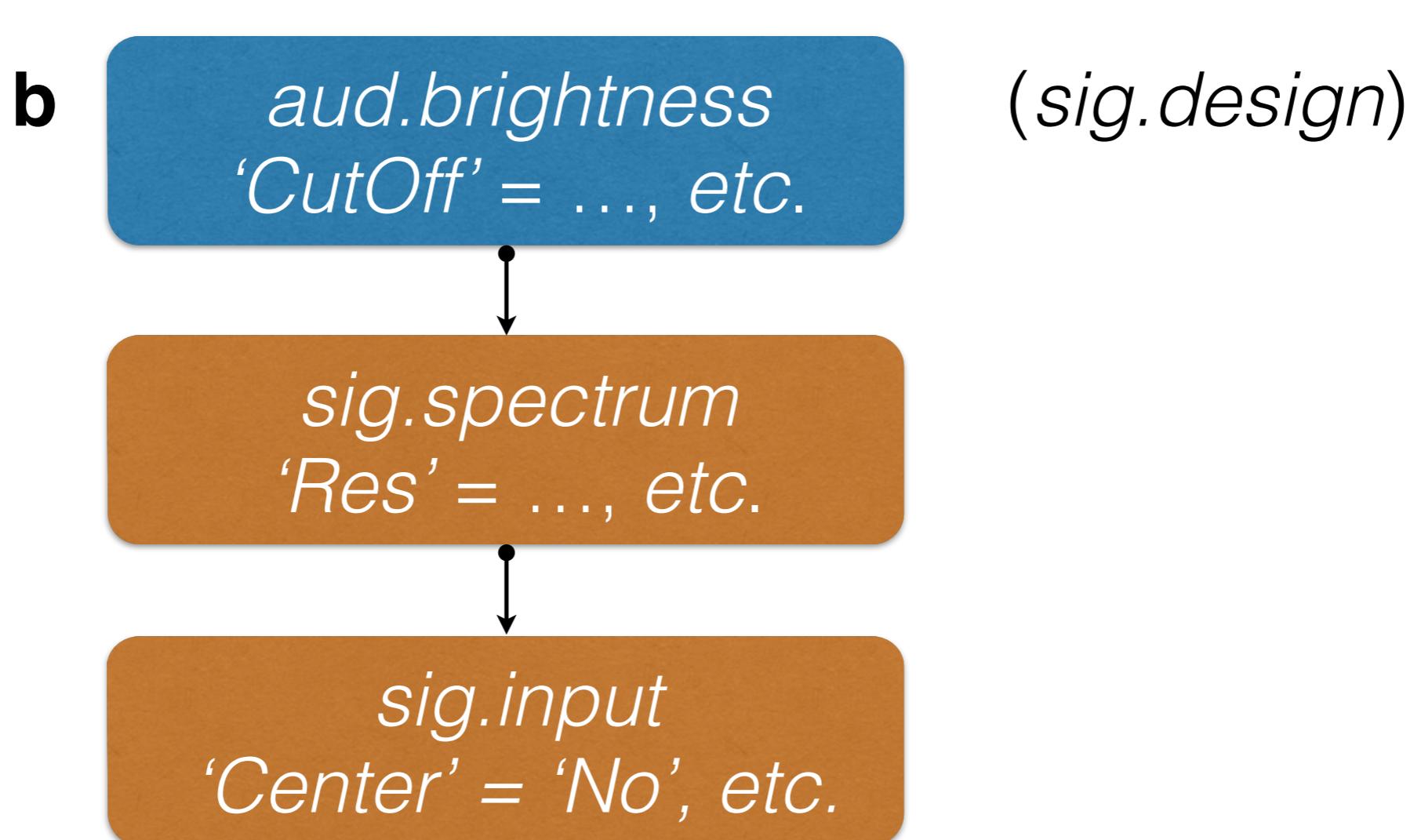
power	1
log	0
phase	<i>sig.data</i>

Ydata	<i>sig.data</i>
yname	'Spectrum'
yunit	
Xaxis	<i>sig.axis</i>
xsampling	0.1682
Sstart	0
Srate	40
Ssize	0
date	15.10.2014
ver	R2014b, MiningSuite 0.8
design	

init

deploy the **implicit data flow** prior to the actual operator

- $b = aud.brightness('ragtime', 'Frame')$



sig.operate operators' main routine

+aud/brightness.m:

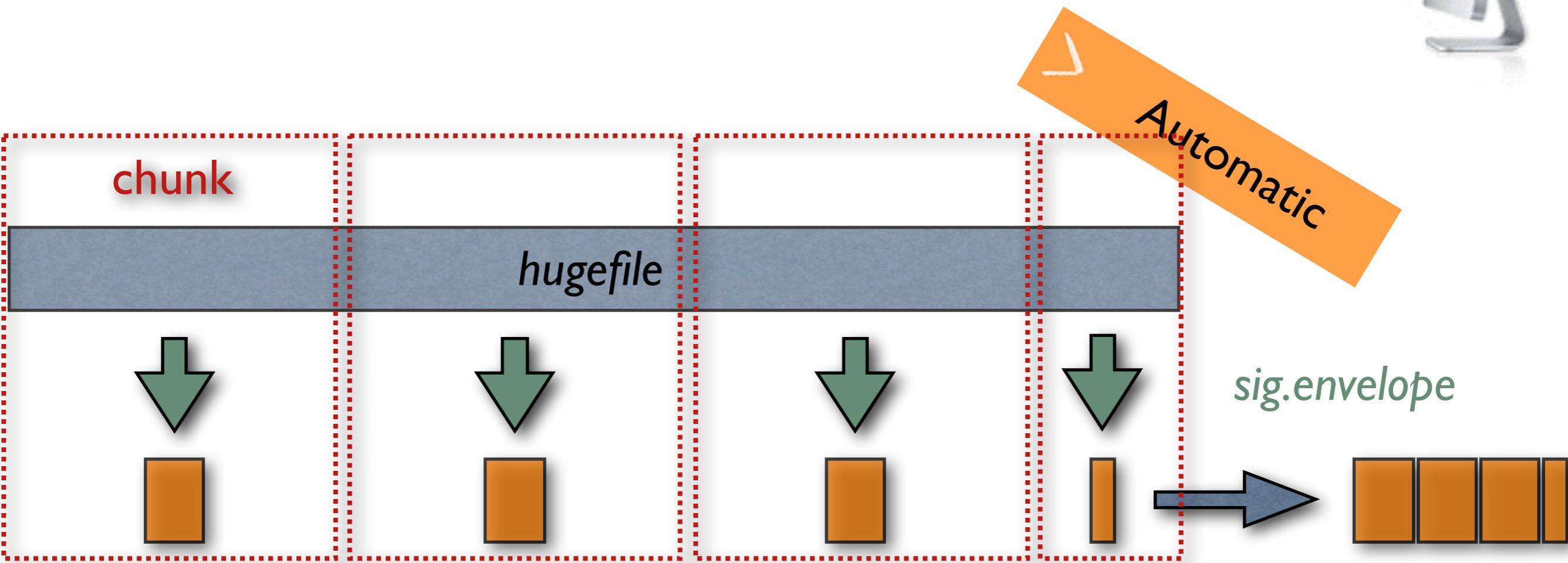
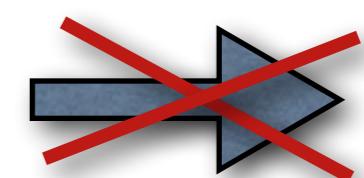
```
function varargout = brightness(varargin)
    varargout = sig.operate('aud', 'brightness', options, ...
        @init, @main, varargin, 'plus');
end
```

aud.brightness ('ragtime', 'Frame')

