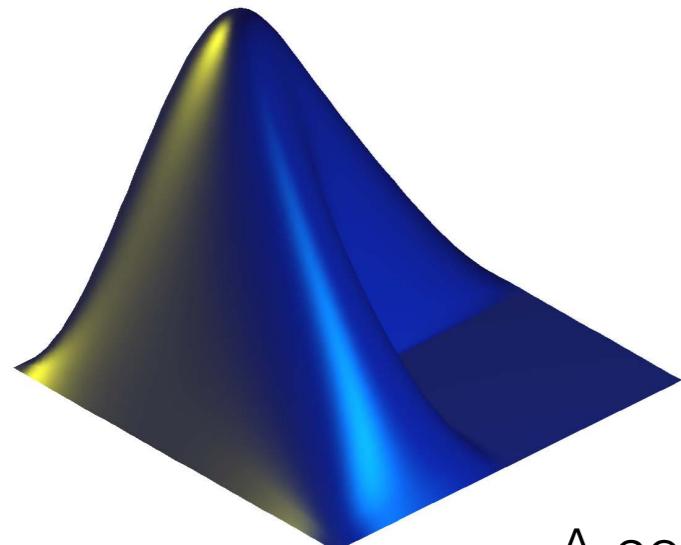


ISMIR 2014 Tutorial, 27.10.2014  
Slides version 0.6, 26.10.2014  
Current version available at:

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



# MiningSuite

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

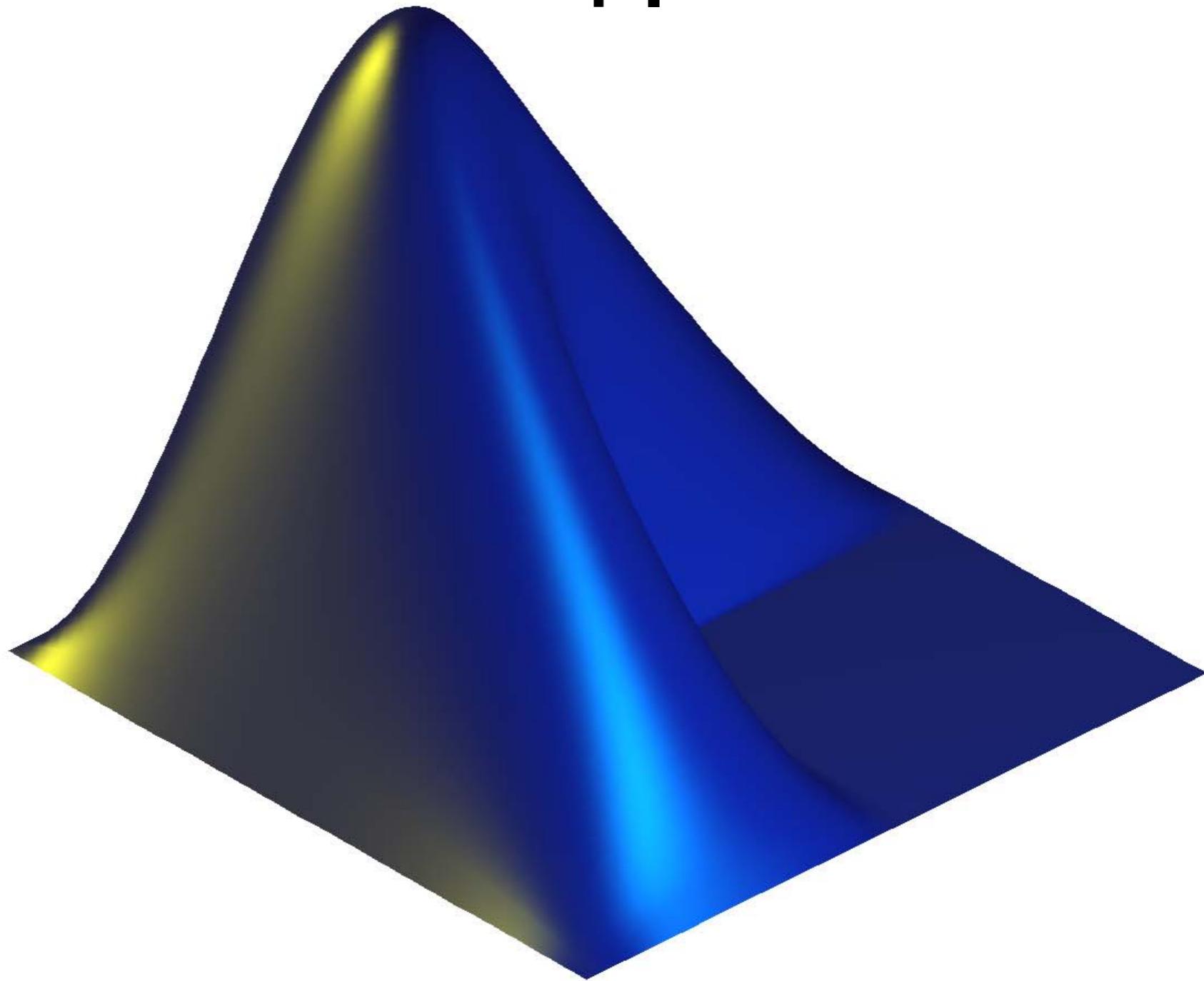
Olivier Lartillot  
Aalborg University, Denmark



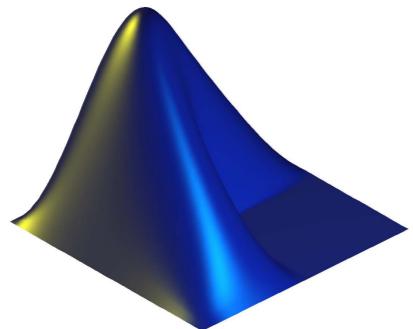
# Tutorial overview

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 audio and symbolic approaches to computational audio/music analysis
- Take benefit of the capabilities of the environment, of the user-friendly syntax and architecture
- Understand how the framework works
- Understand about the choices in the toolbox architecture, and maybe discuss about that?
- Add new codes and new features in the project

# *MIRtoolbox* history

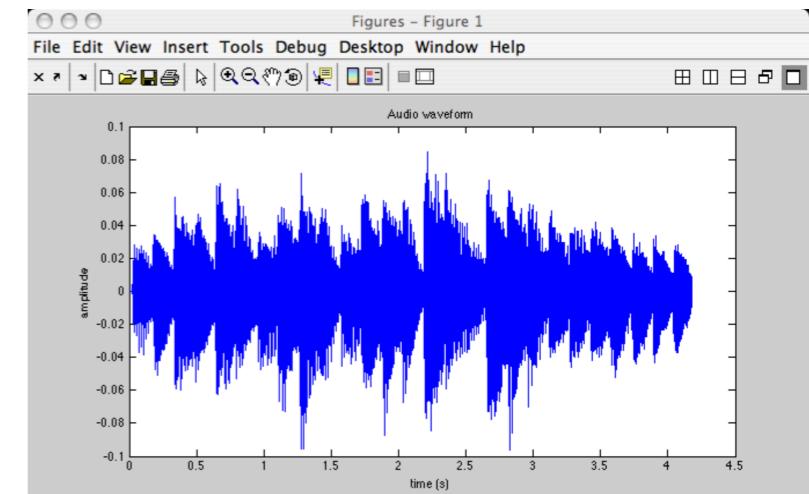
- Matlab toolboxes @ University of Jyväskylä:
  - *MIDItoolbox* (Eerola & Toiviainen, 2004)
  - *Music Therapy Toolbox* (Lartillot, Toiviainen, Erkkilä, 2005)
- European project *Tuning the Brain for Music* (NEST, FP6, 2006–08)
  - **Large range of audio/musical features** extracted from large databases, linked to emotional ratings (Eerola et al., 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

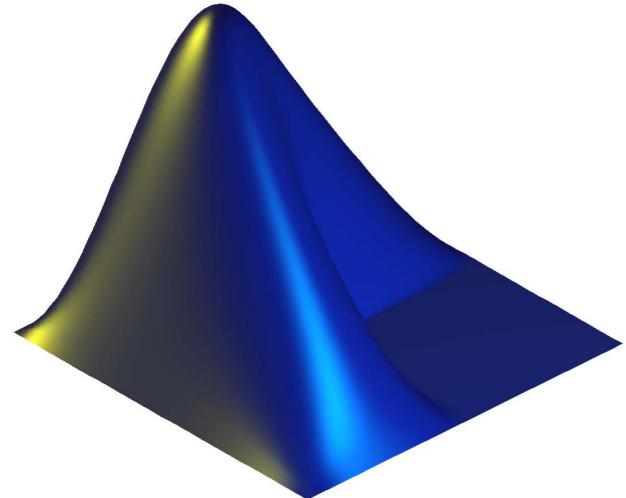


# MiningSuite syntax

- ***sig.audio('ragtime.wav')*** outputs a figure
- ***sig.audio('ragtime.wav');* postpone any computation**
- ***mus[tempo('ragtime.wav')]***
- *mus[tempo('Folder')]* operates on all files in the Current Directory
- *a = sig.audio('ragtime.wav', 'Extract', 0, 60, 's')*
- *a = sig.audio(a, 'Center');*
- *t = mus[tempo(a)]*
- *d = t.getdata* returns the result in *Matlab* array
- ***t.show*** 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**: audio, auditory modelling
- **MUSMINR**: music analysis
- **PATMINR**: pattern mining
- **SEQMINR**: sequence processing
- **VOCMINR**: voice analysis?

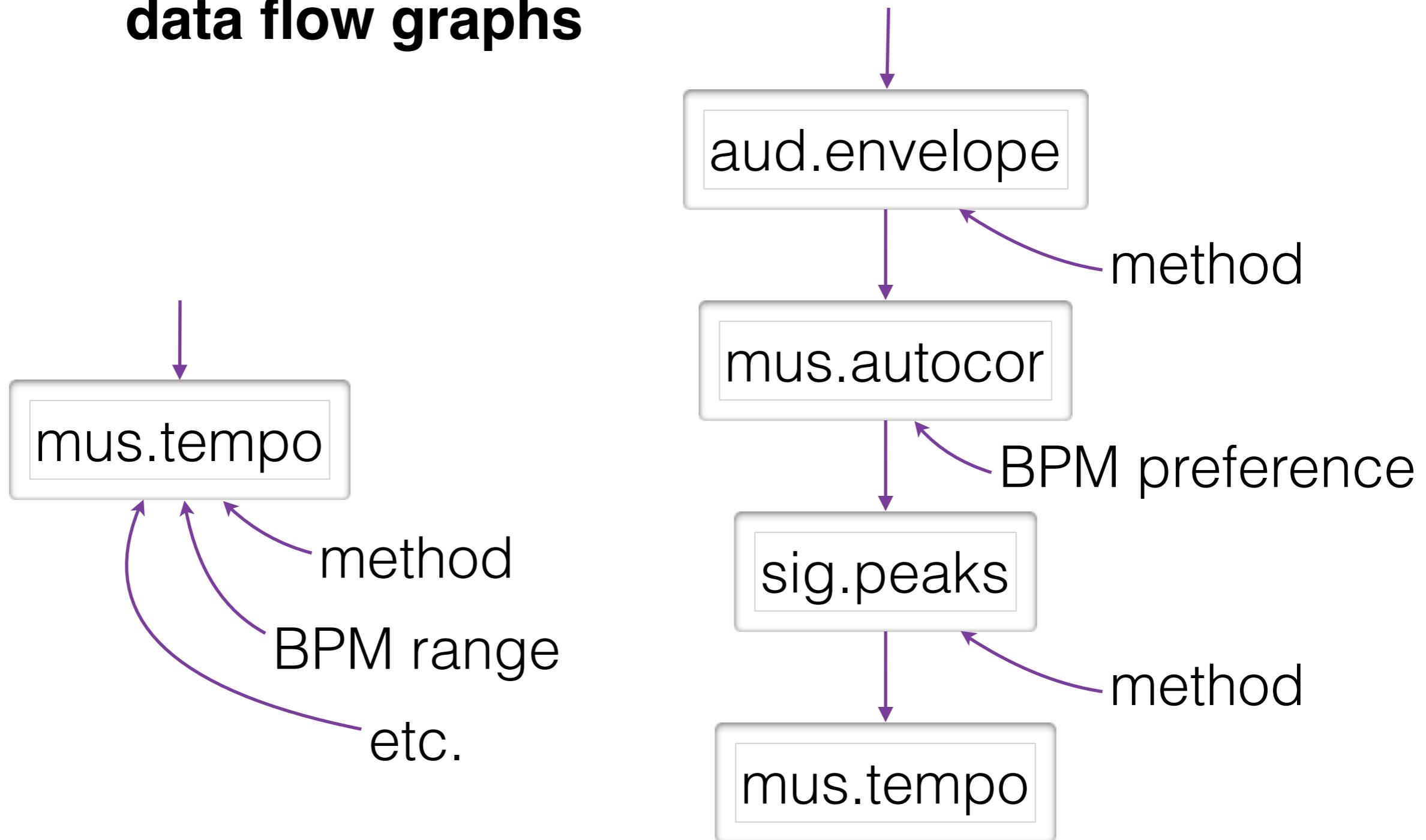
# Packages

- 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

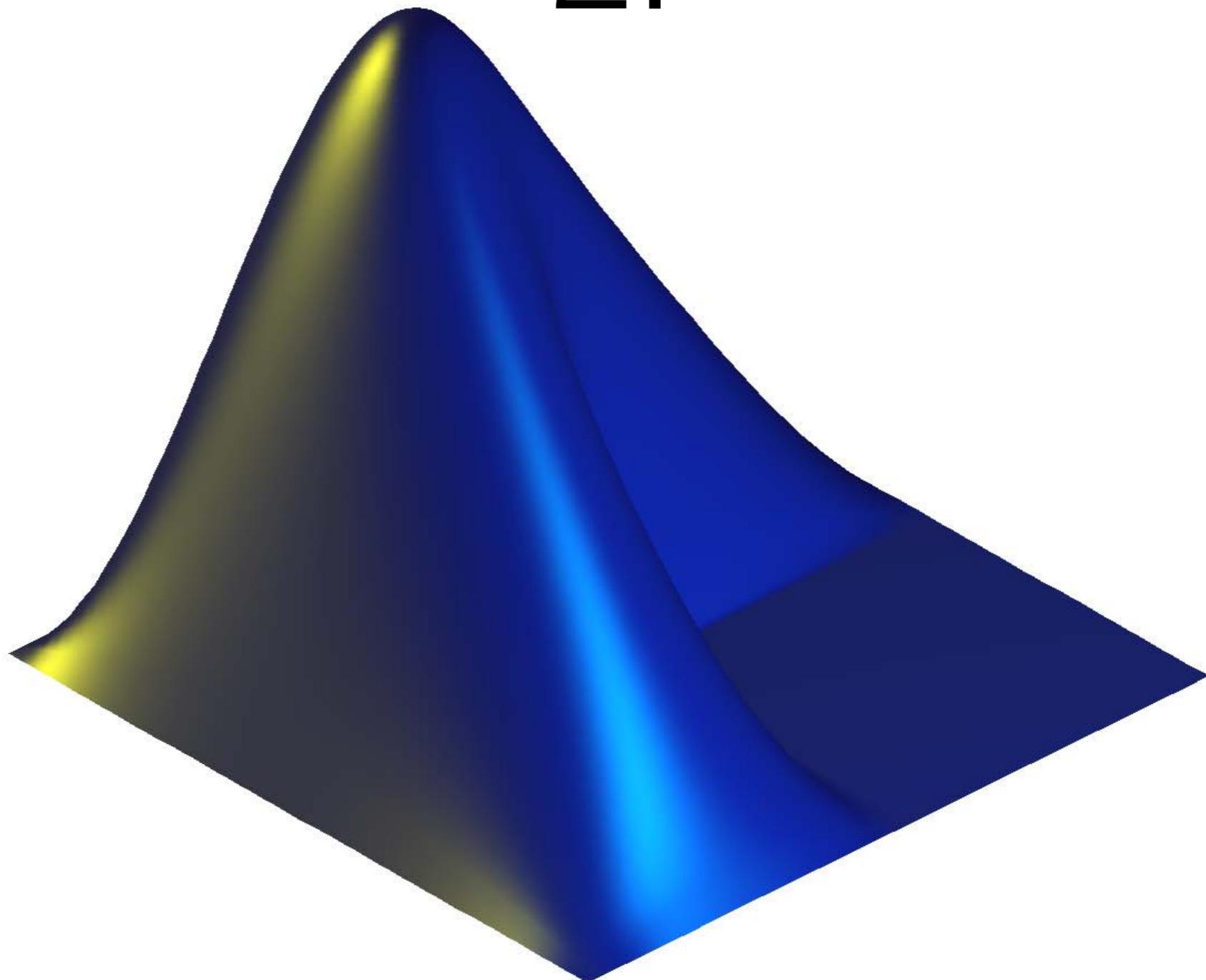
- *MathWorks' Matlab* 7.6 (r2008a) or newer versions.
  - But *Matlab* 8.2 (r2013b) or newer is strongly recommended, because *MiningSuite* easily crashes on previous versions\*.
  - *MiningSuite* based on new object oriented paradigm introduced in *Matlab* 7.6
- *MathWorks' Signal Processing Toolbox*
- Included in the distribution: code from *Auditory Toolbox* (Slaney), *Music Analysis Toolbox* (Pampalk), *MIDI Toolbox* (Eerola et al.)

\* 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. **Not everything implemented. Results cannot be trusted!**
- Version 0.9 with more complete 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.zerocross	sig.spectrum	sig.rolloff	sig.stat...
	sig.cepstrum		
sig.filterbank		sig.simatrix	sig.cluster
sig.envelope			

# AUDMINR

## audio, auditory modeling

aud.spectrum

aud.filterbank

aud.envelope

aud.brightness

aud.attacktime

aud.roughness

aud.attackslope

aud.mfcc

aud.pitch

aud.fluctuation

aud.novelty

aud.score

aud.segment

aud.eventdensity

# MUSMINR

## music theory

mus.spectrum

mus.pitch

mus.temp

mus.chromagram

mus.pulseclar

mus.keystrength

mus.metre

mus.key

mus.keysom

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.

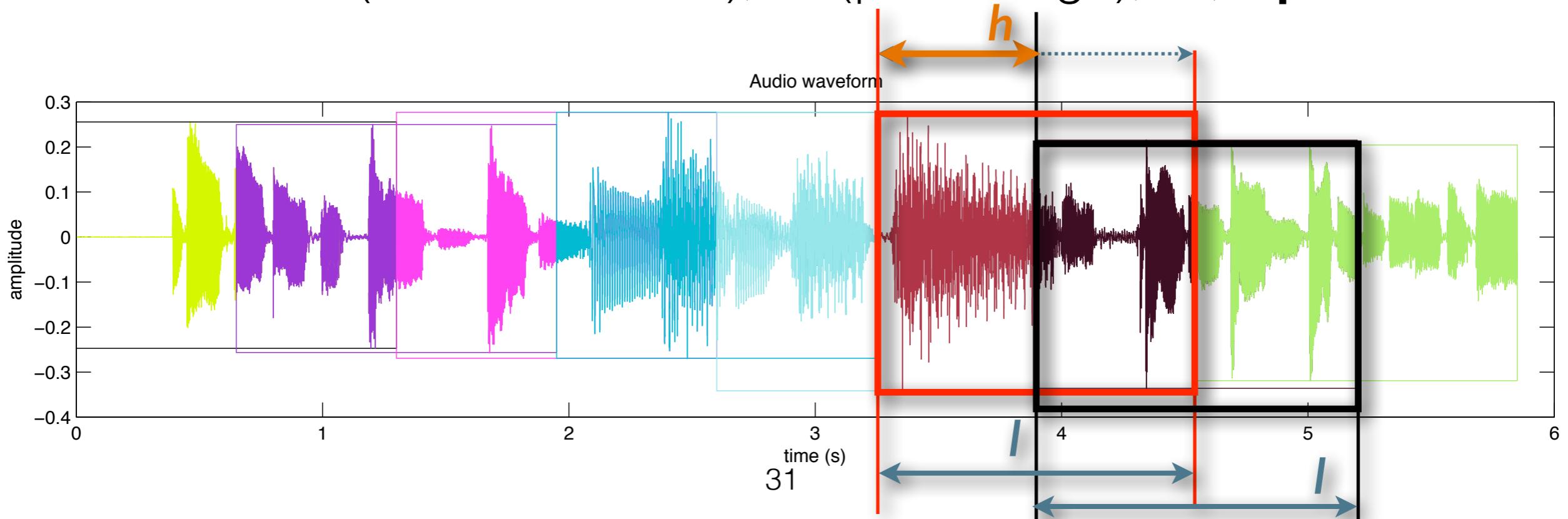
# *.play, .save* sonification

- `a = sig.input(..., 'Trim')`
- `a.play`
- `a.save('newfile.mp3')`

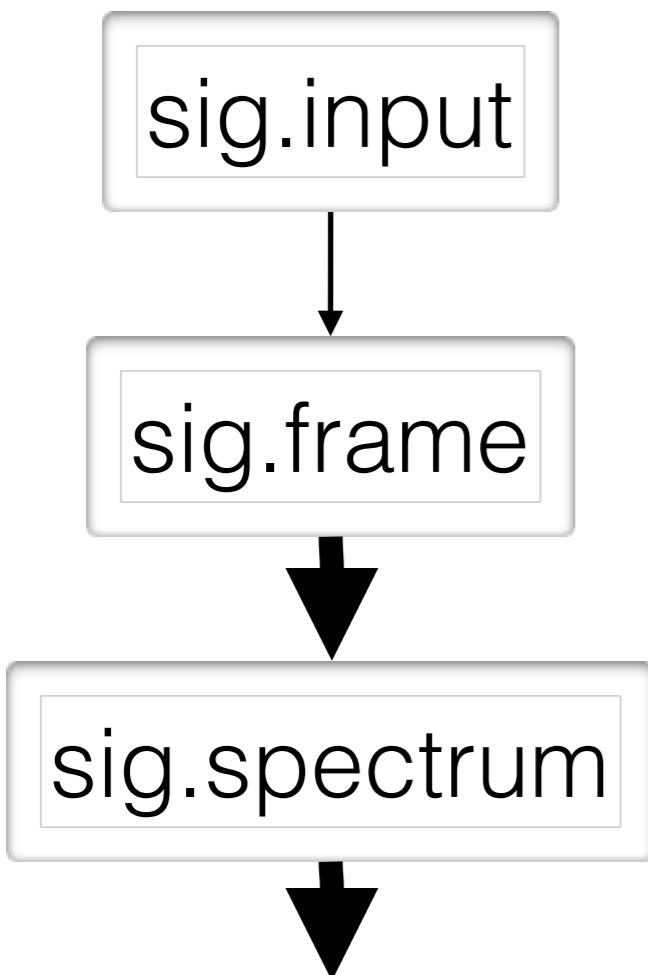
# *sig.frame*

## frame decomposition

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



# *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

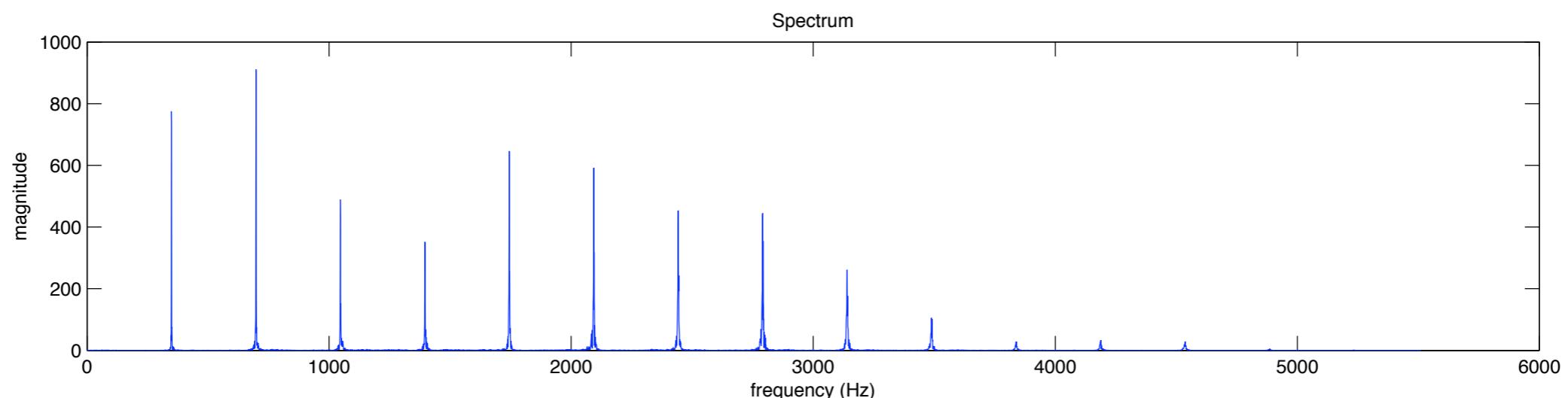
- *sig.input(..., ‘FrameSize’, ..., ‘FrameHop’, ...)*
- *sig.spectrum(..., ‘FrameSize’, ..., ‘FrameHop’, ...)*
- *sig.spectrum(..., ‘Frame’)* (default frame configuration)
- ‘*Frame*’ option available to most operators, each with its own default frame configuration
- Each operator can perform the frame 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:  $s.\text{phase}$

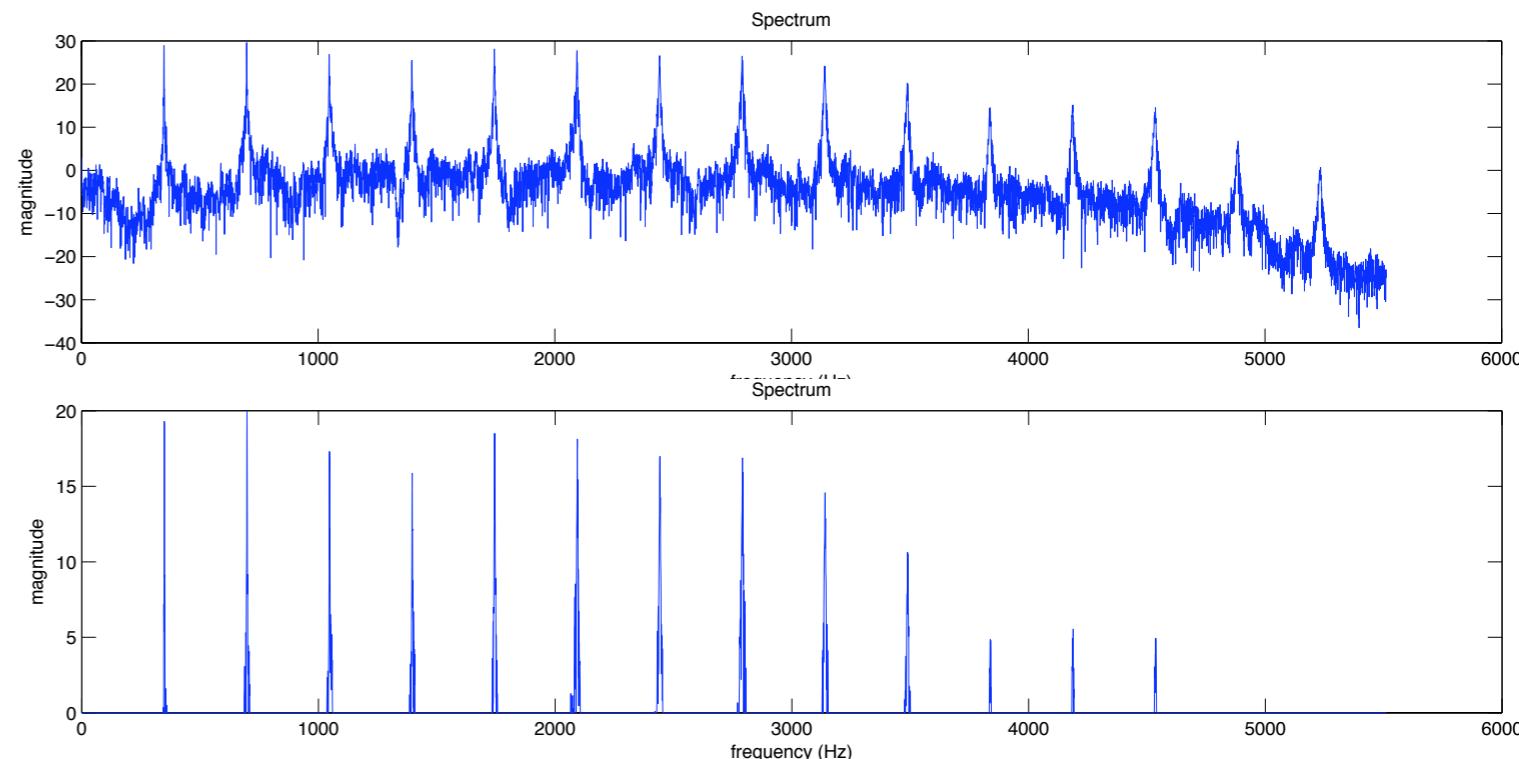
Fast Fourier Transform using Matlab's *fft* function.

# *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

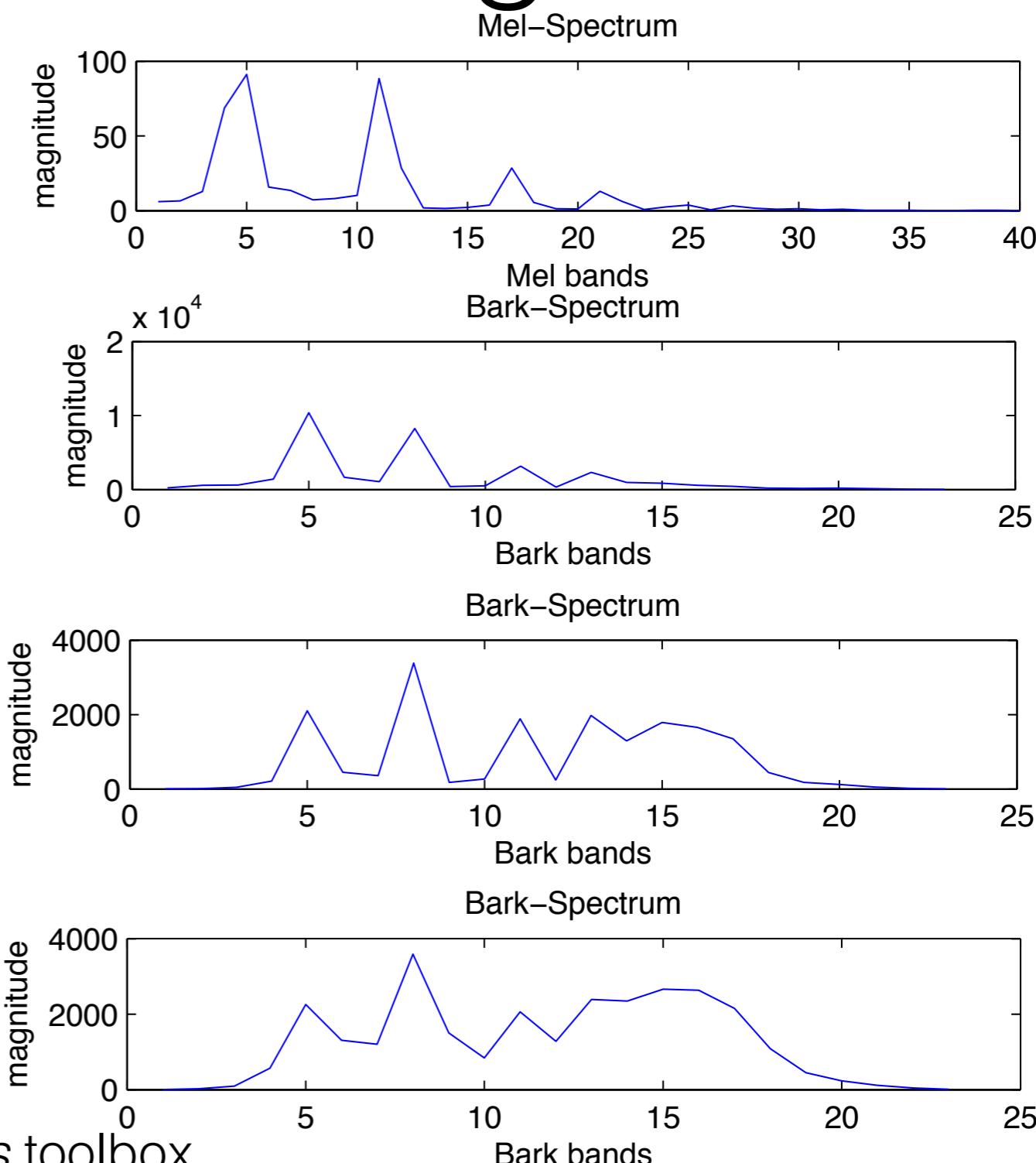
- *mirspectr um(..., 'Normal')* normalizes w.r.t. energy.
- *mirspectr um(..., 'Power')* squares the energy.
- *mirspectr um(..., 'dB')*
  - in dB scale
- *mirspectr um(..., 'dB', th)*
  - th highest dB
- *mirspectr um(..., 'Smooth', o)*
- *mirspectr um(..., 'Gauss', o)*



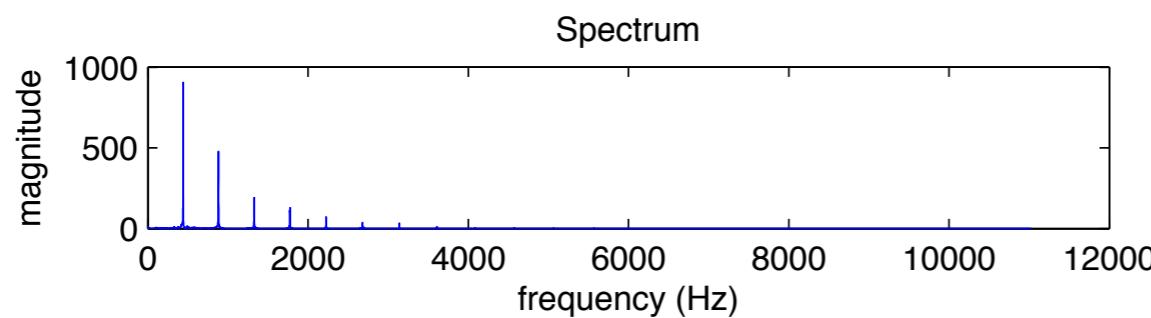
# *aud.spectrum*

## auditory modeling

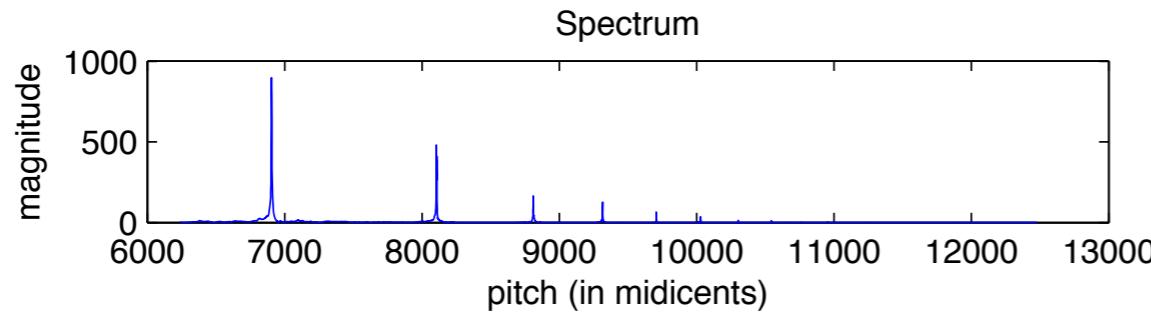
- *aud.spectrum(..., 'Mel')*
  - Mel-band decomposition
- *aud.spectrum(..., 'Bark')*
  - Bark-band decomposition
- *aud.spectrum(..., 'Terhardt')*:
  - Outer-ear modeling
- *aud.spectrum(..., 'Mask')*:
  - Masking effects along bands



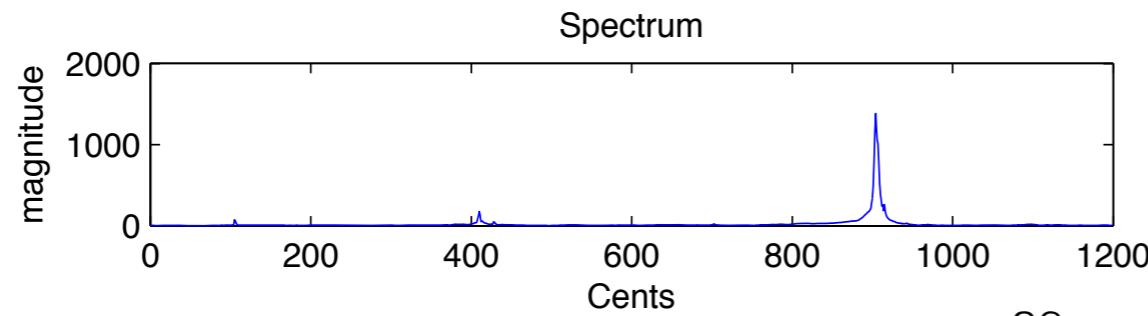
# ***mus.spectrum*** pitch-based distribution



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



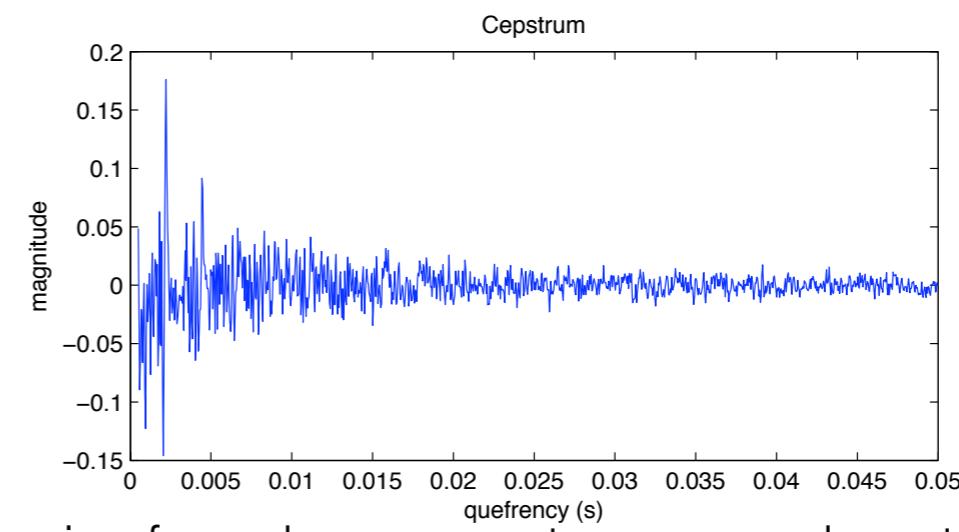
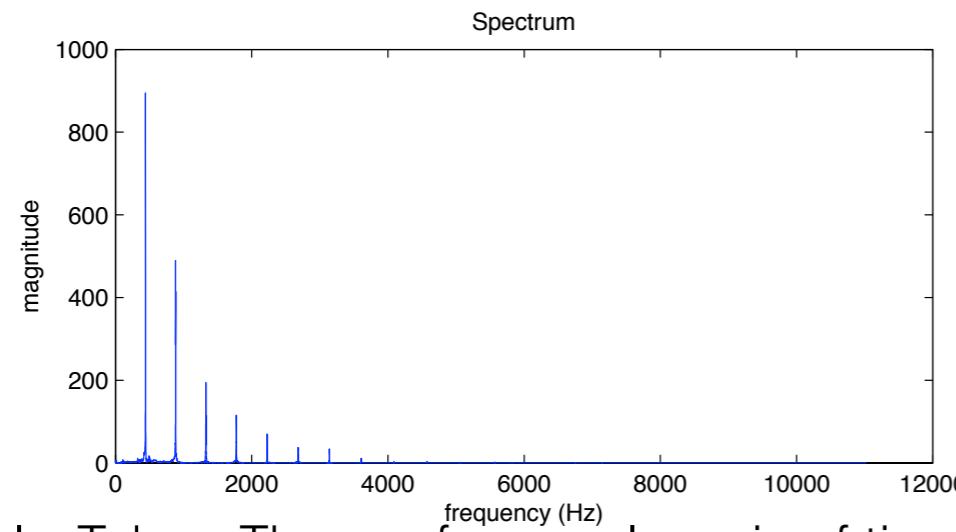
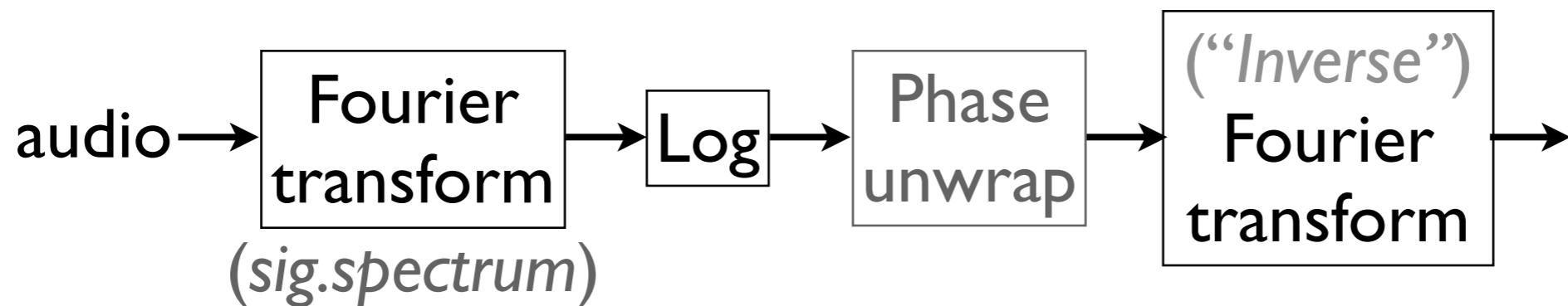
- *aud.spectrum(..., 'Collapsed')*: into one octave, divided into 1200 cents



# *sig.cepstrum*

## cepstral analysis

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

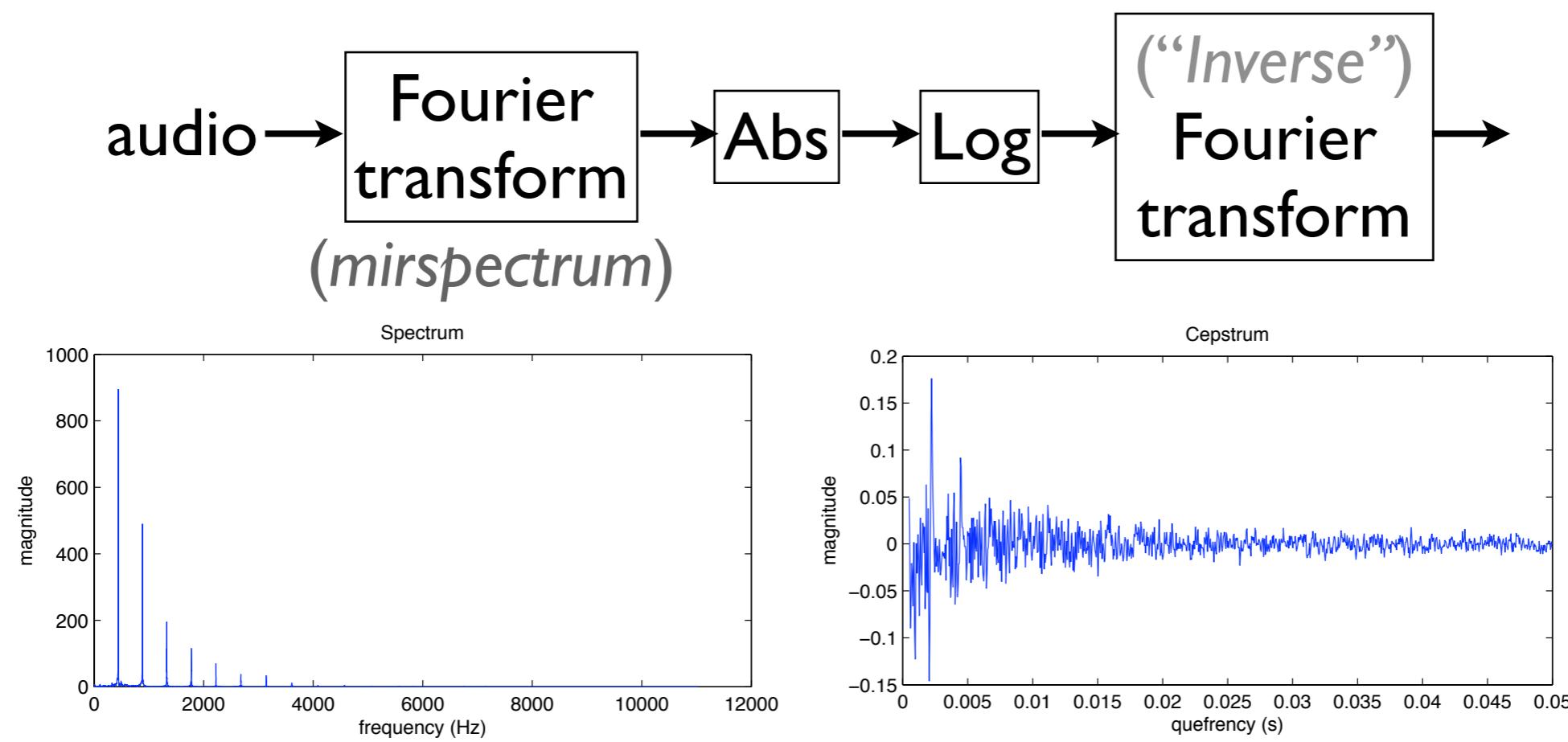


Bogert, Healy, Tukey. The quefrency alalysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. Symposium on Time Series Analysis, Chapter 15, 209-243, Wiley, 1963.

# *sig.cepstrum*

## cepstral analysis

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



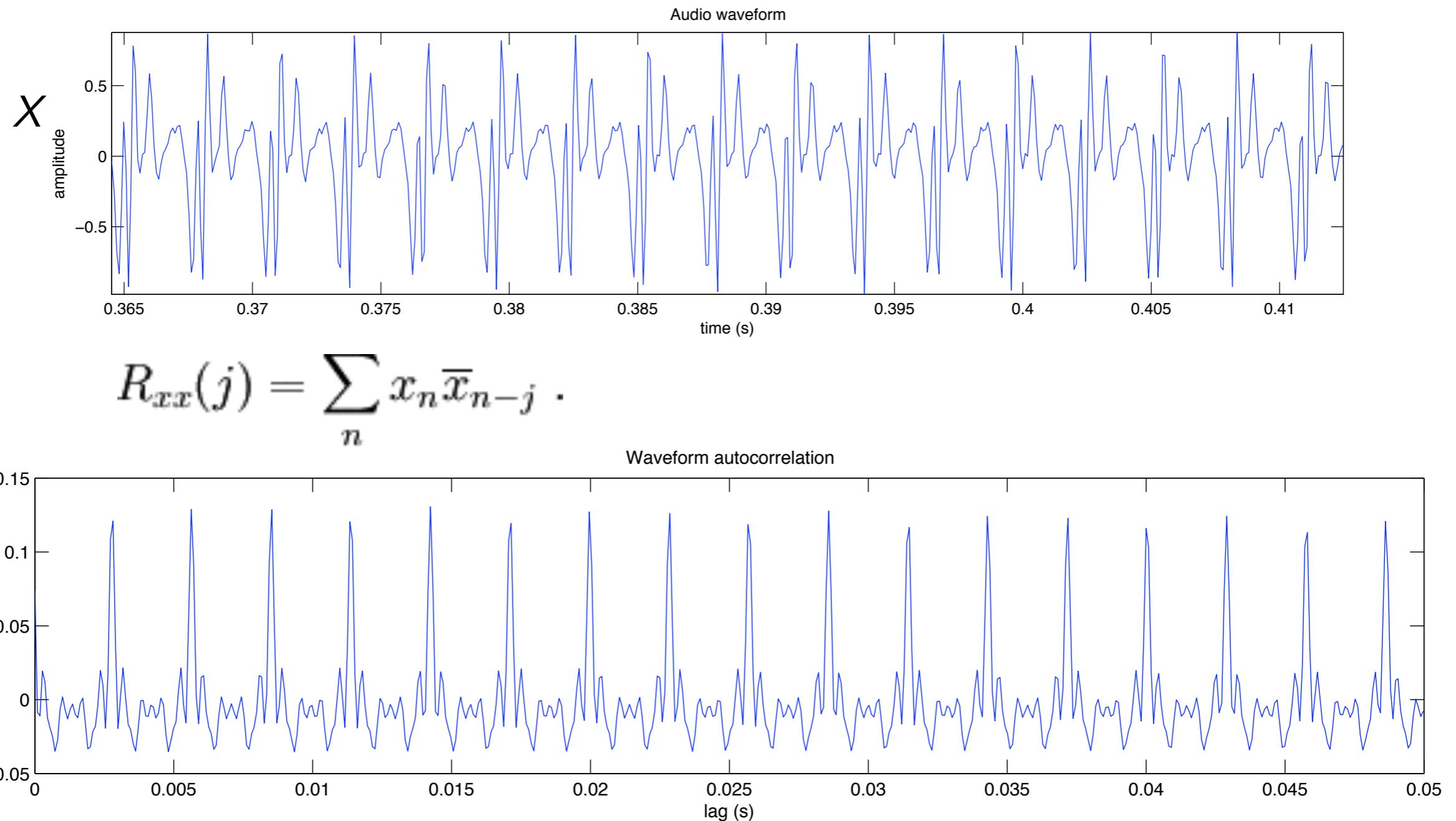
Bogert, Healy, Tukey. The quefrency alalysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. Symposium on Time Series Analysis, Chapter 15, 209-243, Wiley, 1963.

# *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

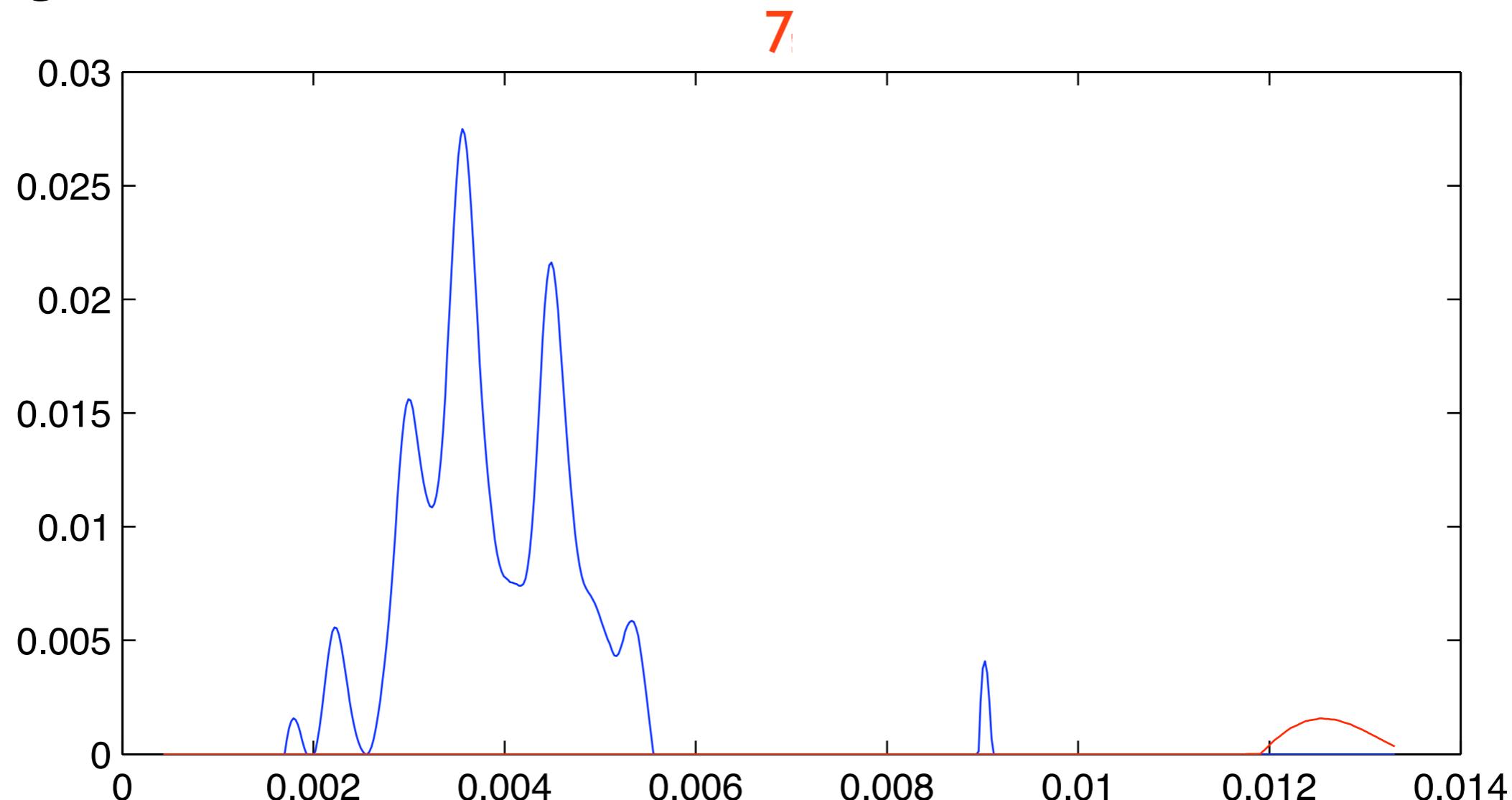
\*Boersma. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound, IFA Proceedings 17: 97-110, 1993.

# *sig.autocor* generalized autocorrelation

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

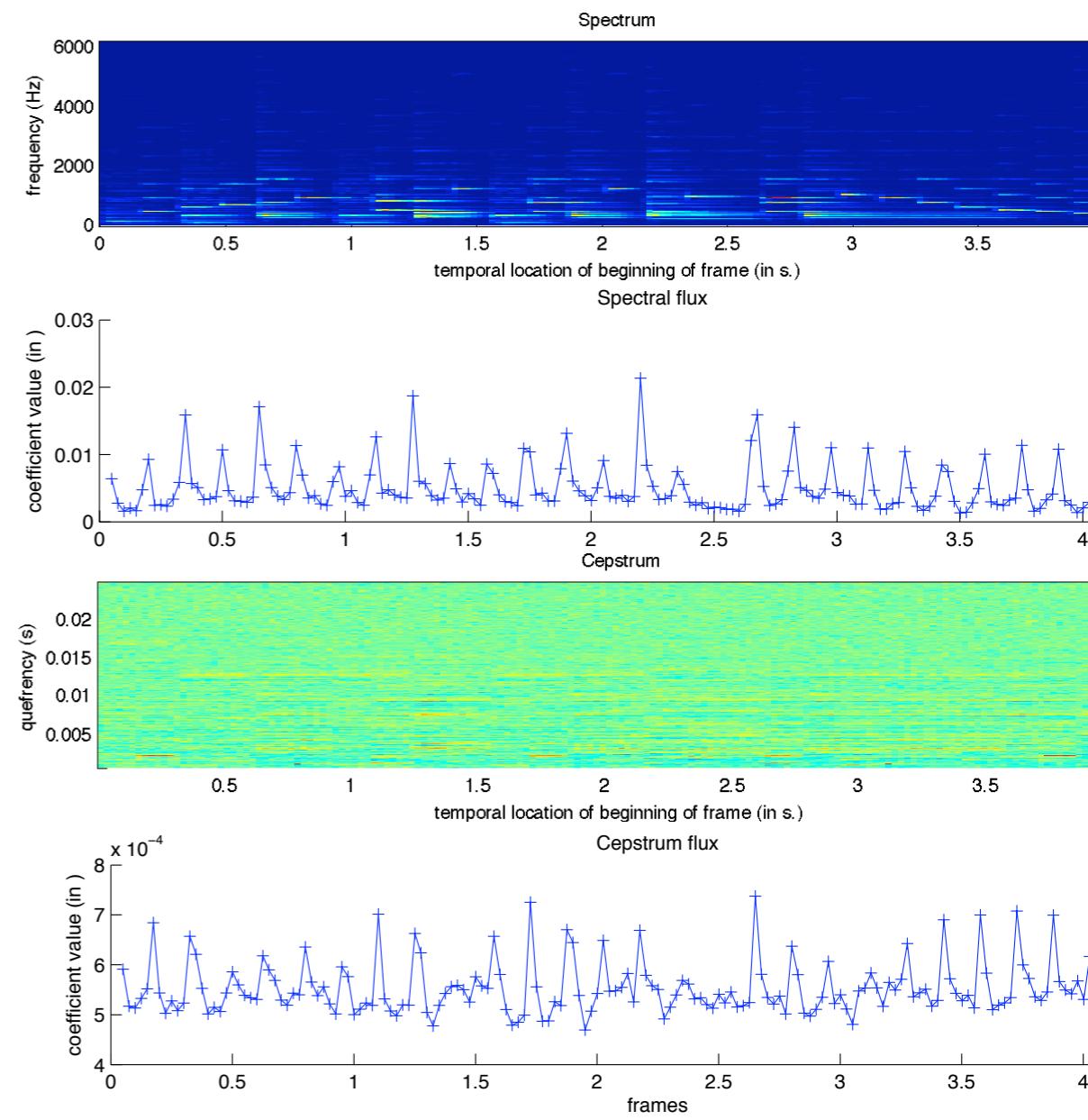
# *sig.autocor* generalized autocorrelation

- *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**’)*

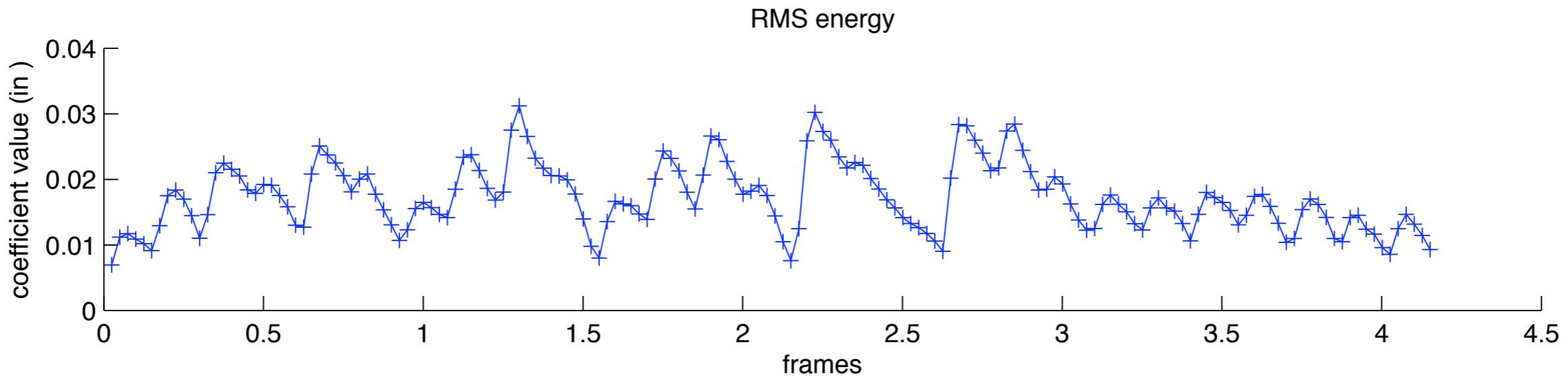
*Audio level*

**Dynamics**  
Sound 

# *sig.rms* root-mean square

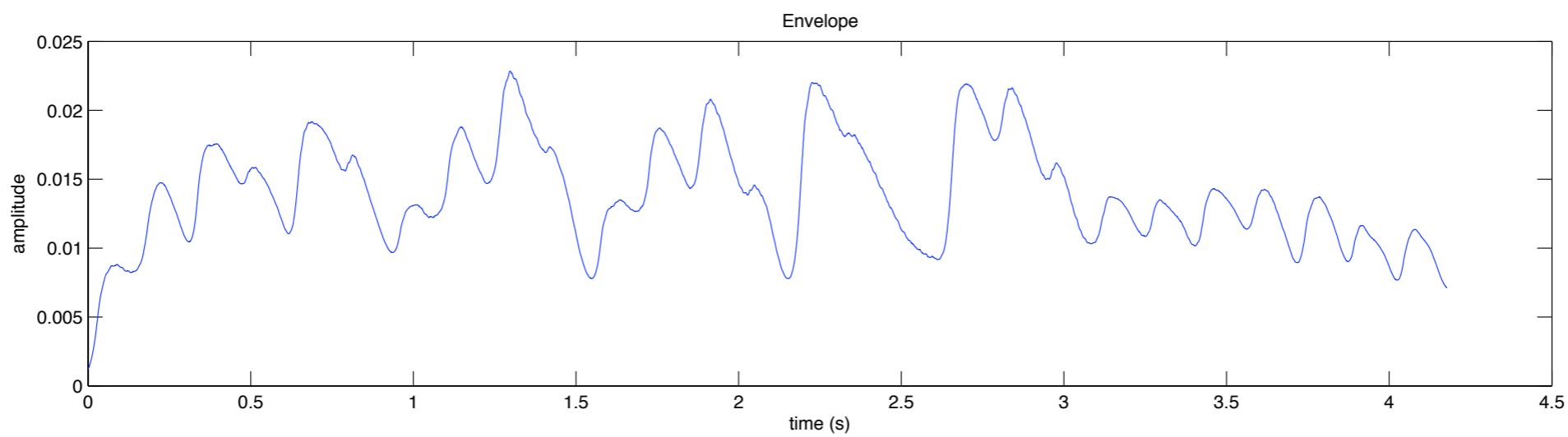
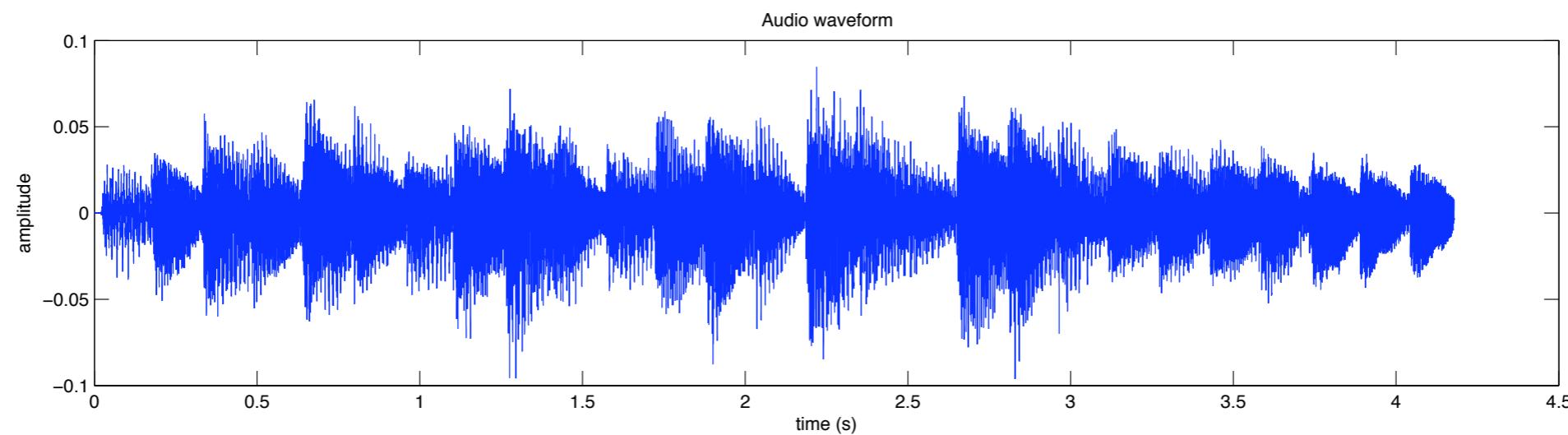
$$x_{\text{rms}} = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \dots + x_n^2}{n}}$$

- *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



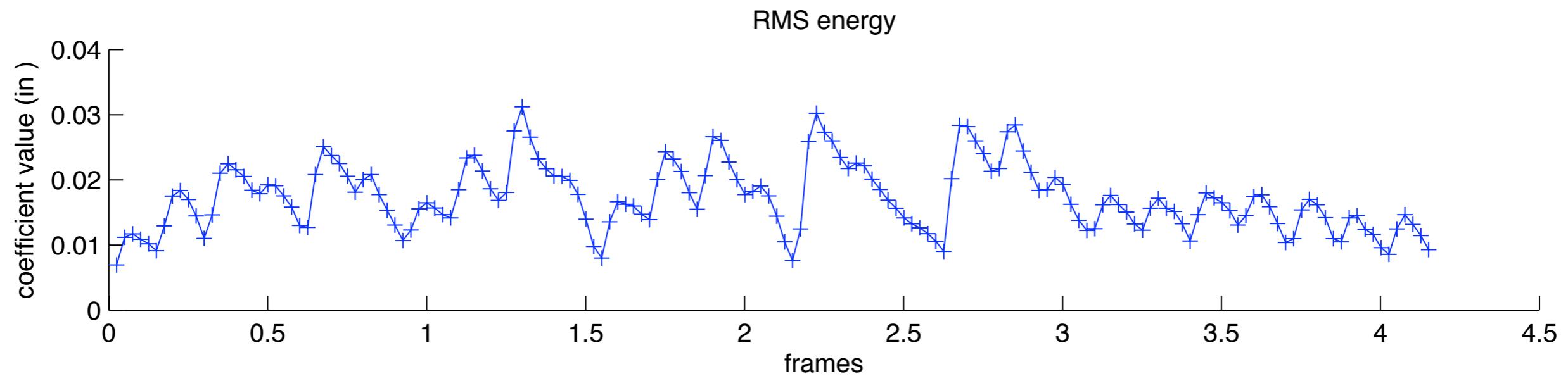
# *sig.envelope* envelope extraction

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

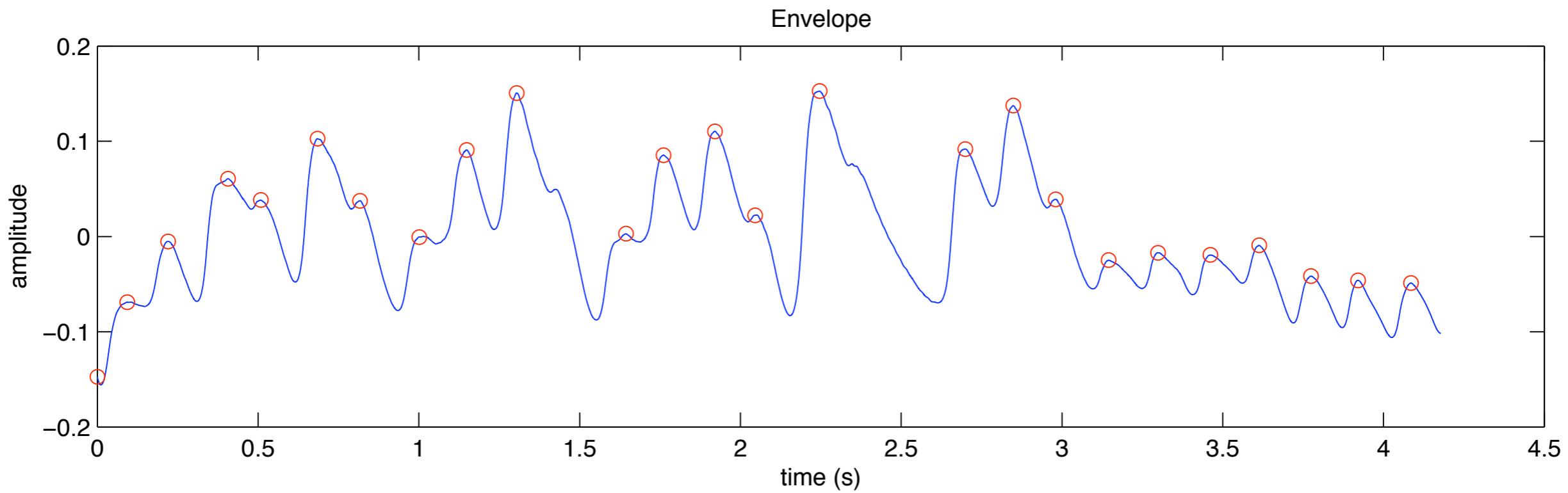


*e.play  
e.save*

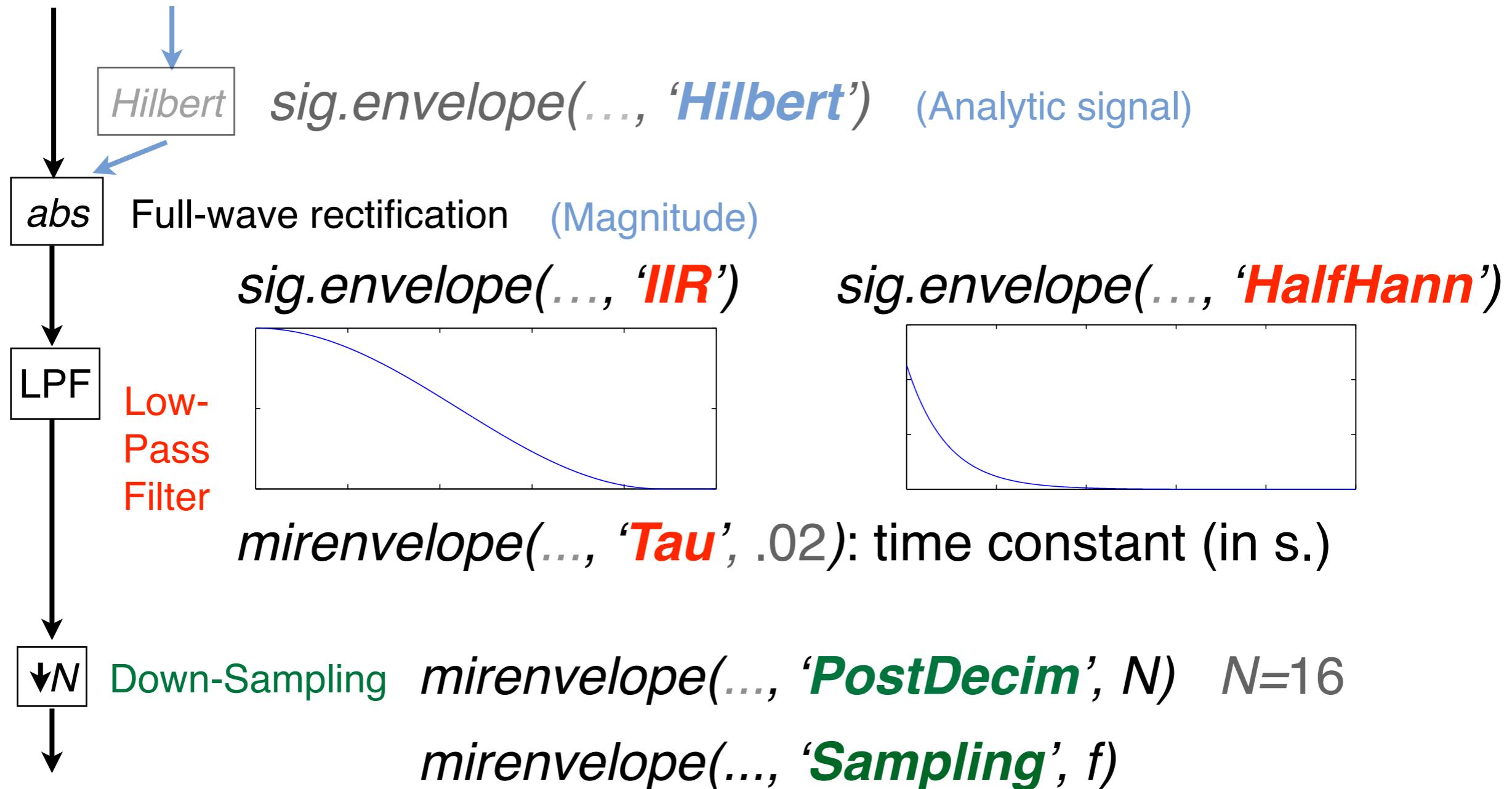
# *sig.rms*



# *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

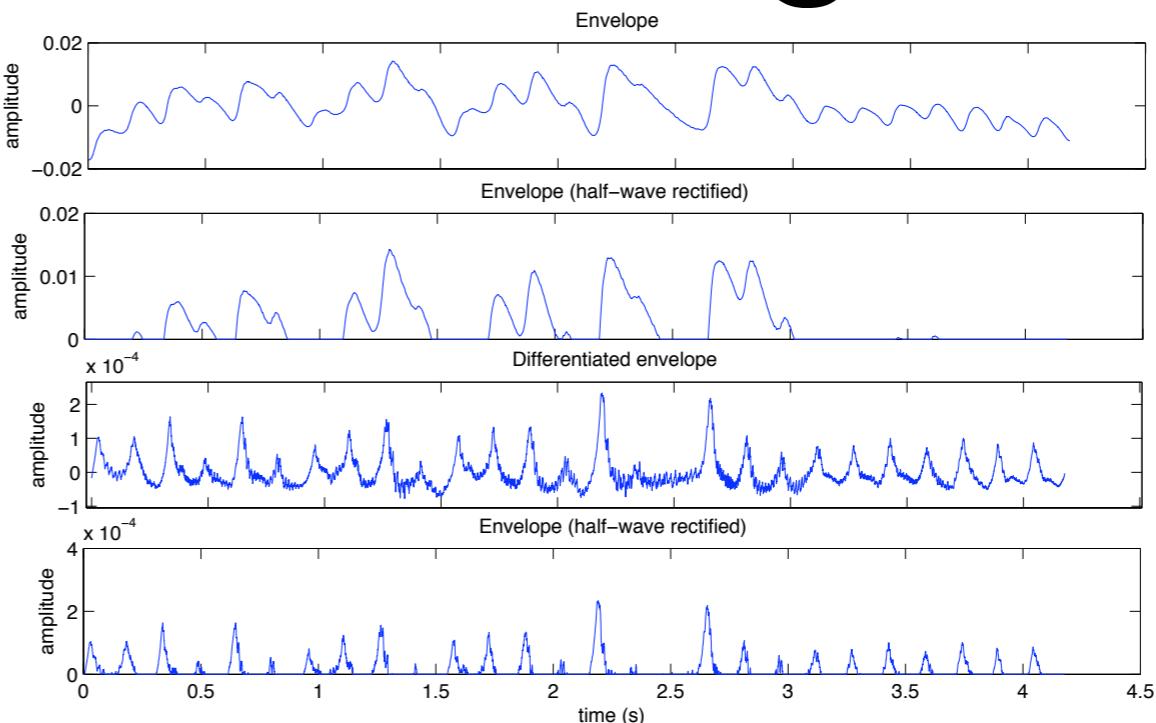
- *sig.envelope(..., 'Center')*

**'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

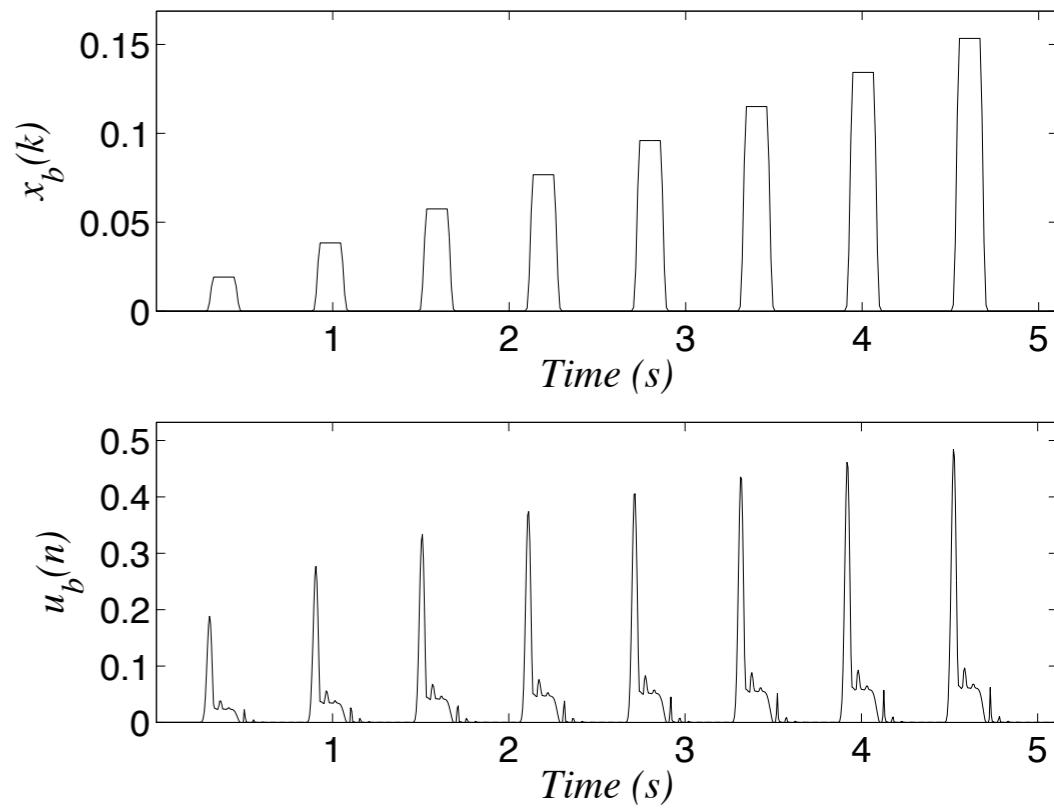
- *aud.envelope(..., 'Log', mu)*,  
mu-law compression, mu = 100

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

- *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)*

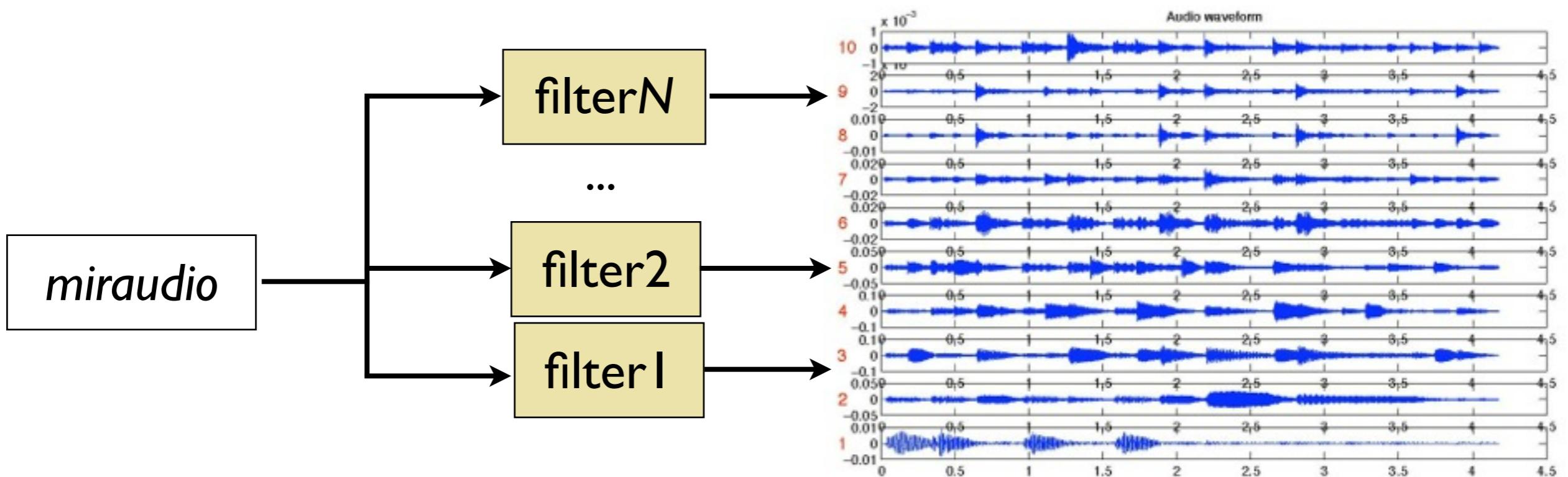


Klapuri, A., A. Eronen and J. Astola. (2006). “Analysis of the meter of acoustic musical signals”, IEEE Transactions on Audio, Speech and Language Processing, 14-1, 342– 355.

