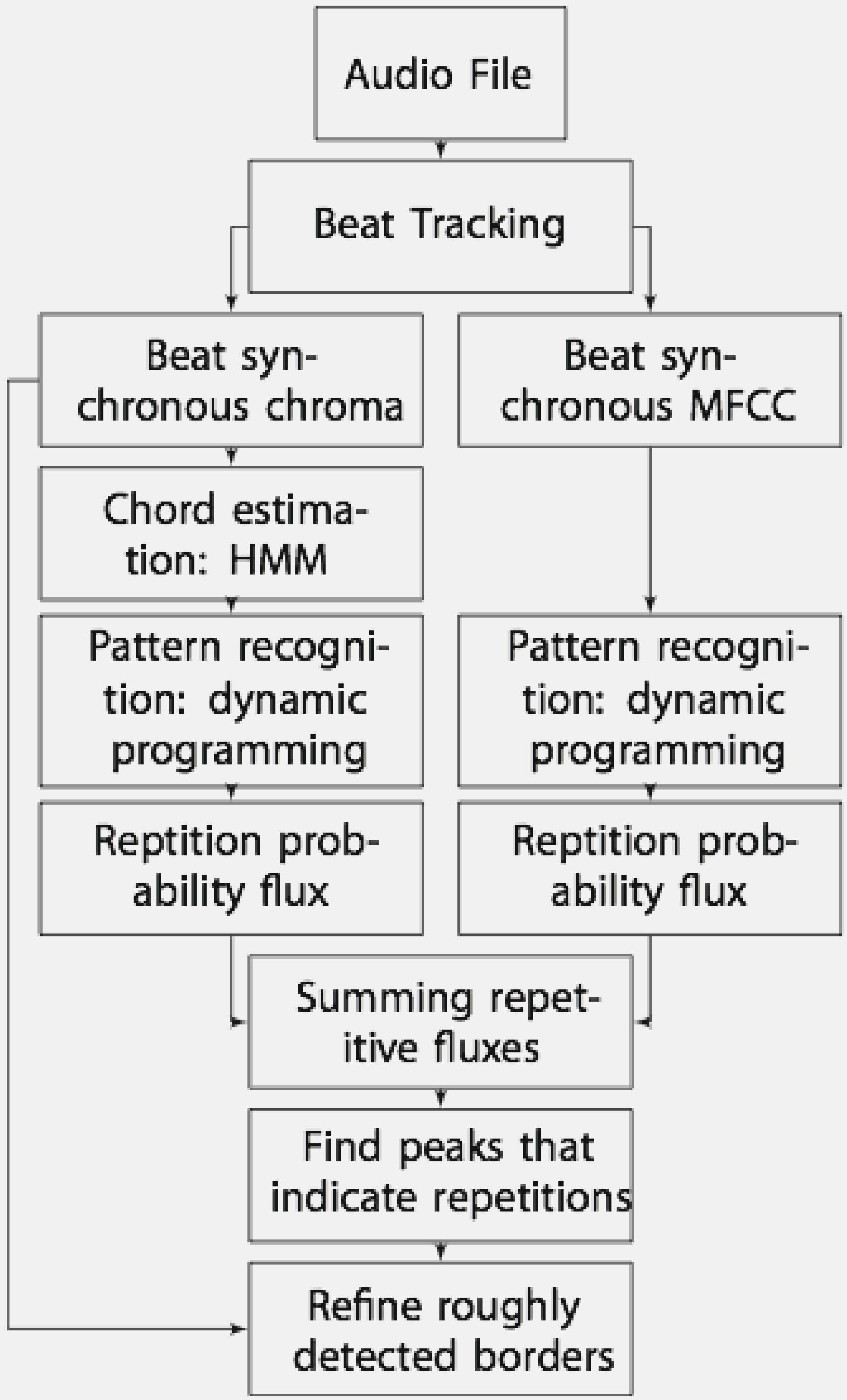
# CHROMA AND MFCC BASED PATTERN RECOGNITION IN AUDIO FILES UTILIZING HIDDEN MARKOV MODELS AND DYNAMIC PROGRAMMING

Alexander Wankhammer
Peter Sciri

- What is musical structure?
  - Musically „relevant“ sections
  - Repeating, distinct parts of a composition
    - Intro – Verse – Chorus – Verse etc.
- How can we describe it?
  - Musical point-of-view:
    - harmonic progression
  - Perceptional PoV:
    - spectral properties



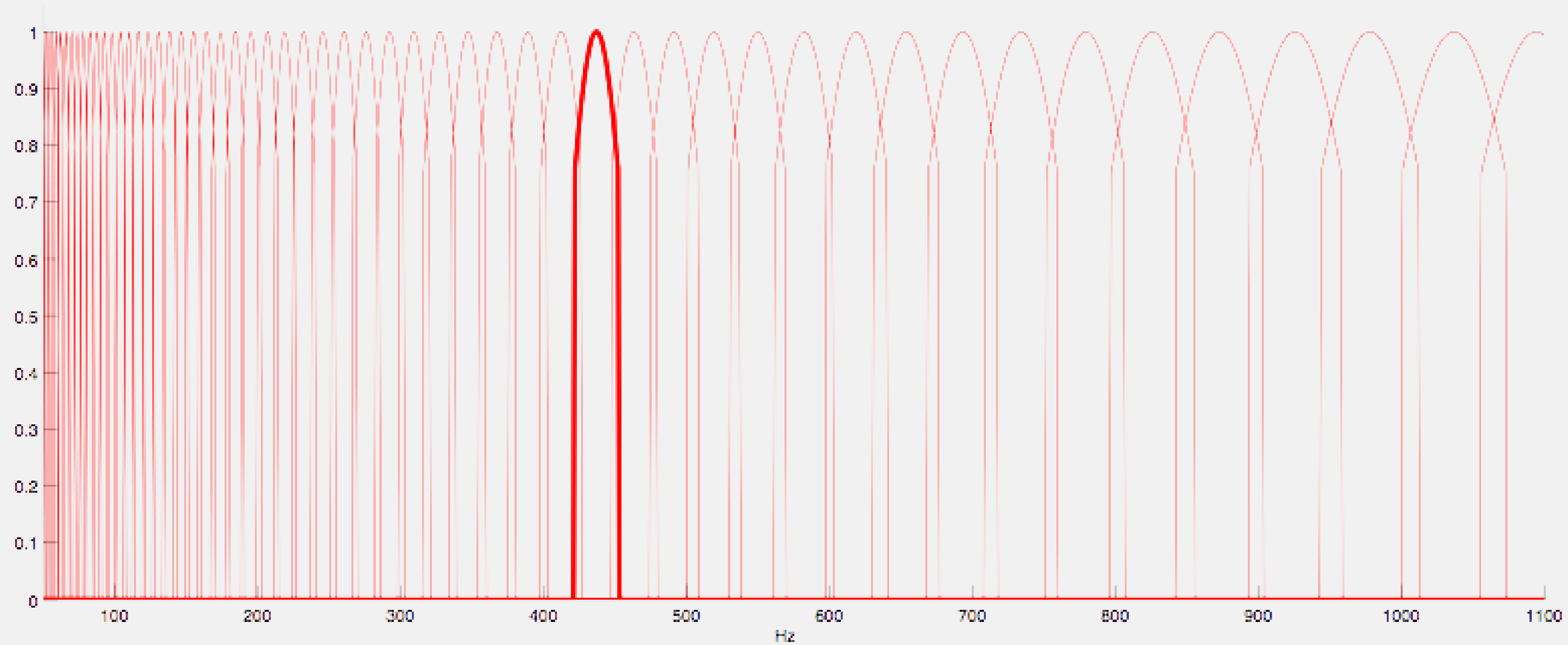

- Read Audio
- Perform beat tracking
- Compute spectral features
- Calculate similarities
- Roughly estimate segment borders
- Refine those borders

- What are appropriate features for
  - harmonic progression?
    - rasterize spectrum into semitone bands
      - $\rightarrow$ Constant-Q Transform
    - treat all octaves equally
      - $\rightarrow$ Chroma (Harmonic Pitch Class Profile)
    - determine a musically meaningful sequence of chords
      - $\rightarrow$ define a Hidden Markov Model (HMM)
  - perceptional information?
    - Mel-Frequency Cepstral Coefficients (MFCC)

- Constant-Q Transform
  - linear resolution of STFT does not match human perception $\rightarrow$ too much „effort“ in HF area
    - summarize energy of semitone bands into scalar values

$$f_{center}(k) = 2^{\frac{k}{12}} f_{min}$$

    - time domain: convolution with complex kernel
    - to reduce computational costs $\rightarrow$ multiplication in frequency domain instead of time domain

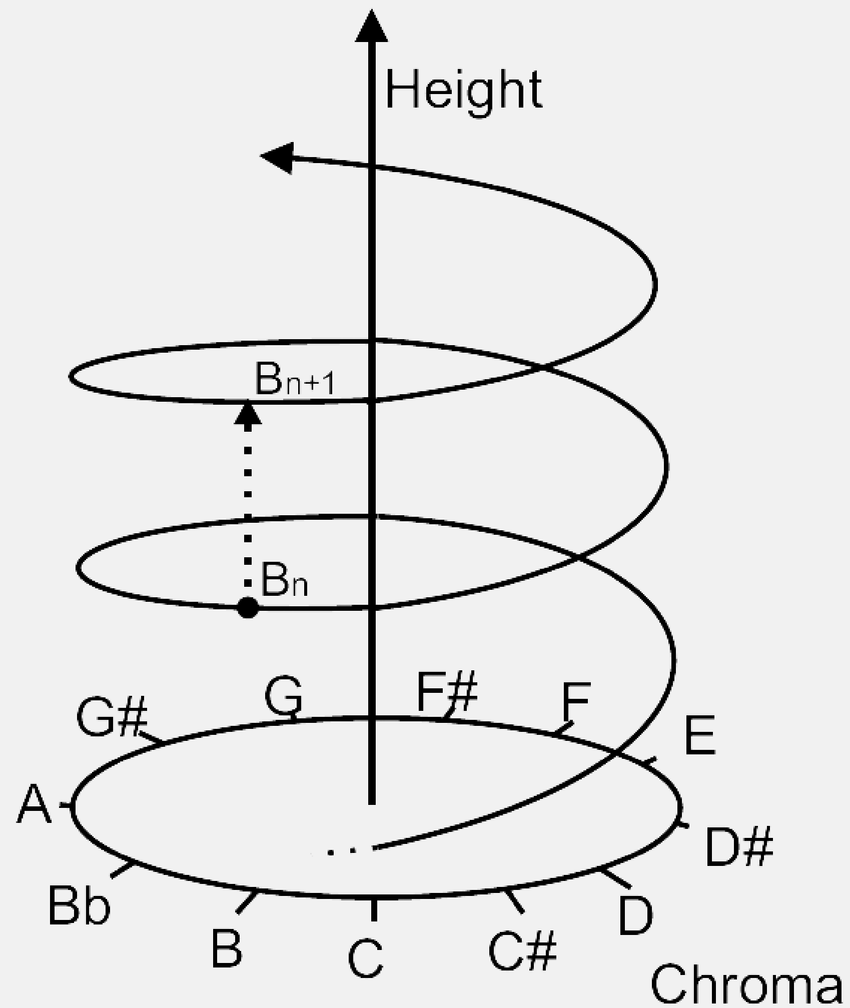


- Chroma = Harmonic Pitch Class Profile
  - chords do not carry information about tonal distribution within octaves
    - summarize energy of all octaves of a tone into a scalar
      - e.g. … B + b + b' + b'' + b''' …
  - 12 dimensional vector

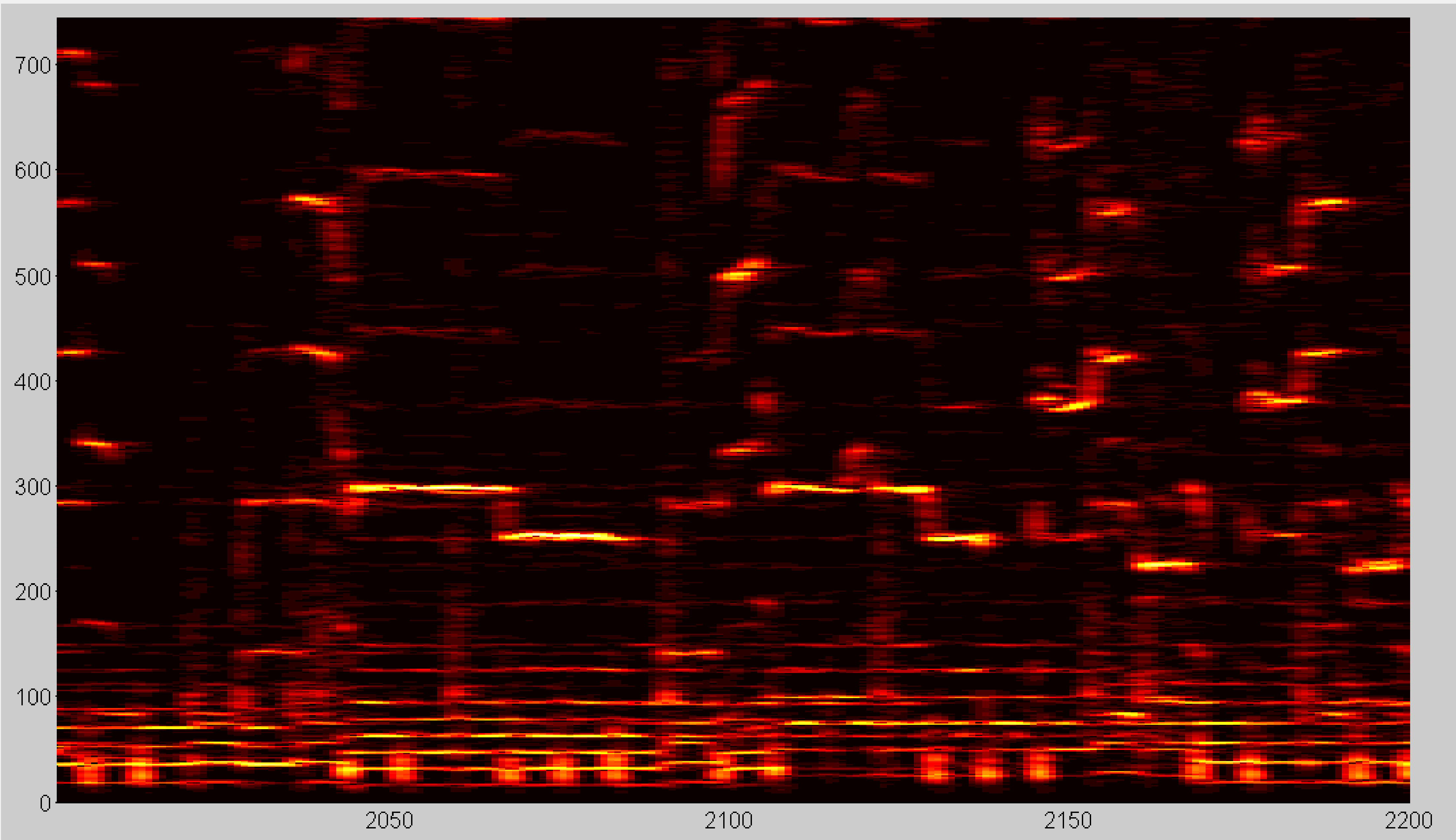
$$HPCP_b = \sum_{m=1}^{M} CQT[b + 12m]$$