- sig.operate*
- first parses the **arguments** in the call,
 - then creates a data flow design:
 - starts with implicit data flow (@**init**)
 - ends with @**main** routine.

memory management

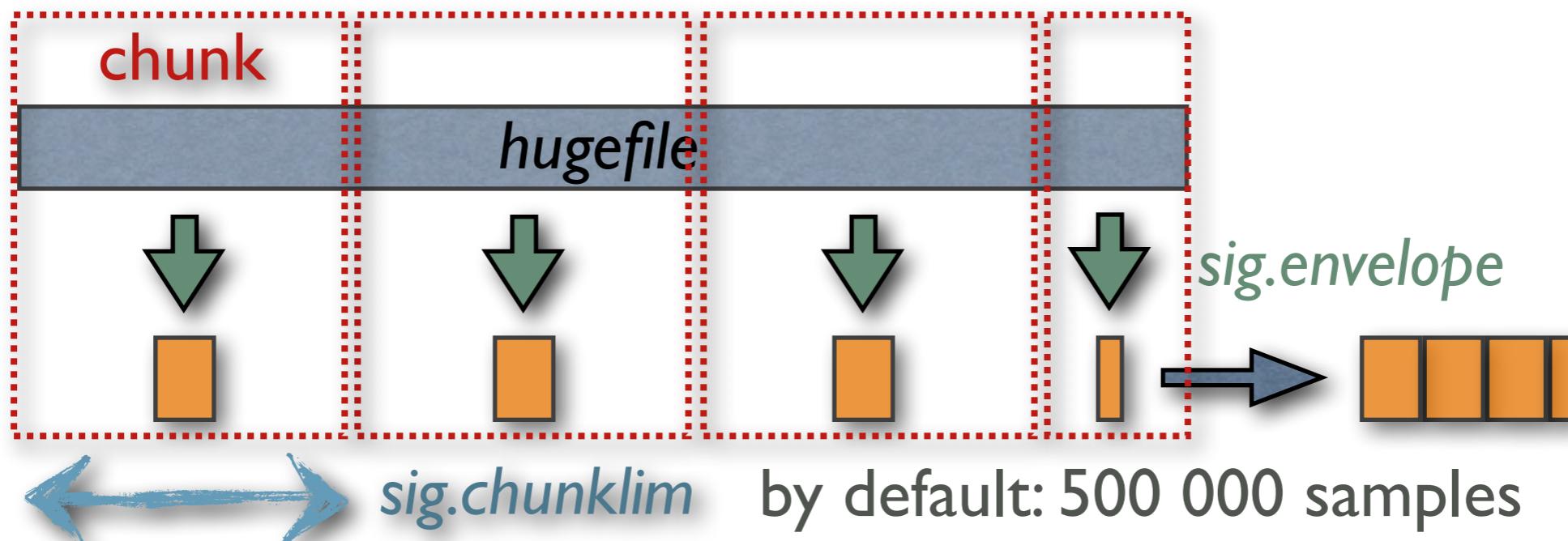
sig.envelope('hugefile');



sig.chunklim

chunk size limitation

sig.envelope('hugefile');



► If memory overflow problems, decrease `sig.chunklim`:

`sig.chunklim(50000)` set to 50 000 samples

sig.design.eval

non-framed-based evaluation

- $a = \text{sig.input};$
- $c = \text{aud.mfcc}(a);$

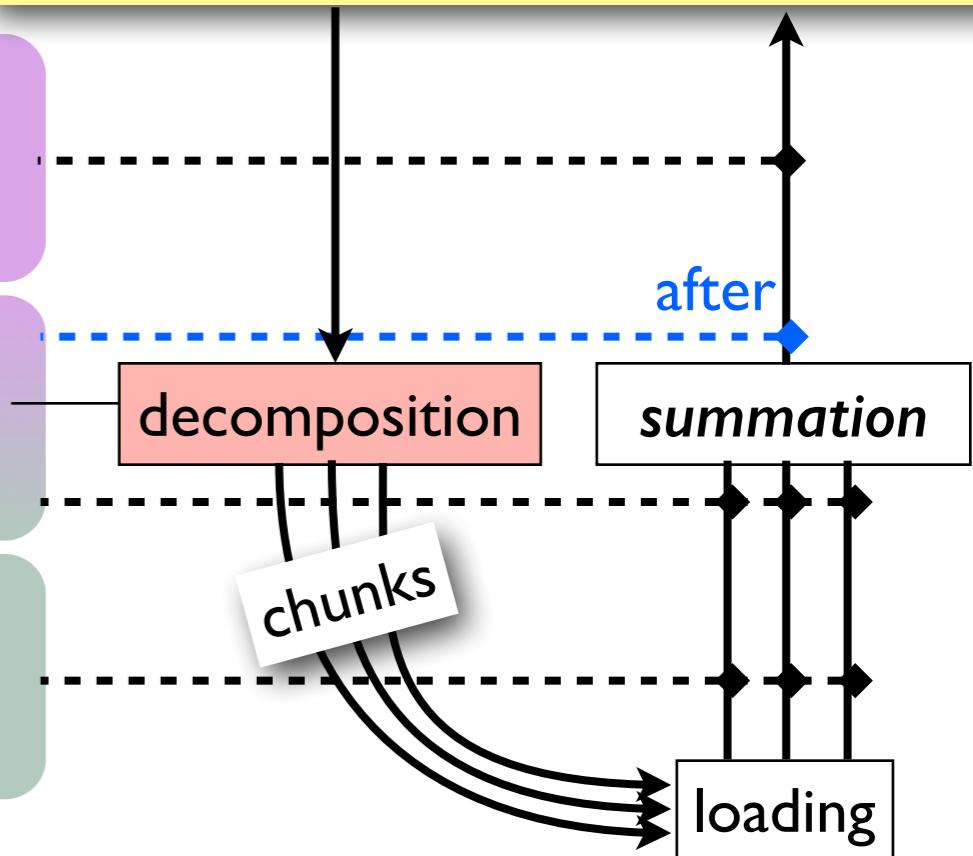
c

```
sig.design      type=aud.mfcc  
options: Rank=1:13, Delta=0, ...
```

a

```
sig.design      type=sig.input  
options: Sampling=11025, ...
```

eval_each(c,'song1.wav')



sig.design.eval

framed-based evaluation

- $a = \text{sig.input};$
- $c = \text{aud.mfcc}(a);$

c

```
sig.design      type=aud.mfcc  
options: Rank=1:13, Delta=0, ...
```

```
sig.design      type=aud.spectrum  
options: Win='hamming', ...
```

```
sig.design      type=sig.frame  
options: Length=.05, Hop=.5 ...
```