# *sig.filterbank*

## filterbank decomposition

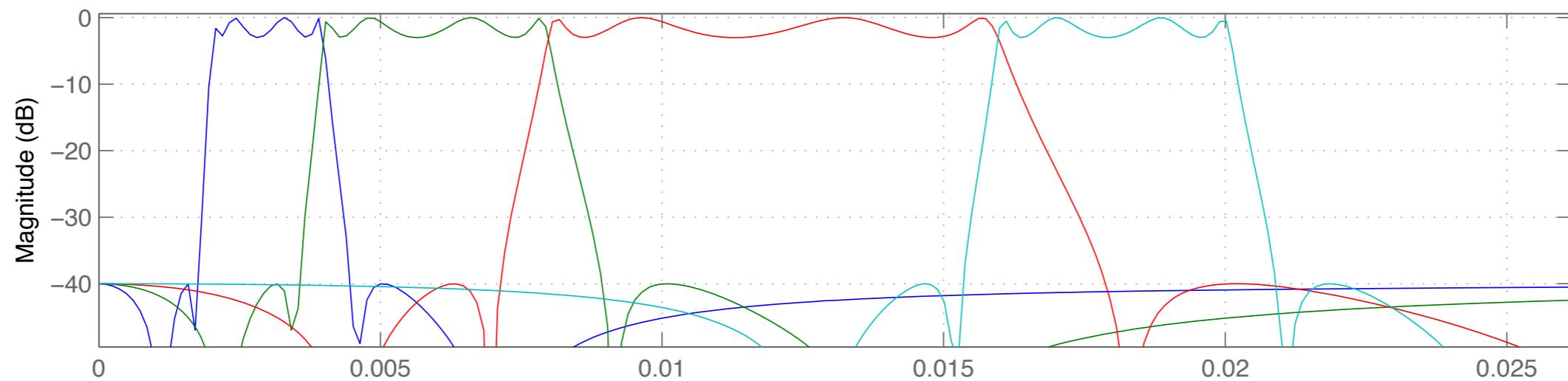
- `f = sig.filterbank(..., 'CutOff', [500, 1000])`
- `f.play`
- `f.save`



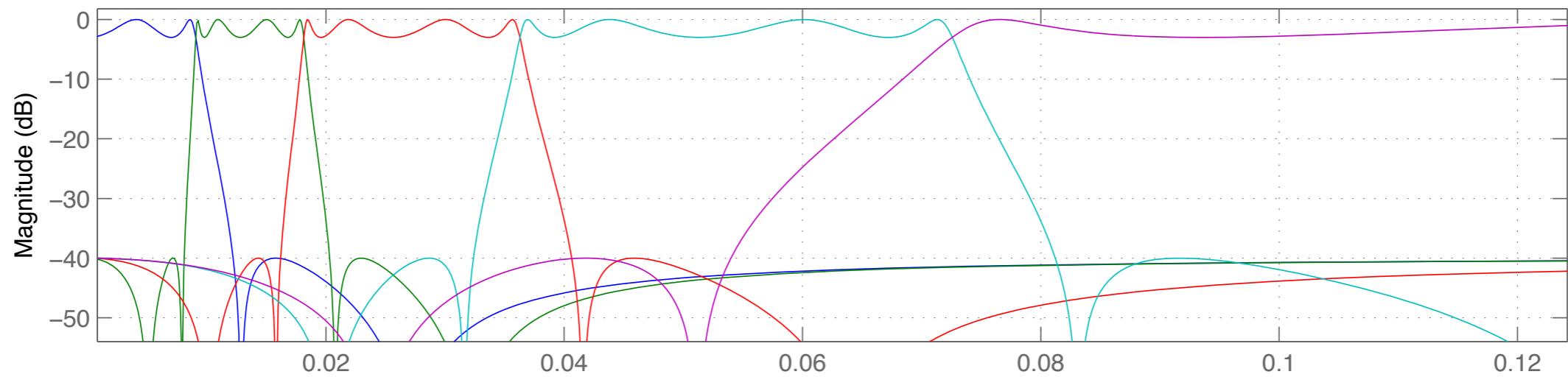
# *sig.filterbank*

## filterbank decomposition

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



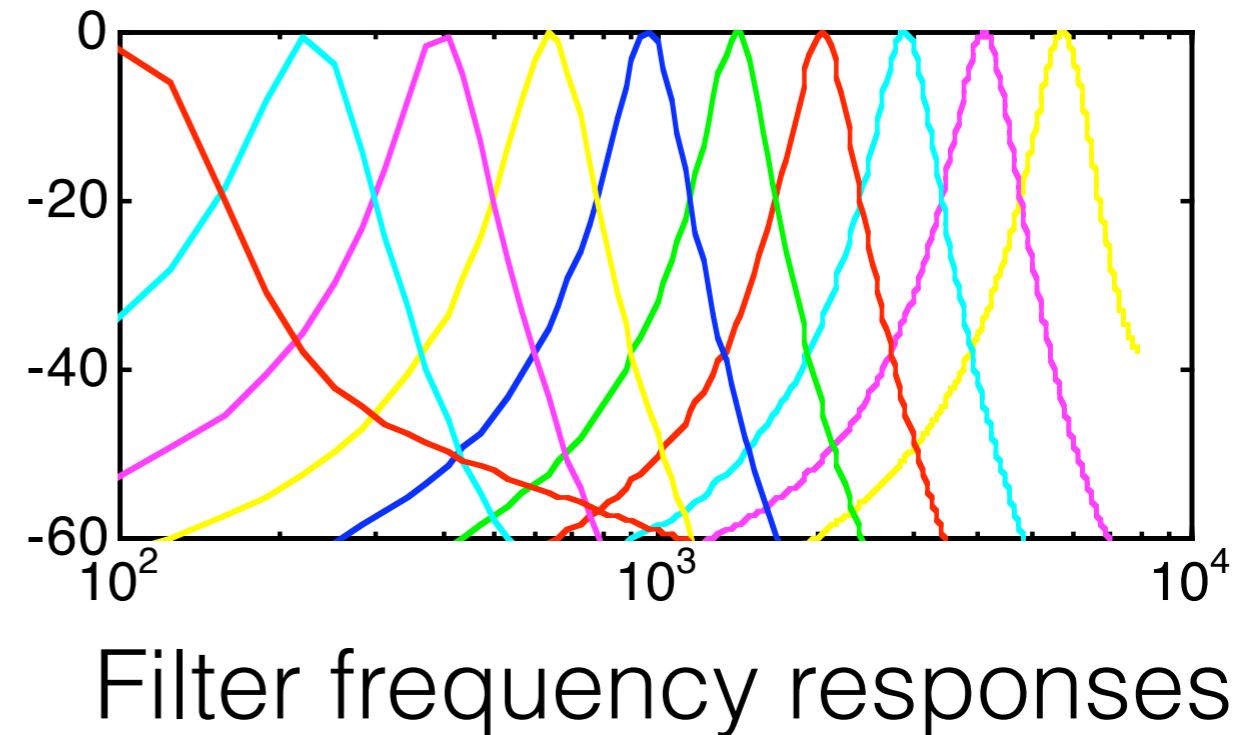
`sig.filterbank(..., 'CutOff', [-Inf 200 400 800 1600 Inf])`



# ***aud.filterbank***

## auditory modeling

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



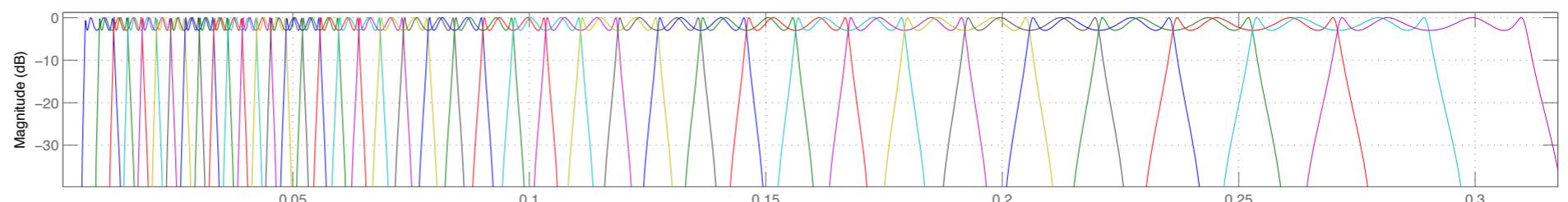
based on *Auditory toolbox*

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

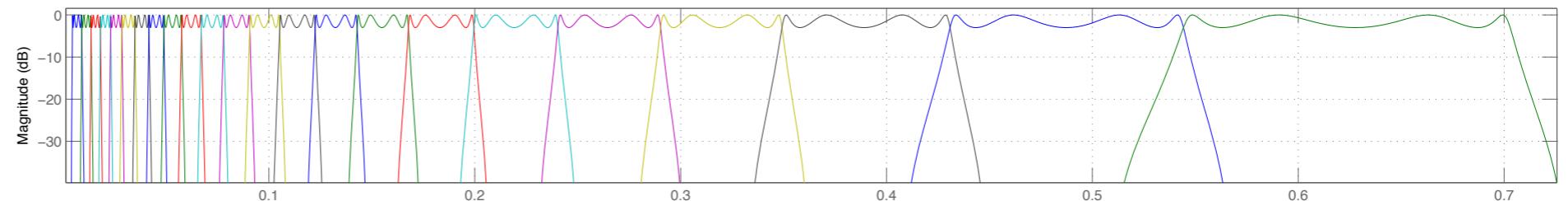
# ***aud.filterbank***

## auditory modeling

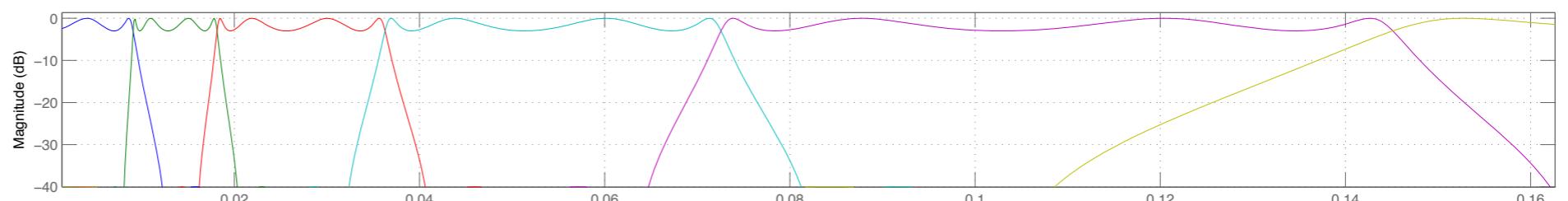
- *aud.filterbank(..., 'Mel')*



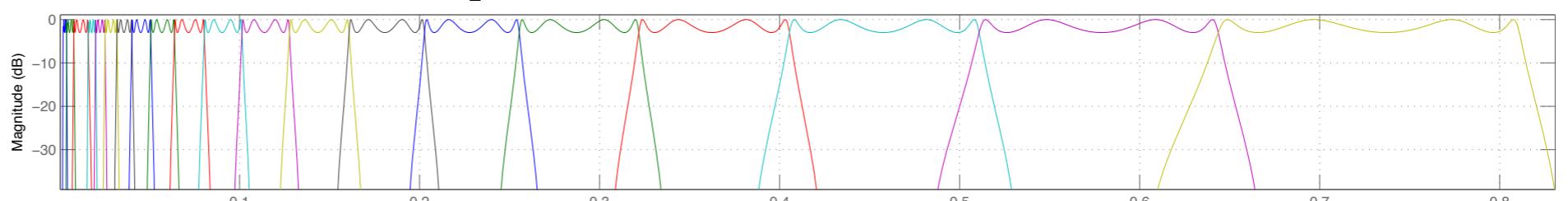
- *aud.filterbank(..., 'Bark')*



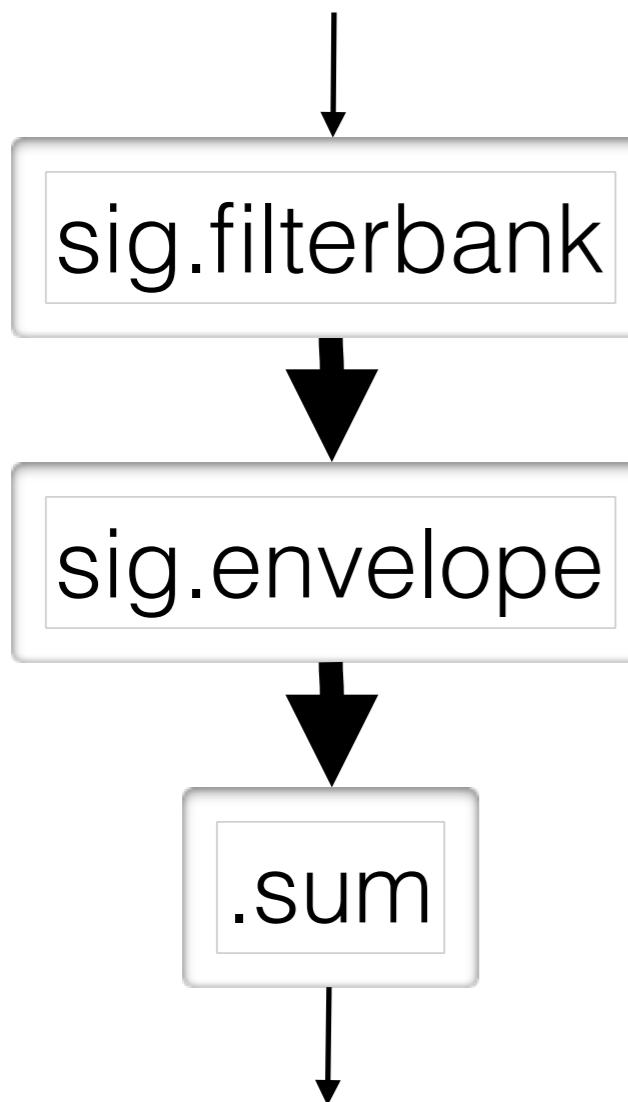
- *aud.filterbank(..., 'Scheirer')*



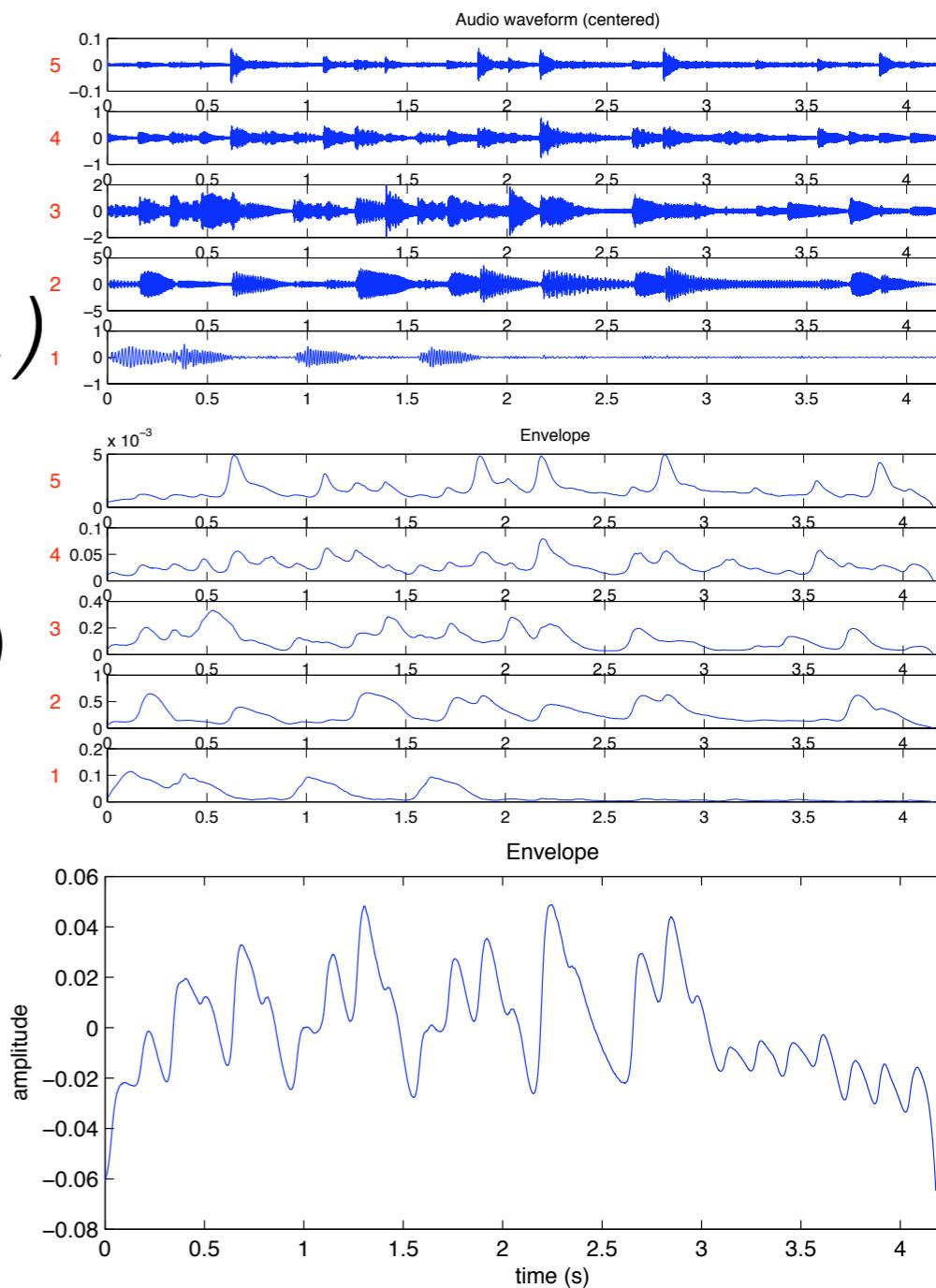
- *aud.filterbank(..., 'Klapuri')*



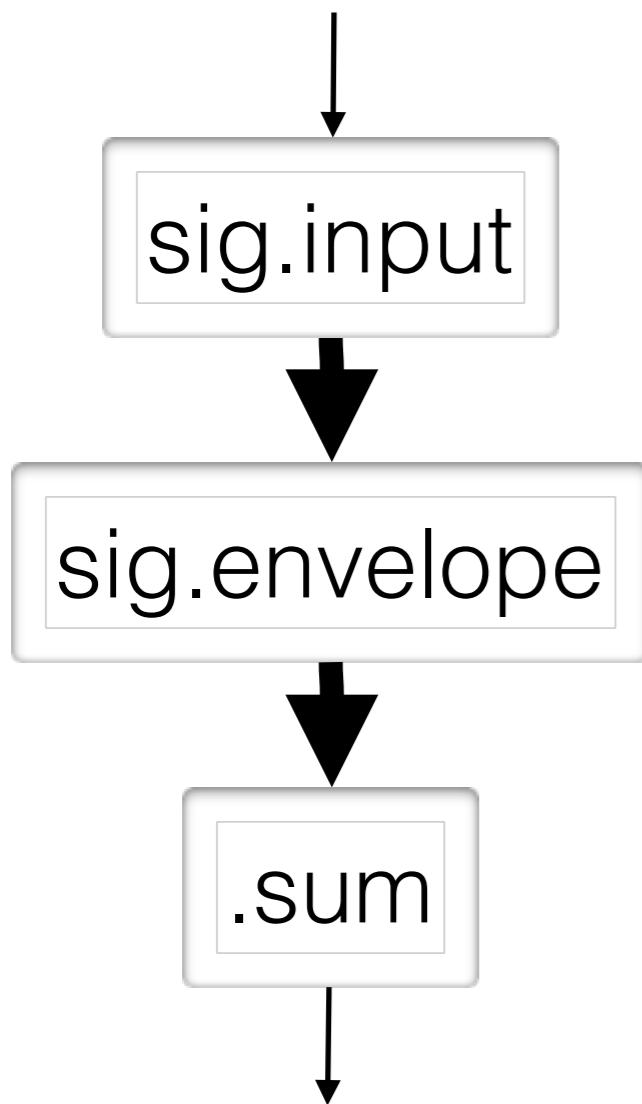
# .sum across-channel summation



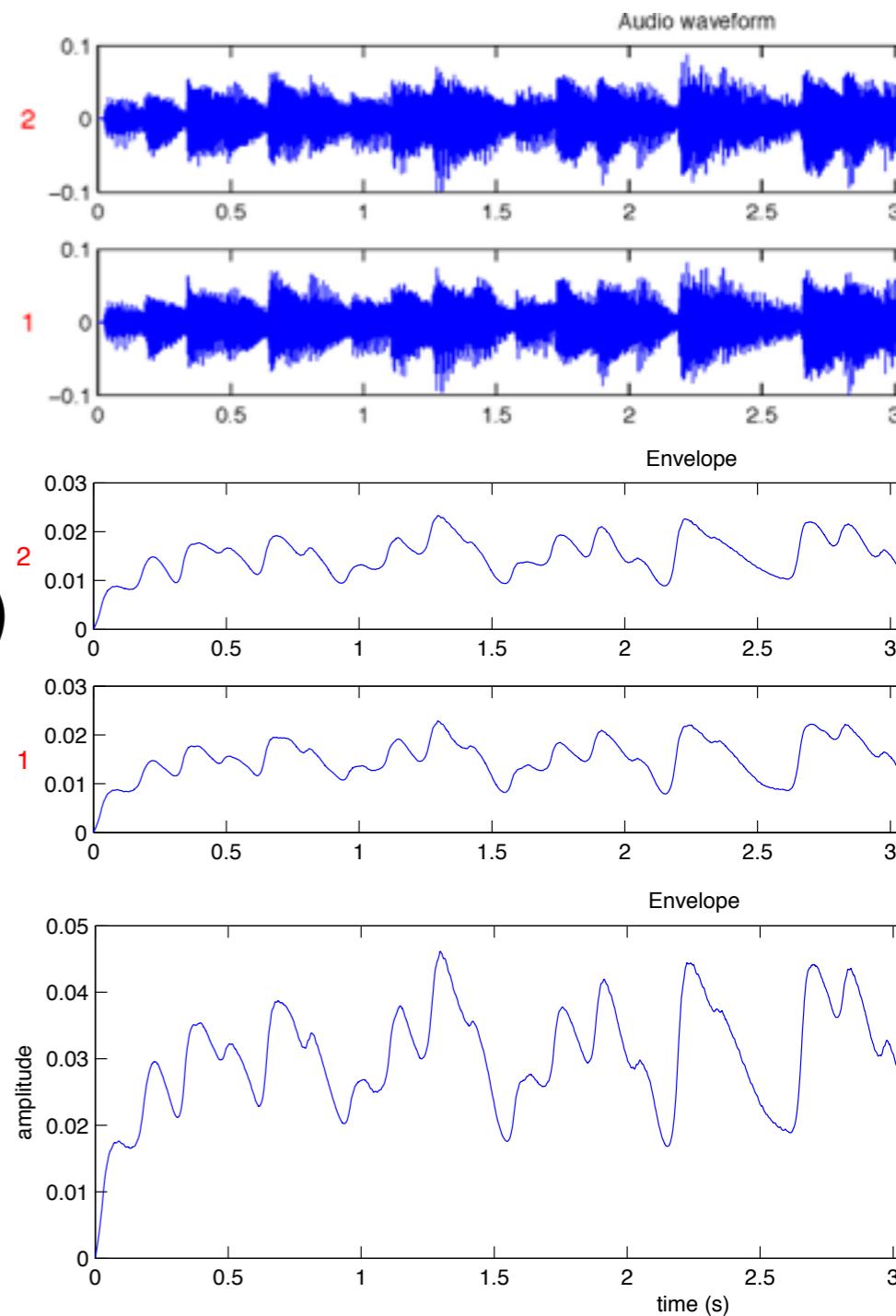
- $f = \text{sig.filterbank}(\dots)$
- $e = \text{sig.envelope}(f)$
- $e.sum$



# .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

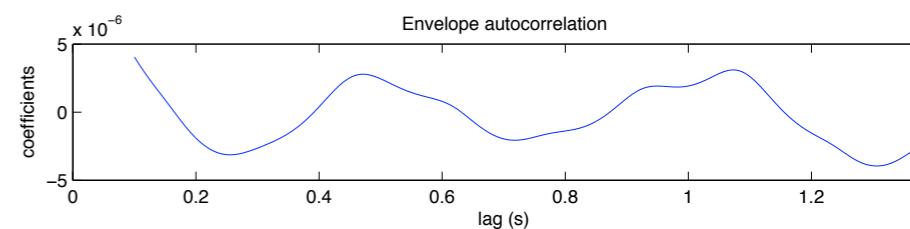
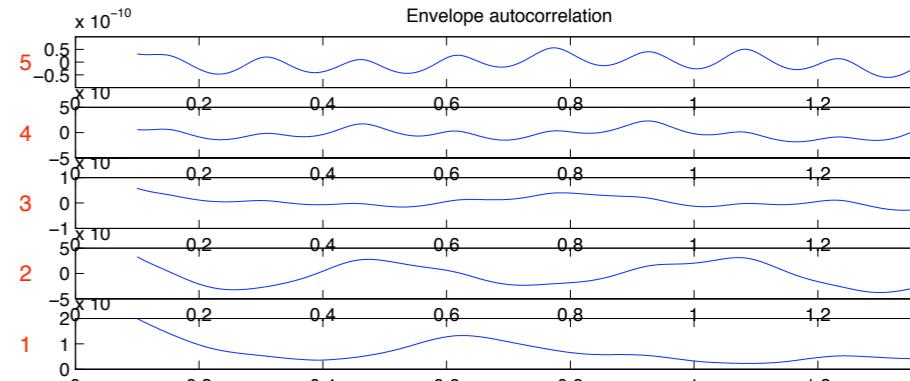
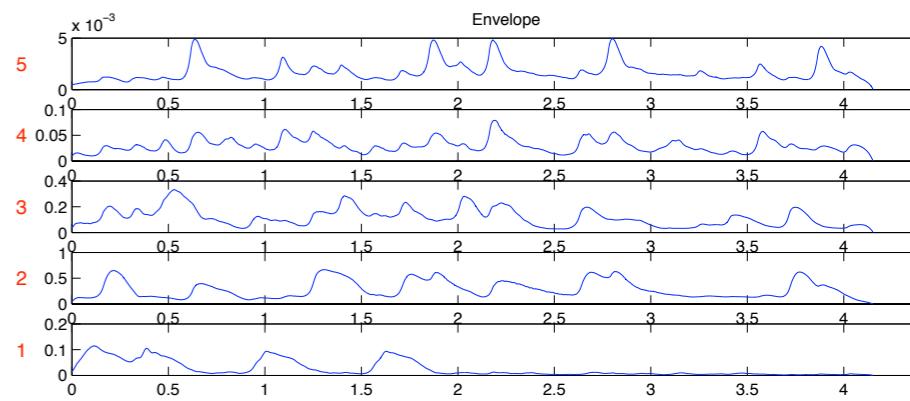
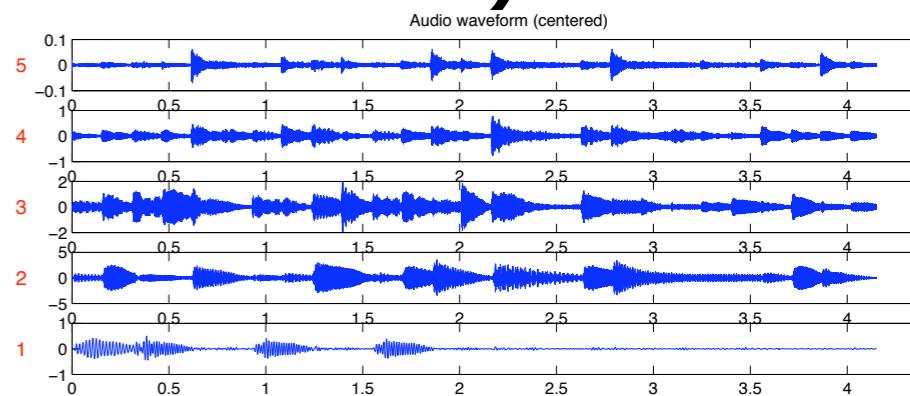
↓  
sig.filterbank

sig.envelope

sig.autocor

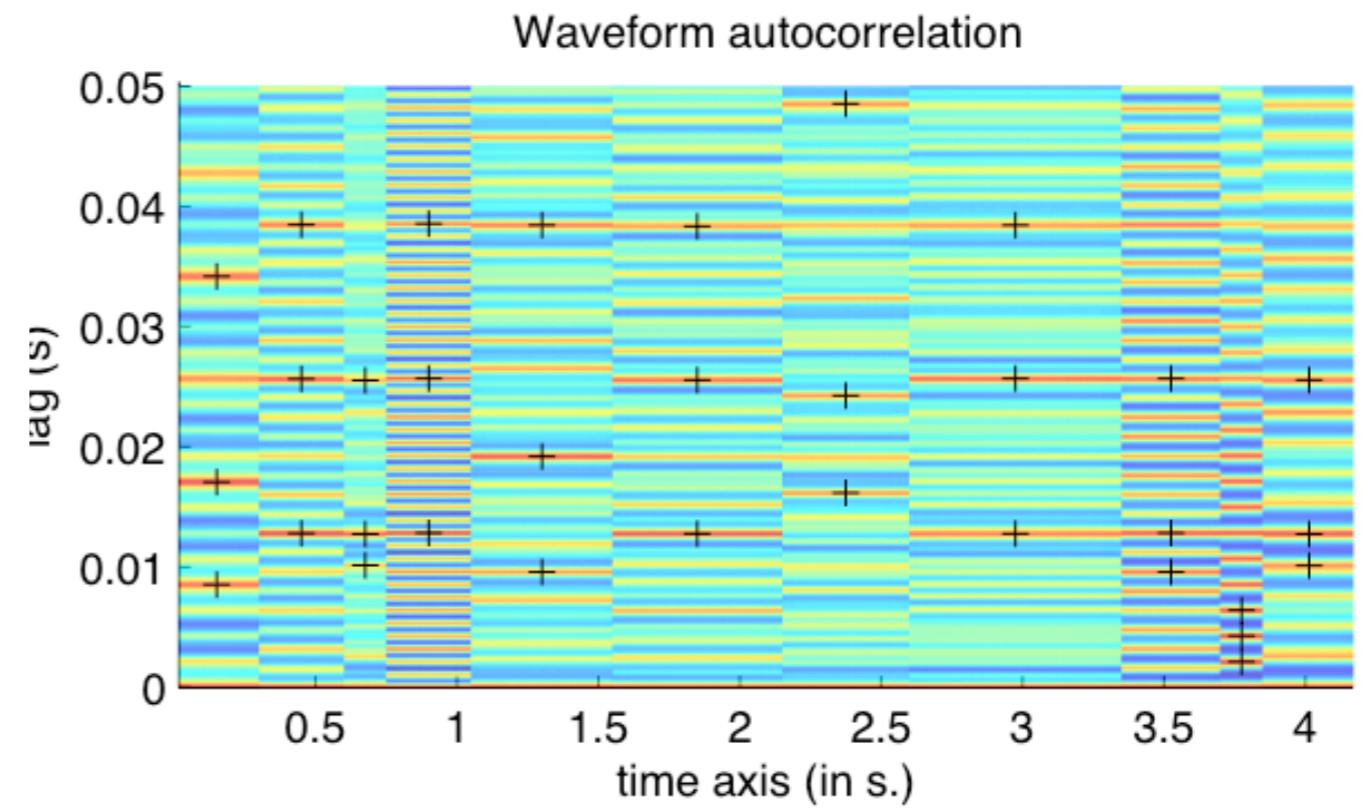
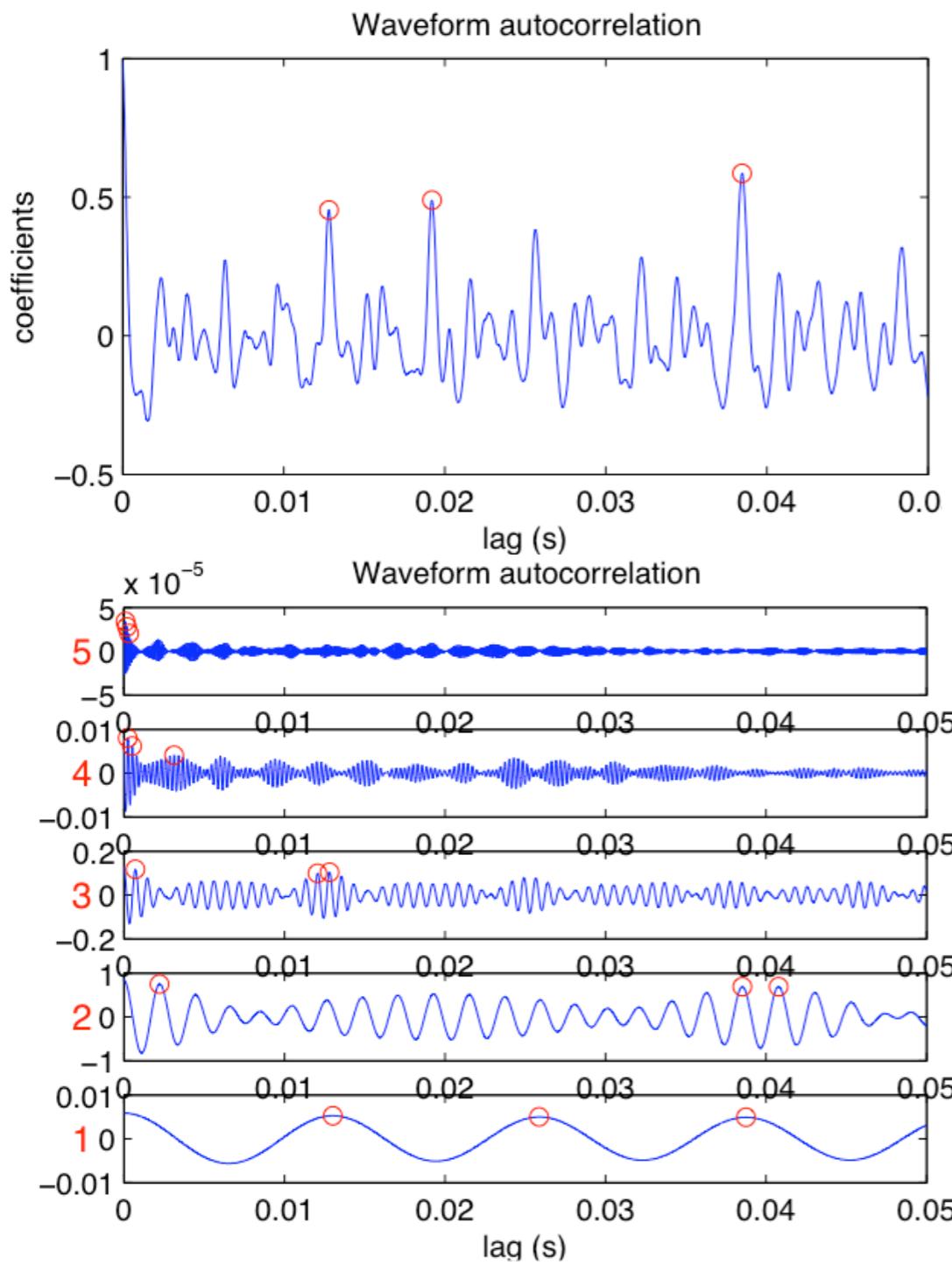
.sum

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



# *sig.peaks*

## peak picking

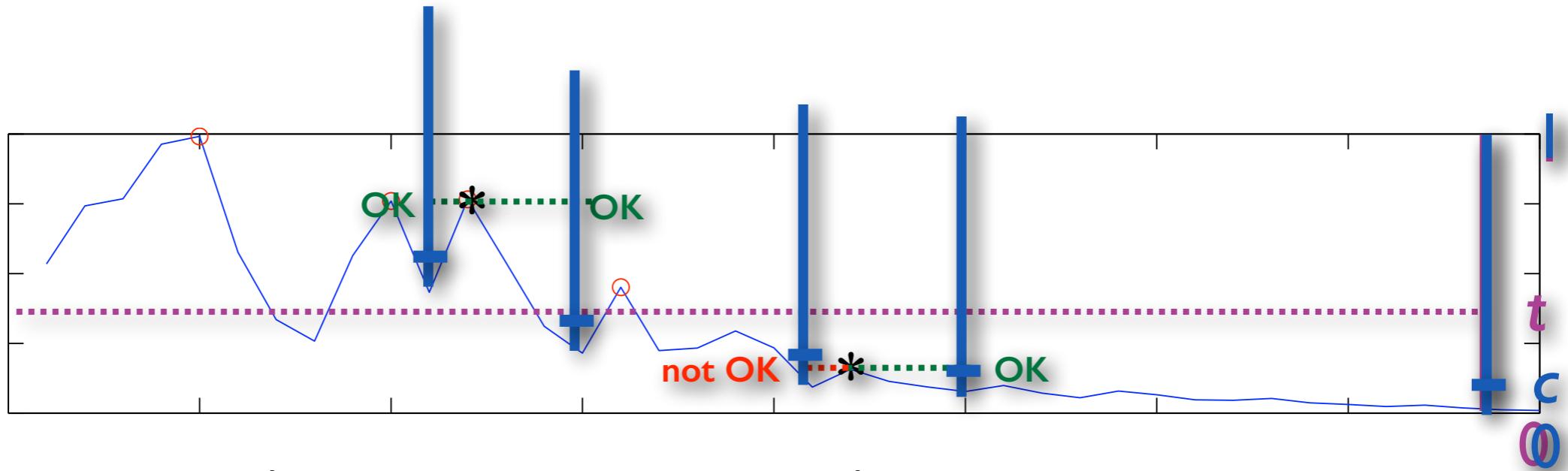


# *sig.peaks*

## 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

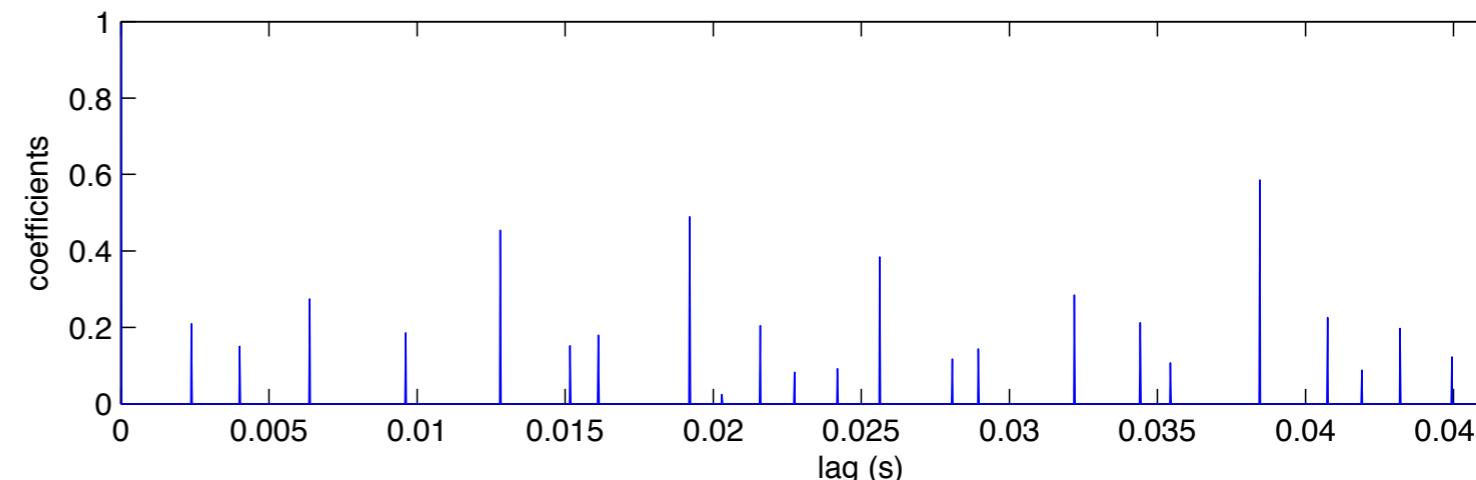
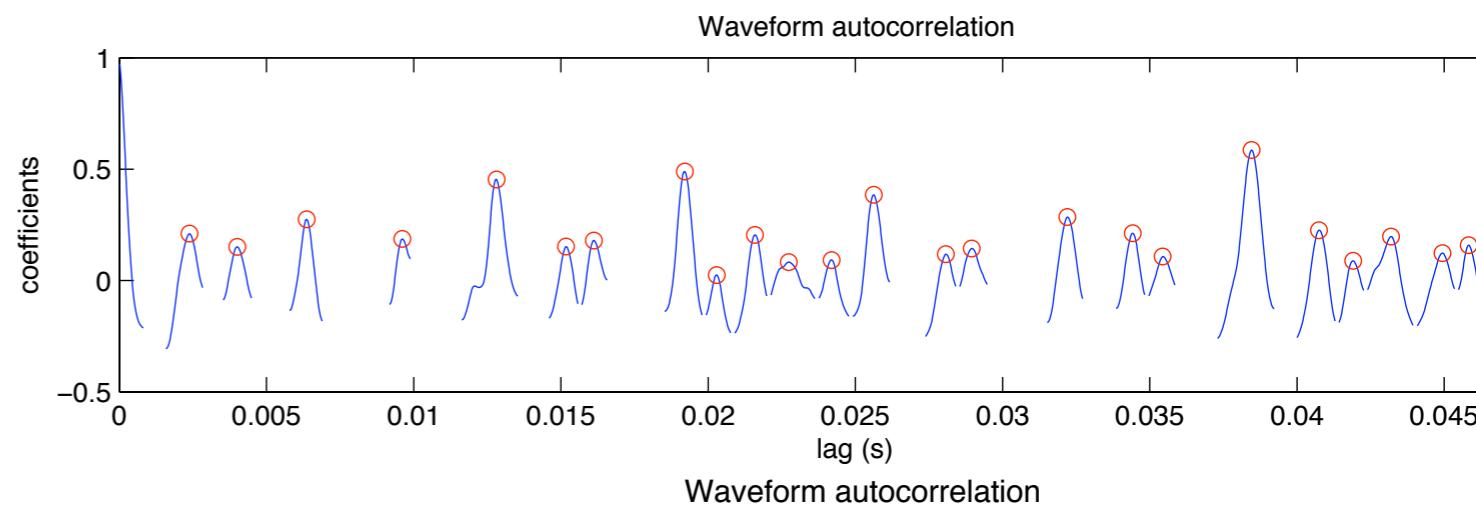
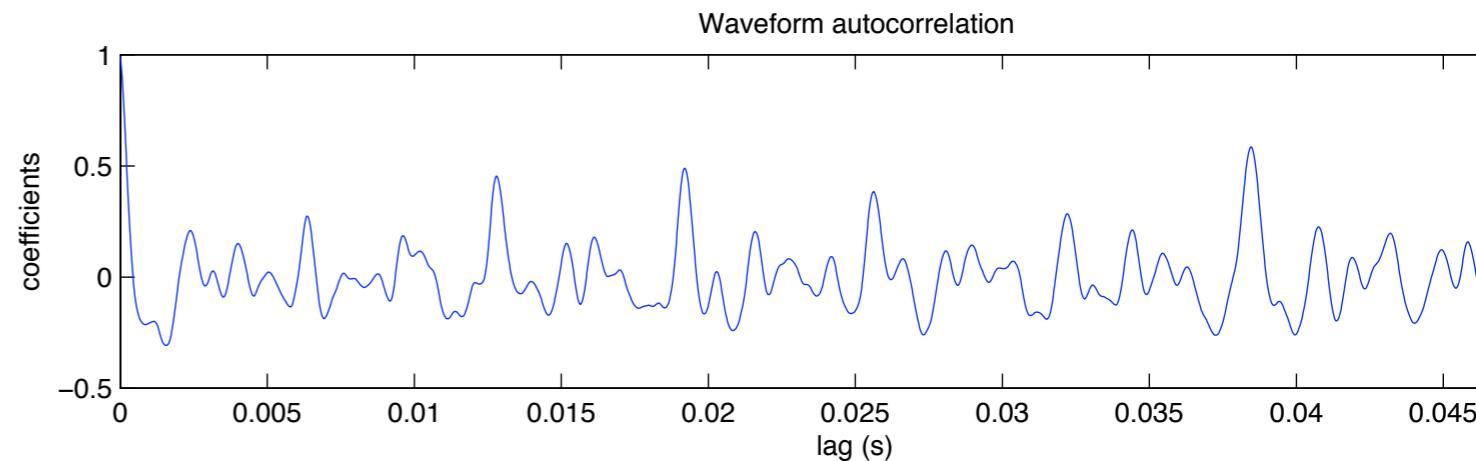
# *sig.peaks* peak picking



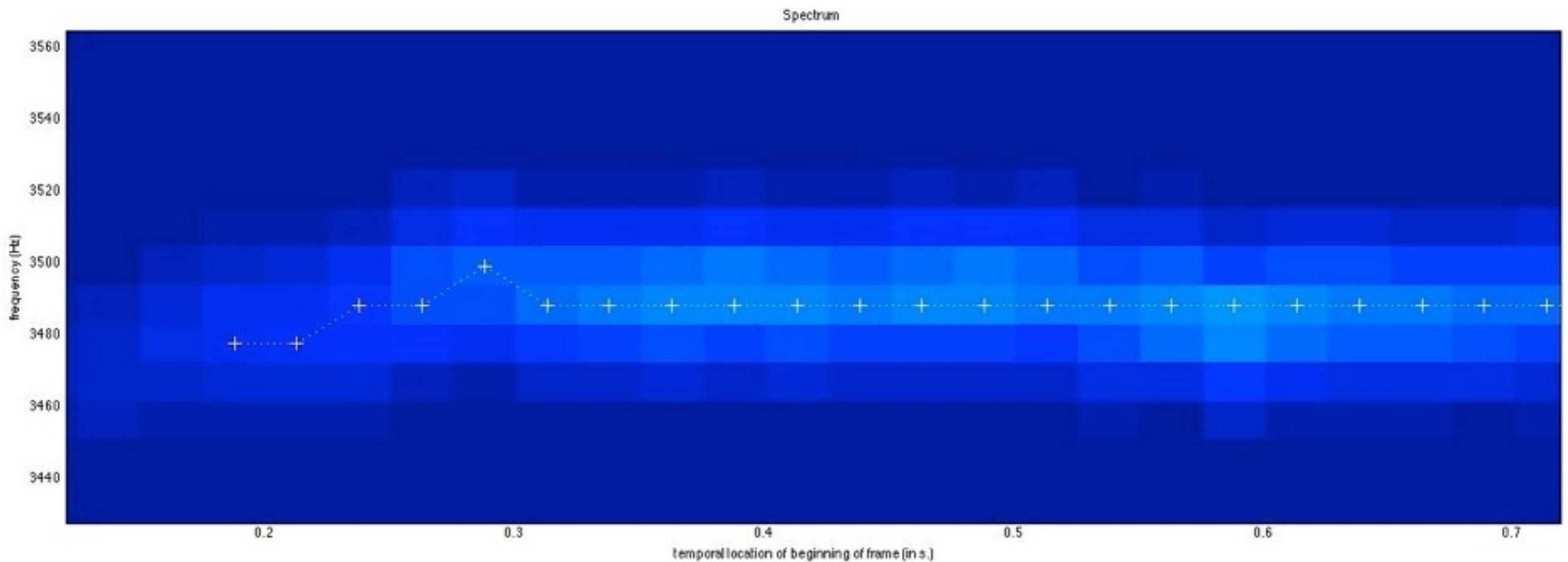
- *sig.peak(..., 'Threshold', 0)*
- *sig.peak(..., 'Contrast', .1)*
- *sig.peak(..., 'SelectFirst', .05)*
- *mus.peak(..., 'Reso', 'SemiTone')*

# *sig.peaks* peak picking

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



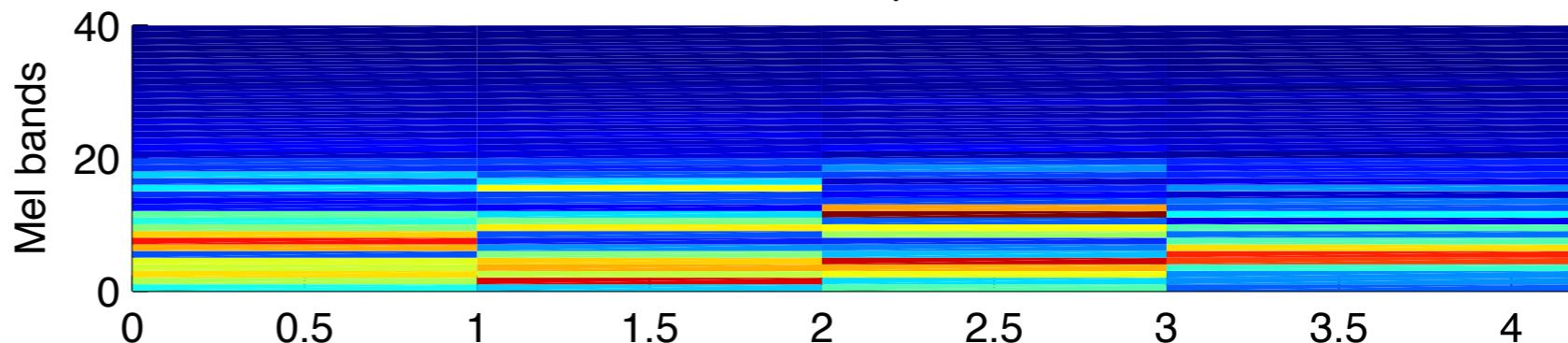
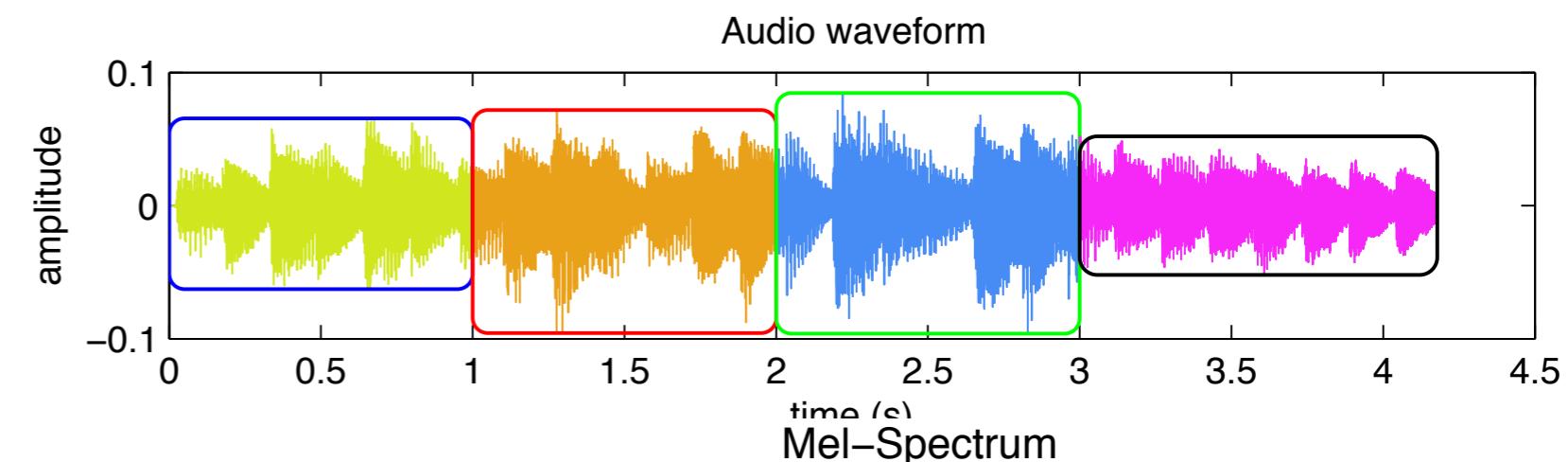
# `sig.peaks(..., 'Track')` peak tracking



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.

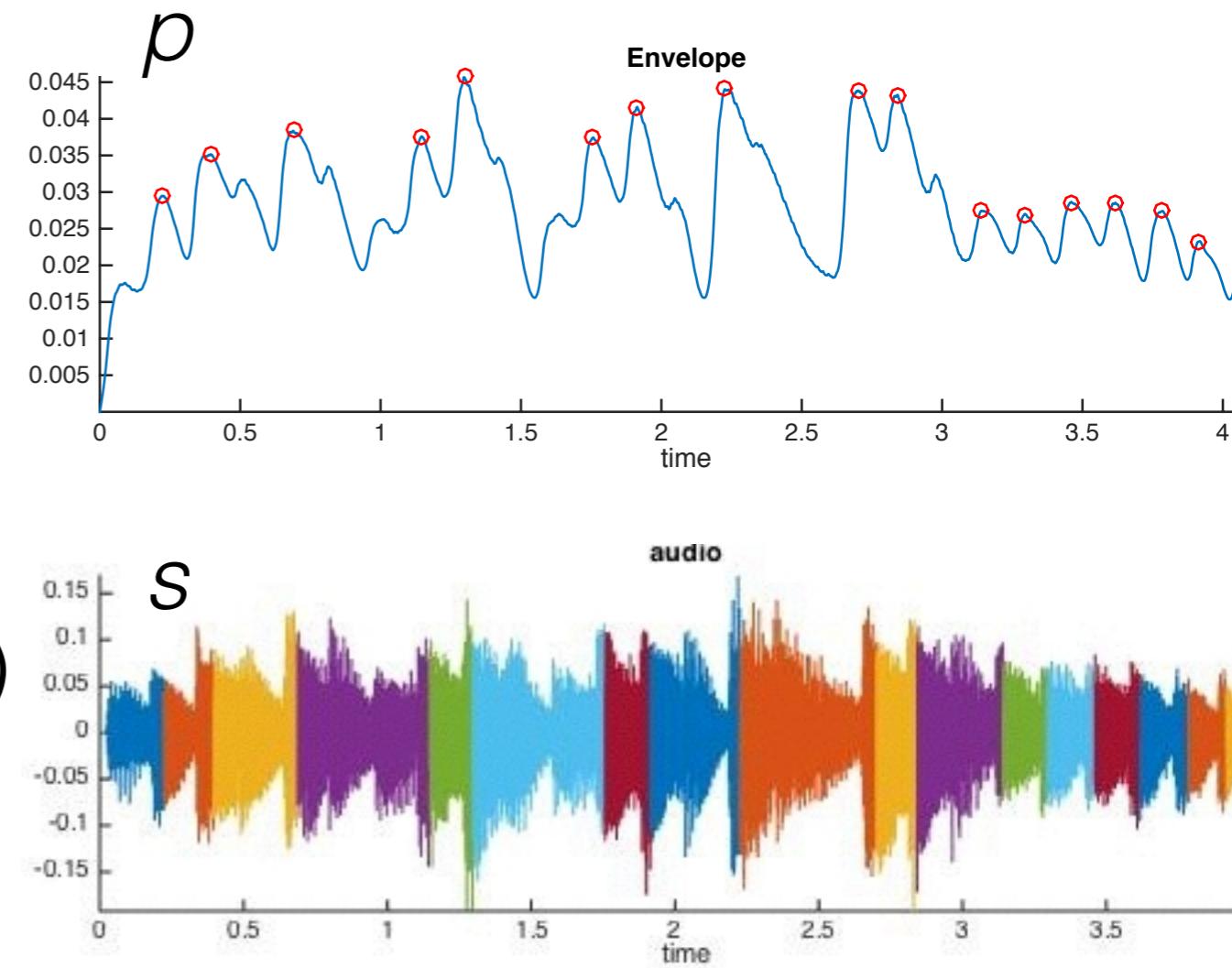
# *sig.segment* segmentation

- $s = \text{sig.segment}(\text{'myfile'}, [1\ 2\ 3])$
- $s.\text{play}(1:2)$
- $s.\text{save}$
- $\text{sig.spectrum}(s)$



# *sig.segment* segmentation

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



*Symbolic level*

*Audio level*

**Notes**

Dynamics

Sound

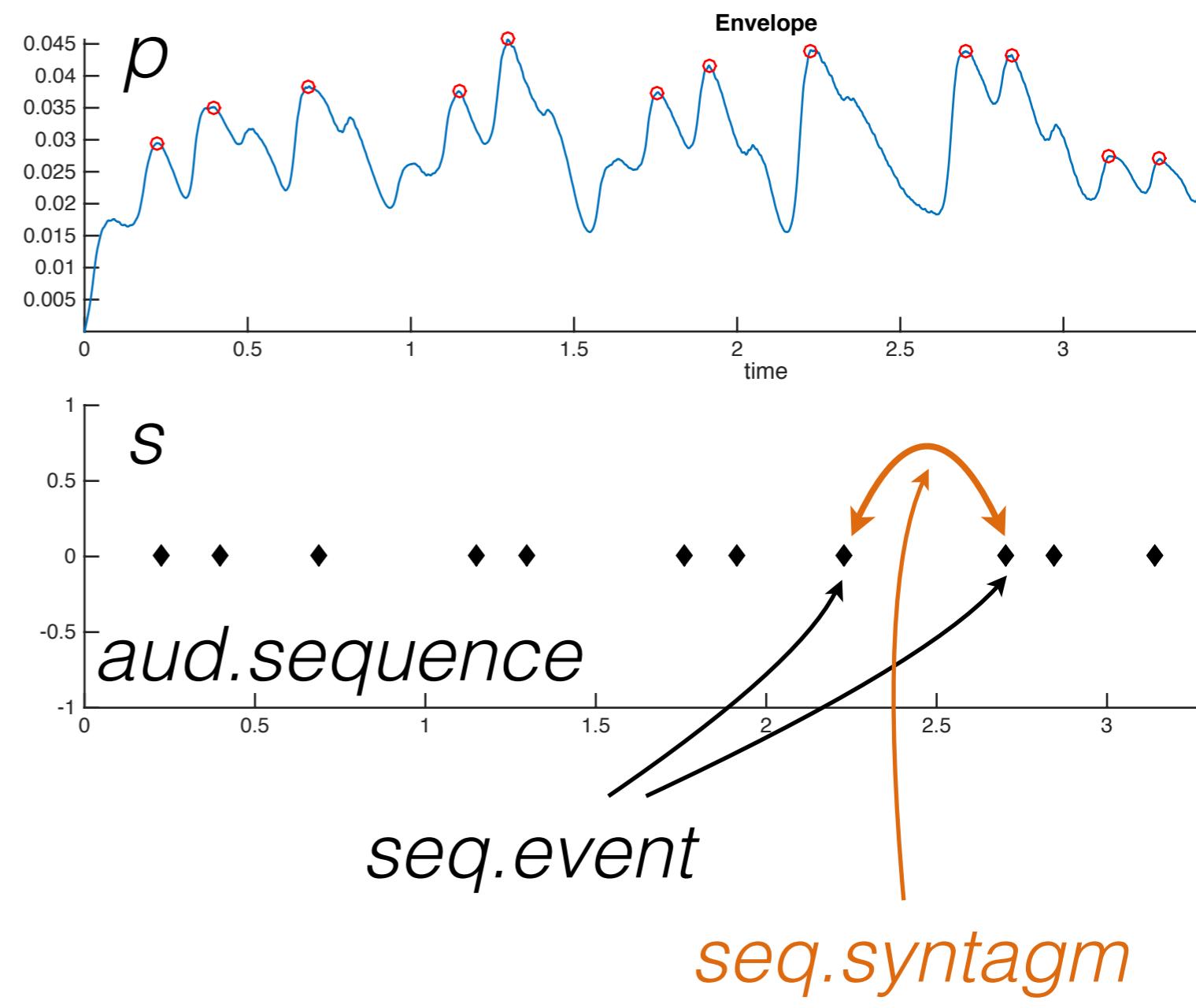


# *aud.score* onset detection

- $e =$   
*sig.envelope('myfile')*
- $p = sig.peaks(e,$   
*'Contrast', .1)*
- $s = aud.score(p)$

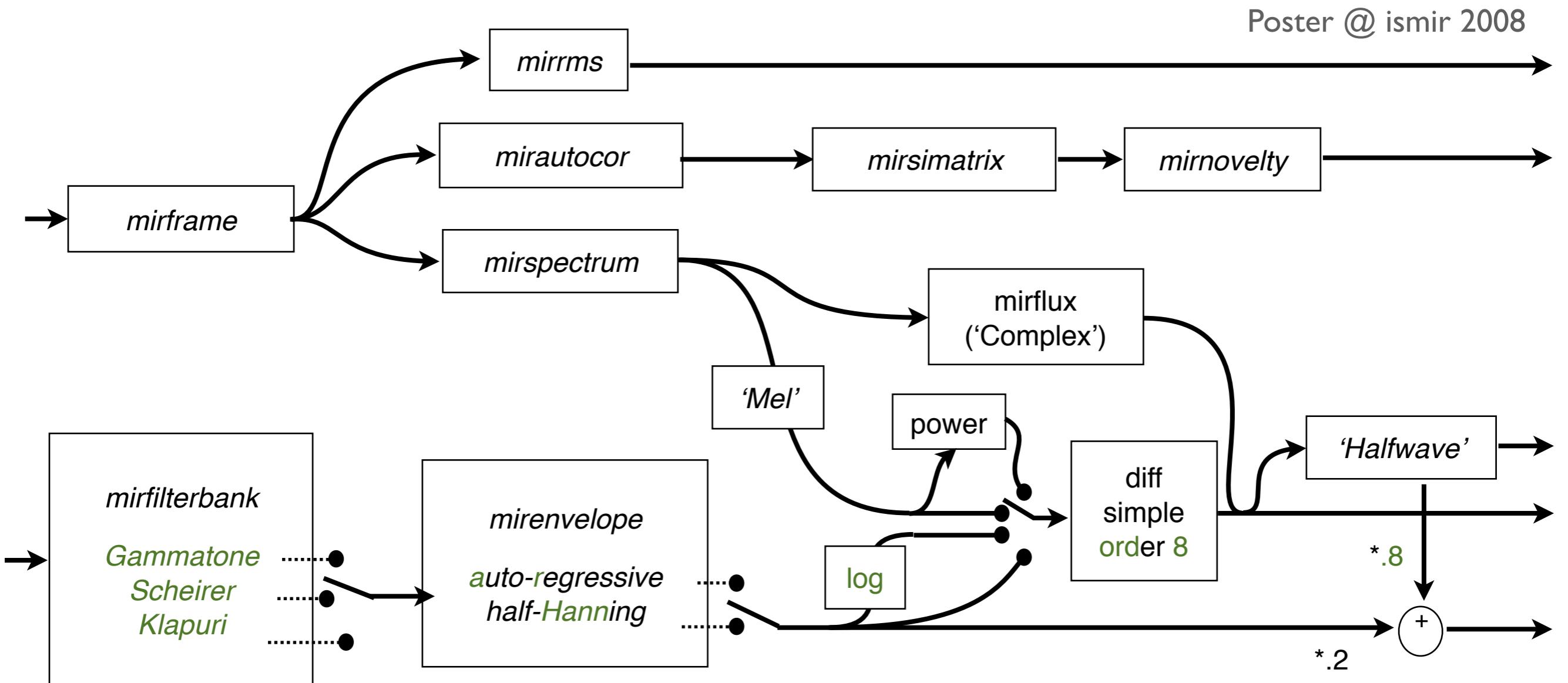
In short:

- $[s p] =$   
*aud.score('myfile',*  
***Contrast***, .1)



# *mironsets*

## onset detection

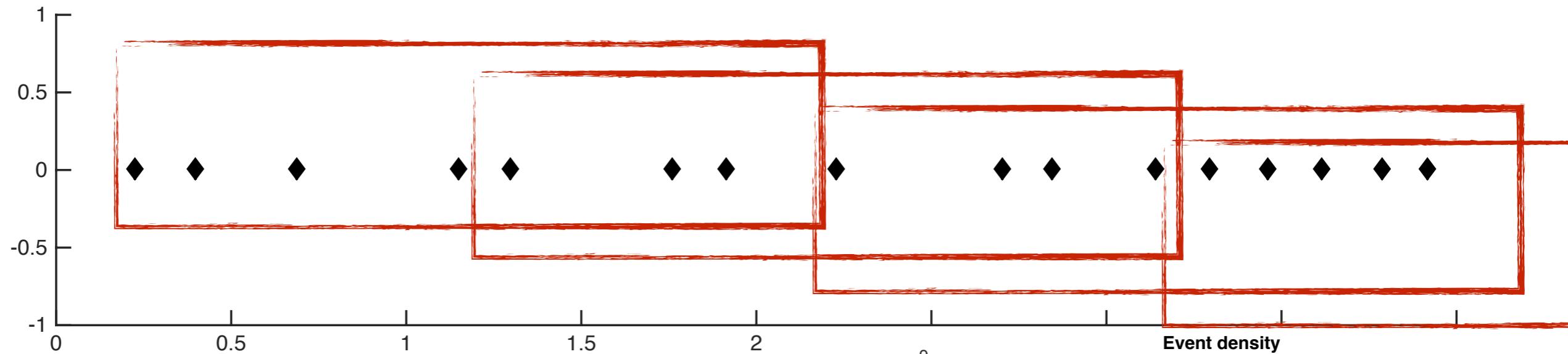


*aud.score(..., 'Scheirer')*

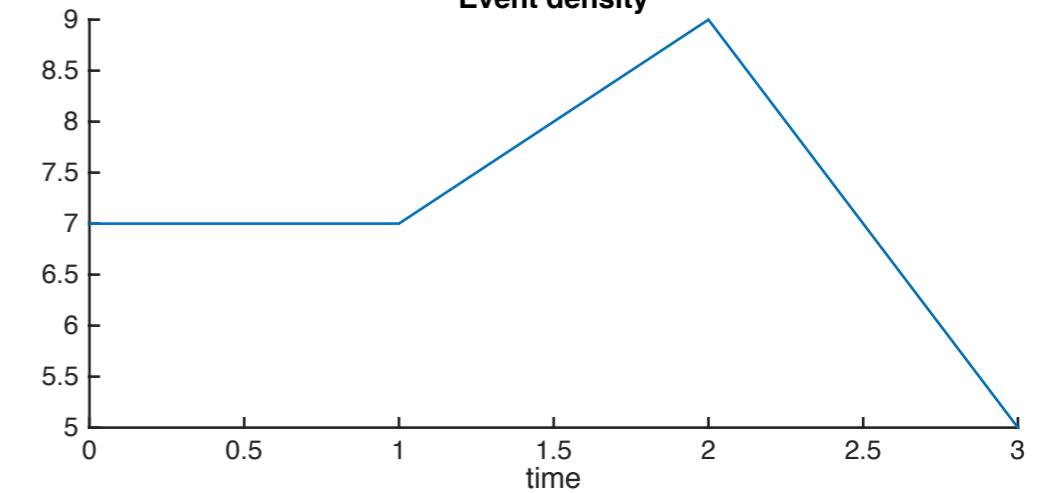
*aud.score(..., 'Klapuri')*

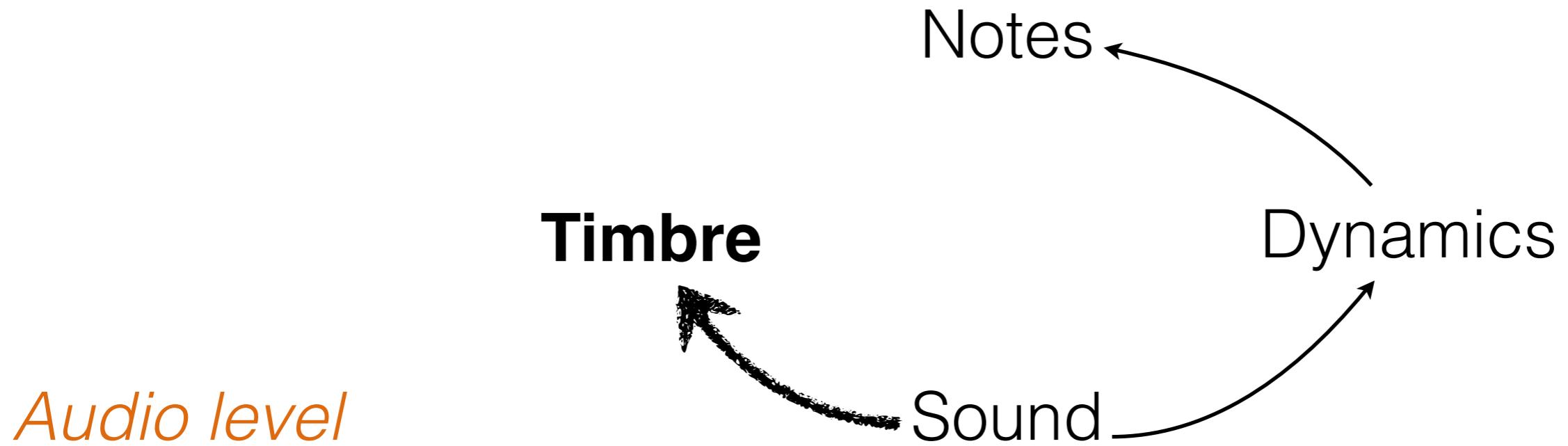
# *aud.eventdensity*

## temporal density of events

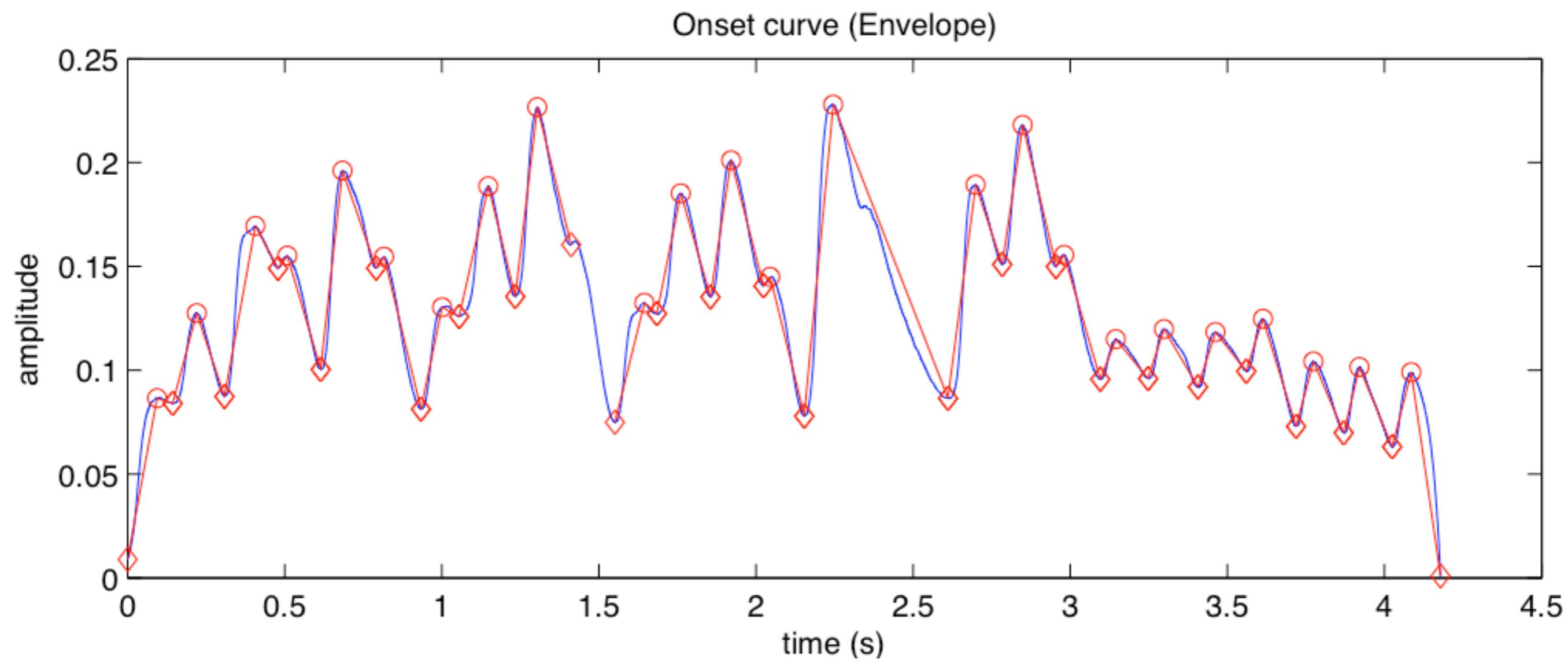


- `a = aud.score('audio.wav')`
- `aud.eventdensity(a)`
- `aud.eventdensity('audio.wav', 'Frame')`
- `aud.eventdensity('score.mid')`



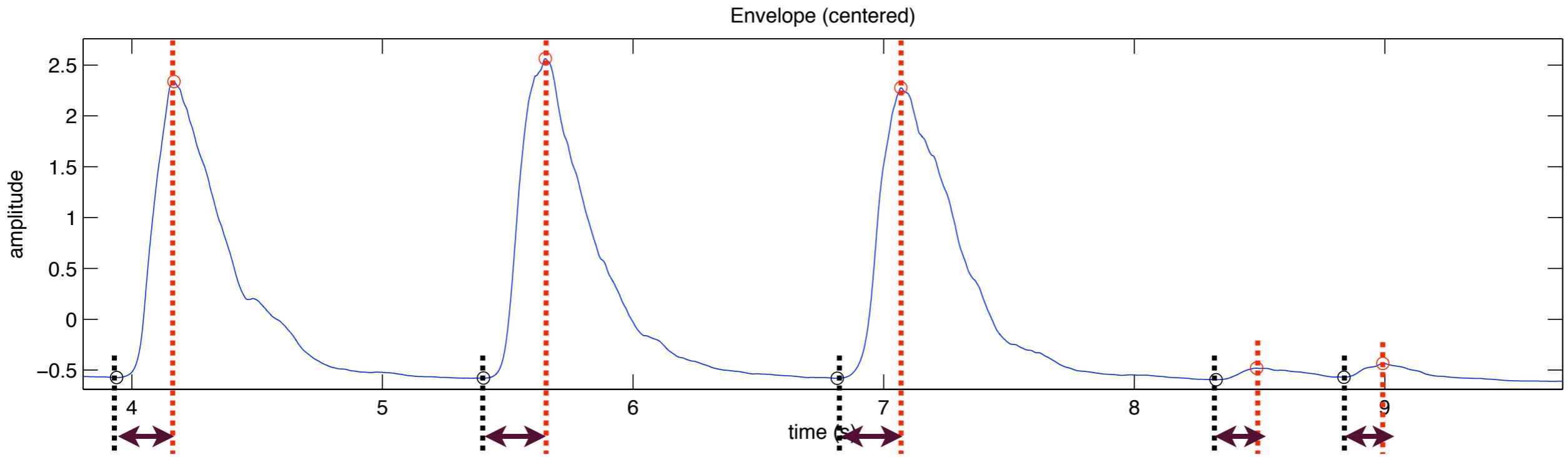


*aud.score(...,  
'Attack', 'Release')*



# *aud.attacktime*

## duration of note attacks

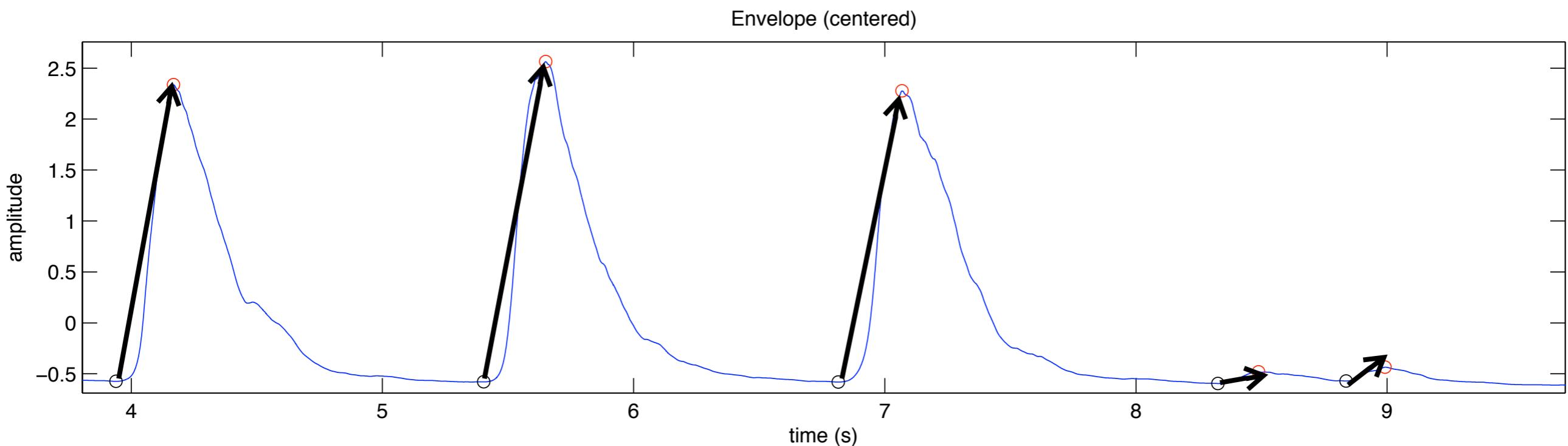


- *aud.attacktime(..., ‘Lin’)*: duration in seconds
- *aud.attacktime(..., ‘Log’)*: duration in log scale

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.