$$1 \leq b \leq 12$$

- M … number of octaves involved

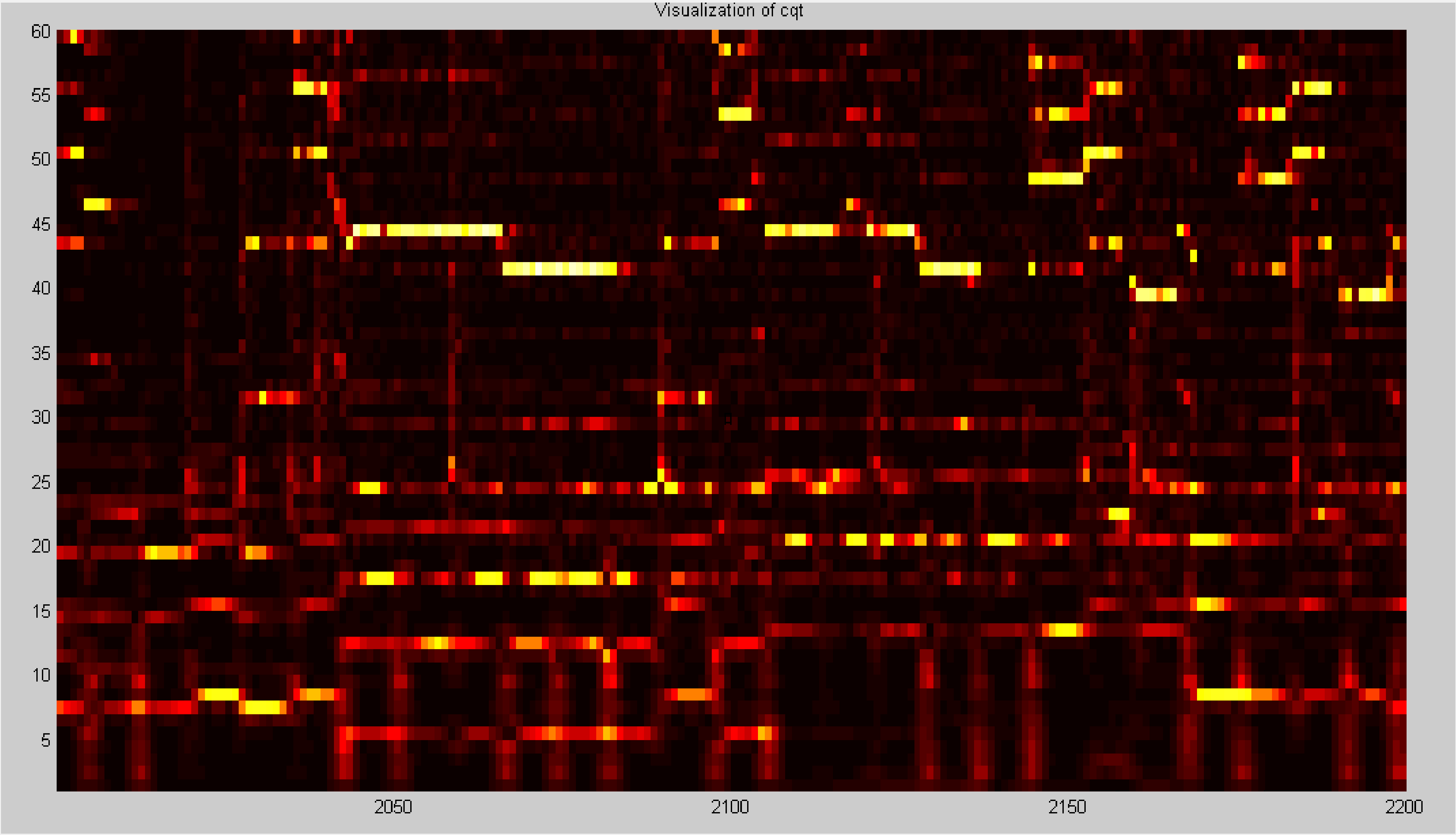


- Spectrogram vs. Constant-Q-gram vs. Chromagram



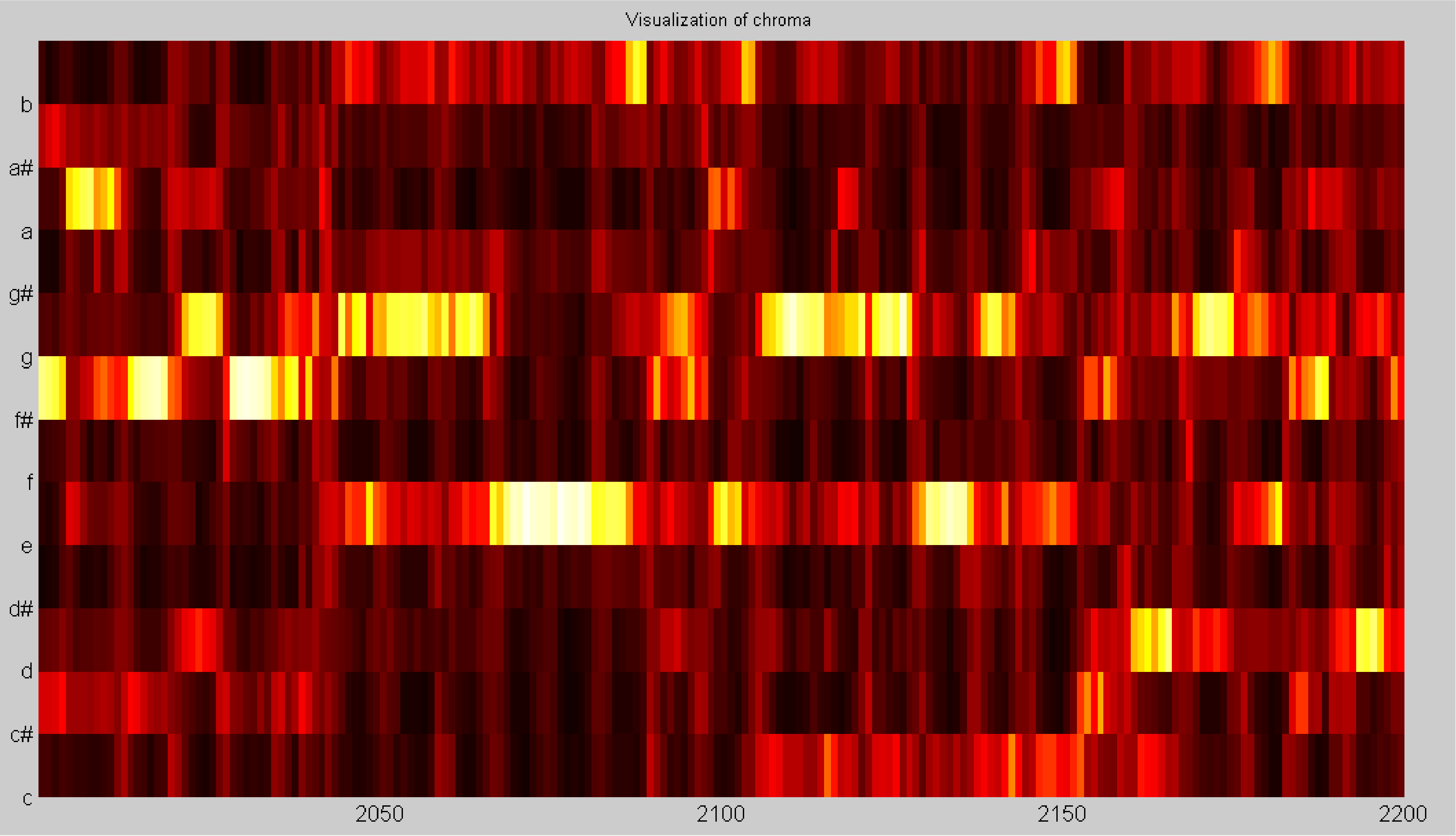

Spectrogram

- Spectrogram vs. Constant-Q-gram vs. Chromagram



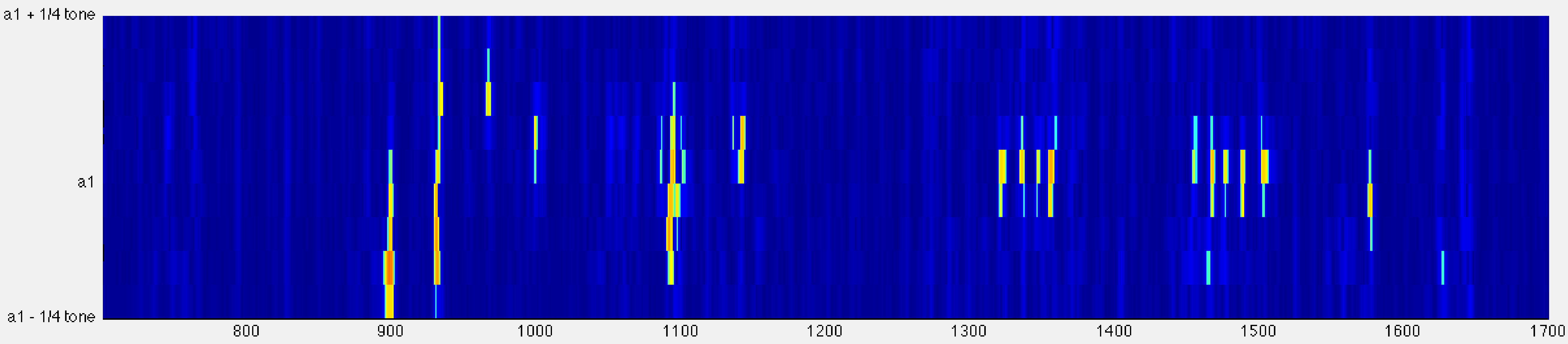

CQT-gram

- Spectrogram vs. Constant-Q-gram vs. Chromagram



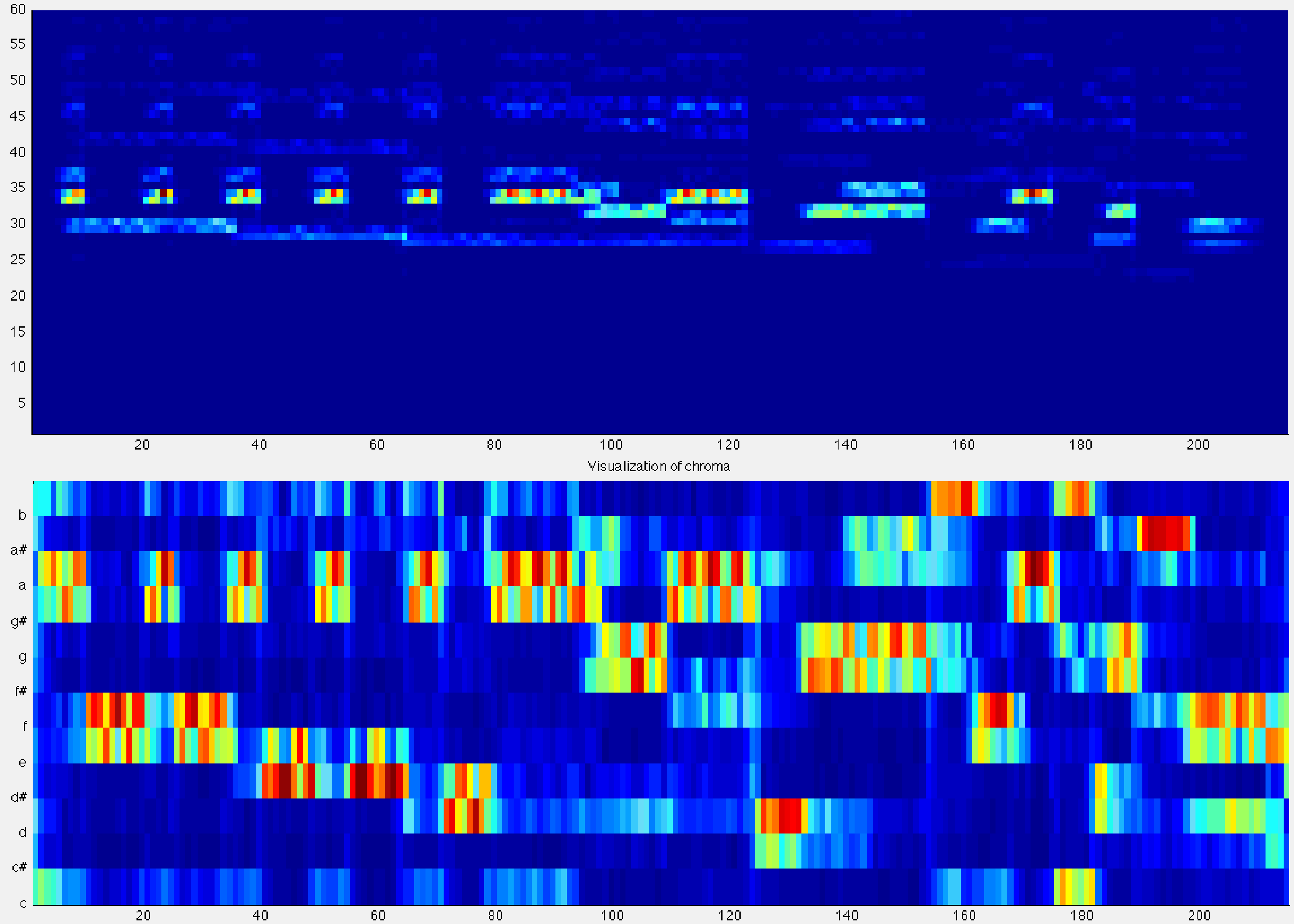

Chromagram

- What if the song is not tuned to 440Hz?

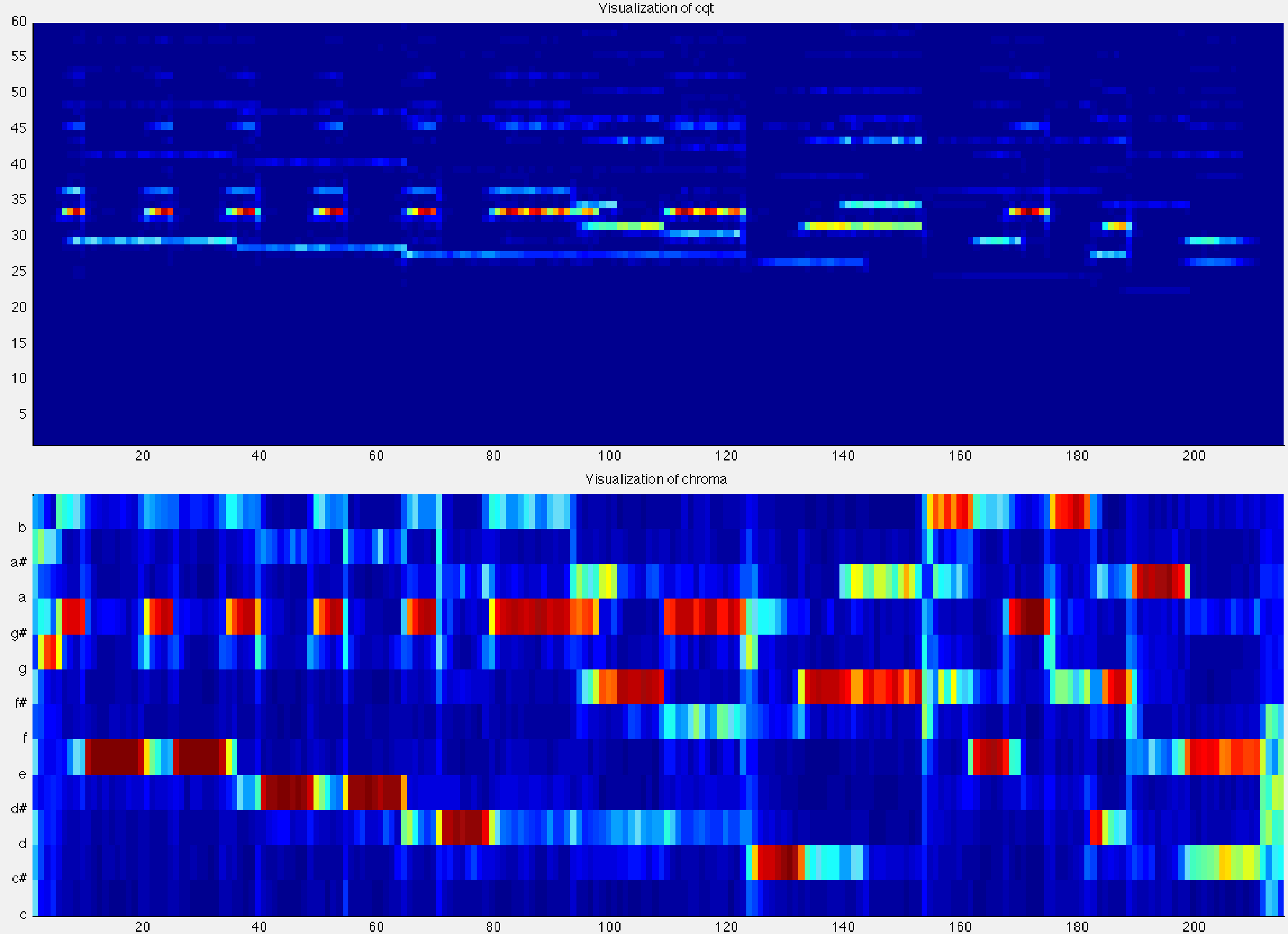



- observe a' at 440Hz +/- ¼ tone
- sum distributed energy over time
- pick maximum to detect tuning center
- use center as basis for Constant-Q Transform

- Intro of „Beatles – Strawberry Fields Forever“ BEFORE tuning:

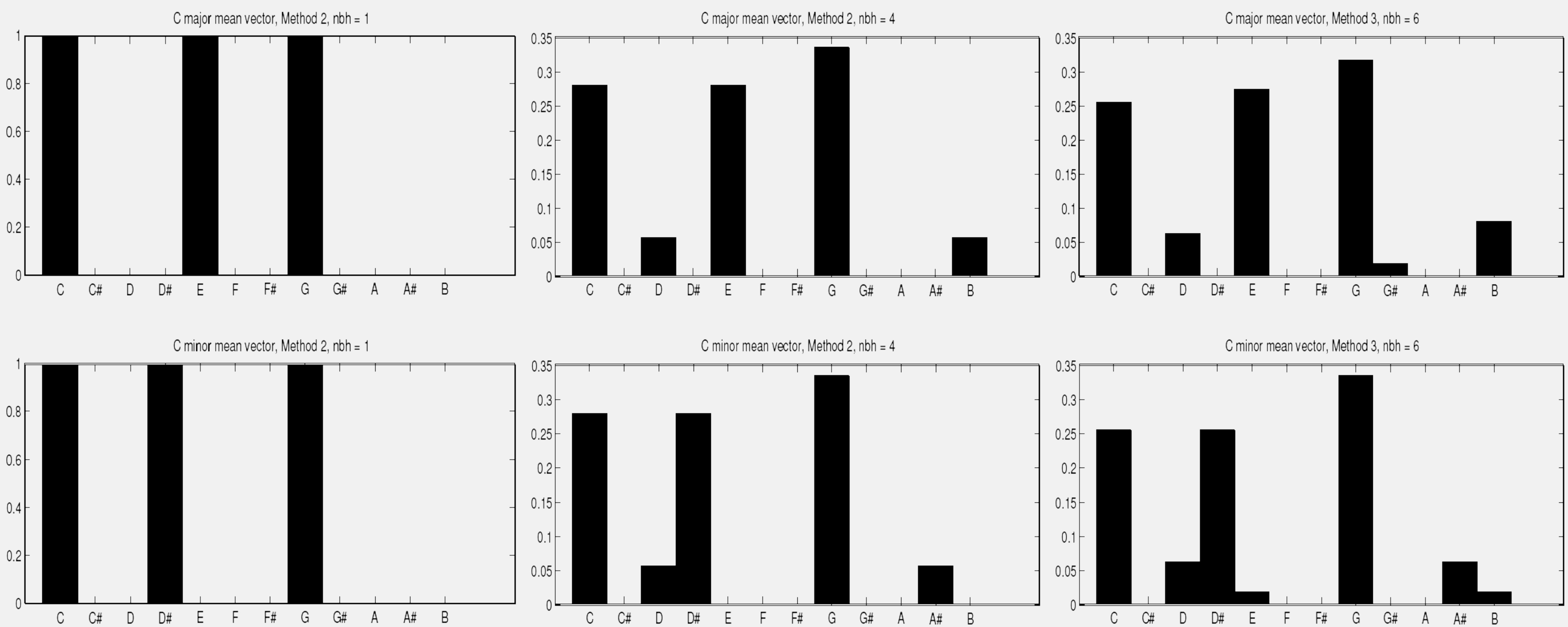


- Intro of „Beatles – Strawberry Fields Forever" **AFTER** tuning:



- Chord Detection
  - 2 commonly used methods based on chroma:
    1. correlation with a chord pattern
    2. Hidden Markov Model
  - Requirements for the resulting sequence:
    - musically meaningful
    - consistent
    - not necessarily perfect while consistent over time

- Method I: *direct correlation with chord patterns*
  - generate pattern for all major and minor chords
  - considdering *n* harmonics → improves performance
  - too many harmonics → overdefined system



- Analysis:
  - PRO:
    - performace realtively good
    - quick and easy implementation
  - CON:
    - patterns only include 3 tones → real life harmonies often contain tensions