a

```
sig.design      type=sig.input  
options: Sampling=11025, ...
```

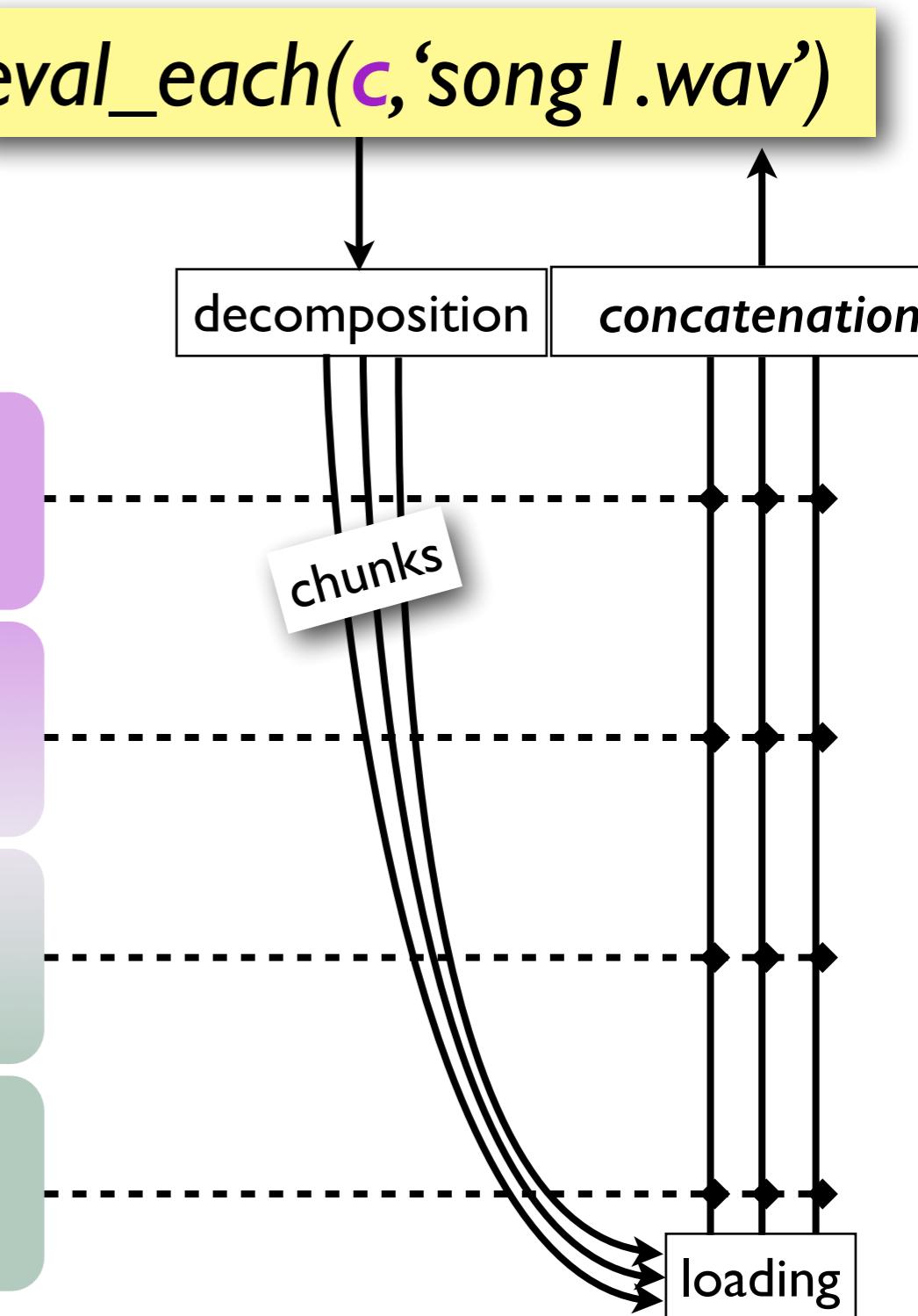
eval_each(**c**, 'song1.wav')

decomposition

concatenation

chunks

loading



sig.design.eval

framed based evaluation

- `e = sig.envelope`
- `a = sig.autocor(e, 'Frame');`

`eval_each(a, 'song1.wav')`

a

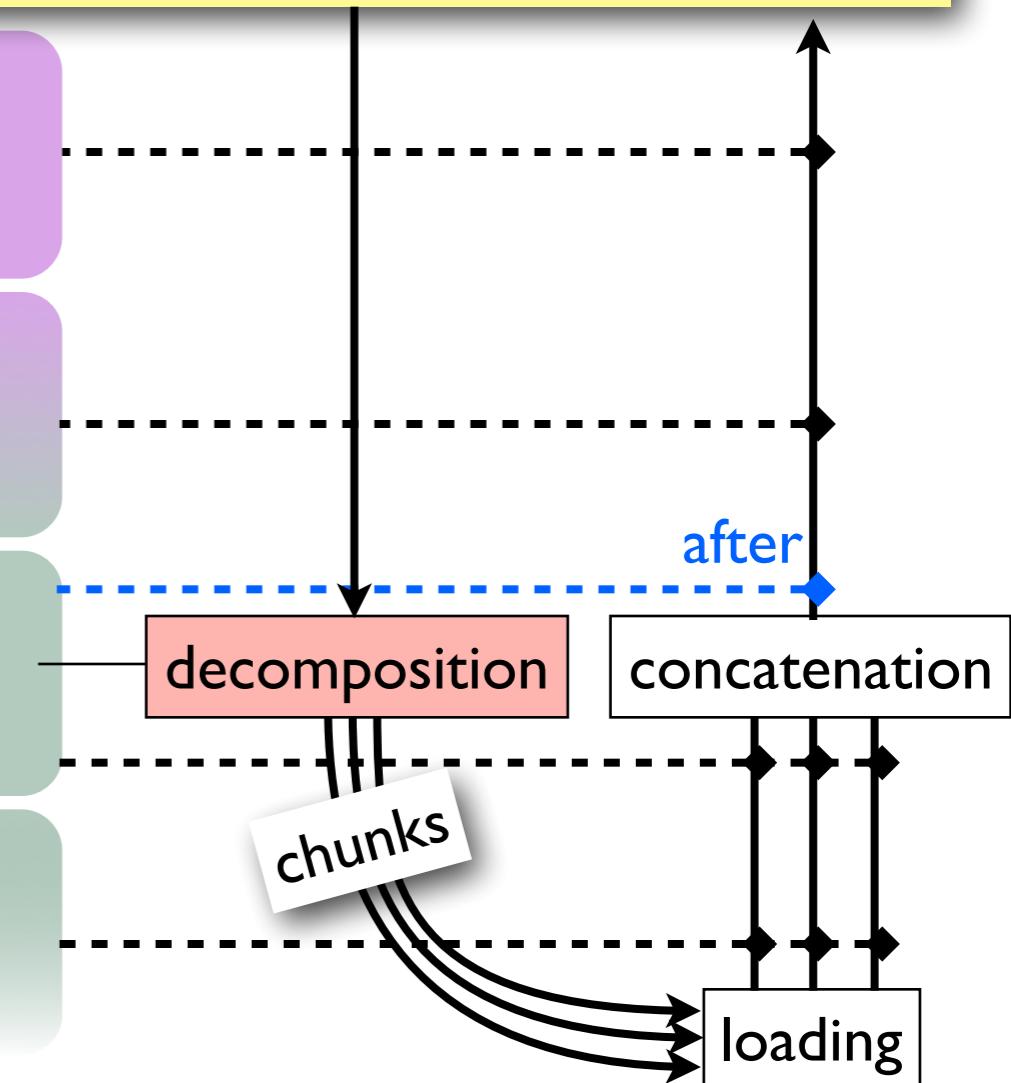
`sig.design type=sig.autocor
options: Rank=1:13, Delta=0, ...`