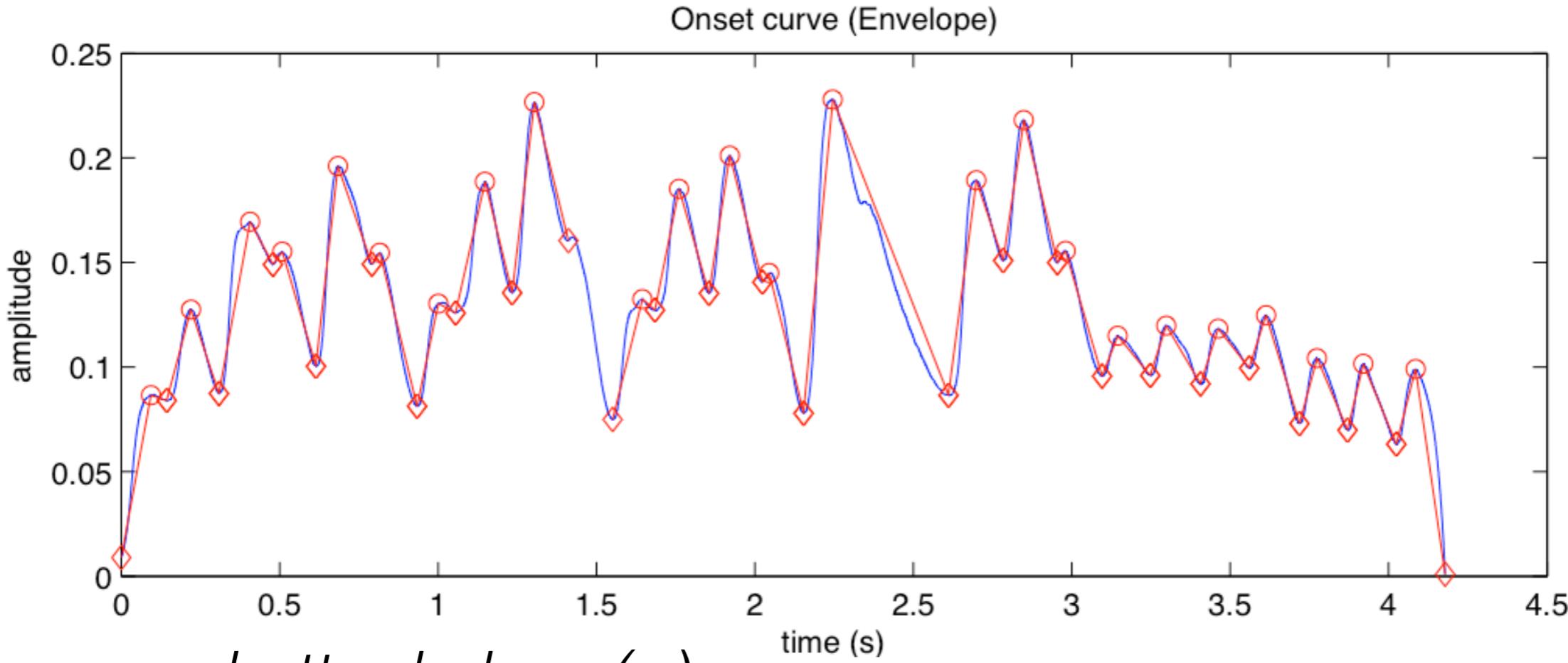
# *aud.attackslope*

## average slope of note attacks

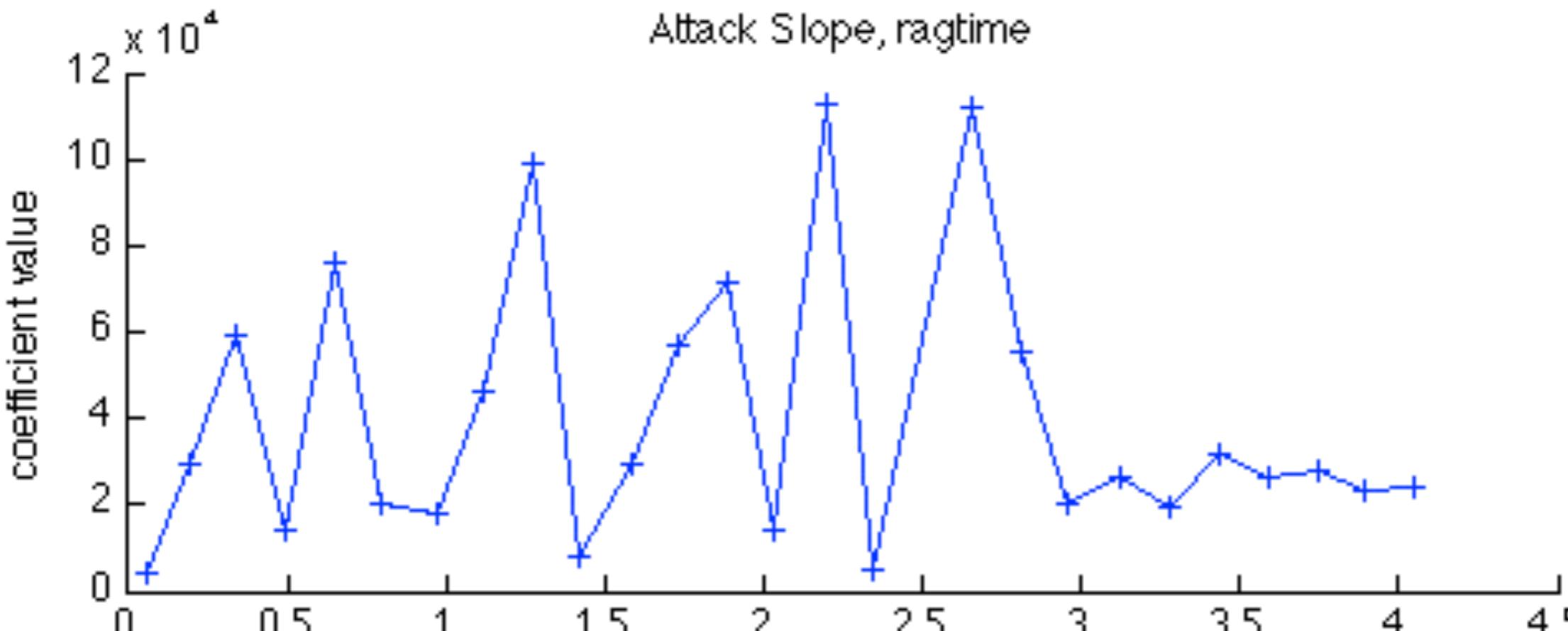


- *mirattackslope(..., 'Diff')*: average slope
- *mirattackslope(..., 'Gauss')*: gaussian average, highlighting the middle of the attack phase

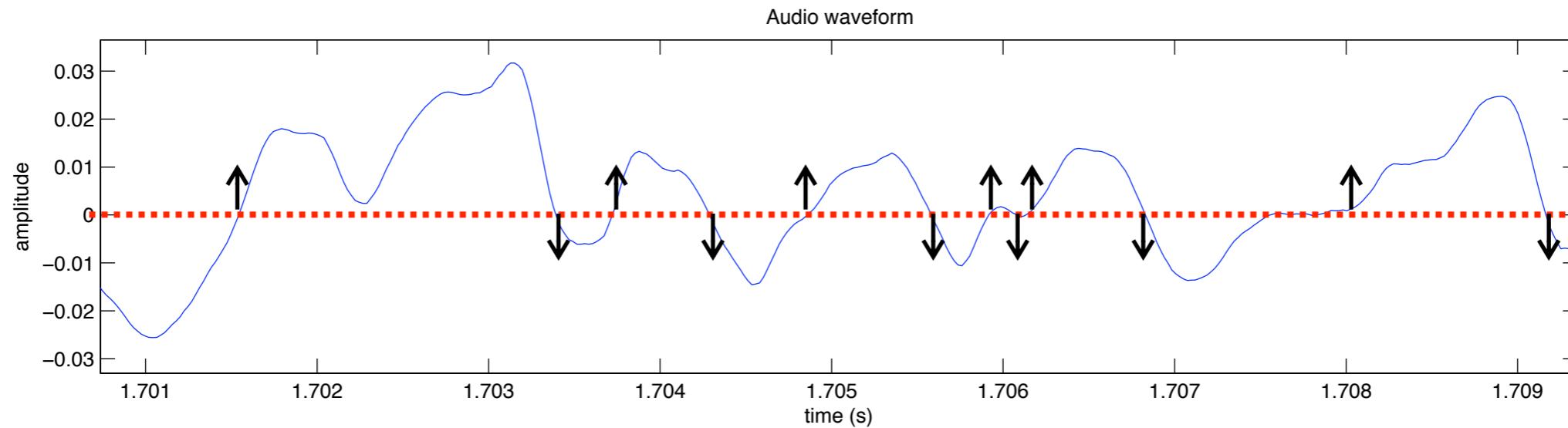
- $o = \text{aud.onsets}(\text{'audiofile'}, \text{'Attack'}, \text{'Release'})$



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

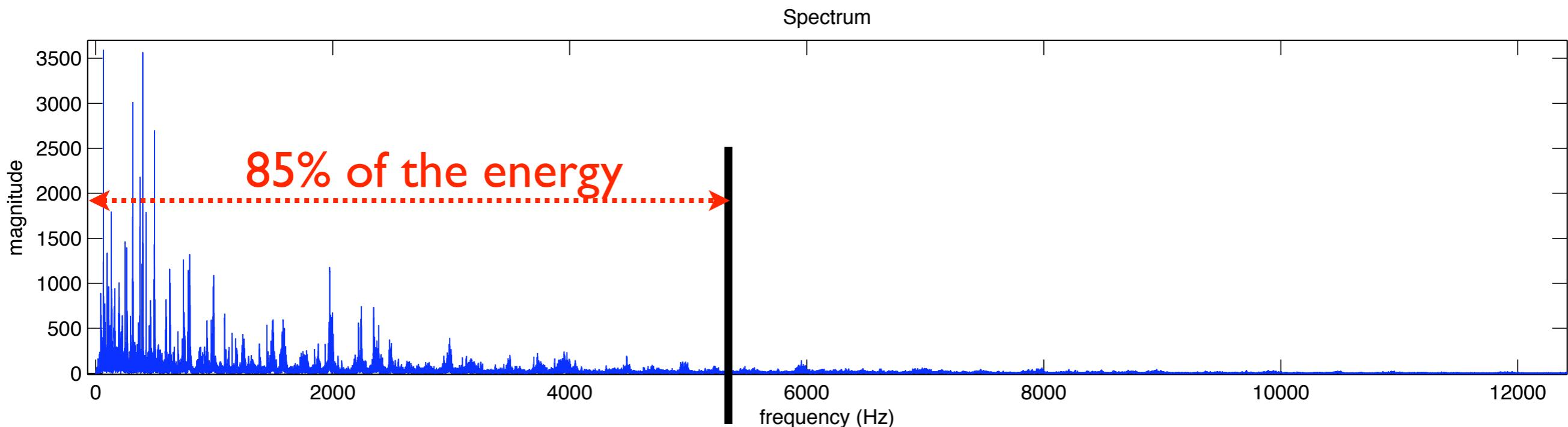


# *sig.zerocross* waveform sign-change rate



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

# *sig.rolloff* high-frequency energy (I)

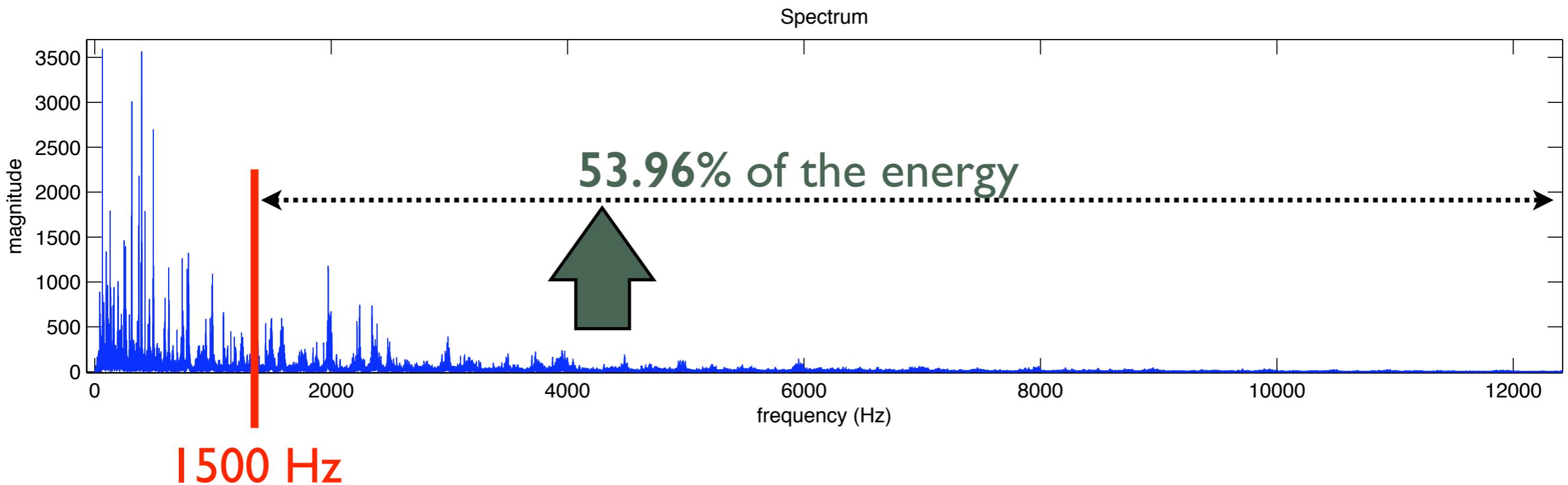


*mirrolloff(..., 'Threshold', .85)*

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)

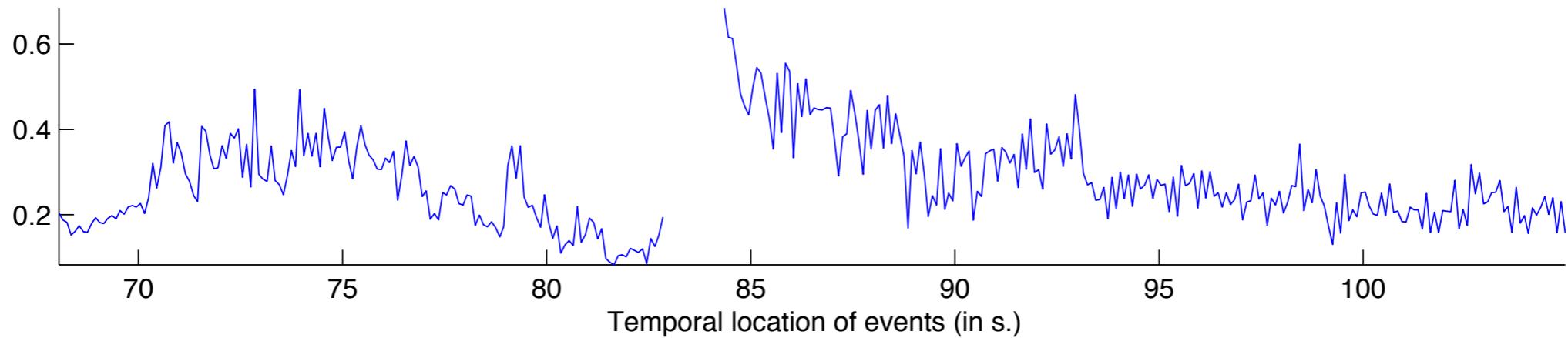


- *aud.brightness(..., 'CutOff', 1500)* (in Hz)
- *aud.brightness(..., 'Unit', u)*    u = '/1' or '%'
  - 3000 Hz in Juslin 2001, p. 1802.
  - 1500 Hz and 1000 Hz in Laukka, Juslin and Bresin 2005.

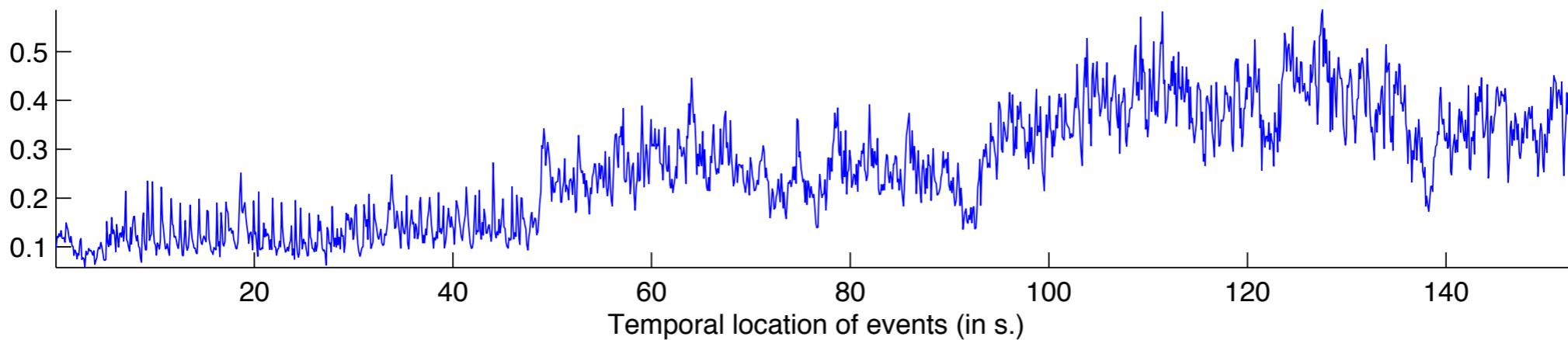
# *aud.brightness* high-frequency energy (II)

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

frame length = .05 s  
frame hop = 50%



Beethoven, 9th Symphony, *Scherzo*

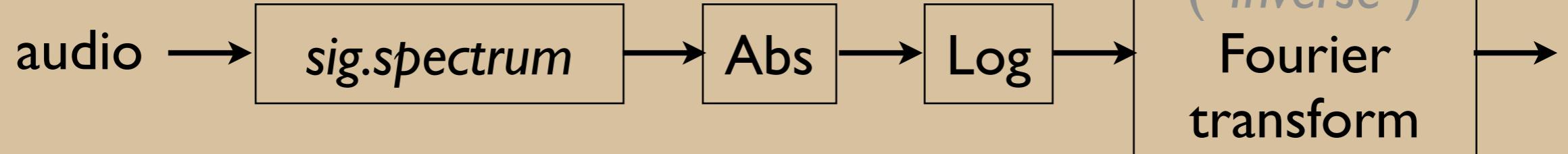


Beethoven, 7th Symphony, *Allegretto*

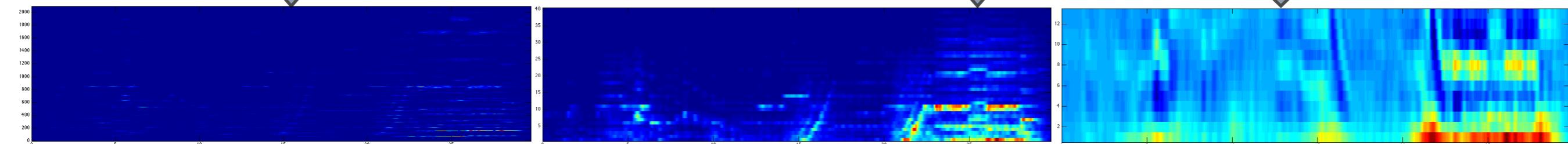
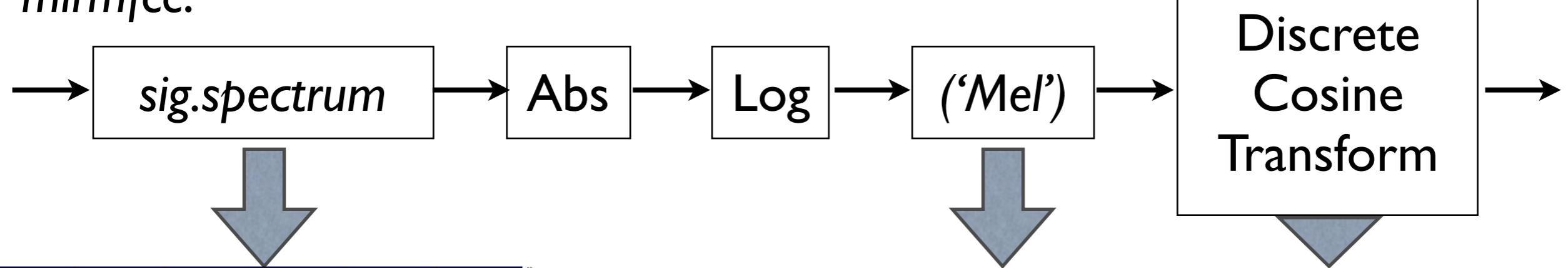
# *aud.mfcc*

## mel-frequency cepstral coefficients

*mircepstrum:*



*mirmfcc:*

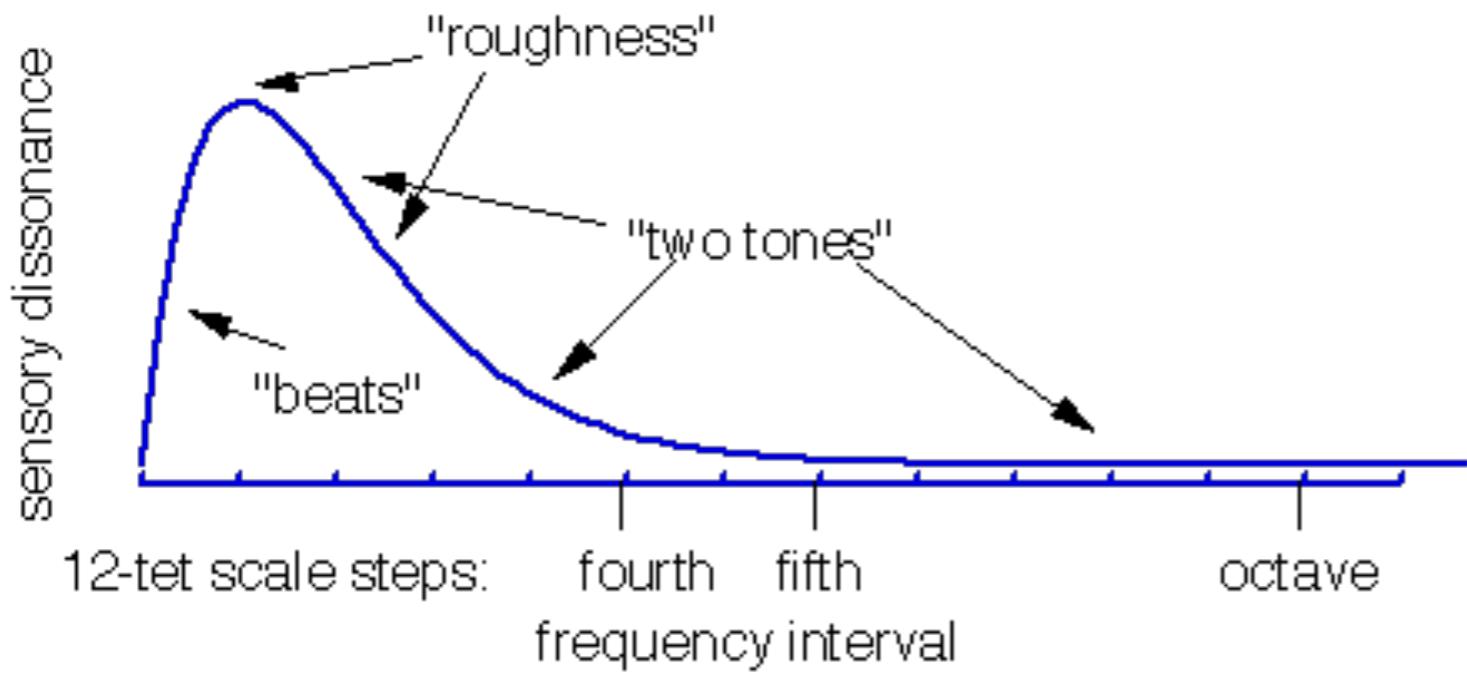


- Description of spectral shape

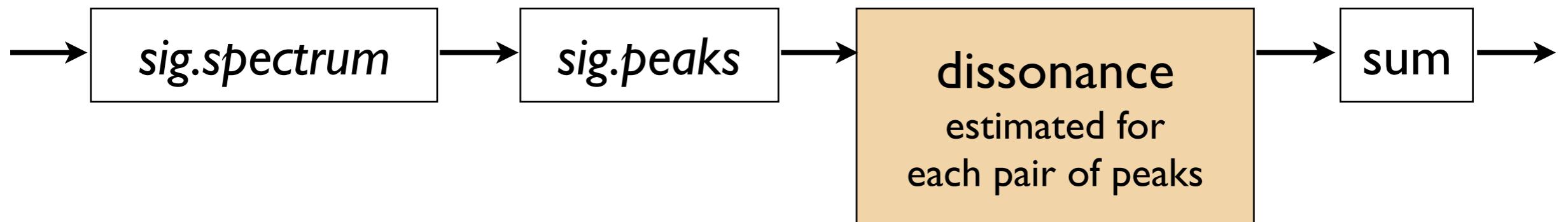
# *aud.roughness*

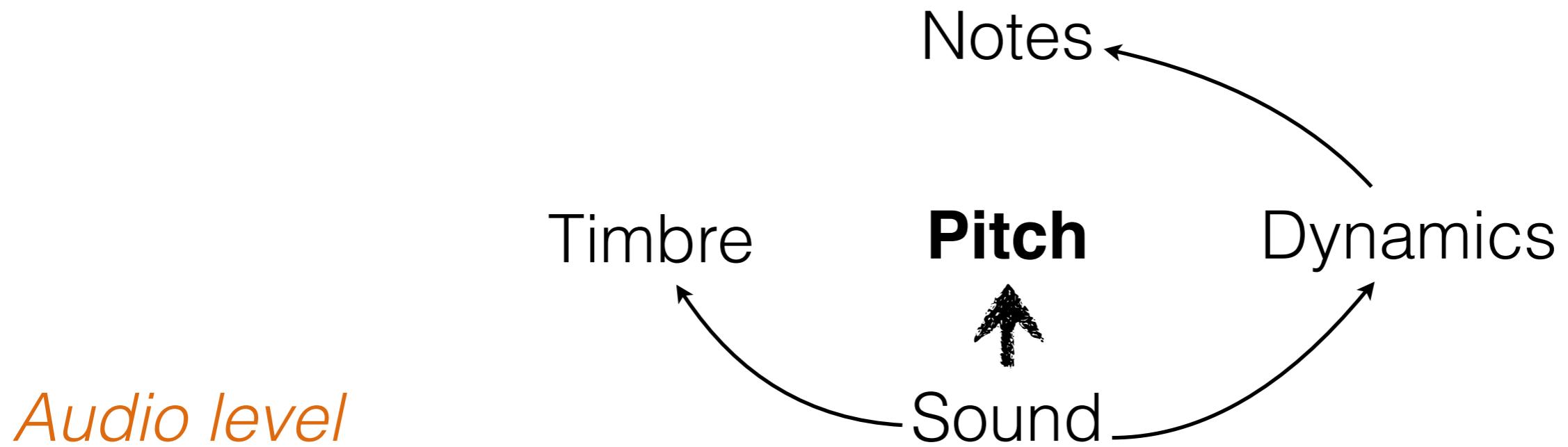
## sensory dissonance

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

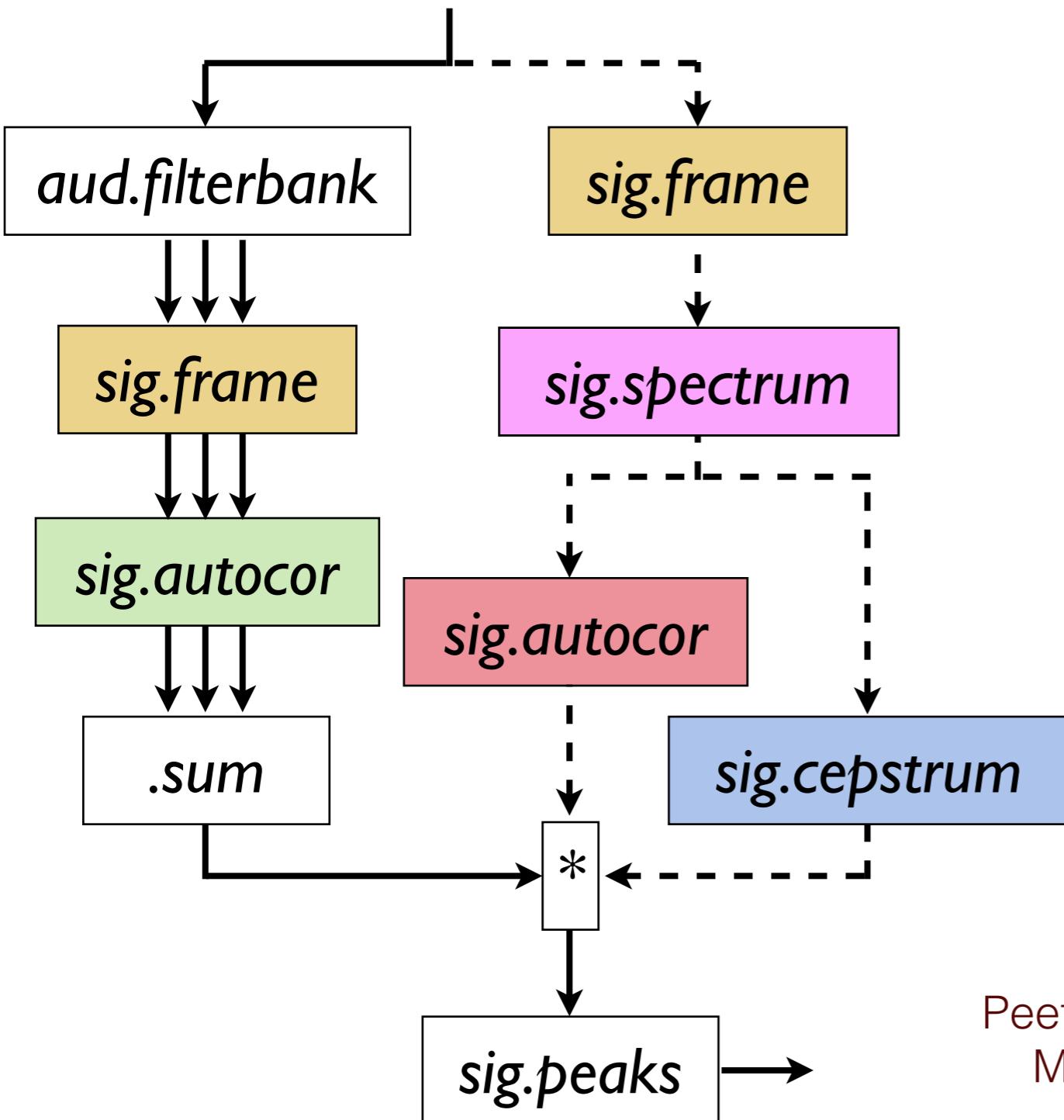


Dissonance produced by two sinusoids depending on their frequency ratio



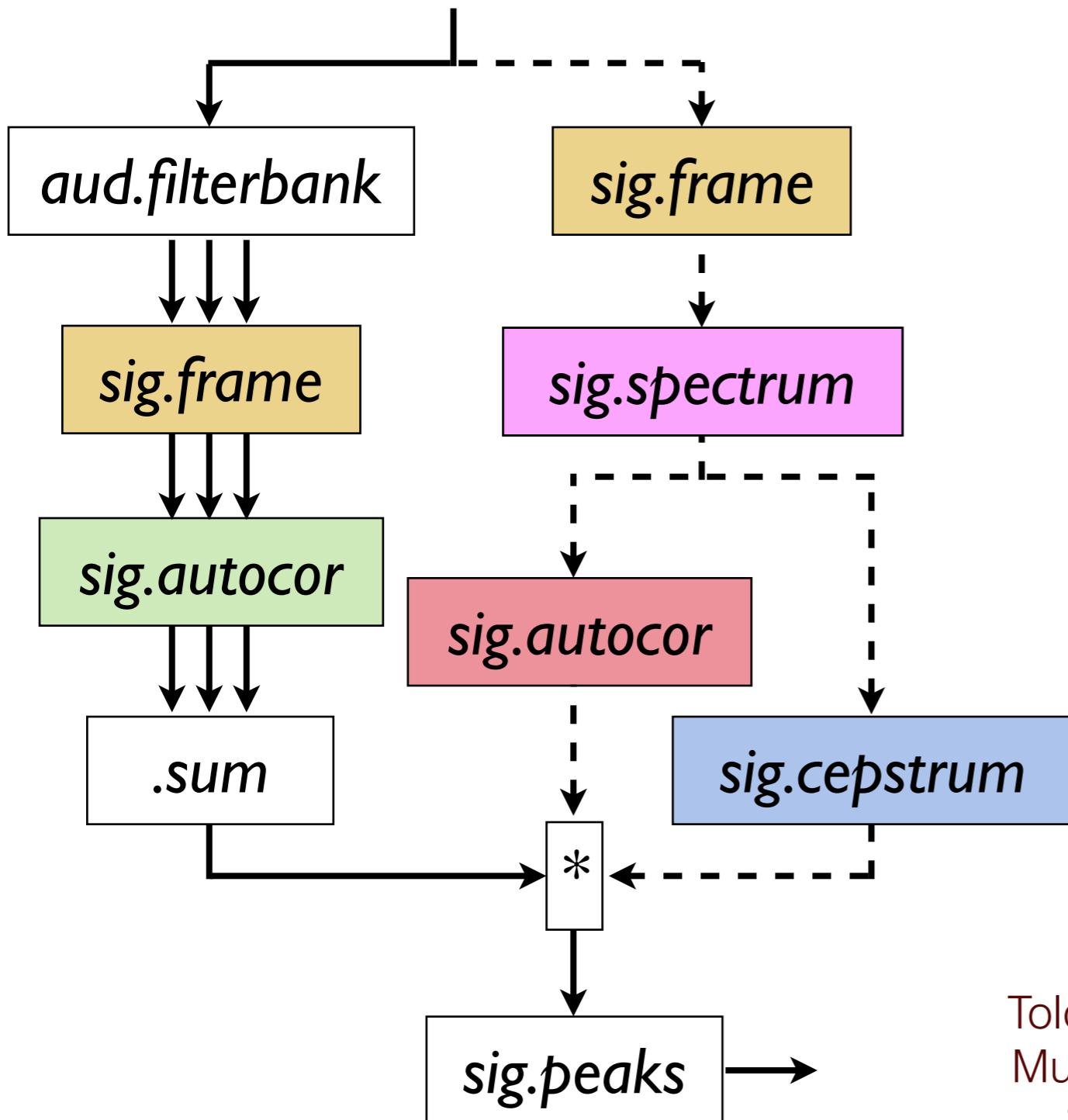


# *aud.pitch* f0 estimation



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

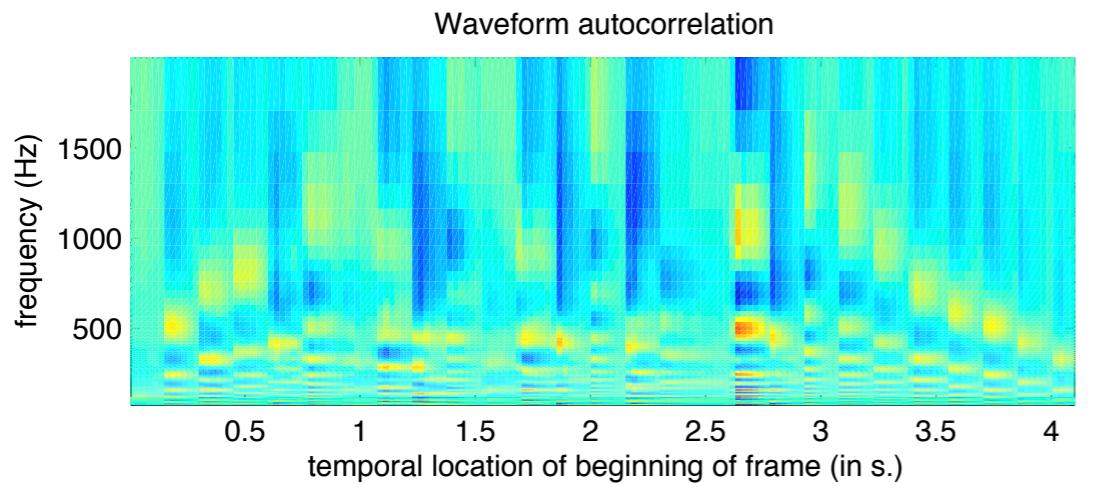
# *aud.pitch* f0 estimation



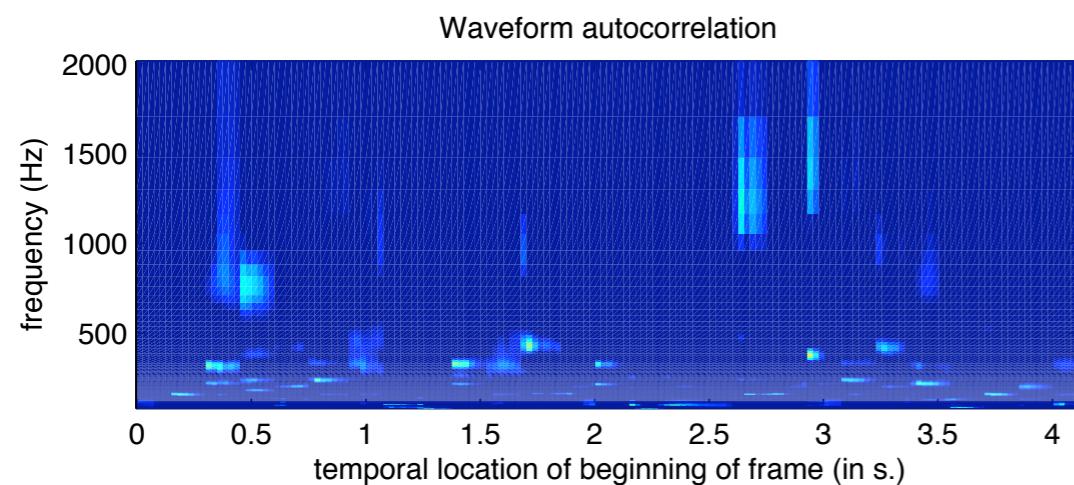
- *aud.pitch(..., 'Tolonen')*
- *aud.pitch(..., '2Channels', 'Enhanced', 2:10, 'Generalized', .67)*

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

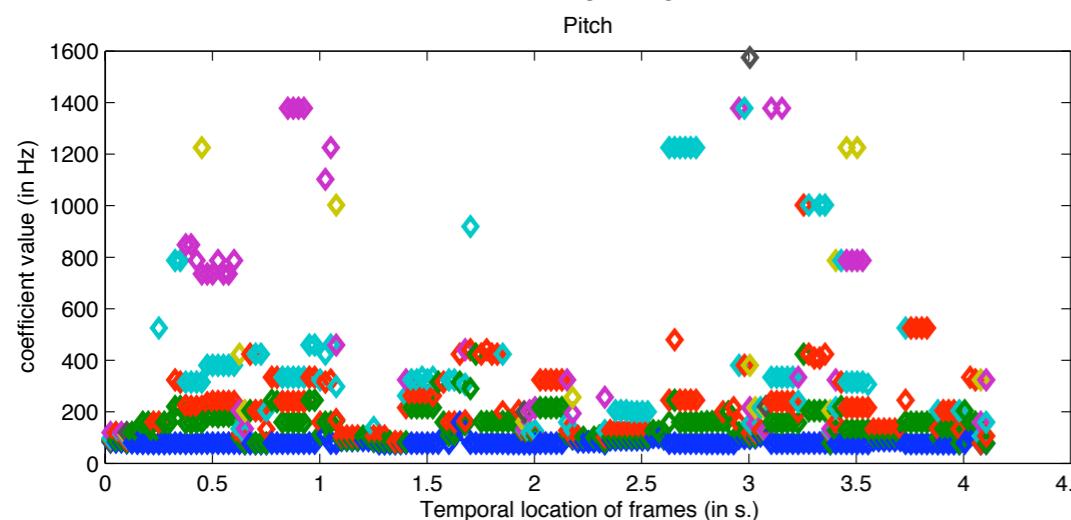
# *aud.pitch* f0 estimation



- $ac = \text{sig.autocor}(\text{'ragtime'}, \text{'Frame'})$



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



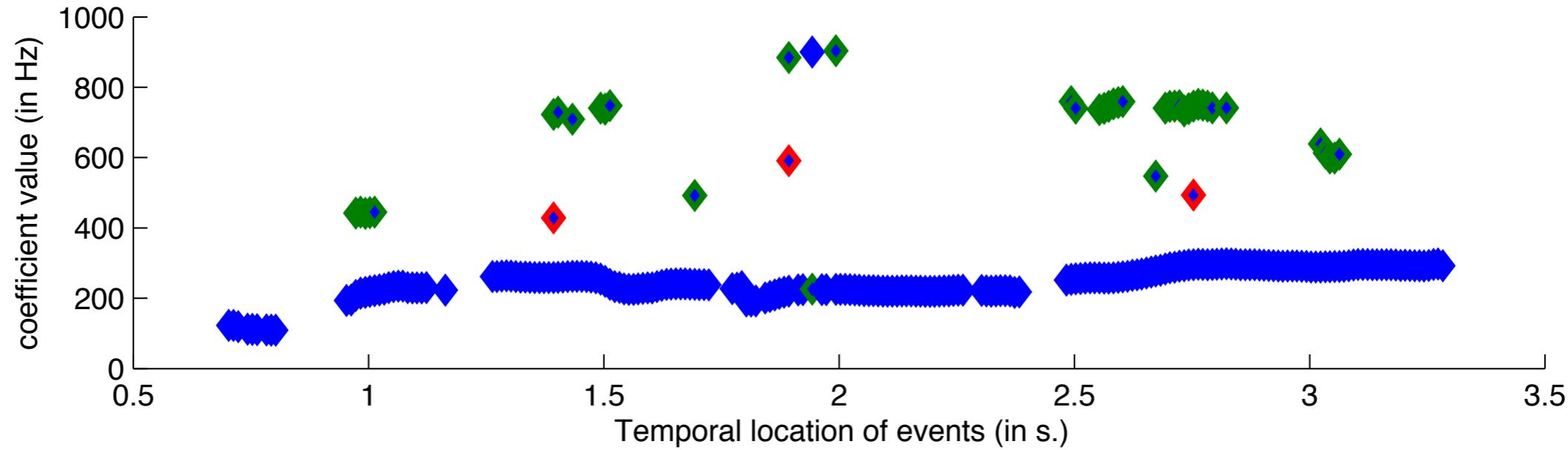
- $\text{aud.pitch}(ac)$

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

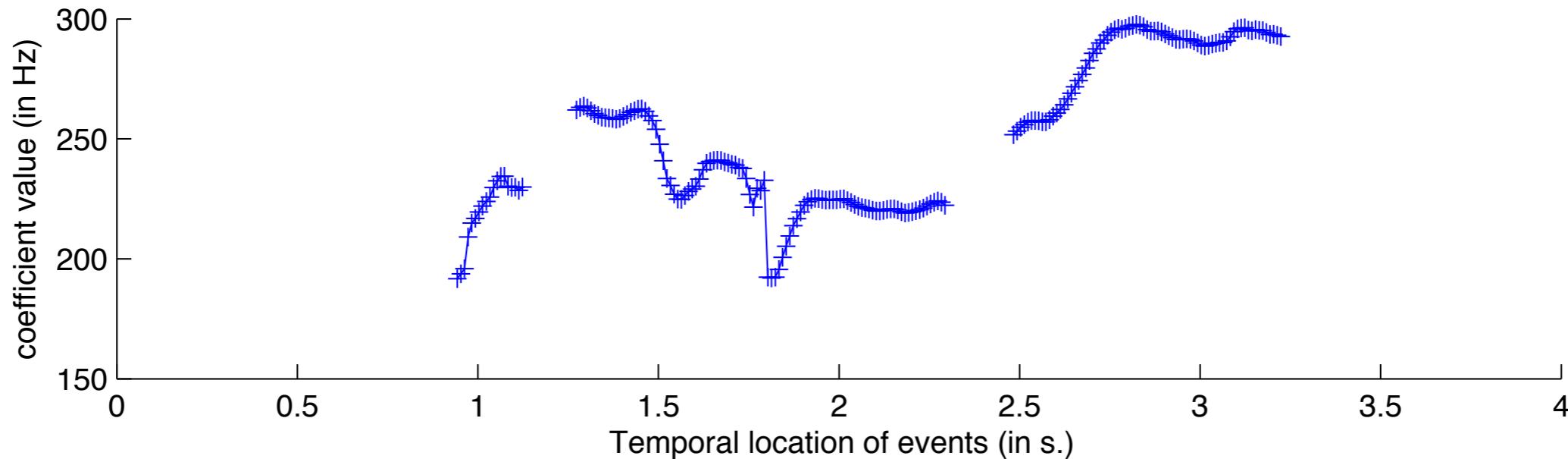
# *aud.pitch* f0 estimation

*p.play*  
*p.save*

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

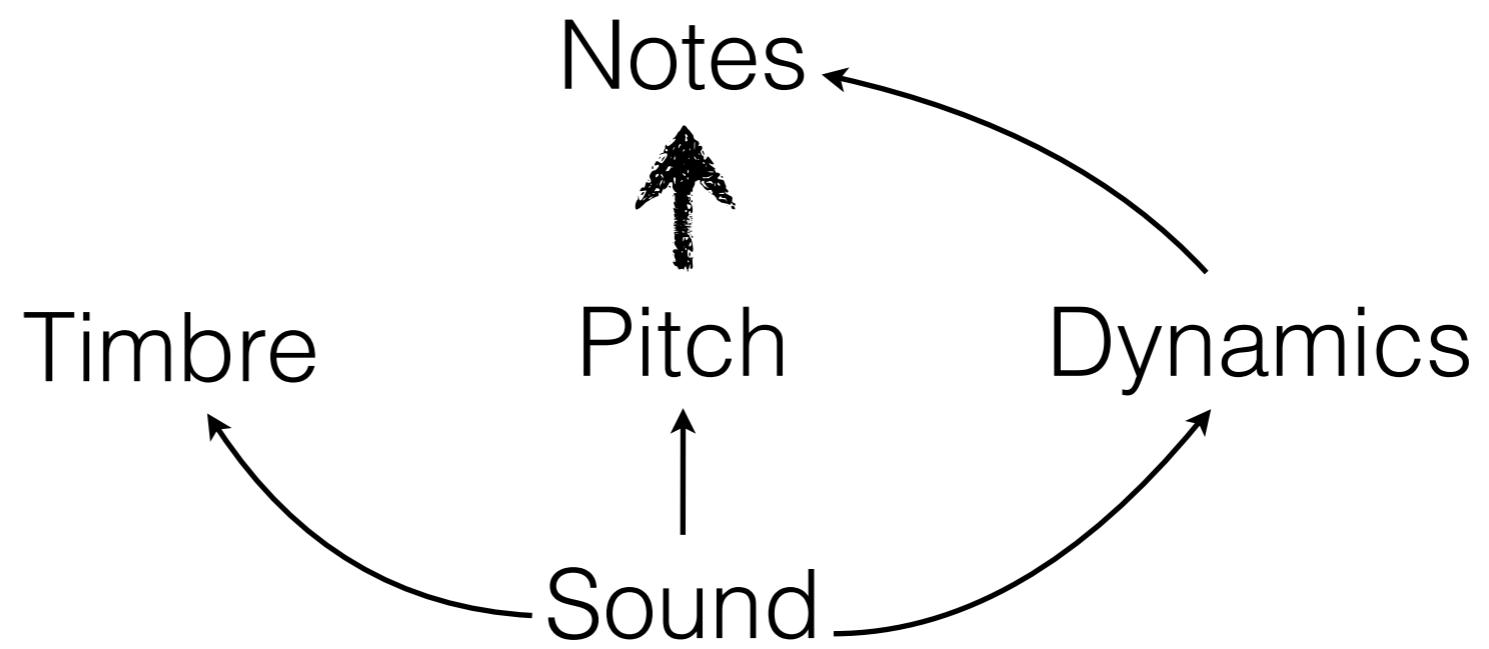


- `aud.pitch(..., 'Frame', 'Mono', 'Max', 400)`

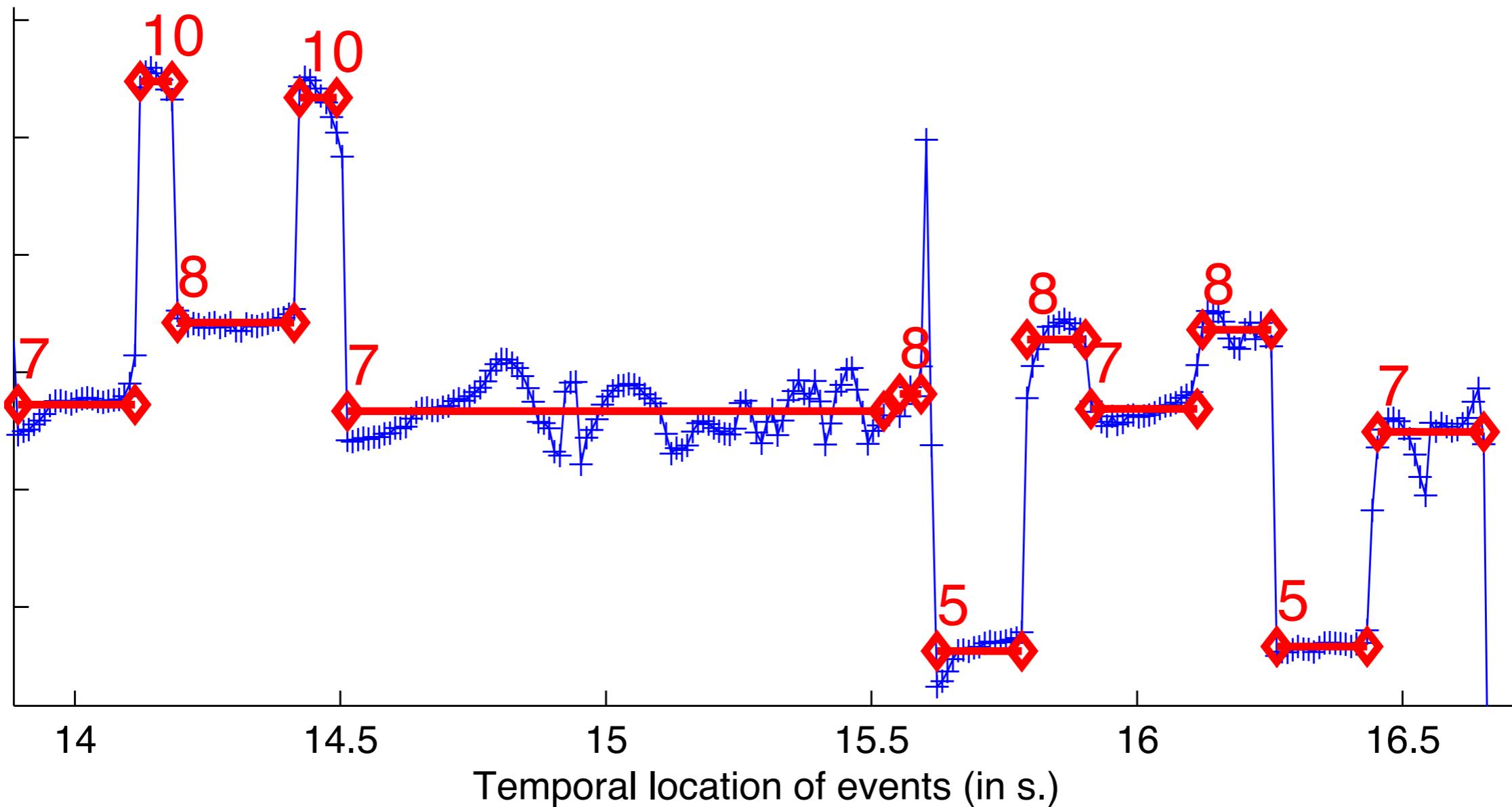


*Symbolic level*

*Audio level*



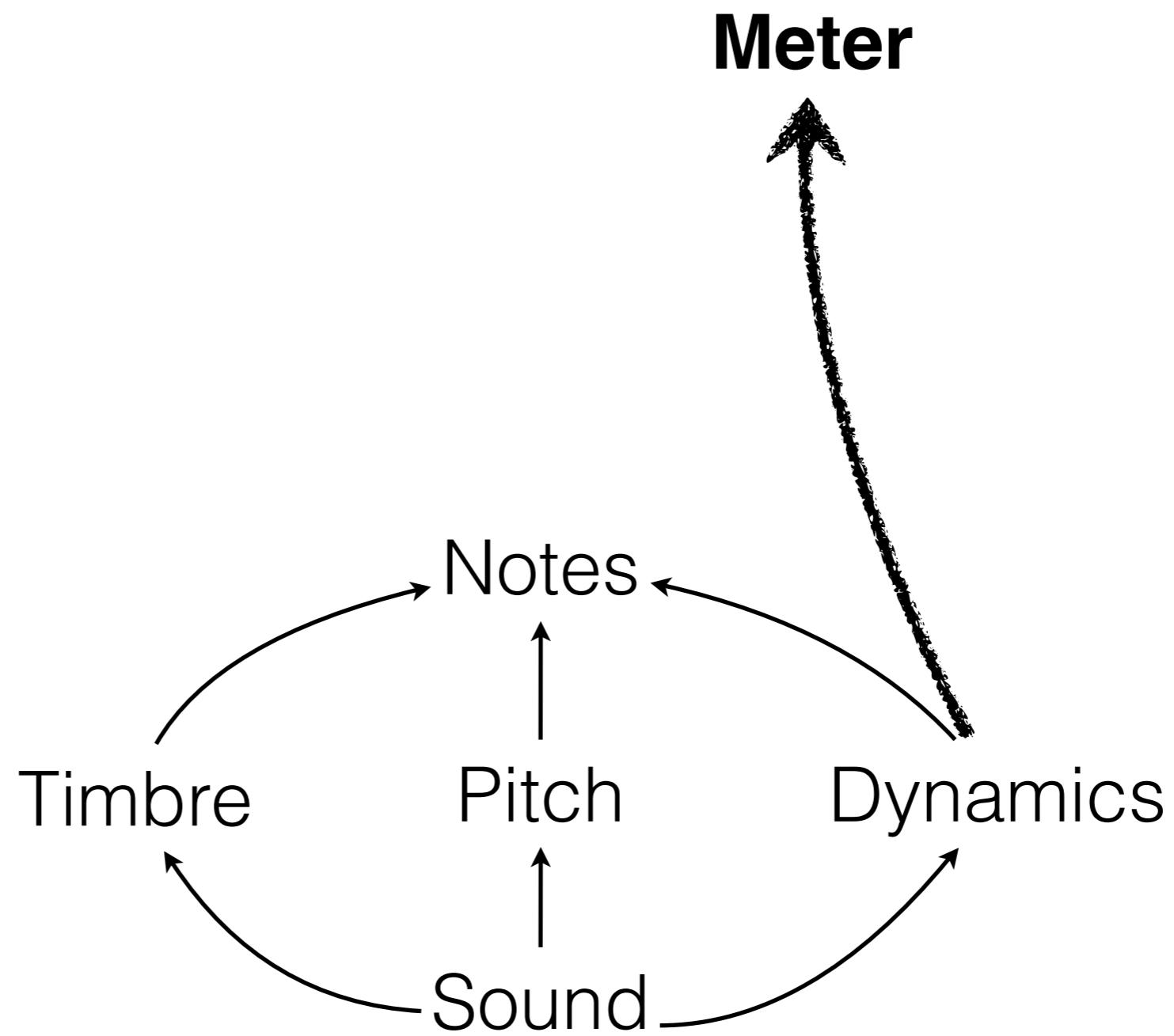
*mus.pitch*  
pitch segmentation and discretisation



*Structural  
levels*

*Symbolic level*

*Audio level*



# *aud.fluctuation*

## rhythmic periodicity along auditory channels

`aud.spectrum`

Bark bands

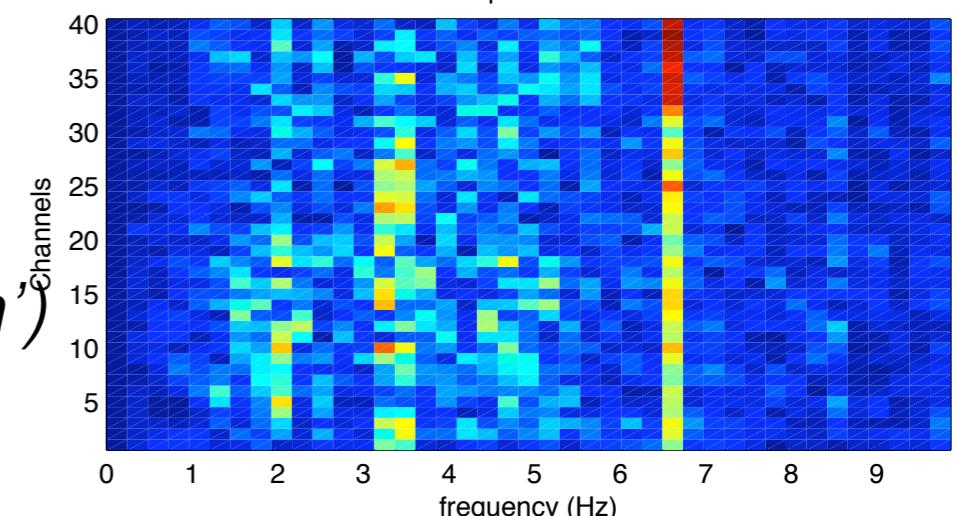
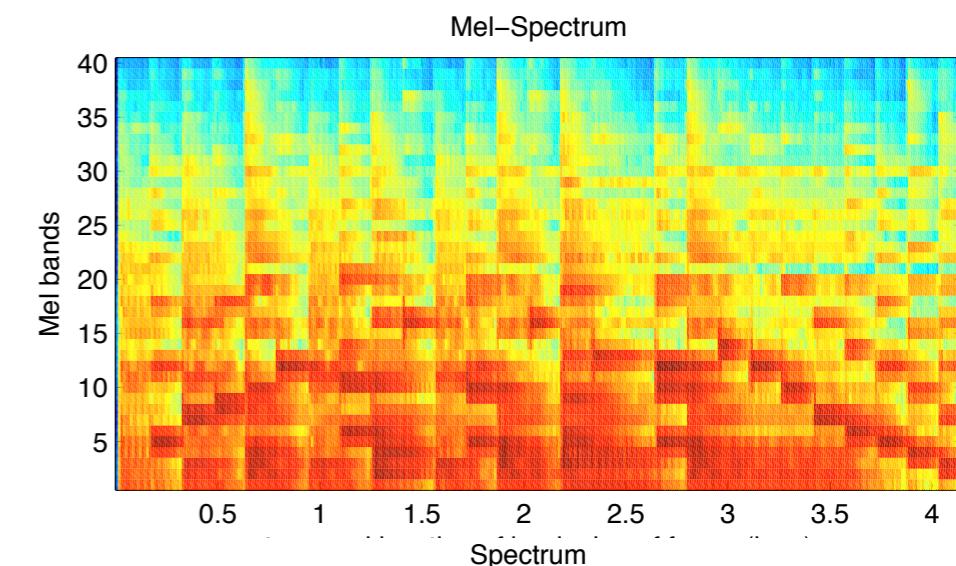
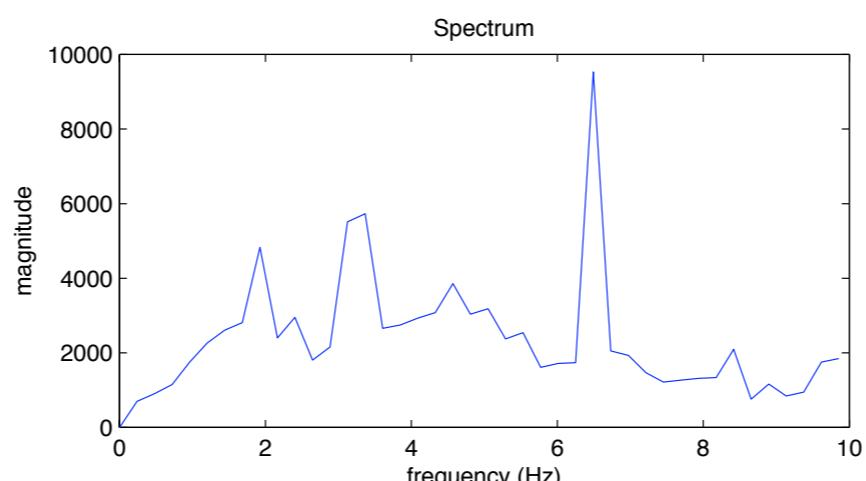
`aud.spectrum`

Bark bands

`sig.sum`

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

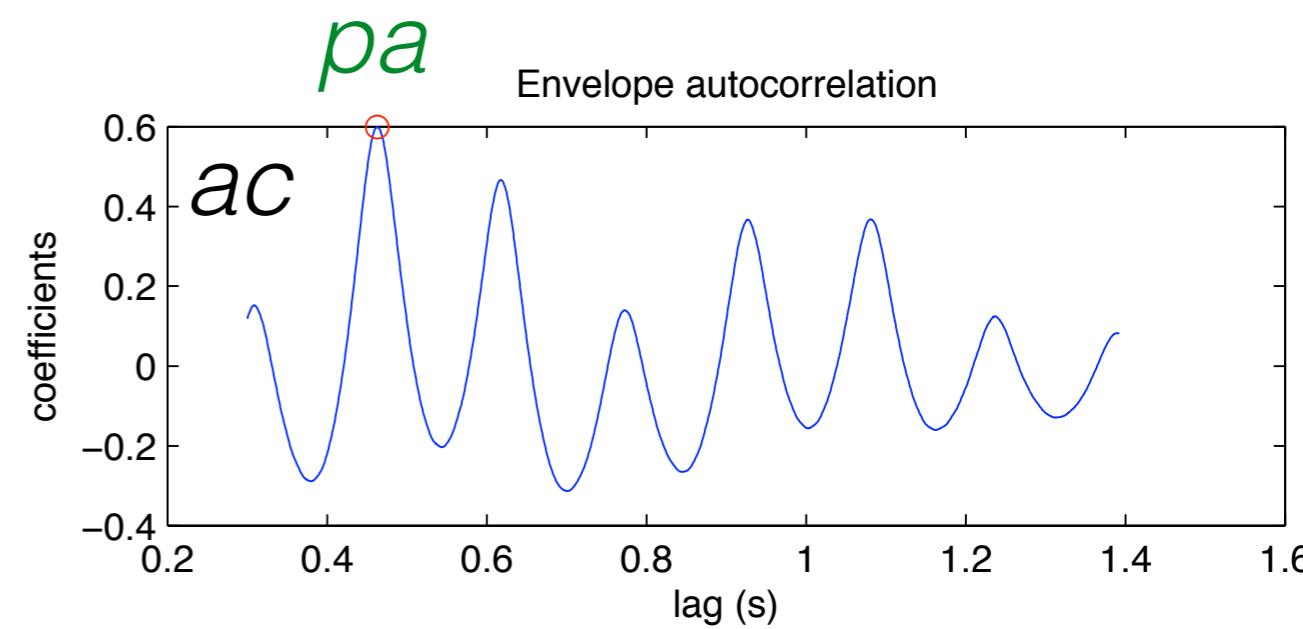
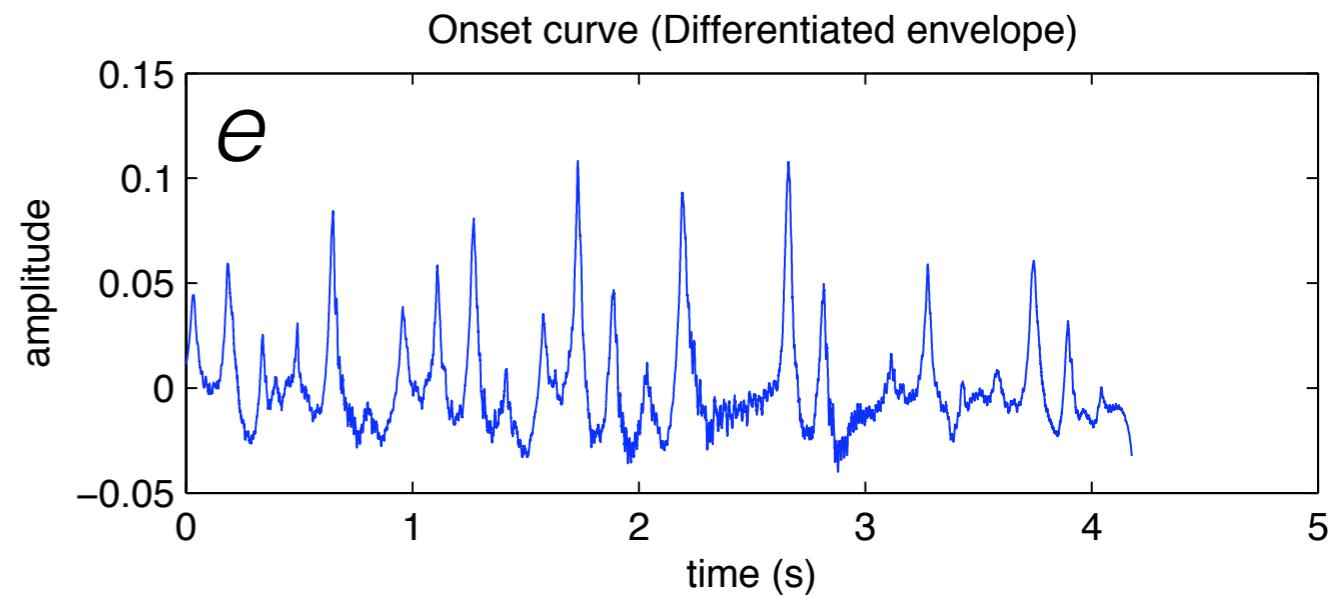
(**'AlongBands'**, *'Max'*, 10,  
*'Window'*, *'No'*,  
**'Resonance'**, **'Fluctuation'**)