      → ambiguities → false detections:

      e.g. F6 =?= d7
    - no intelligence concerning sequence of chords

- Method II: *Hidden Markov Model*
  - introduces additional intelligence
  - formal description:

$$\lambda = \{Q, A, O, B, \pi\}$$

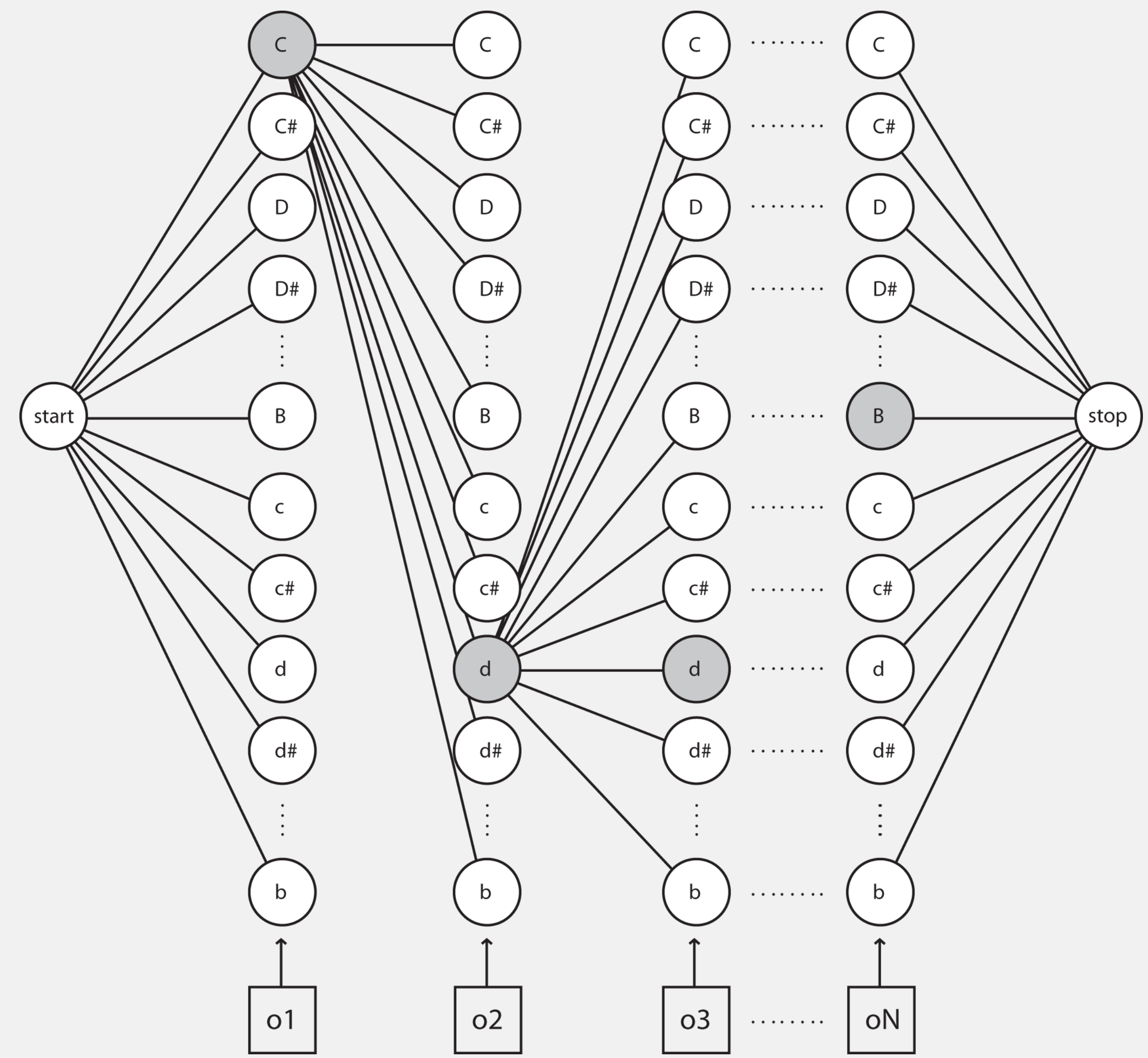
Q … set of available sates

A … transition probabilities

O … observations

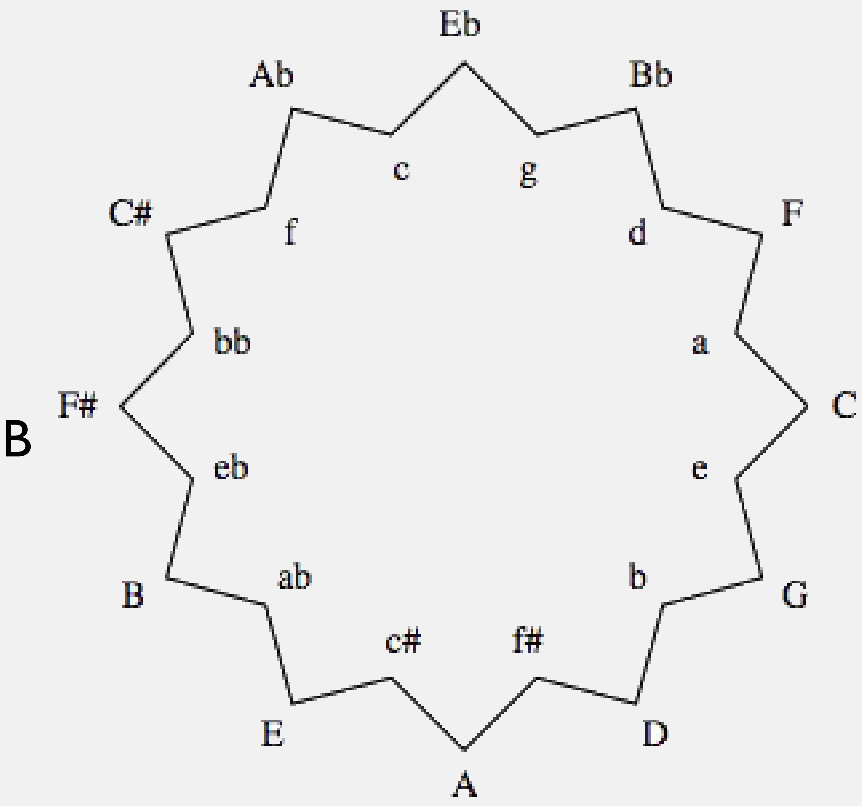
B … observation/emission probabilities

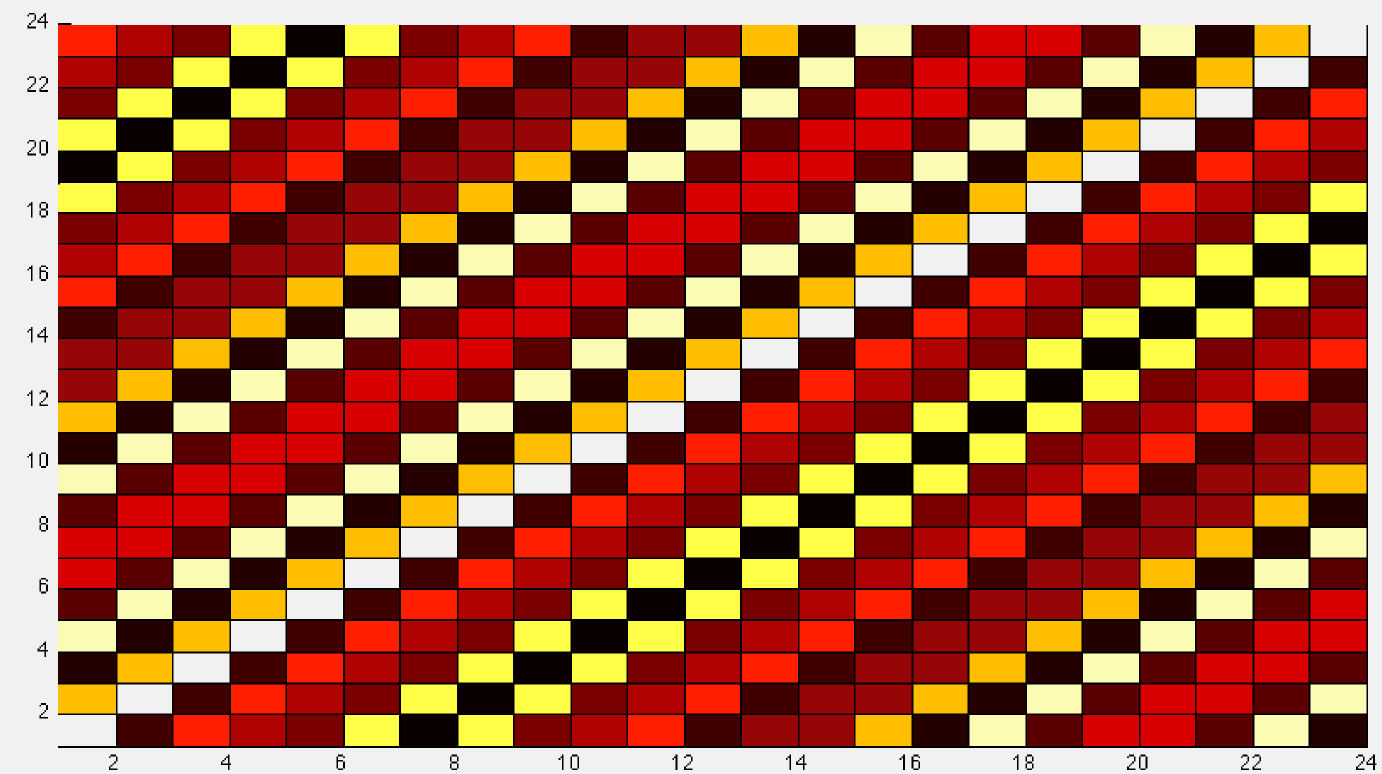
π … initial probabilities



- Defining the model:
  - Available states Q:
    - 12 major chords C, C#, D, D#, E, F, F#, G, G#, A, A#, B
    - 12 minor chords c, c#, d, d#, e, f, f#, g, g#, a, a#, b
  - Transition probabilities A:
    - derived from circle of fifths
    - defined distances determine probability of transition
    - close relatives: fifth, major/minor third

      → higher probabilities for transitions



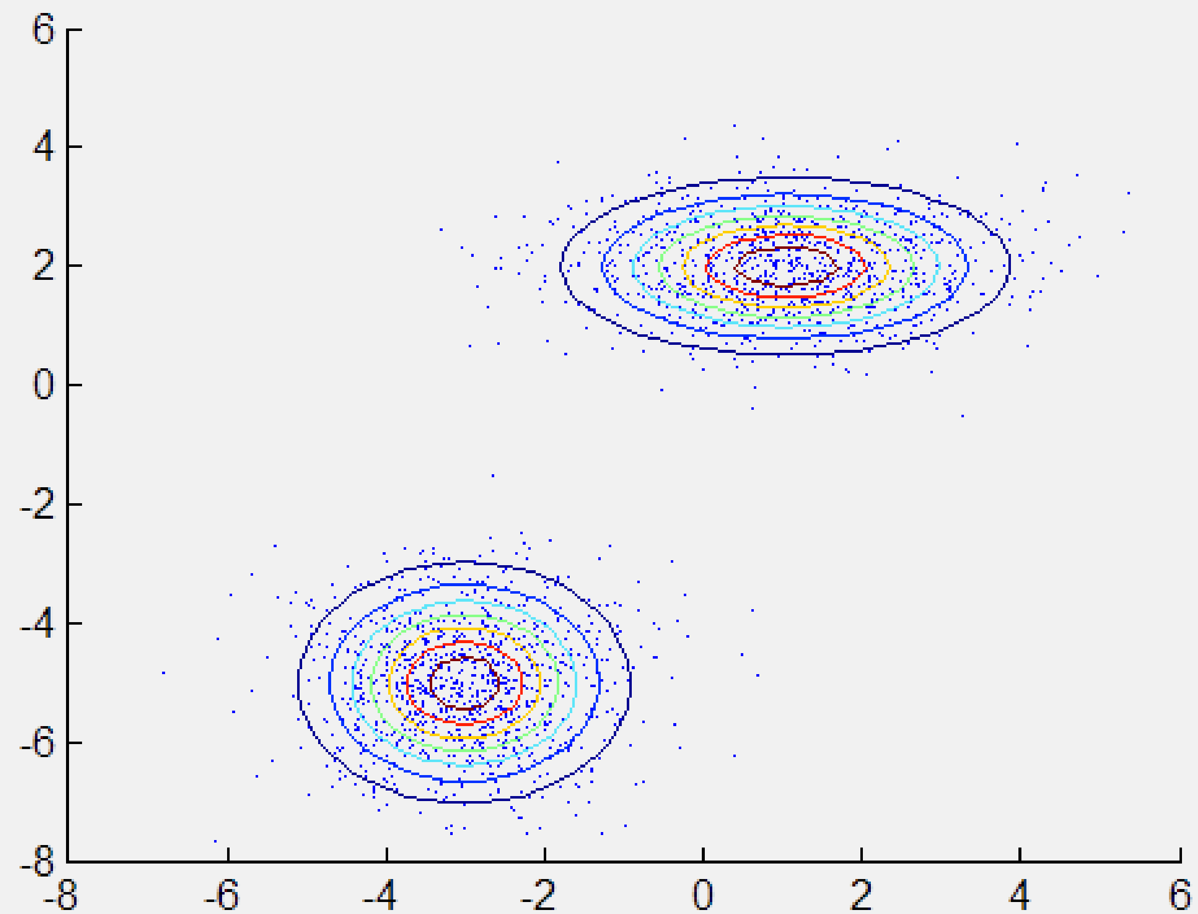



- Defining the model:
  - Observation probabilities B:
    - derived from chord patterns
    - Gaussian Mixture Models (GMMs)
      - each chord is modeled as multivariate

        Gaussian mixture

        → 24 12 dimensional mean vectors μ

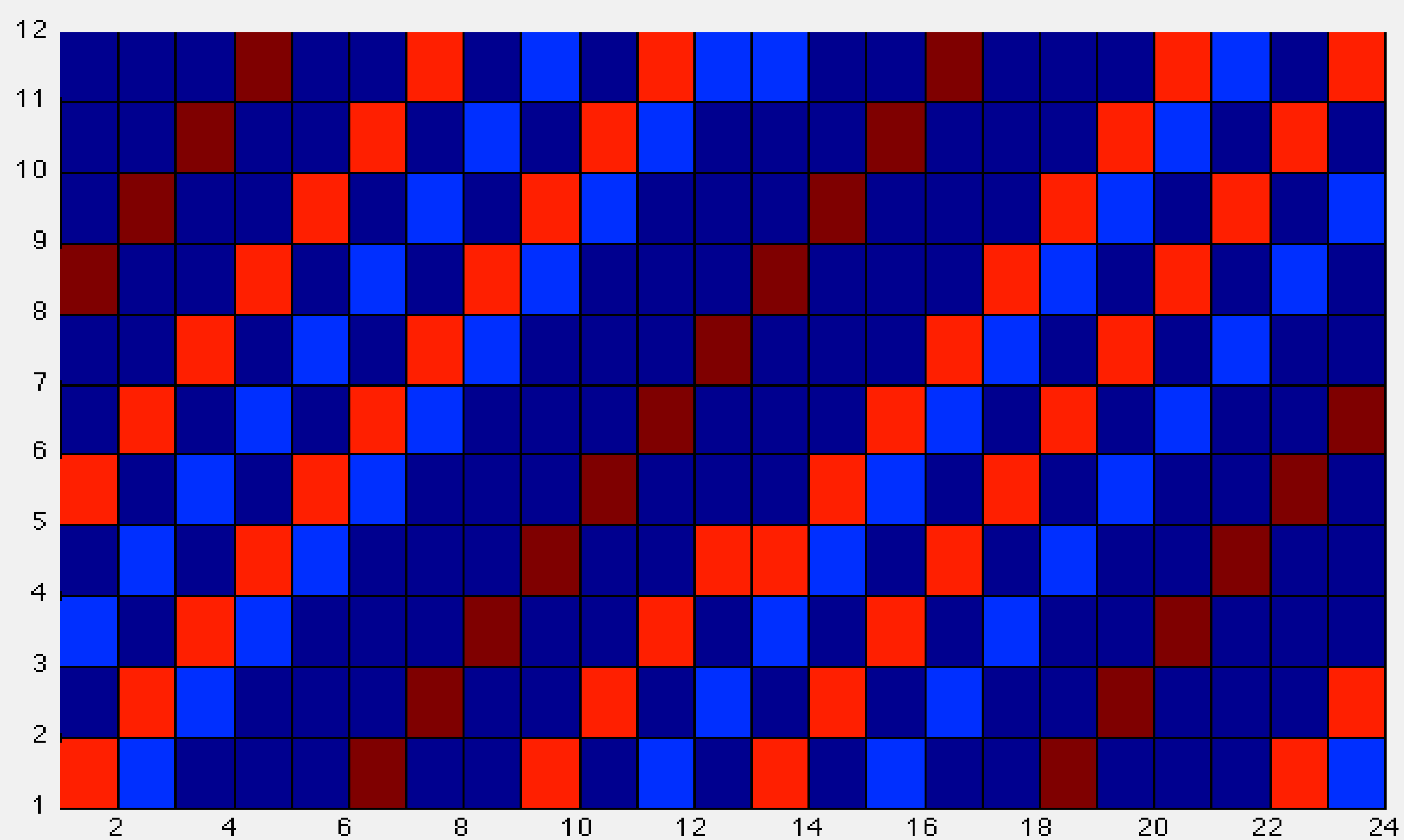
        → 24 12x12 covariance matrices Σ
    - order of GMMs determines computational costs – yet costly only once as no learning
  - Initial probabilities π
    - equally distributed

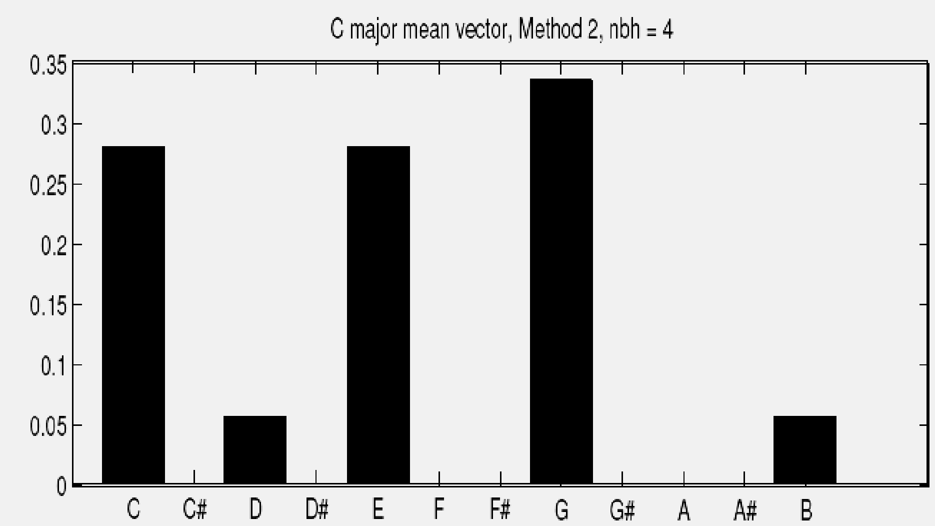


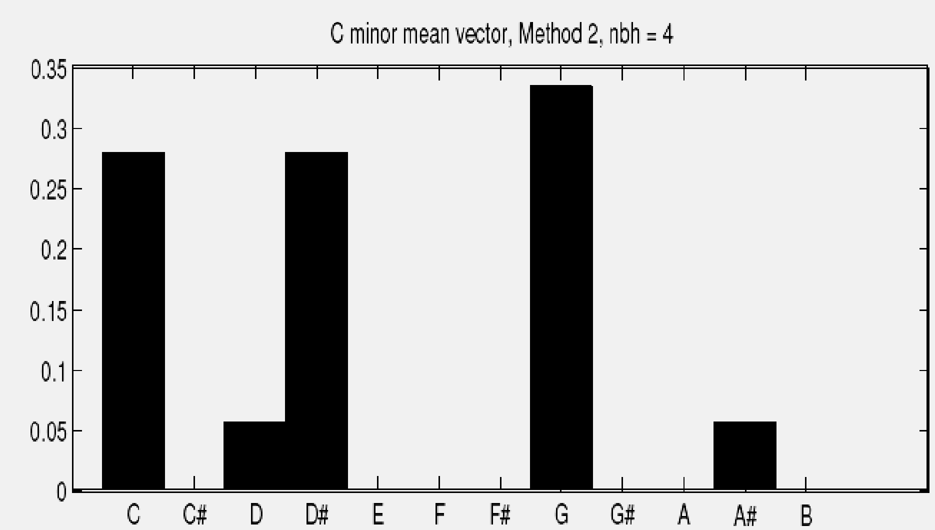
# features



- Means: μ matrix
  - 24 vectors for each chord in major an minor
  - basic 3 tones of a chord extended with n overtones
  - major and minor
  - same as used in direct correlation method



- Covariance matrices:
  - variance between pairs of feature dimensions
  - define 'form' of gaussian in 12 dimensional feature space
  - each μ vector has a corresponding covariance matrix
  - eg. in 2d:

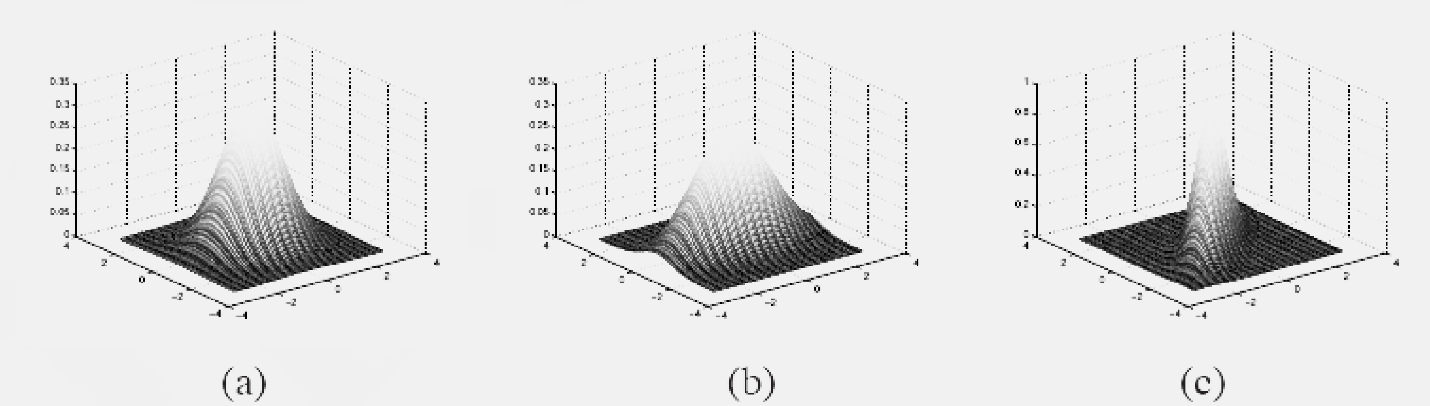
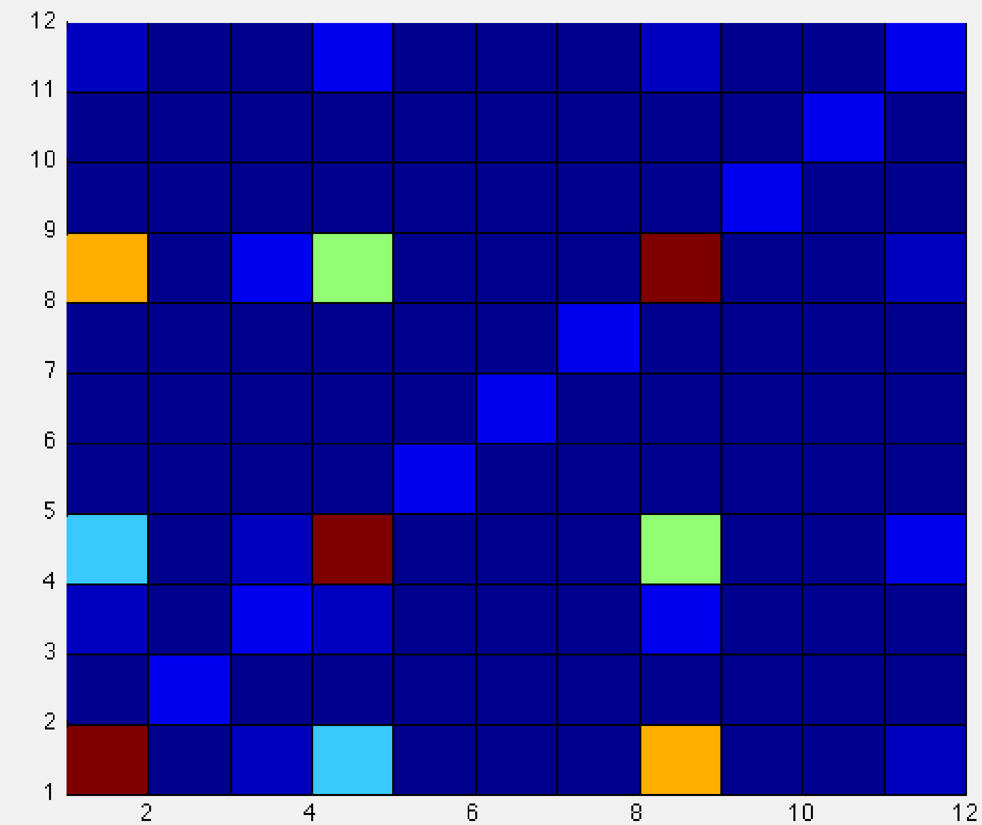
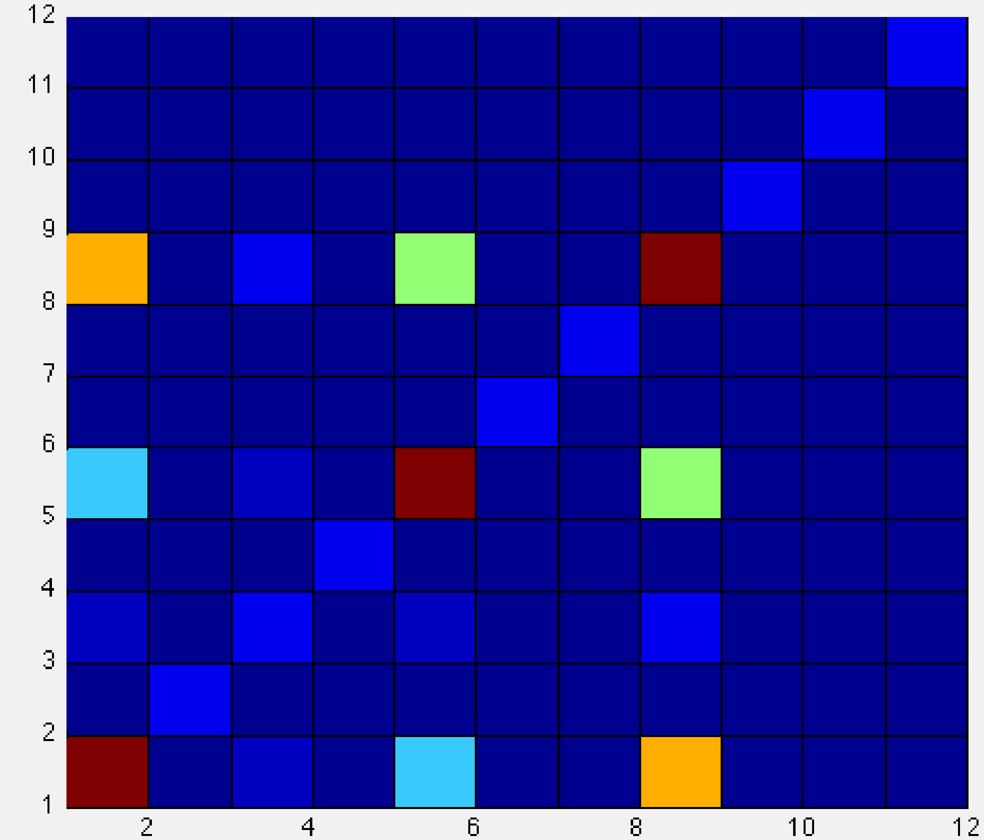


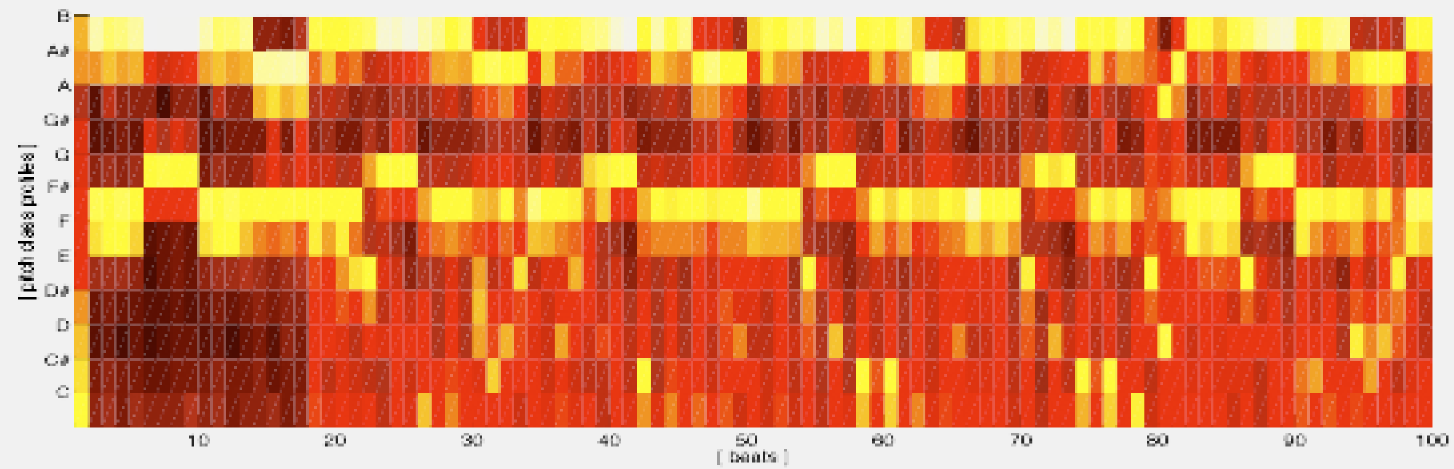

**Figure 9.20** Three different multivariate Gaussians in two dimensions. The first two have diagonal covariance matrices, one with equal variance in the two dimensions $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, the second with different variances in the two dimensions, $\begin{bmatrix} .6 & 0 \\ 0 & 2 \end{bmatrix}$, and the third with non-zero elements in the off-diagonal of the covariance matrix: $\begin{bmatrix} 1 & .8 \\ .8 & 1 \end{bmatrix}$.



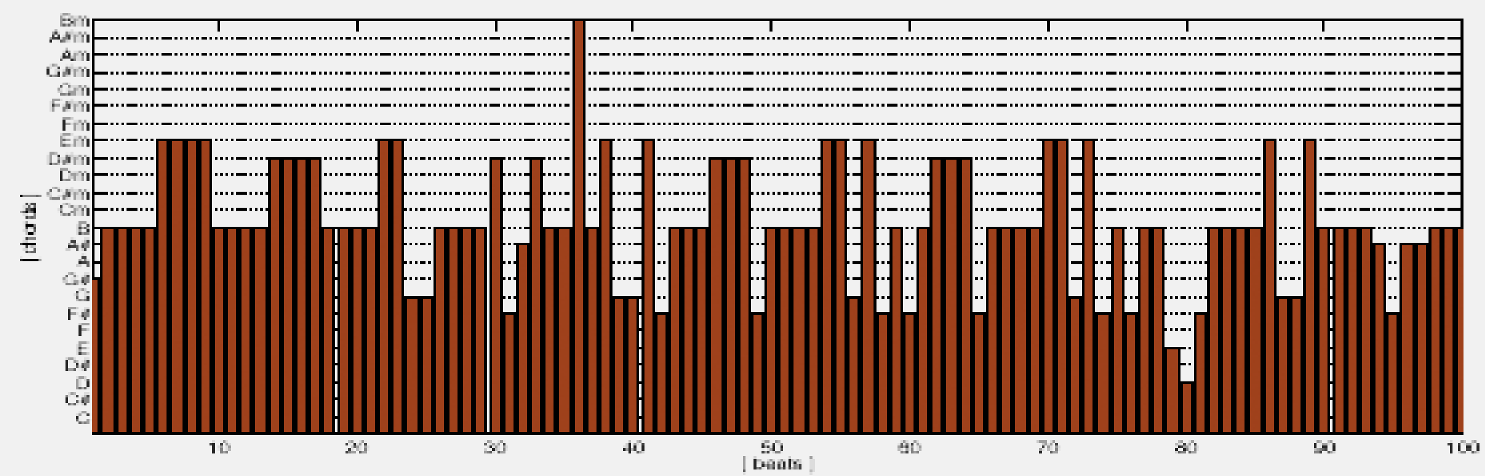



- Letting the model work...
  - very appropriate results
  - not perfect but very consistent

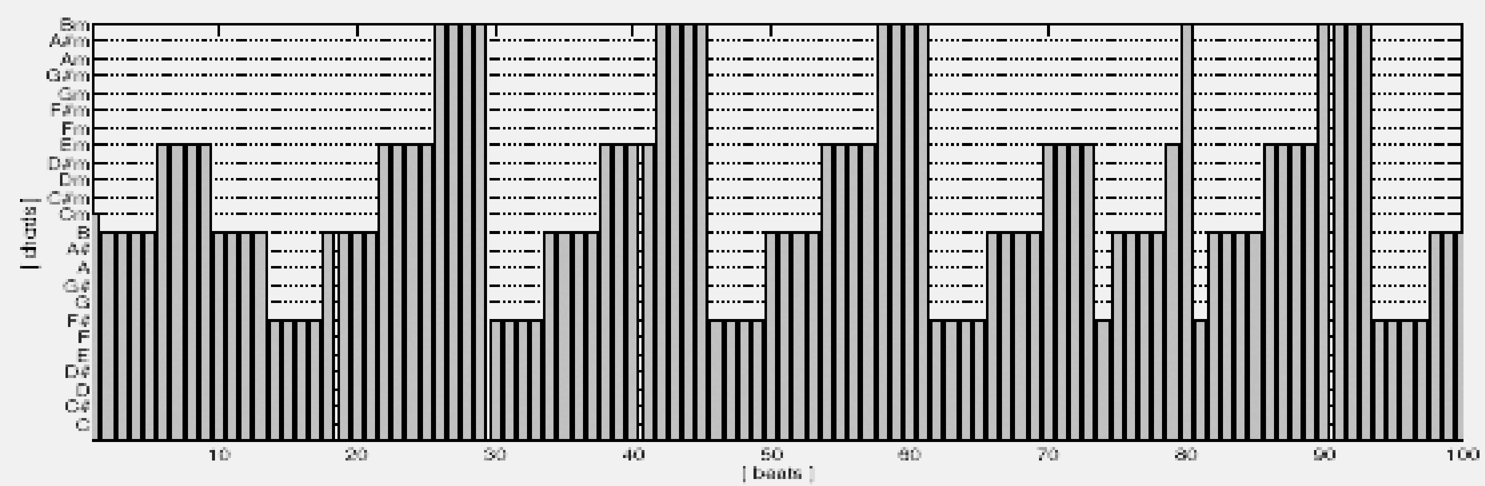
  → necessary for pattern recognition




*beat averaged chroma of "Portishead - Wandering Star"*

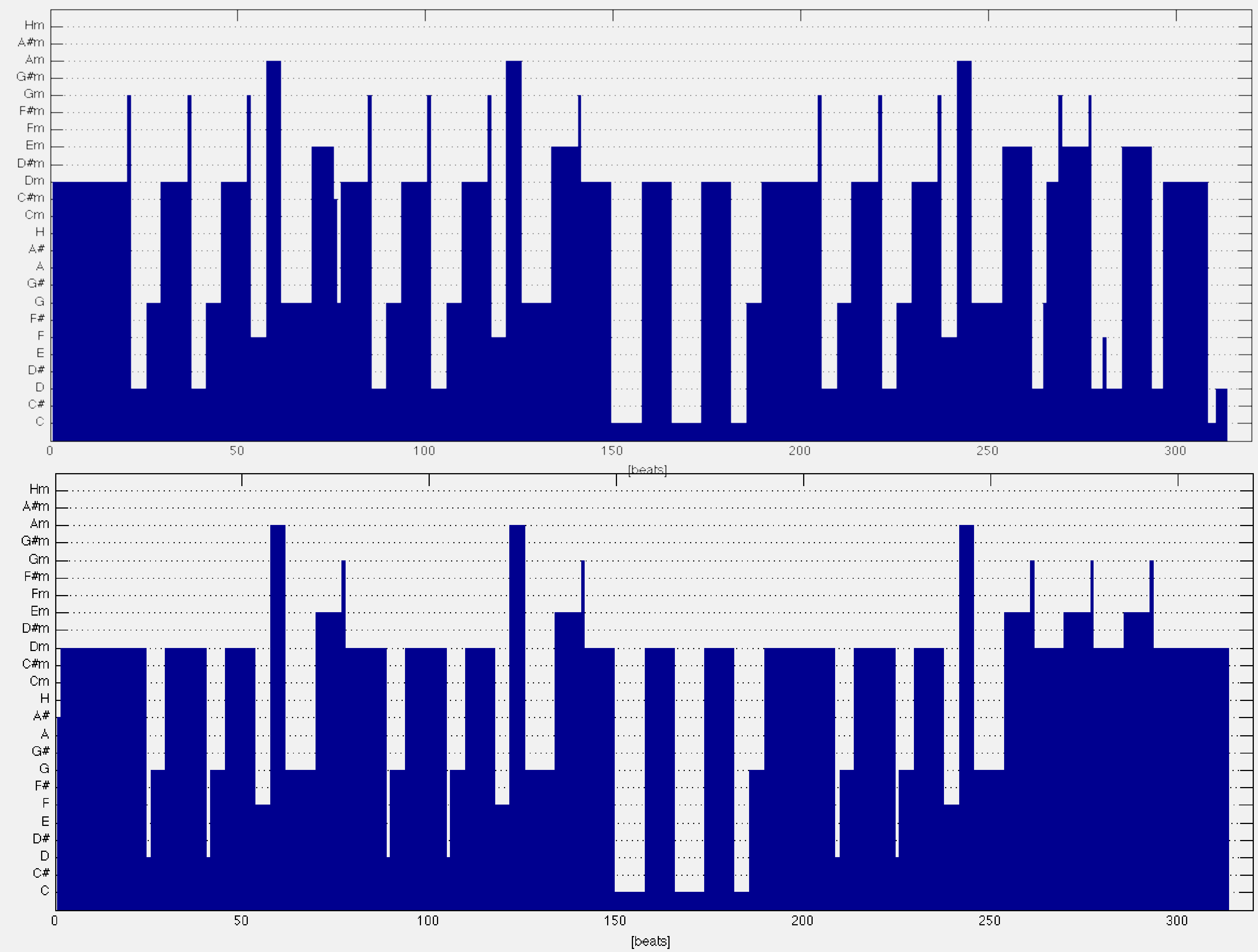



*directly detected chords by correlating chroma with chord patterns*



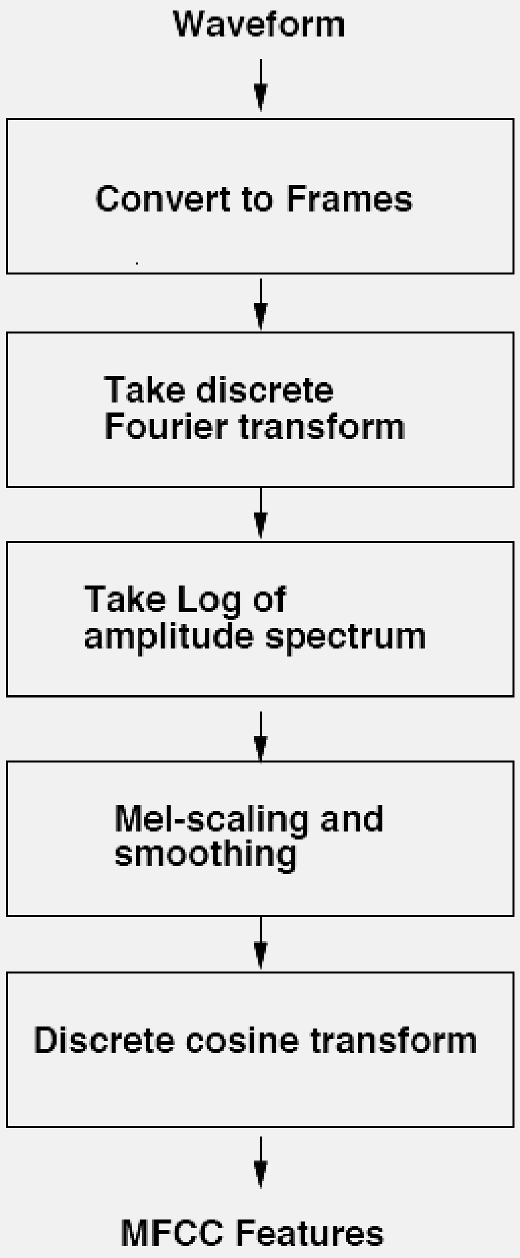

*chords estimated by HMM from chroma*

- so why don't we use a trained HMM?
  - Baum-Welch (EM) algorithm trains transition and observation probabilities
  - need for an appropriate training corpus
  - training → smoothing
    - loss of detailed information
    - decreases performance of pattern recognition (also for human)
    - example: REM – Automatic For The People
      - untrained vs. trained



- MFCC:
  - commonly used in speech signal processing
  - measure to describe spectral properties of signal
  - adapted to human perception
  - compact/efficient measure
  - 10 MFCC components used in algorithm