`sig.design type=sig.frame
options: Win='hamming', ...`

e

`sig.design type=sig.envelope
options: Length=.05, Hop=.5 ...`

`sig.design type=sig.input
options: Sampling=11025, ...`



seq.sequence

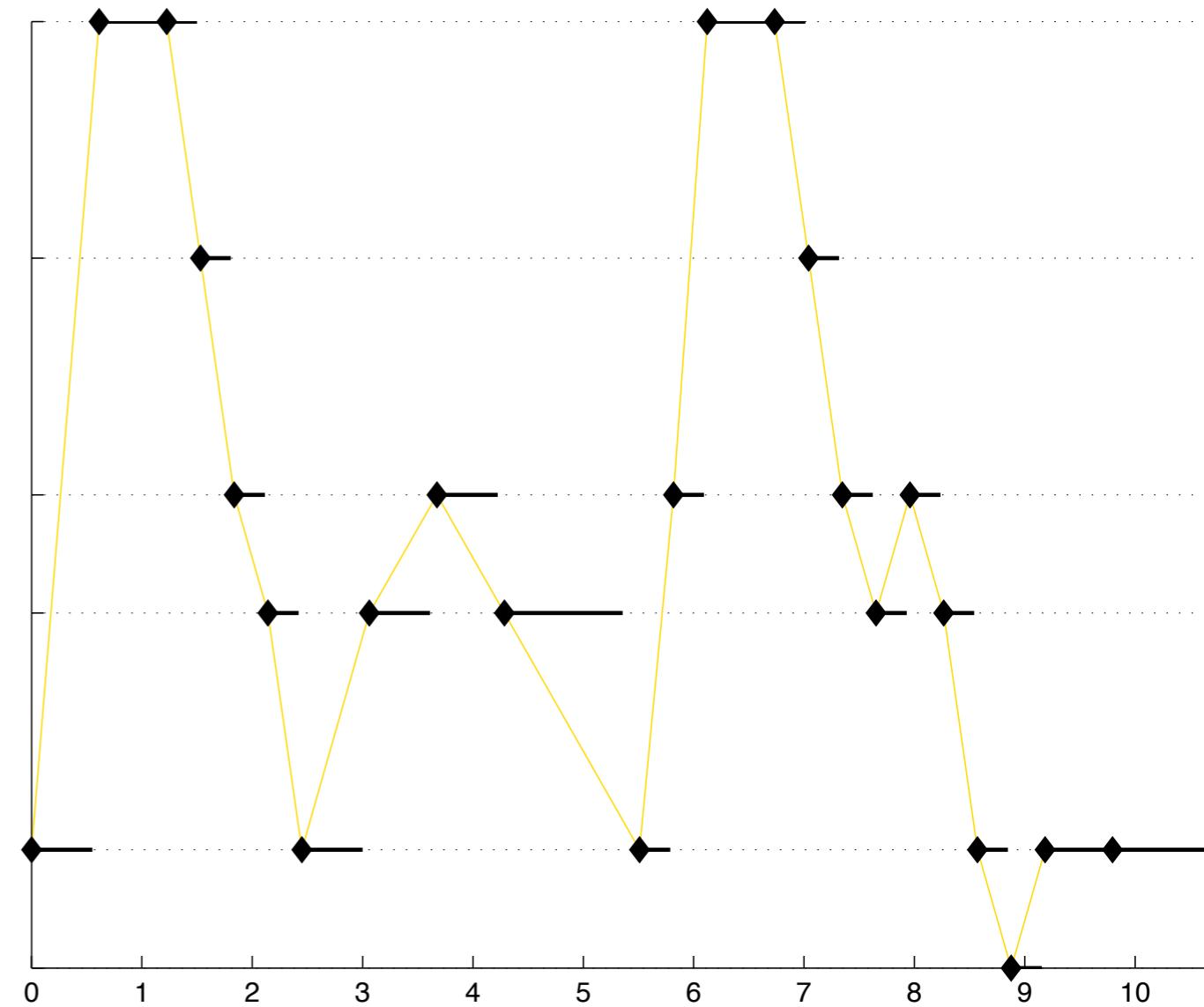
symbolic sequence

- Routines for symbolic sequence management.
Elements of *seq.sequence* of class *seq.event*.
- ...

seq.event note characterization

Each note is a *seq.event*:

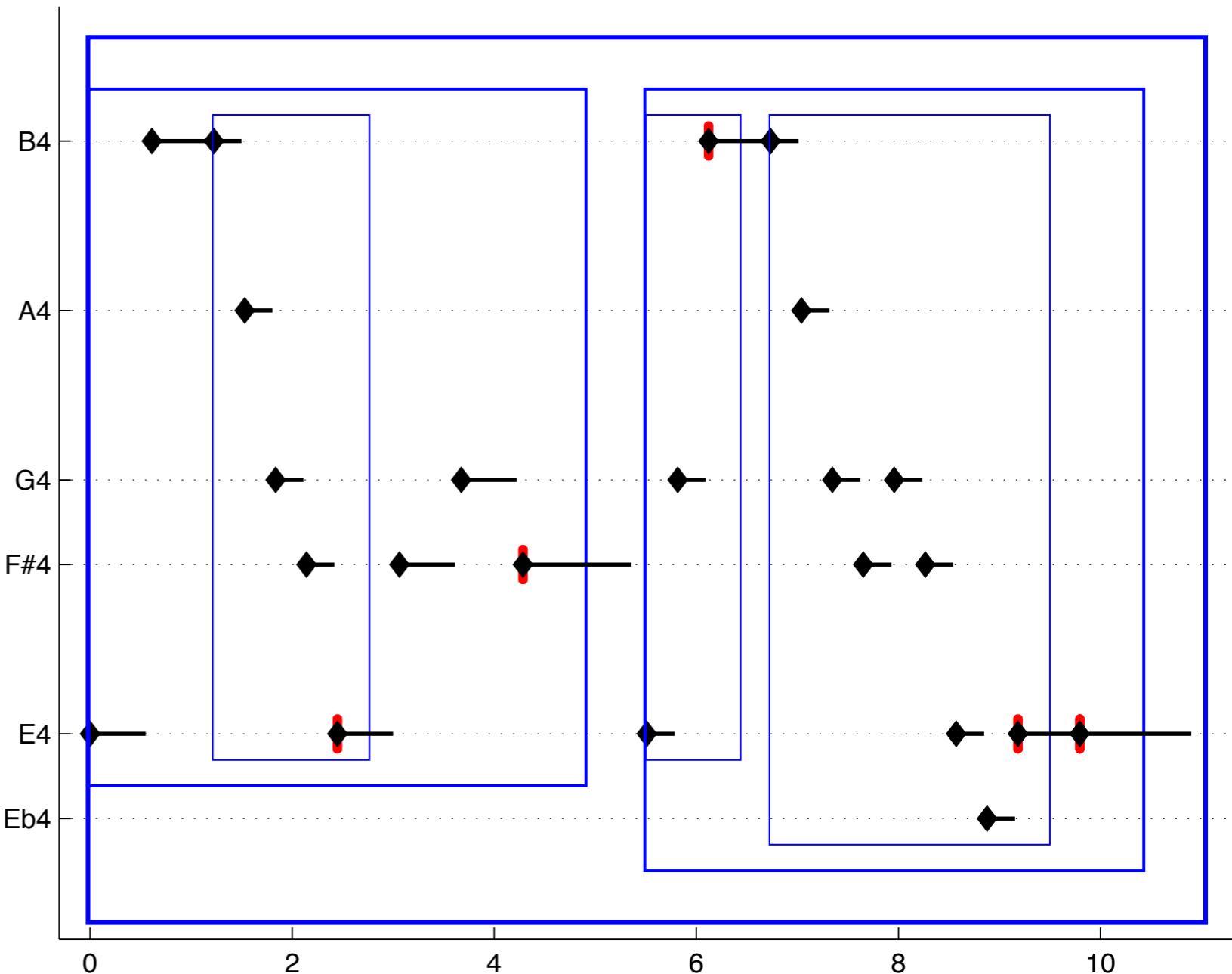
- characterization of the note: musical parameter (instance of class *seq.paramstruct*)
- previous note
- next note