# *mus.tempo*

## tempo estimation

- $e = \text{sig.envelope}(\text{'mysong'}, \text{'Diff'})$
- $ac = \text{sig.autocor}(do)$
- $pa = \text{sig.peaks}(ac, \text{'Total'}, 1)$
- $\text{mus.tempo}(pa)$
- **$t = \text{mus.tempo}(\text{'mysong'})$**
- $t = t.\text{eval}$



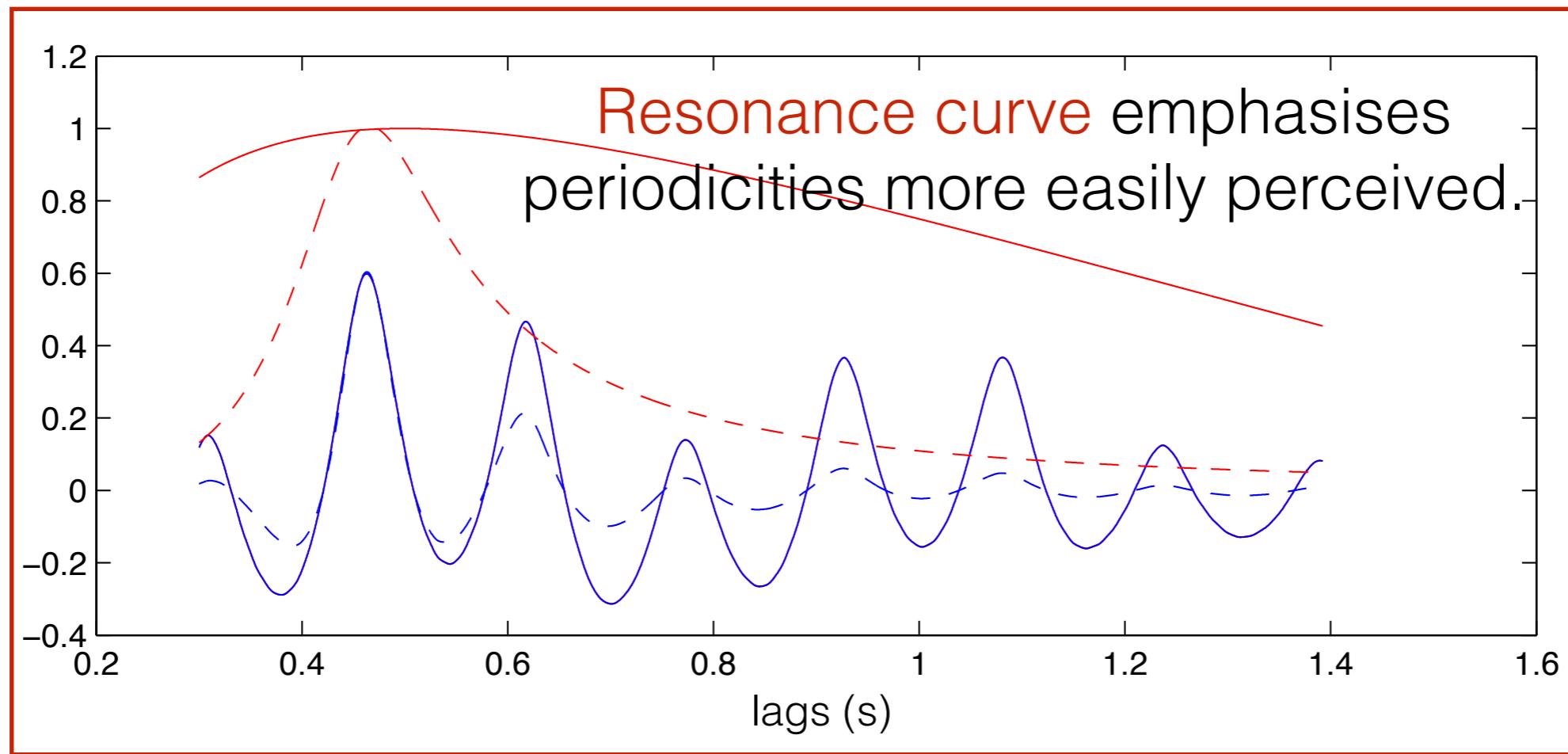
$t = \{\text{sig.signal}, \text{sig.Autocor}\}$

tempo: 129.6333 bpm

# *mus.tempo*

## tempo estimation

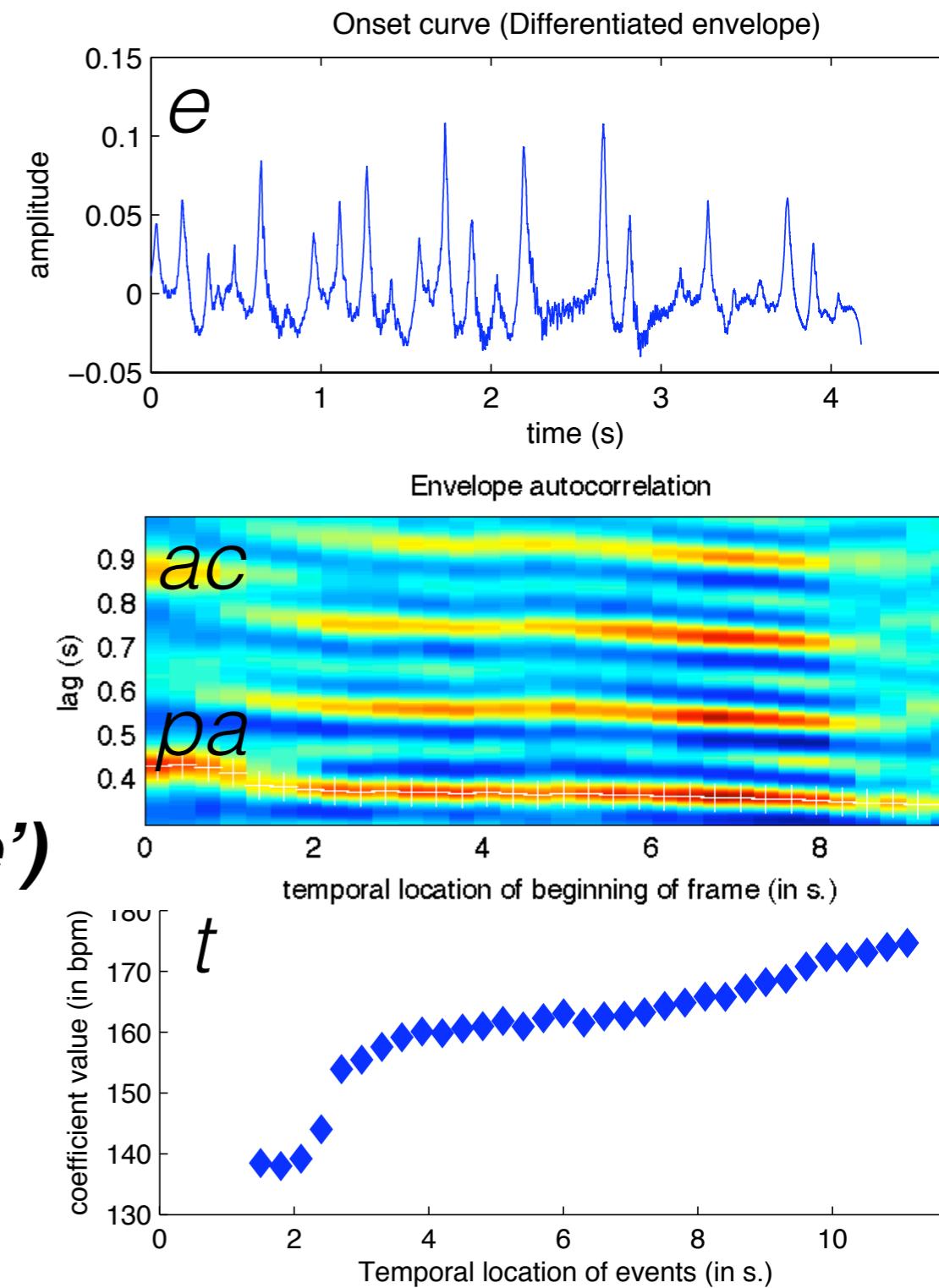
- $e = \text{sig.envelope}(\text{'mysong'}, \text{'Diff'})$
- $ac = \text{aud.autocor}(do, \text{'Resonance'})$



# *mus.tempo*

## tempo estimation

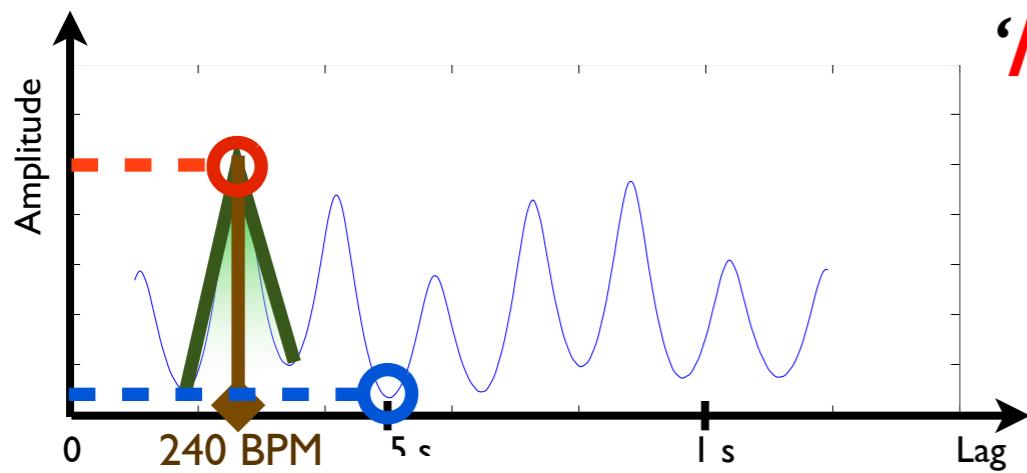
- $e = \text{sig.envelope}(\text{'mysong'}, \text{'Diff'})$
  - $ac = \text{aud.autocor}(do, \text{'Frame'}, \text{'Resonance'})$
  - $pa = \text{sig.peaks}(ac, \text{'Total'}, 1)$
  - $\text{mus.tempo}(pa)$
  - $t = \text{mus.tempo}(\text{'mysong'}, \text{'Frame'})$
  - $t = t.\text{eval}$
- $t = \{\text{sig.signal}, \text{sig.Autocor}\}$



# *mus.pulseclarity*

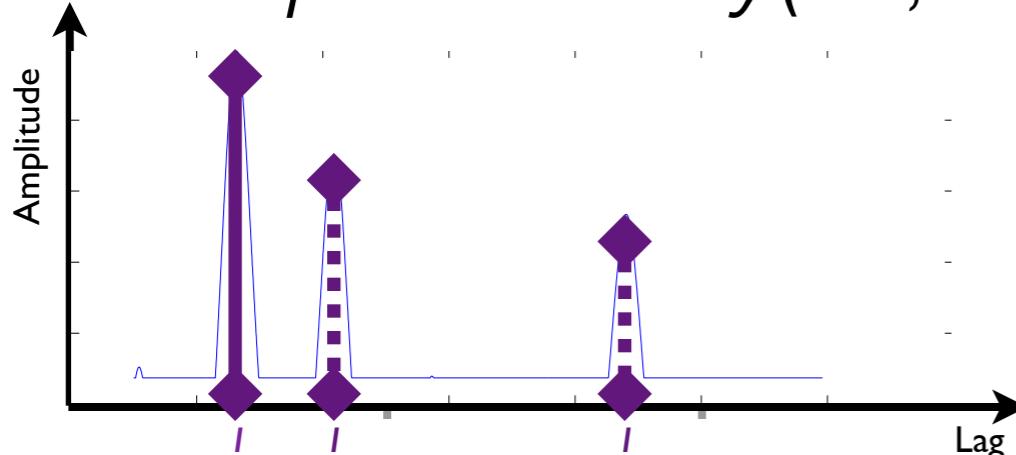
## rhythmic clarity, beat strength

*mus.pulseclarity(..., 'Enhanced', 'No')*



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

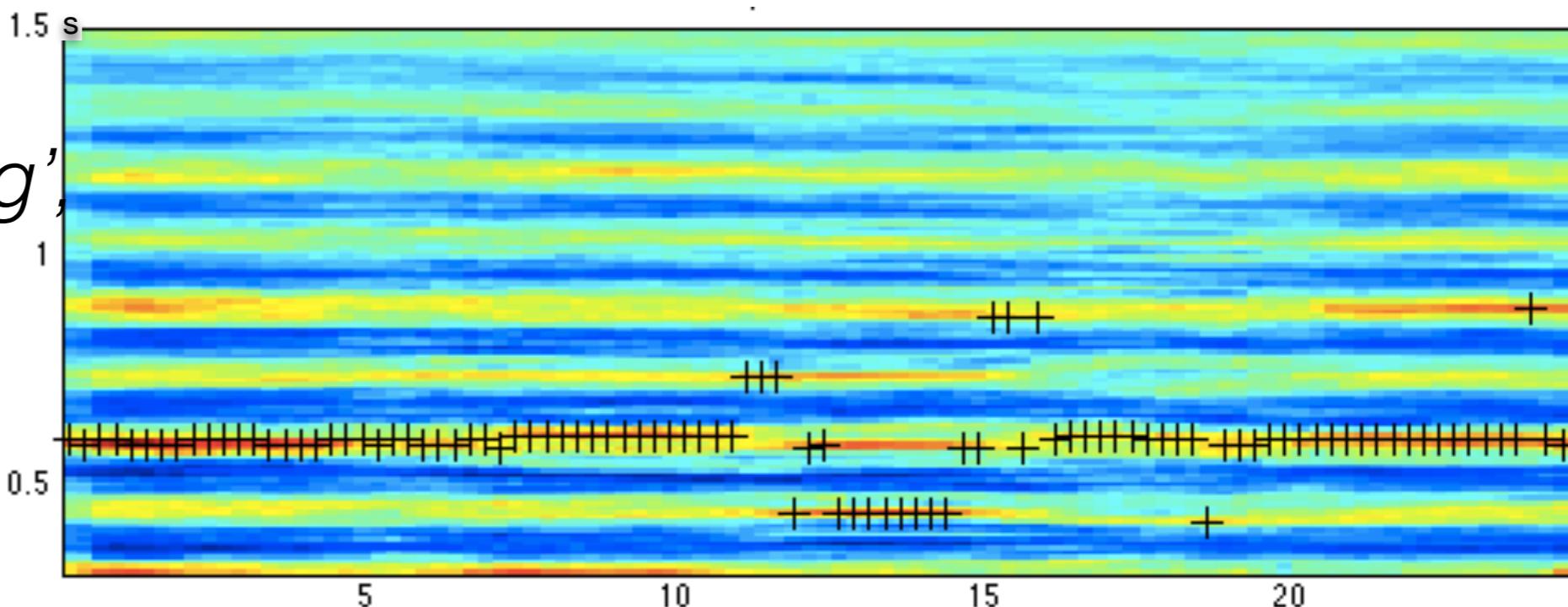
*mus.pulseclarity(..., 'Enhanced', 'Yes')*



'EntropyAutocor'  
'InterfAutocor'

Lartillot et al., ISMIR 2008

`mus.tempo('mysong',  
'Frame')`



Pulsation does not always focus on one single metrical level.

`mus.metre` tracks all metrical levels in parallel.

Metrical Hierarchy

Metrical levels:

8

7

2

6

5

4

3

1

0.5

0

Pulsation periods [in s.]

Dominant levels

2.5

5

10

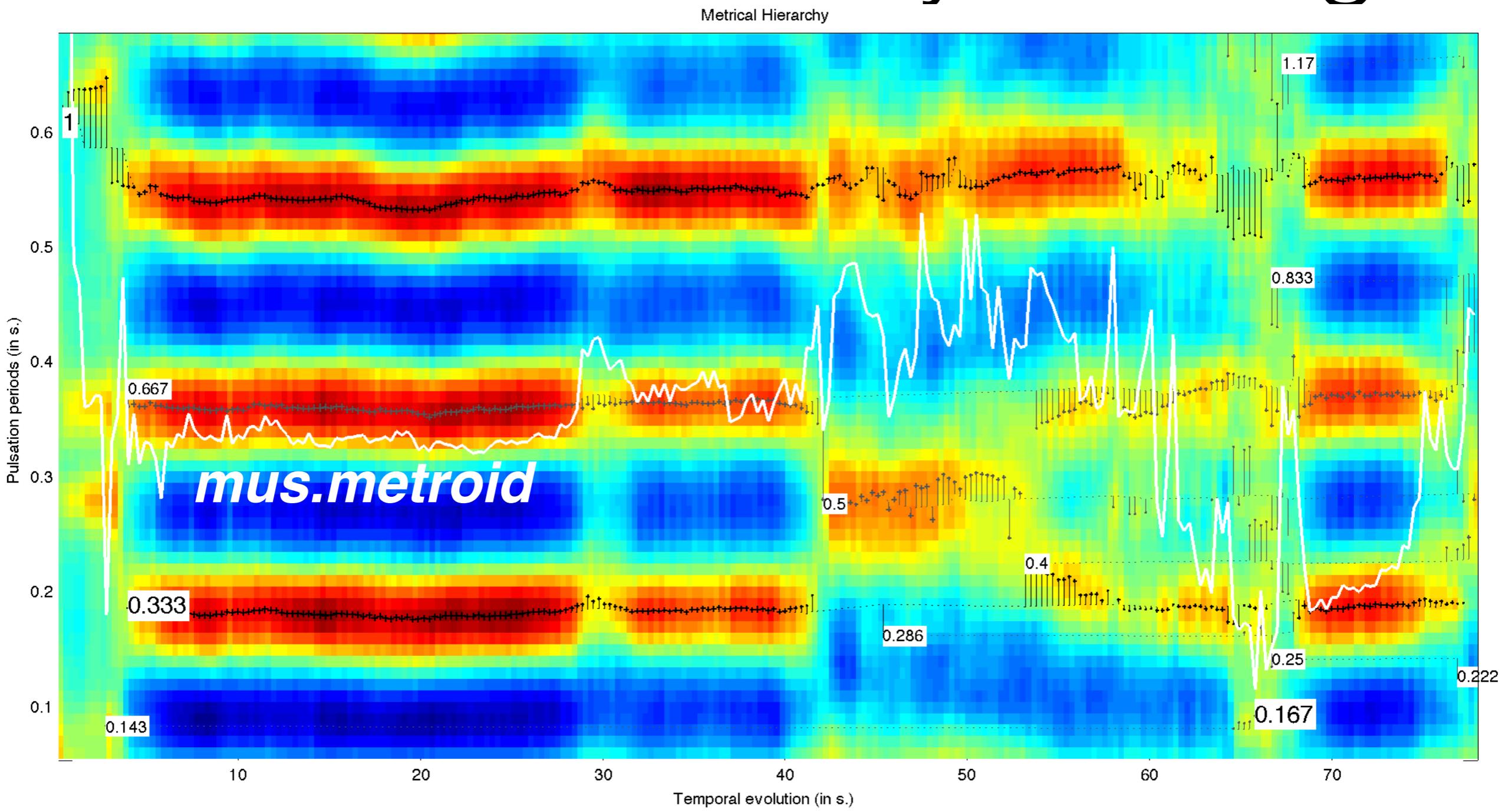
15

20

Temporal evolution (in s.)

# *mus.metre*

## metrical hierarchy tracking

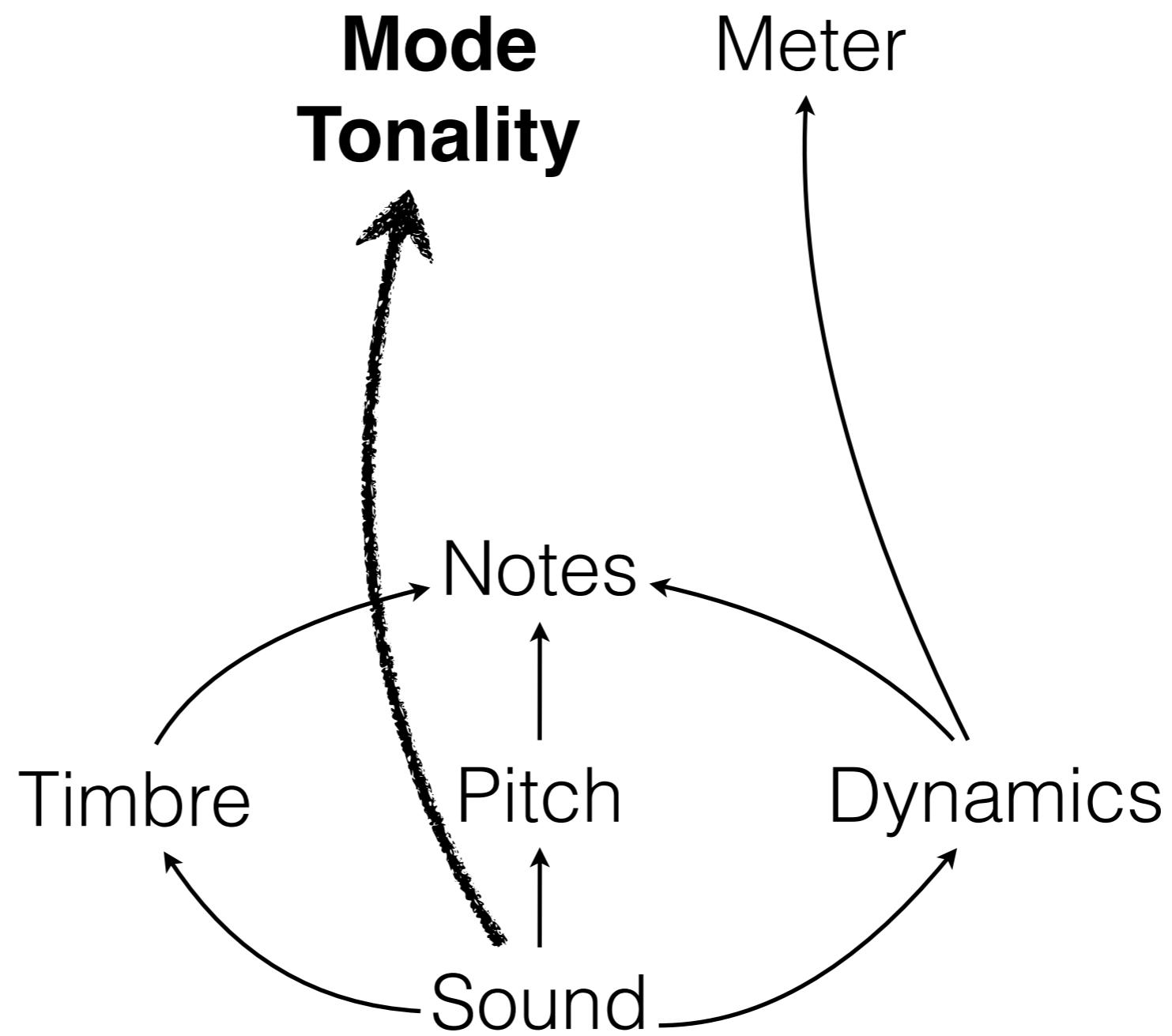


Beethoven, 9th Symphony, Scherzo

*Structural  
levels*

*Symbolic level*

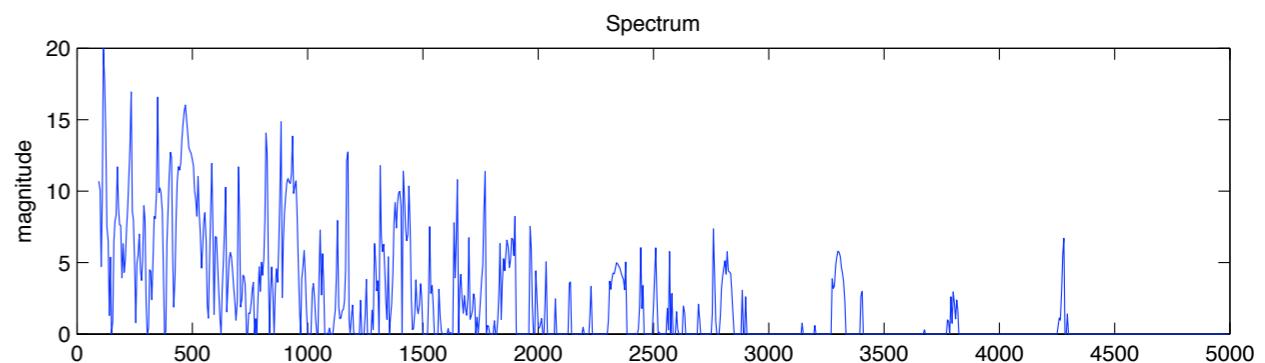
*Audio level*



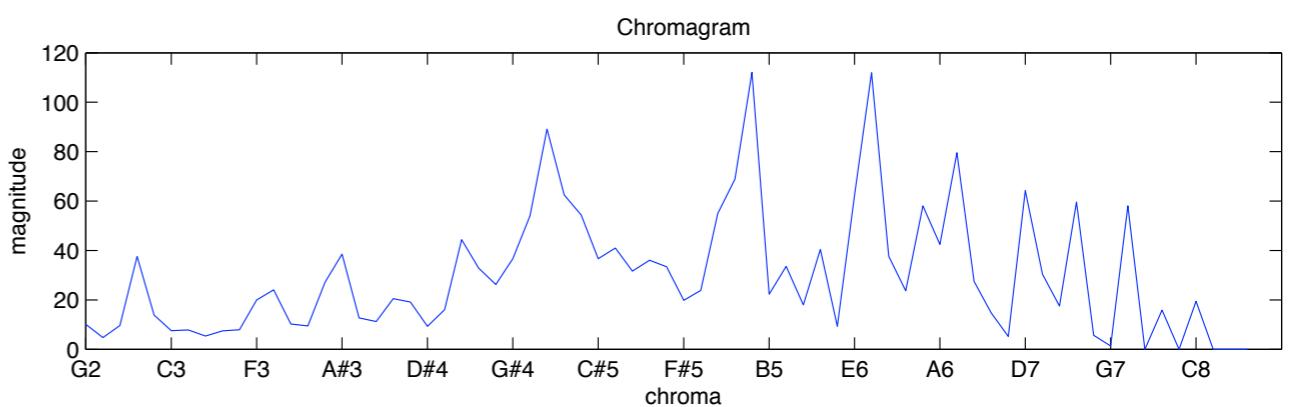
# *mus.chromagram*

## energy distribution along pitches

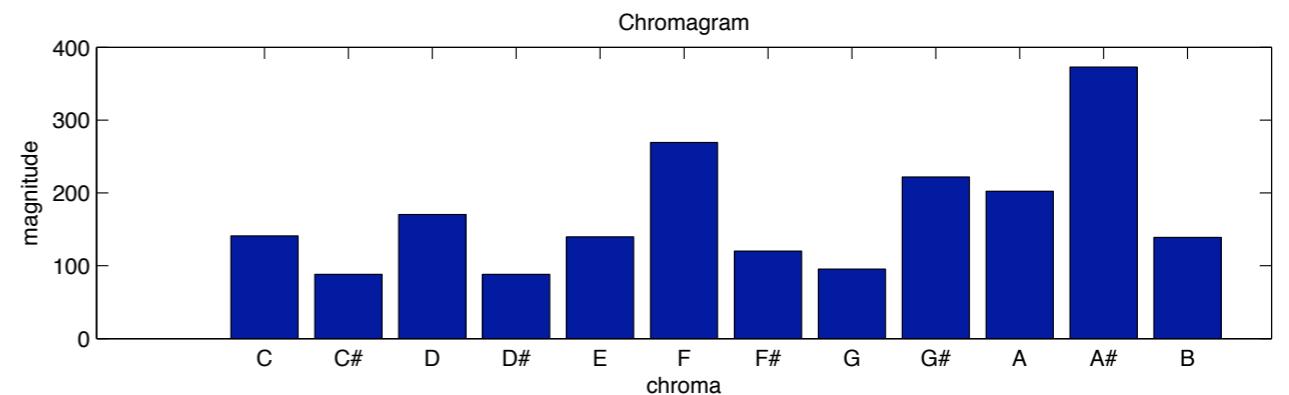
- $s = \text{sig.spectrum}(a, \text{'dB'}, 20, \text{'Min'}, 100, \text{'Max'}, 6400)$



- $C = \text{mus.chromagram}(s, \text{'Wrap'}, \text{'no'})$



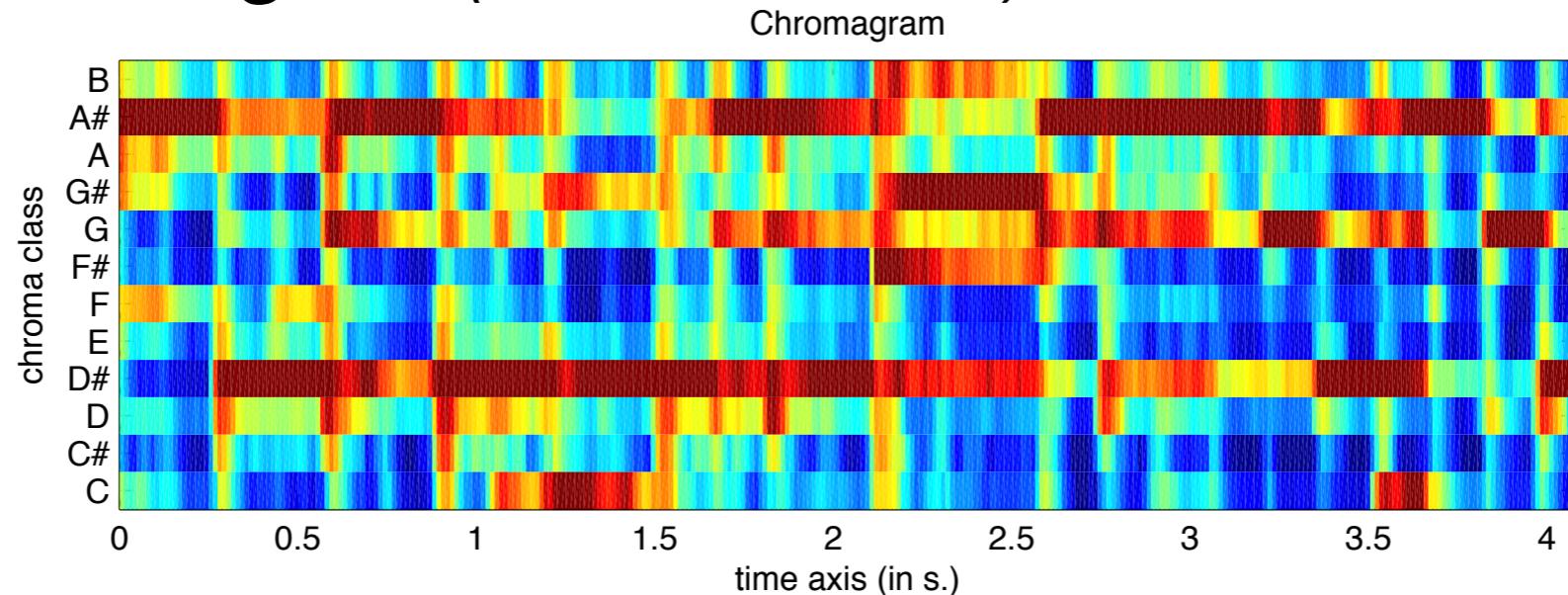
- $C = \text{mus.chromagram}(c, \text{'Wrap'}, \text{'yes'})$



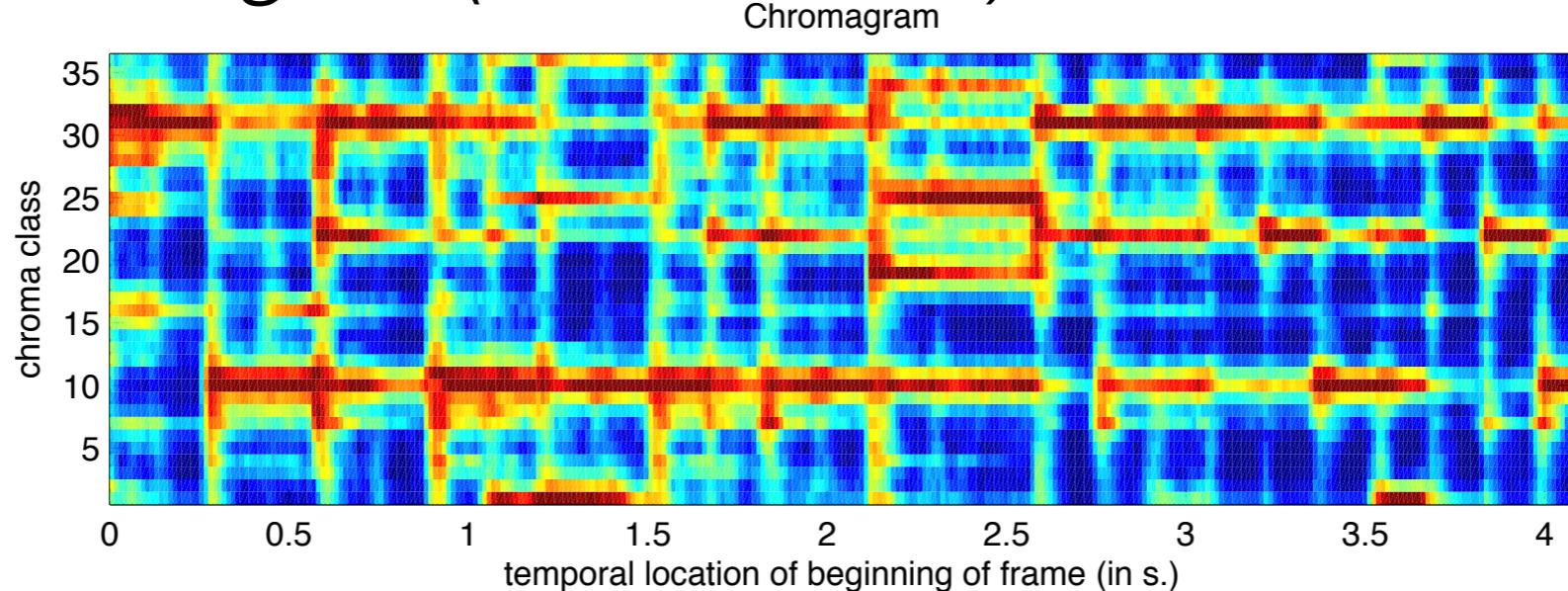
# *mus.chromagram*

## chroma resolution

- *mus.chromagram(..., 'Res', 12)*



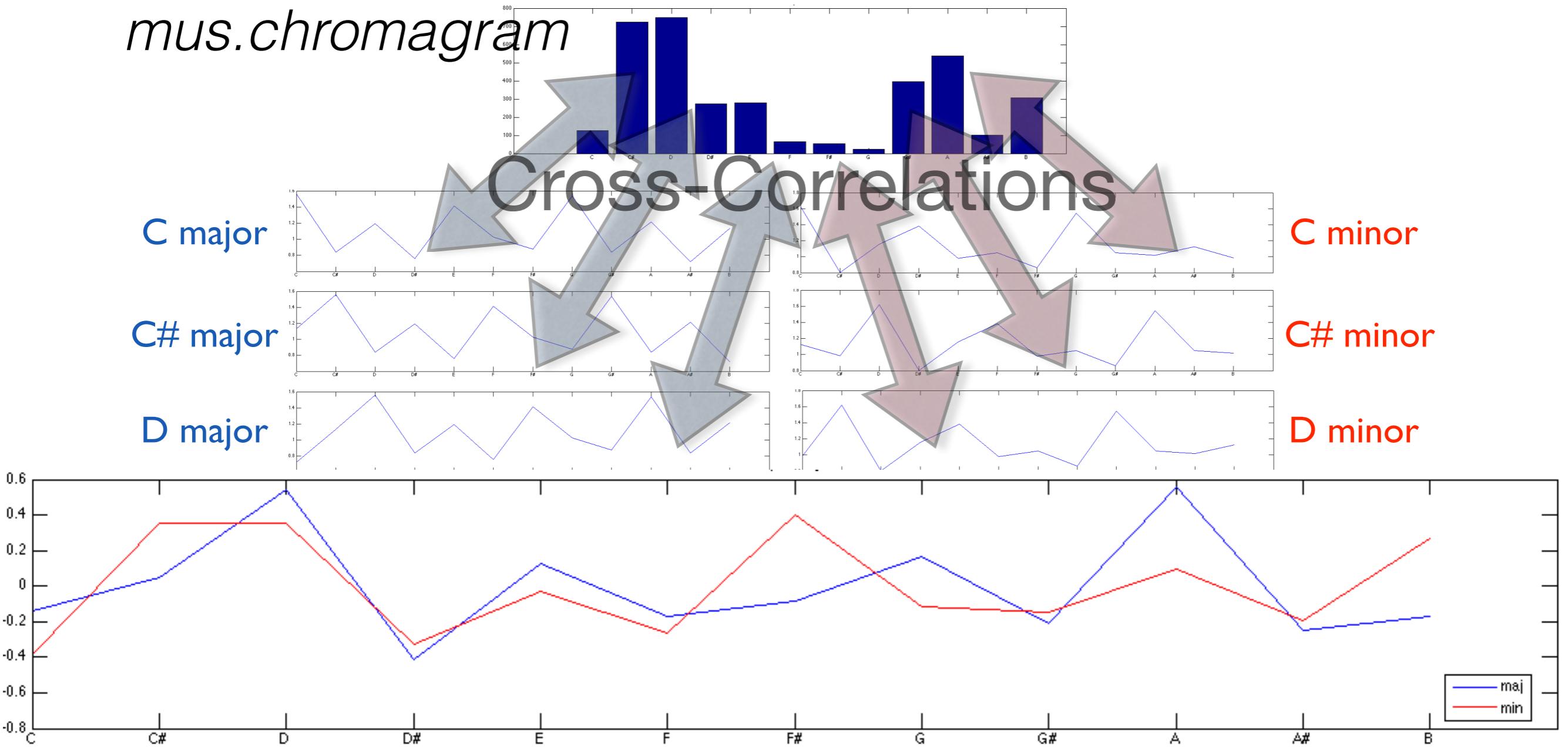
- *mus.chromagram(..., 'Res', 36)*



# *mus.keystrength*

## probability of key candidates

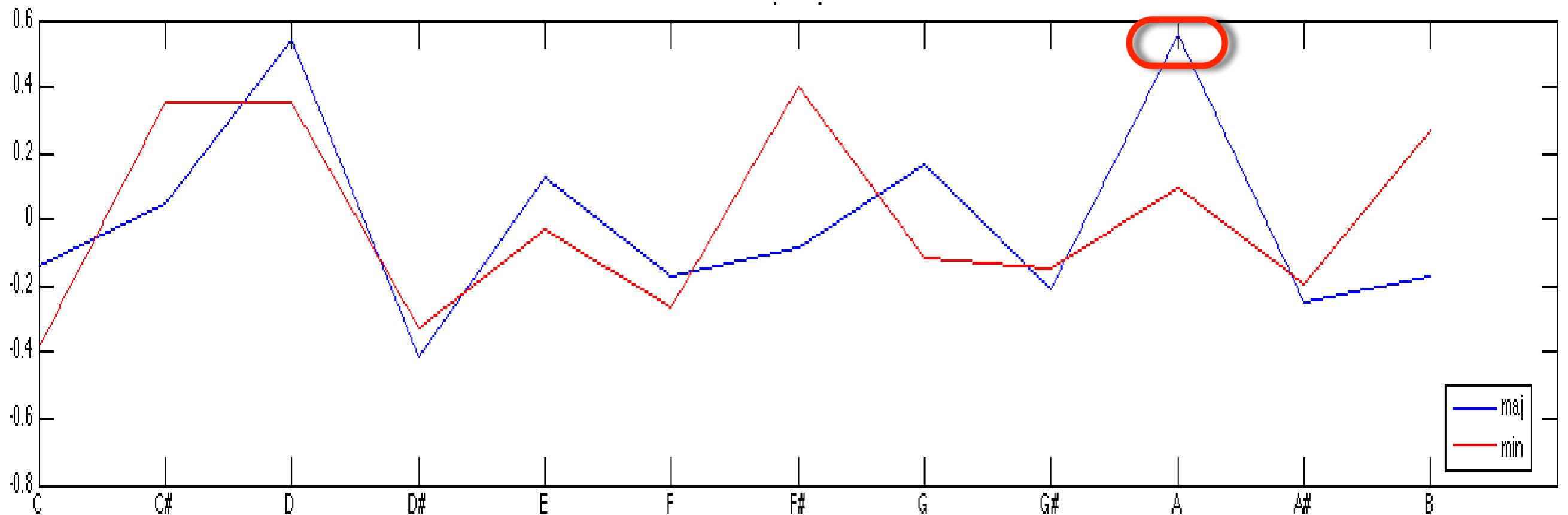
*mus.chromagram*



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(...))*

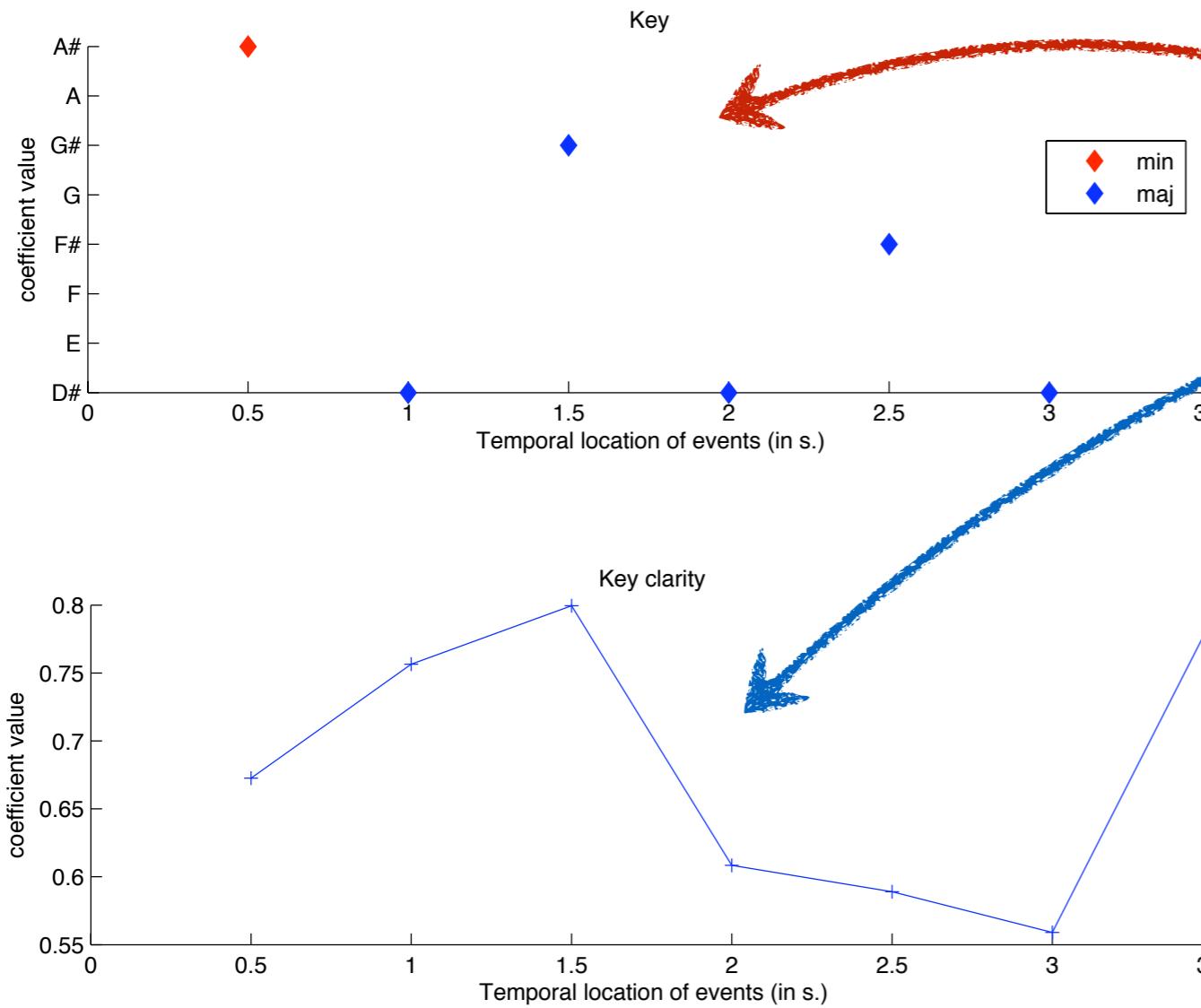
- *mus.key(..., 'Total', 1)*

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*

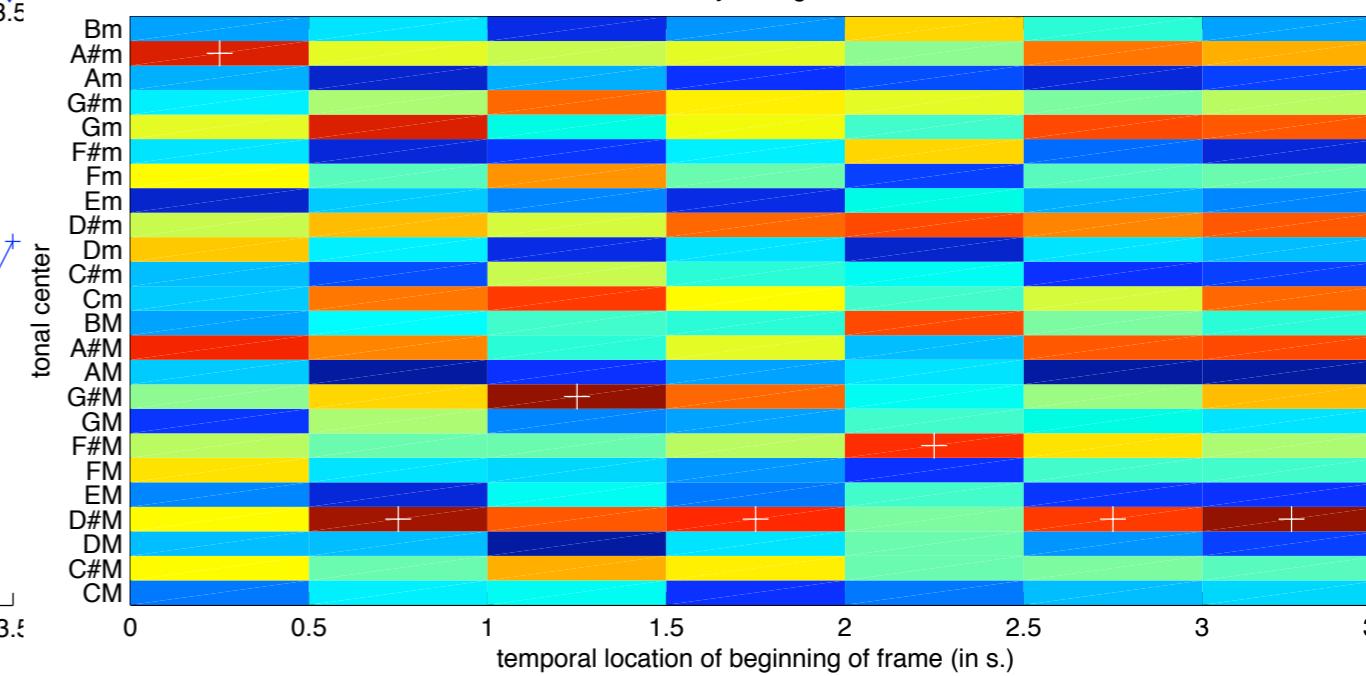
## key estimation

```
• k = mus.key('mysong', 'Frame')
```



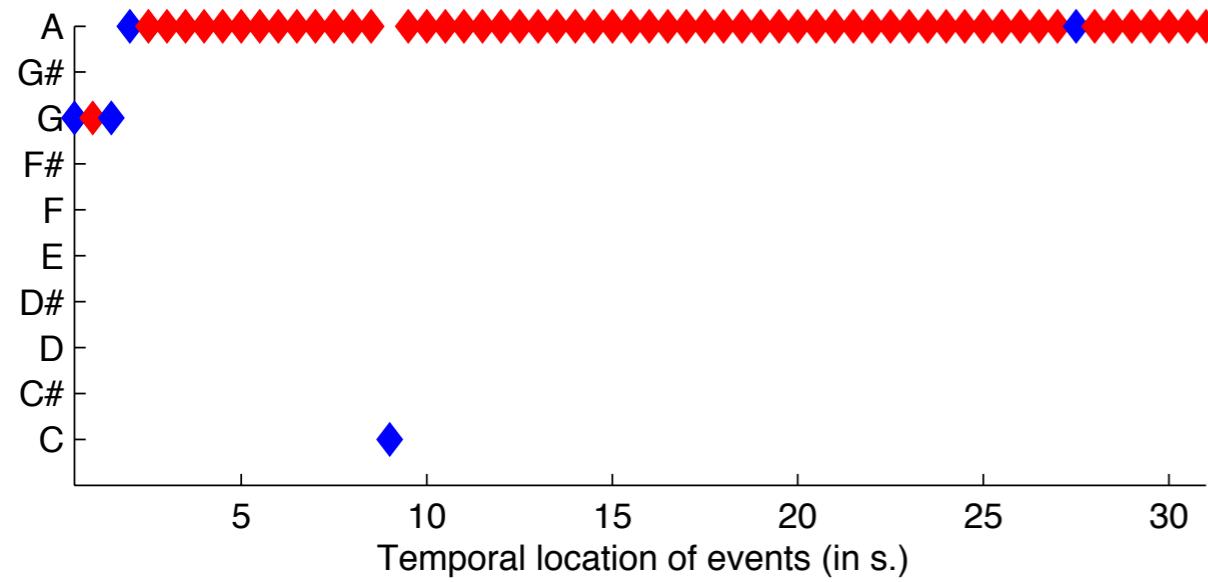
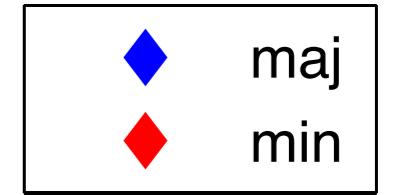
- $k = k.eval$

~~k = {sig.signal, sig.signal, mus.Keystrength}~~

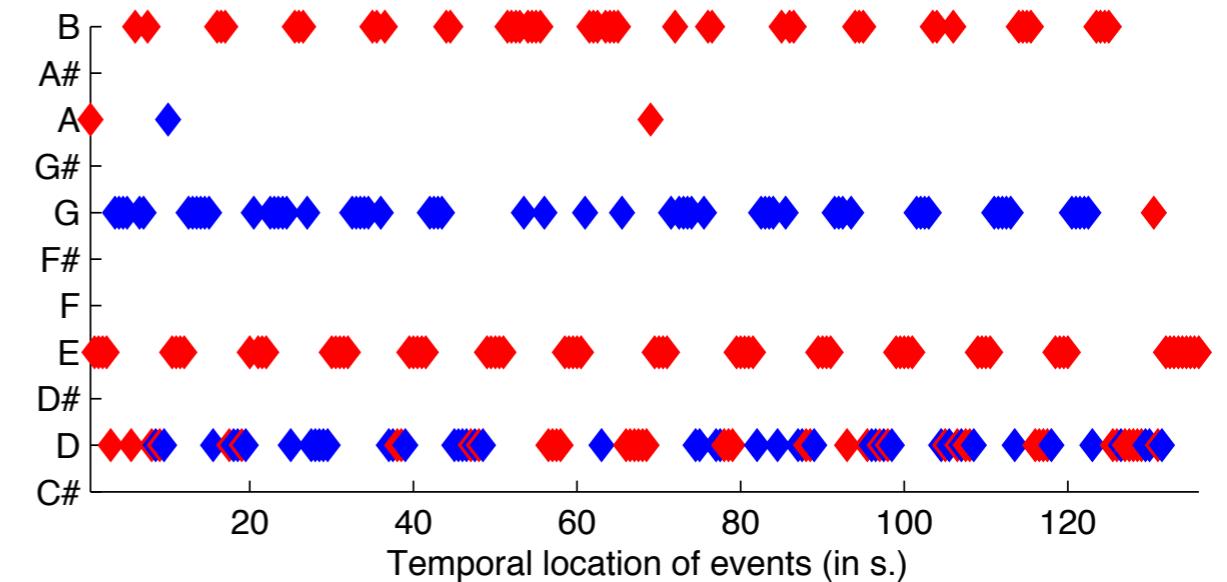


# *mus.key*

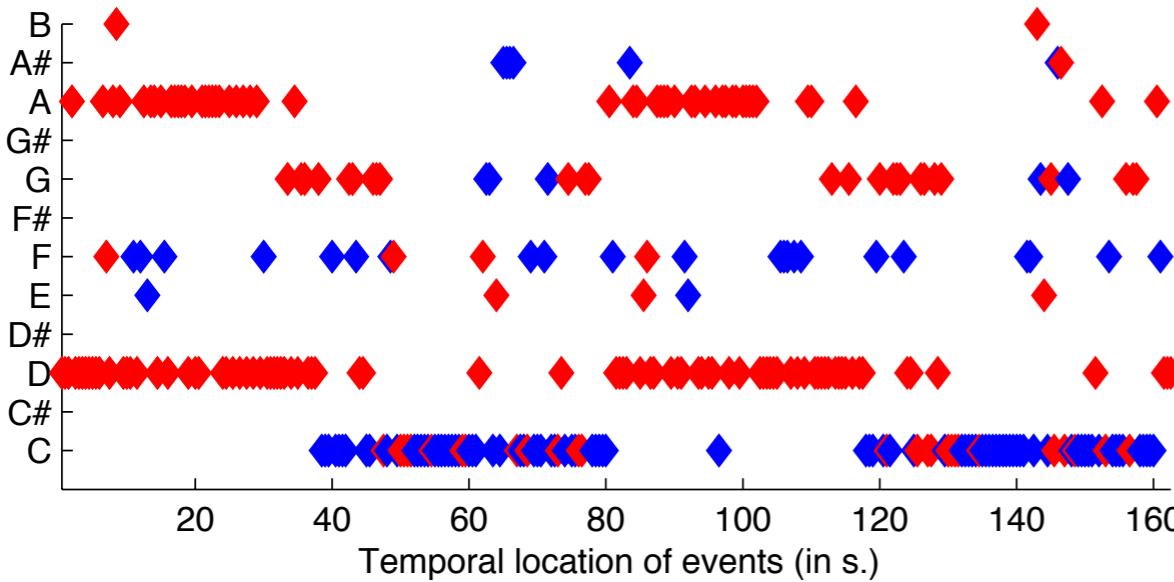
## key estimation



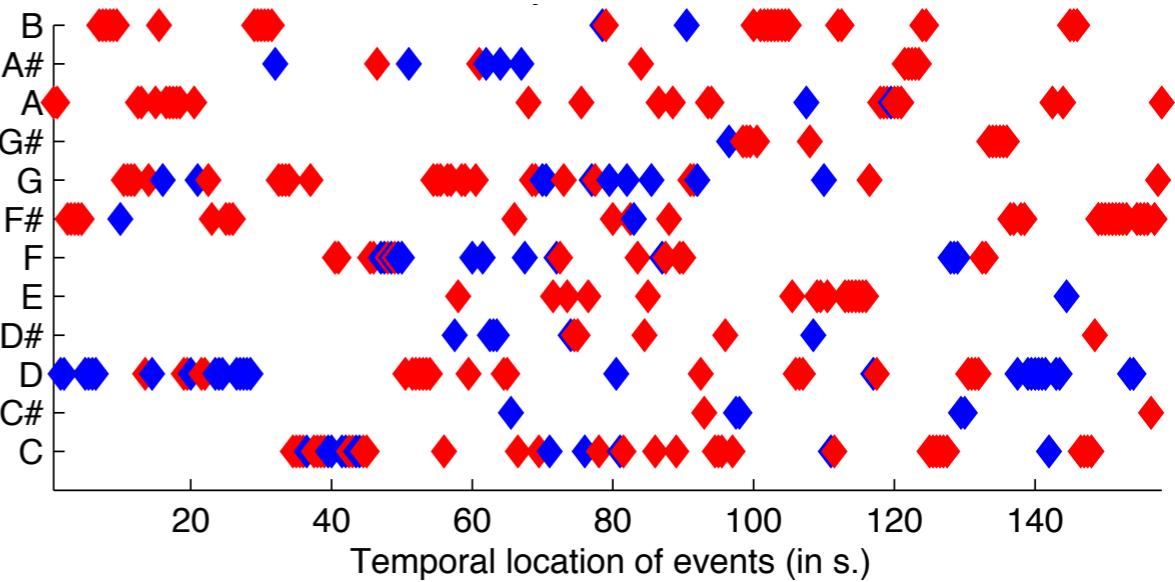
Monteverdi, *Hor che'l ciel e la terra*, 1st part



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



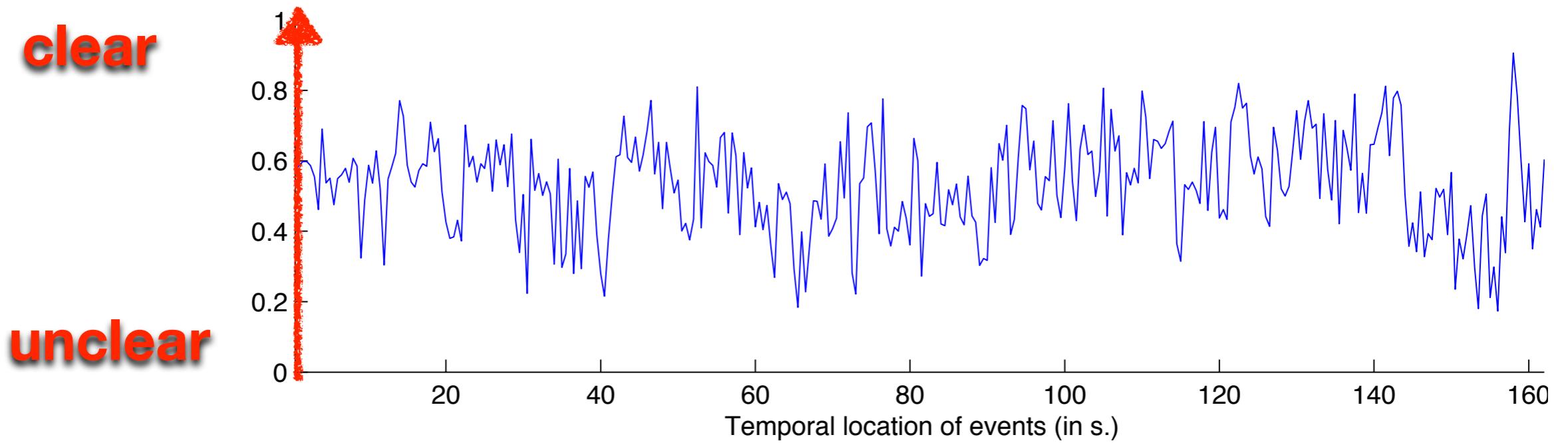
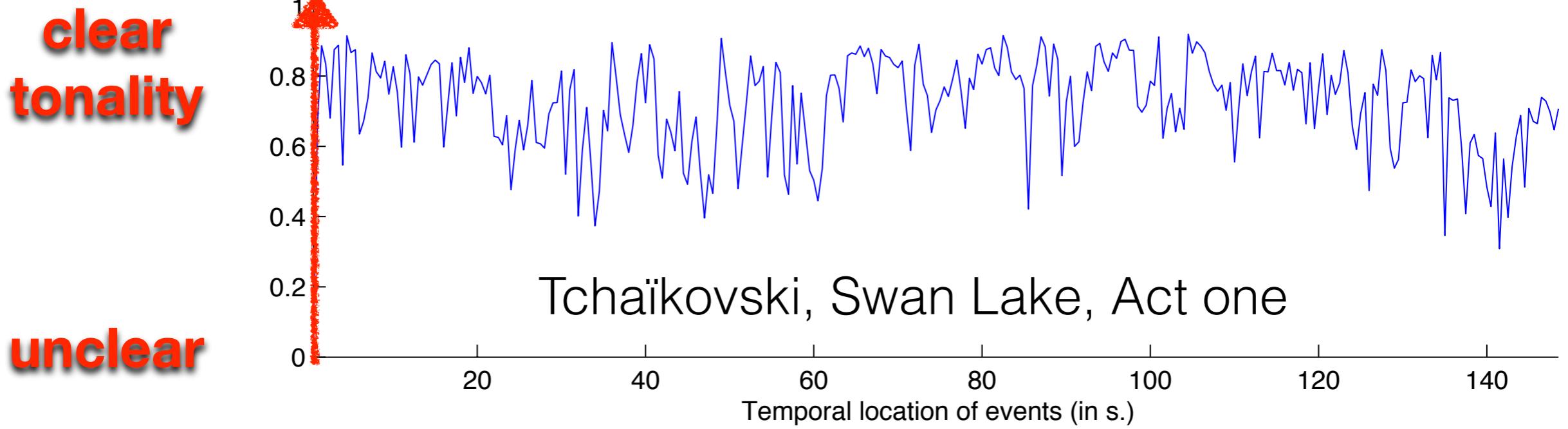
Beethoven, 9th Symphony, Scherzo



Schönberg, *Verklärte Nacht, Sehr Ruhig*

# $[k \ c] = mus.key$

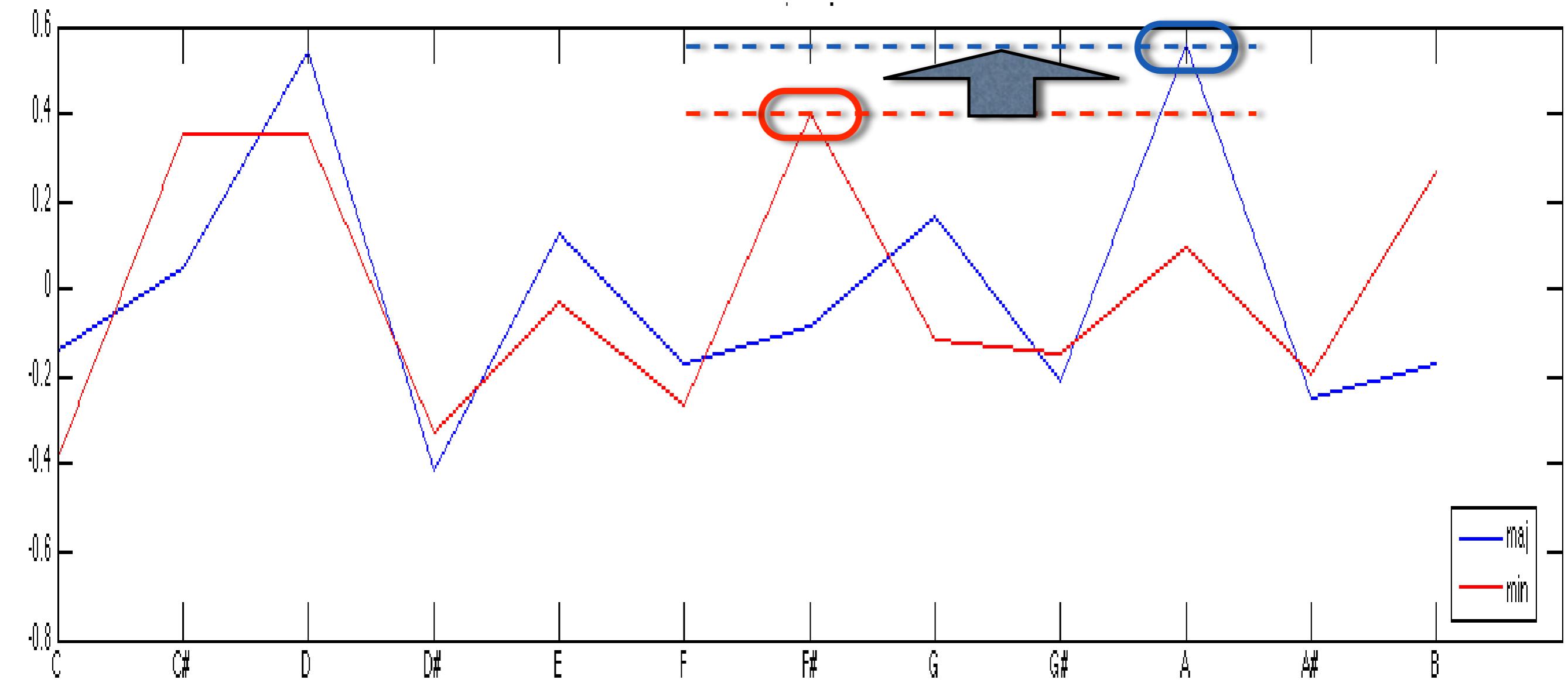
## key clarity



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

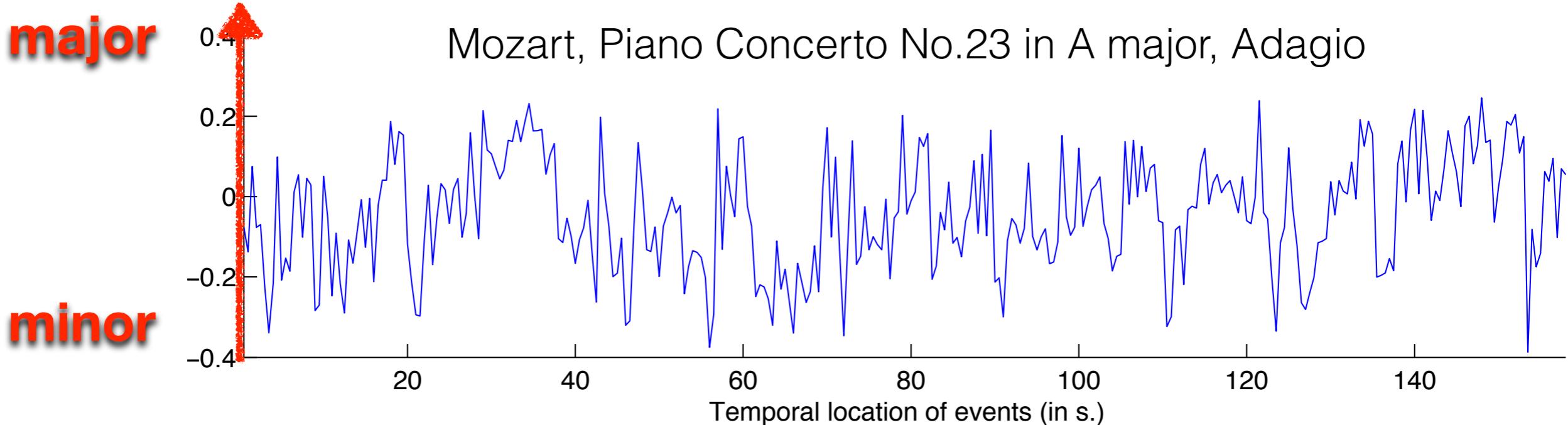
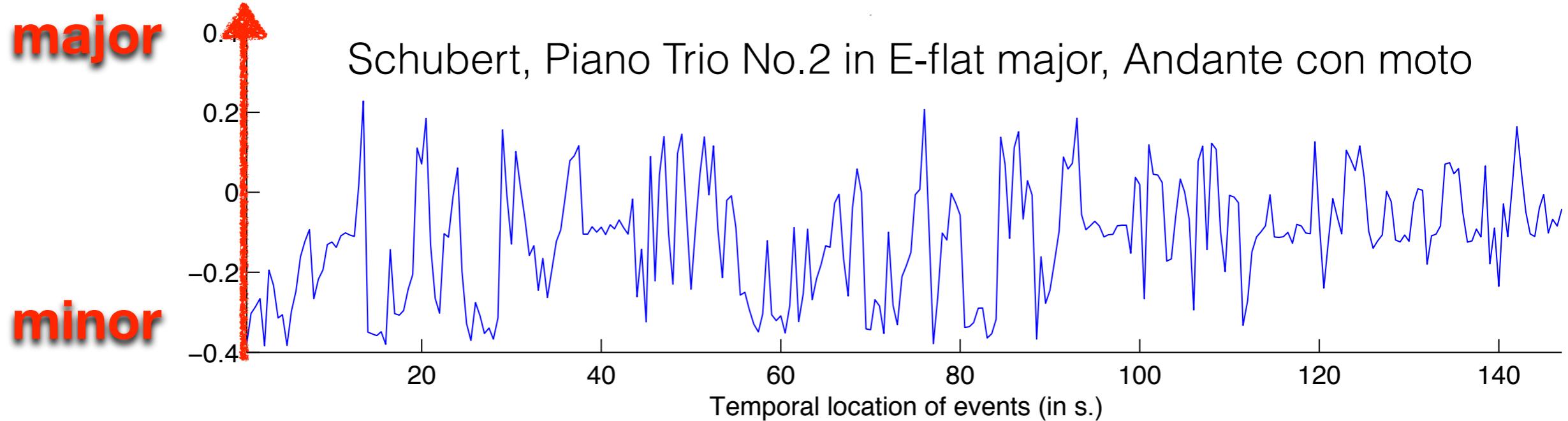
# *mus.mode*

## mode estimation



# *mus.mode*

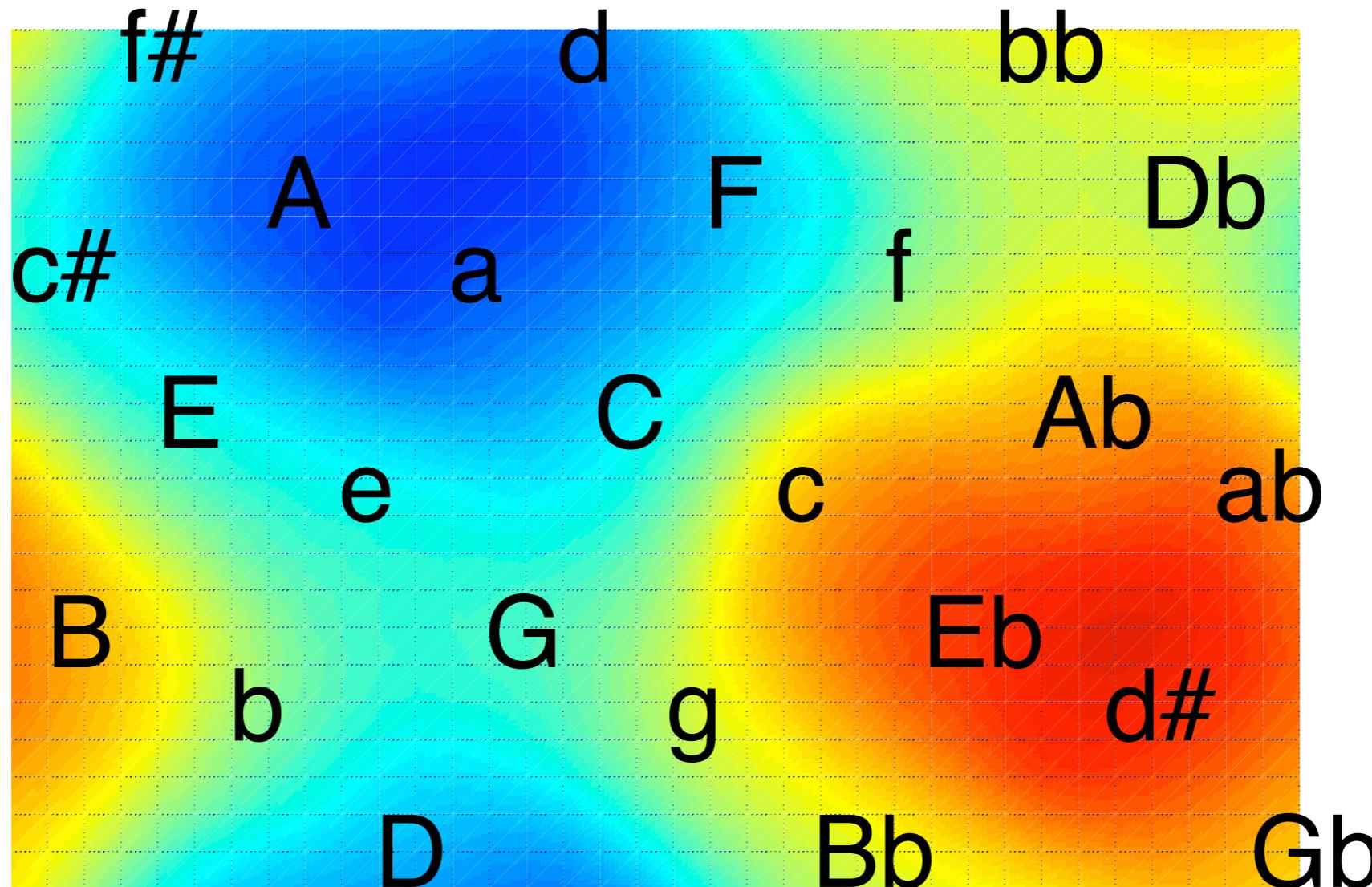
## mode estimation



# *mus.keysom*

## self-organizing map

Self-organizing map projection of chromagram

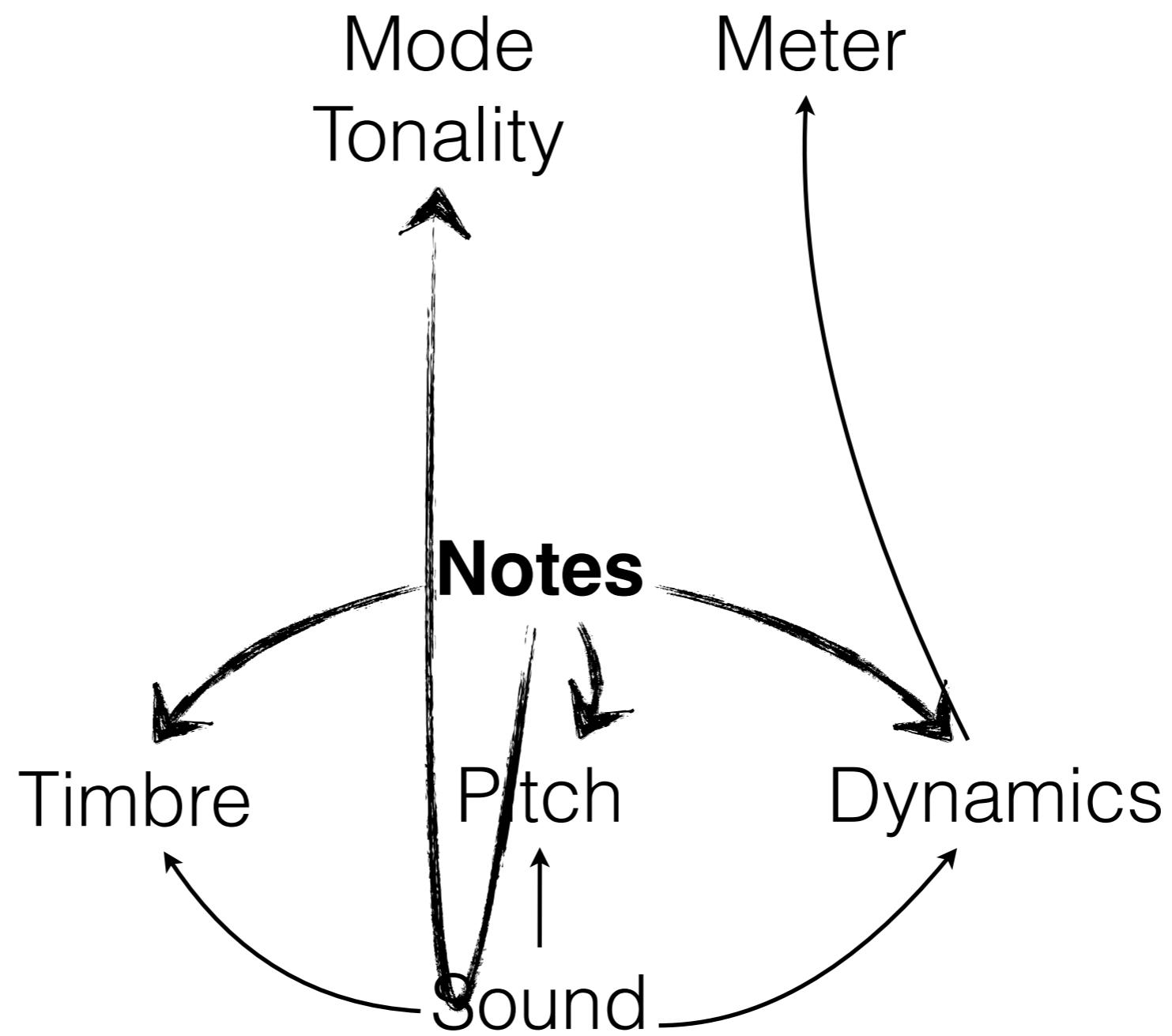


Toiviainen & Krumhansl, “Measuring and modeling real-time responses to music: The dynamics of tonality induction”, Perception 32-6, pp. 741–766, 2003.