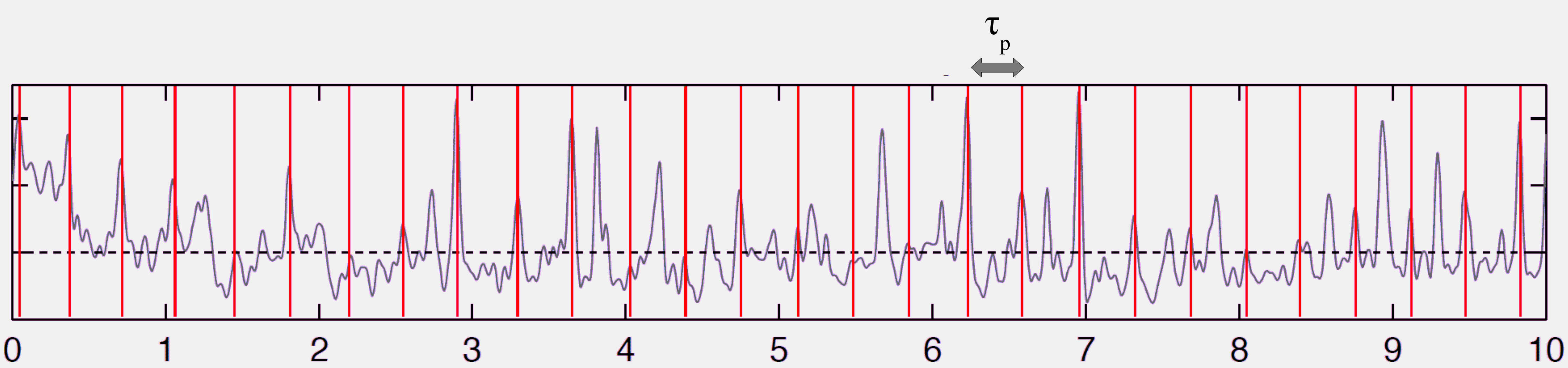
$$MFCC = DCT\left\{\log(|FFT|) \cdot W_{Mel}\right\}$$



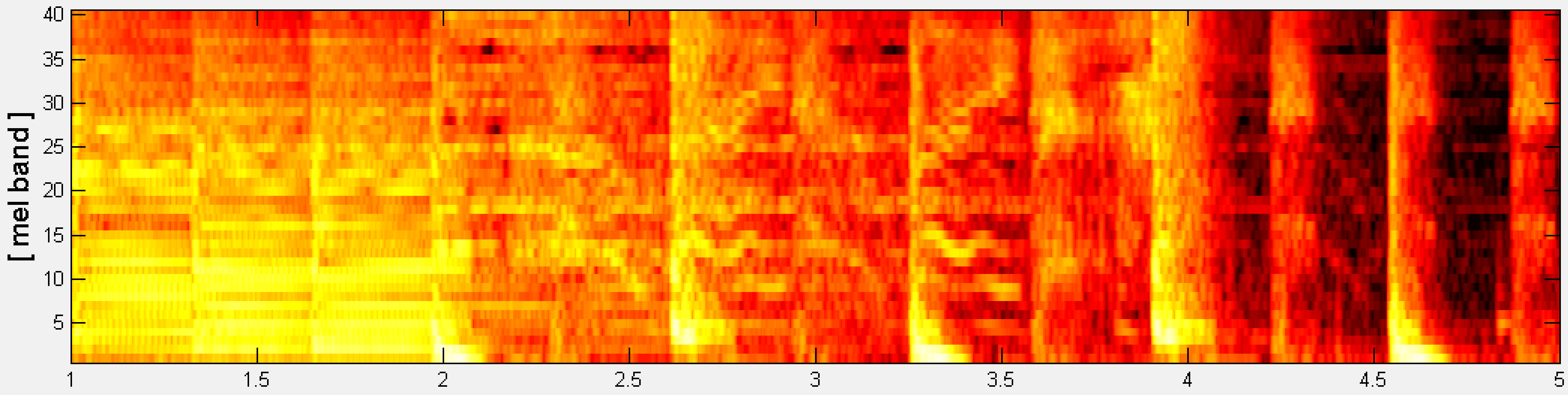
- possible „borders“
  - fixed number of frames
    - onsets ignored → large influence of transient events
  - onsets (→ onset detection)
    - spectral flux, lpc-error signal, complex flux, …
    - very large diversity in duration
  - beats (→ beat detection)
    - musically stable sections
    - (almost) constant → (almost) same time instances for comparison

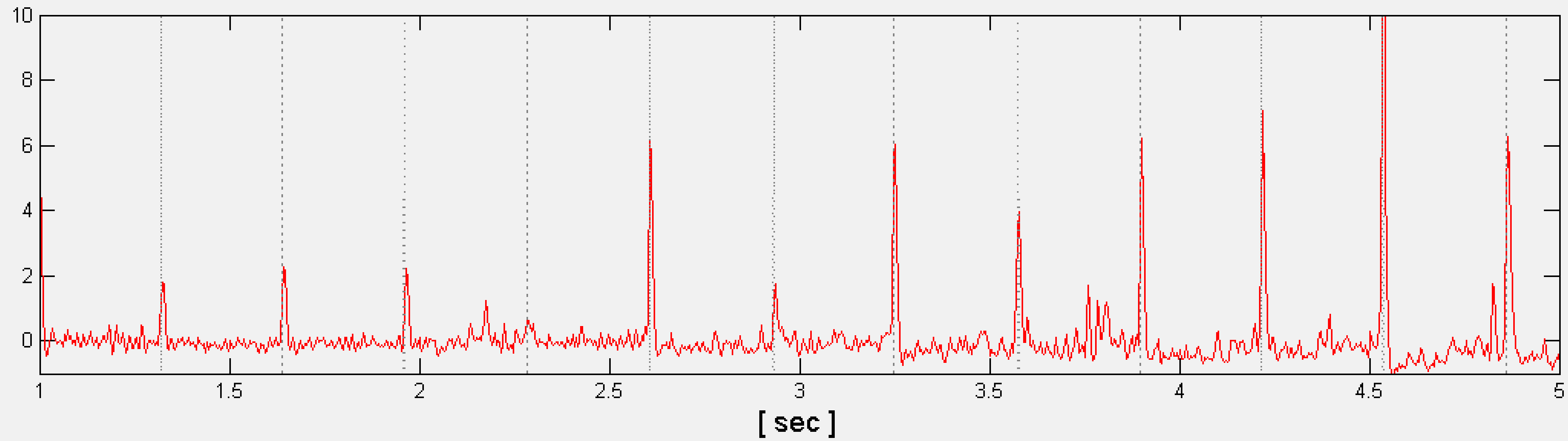
- approach by Dan Ellis
  - maximization of a decision cost function
    - $$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$$
      - $t_i$ → best scoring time sequence (position of „best“ beat borders)
      - $O(t_i)$ → perceived onset positions
      - $F(t_i\text{-}t_{i-1}, \tau_p)$ → locally-constant inter onset intervalls $F(t_i\text{-}t_{i-1}, \tau_p)$

O(t) and target period $\tau_p$ as input

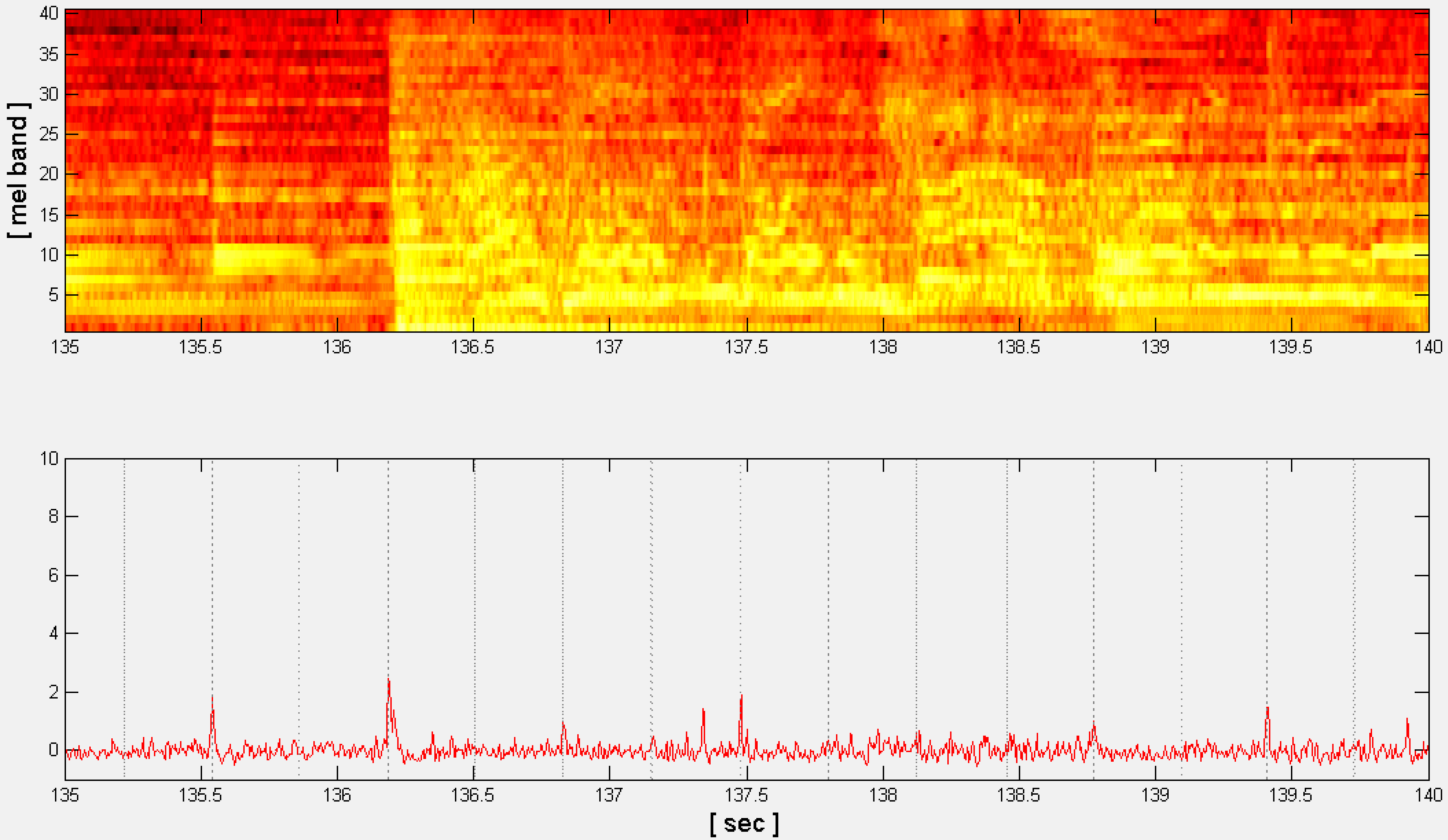


- onset strengths envelope → O(t)
  - 40 Mel Bands → 1st order difference

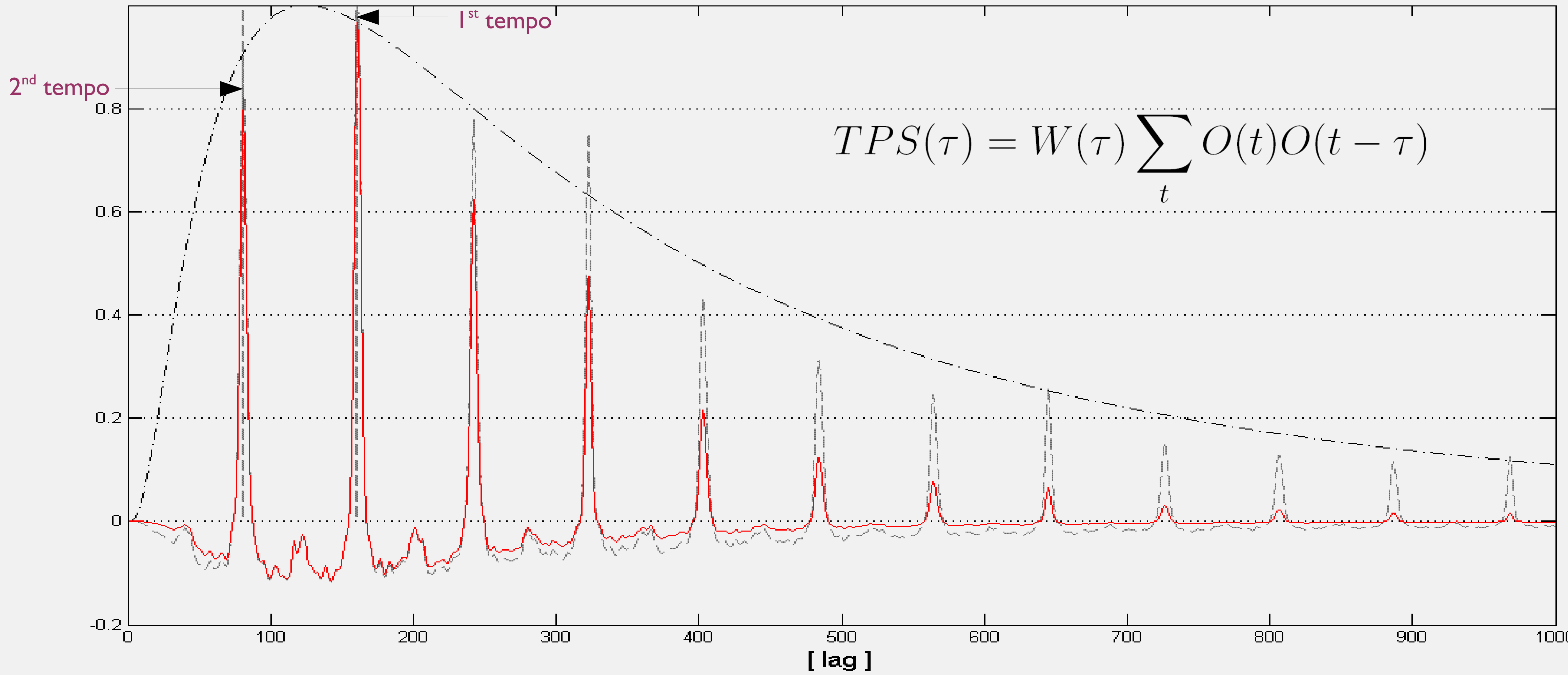


- onset strengths envelope → O(t)
  - 40 Mel Bands → 1st order difference



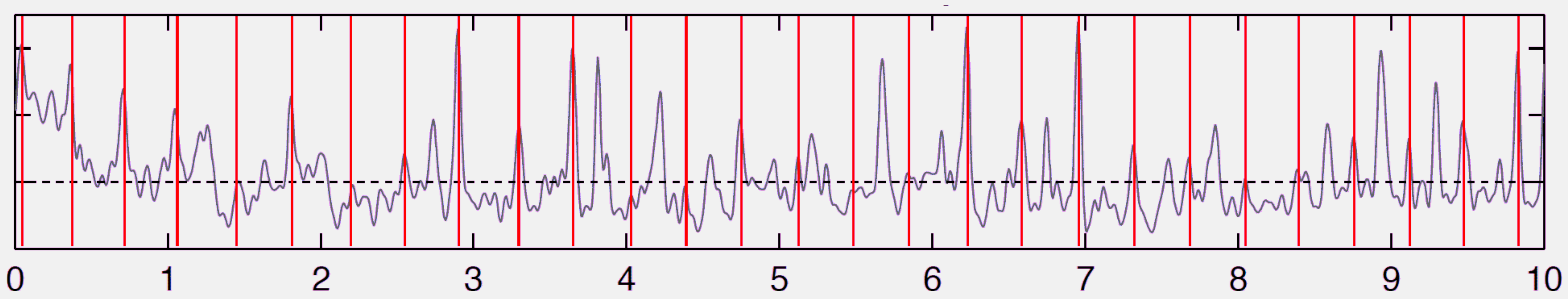
- target tempo
  - autocorrelation → perceptual weighting window ($\tau_0$, $\sigma$) → primary tempo
  - 2 beat estimates → secondary tempo period (0.33, 0.5, 2, 3)
    - use largest peak of secondary tempo → compare to primary tempo → use faster one




$$TPS(\tau) = W(\tau) \sum_t O(t) O(t - \tau)$$

- output
  - indices of the optimal set of beat times
  - best scoring time sequence $t_i$
    - $$C(\{t_i\}) = \sum_{i=1}^{N} O(t_i) + \alpha \sum_{i=2}^{N} F(t_i - t_{i-1}, \tau_p)$$