`mus.score(..., 'Group')` hierarchical grouping

`mus.score('laksin.mid',
'Group')`

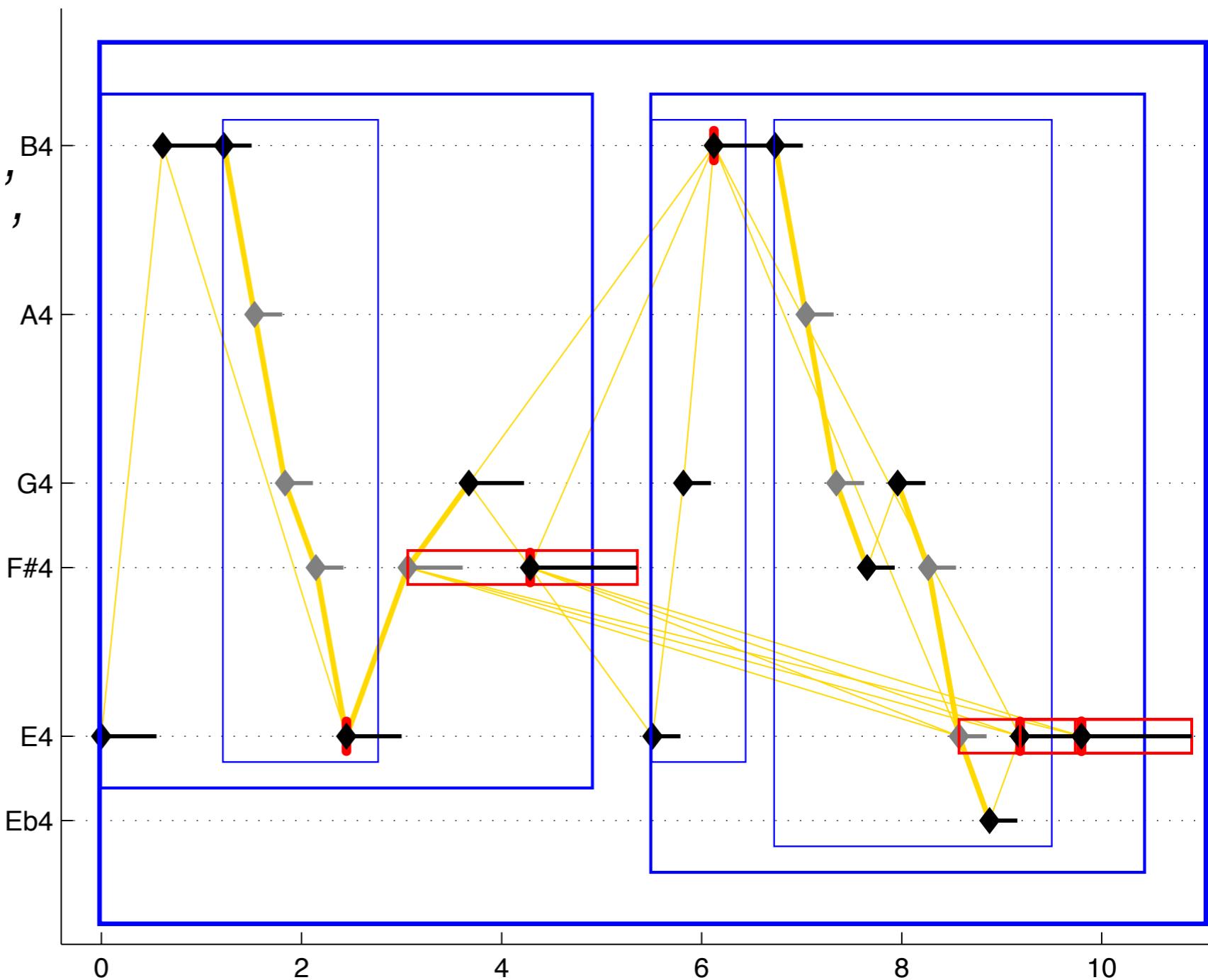
Groups, like notes, are instances of `seq.event`



mus.score(..., 'Reduce') ornamentation reduction

*mus.score('laksin.mid',
'Group', 'Reduce')*

Each syntagmatic
relation between 2
notes is instance of
seq.syntagm



mus.paramstruct

musical dimensions

each note:

- chromatic pitch
- chromatic pitch class
- diatonic pitch (letter, accident, octave)
- diatonic pitch class
- onset, offset times (in s.)
- metrical position
- channel
- harmony, etc.

interval between notes:

- chromatic pitch interval
- chromatic pitch interval class
- diatonic pitch interval (number, quality)
- diatonic pitch interval class
- gross contour
- inter onset interval
- rhythmic value

more general

seq.param parameter management

- $\text{common}(p1, p2)$ returns the common parametric description
- $p1.\text{implies}(p2)$ tests whether $p2$ is more general than $p1$