*Structural  
levels*

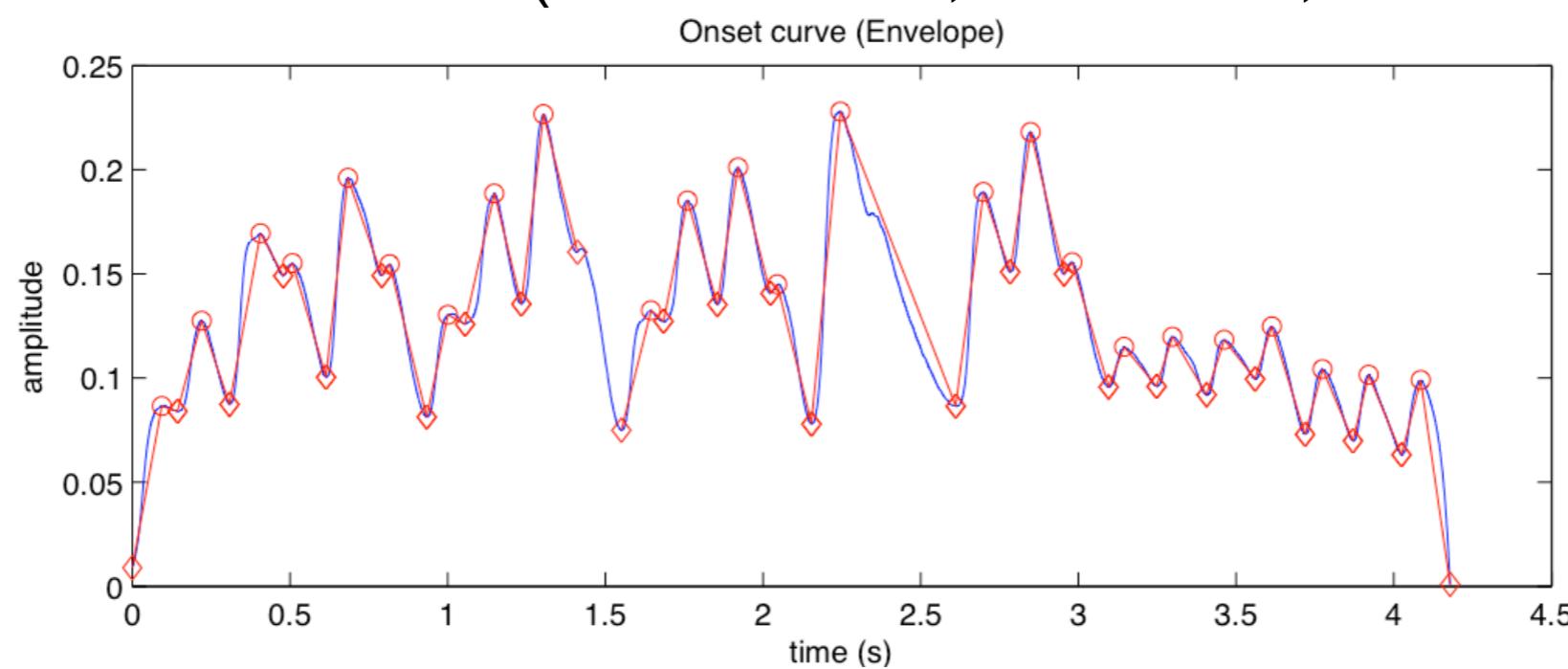
*Symbolic level*

*Audio level*

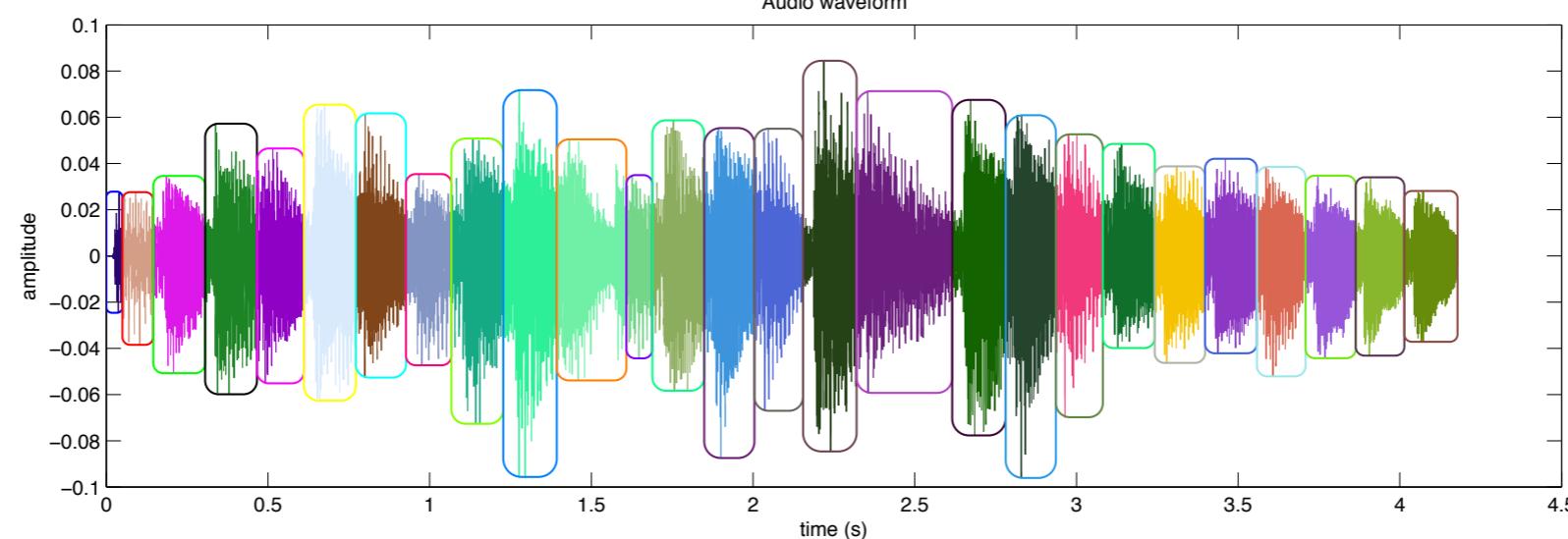


# onset-based segmentation

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

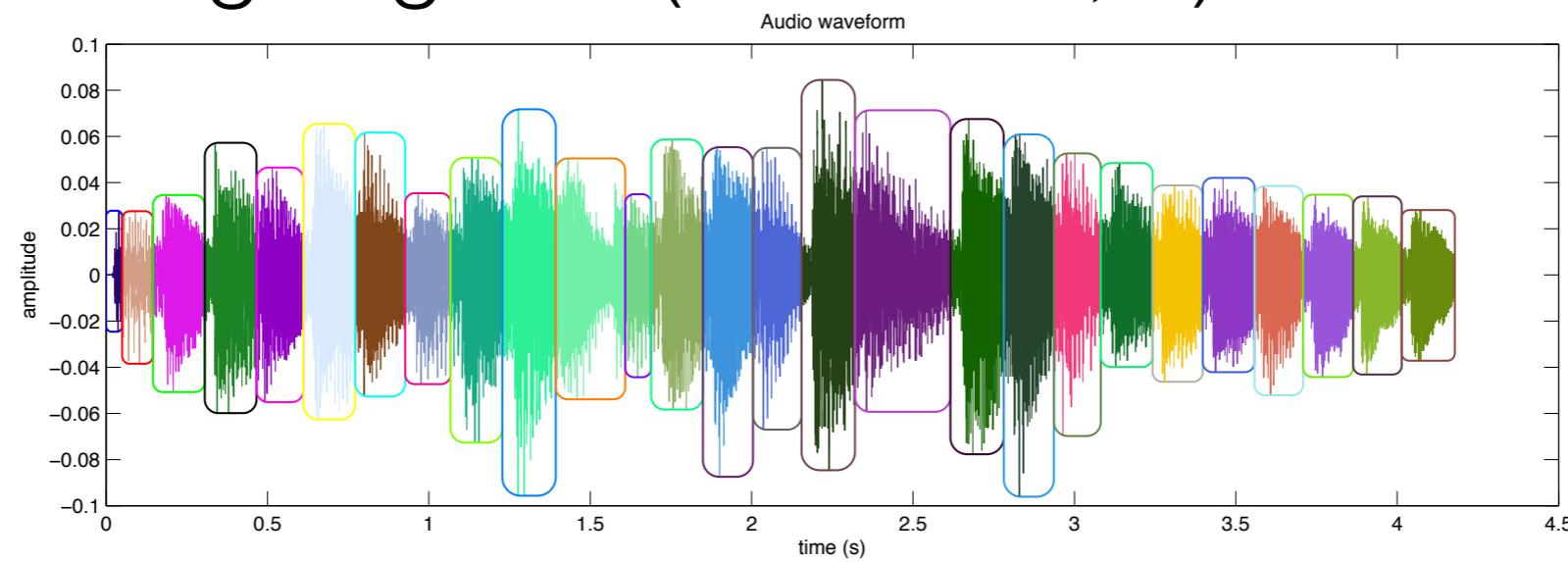


`sig.segment('audiofile', o)`



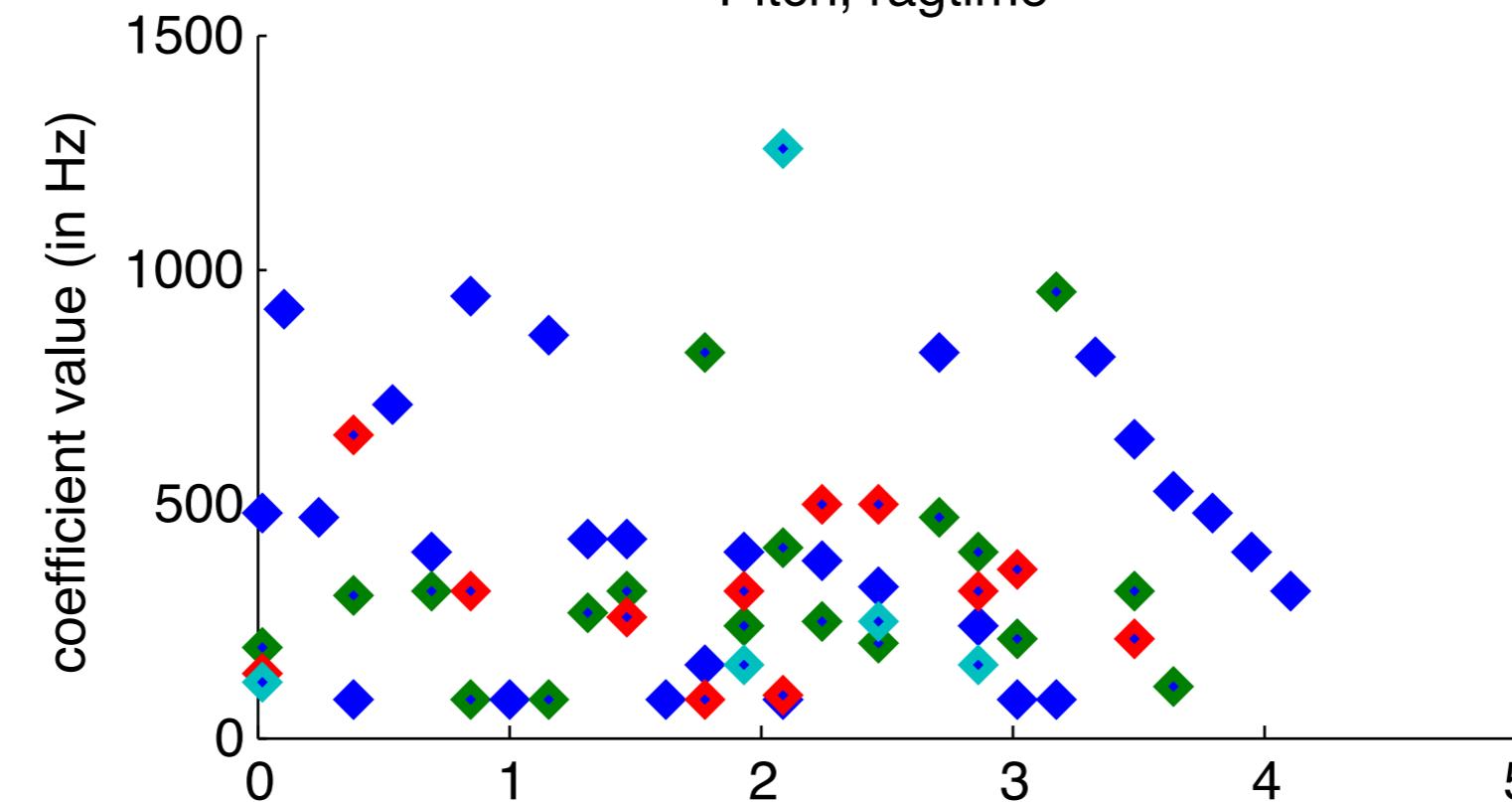
# onset-based segmentation

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



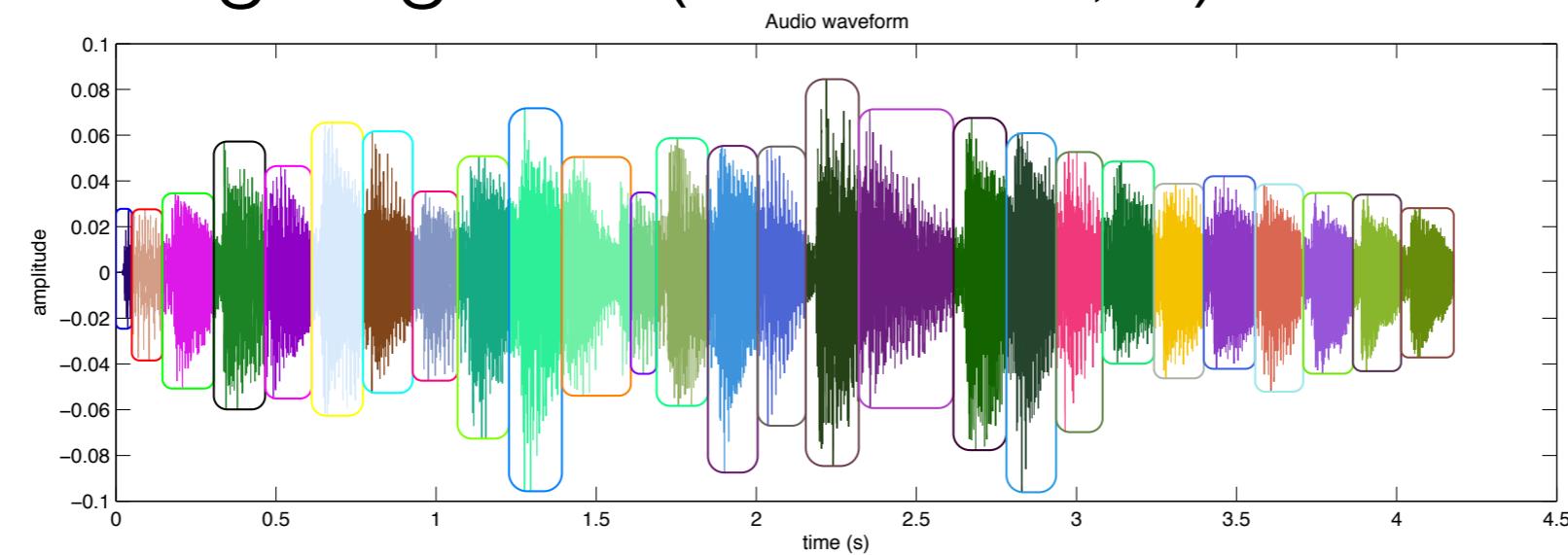
`mus.pitch(s)`

Pitch, ragtime



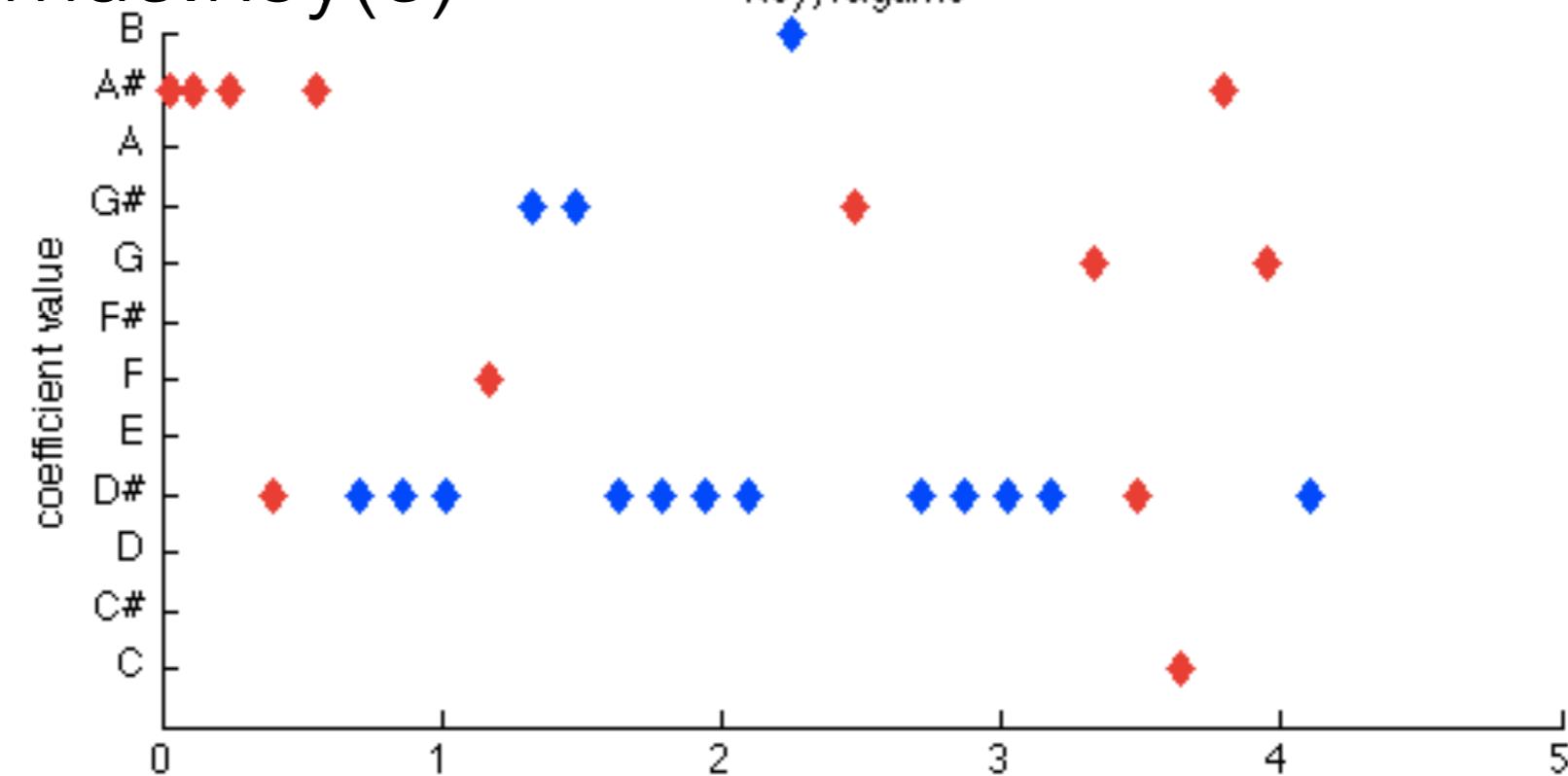
# onset-based segmentation

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



`mus.key(s)`

Key, ragtime



*Structural  
levels*

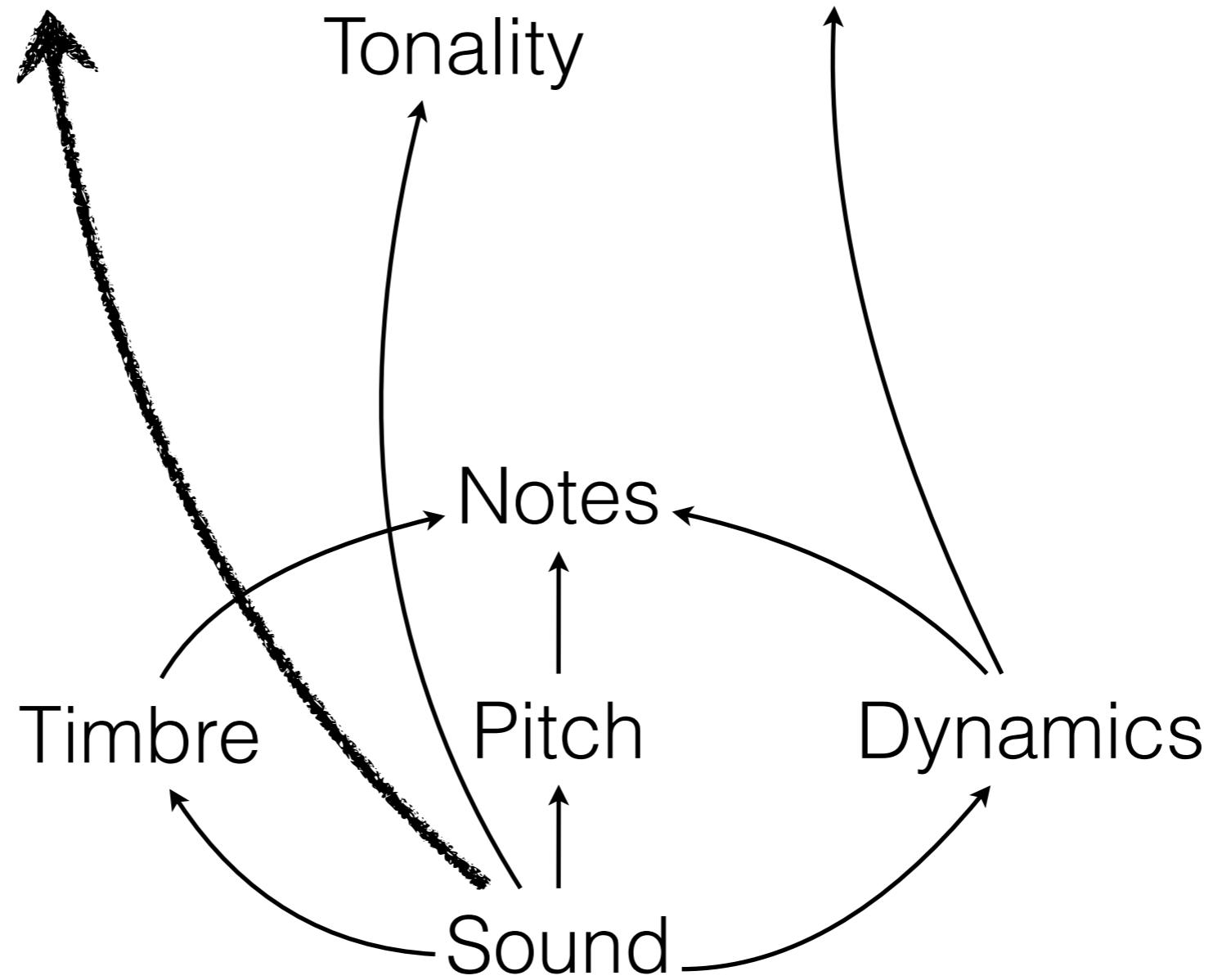
## Groups

Mode  
Tonality

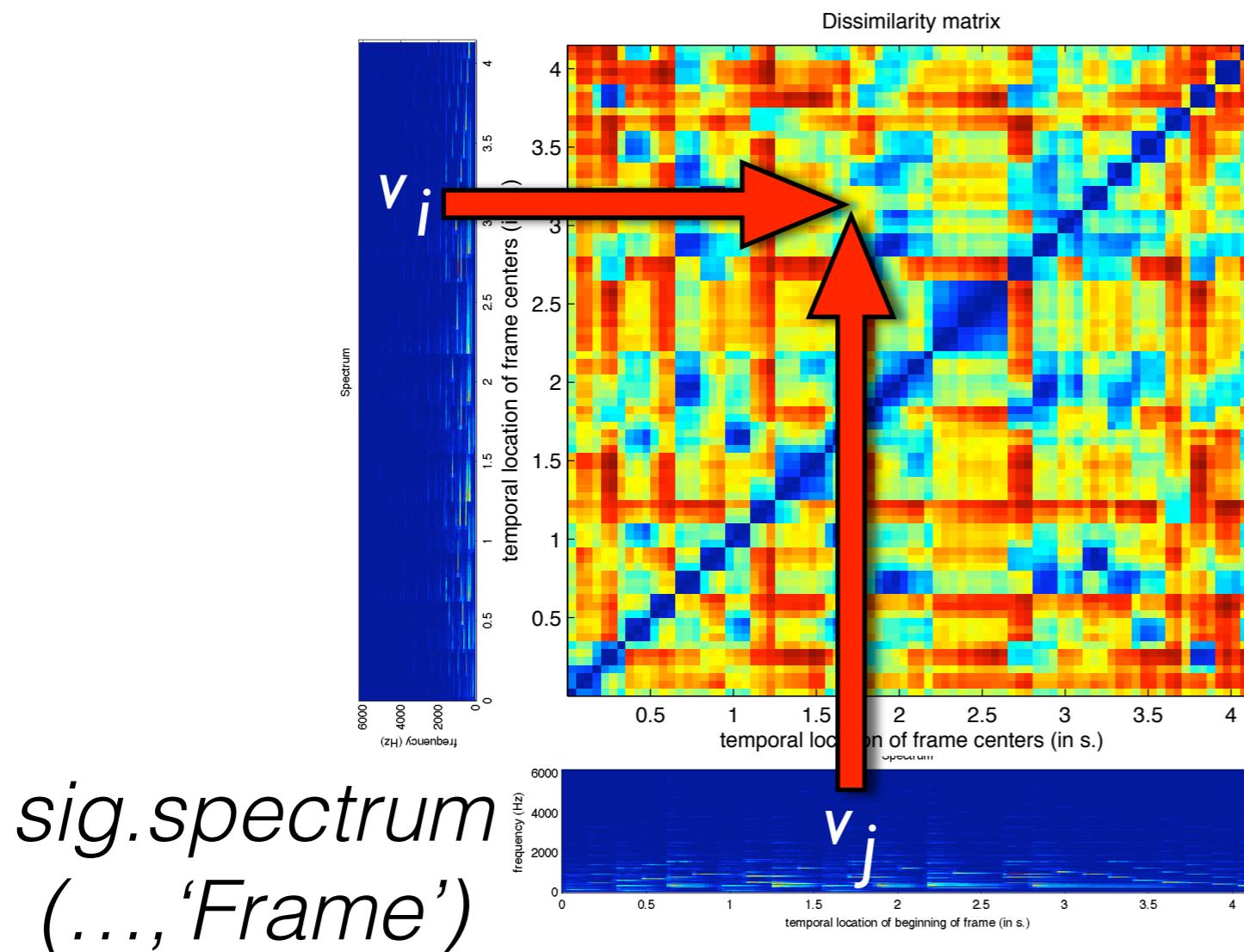
Meter

*Symbolic level*

*Audio level*



# *sig.simatrix* dissimilarity matrix



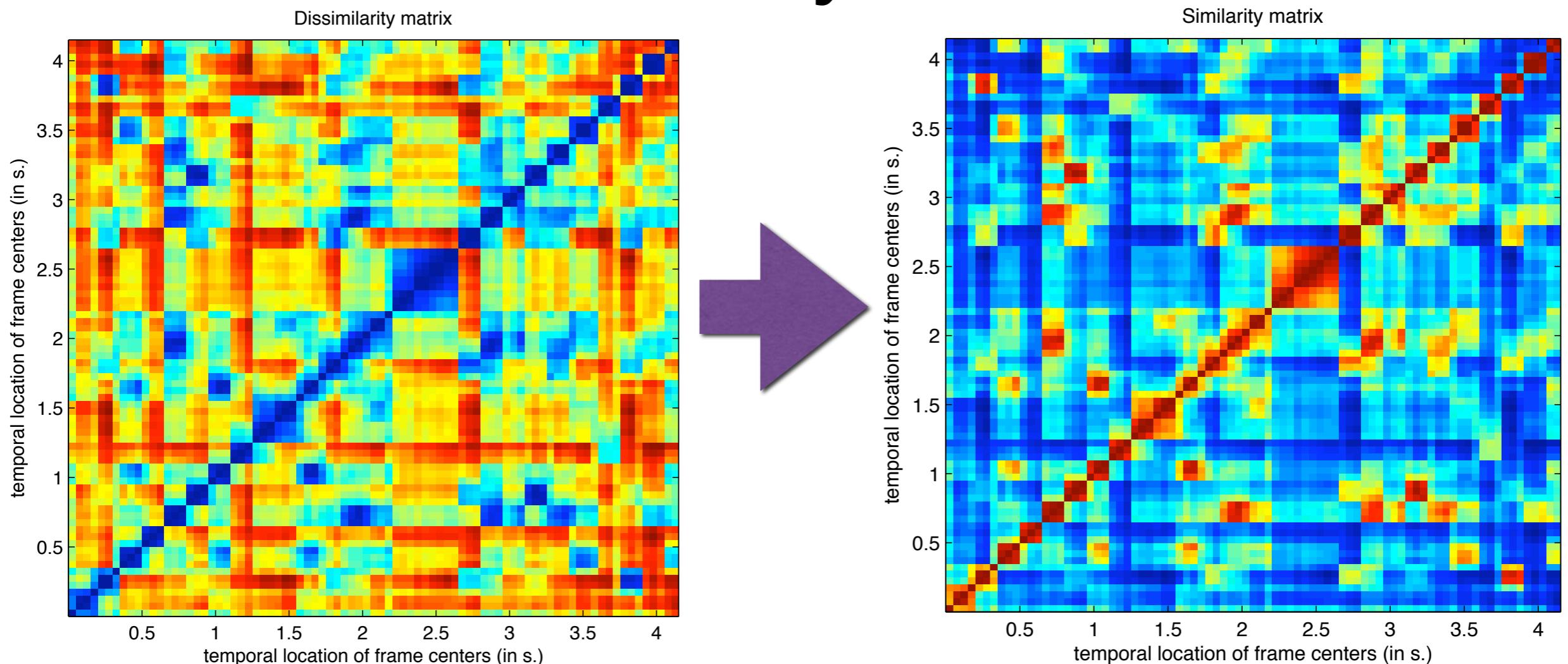
*sig.spectrum*  
(..., 'Frame')

- *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

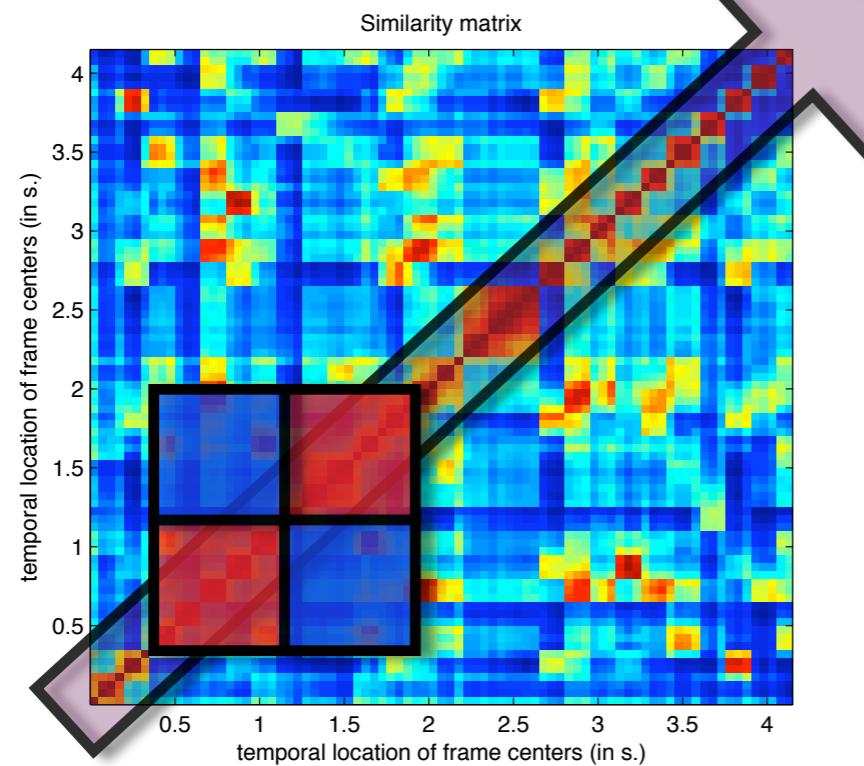


- *sig.simatrix(a, ‘**Similarity**’, ‘*exponential*’)*  
$$d_{exp}(v_i, v_j) = \exp(-d_{cos}(v_i, v_j))$$

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

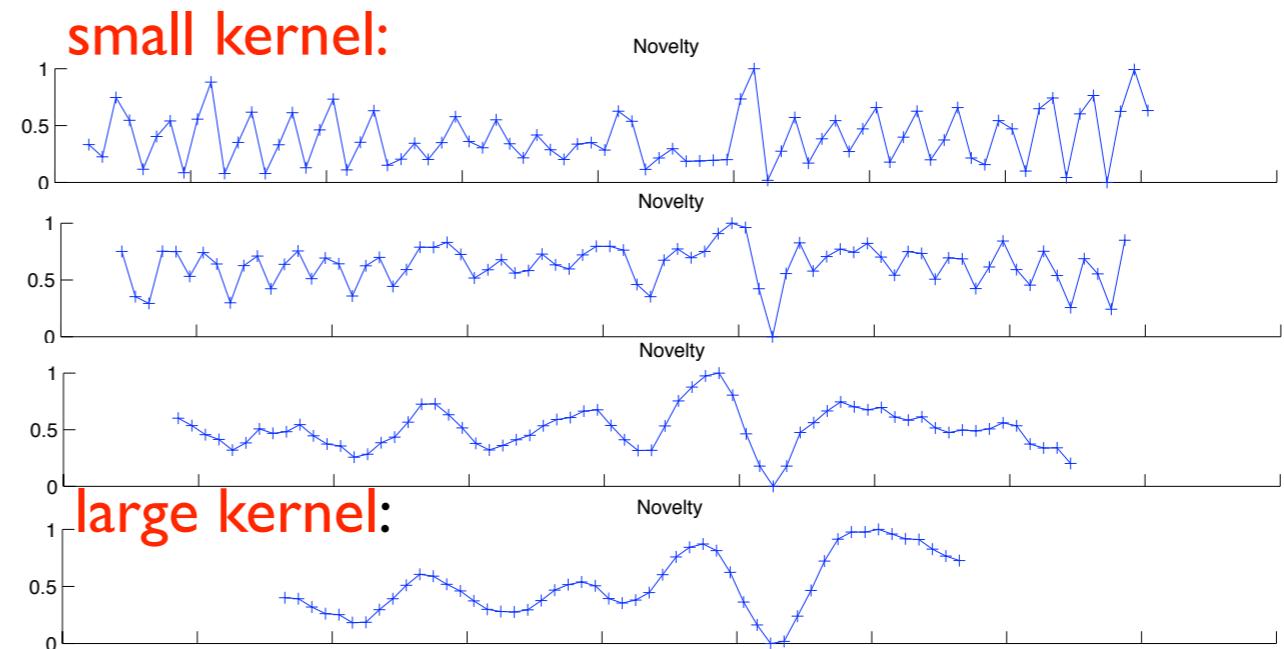
# *aud.novelty* novelty curve

*sig.simatrix(a)*



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

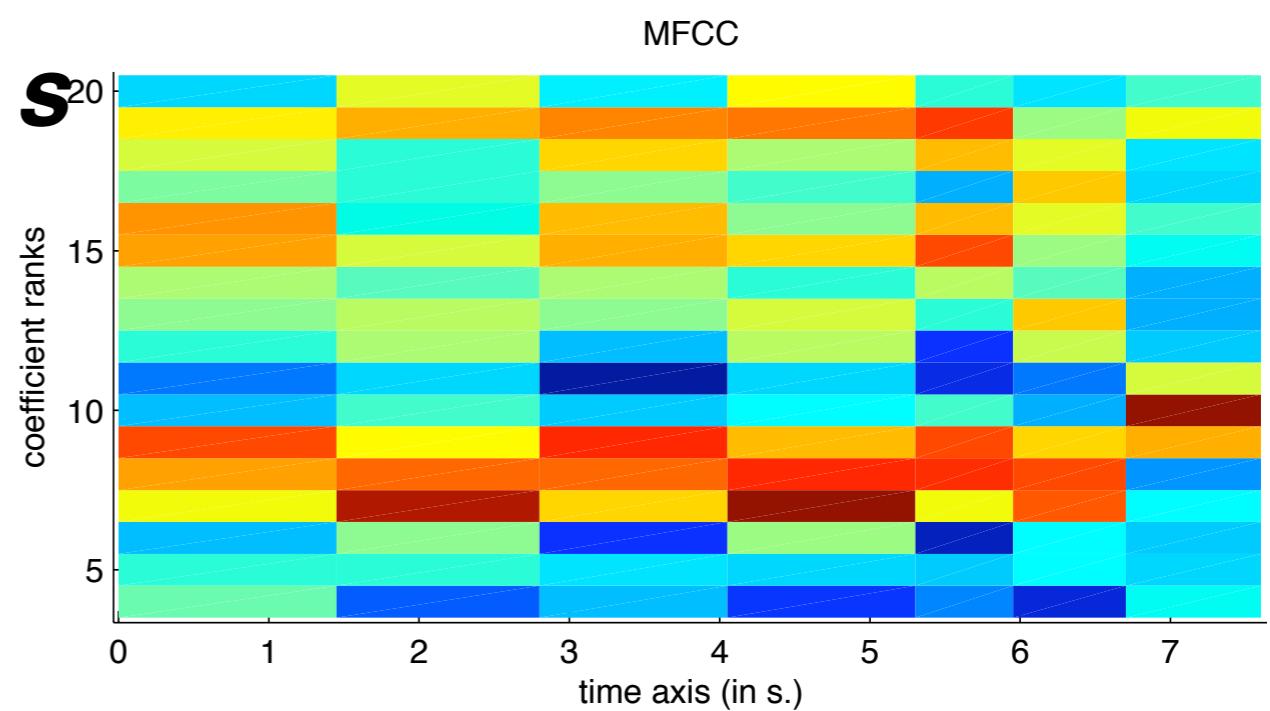
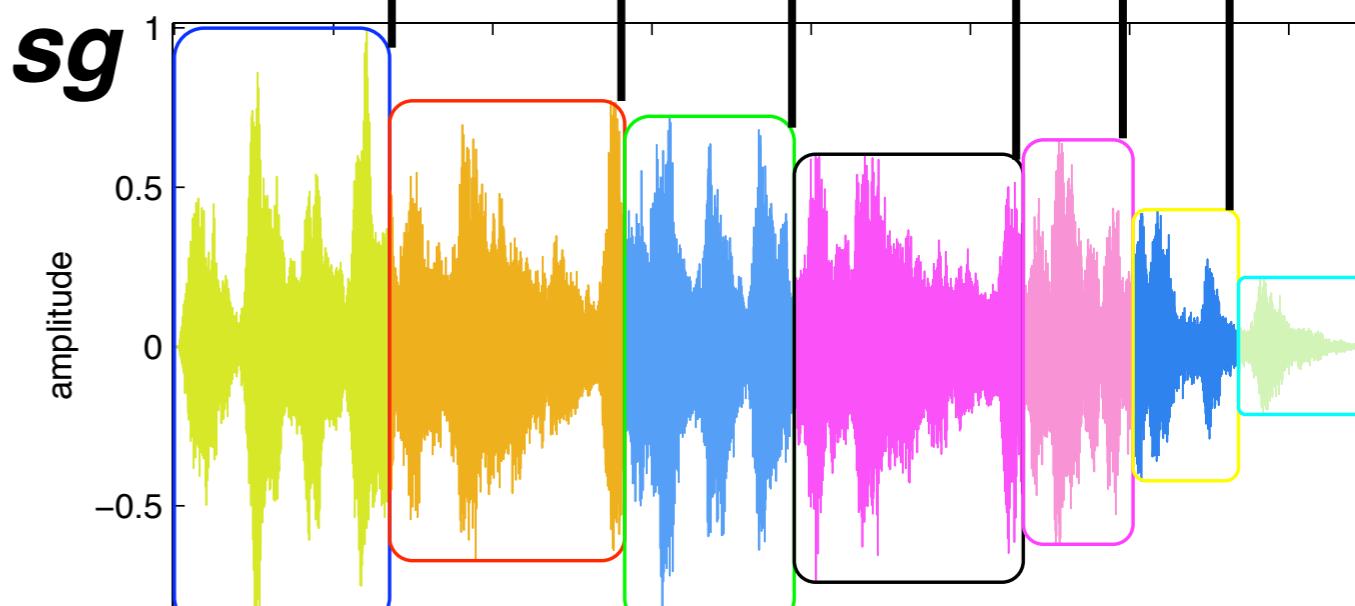
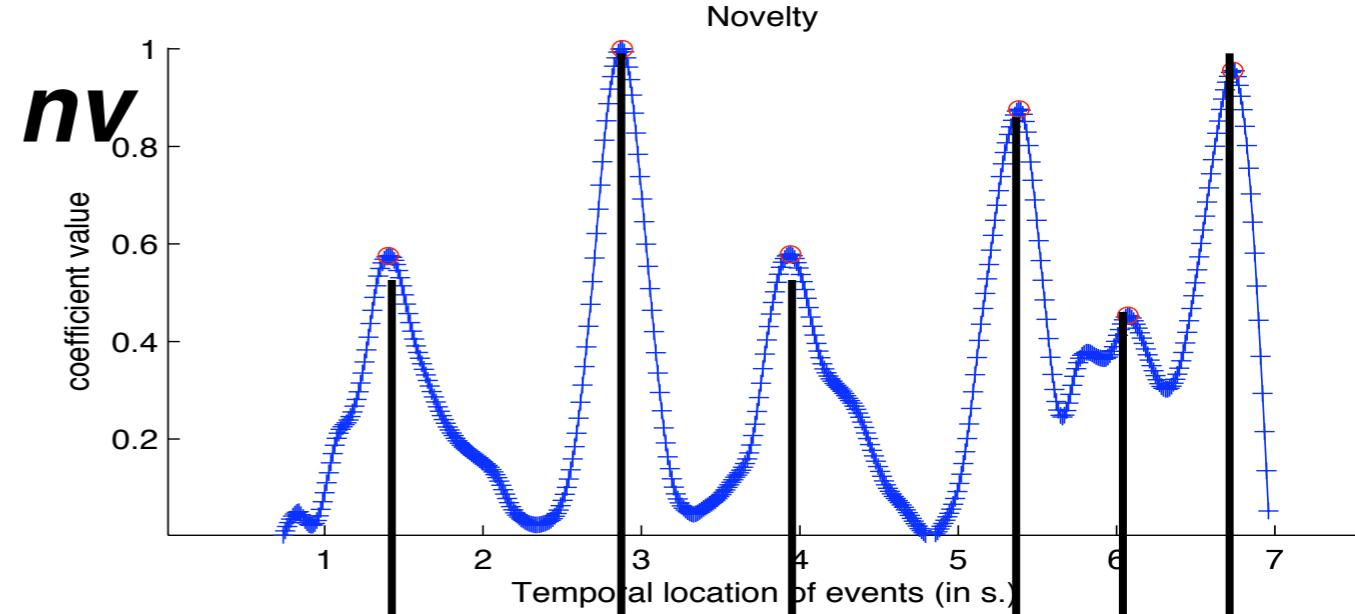
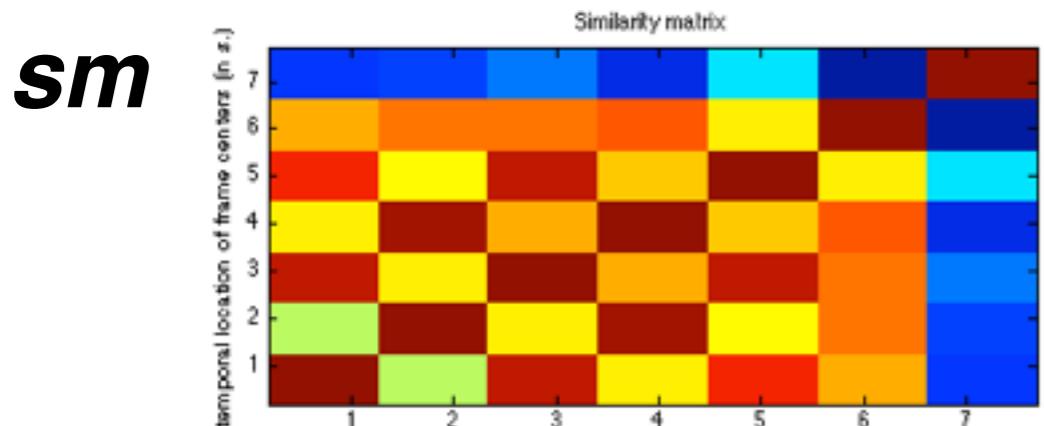
$s = 128$  samples



Convolution with Gaussian  
checkerboard kernel

# *aud.segment* novelty-based segmentation

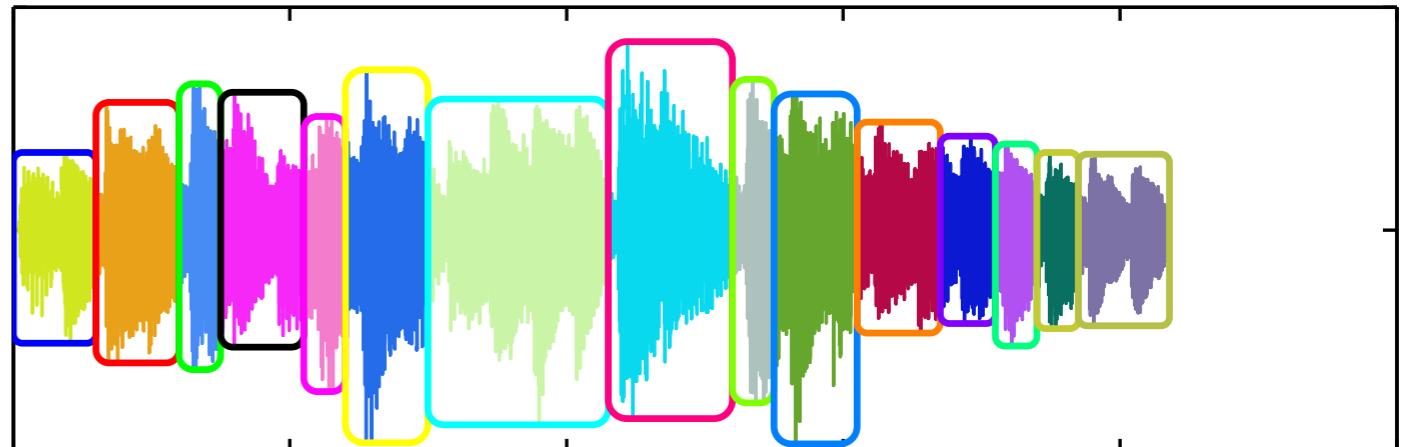
- $nv = aud.novelty(sm)$
- $sg = sig.segment('mysong', nv)$
- $sg = sig.segment('mysong')$
- $sg.play$
- $s = aud.mfcc(sg)$
- $sm = sig.simatrix(s)$



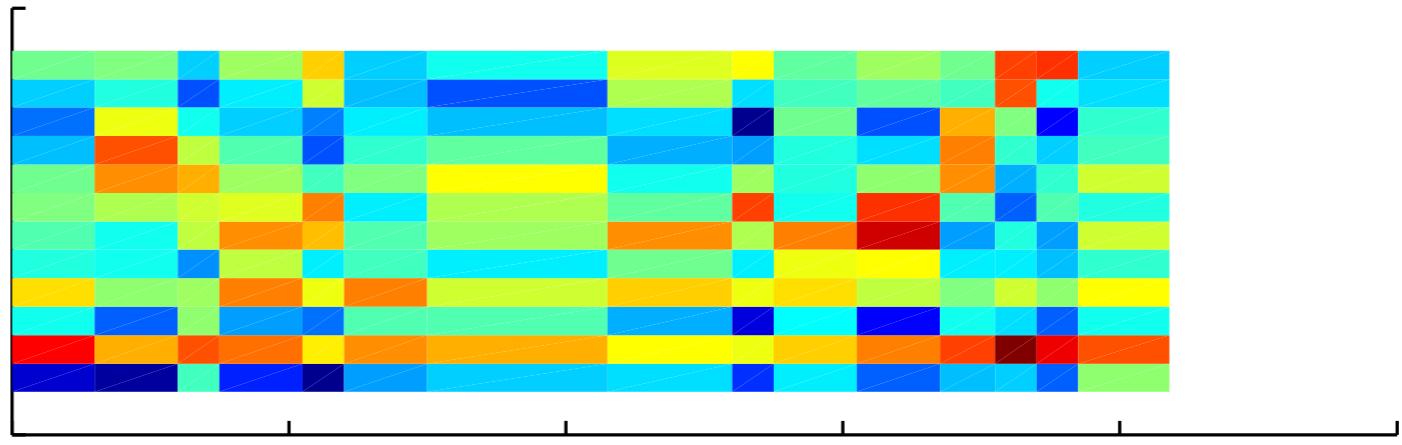
# *sig.cluster*

## clustering of audio segments

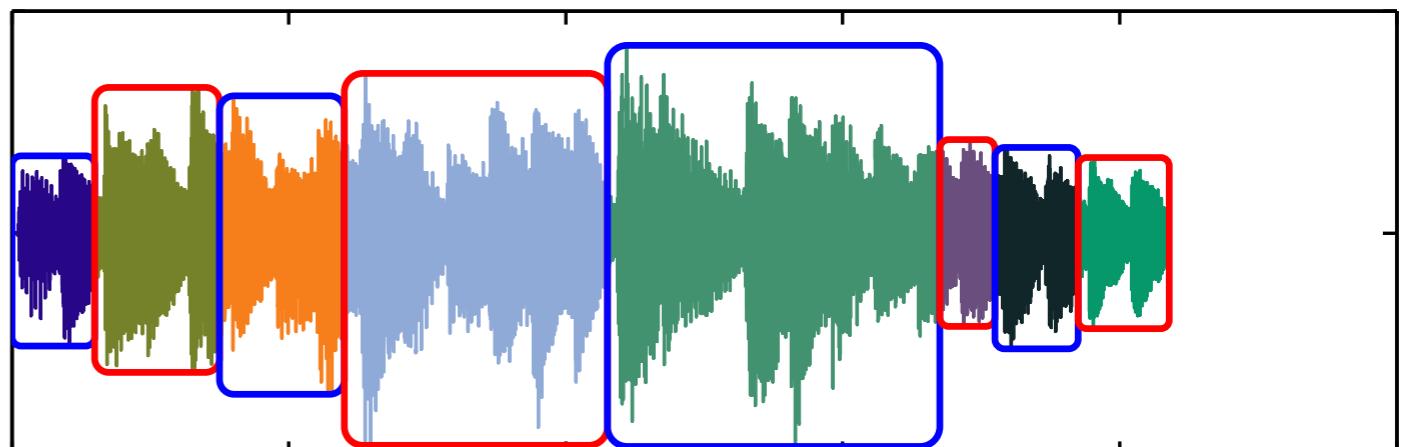
- $sg = aud.segment(a)$



- $cc = aud.mfcc(sg)$



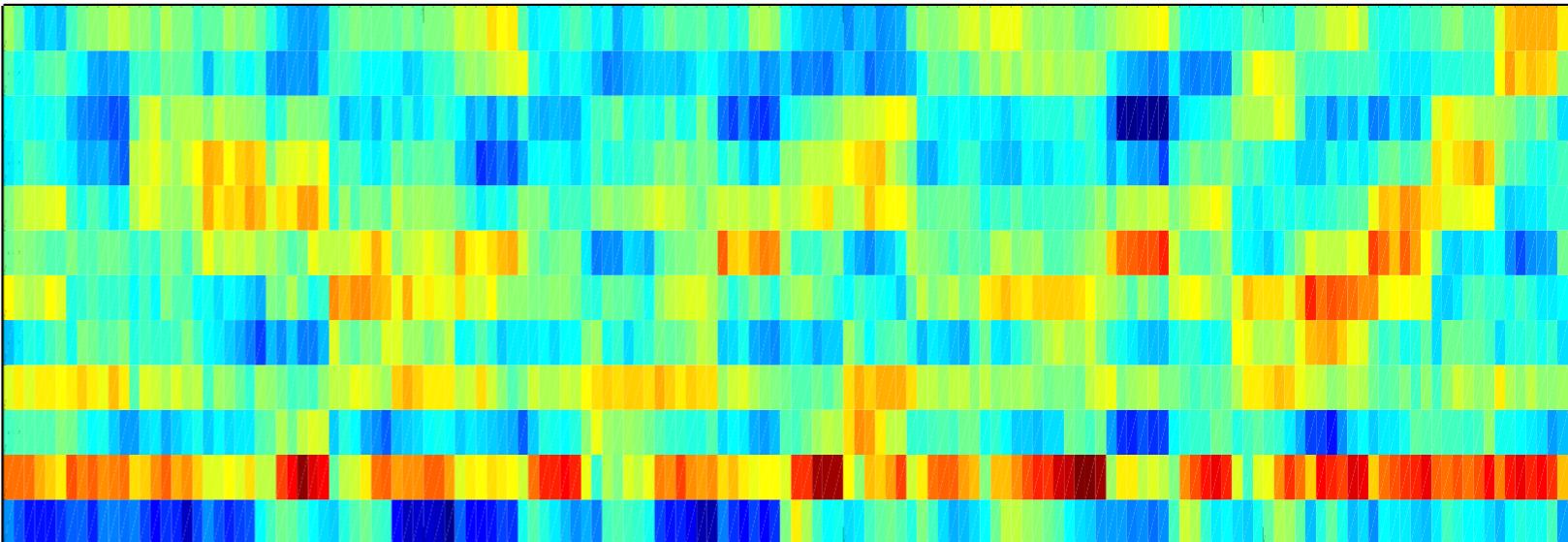
- $sig.cluster(sg, cc)$



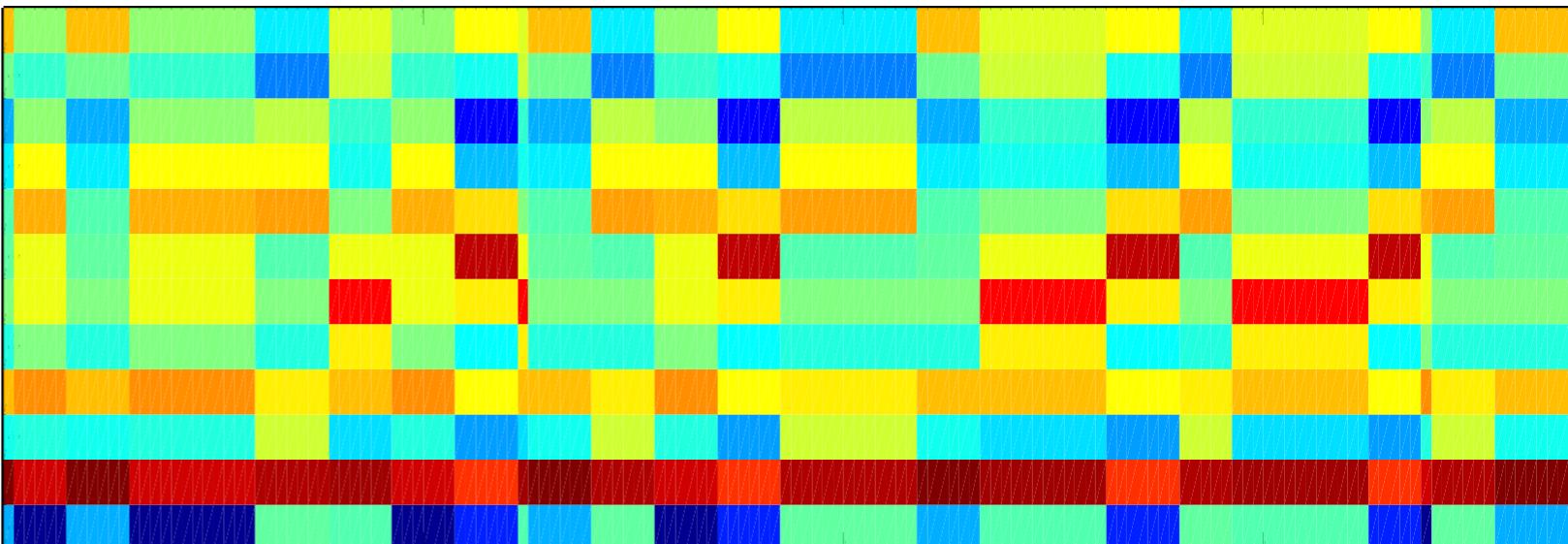
# *sig.cluster*

## clustering of frame-decomposed feature

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



- $sig.cluster(cc, 4)$

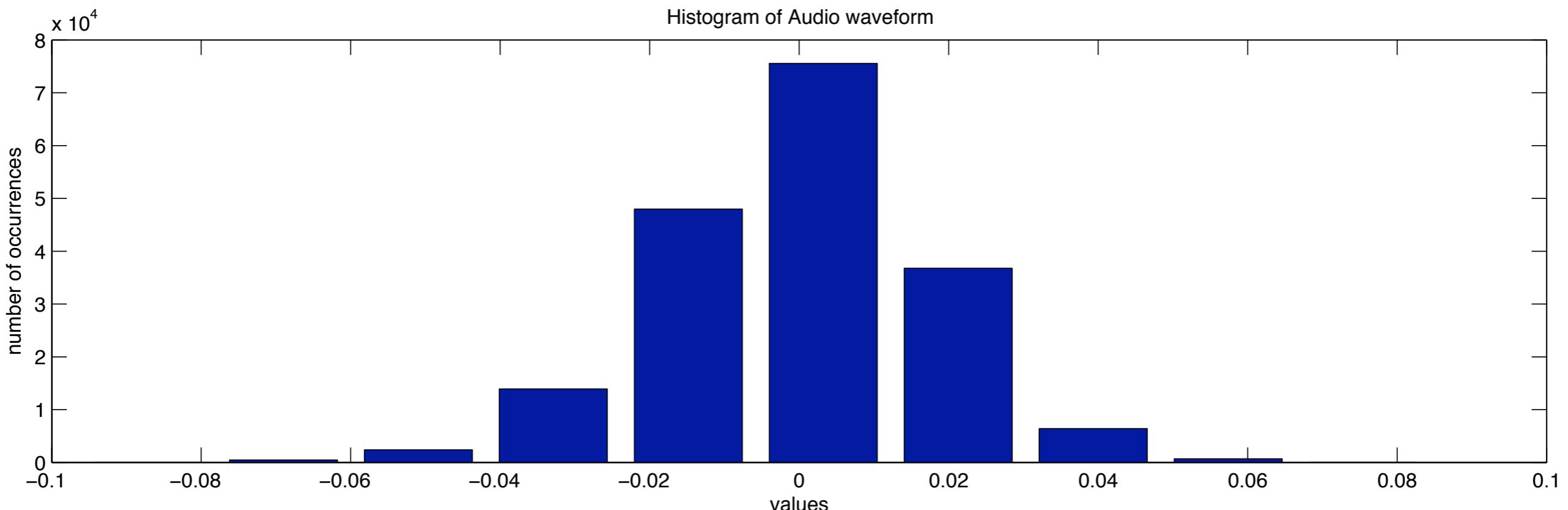


# *sig.stat*

## basic statistics

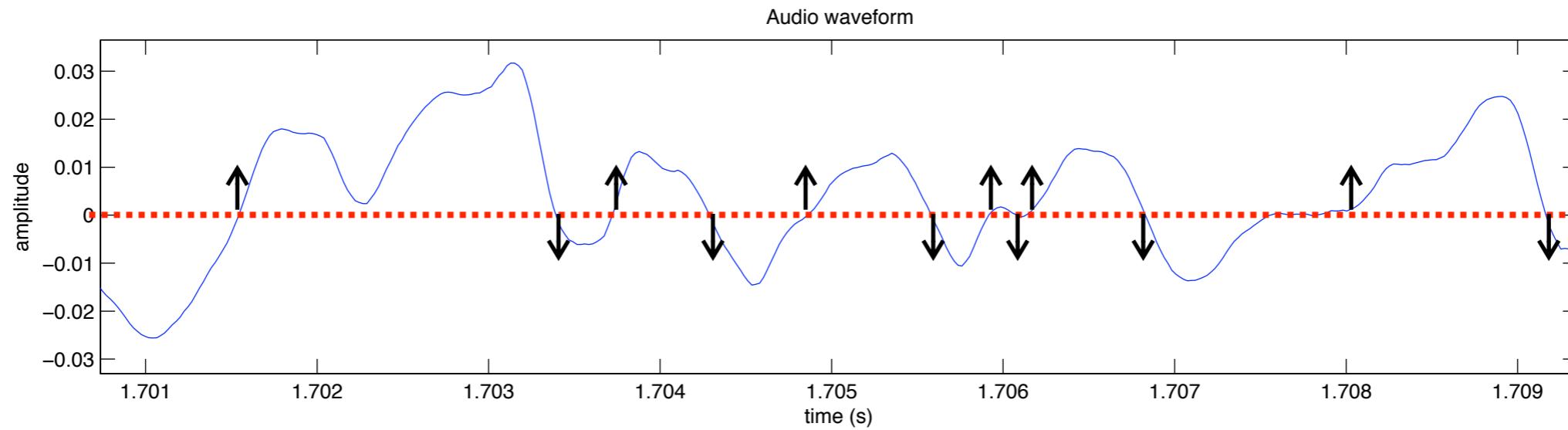
- mean
- standard deviation
- temporal slope
- main periodicity:
  - frequency
  - amplitude
  - periodicity entropy

# *sig.histo* histogram



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

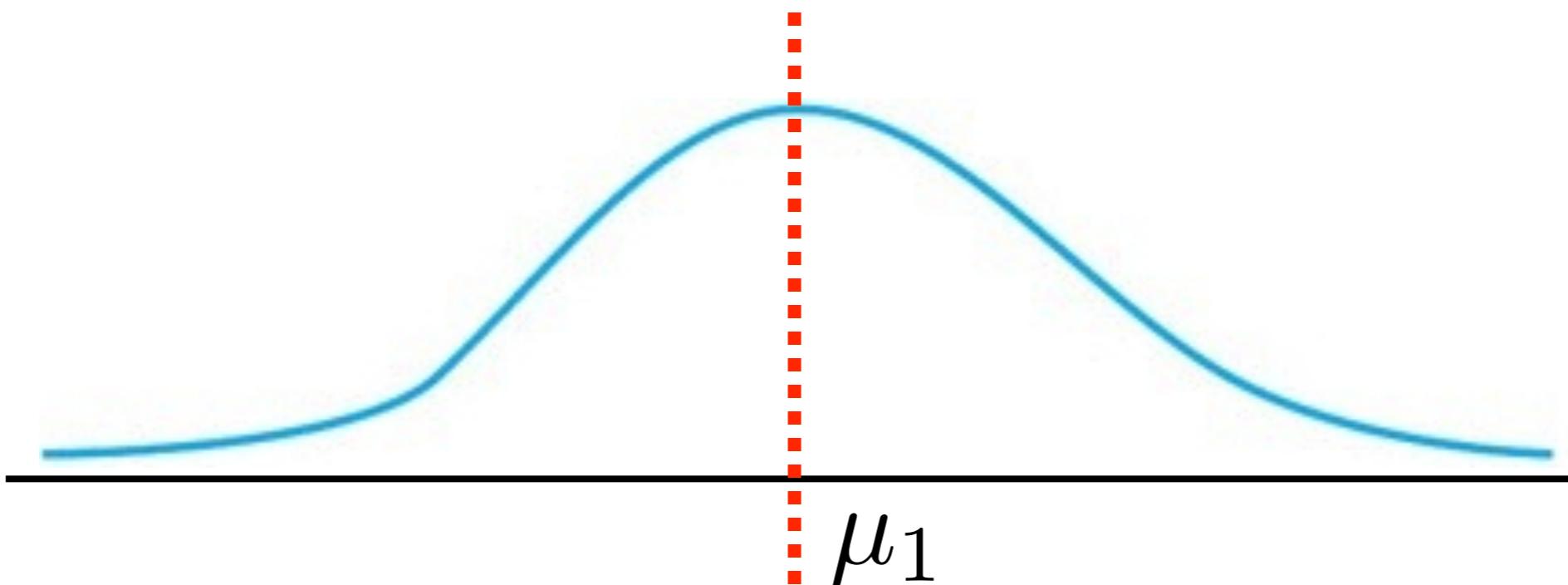
# *sig.zerocross* waveform sign-change rate



- 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

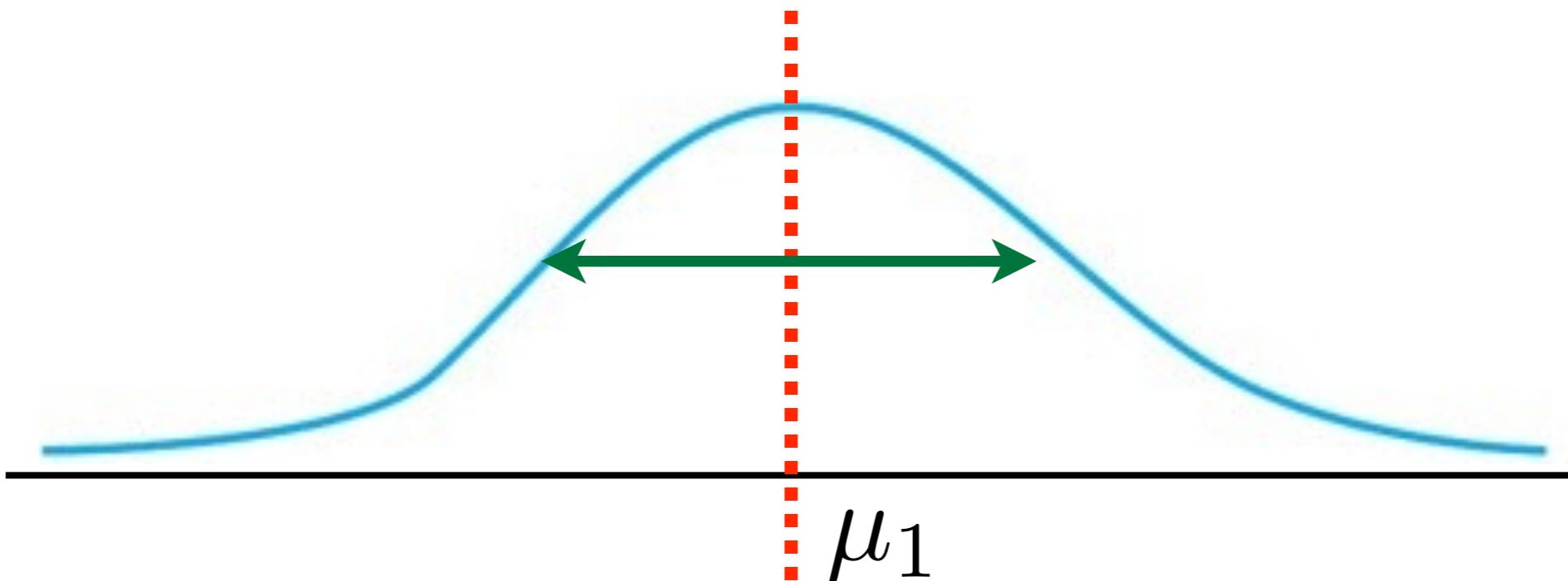
first moment:  $\mu_1 = \int x f(x) dx$



# *sig.spread* variance, dispersion

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

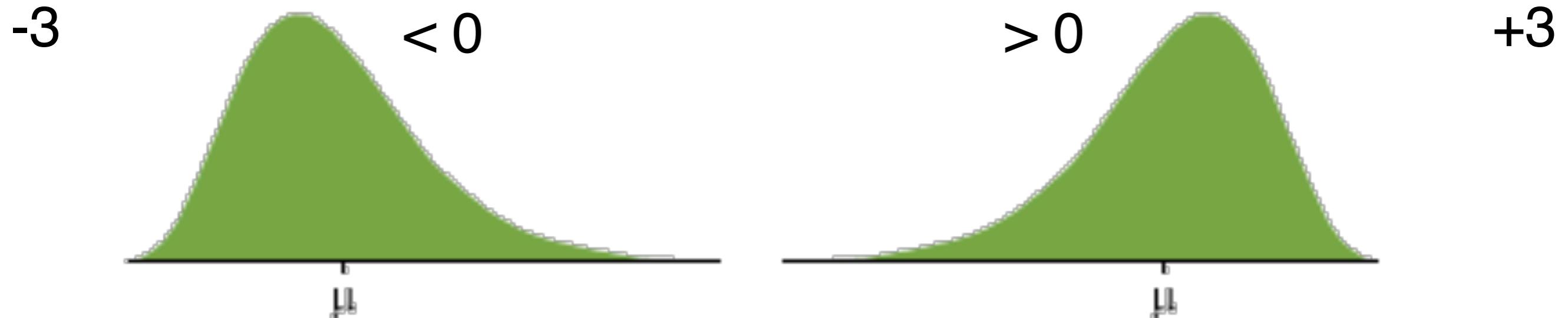
standard deviation:  $\sigma$



# *sig.skewness* non-symmetry

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

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

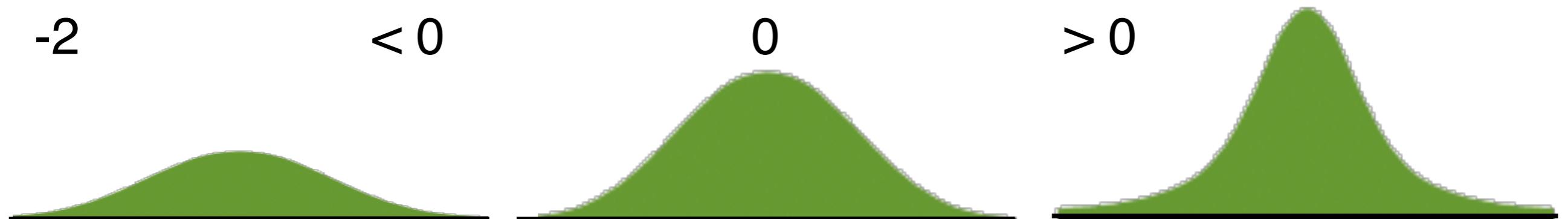


# *sig.kurtosis* pickiness

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

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

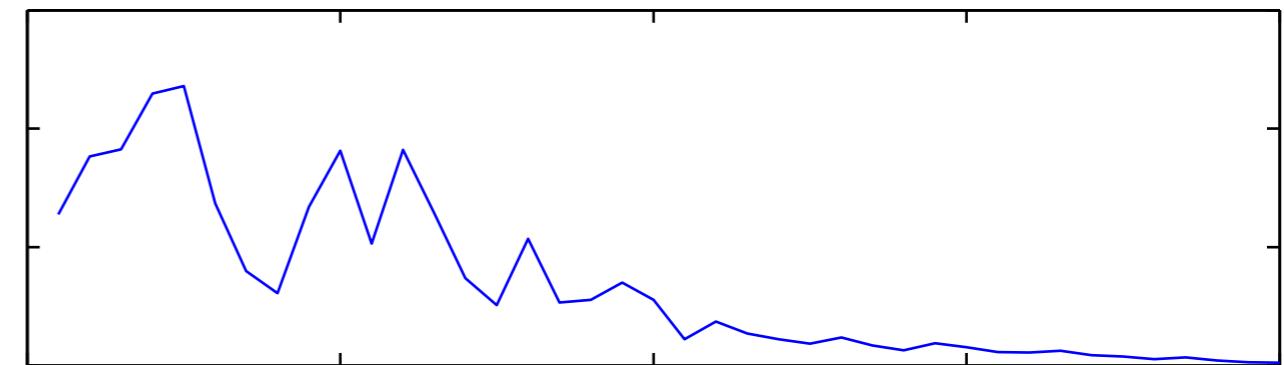
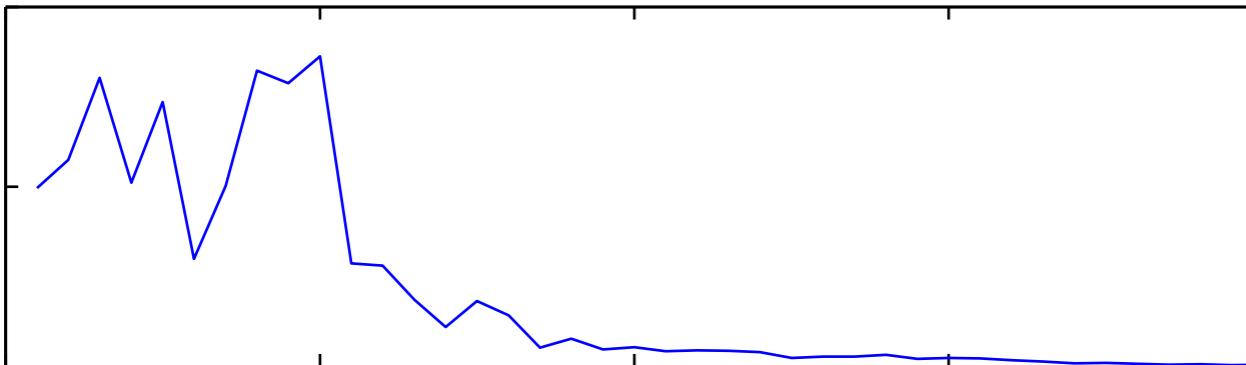
# *aud.play* feature-based playlist

- `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.dist*

## distance between features

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



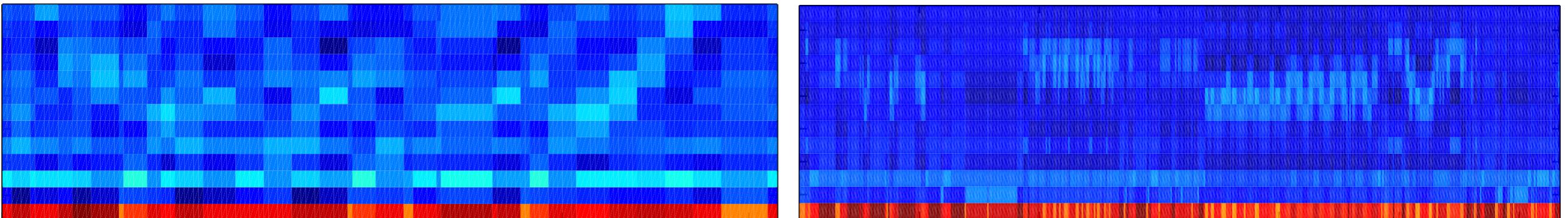
- *sig.dist(s1, s2, 'Cosine')*

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

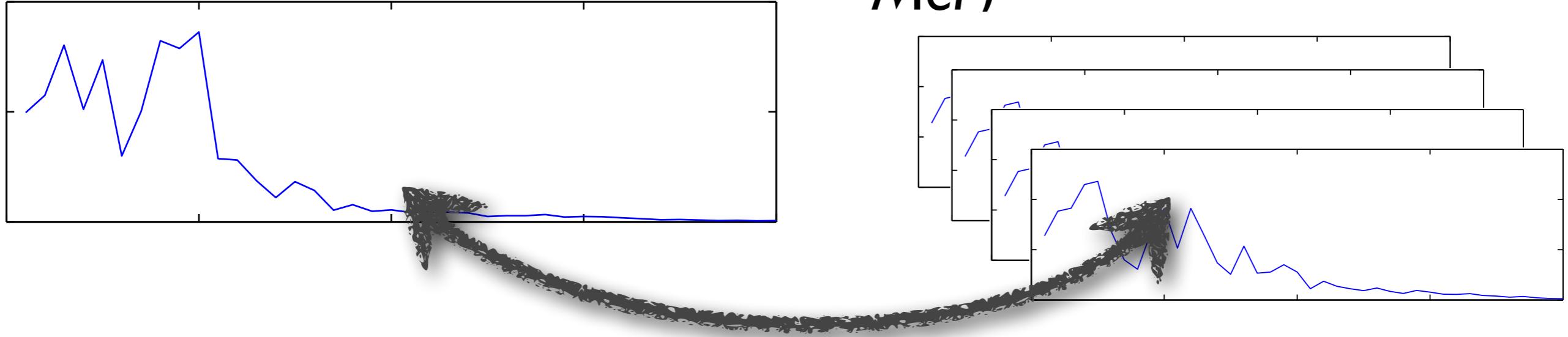
- $sig.dist(c1, c2)$

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**

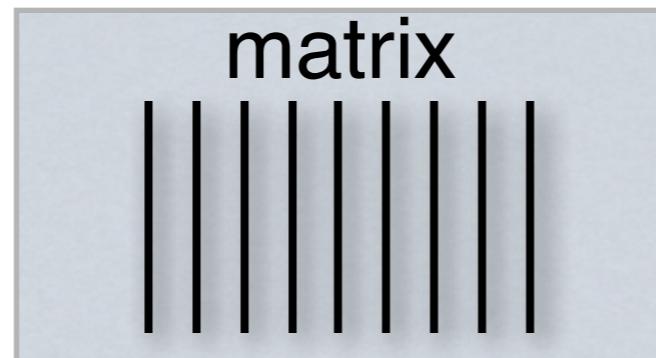
*.getdata*

returns data in Matlab format

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

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

*s.getdata*



**s**

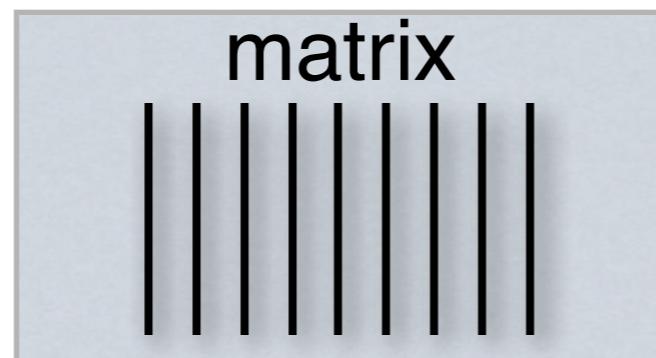
*.getdata*

returns data in Matlab format

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

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

*f.getdata*



**f**

# *.getdata*

## returns data in Matlab format

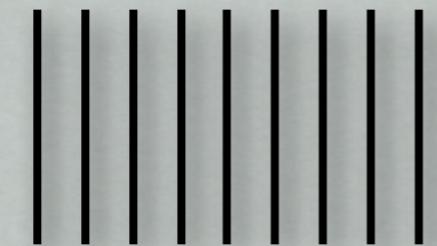
```
sg = sig.segment('file')  
f = sig.filterbank(sg, ——————  
'Frame');
```

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

*f.getdata*

cell array

matrix



matrix

...