$$F(\Delta t, \tau) = -\left(\log \frac{\Delta t}{\tau}\right)^2$$



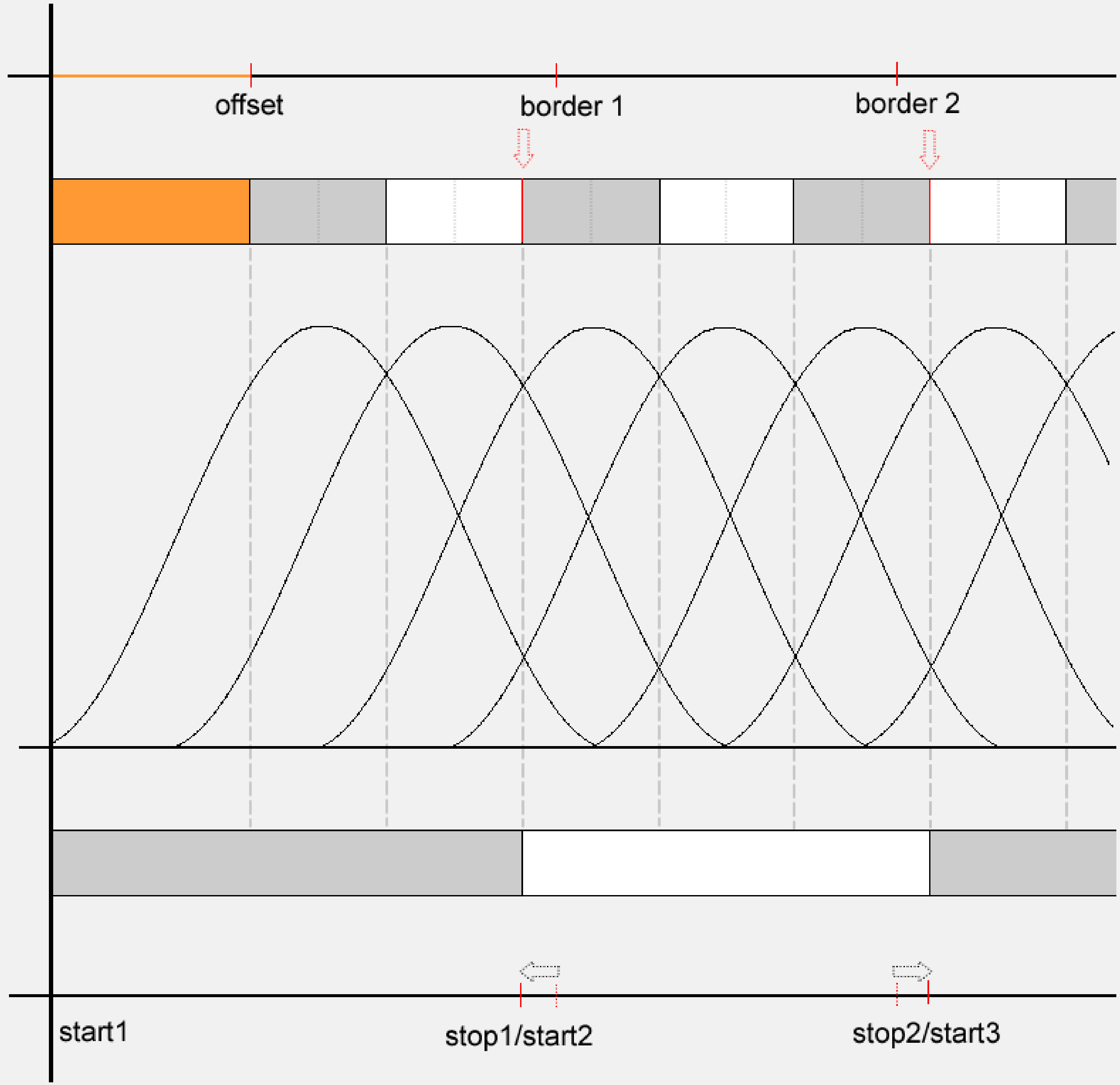
*sample borders*

*align to frame borders*

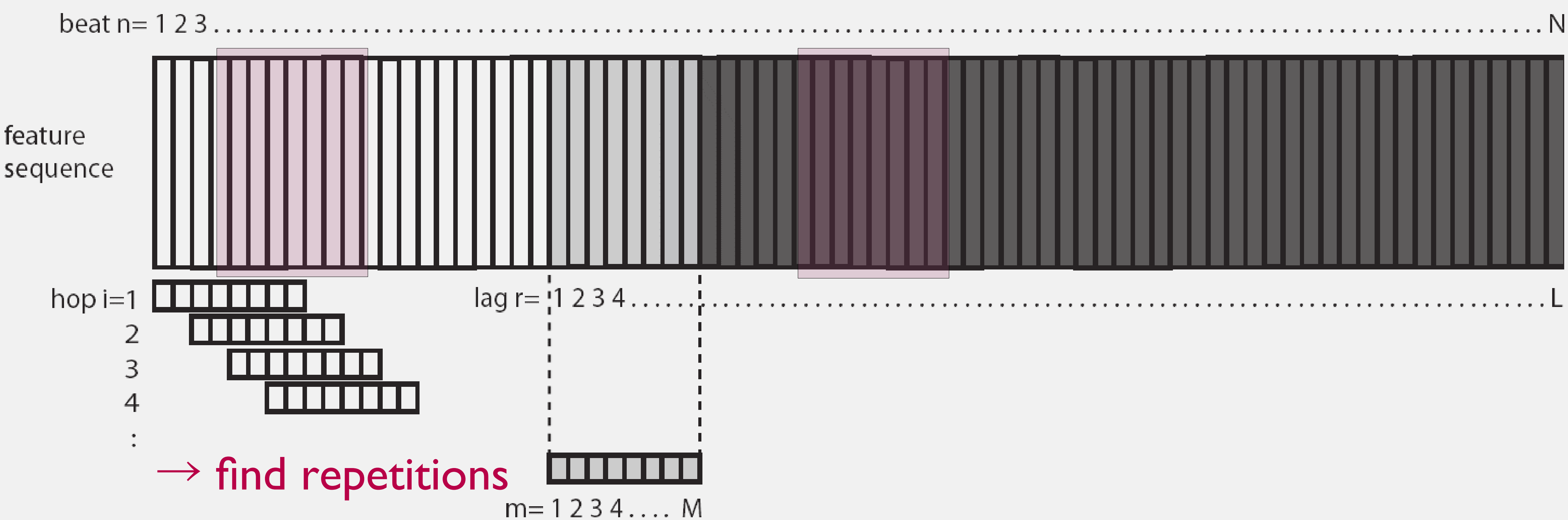
*¾ overlap of frames*

*frame merging*
*feature averaging*

*segment borders*

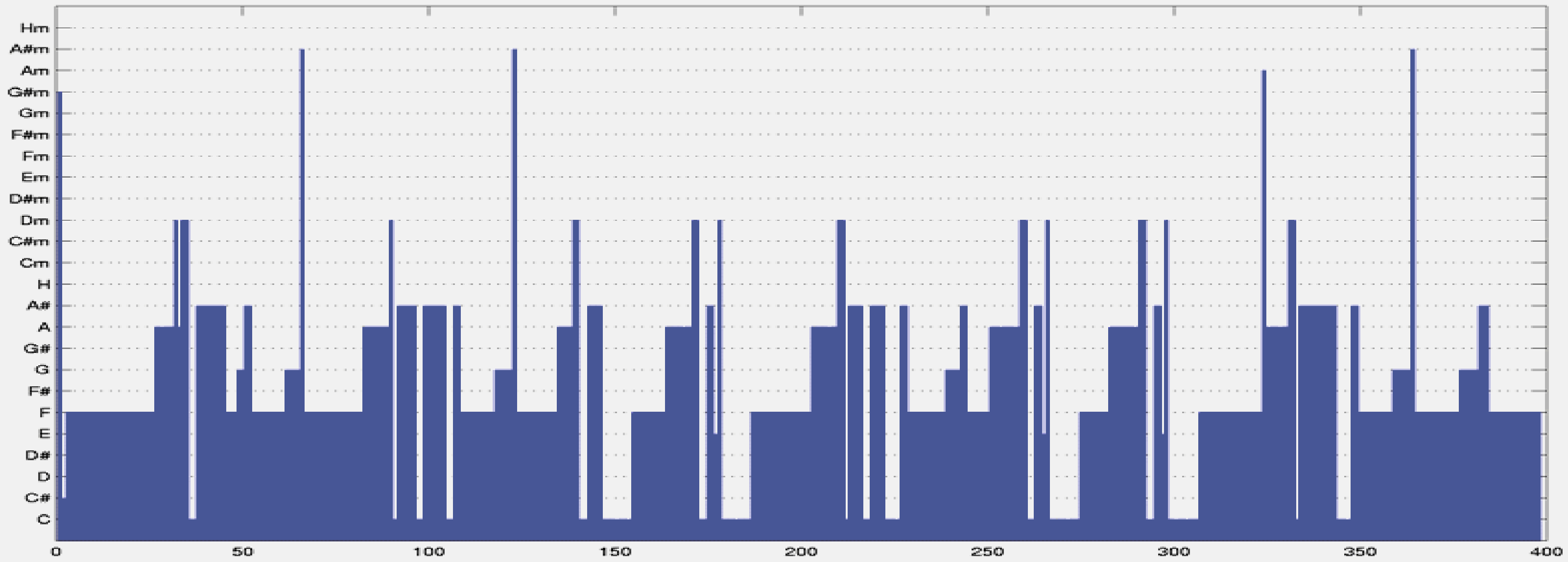


- detect repeated patterns (within feature sequences)

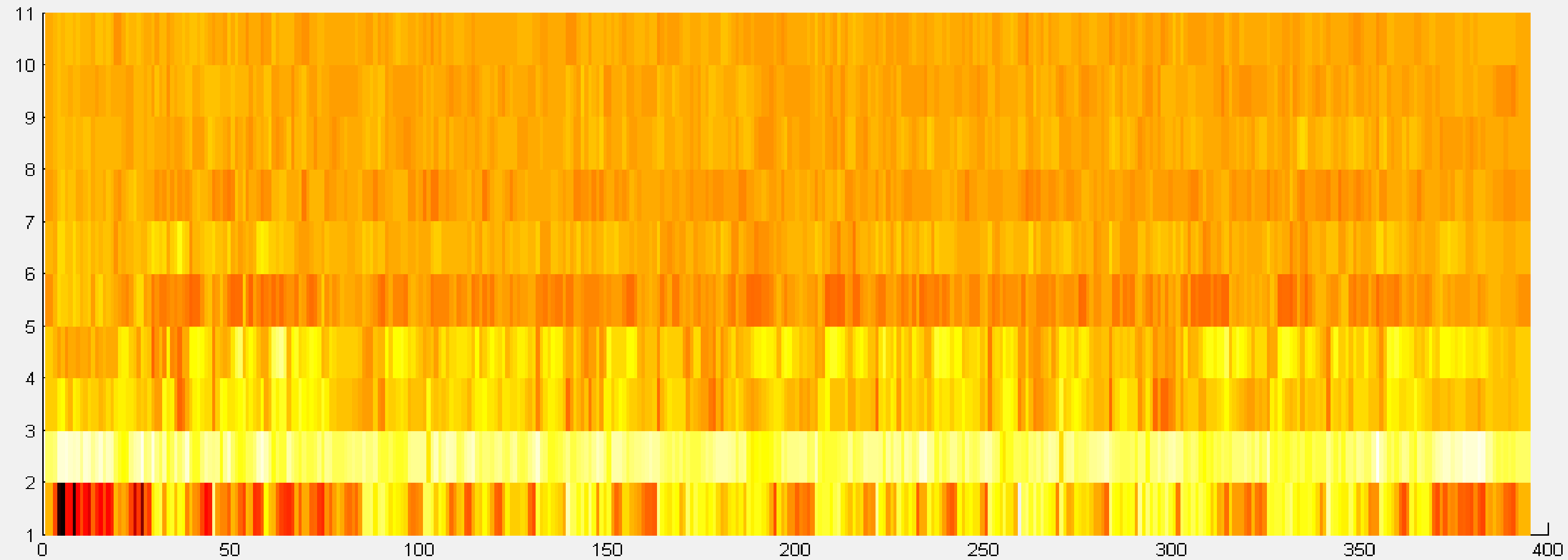



- segmentation of feature (vector) sequence $V[1, n]$
  - overlapped segments of fixed length $s_i = V[j, j+N-1]$
  - match each segment ($s_i = V[j, j+N-1]$) with feature sequence starting from this segment $V[j, n]$

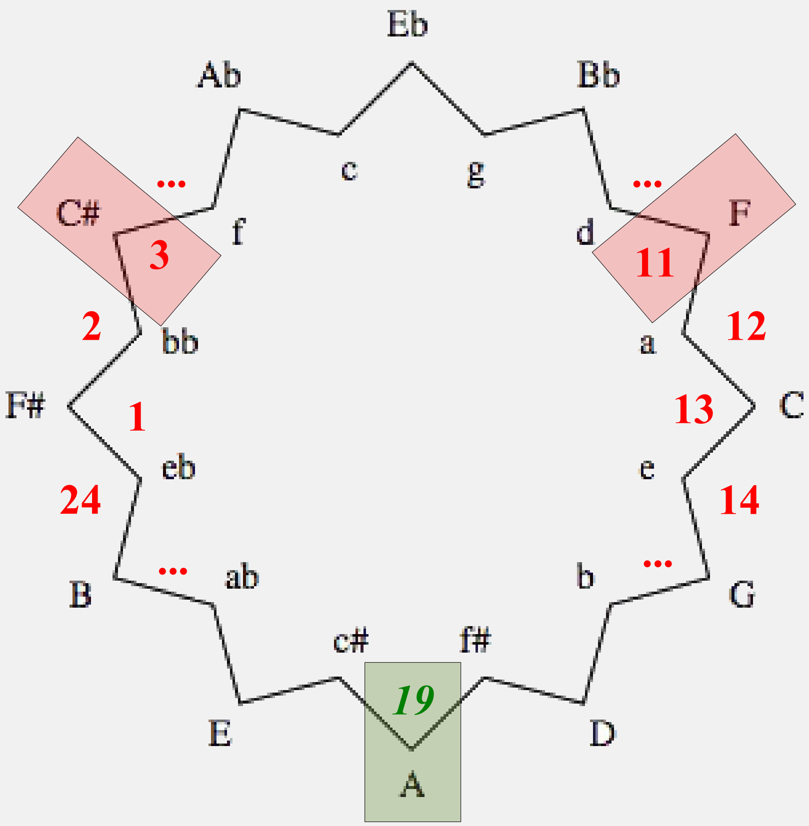
- calculation of distances
  - chord sequences (scalar numbers) → one dimensional




  - mfcc vectors → 10 dimensional vectors



- calculation of distances
  - „one dimensional“ features (e.g. chord numbers)
    - $$d_c(v_m, v_r) = \frac{1}{12}\begin{cases} |v_m - v_r| & \text{if} \quad |v_m - v_r| \leq 12 \\ 12 - \bmod|v_m - v_r| & else \end{cases}$$



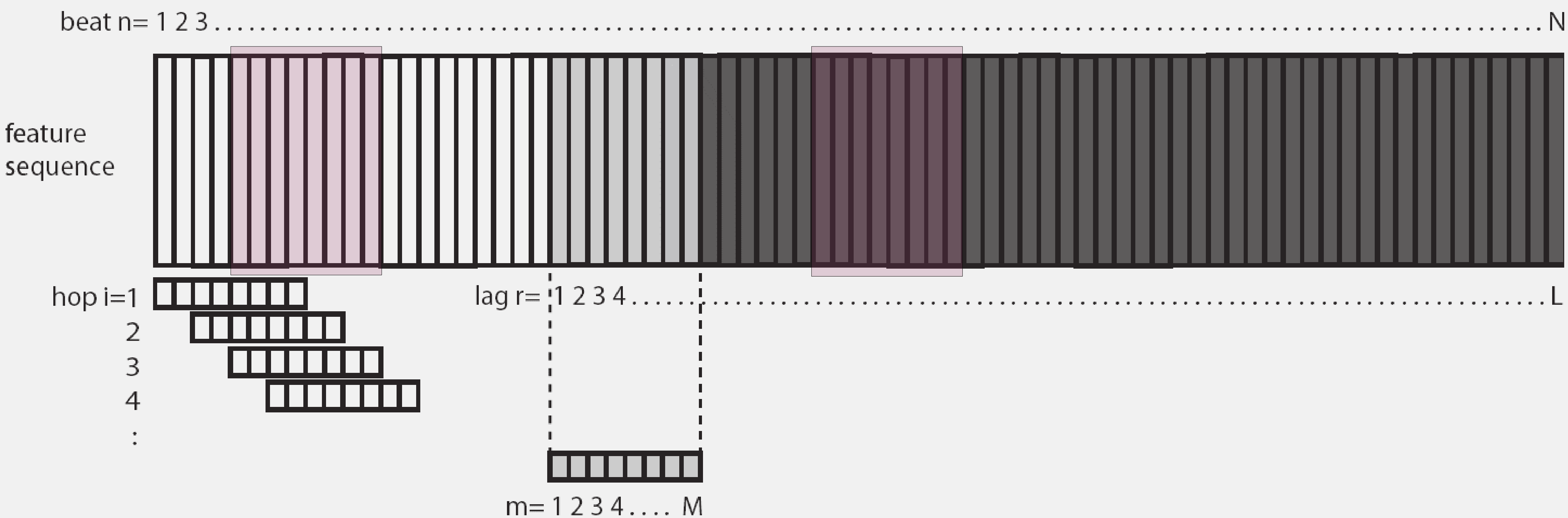

A → F: 19 – 11 = 8
A → C#: 12 – mod(19 – 3 ,12) = 8

  multidimensional features (e.g. mfcc vectors)
    - $$d_{MFCC}(\vec{v_m}, \vec{v_r}) = 0.5 - 0.5\frac{\vec{v_m} \bullet \vec{v_r}}{|\vec{v_m}||\vec{v_r}|}$$
    - normalized dot-product → modified cosine distance

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = |b||a|\cos\theta$$

$$\cos\theta = \frac{a \cdot b}{|b||a|}$$

- find (positions) of repeated patterns
  - approximate pattern matching (*hop = 2, segment length = 8*)




  - dynamic programming → sequence alignment
    - find best matches inside feature sequence
    - insertions and delitions allowed

- dynamic programming matrix
  - define cost of substitution, deletion and insertion
  - cost of substitution = "distance" $d_{m,r}$
  - cost of insertion and deletion $e = (0.1 + d_{m,r})\, e_0$

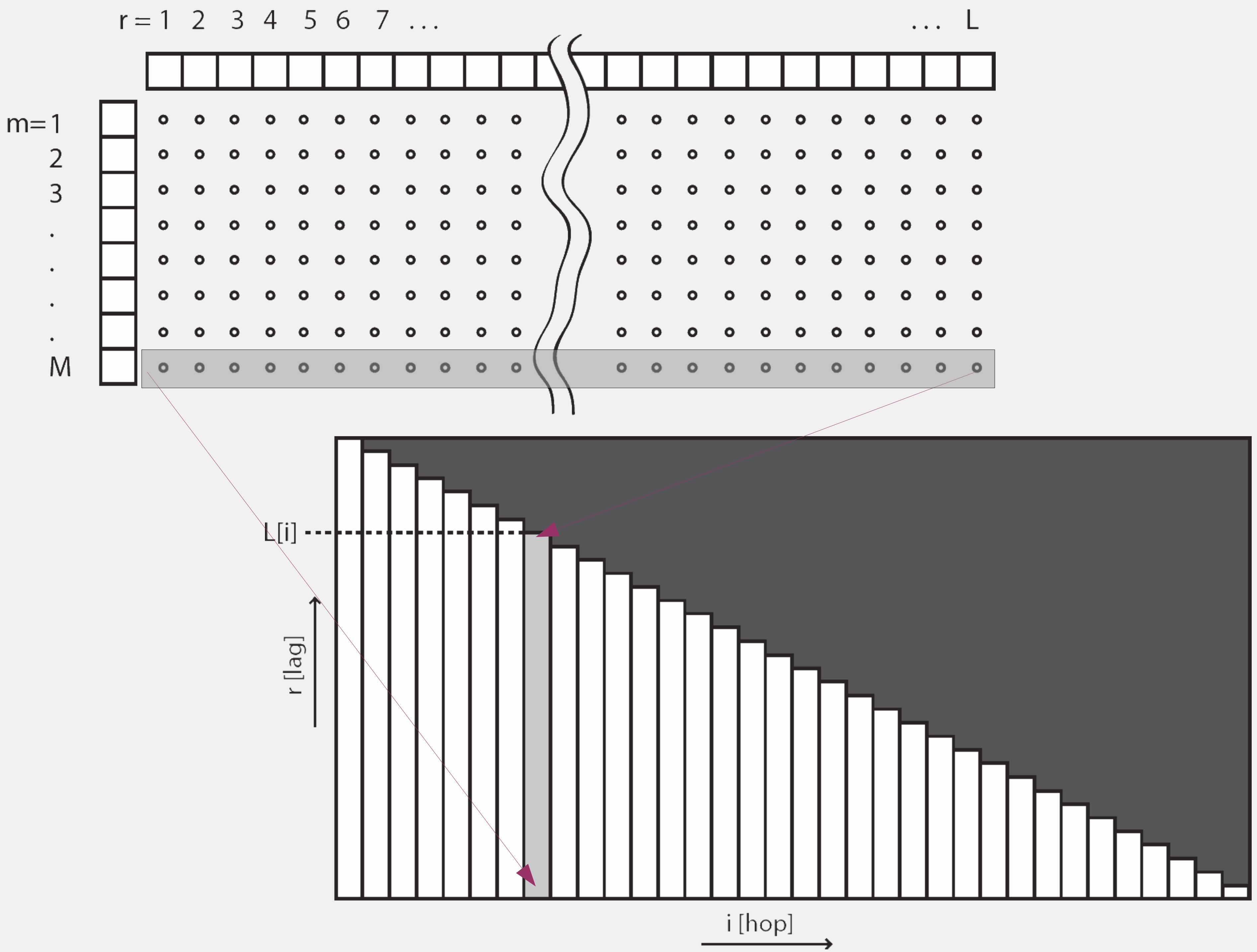
$$\begin{tikzcd}
D_i[m-1, r-1]+d_{m,r} \arrow[dr] & D_i[m-1,r]+e \arrow[d] \\
D_i[m, r-1]+e \arrow[r] & \mathbf{D_i[m,r]}
\end{tikzcd}$$

$$D_i[m,r] = \min \begin{cases} D_i[m-1,r] + e & \text{for} \quad m \geq 1 \\ D_i[m,r-1] + e & \text{for} \quad r \geq 1 \\ D_i[m-1,r-1] + d_{m,r} & \text{for} \quad else \end{cases}$$