seq.syntagm

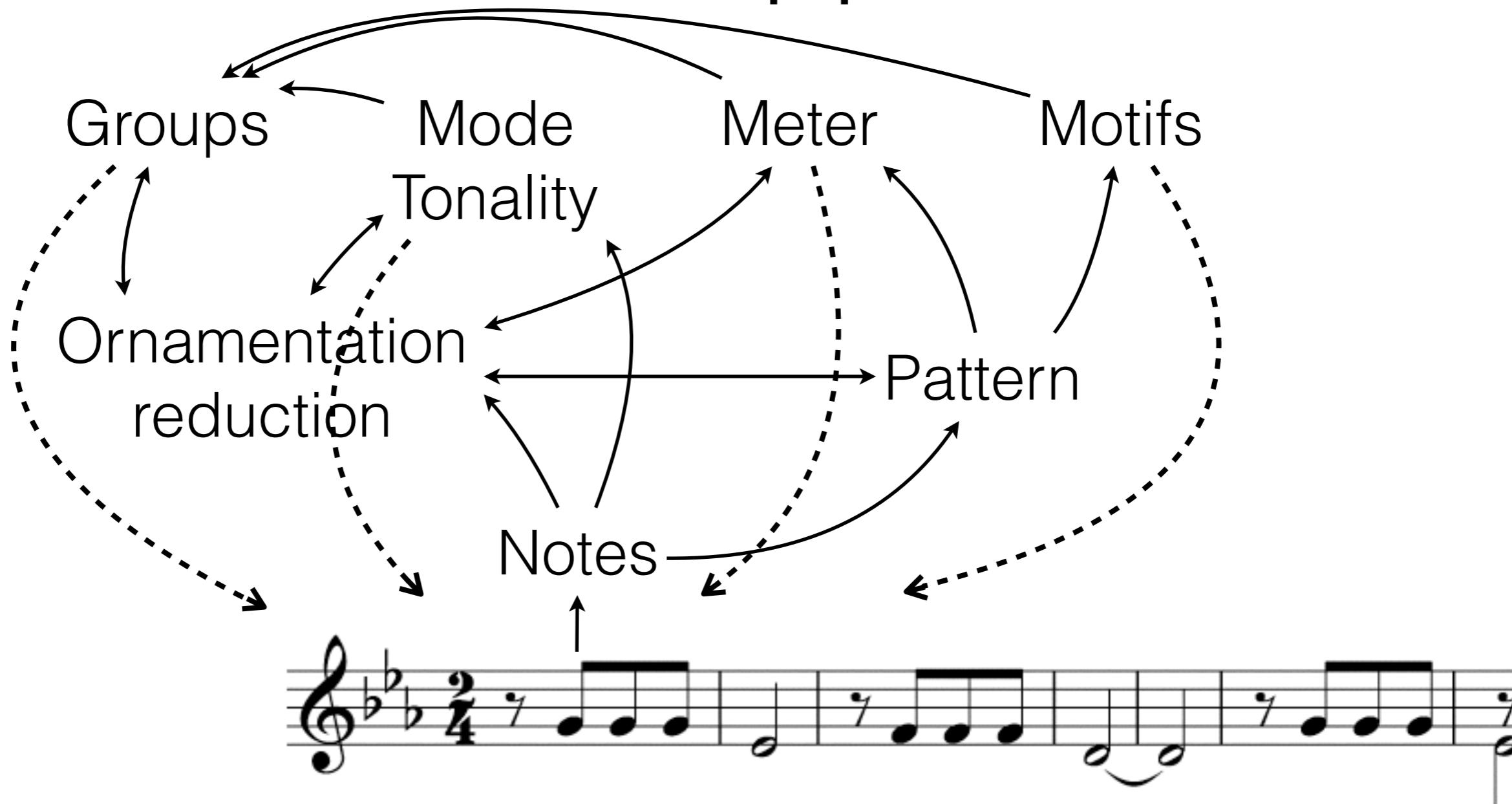


$s = \text{seq.syntagm}(n1, n2)$

- $s.\text{param}$ is automatically computed from $n1.\text{param}$ and $n2.\text{param}$

mus.score

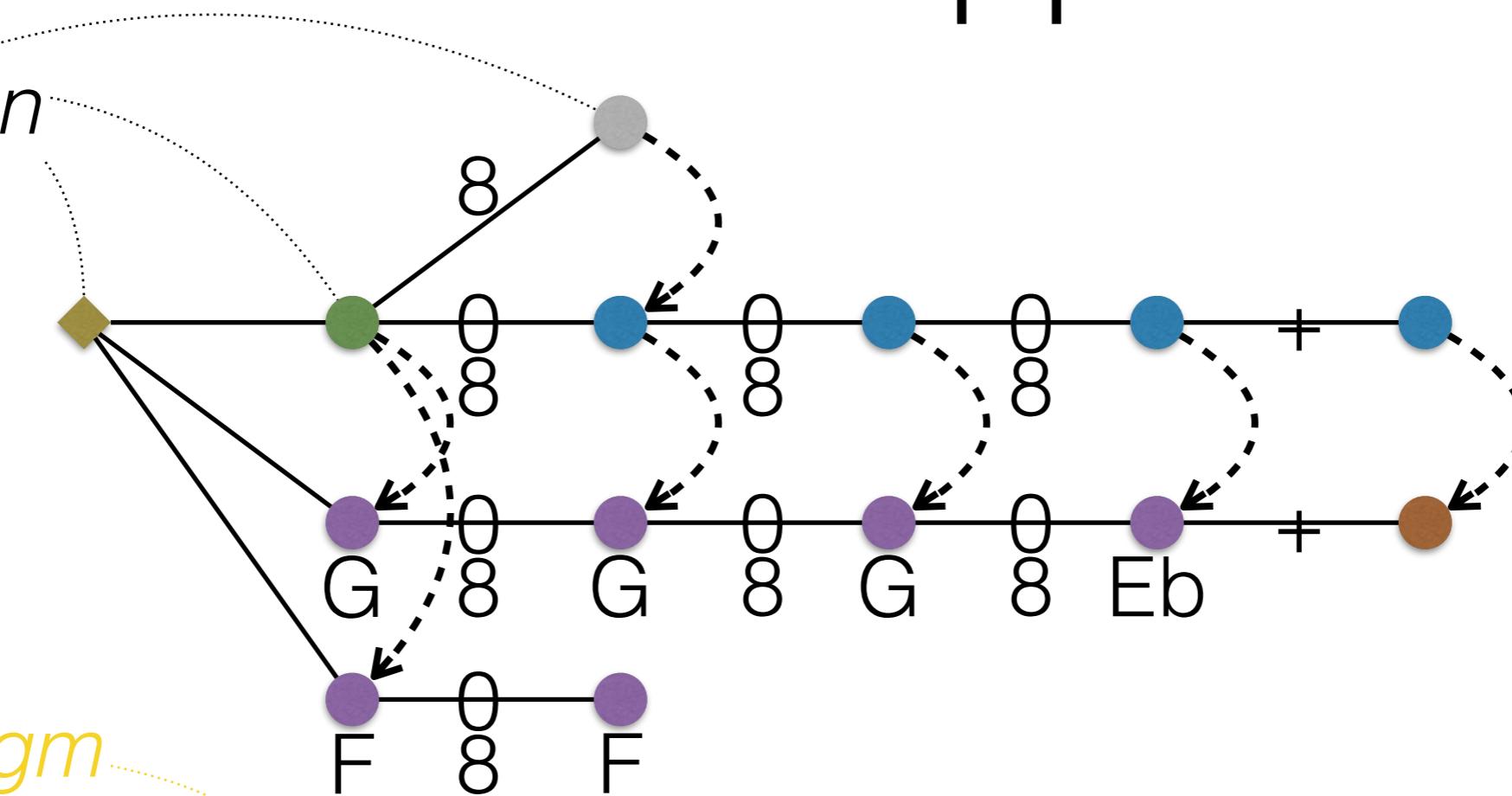
incremental approach



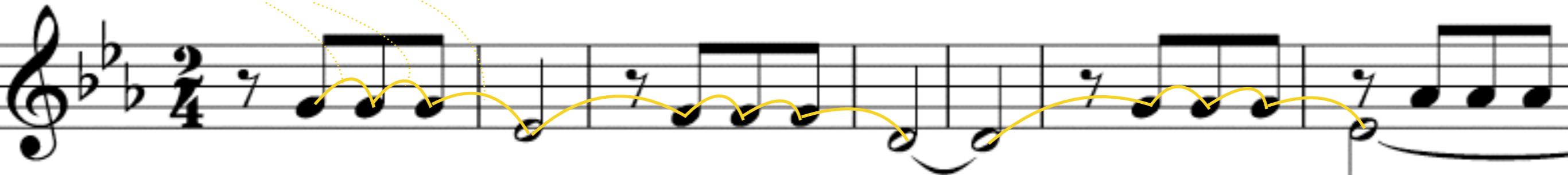
Each successive note is progressively integrated to all musical analyses, driving interdependencies.