**f**

*.getpeakpos, .getpeakval*  
returns data in Matlab format

**p** = sig.peaks...

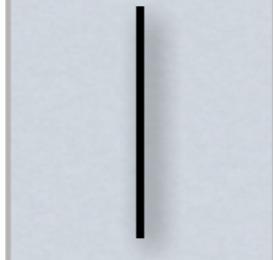


Encapsulated data  
numerical data,  
related sampling rates,  
related file name,  
etc.

**p**

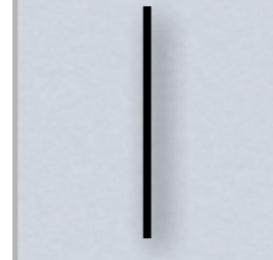
*p.getpeakpos*

vector



*p.getpeakval*

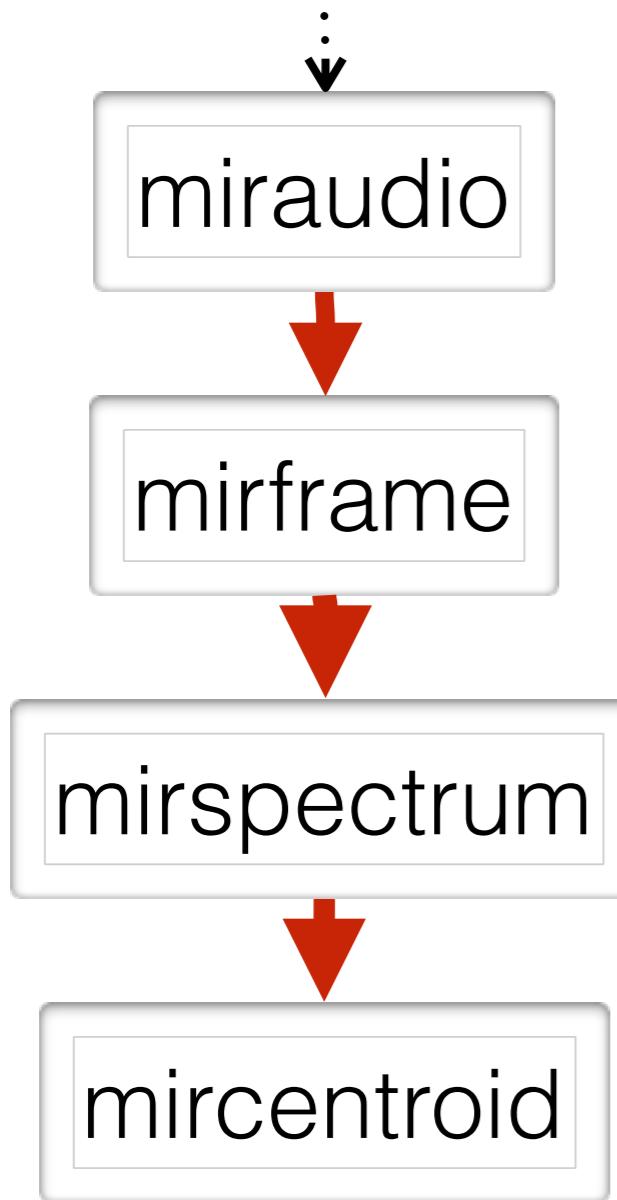
vector





# Limitations of data flow in *MIRtoolbox*

long audio file,  
batch of files

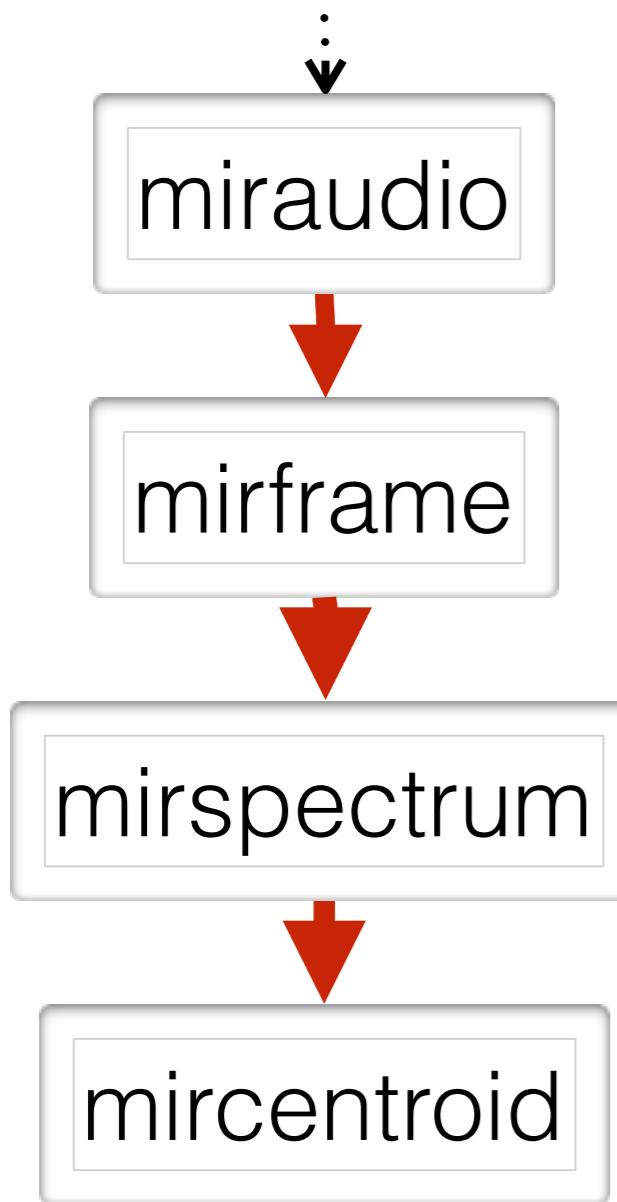


- `a = miraudio('bigfile')`
- `f = mirframe(a)`
- `s = mirspectrum(f)`
- `mircentroid(s)`
- `mircentroid('bigfile', 'Frame')`

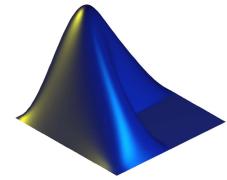


# Data flow graph design & evaluation

long audio file,  
batch of files

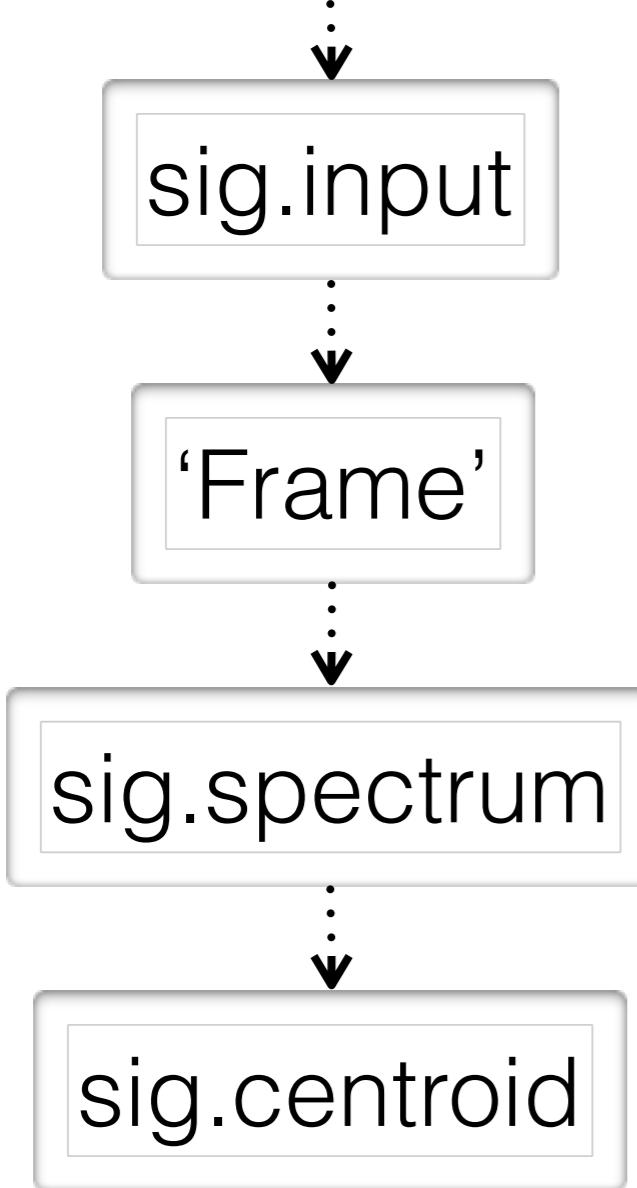


- `a = miraudio('Design', ...)`
- `s = mirspectrum(f, 'Frame', ...)`
- `c = mircentroid(s)`
- `mireval(c, 'bigfile')`



# 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);` → 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.eval*

## data flow graph evaluation

- `a = sig.input('bigfile', ...);`
  - `s = sig.spectrum(a);`
  - `c = sig.centroid(s)`
  - `c.eval`
  - `c.eval('anotherfile')`
- Evaluate the data flow graph for the **particular audio file** (or '**Folder**') assigned during the graph design
- Apply the same data flow graph to **another audio file** (or '**Folder**')

# *sig.design.show*

## data flow graph display

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

```
> sig.input ( 'ragtime' )
frameconfig: 0
mix: 'Pre'
sampling: 0
center: 0
sampling: 0
extract: []
trim: 0
trimwhere: 'BothEnds'
trimthreshold: 0.0600
halfwave: 0
```

```
> sig.spectrum ( ... )
win: 'hamming'
min: 0
max: Inf
mr: 0
res: NaN
length: NaN
zp: 0
wr: 0
octave: 0
constq: 0
alongbands: 0
ni: 0
collapsed: 0
rapid: 0
phase: 1
nl: 0
norm: 0
mprod: []
msum: []
log: 0
db: 0
pow: 0
collapsed: 0
aver: 0
gauss: 0
timesmooth: 0
```

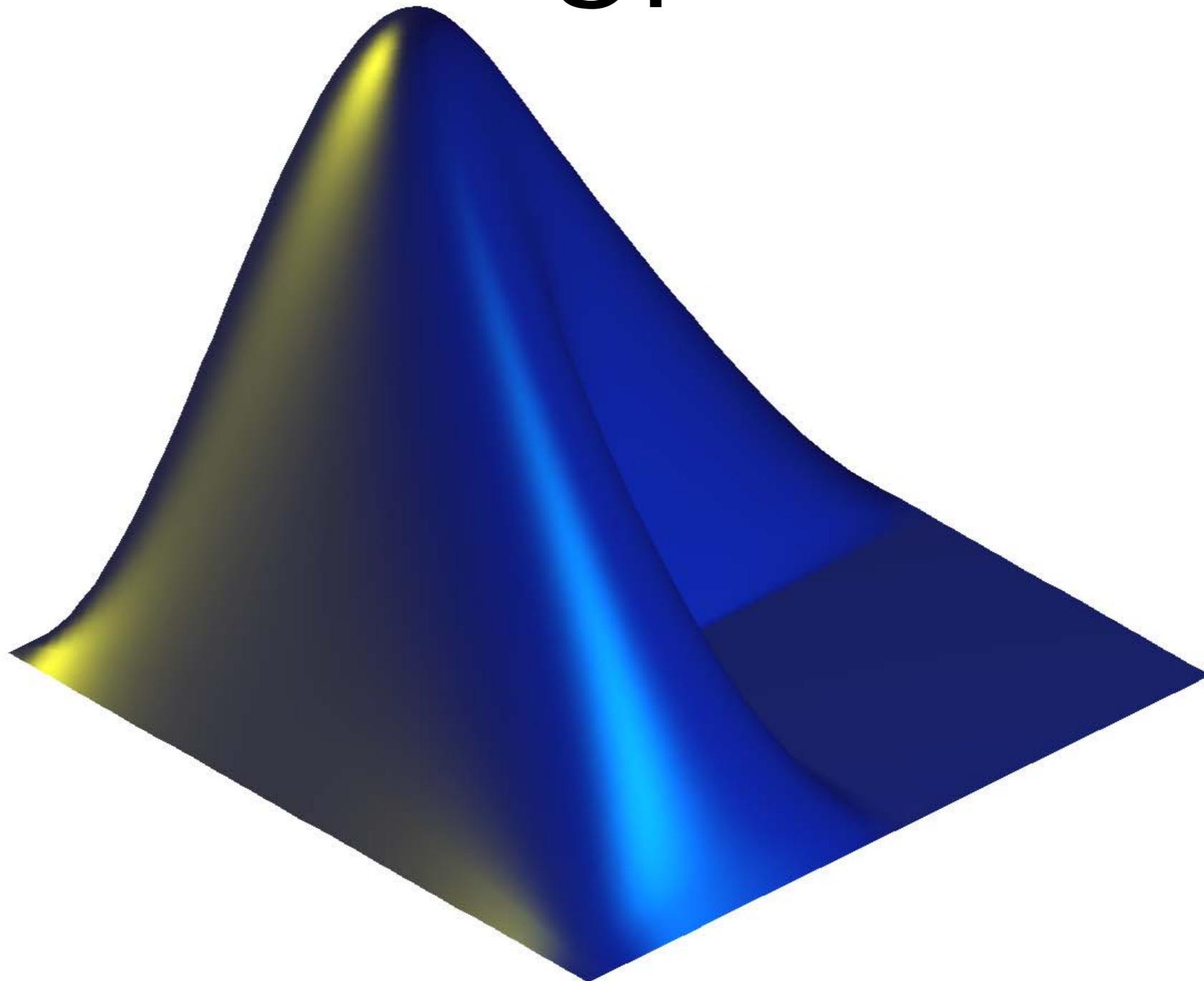
```
> sig.centroid ( ... )
```

# *sig.signal.design*

## design stored in the results

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

3.



Symbolic approaches

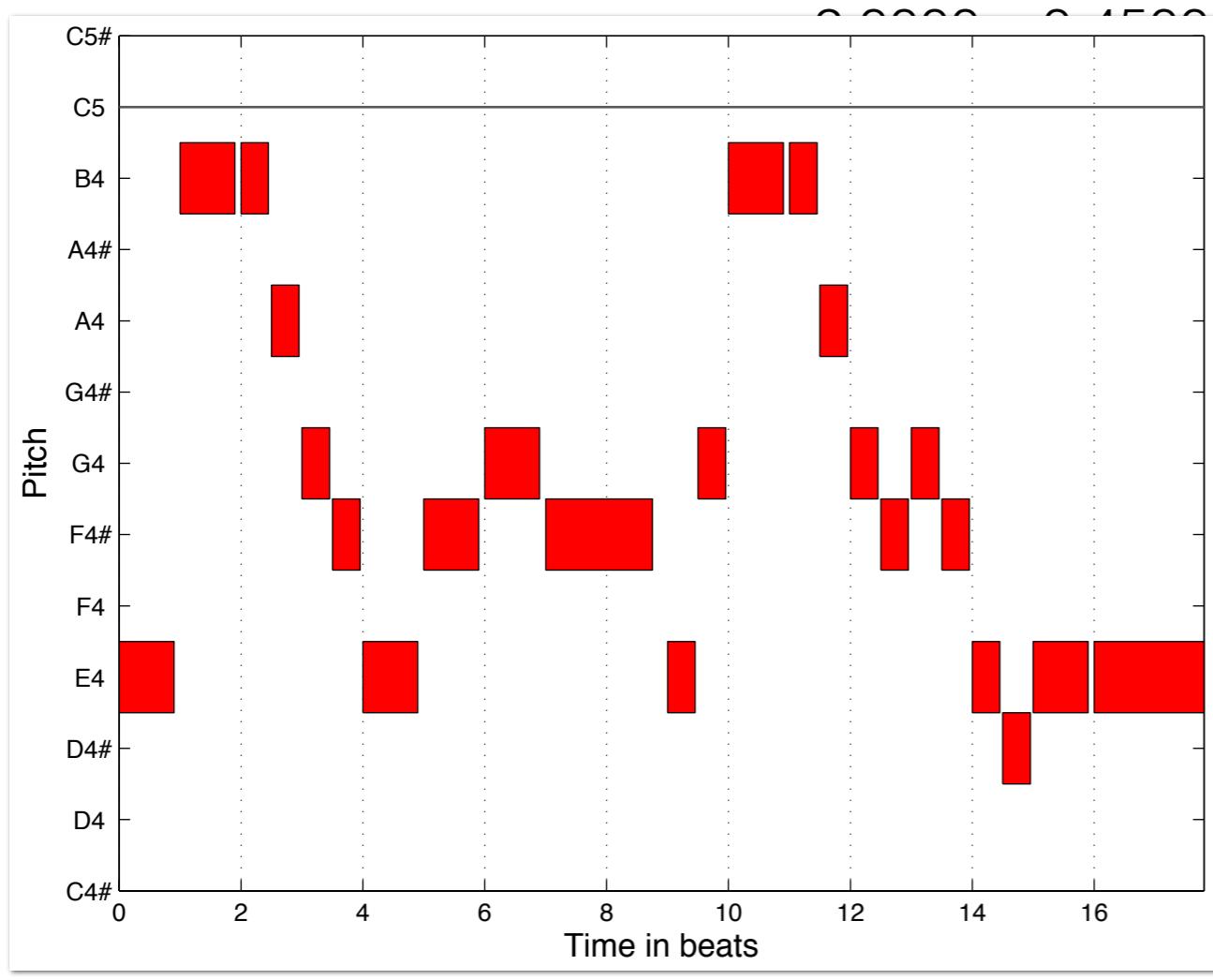
# MIDI Toolbox

- 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	7.9592	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

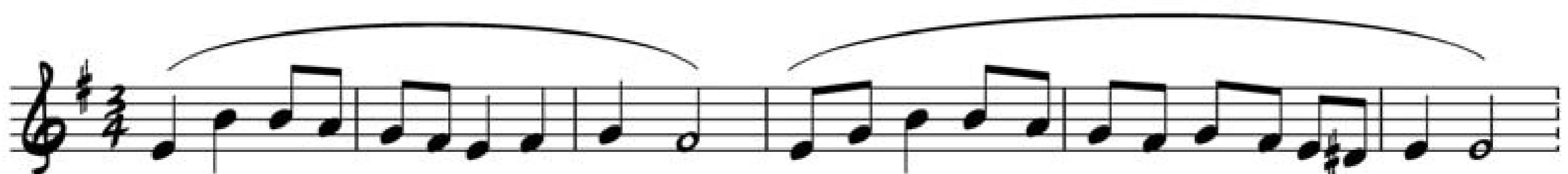
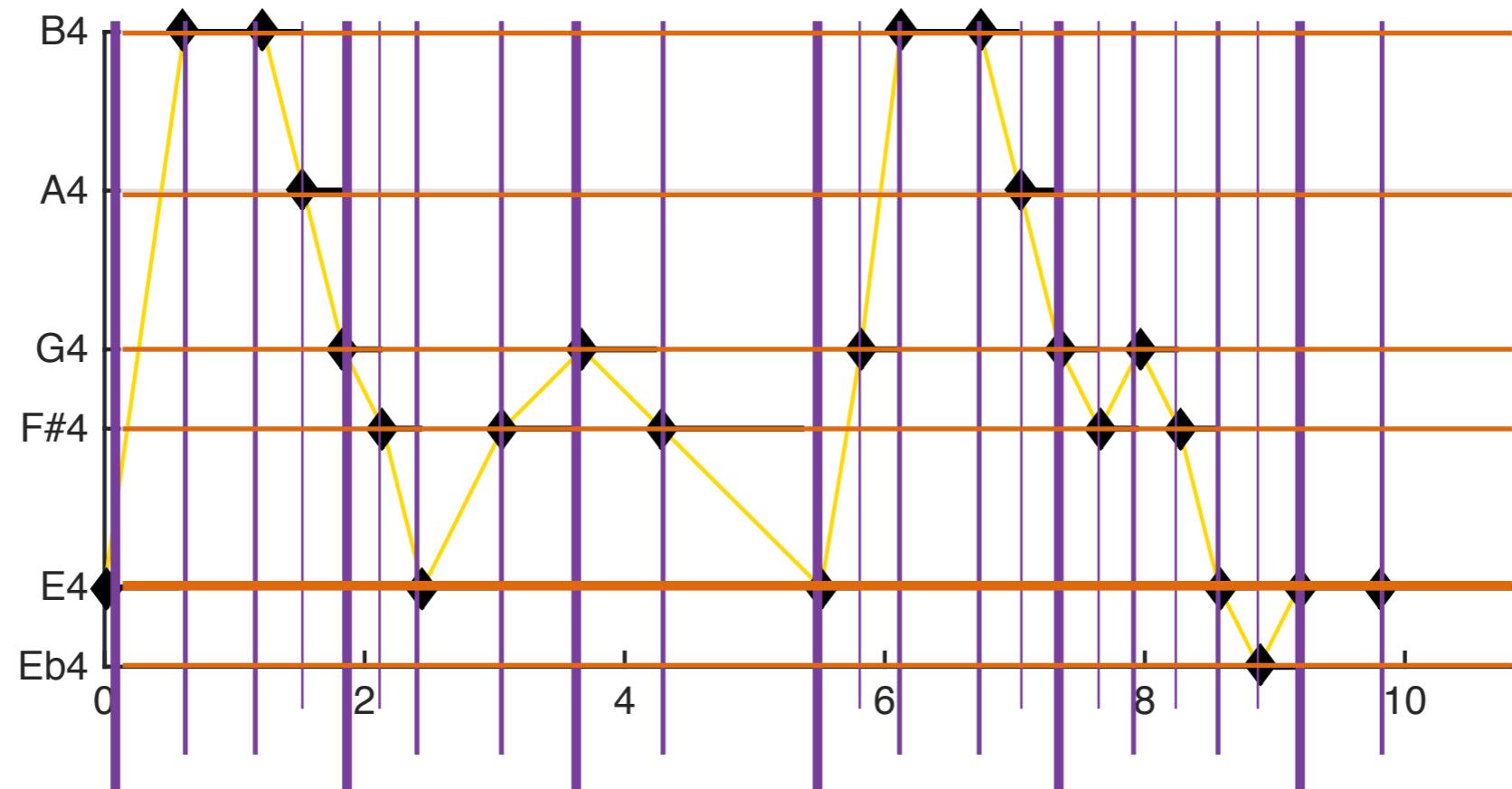


- **metrical grid**: hierarchical construction of pulsations over multiple metrical levels
- **modal and tonal spaces**: mapping pitch values on scale patterns (on delimited temporal regions)
- ✓ • **syntagmatic chains**: successive notes forming voices, enabling to express relative distance between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

# *mus.score* score information

*mus.save*  
*mus.play*

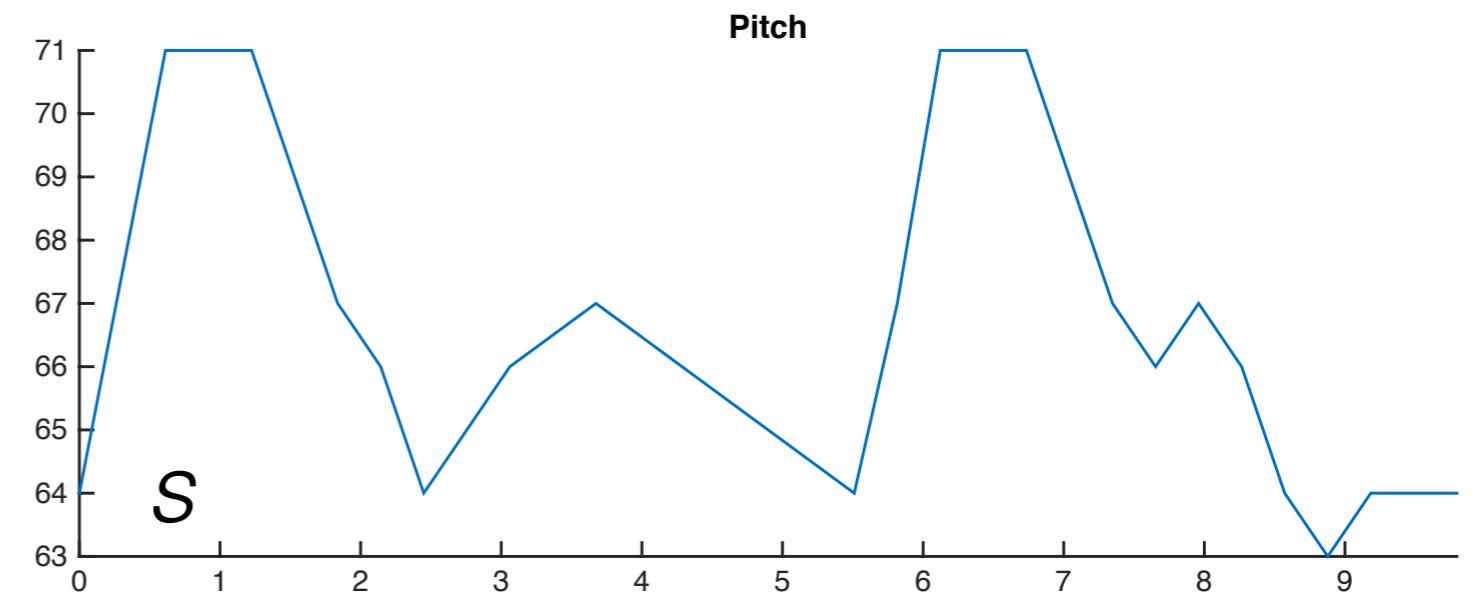
*mus.score('laksin.mid')*



# *mus.pitch* pitch contour

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

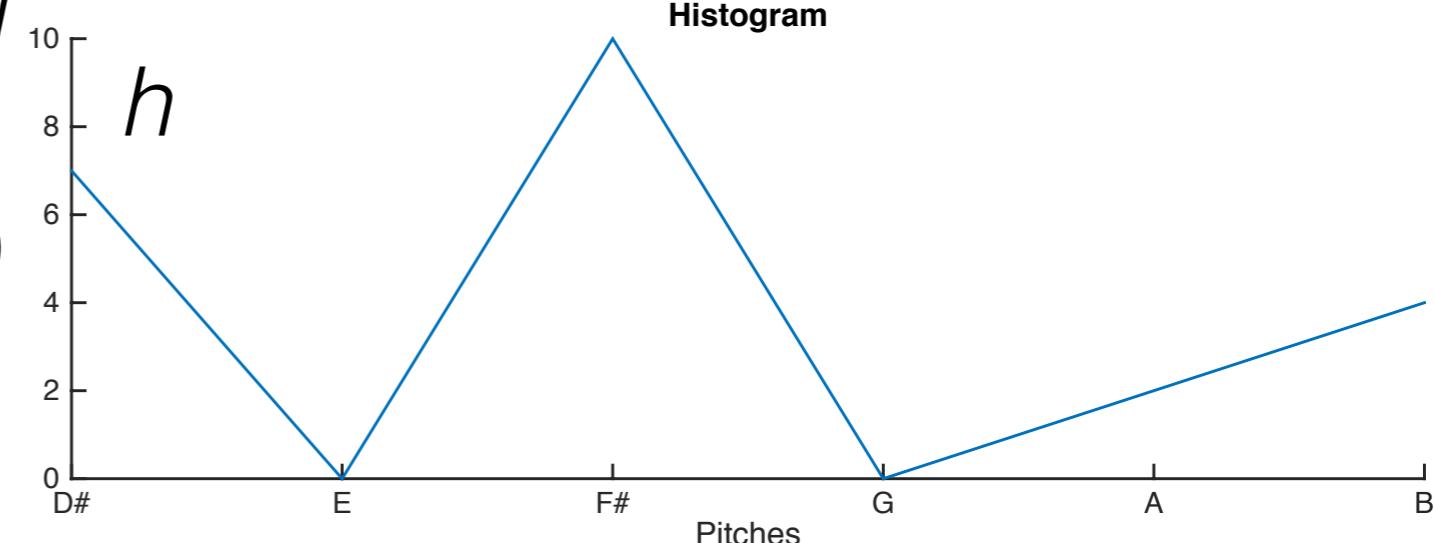
- $m$  is of class  
*mus.sequence*



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

- $s$  is of class *sig.signal*

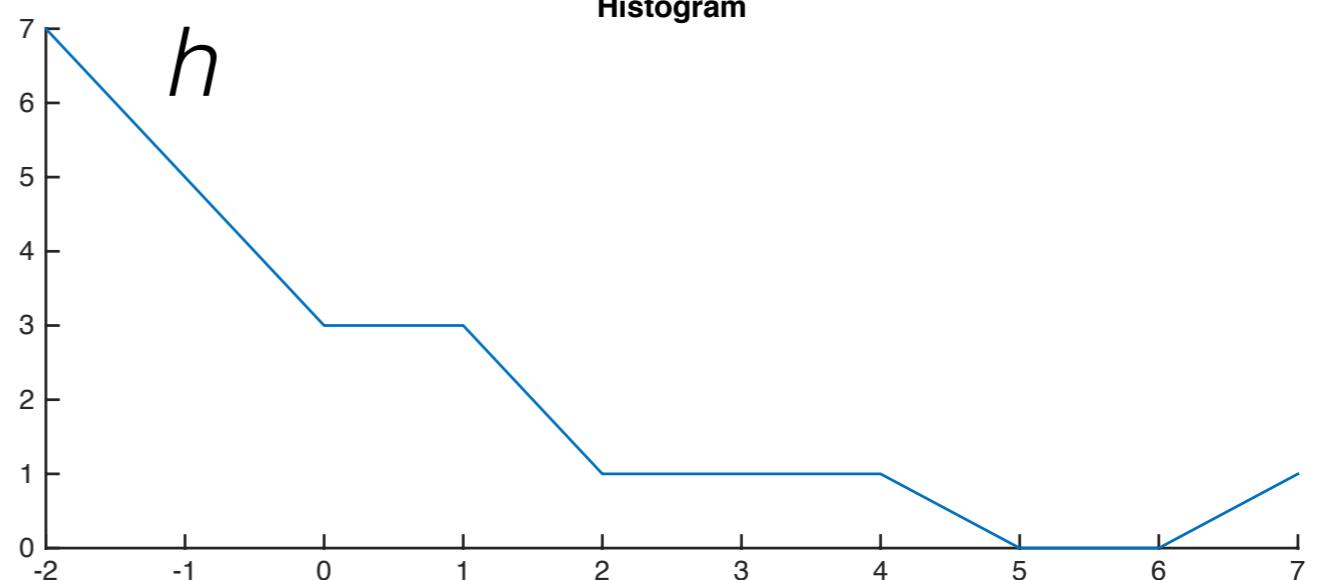
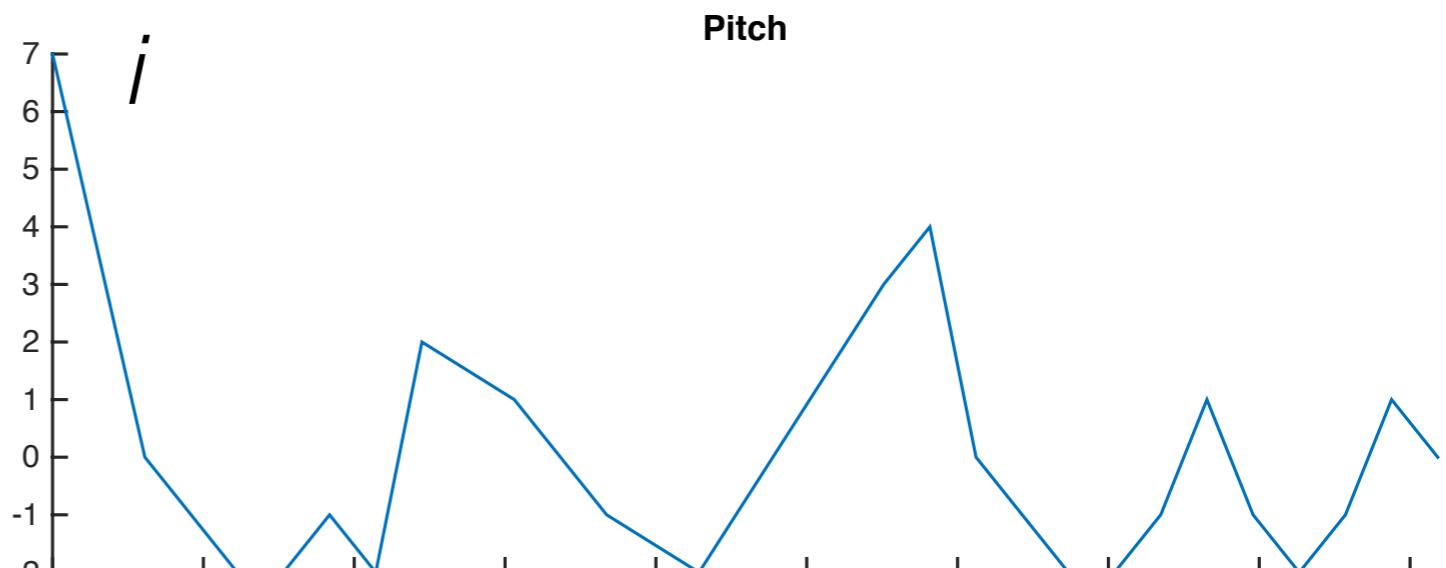
- $h = \textbf{\textit{mus.hist}}(s, \text{'Class'})$



# *mus.pitch('Inter')*

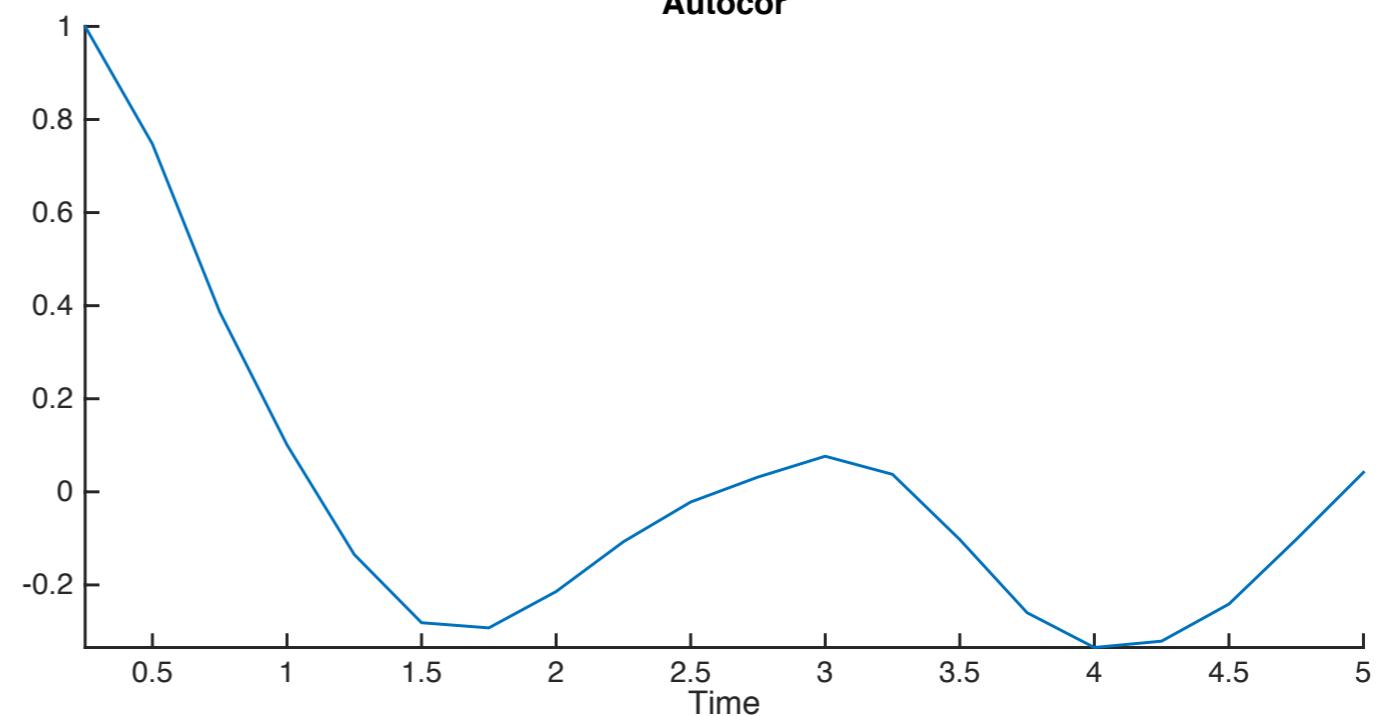
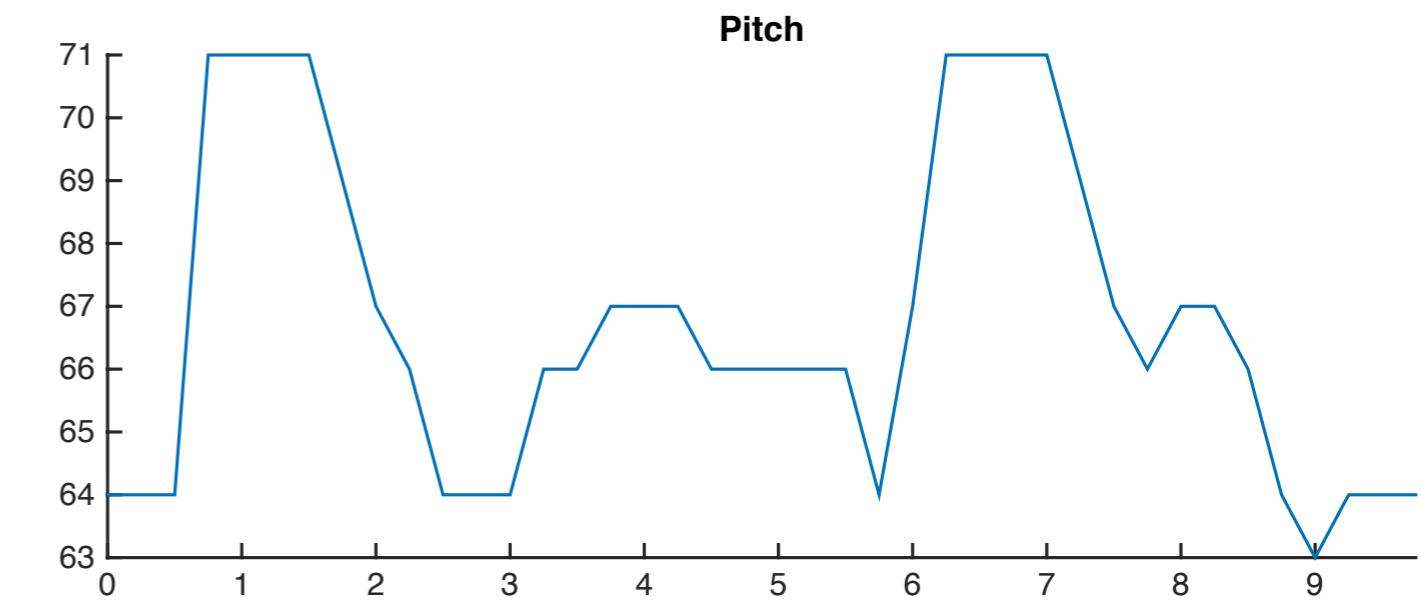
## pitch interval contour

- $m = \text{mus.score('myfile')}$
- $i = \text{mus.pitch}(m, \text{'Inter'})$
- $h = \text{mus.histo}(i)$
- $\text{mus.histo}(i, \text{'Sign'}, 0)$



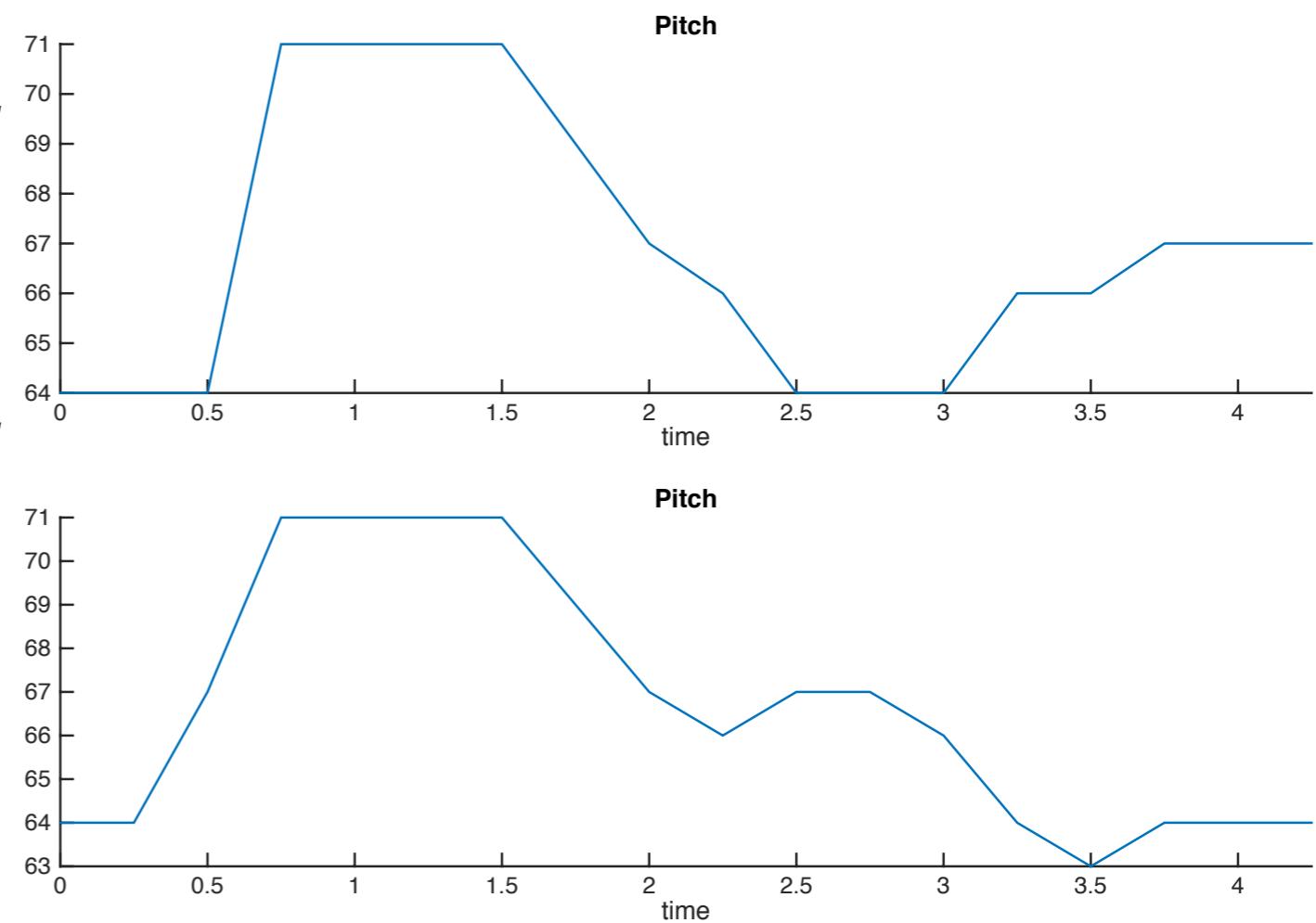
# *mus.pitch* pitch contour

- $m = \text{mus.score}(\text{'myfile'})$
- $c = \text{mus.pitch}(m, \text{'Sampling'}, .25)$
- $\text{mus.autocor}(c)$



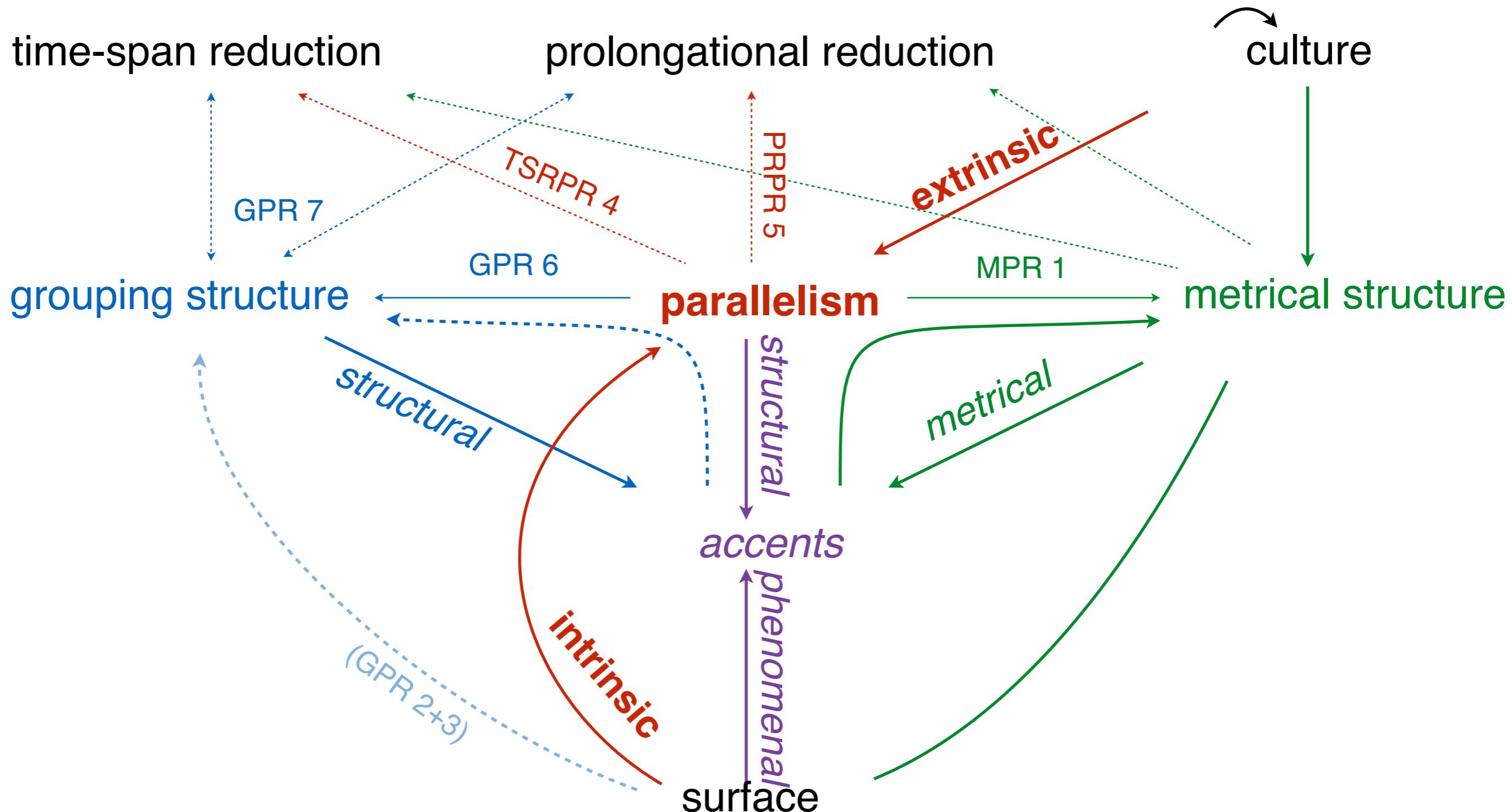
# *mus.pitch* pitch contour

- $m1 = \text{mus.score}(\text{'myfile'}, \text{'EndTime'}, 5)$
- $m2 = \text{mus.score}(\text{'myfile'}, \text{'StartTime'}, 5)$
- $p1 = \text{mus.pitch}(m1, \text{'Sampling'}, .25)$
- $p2 = \text{mus.pitch}(m2, \text{'Sampling'}, .25)$



- ***sig.dist(p1, p2, 'Cosine')***

# 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

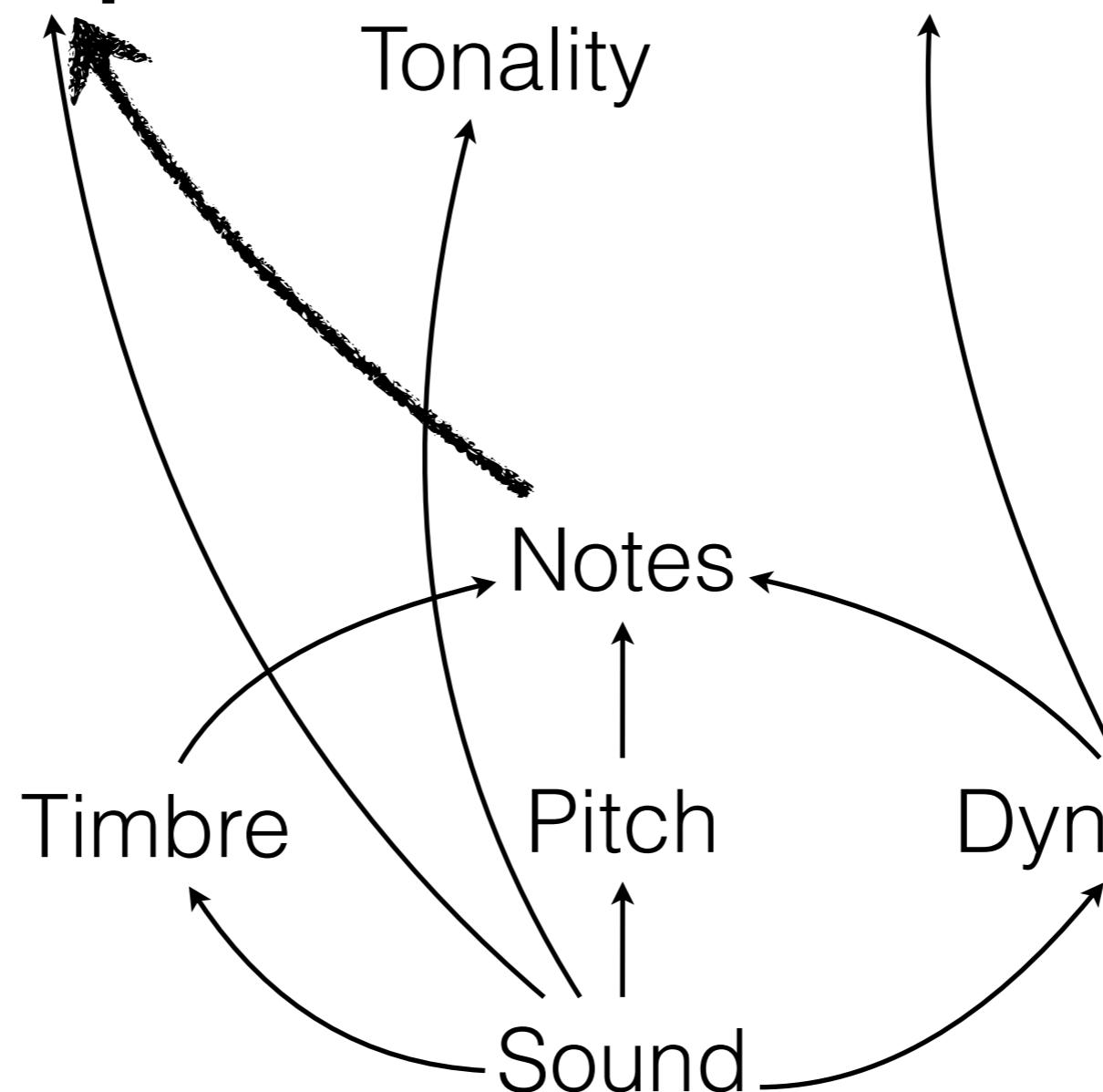
Notes

Dynamics

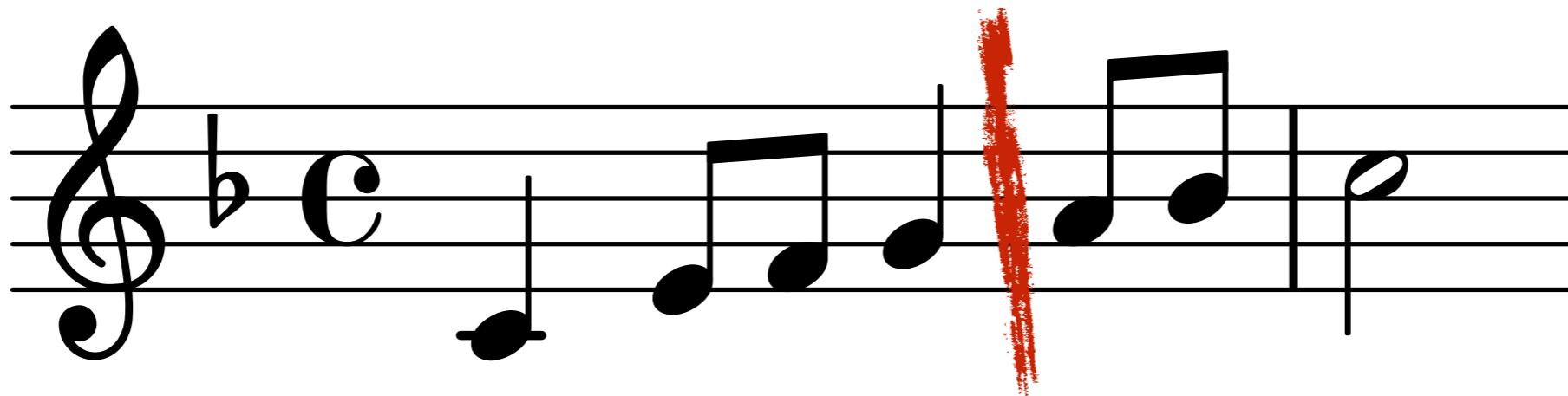
Pitch

Sound

*Audio level*



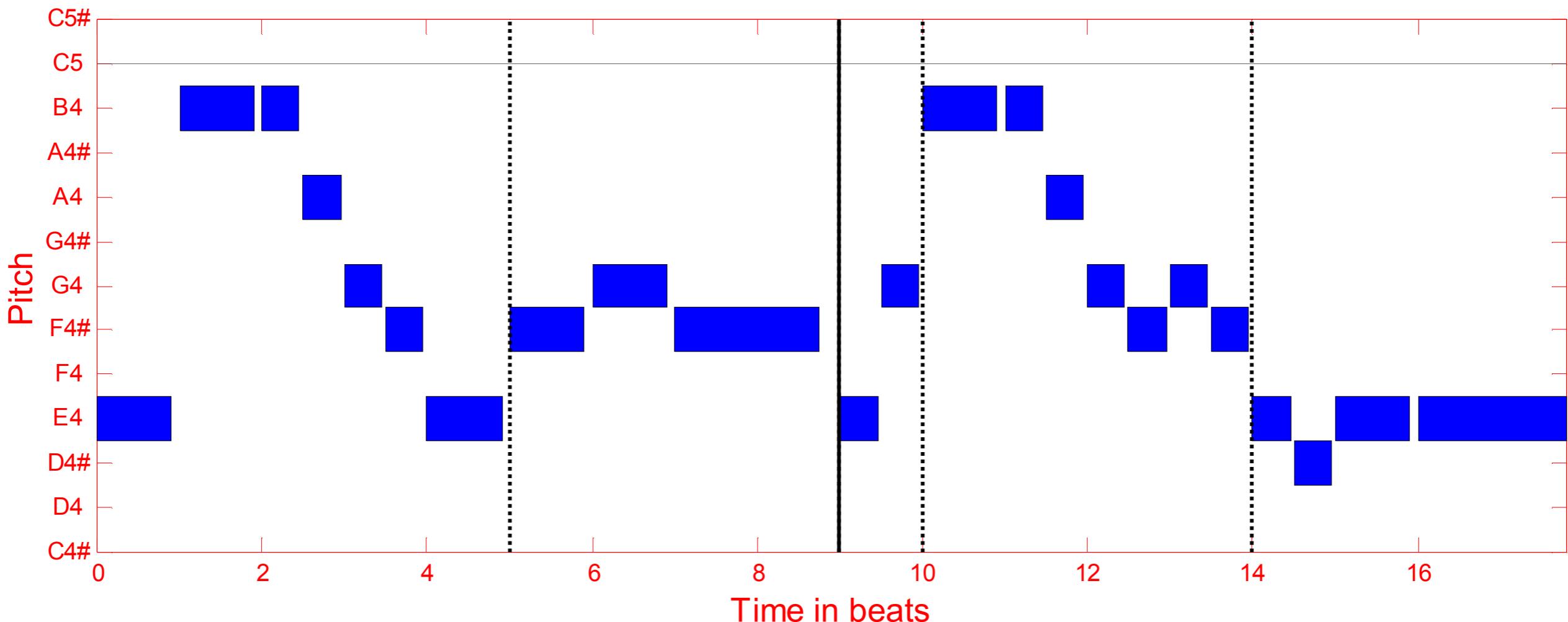
# *mus.score(..., 'Segment')* local segmentation



- Tenney & Polansky, 1980
- LBDM (Cambouropoulos, 1997)
- Bod, 2002

# *mus.score(..., 'Segment')*

## local segmentation



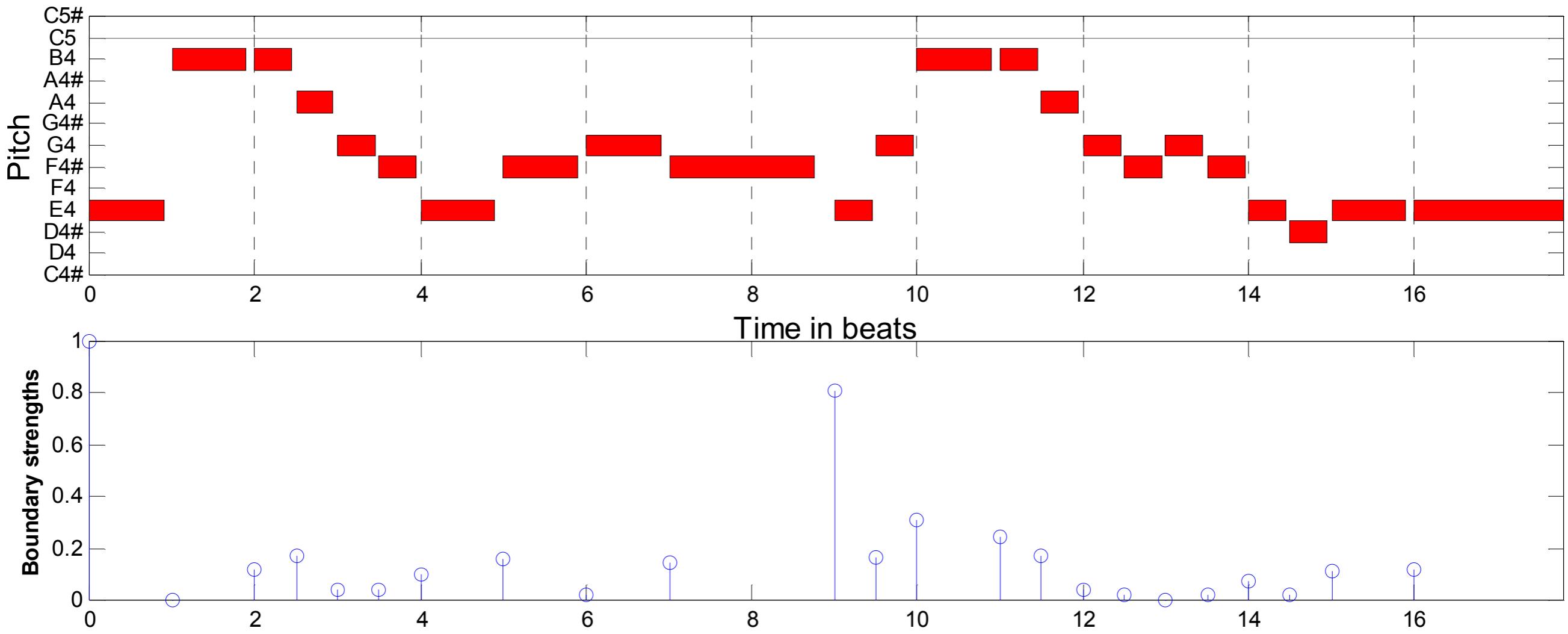
- *mus.score(..., 'Segment', 'TP')*

(Tenney & Polansky, 1980)

from *MIDI Toolbox*

# *mus.score(..., 'Segment')*

## local segmentation



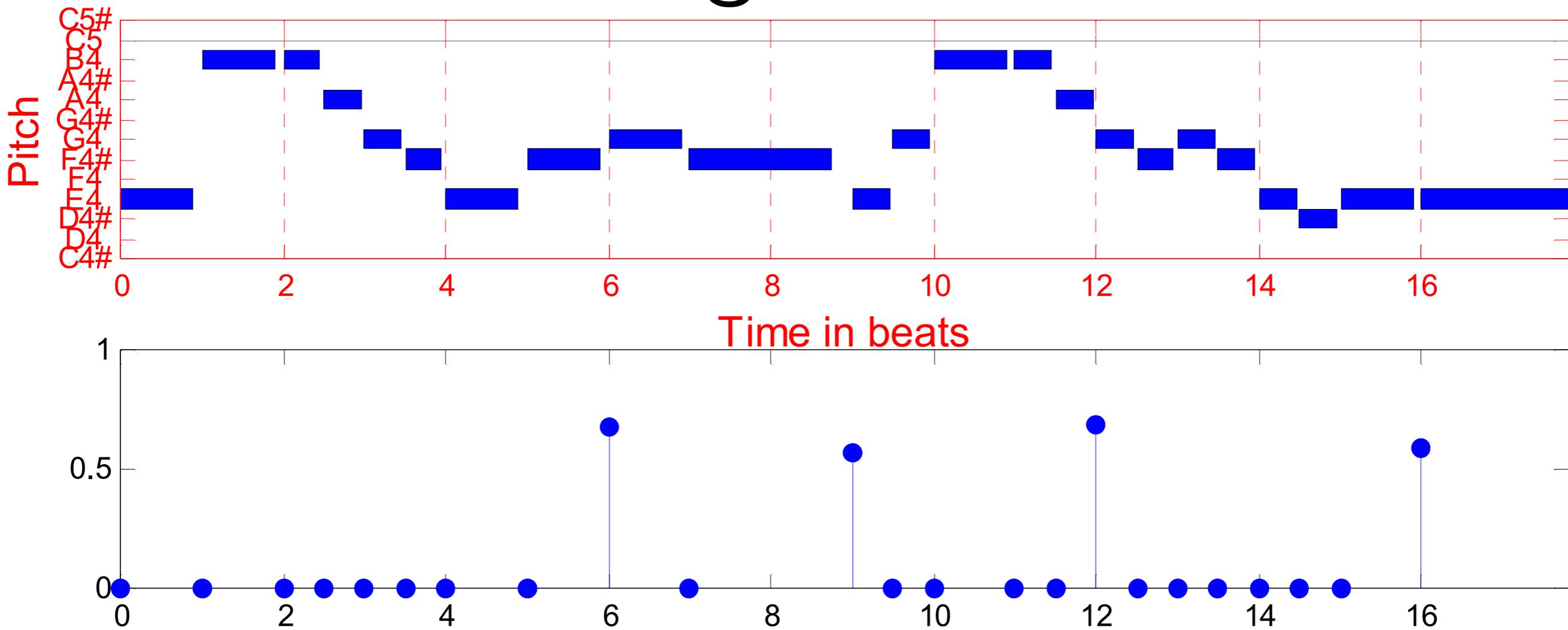
- *mus.score(..., 'Segment', 'LBDM')*

(Cambouropoulos, 1997)

from *MIDI Toolbox*

# *mus.score(..., 'Segment')*

## local segmentation

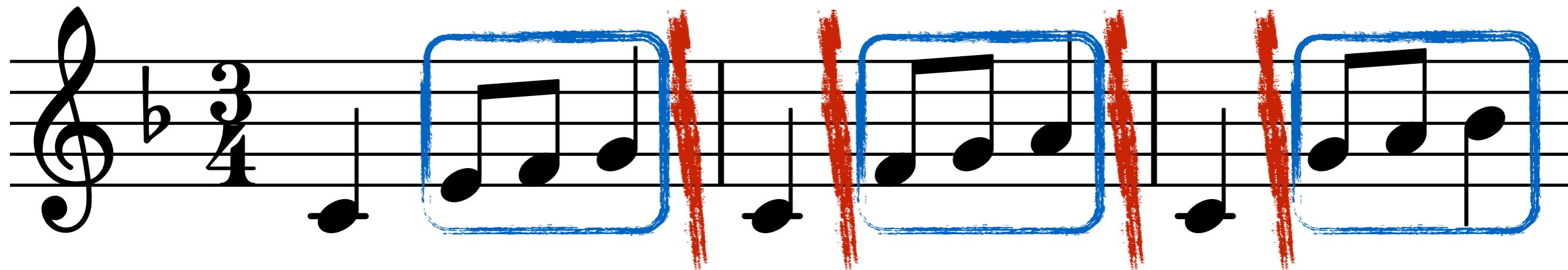


- *mus.score(..., 'Segment', 'Essen')*

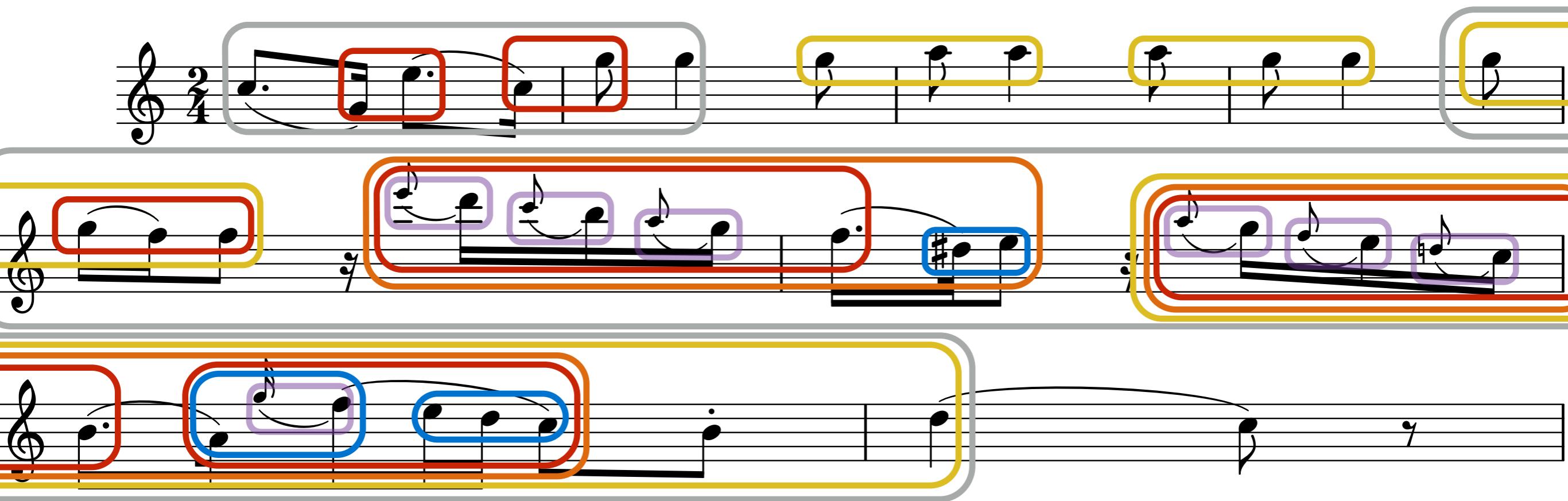
(Bod, 2002)

from *MIDI Toolbox*

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

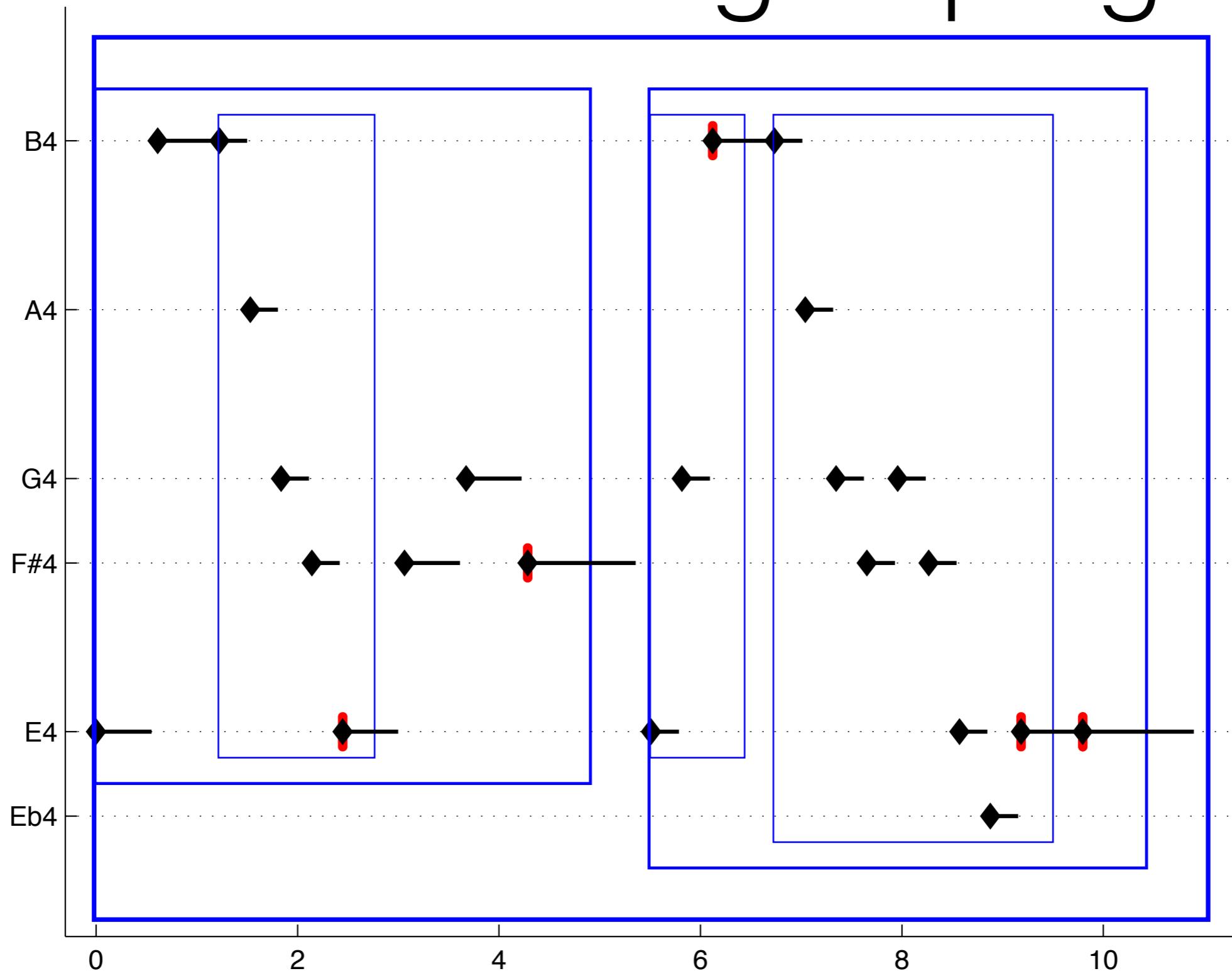


# *mus.score(..., 'Group')* hierarchical grouping



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

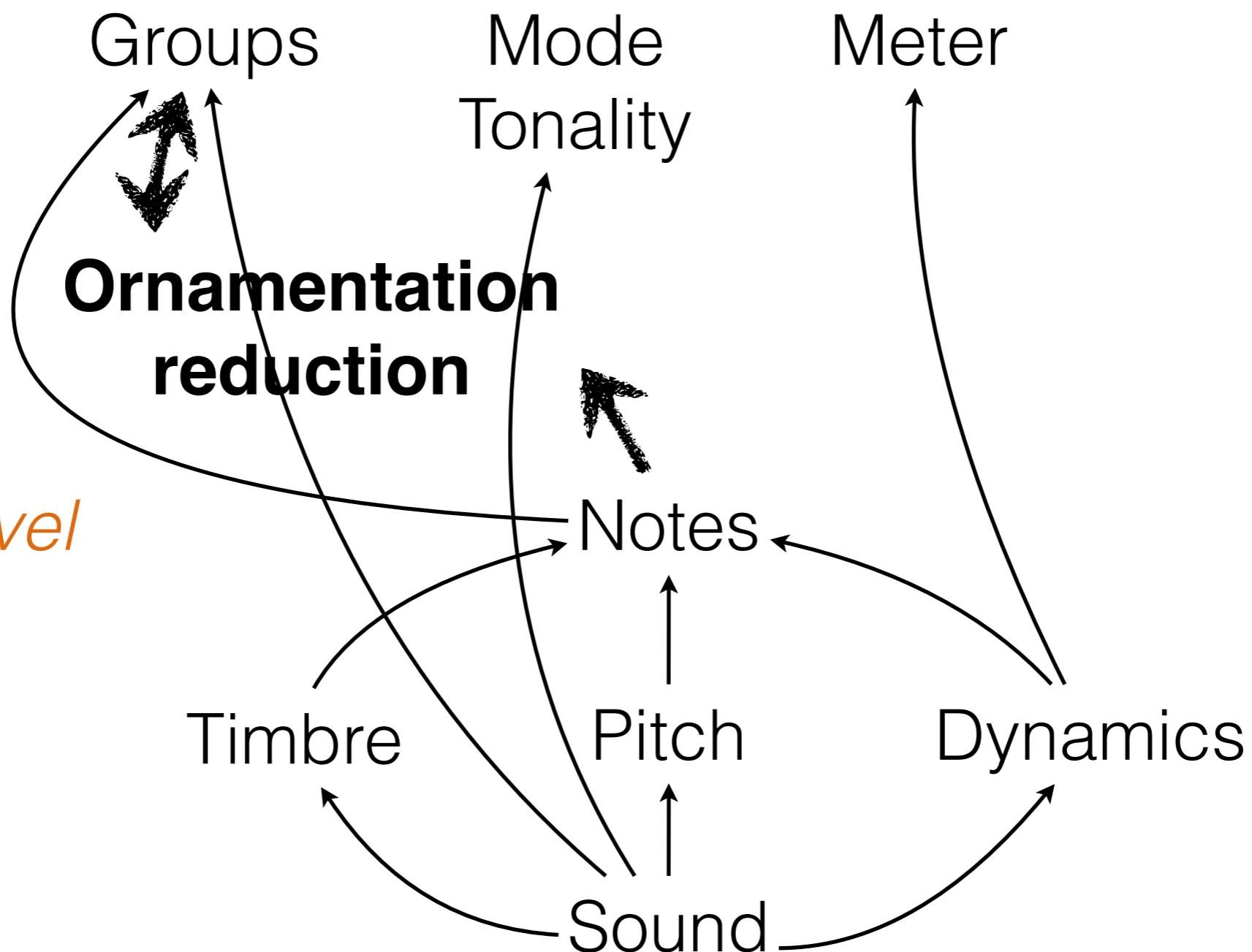
*mus.score(..., 'Group')*  
hierarchical grouping