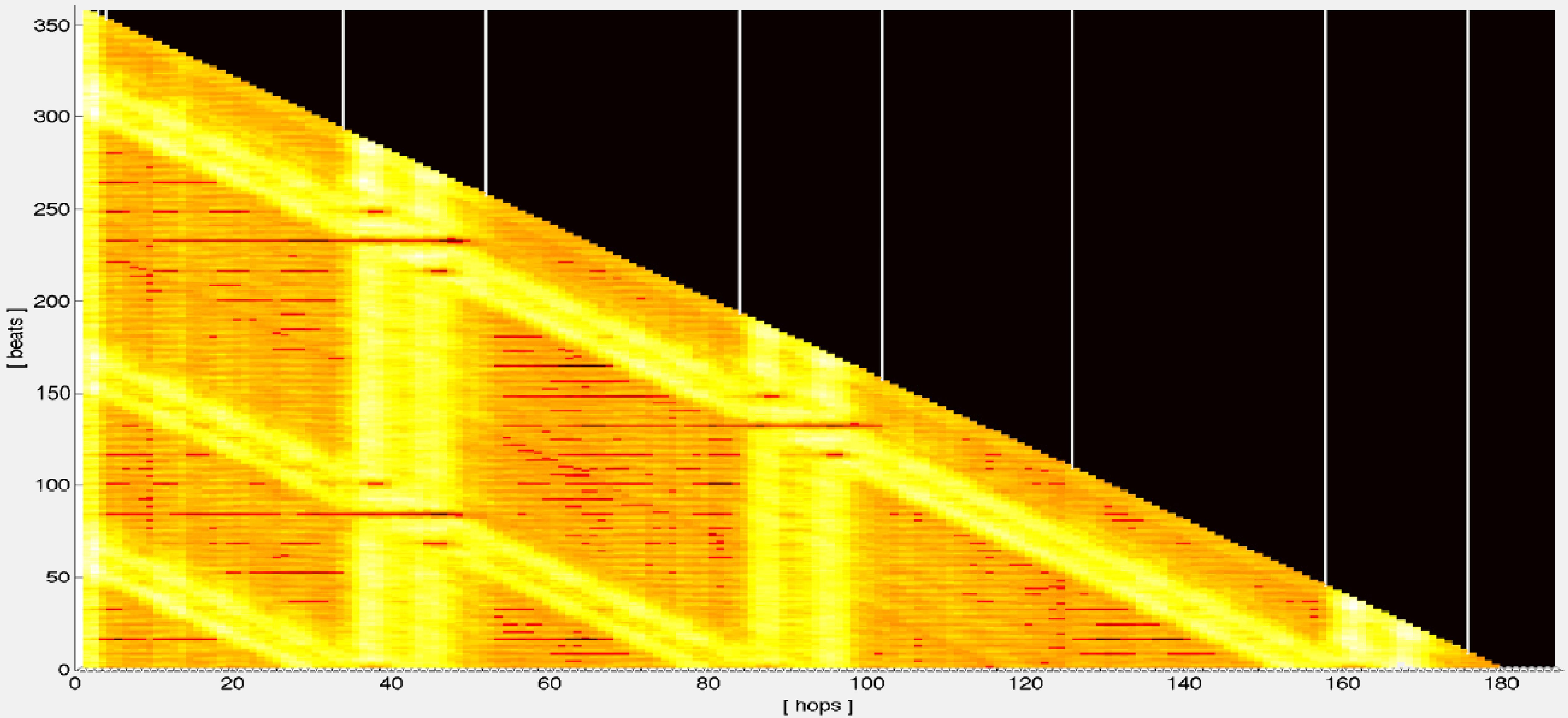
| | *2* | *7* | ***1*** | ***8*** | ***4*** | *5* | *2* | *6* | ***1*** | ***3*** | ***4*** | ***8*** | ***4*** | *9* | *3* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 1 | 6 | 0 | 7 | 3 | 4 | 1 | 5 | 0 | 2 | 3 | 7 | 3 | 8 | 2 |
| **3** | 2.1 | 5 | 2.1 | 5 | 4.1 | 5 | 2.1 | 4 | 2.1 | 0 | 1.1 | 6.2 | 4.1 | 9 | 2.1 |
| **8** | 8.2 | 3.1 | 9.2 | 2.1 | 6.2 | 7.1 | 8.2 | 4.1 | 9.2 | 5.1 | 4 | 1.1 | 5.2 | 5.1 | 7.2 |
| **4** | 7.1 | 6.2 | 6.1 | 6.2 | **2.1** | 3.2 | 5.3 | 6.2 | 7.1 | 6.2 | 4.1 | 5.2 | **1.1** | 6.2 | 6.1 |

## matching functions

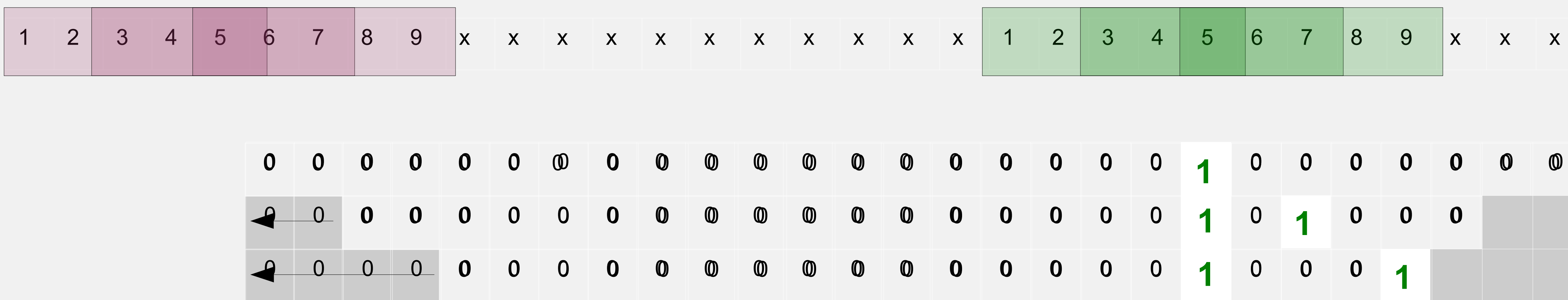
*D[i]: M=8*

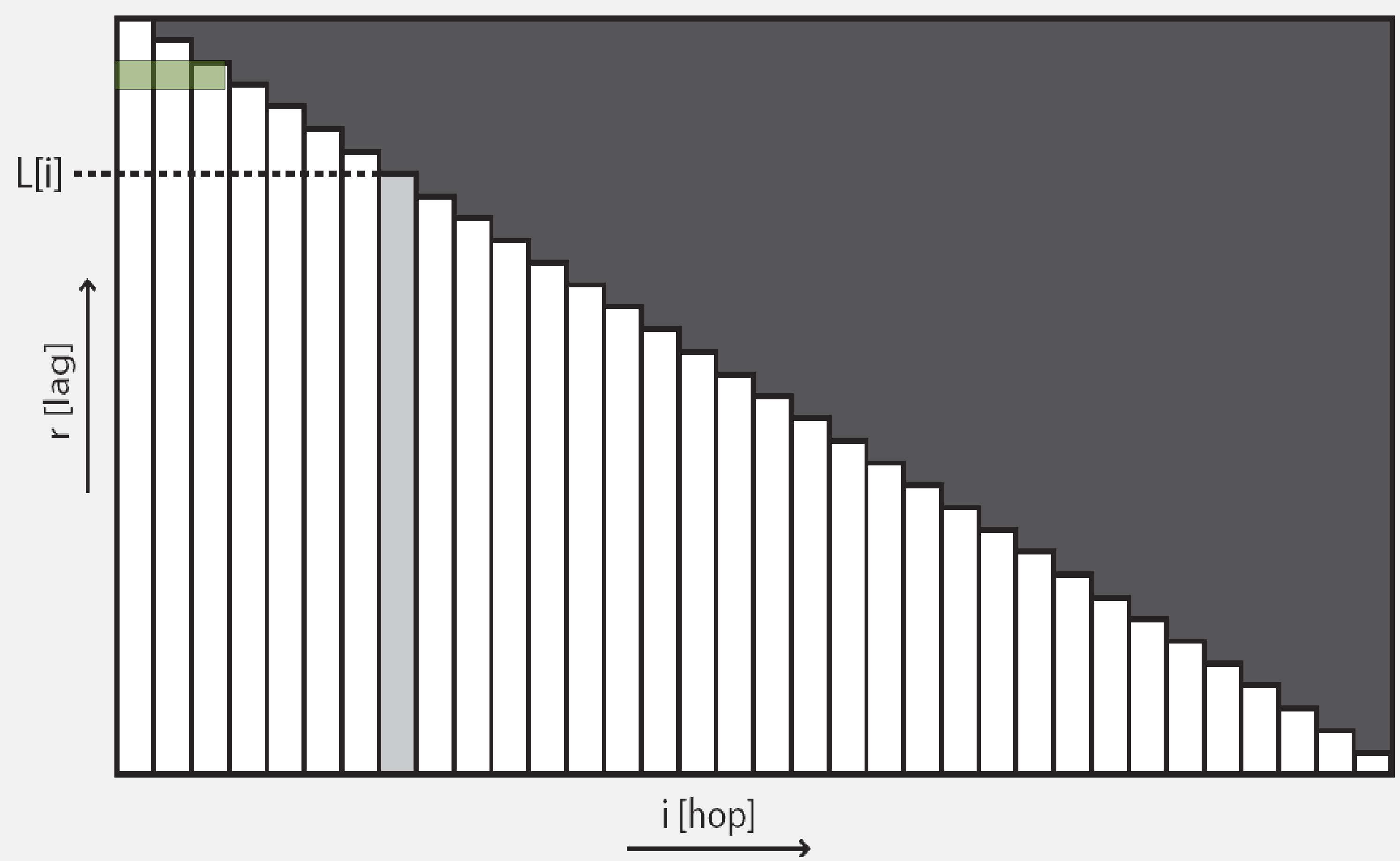


- matching matrix M[i,r]
  - horizontal lines → *approach 1*
  - vertical blocks → *approach 2*

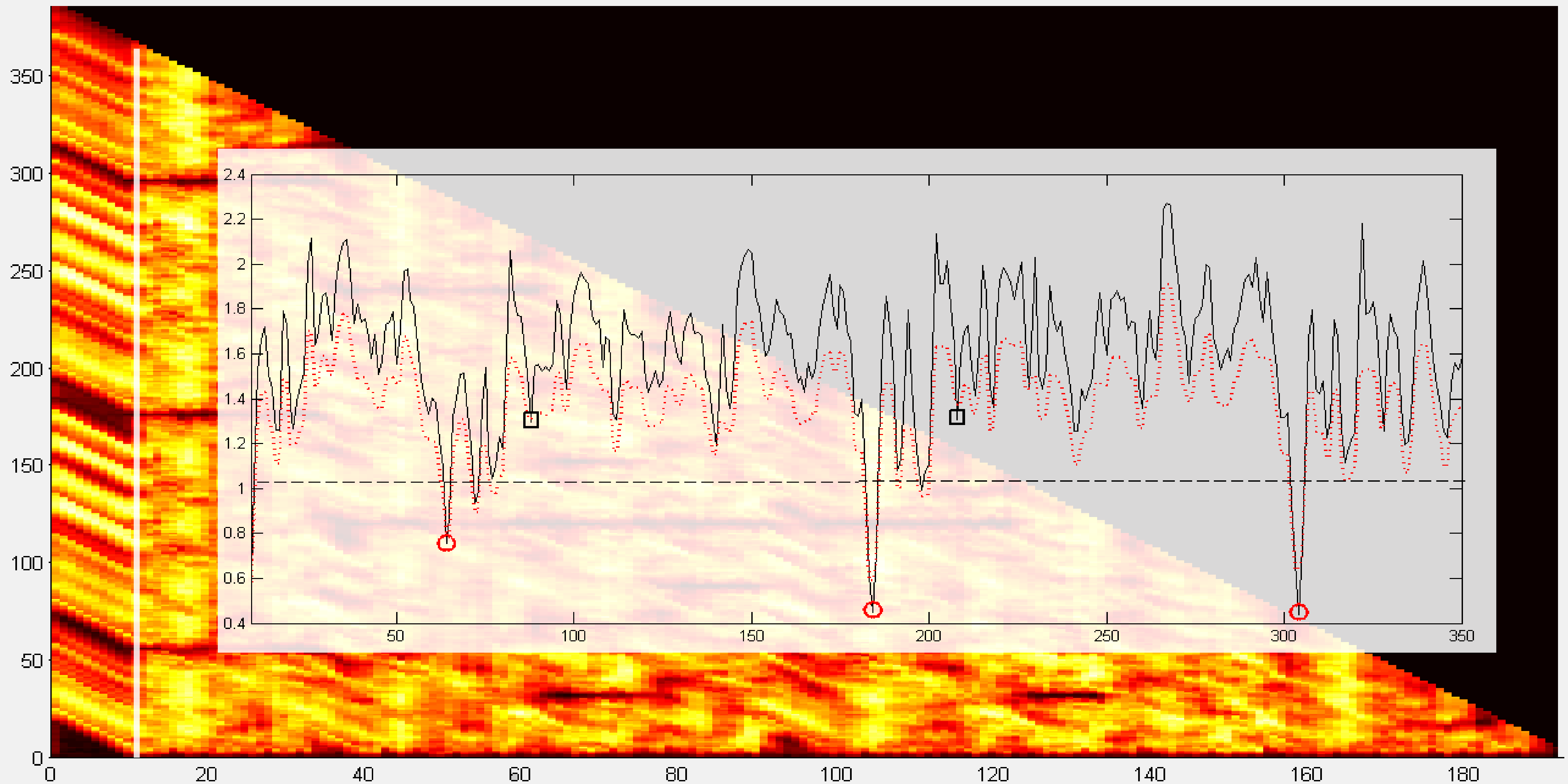


## horizontal lines

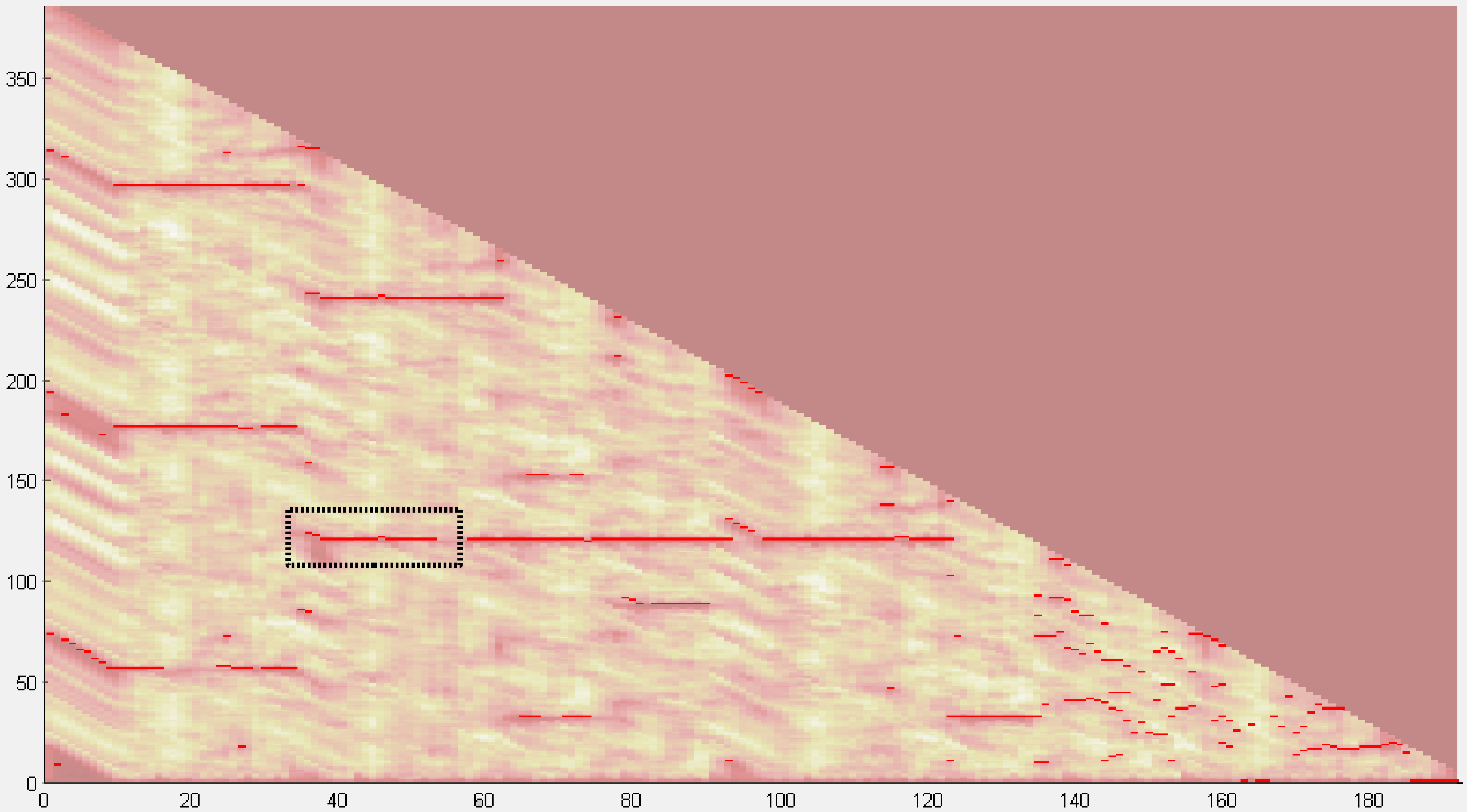


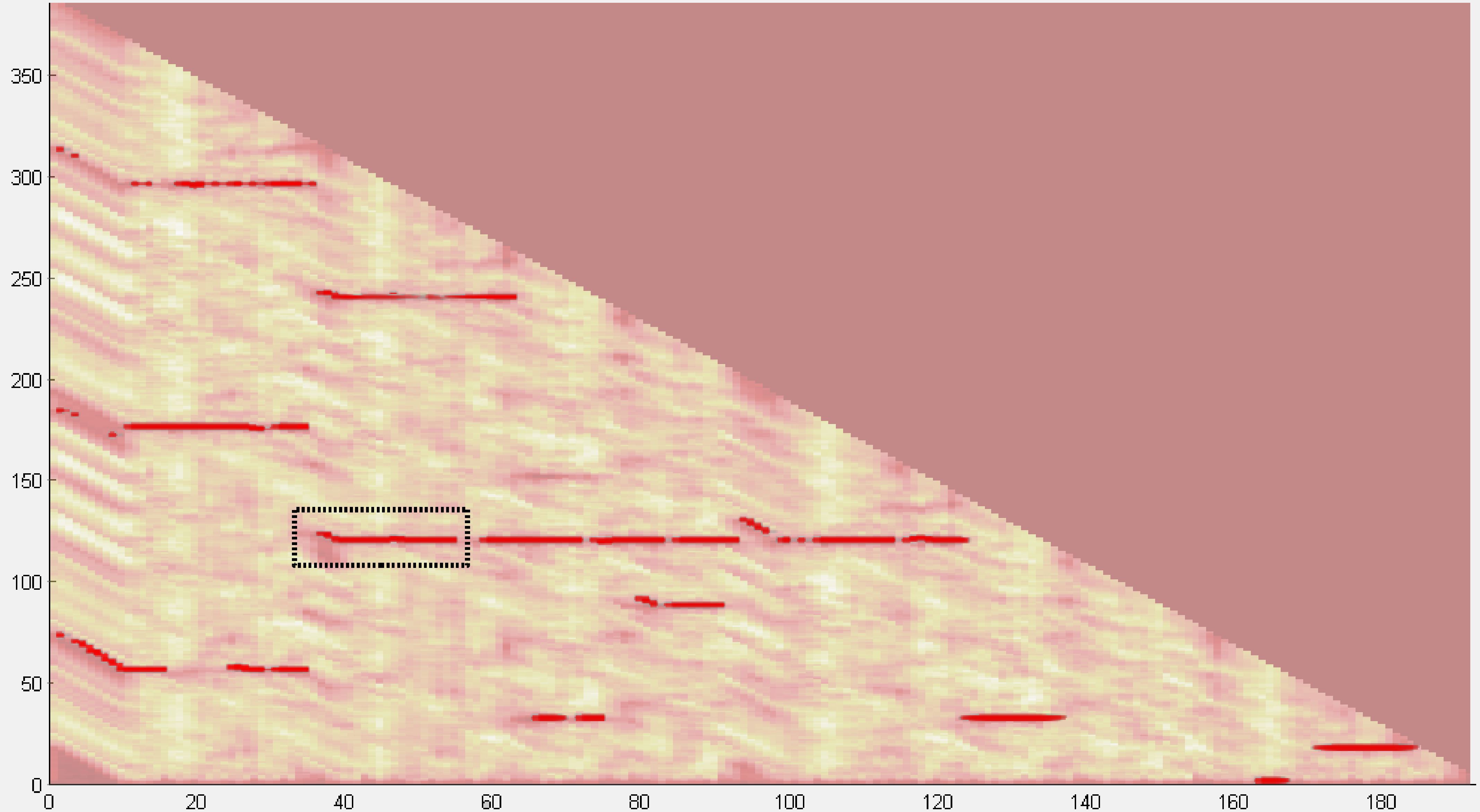
- line detection → find (almost exact) repetitions
  - detection of minima inside matching function → binary matrix
    - “1” at valley positions | “0” no valley