mus.score incremental approach

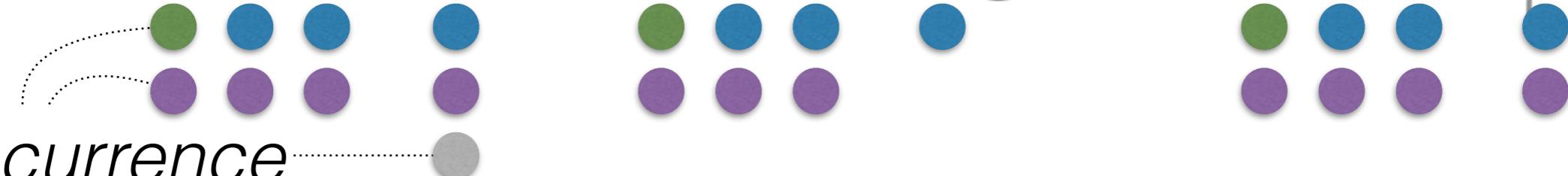
pat.pattern



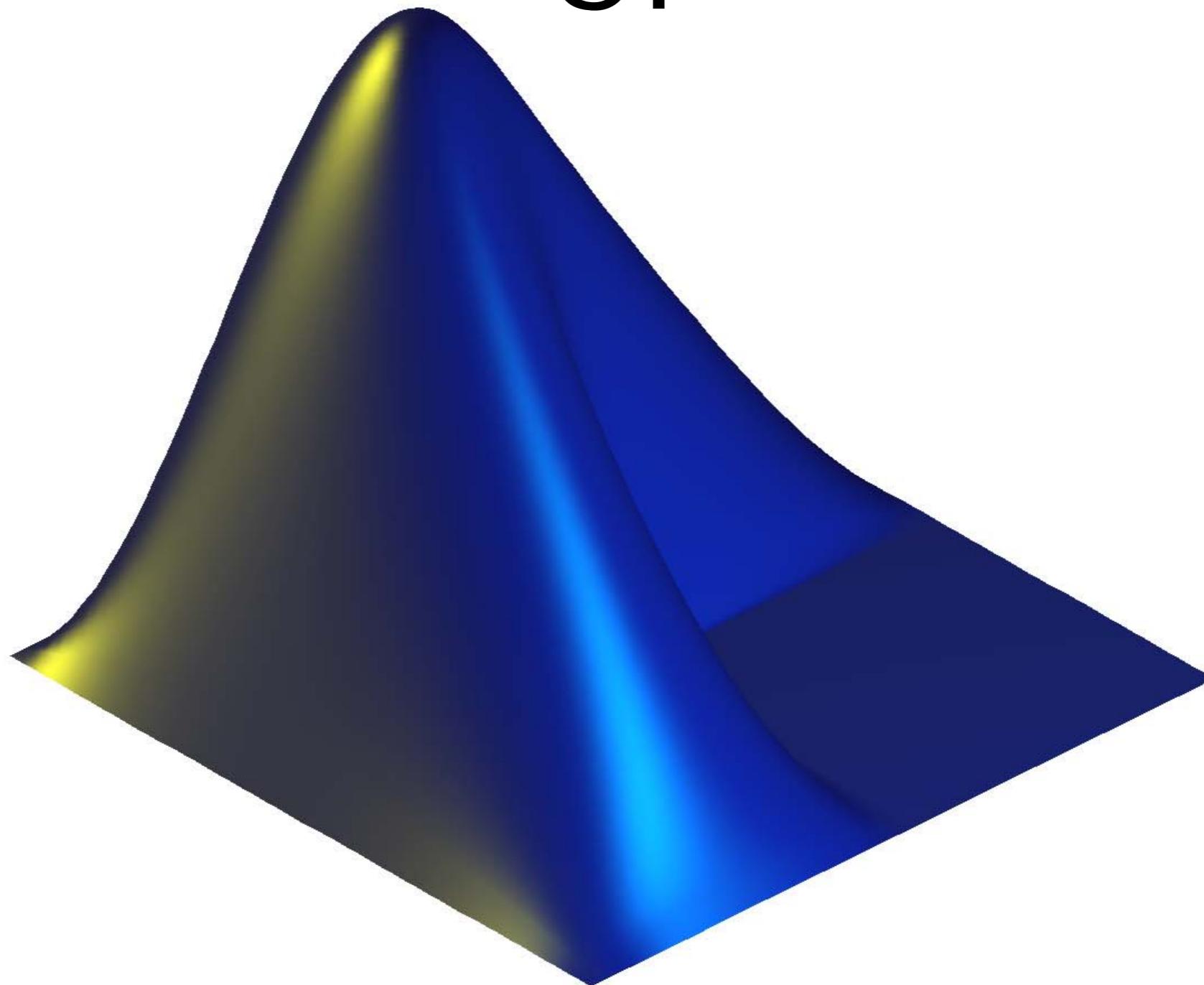
seq.syntagma



pat.occurrence



5.



How you can contribute

code structure of an operator

my_operator.m

```
function varargout = sig.my_operator(varargin)
    varargout = sig.operate('sig','my_operator',...
        options,@init,@main,varargin,'plus');
end
```

options specification

my_operator(filename, ‘Threshold’, 100, ‘Normalize’)

```
thres.key = ‘Threshold’;
```

```
thres.type = ‘Numeric’;
```

```
three.default = 50;
```

```
options.thres = thres;
```

```
norm.key = ‘Normalize’;
```

```
norm.type = ‘Boolean’;
```

```
norm.default = 0;
```

```
options.norm = norm;
```

options specification

my_operator(filename, ‘Domain’, ‘Symbolic’)

```
domain.key = ‘Domain’;
```

```
domain.type = ‘String’;
```

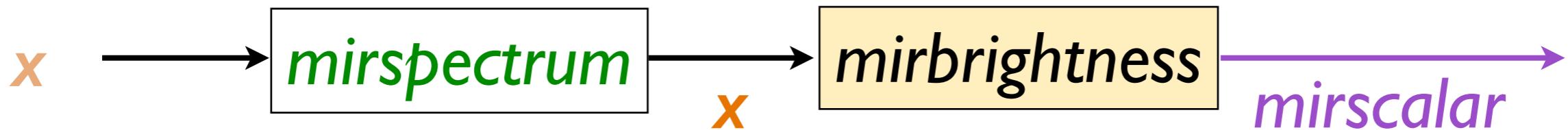
```
domain.choice = {‘Signal’, ‘Symbolic’};
```

```
domain.default = ‘Signal’;
```

```
options.domain = domain;
```

init

implicit flowchart specification



```
varargout = mirfunction(@mirbrightness, x, varargin, nargout, specif, @init,  
@main);
```

```
function [x type] = init(x, option)  
if not(isamir(x, 'mirspectrum'))  
    x = mirspectrum(x);  
end  
type = 'mirscalar';
```

main

function main algorithm

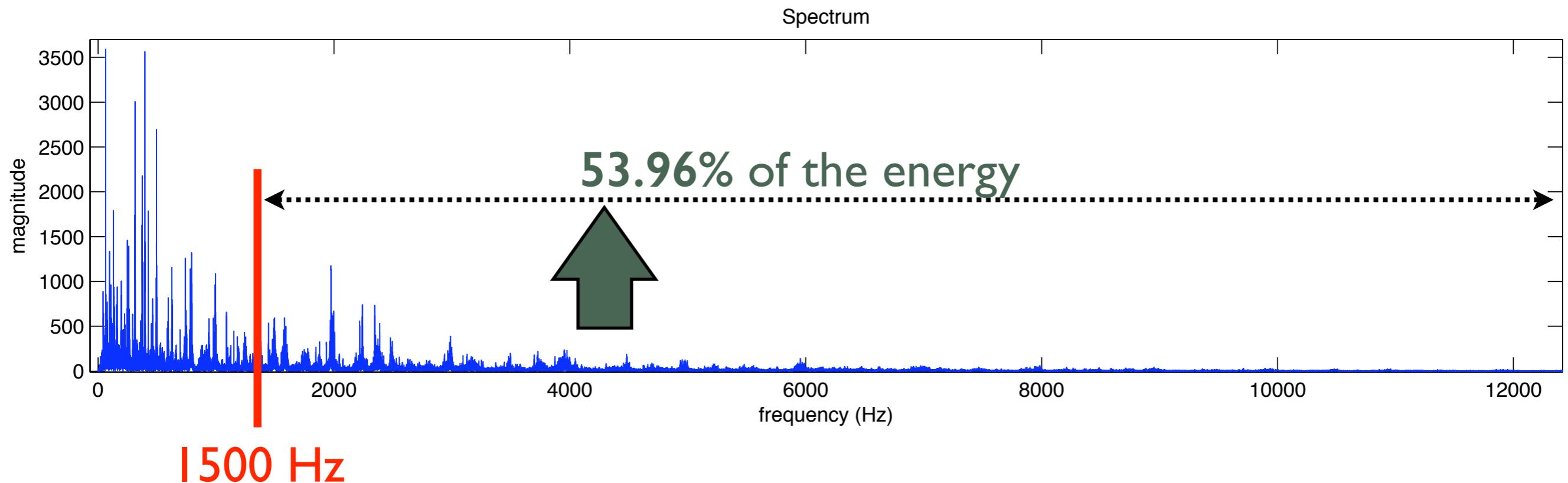
```
varargout = mirfunction(@mirbrightness, x, varargin, nargout, specif, @init,
@main);
```

```
function b = main(s, option)
m = get(s, 'Magnitude');
f = get(s, 'Frequency');
v = mircompute(@algo, m, f, option.cutoff, option.unit);
b = mirscalar(s, 'Data', v, 'Title', 'Brightness');
```

```
function v = algo(m, f, k, u)
v = sum(m(f(:, I)>k, :)) ./ sum(m);
if strcmp(u, '%')      v = v*100;    end
```