*Structural  
levels*

*Symbolic level*

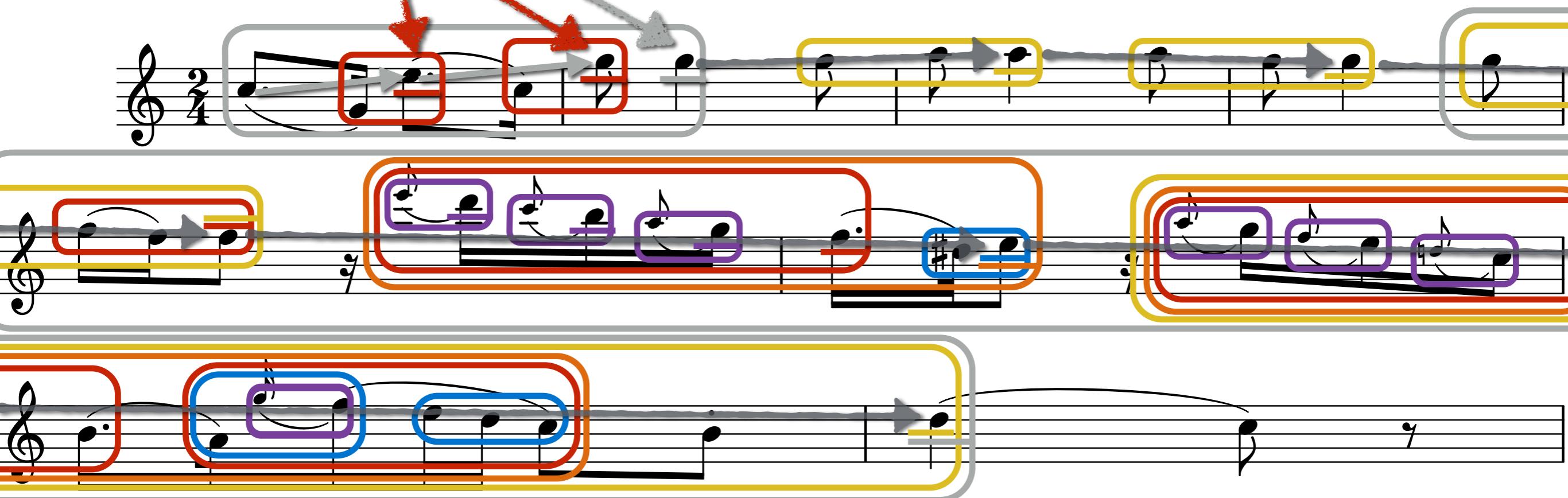
*Audio level*



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

head

(Lerdahl & Jackendoff)

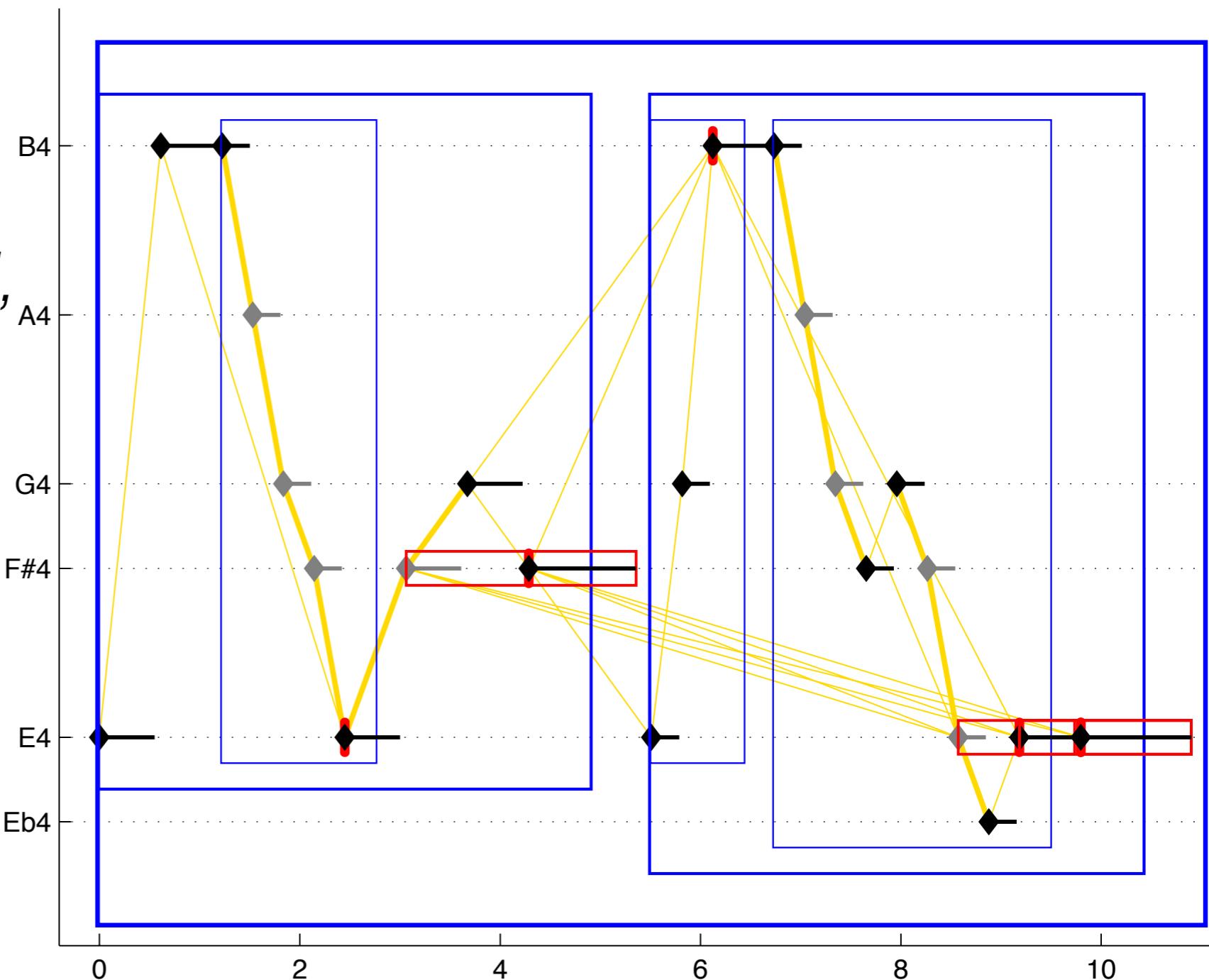


Mozart, Variation XI on “Ah, vous dirai-je maman”, K.265/300e

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

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

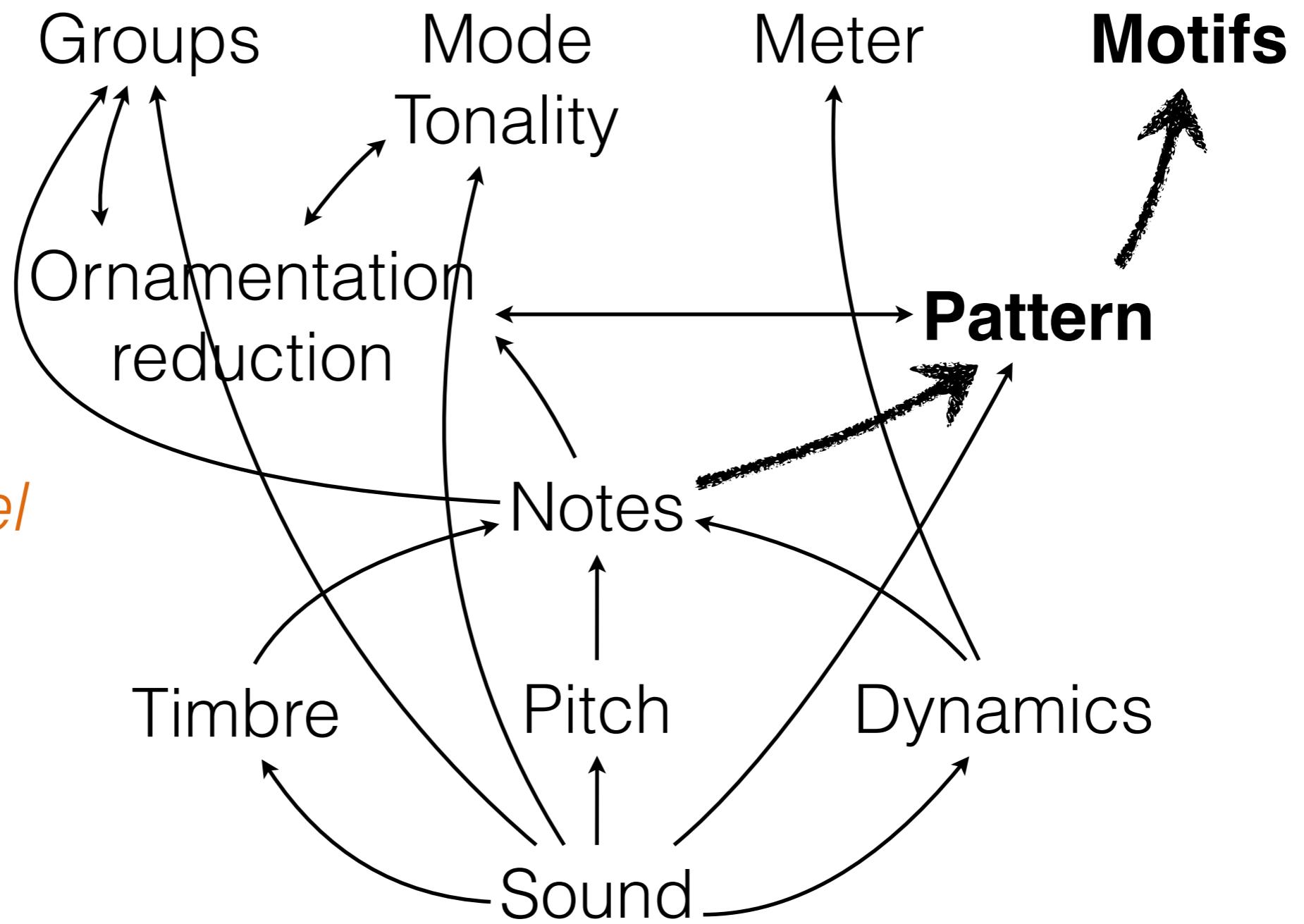
Construction of a  
*syntagmatic network*



*Structural  
levels*

*Symbolic level*

*Audio level*



# motivic pattern analysis

Bach Invention in D minor

Kresky (1977) analysis

Name	Kresky's interpretation
<b>A</b>	Opening motive
<b>A'</b>	First sequence unit
<b>B</b>	Accompanying figure
<b>a</b>	Motive shape
<b>a1</b>	Reversed motive shape
<b>a2=c</b>	Triadic structure
<b>a3=a3'</b>	Retarded scalar climb
<b>a4=C</b>	Bass line
<b>a5</b>	Seconde sequence unit
<b>a6</b>	(not considered)
<b>b</b>	Three-note scale
<b>b'</b>	Identifying a5 with a3
<b>b''</b>	Same, transposed
<b>b'''</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 (two notes), B (one note), c (two notes), and D (one note). Measure 2 continues with a similar pattern: A (two notes), a (one note), b (two notes). The bass staff provides harmonic support.

Computer analysis of Bach's Invention in D minor, measures 5-6. The analysis highlights motivic patterns across both staves. In the treble staff, motif A is identified at measure 5, followed by variations a3' and a3'' at measure 6. Motif E appears in the bass staff at measure 6. The bass staff also features motif C, which is repeated at 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, motifs a5, b', and b'' are identified at measure 10. Motif E is present in the bass staff. At measure 11, motifs a5, b'', and b'' are again highlighted. The bass staff also features motif A' at measure 11.

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

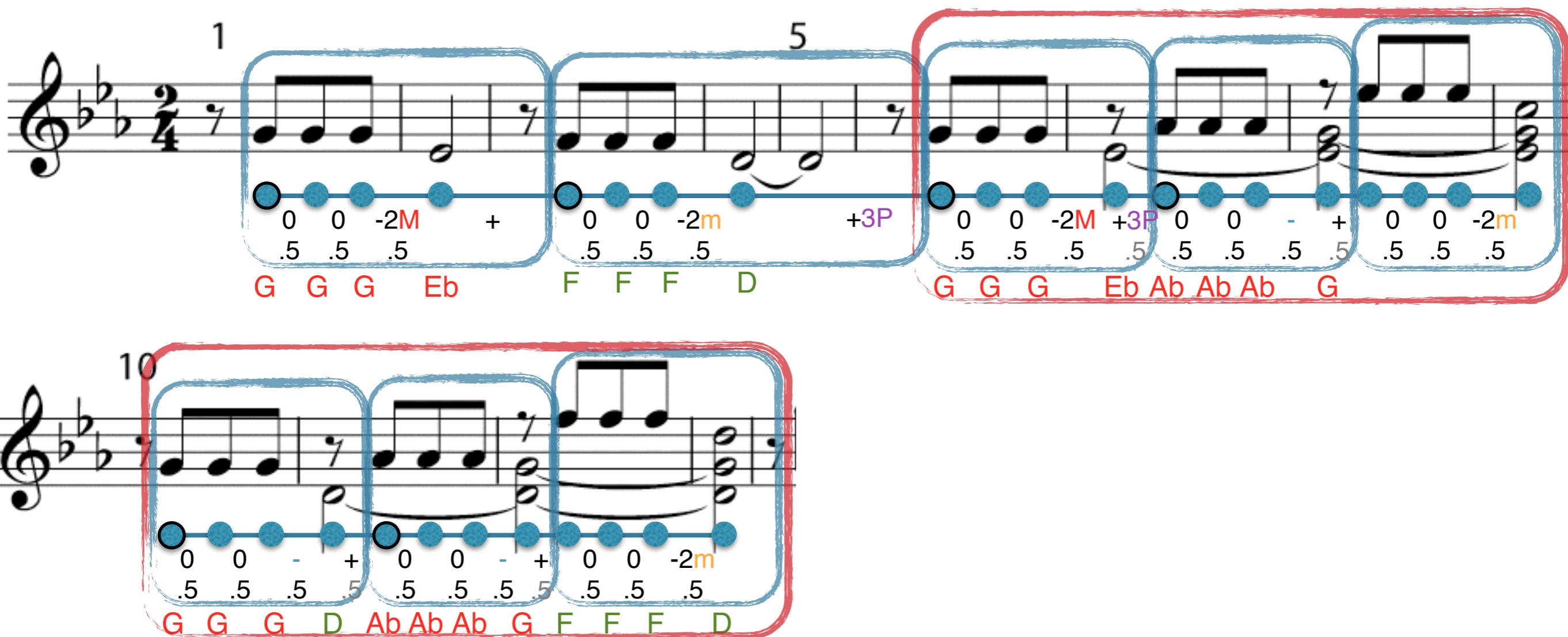
# motivic pattern analysis

## Bach Invention in D minor

Name	Kresky's interpretation	Computational description	
		Melodic	Rhythmic
<b>A</b>	Opening motive	(D,E,F,G,A,Bb,C#,Bb,A,G,F,E,F)	16th notes
<b>A'</b>	First sequence unit	(-2,+1,+1,+1,+1,-6,+6,-1,-1,-1,+1)	16th notes
<b>B</b>	Accompanying figure	(F,A,D+,-,+,+,-)	8th notes
<b>a</b>	Motive shape	(prefix of A)	
<b>a1</b>	Reversed motive shape	(not detected)	
<b>a2=c</b>	Triadic structure	(+,+,-)	8th notes
<b>a3=a3'</b>	Retarded scalar climb	(-2,+1,+1,+1,+1,-6)	16th notes
<b>a4=c</b>	Bass line	(+7,-5,+1,+1,+1,-6)	8th notes
<b>a5</b>	Seconde sequence unit	(+1,+1,-2,+1,+1,-6)	16th notes
<b>a6</b>	(not considered)	(-,+1,+1,+1)	16th notes
<b>b</b>	Three-note scale	(+1,+1)	16th notes
<b>b'</b>	Identifying a5 with a3	(D5,E5,F5)	16th notes
<b>b''</b>	Same, transposed	(C5,D5,E5)	16th notes
<b>b'''</b>	Three shape in bass line	(not detected)	
<b>D</b>	(not considered)	(-1,+1,+1,+1)	16th except 1st
<b>E</b>	(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 in two columns and three rows. The first column contains L1, M1, and U1; the second column contains L2, M2, and U3. Each staff is labeled with a name (L1, M1, U1, L2, M2, U3) above it. The music is written in common time with a key signature of one sharp (F#). The notation consists of vertical stems and horizontal beams connecting them. Measures are separated by vertical bar lines. Measure numbers are indicated by small numerals at the beginning of each measure. The first staff (L1) starts with a measure containing two eighth notes. The second staff (M1) starts with a measure containing a quarter note followed by a half note. The third staff (U1) starts with a measure containing a quarter note followed by a half note. The fourth staff (L2) starts with a measure containing a quarter note followed by a half note. The fifth staff (M2) starts with a measure containing a quarter note followed by a half note. The sixth staff (U3) starts with a measure containing a quarter note followed by a half note.

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

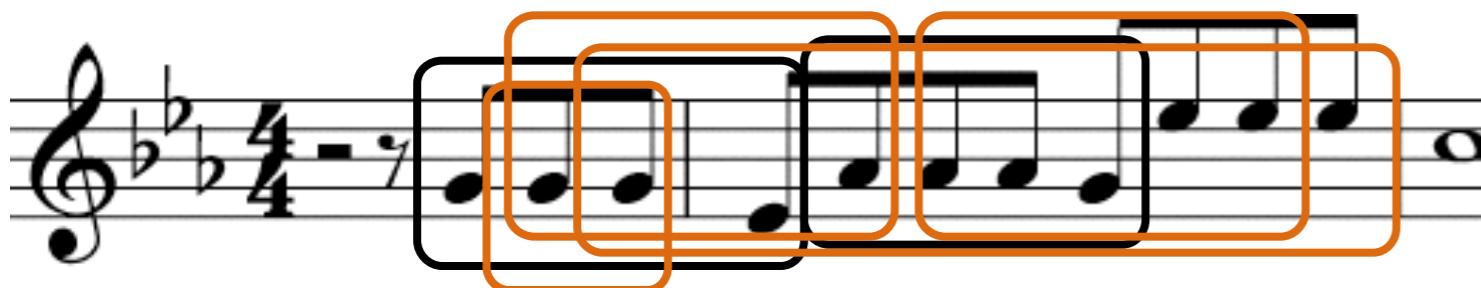
*mus.minr('beethoven', 'Motif')*



# structure complexity



# Pattern extraction



# Pattern selection (longest, frequent, ...)



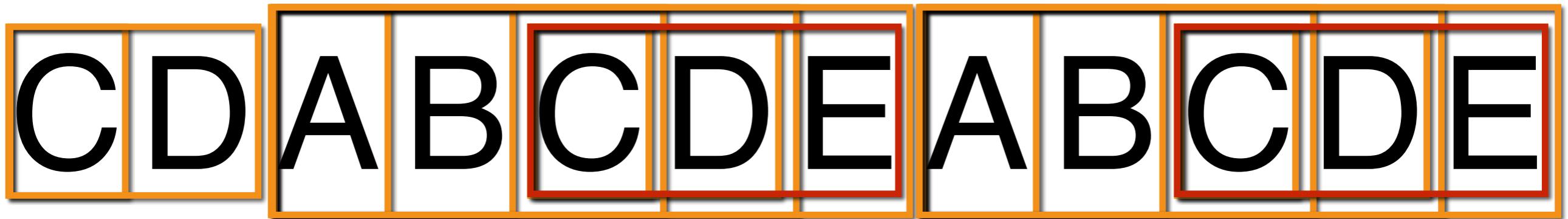
- Large set of **unrelevant structures** during extraction phase
    - cannot be computed extensively
  - Imperfections of the selection phase:
    - expected patterns accidentally deleted
    - insufficiently selective
  - Improvement of the extraction process

# closed patterns



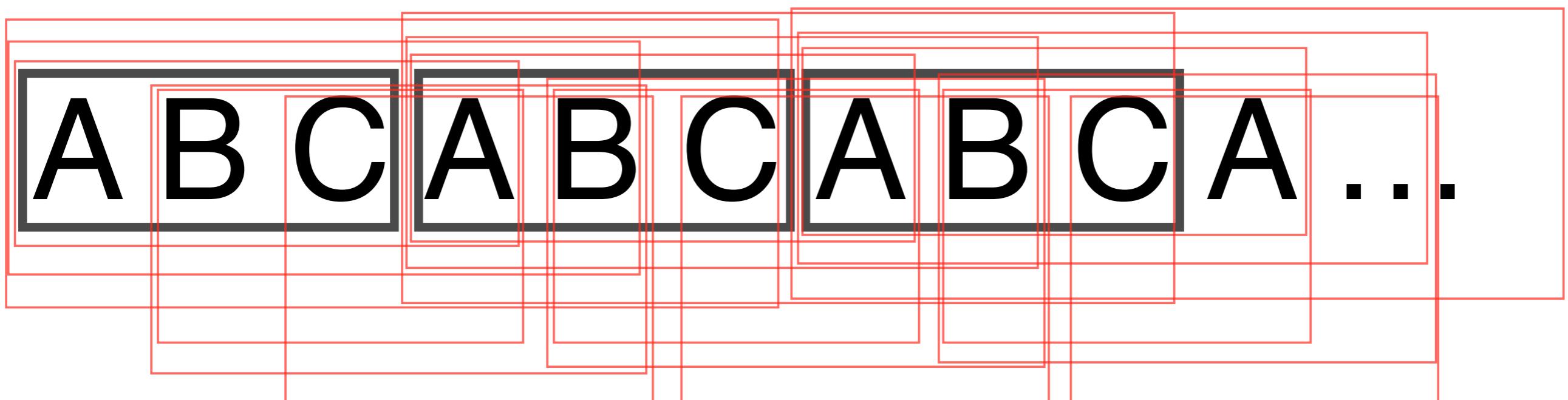
Complete and compact description of pattern repetition

# incremental approach



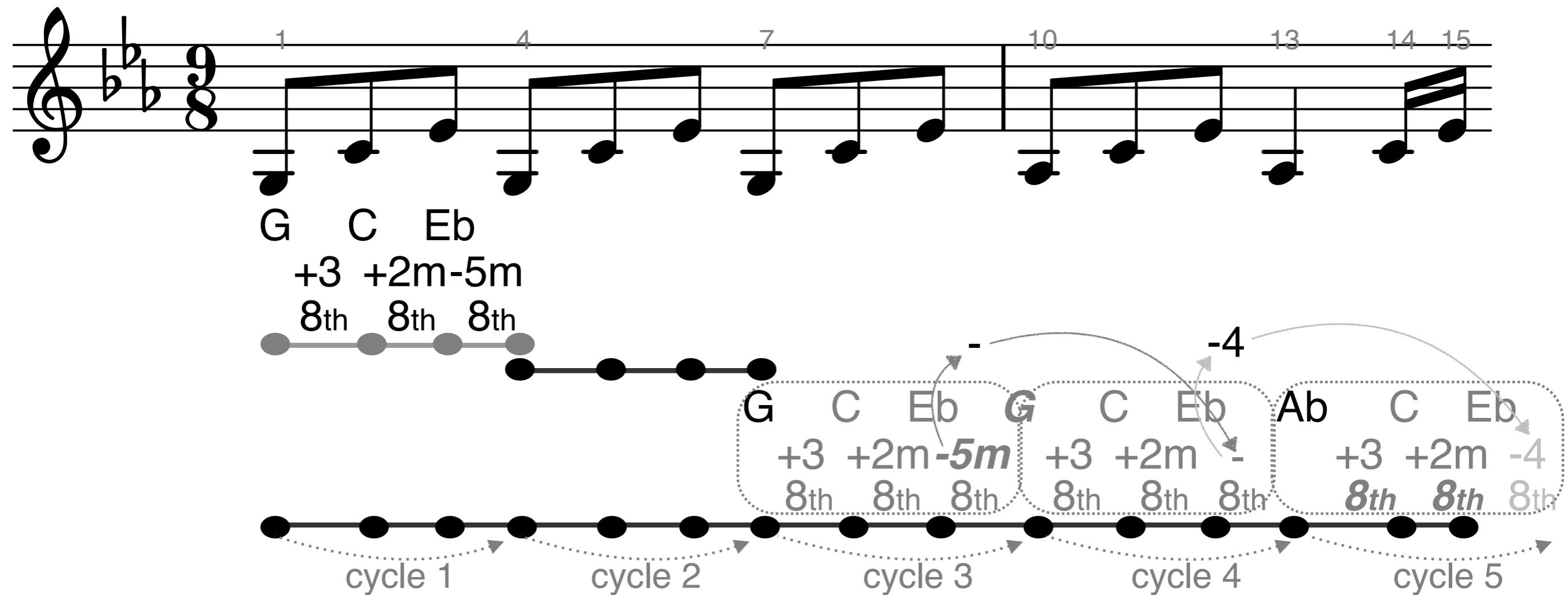
- Progressively analysing music through one single pass
- Controls structural complexity in a way similar to the way listeners perceive music.

# pattern cyclicity



Cambouropoulos, 1998

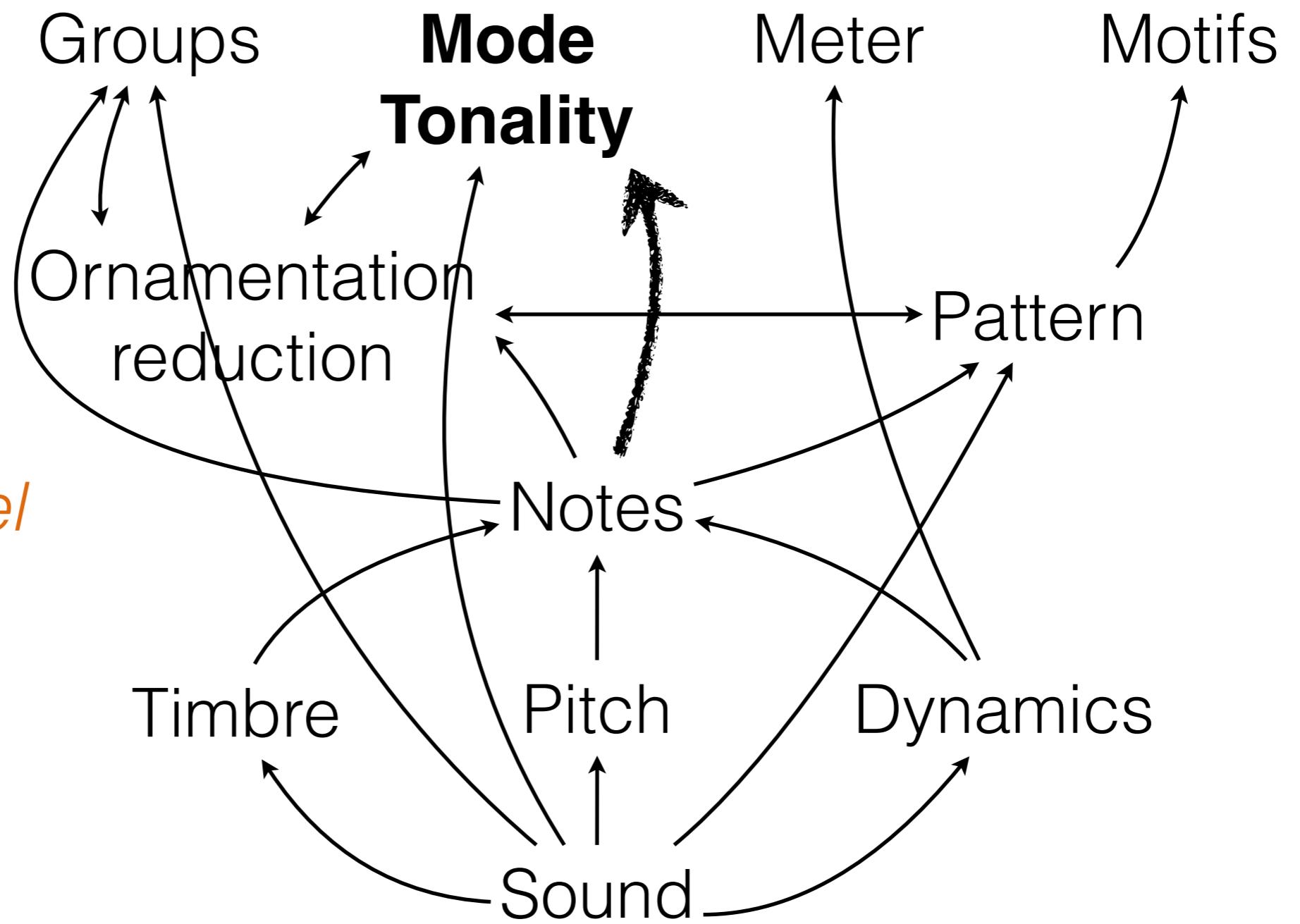
# heterogeneous pattern cycles



*Structural  
levels*

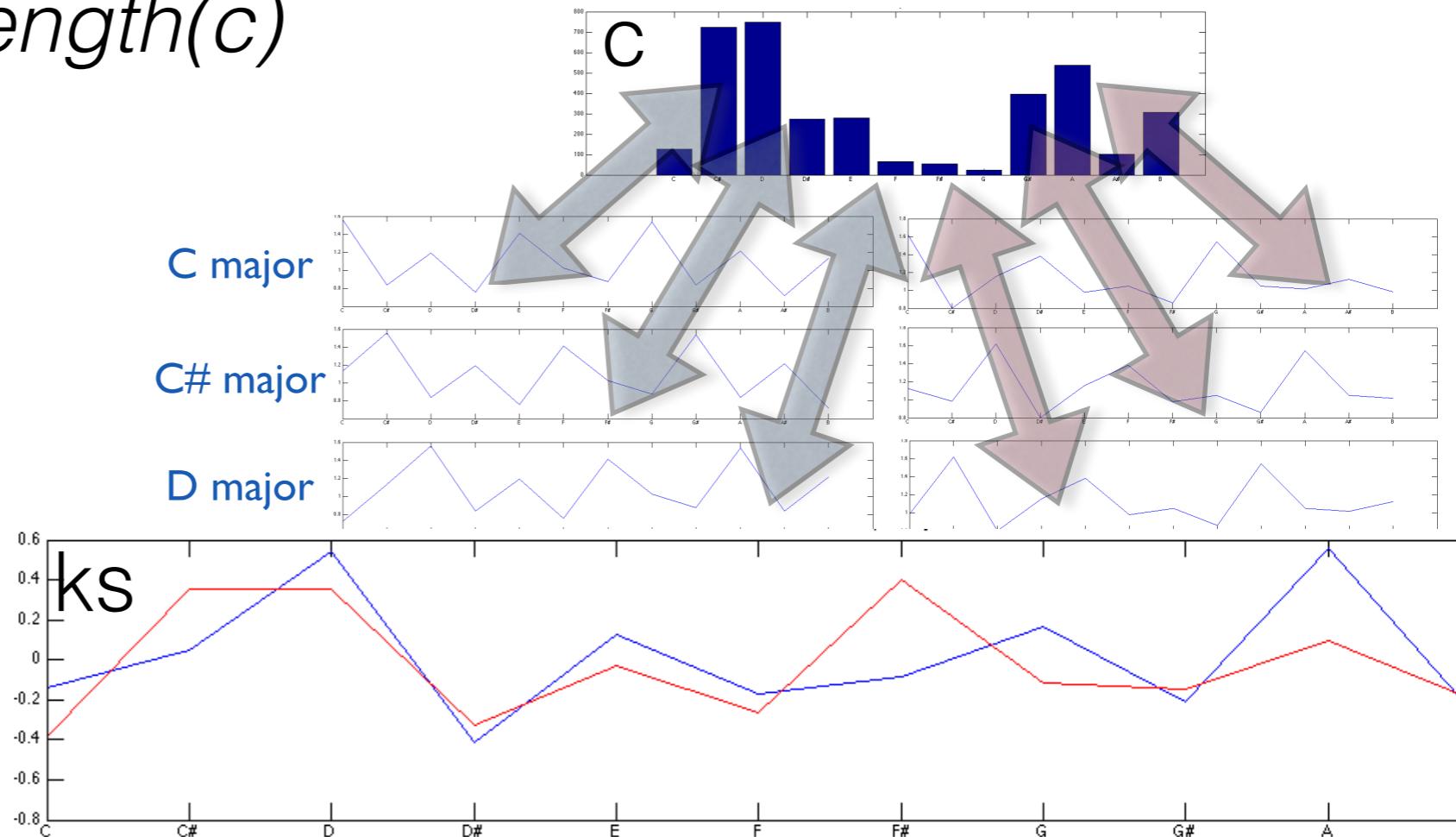
*Symbolic level*

*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 emphasise chord changes
  - etc.

# Score-level m/tonal analysis

CM

CM

1 2 4

p

legato.

3

CM

Dm

1 3 5

II

3

GM

V

3 5

2 cresc.

1

3

CM

1

3

Am

VI

f

1

3

DM

I

1 2 5

p

1

3

GM

V

1 3

f

1

3

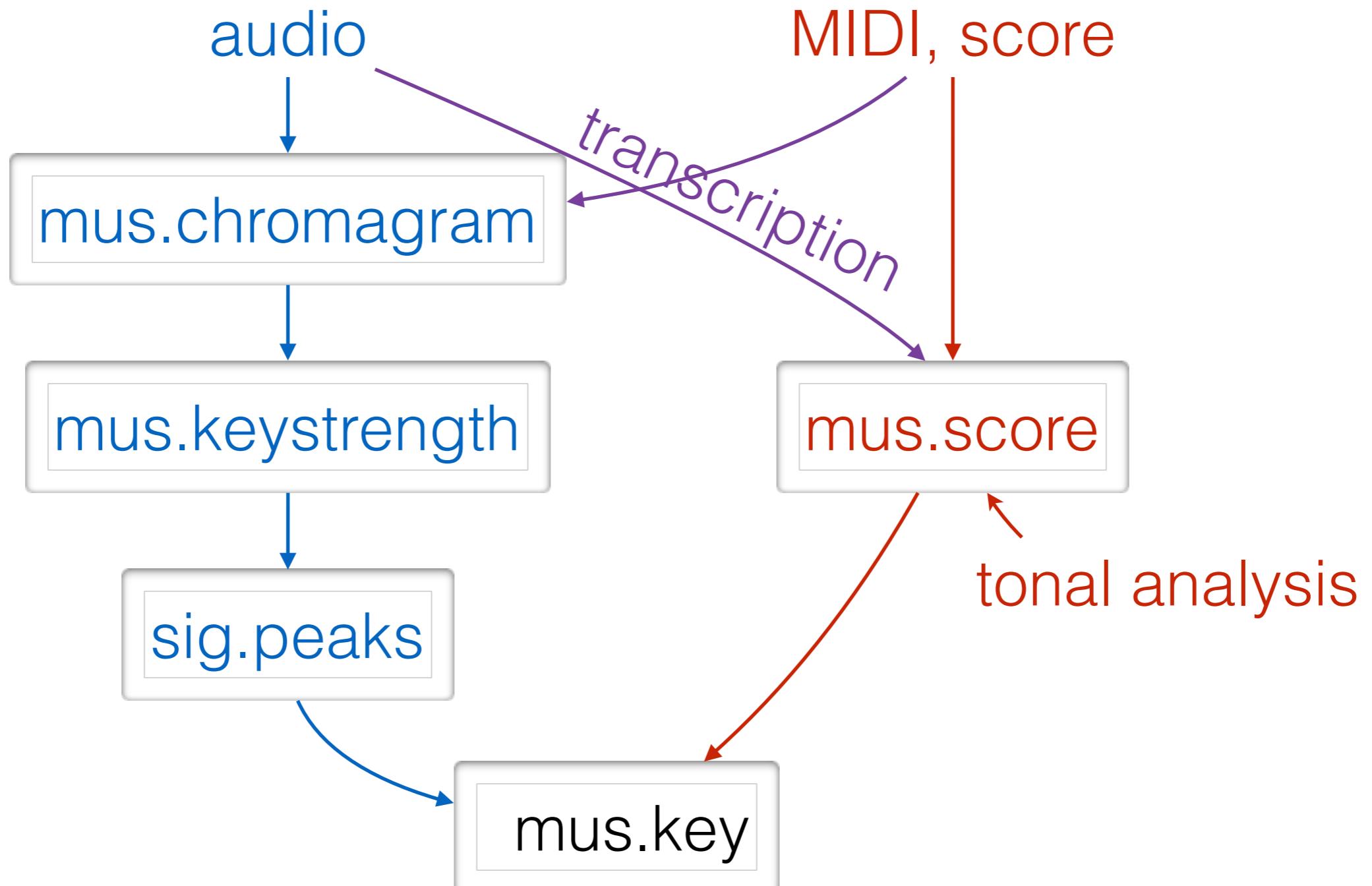
CM

I

1

3

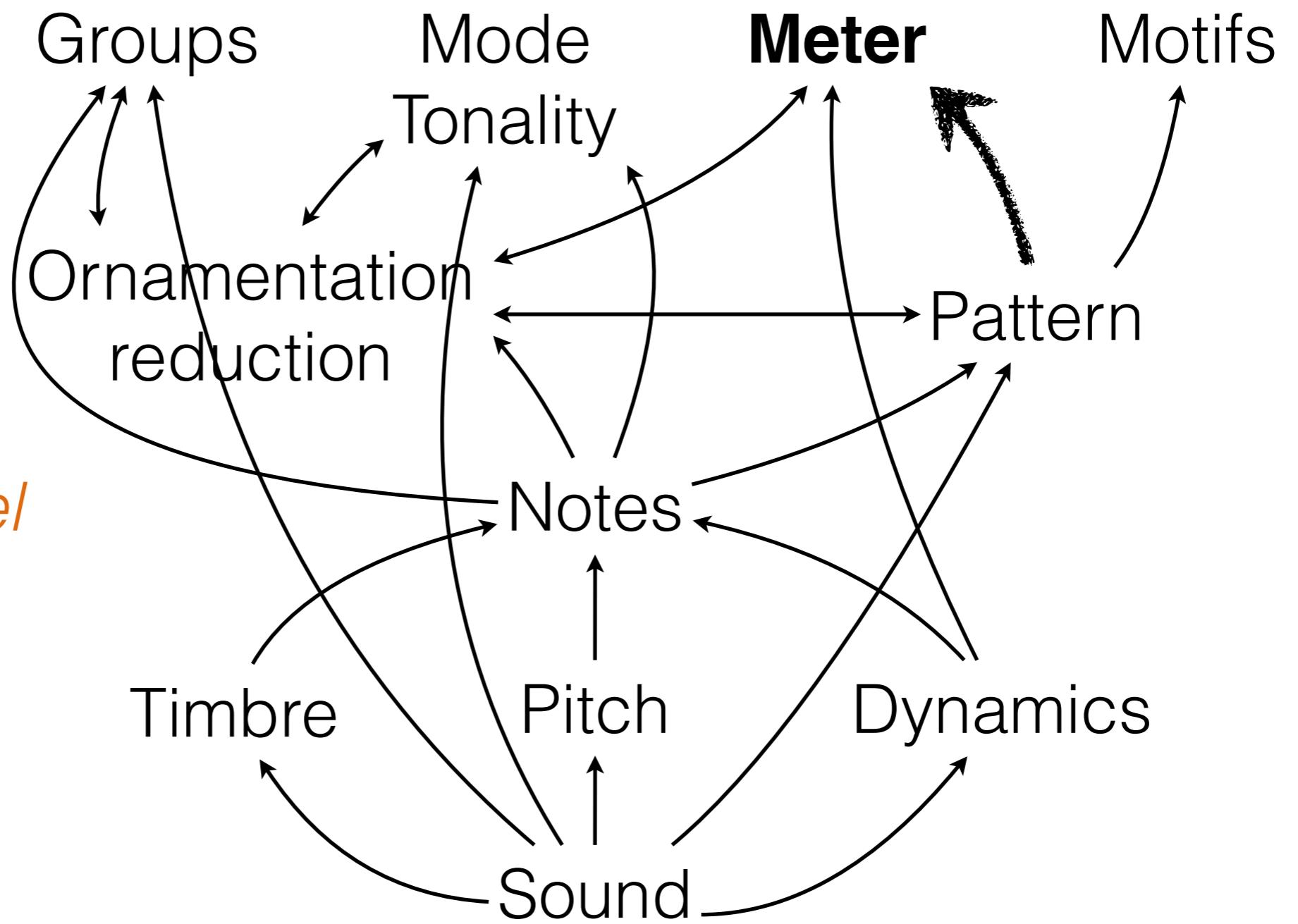
# Audio / symbolic



*Structural  
levels*

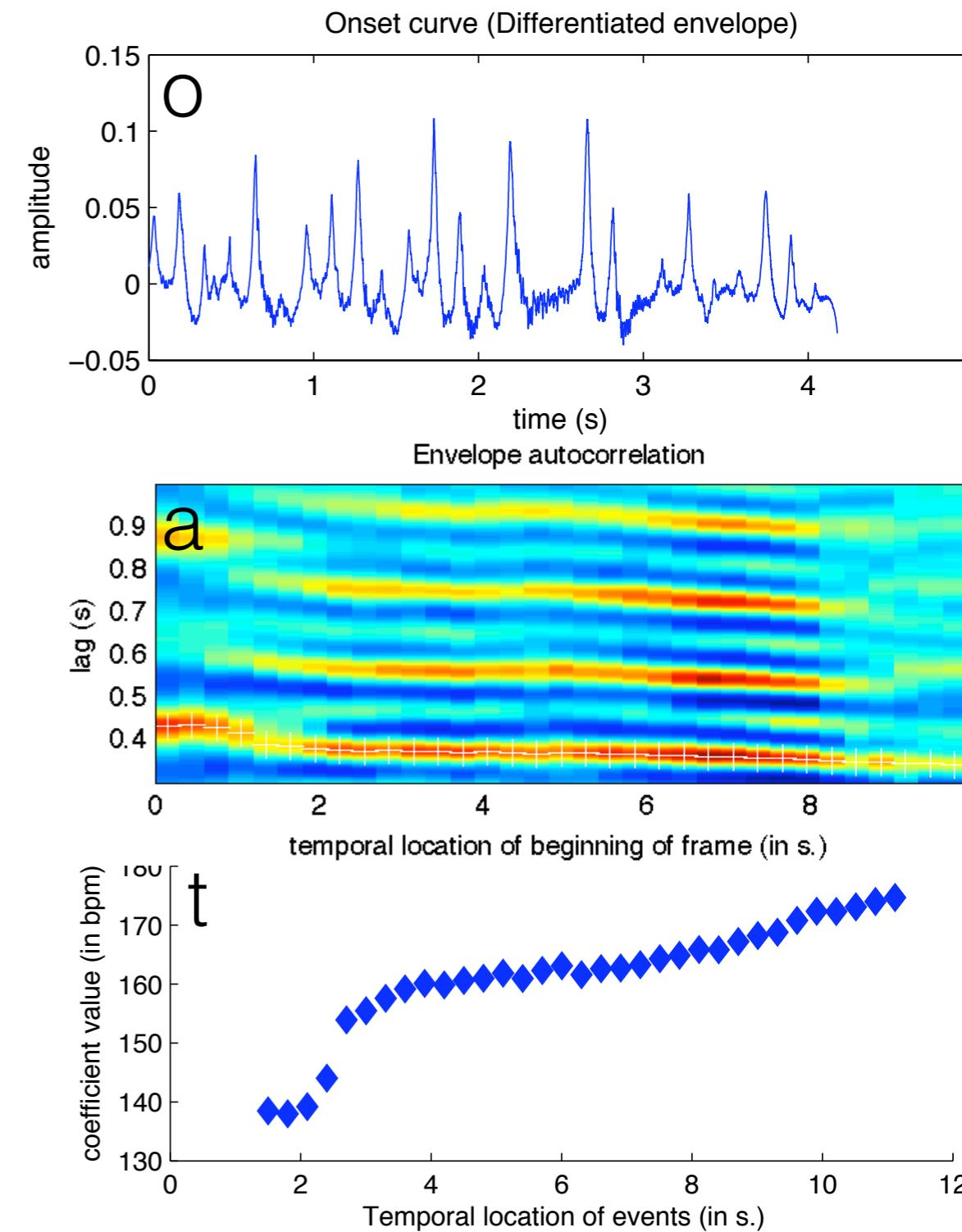
*Symbolic level*

*Audio level*

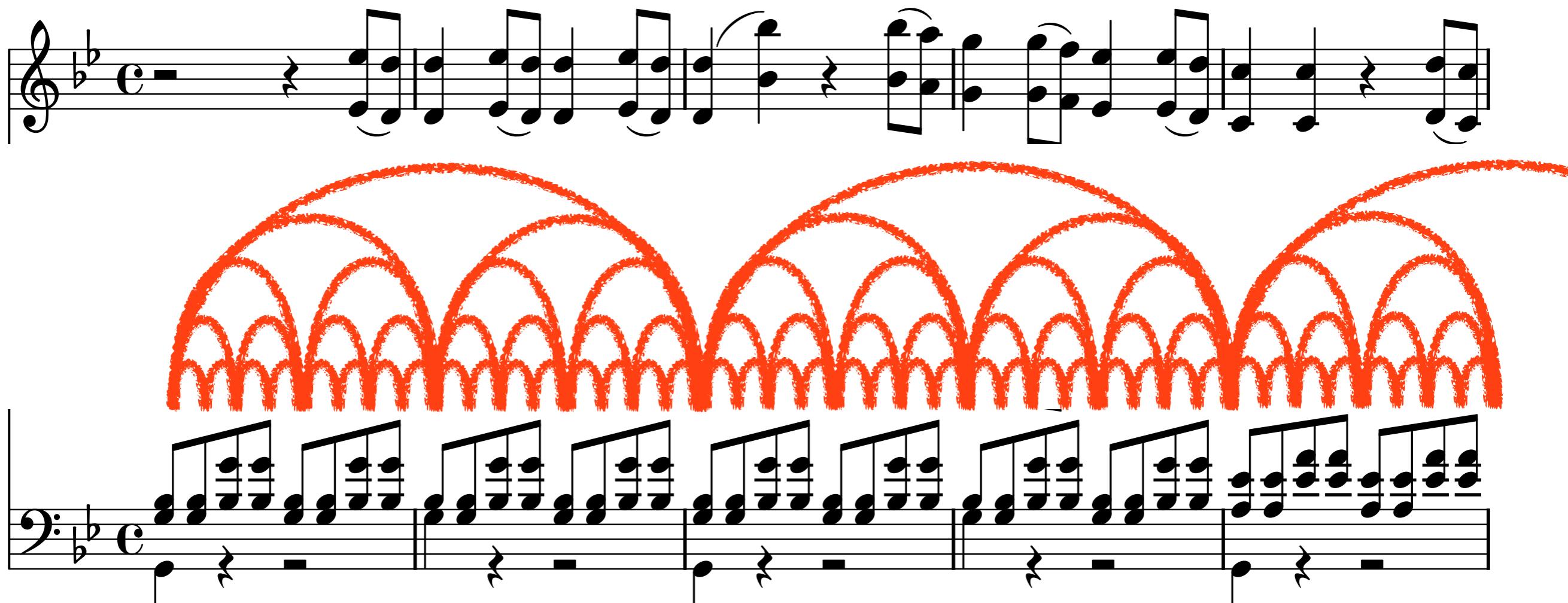


# Statistical metrical analysis

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

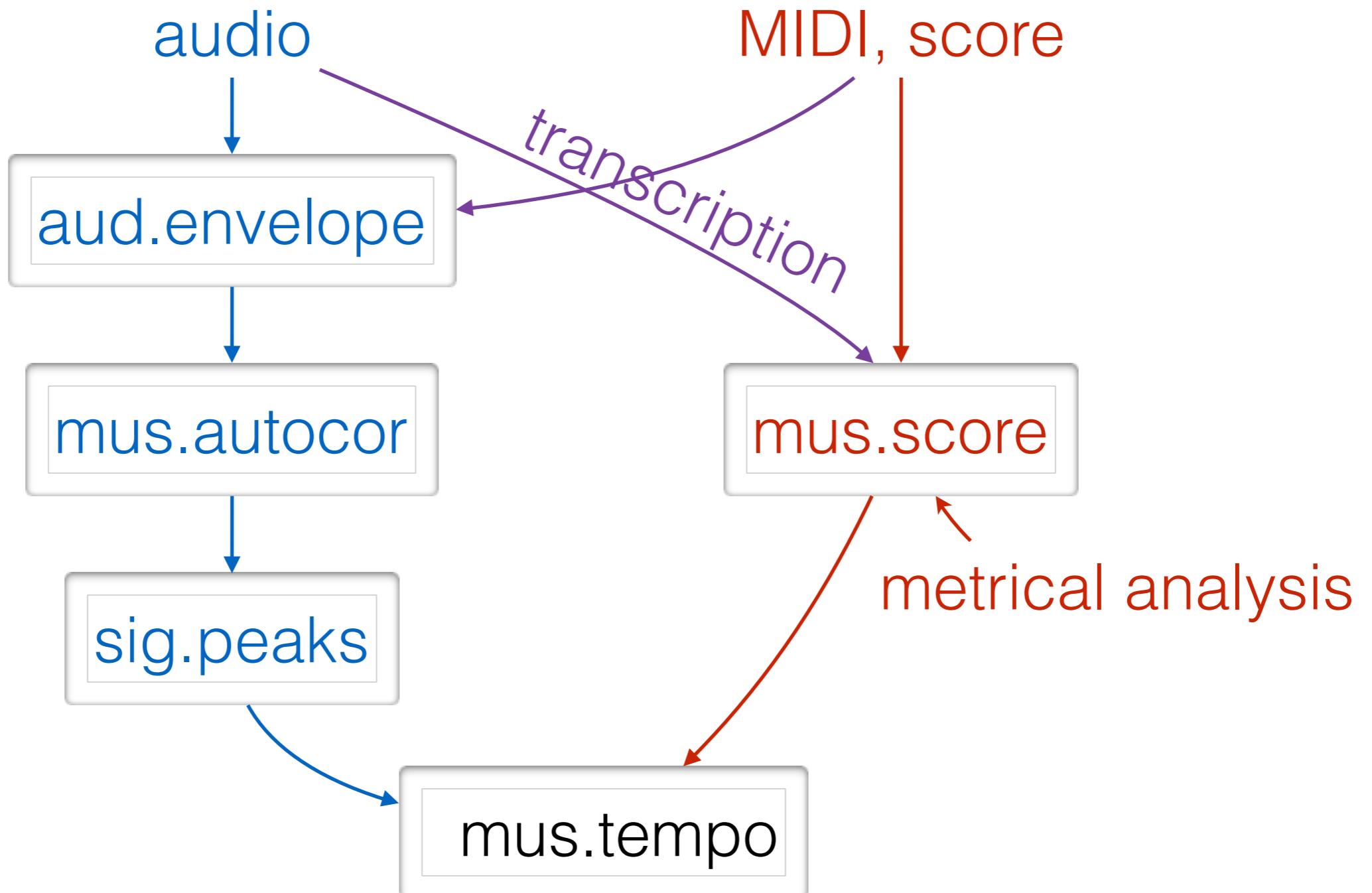


# Score-level metrical analysis

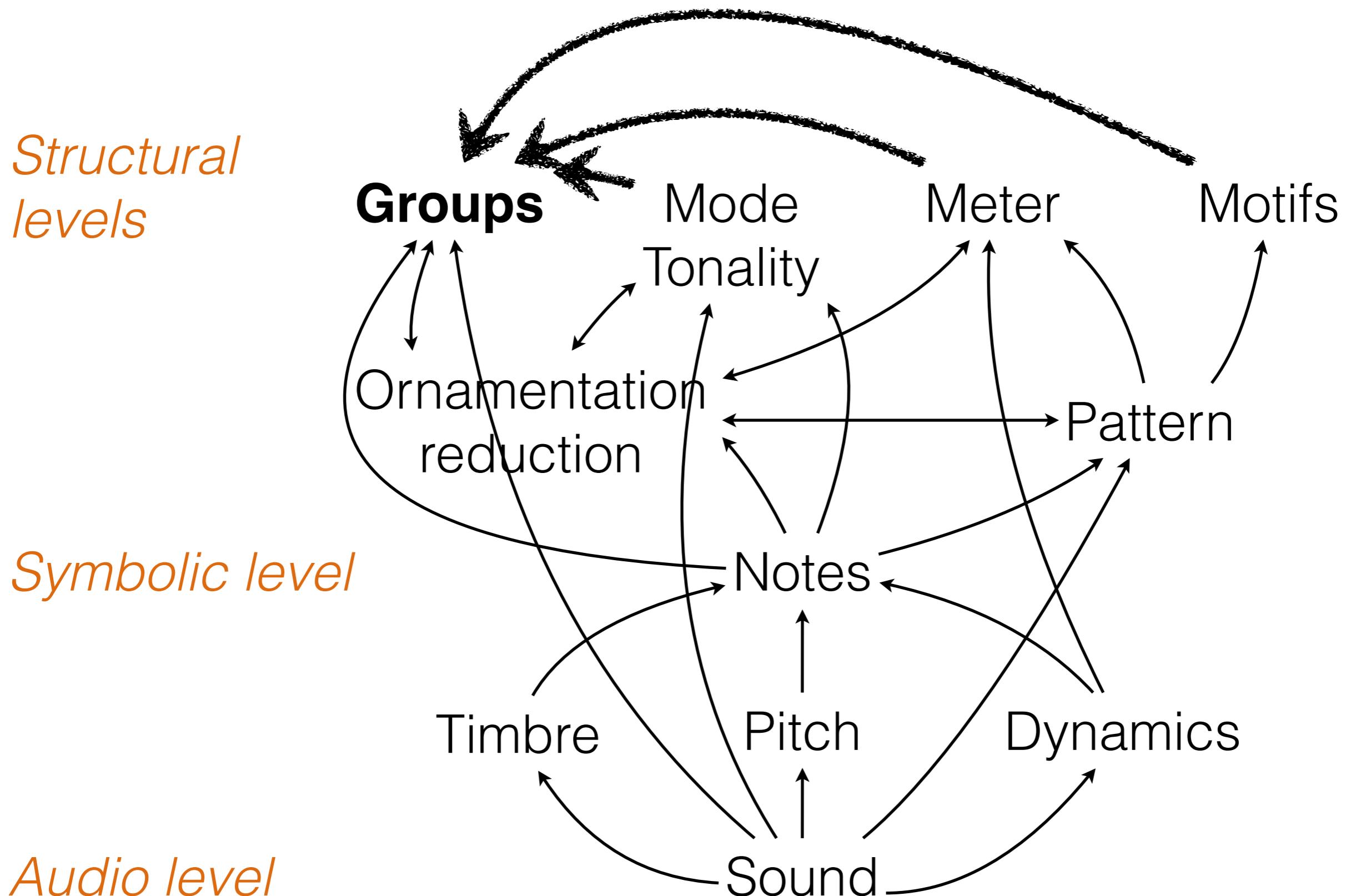


O. Lartillot, “Reflexions towards a generative theory of musical parallelism”, *Musicae Scientiae*, DF 5, 2010.

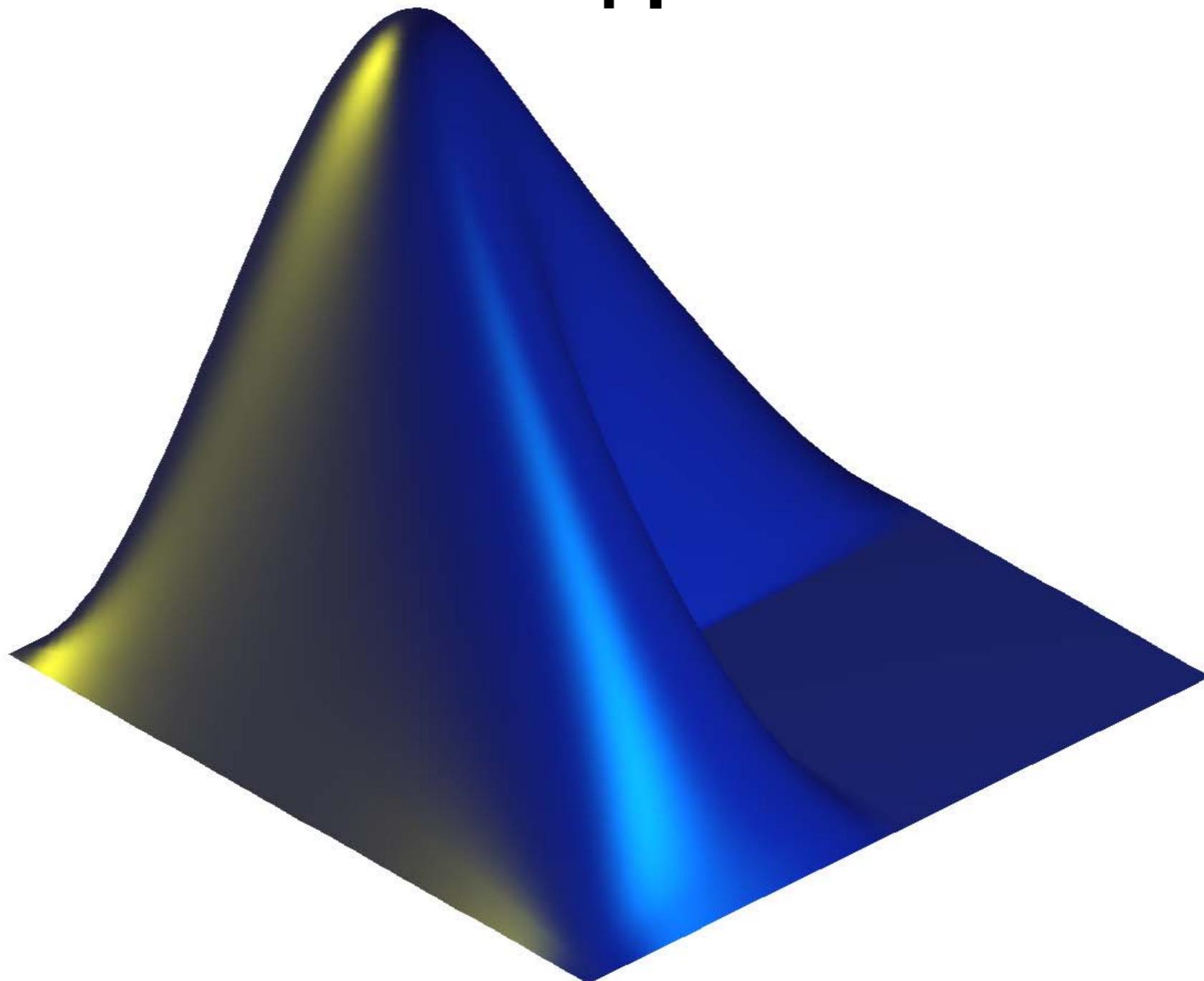
# Audio / symbolic



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



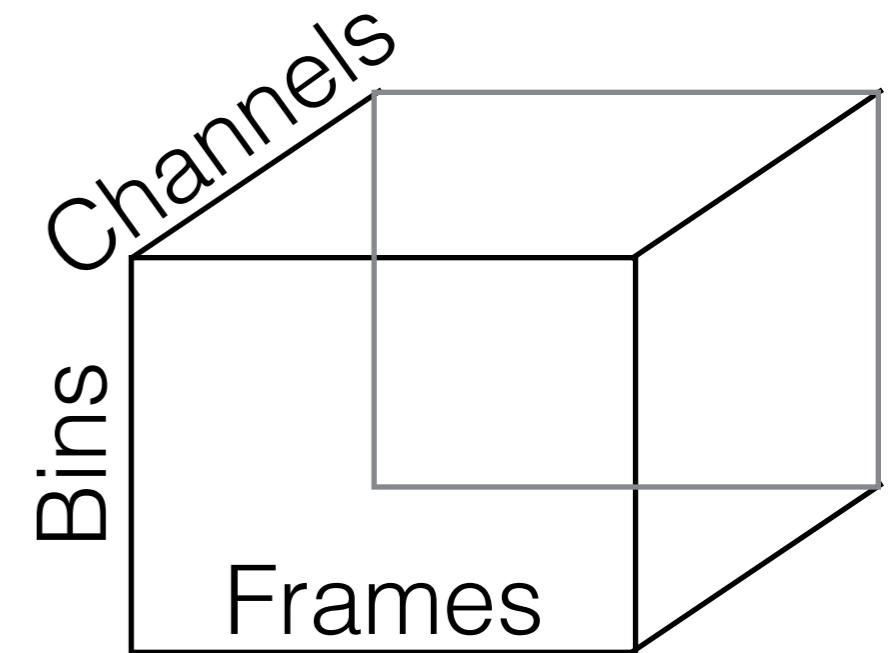
4.



How it works

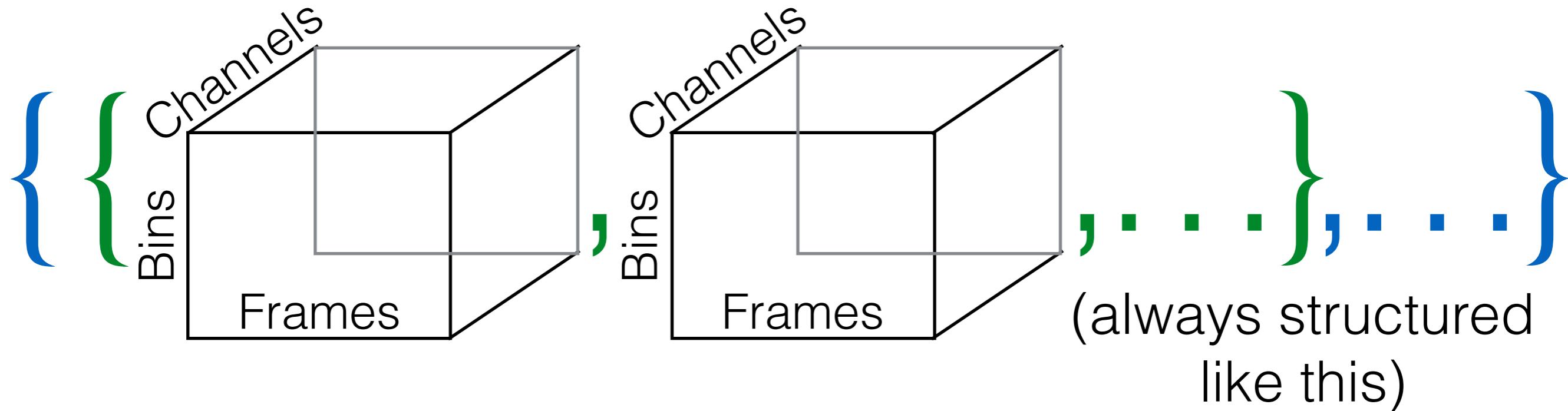
# Matrix convention in *MIRtoolbox*

- In standard *Matlab* code, processing of matrix dimensions makes the code somewhat obscure.
- In *MIRtoolbox*, one single matrix convention:
- This convention should be followed everywhere in the code, and is not explicitly explained to the readers.



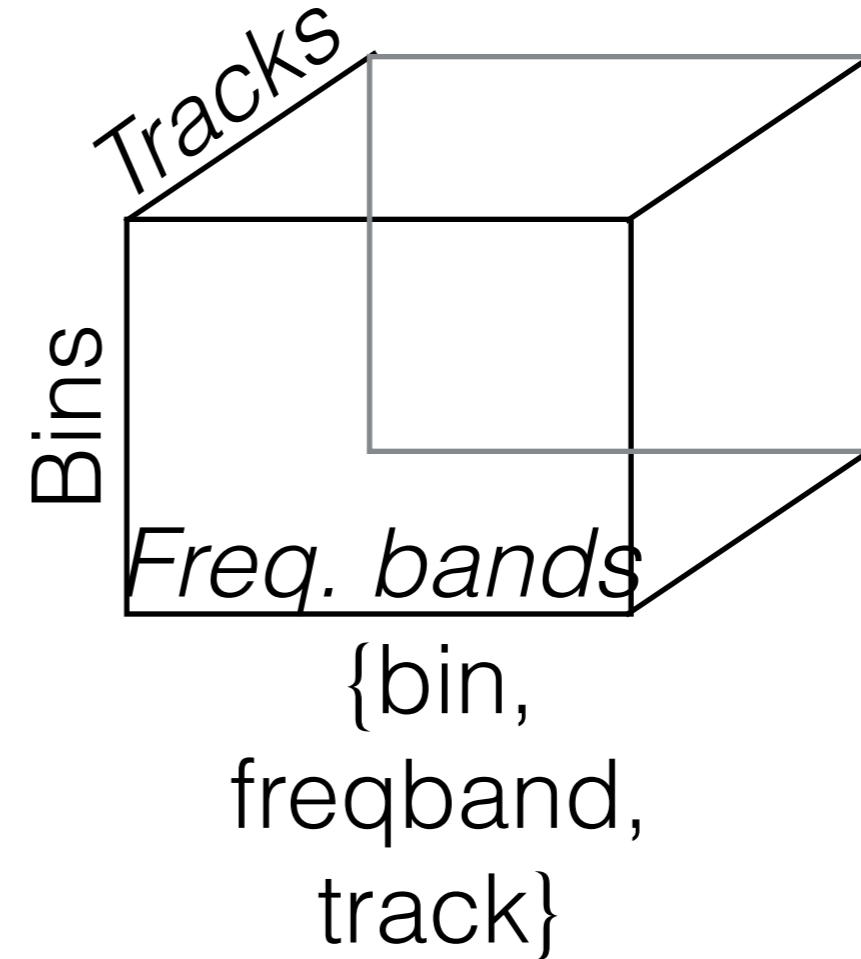
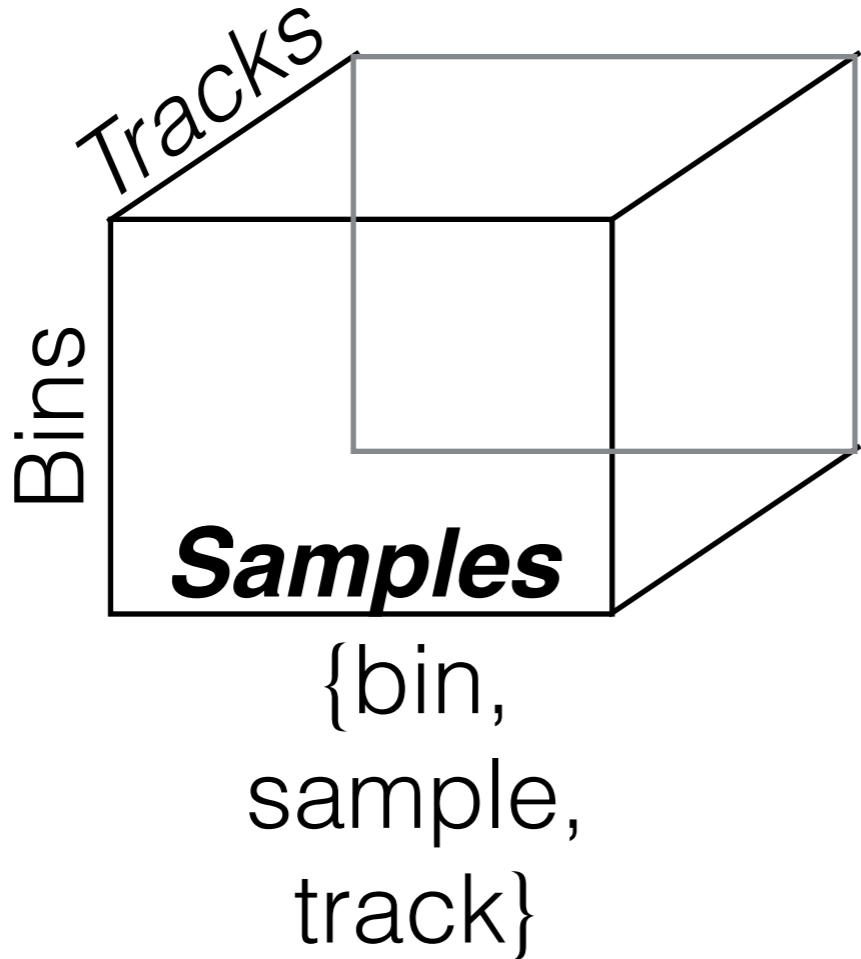
# Matrix convention in *MIRtoolbox*

- One single convention that needs to allow complex things such as the data of a batch of audio files that are each segmented.



# Matrix convention in MiningSuite

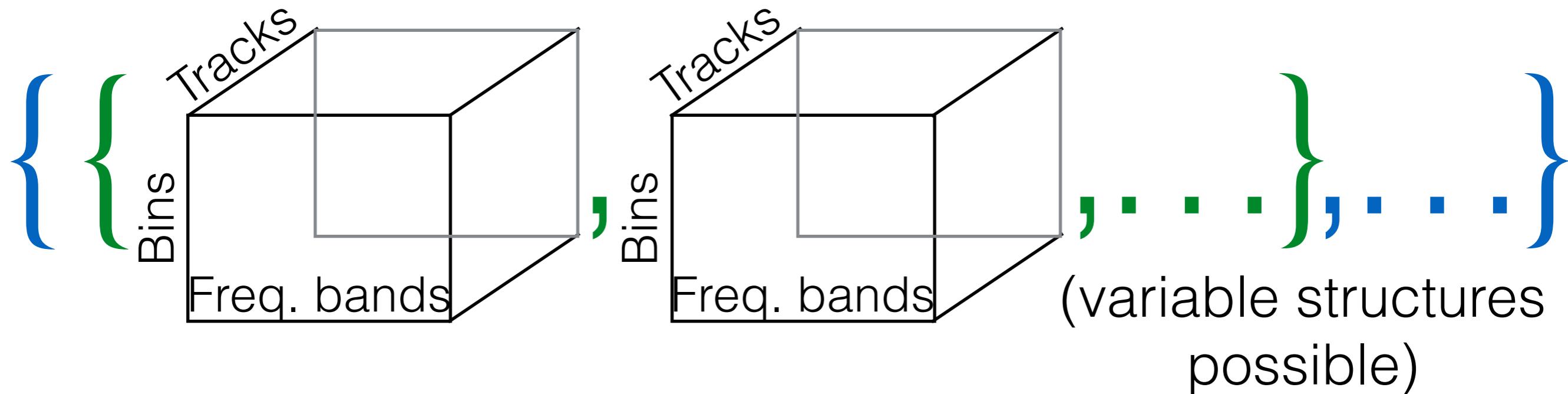
- There is no more constraint on the choice of dimensions.



- Allows configuration that were not possible in *MIRtoolbox*.

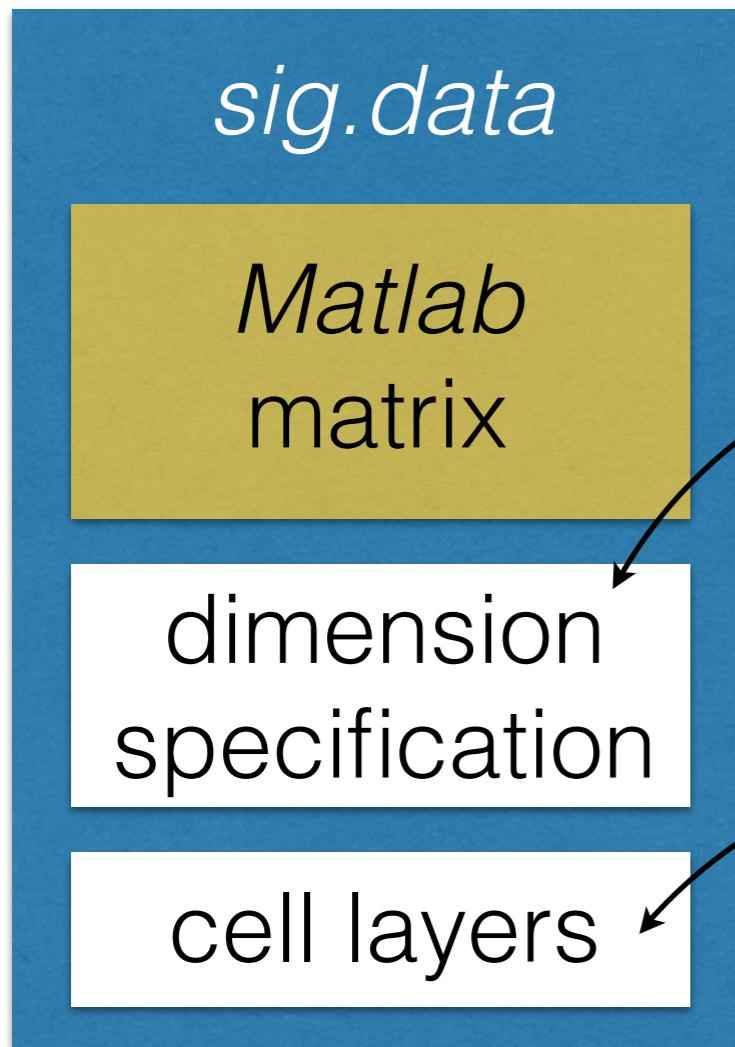
# Matrix convention in MiningSuite

- More complex configurations can be freely designed, outside of any “pre-wired” configuration.



# *sig.data*

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



For instance:  
{bin,sample,channel}

Indicating if the data is a cell-array,  
corresponding to batch of audio files,  
sequence of segmented data, etc.

# *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.data*

```
function out = routine(d,...)
```

```
    e = d.apply(@algo,{...},{'bin'},2);
```

```
    out = {e};
```

```
end
```

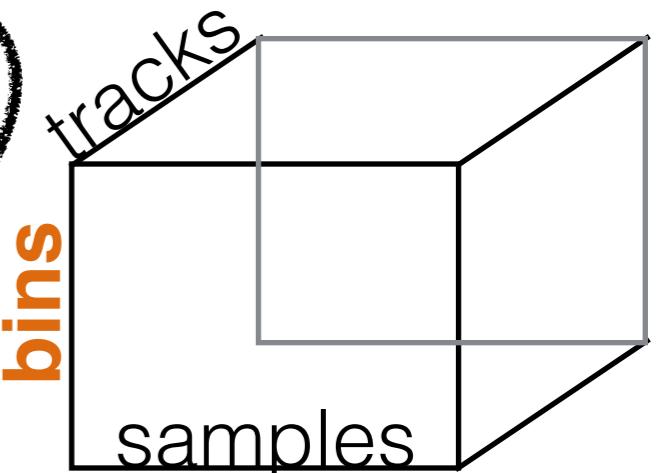
```
function ceps = algo(m,...)
```

```
...
```

```
    ceps = mfccDCTMatrix * m;
```

```
end
```

**d:**



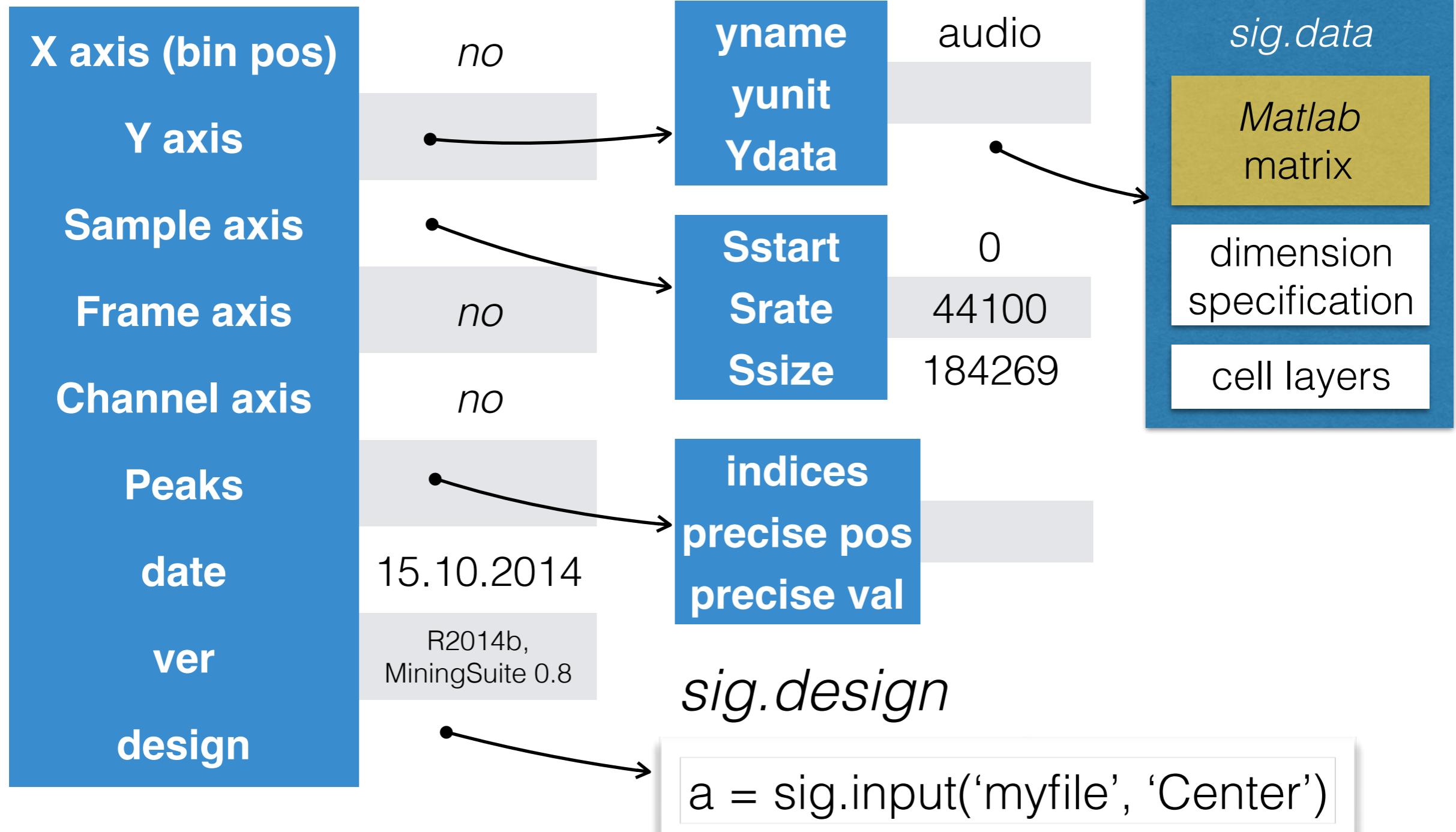
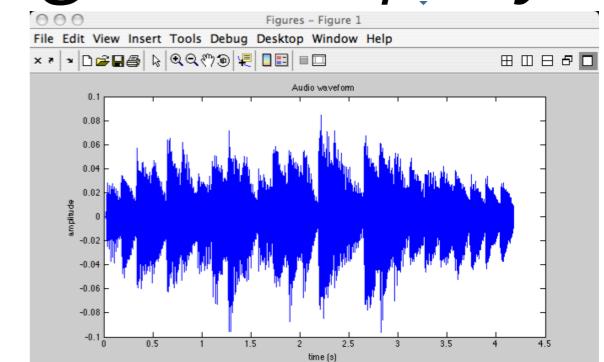
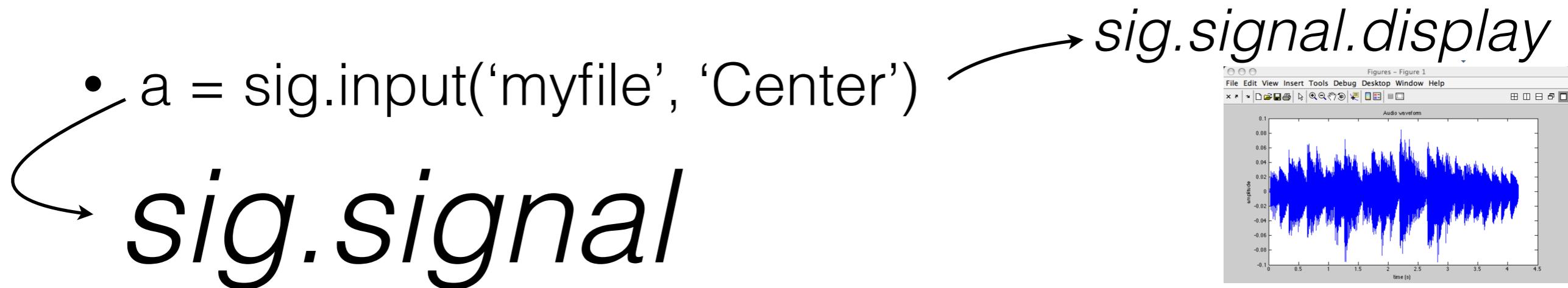
**bins**

**2 dimensions:**

**m:**

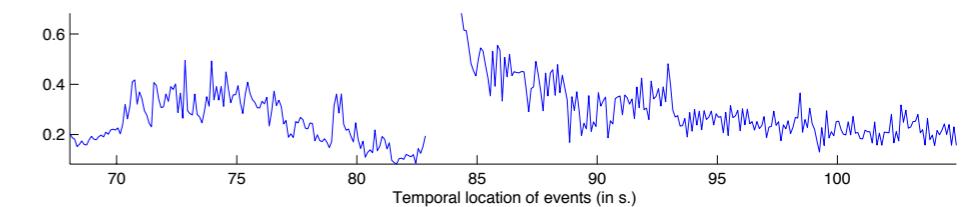
**bins**





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

`sig.signal.display`



# `sig.signal`

<b>X axis (bin pos)</b>	<i>no</i>
<b>Y axis</b>	
<b>Sample axis</b>	<i>no</i>
<b>Frame axis</b>	<i>no</i>
<b>Channel axis</b>	<i>no</i>
<b>Peaks</b>	
<b>date</b>	15.10.2014
<b>ver</b>	R2014b, MiningSuite 0.8
<b>design</b>	

<b>yname</b>
<b>yunit</b>
<b>Ydata</b>

<b>Sstart</b>
<b>Srate</b>
<b>Ssize</b>

<b>indices</b>
<b>precise pos</b>
<b>precise val</b>

`brightness`

0
<b>40</b>
.05

<b>sig.data</b>
Matlab matrix
dimension specification
cell layers

`sig.design`

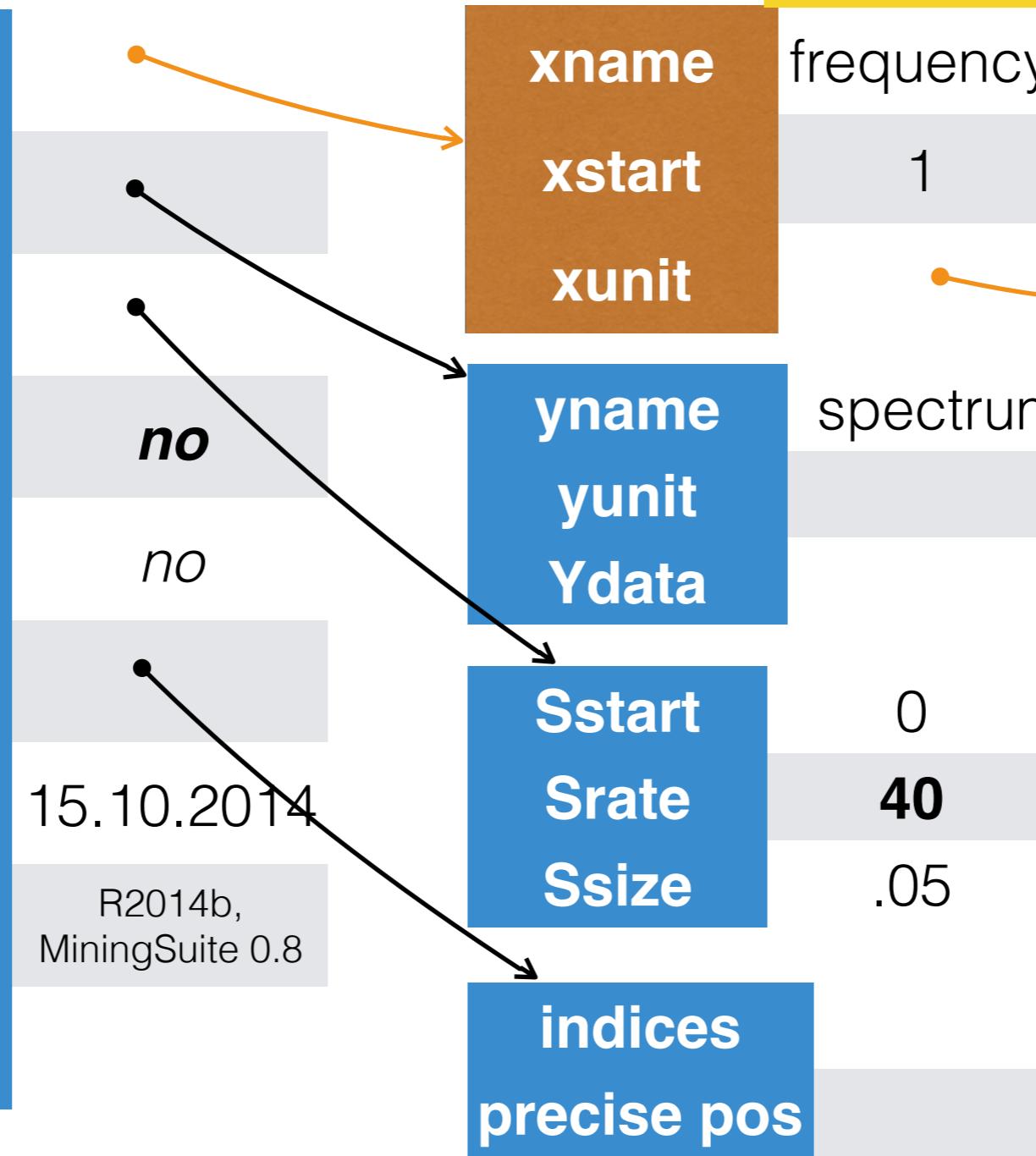
`b = sig.brightness(a, 'Frame')`

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

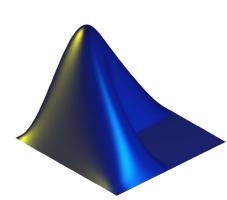
Information such as time positions are regenerated on the fly.

# *sig.Spectrum*

<b>X axis</b>
<b>Y axis</b>
<b>Sample axis</b>
<b>Frame axis</b>
<b>Channel axis</b>
<b>Peaks</b>
<b>date</b>
<b>ver</b>
<b>design</b>

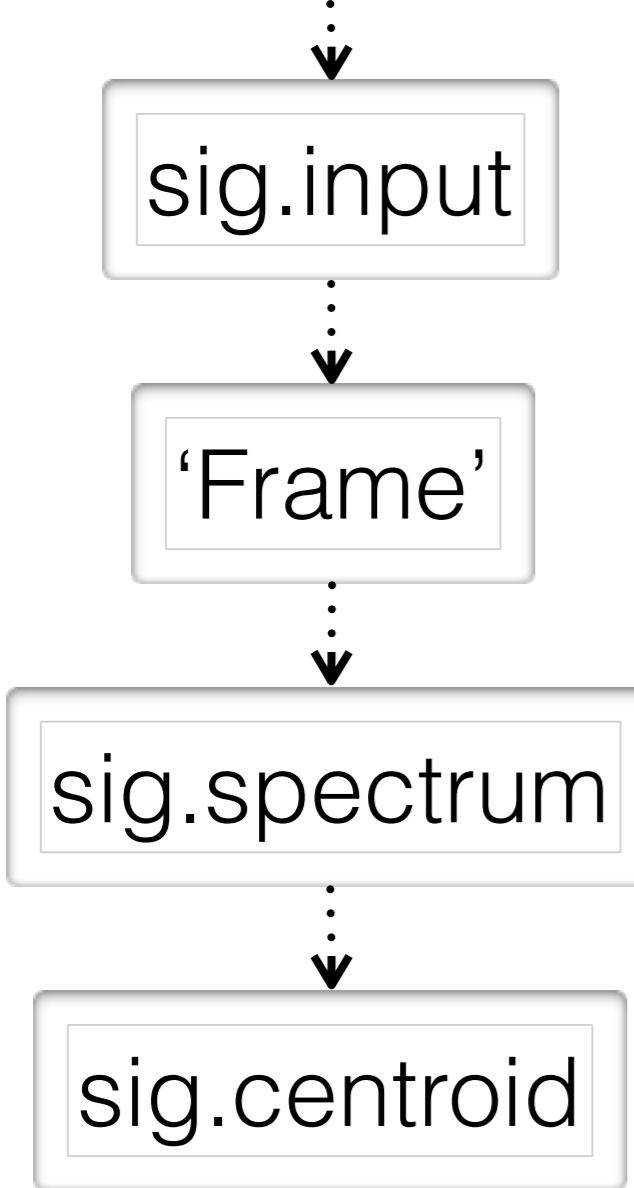


<i>sig.axis</i>	<i>sig.unit</i>
<code>xname</code>	Hz'
<code>xstart</code>	0
<code>xunit</code>	10.76
<code>yname</code>	@..
<code>yunit</code>	@..
<code>Ydata</code>	
<code>Sstart</code>	0
<code>Srate</code>	40
<code>Ssize</code>	.05
<code>indices</code>	
<code>precise pos</code>	
<code>precise val</code>	
<code>power</code>	1
<code>log</code>	0
<code>phase</code>	<i>sig.data</i>



# Data flow graph in MiningSuite

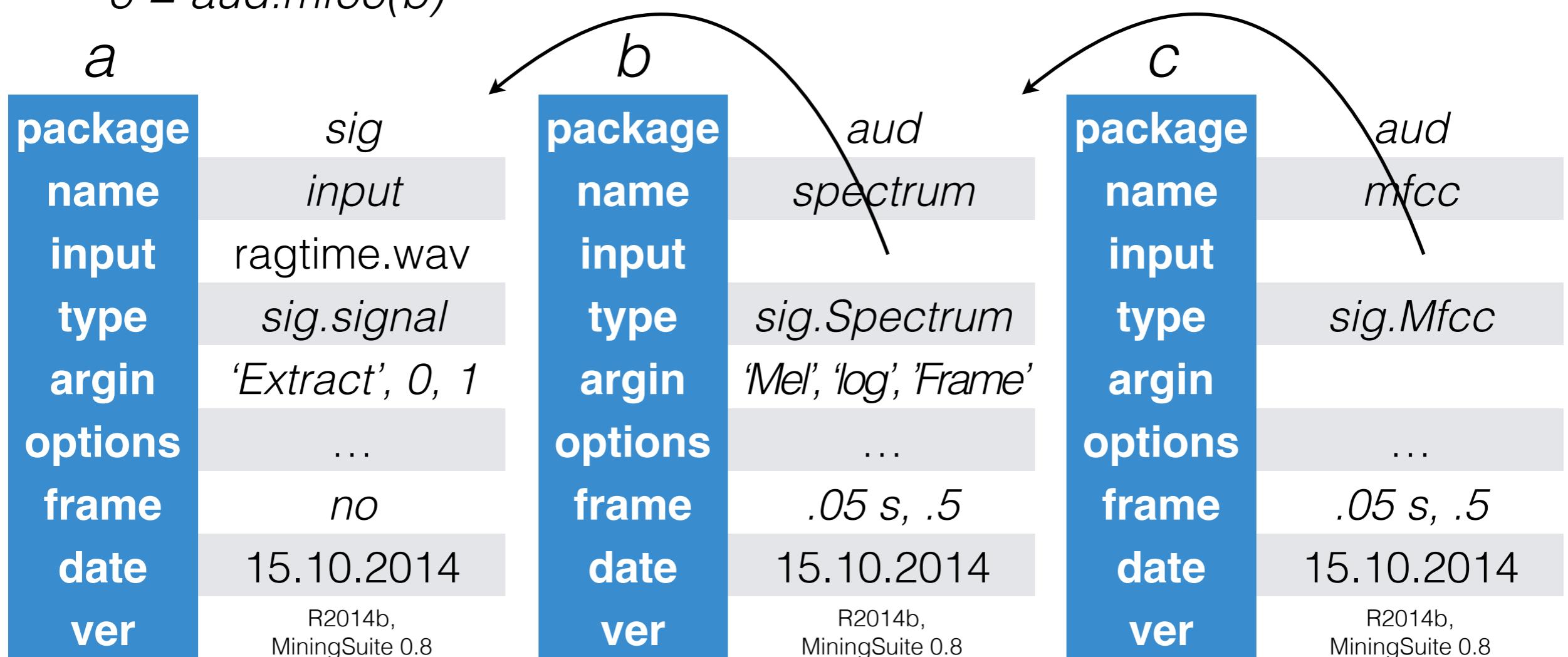
long audio file,  
batch of files



- `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.)

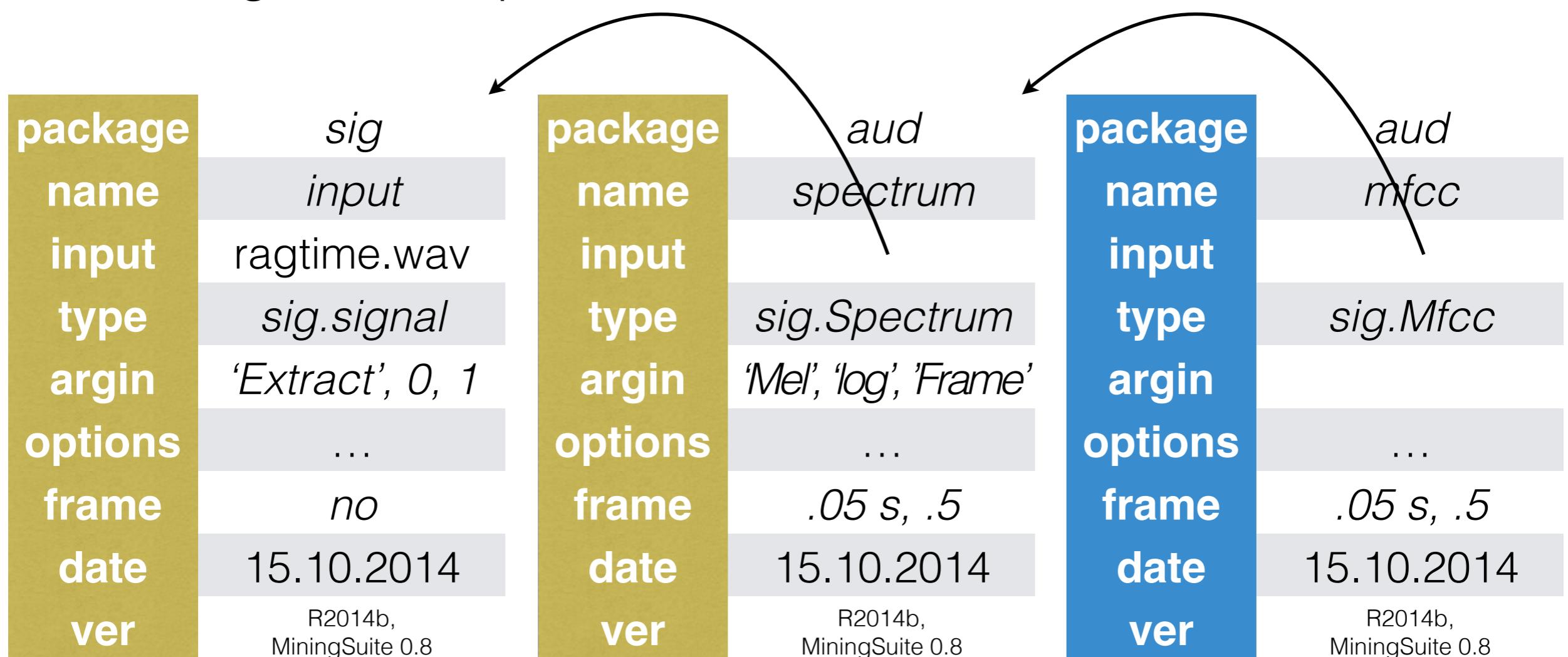
# *sig.design*

- `a = sig.input('ragtime.wav', 'Extract', 0, 1);`
- `b = aud.spectrum(a, 'Mel', 'log', 'Frame');`
- `c = aud.mfcc(b)`

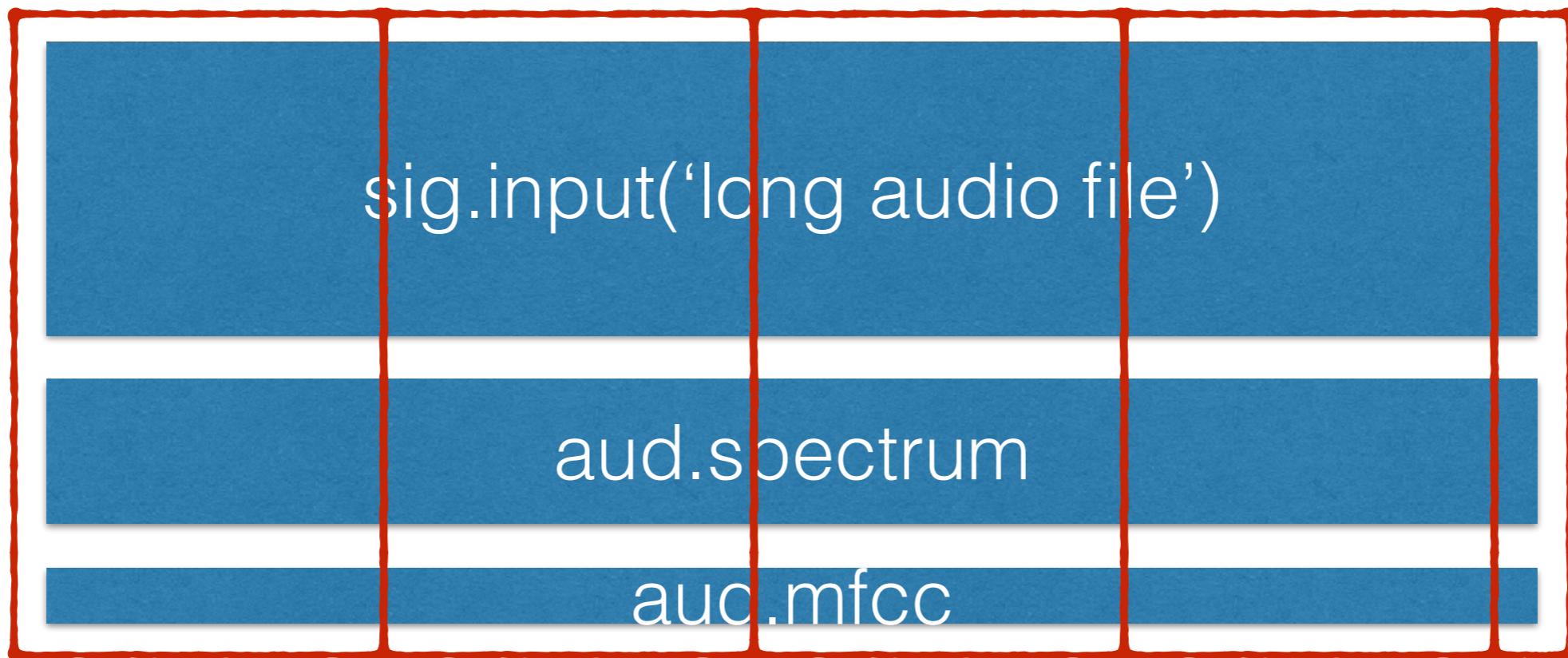


# *sig.design*

- `c = aud.mfcc('ragtime.wav', 'Frame')`
- deploying the implicit data flow prior to the actual operator, during the '**Init**' phase.



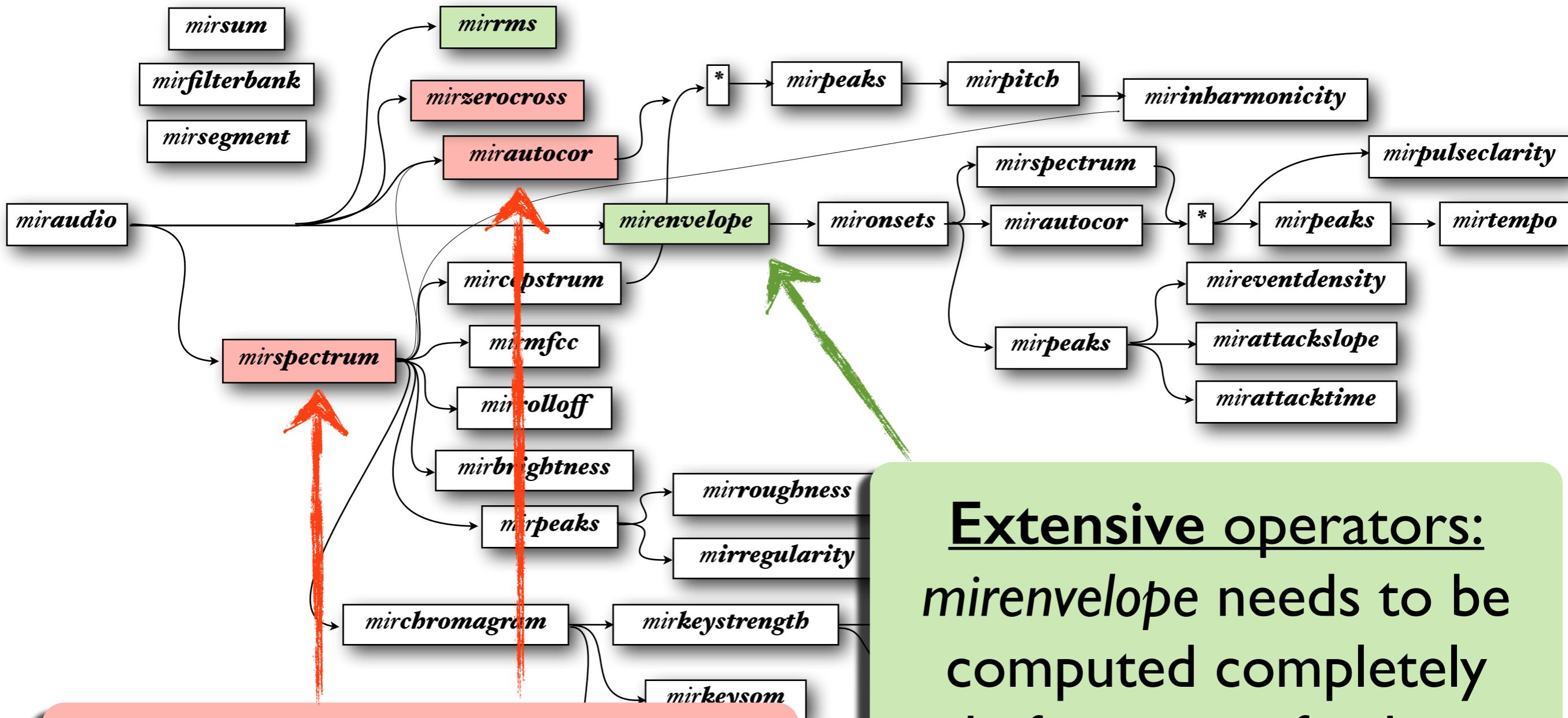
# *sig.design.eval*



- Divide the audio file into successive chunks.
- Compute the whole process on each chunk separately.
- Recombine the final results by concatenation.



# when there is no frame decomposition



Intensive operators:

Chunk decomposition needs  
to be performed here

Extensive operators:  
mirenvelope needs to be  
computed completely  
before going further



# mireval

## non-frame-based evaluation

- `a = miraudio('Design');`
- `c = mirmfcc(a);`
- `mireval(c,'Folder')`

very complex paradigm for a situation not practically usual in MIR.

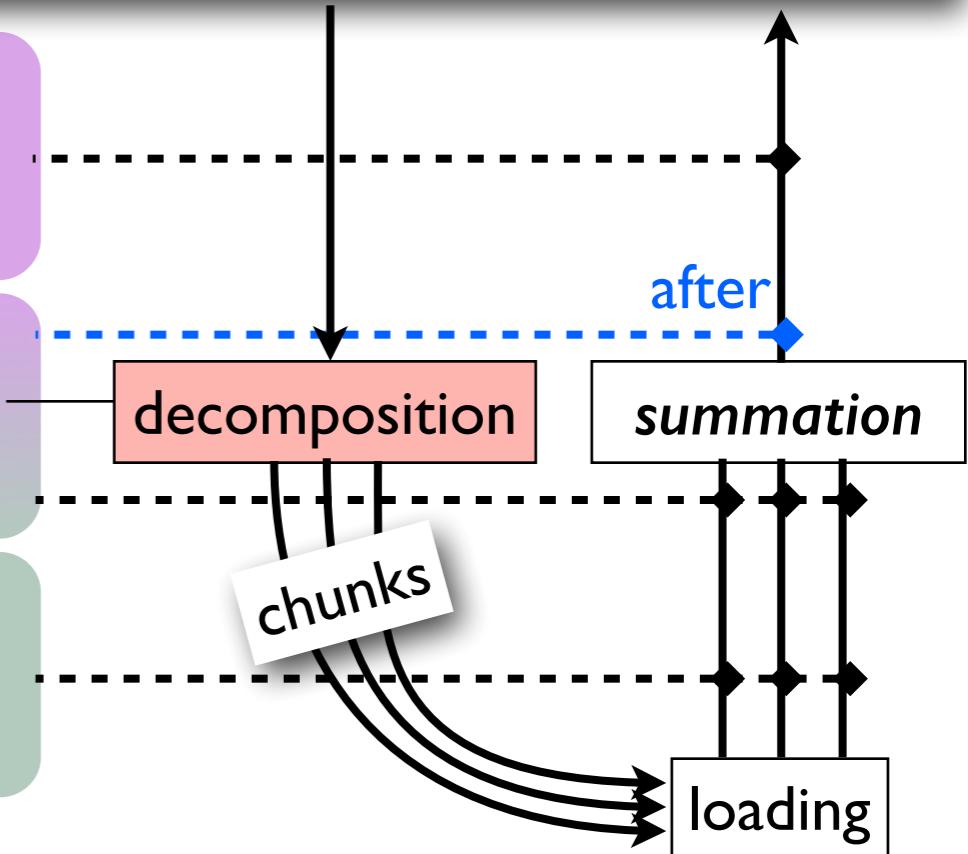
**MiningSuite** is much more simple

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

`c @mirdesign type=@mirmfcc  
options: Rank=1:13, Delta=0, ...`

`@mirdesign type=@mirspectrum  
options: Win='hamming', ...`

`a @mirdesign type=@miraudio  
options: Sampling=11025, ...`



*sig.design.eval('Folder')*

- $\mathbf{a} = \text{aud.mfcc}(\text{'Folder'}, \dots)$
- $d = \mathbf{a.eval}$ 
  - For each separate file in the folder:
    - evaluate the design on that file (using the chunk decomposition)
    - store the results

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

***sig.design*** object, storing only the data flow graph

-  ***a*** = *aud.mfcc('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***

# *sig.operate* operators' main routine

+aud/brightness.m:

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

# 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’;
```

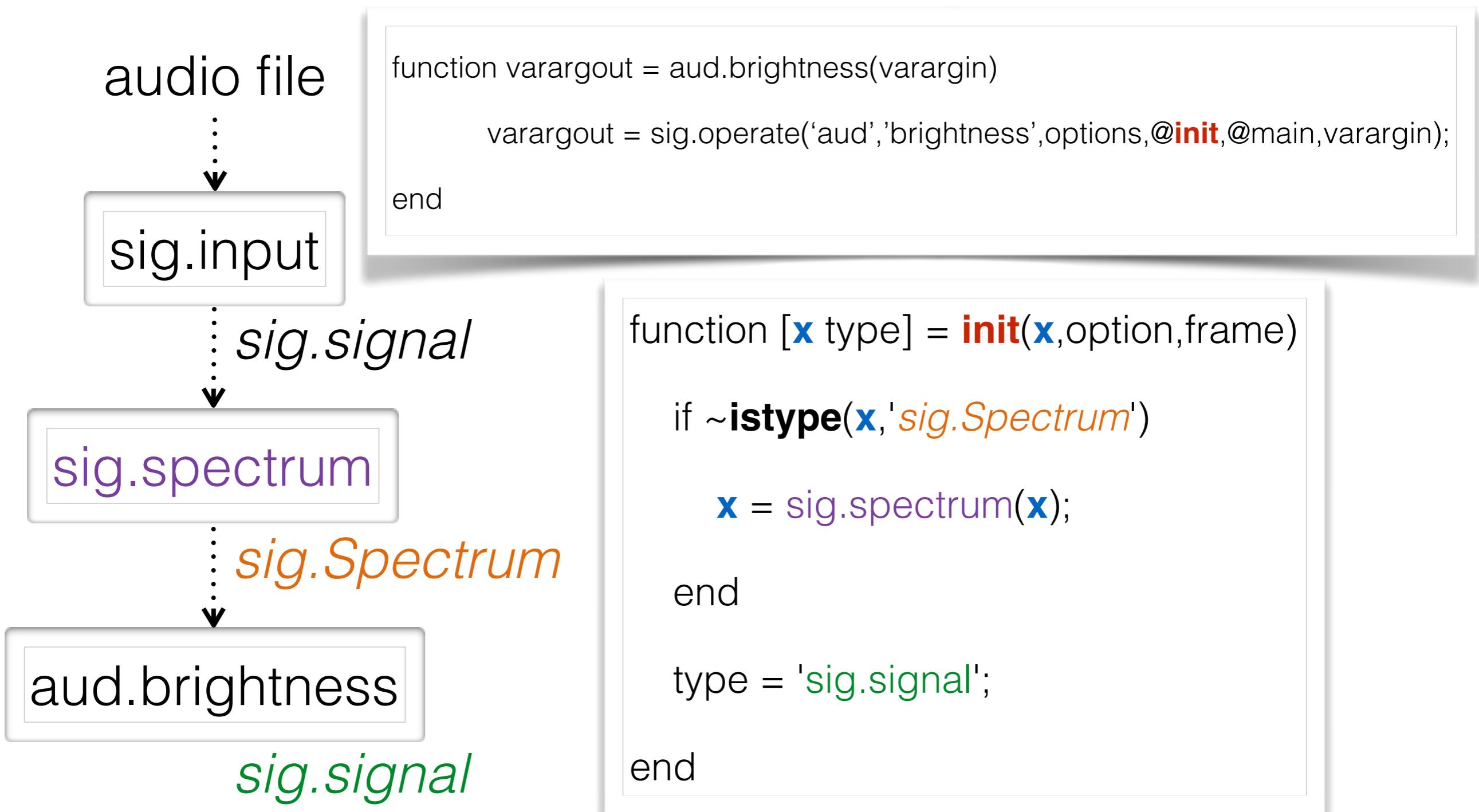
```
domain.choice = {‘Signal’, ‘Symbolic’};
```

```
domain.default = ‘Signal’;
```

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

# *@init*

## implicit data flow deployment



```
function varargout = aud.brightness(varargin)

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

end
```

```
function out = main(in,option)
    x = in{1};

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

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

        x = sig.signal(res,'Name','Brightness','Srate',x.Srate,'Ssize',x.Ssize);
    end

    out = {x};

end
```

```
function out = main(in,option)
```

```
...
```

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

**x.Ydata:** { { , , ... } , ... }

```
function out = routine(d,f,f0)
```

**d:**

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

```
out = {e};
```

```
end
```

**sig.compute**  
applies the  
routine to the  
successive  
segments,  
successive  
audio files,  
etc.

```
function out = main(in,option)
```

```
...
```

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

```
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
```

***sig.data.apply***

applies the sub-routine algo to the data **properly formatted**, with loops if needed.

+aud/mfcc.m:

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

```
function out = routine(d,...)
```

```
    e = d.apply(@algo,{...},{'bin'},2);
```

```
    out = {e};
```

```
end
```

```
function ceps = algo(m,...)
```

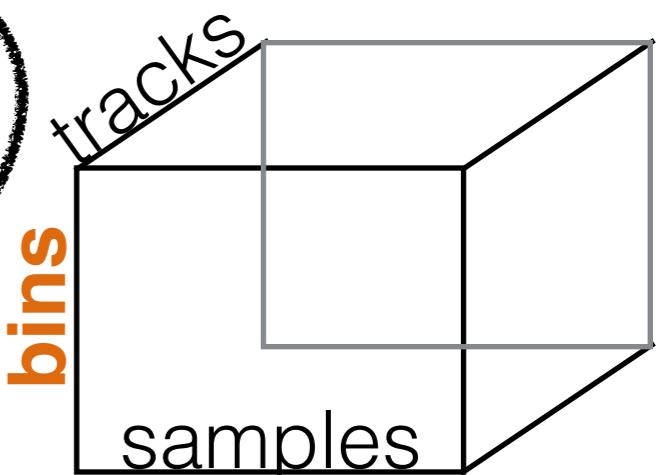
```
...
```

```
    ceps = mfccDCTMatrix * m;
```

```
end
```

**d:**

**bins**



**2 dimensions:**

**m:**

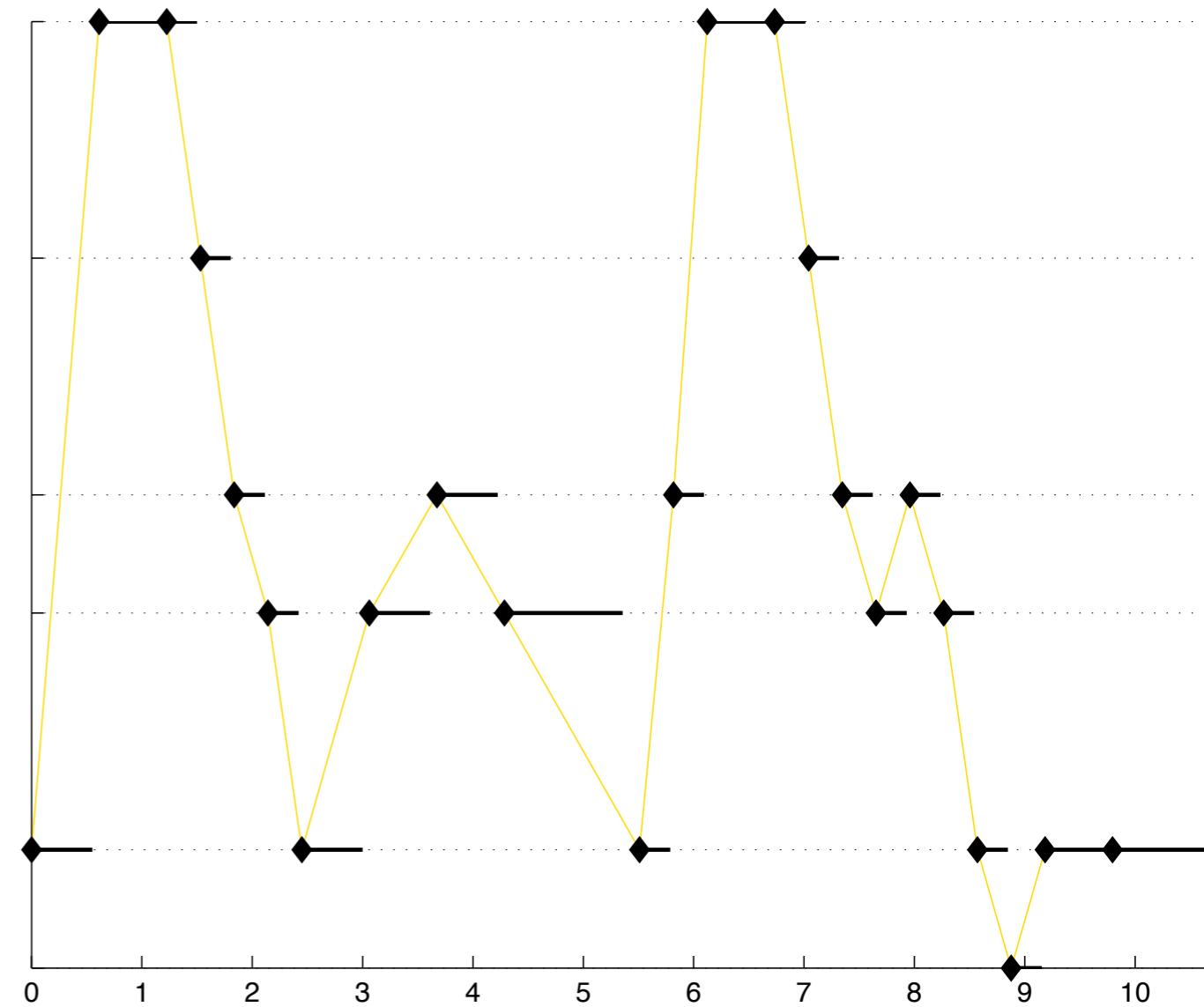
**bins**



# *seq.event* note characterization

Each note is a *seq.event*:

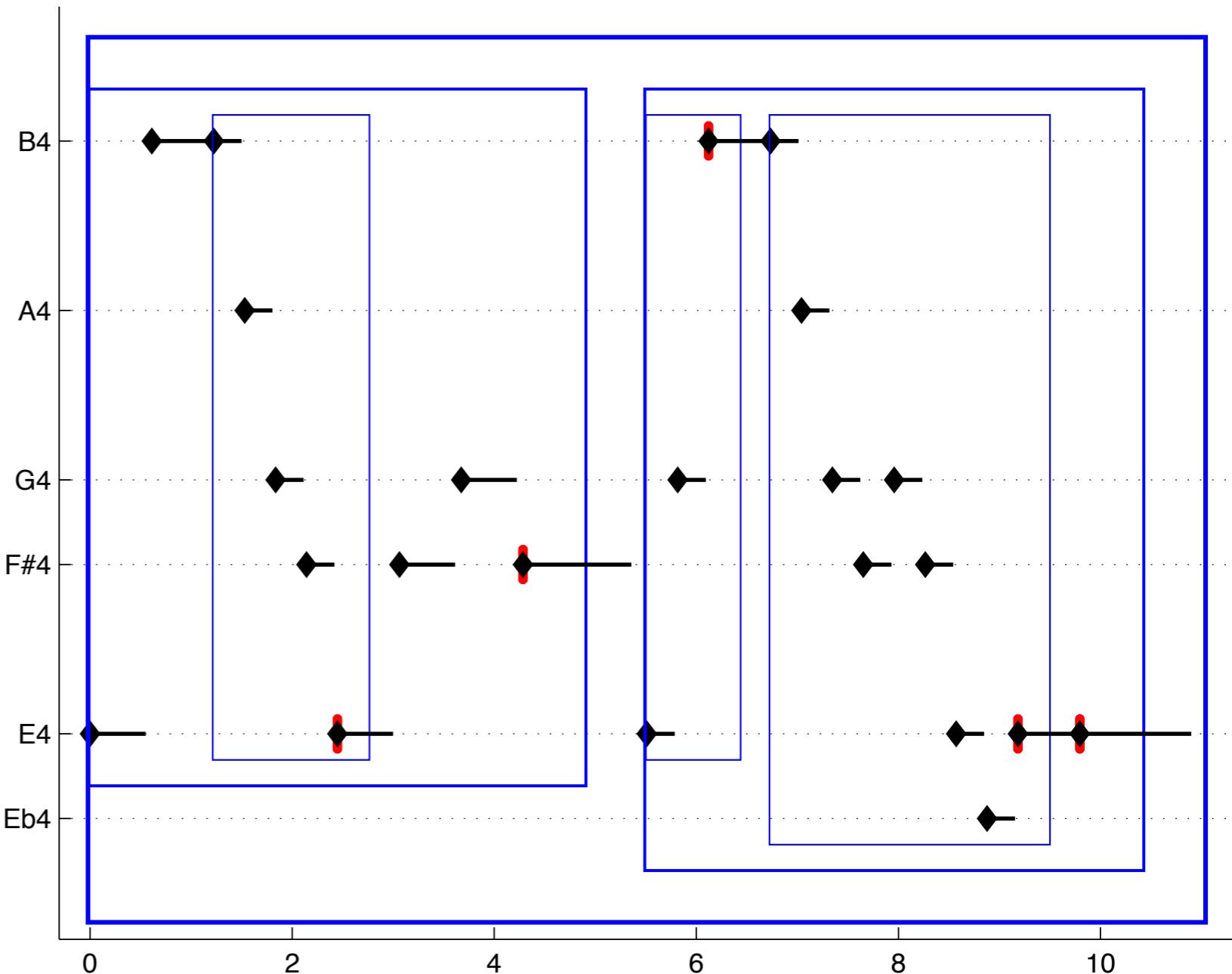
- characterization of the note: musical parameter (instance of class *seq.paramstruct*)
- previous note
- next note



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

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

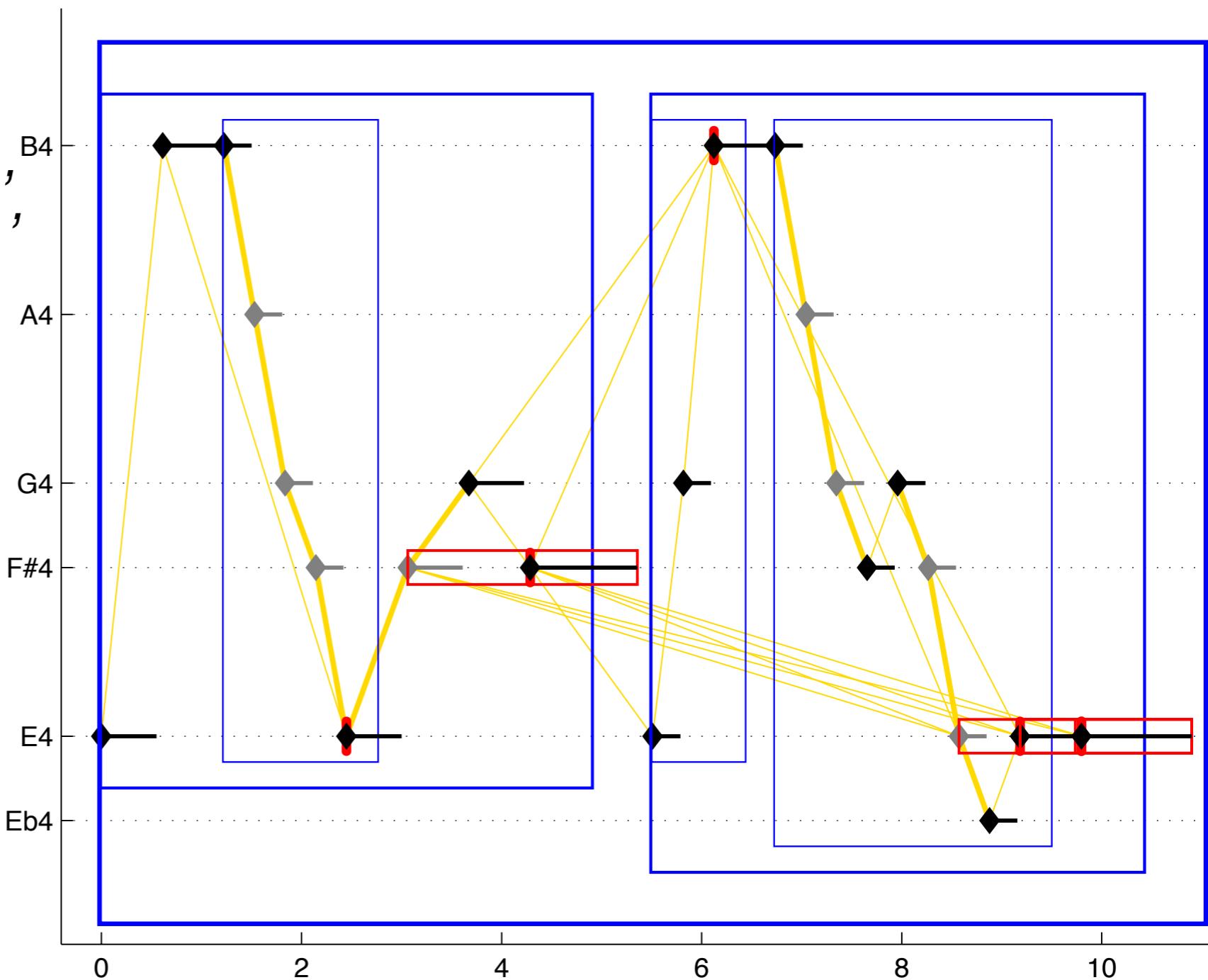
Groups, like notes, are instances of `seq.event`



# *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.

*more general*

### 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

# *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*

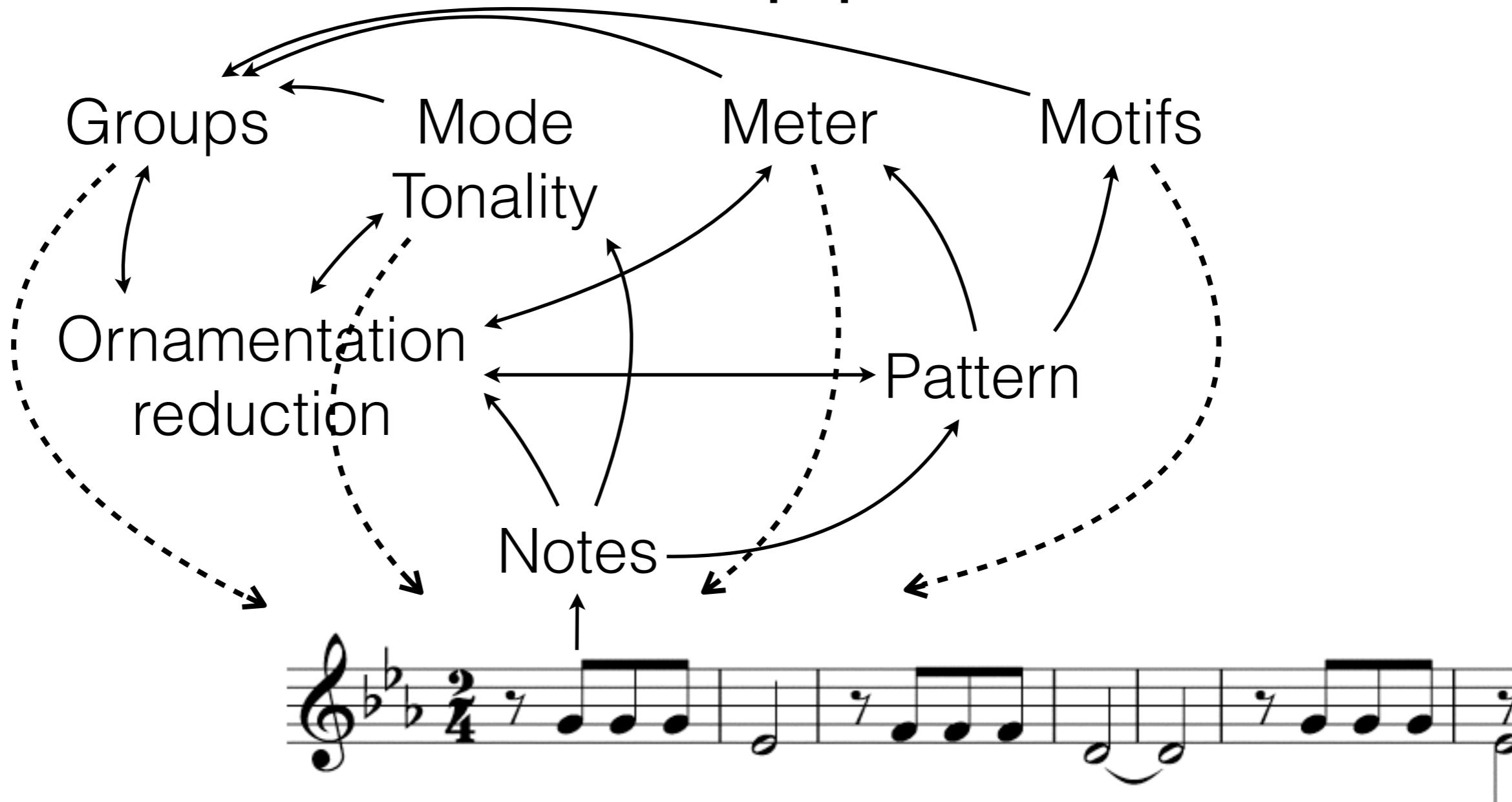


$s = \text{seq.syntagm}(n1, n2)$

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

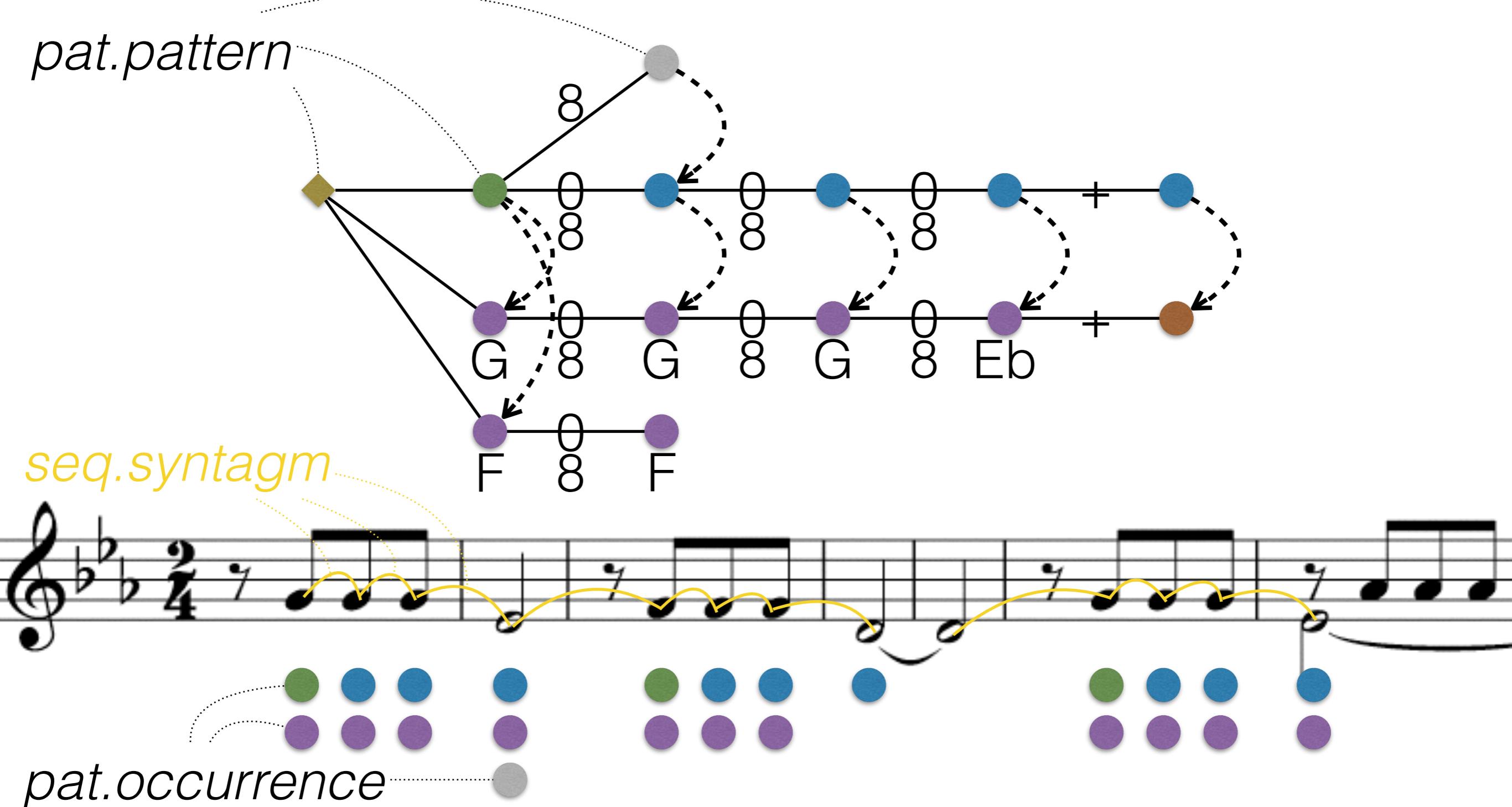
# *mus.score*

## incremental approach

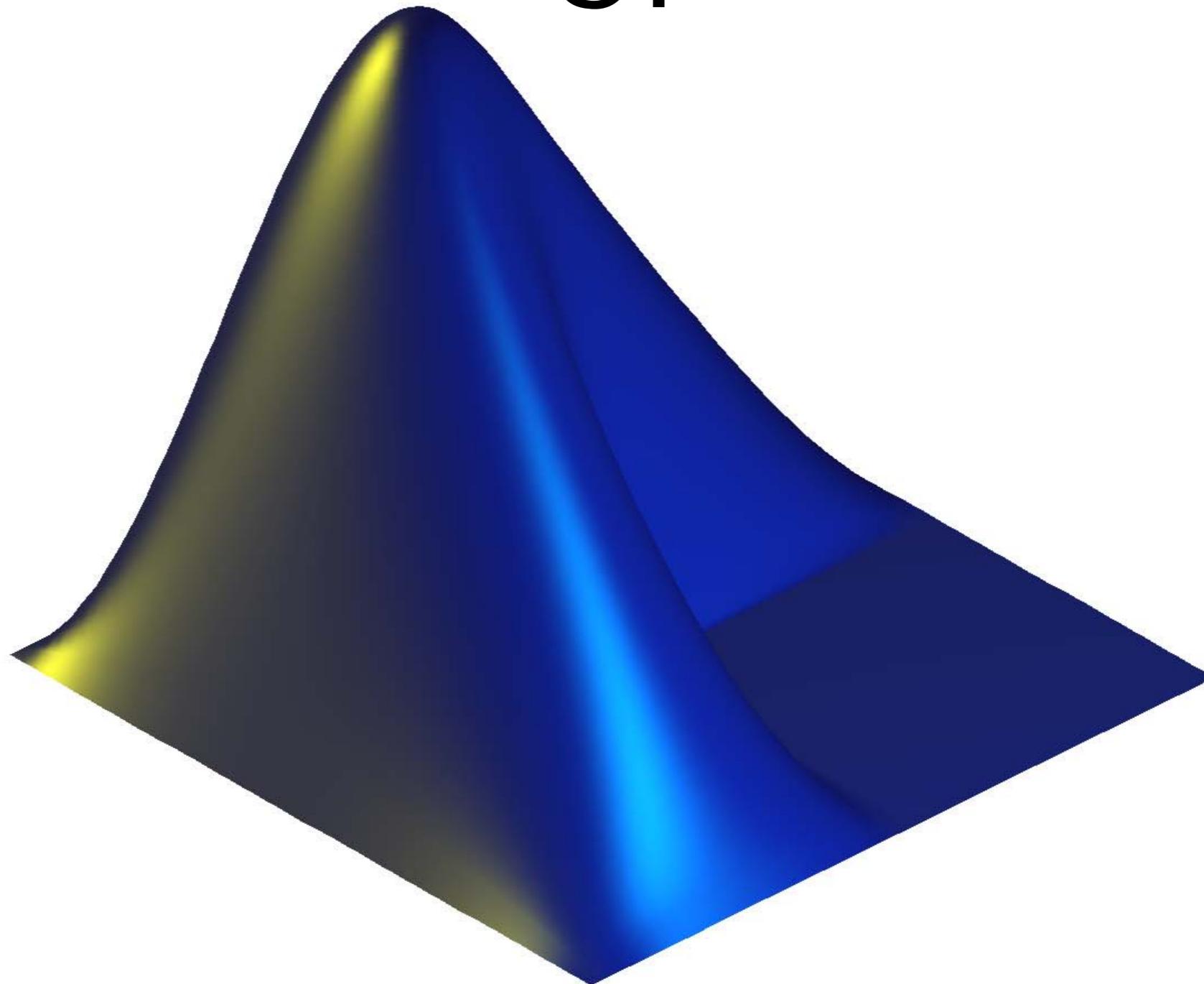


Each successive note is progressively integrated to all musical analyses, driving interdependencies.

# *mus.score* incremental approach



5.



How you can contribute

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

- All releases, *SubVersion* repository, integrated with code review environment
- User's Manual and documentations in wiki environment
- Mailing lists: news, discussion list related to ongoing development, commits, issues registered and modified, discussion list for users
- Tickets to issue bug reports

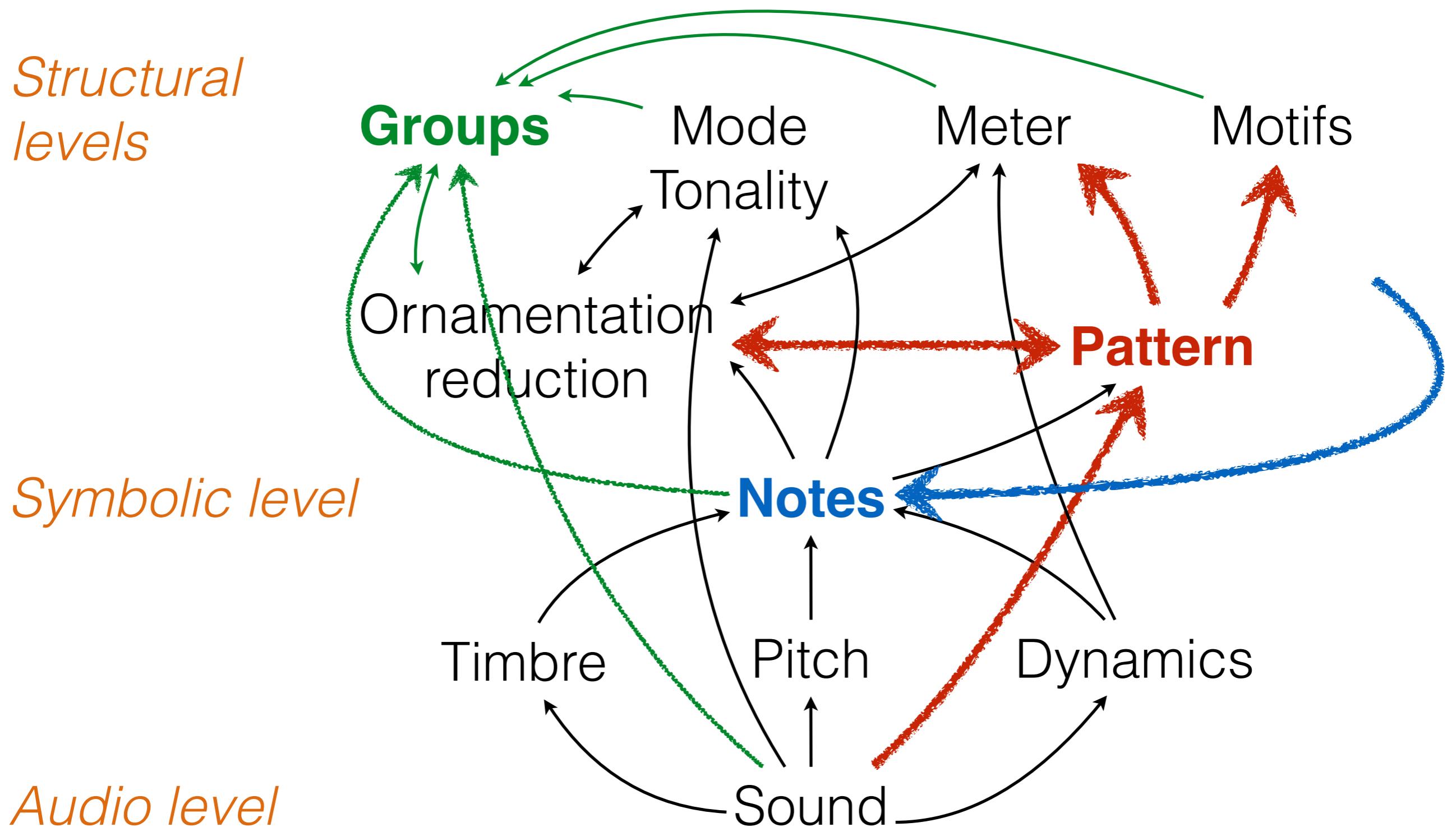
# Open-source project

- *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 refines lower-level transcription.
- Pattern mining is a central process in symbolic-based music analysis.



# 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
- Thanks to all *MIRtoolbox* contributors and users
- Thanks to *MiningSuite* beta-testers