aud.brightness

example of function implementation



- *aud.brightness(..., 'CutOff', 1500)* (in Hz)
- *aud.brightness(..., 'Unit', u)* u = '/1' or '%'

```

function varargout = brightness(varargin)

    varargout =
sig.operate('aud','brightness',...
    initoptions,@init,@main,varargin,'plus');

end

```

```

function options = initoptions

    options = sig.signal.signaloptions(.05,.5);

    cutoff.key = 'CutOff';

    cutoff.type = 'Numeric';

    cutoff.default = 1500;

    options.cutoff = cutoff;

end

```

```

function [x type] = init(x,option,frame)

    x = sig.spectrum(x);

    type = 'sig.signal';

end

```

```

function out = main(in,option,postoption)

    x = in{1};

    if ~strcmpi(x.yname,'Brightness')

        res =
sig.compute@routine,x.Ydata,x.xdata,option.cutoff);

        x = sig.signal(res{1},'Name','Brightness',...
            'Srate',x.Srate,'Ssize',x.Ssize);

    end

    out = {x};

end

```

```

function out = routine(d,f,f0)

    e = d.apply@algo,{f,f0},{'element'},3);

    out = {e};

end

```

```

function y = algo(m,f,f0)

    y = sum(m(f > f0,:,:)) ./ sum(m);

end

```

[https://code.google.com/p/
miningsuite/](https://code.google.com/p/miningsuite/)

- All releases, *SubVersion* repository
- User's Manual and documentations in wiki environment
- Mailing lists: news, discussion, commitments, etc.
- Tickets to issue bug reports

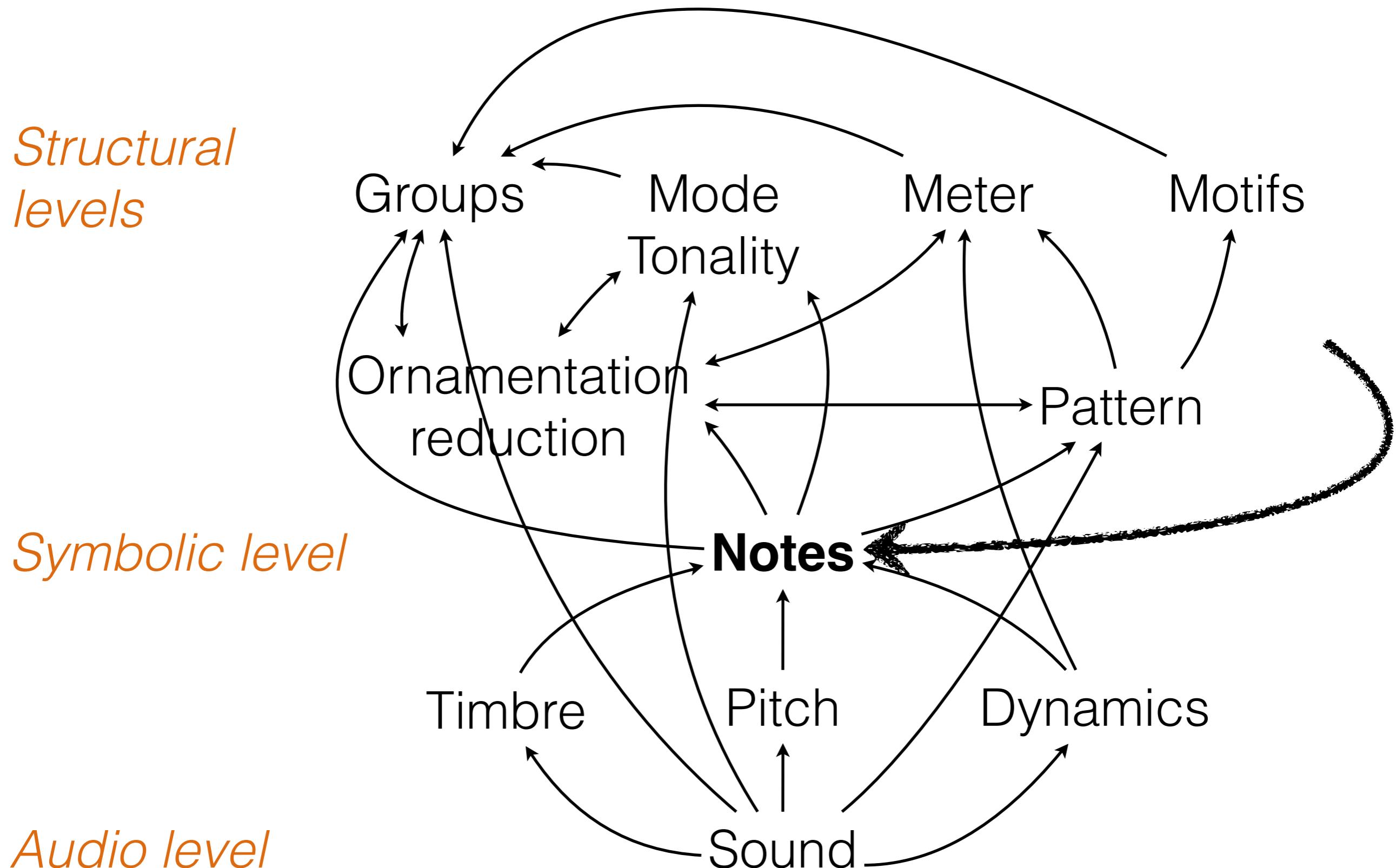
Open-source project

- *MIRtoolbox* and initial version of *MiningSuite* is mainly the work of one person. Transition to a tool controlled by a community following standard open-source protocols.
- Whole code should be clearly readable, and be subject to correction/modification/enrichment by open community, after open discussions.
- Further development of the toolbox core (architecture, new perspectives) also subject to open discussion and community-based collaboration.

Open-source project Contribution acknowledgement

- Each contributor's participation is acknowledged, in particular in the copyright notice of source files to which he or she contributed.
- Each new model based on particular research is acknowledged in the documentation. In particular, users of the model are asked to cite the related research papers in their own papers.

- High-level musicological analysis could help refine the lower-level note transcription.



Future directions (on my side)

- Melodic transformations (ornamentation)
- Modelling form and style
- Metrical, tonal/modal analysis on symbolic domain
- Polyphonic analysis on symbolic domain



Future directions (We need you!)

- Systematic test to check the validity of the results, to be run before each public version release
 - Measuring and controlling the variations of the results between versions
- Integrating other methods from the MIR literature
- Any other idea?
- We can also discuss about it during ISMIR.

Acknowledgments

- Academy of Finland research fellowship, 2009-14
 - Finnish Centre of Excellence in Interdisciplinary Music Research, University of Jyväskylä
- Learning to Create
 - Aalborg University, MusIC group