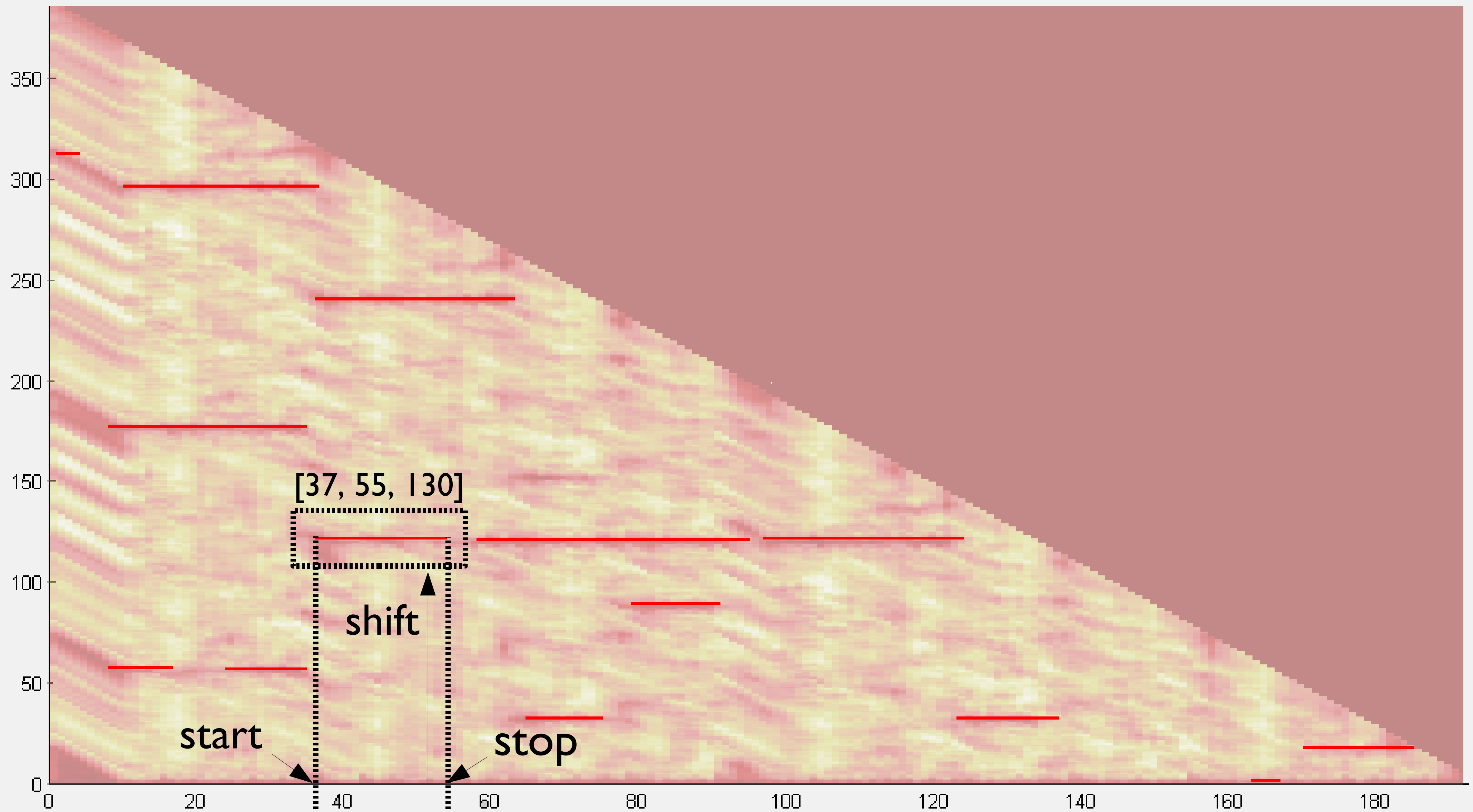
- detection matrix
  - all detected valleys



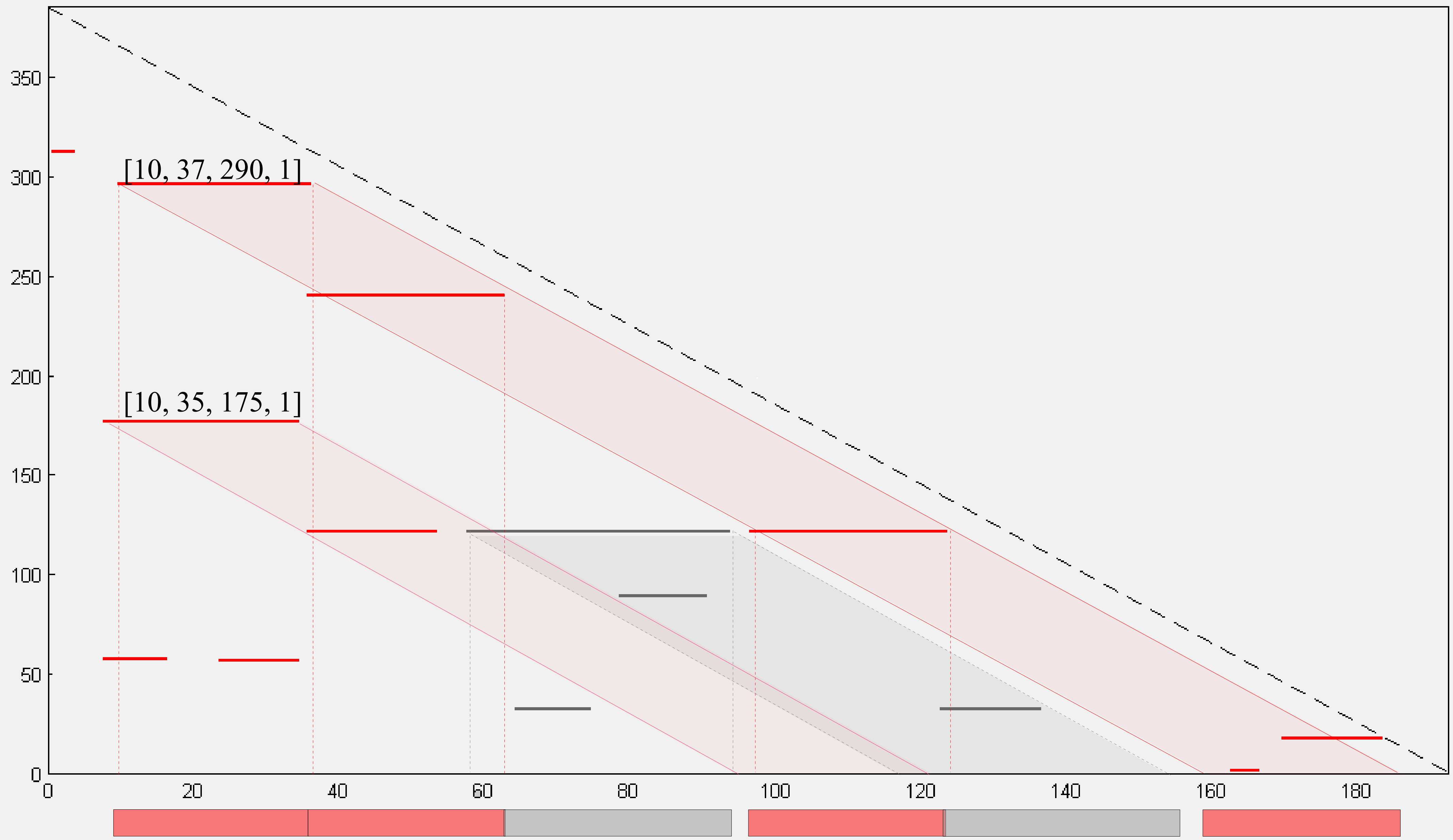
- matrix „cleaning“
  - delete „too short“ segments
  - apply gaussian blurring-kernel to matrix



- line detection
  - connect segments and get mean row-index
  - create segment vecotrs: [start, stop, shift]



- segment extraction
  - extract segments
  - merge segments basd on overlap/position [start, stop, shift, seg]



- extracted segments → information stored in vectors
  - [start, stop, shift, seg]
    - overlapping segments → “merge” segments automatically (if belonging to same song-segment)
    - compare more “important” (more detections) segments to others
      - merge, if overlapping is big
      - adapt segment number to number of „important segment“

| | | |
|---|---|---|
| [11,58,1;187,234,1] | [11,58,1;187,234,1] | [11,58,1,1;187,234,1,1] |
| [11,78,2;308,375,2] | [11,78,2;308,375,**2**] | [11,78,2,1;308,375,2,2] |
| [61,148,3;182,269,3] | [61,148,3;182,269,3] | [61,148,3,3;182,269,3,1] |
| [71,128,4;312,369,4] | [71,128,4;312,369,4] | [71,128,4,3;312,369,4,2] |
| [121,158,5;154,191,5] | [121,191,5] | [121,191,5,5] |
| [151,188,6;240,277,6] | [151,188,6;240,277,6] | [151,188,6,5;240,277,6,1] |
| [161,248,7;283,370,7] | [161,248,7;283,370,7] | [161,248,7,1;283,370,7,2] |
| [181,208,8;357,384,8] | [181,208,8;357,384,8] | [181,208,8,1;357,384,8,2] |
| [191,218,9;241,268,9] | [191,218,9;241,268,9] | [191,218,9,1;241,268,9,1] |
| [211,238,10;349,376,10] | [211,238,10;349,376,10] | [211,238,10,1;349,376,10,2] |
| [241,268,11;274,301,11] | [241,268,11;274,301,11] | [241,268,11,1;274,301,11,2] |
| [241,318,12;317,394,12] | [241,394,12] | [241,394,12,2] |
| [261,288,13;308,335,13] | [261,288,13;308,335,13] | [261,288,13,1;308,335,13,2] |
| [271,318,14;312,359,14] | [271,359,**14**] | [271,359,14,**2**] |
| [291,318,15;338,365,15] | [291,318,15;338,365,15] | [291,318,15,2;338,365,15,2] |
| [311,338,16;363,390,16] | [311,338,16;363,390,16] | [311,338,16,2;363,390,16,2] |
| [311,358,17;343,390,17] | [311,390,17] | [311,390,17,2] |
| [331,358,18;354,381,18] | [331,381,18] | [331,381,18,2] |
| [341,368,19;359,386,19] | [341,386,19] | [341,386,19,2] |

→

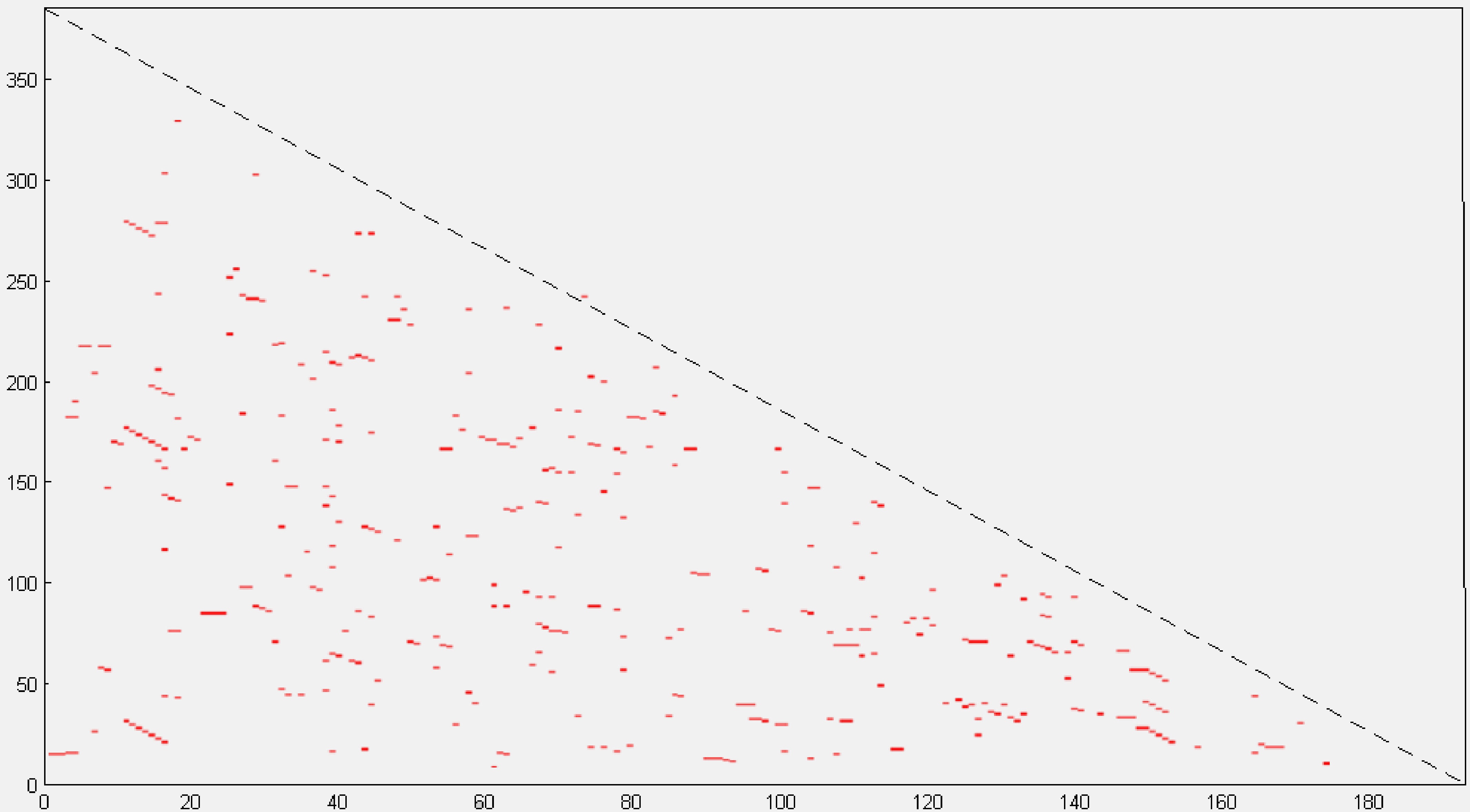
**detected segments**

| | | |
|---|---|---|
| 0.0001 | 33.6456 | 1.0000 |
| 33.6457 | 47.5776 | 5.0000 |
| 47.5777 | 64.9461 | 1.0000 |
| 64.9462 | 82.3146 | 3.0000 |
| 82.3147 | 110.8752 | 5.0000 |
| 110.8753 | 124.1106 | 10.0000 |
| 124.1107 | 151.3244 | 4.0000 |
| 151.3244 | 156.7114 | 10.0000 |
| 156.7115 | 198.8324 | 1.0000 |
| 198.8325 | 206.0770 | 0 |

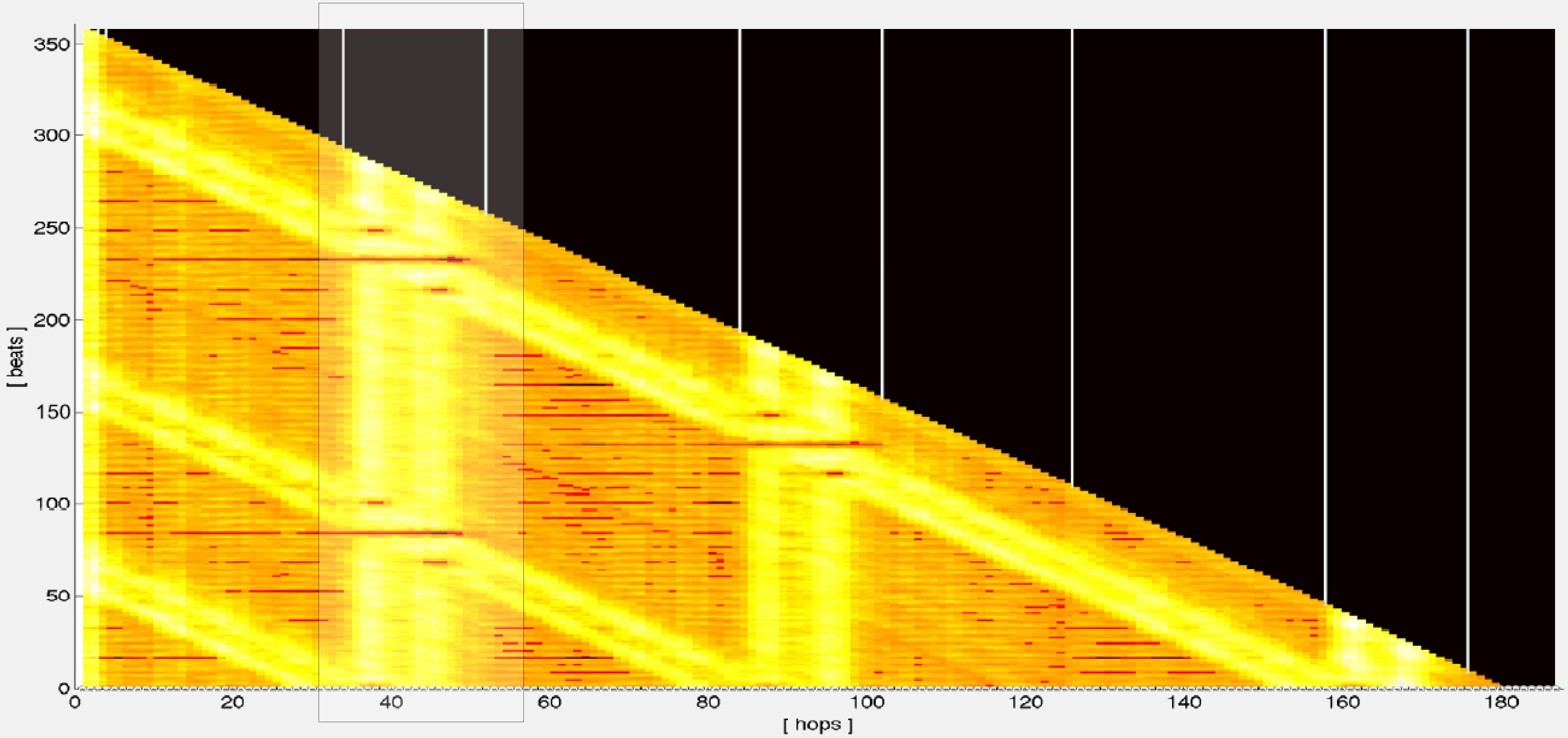
**real segments**

| | | |
|---|---|---|
| 0.0000000 | 6.1448290 | Intro |
| 6.1448290 | 32.4530380 | Verse |
| 32.4530380 | 50.5065530 | Bridge |
| 50.5065530 | 68.4671880 | Refrain |
| 68.4671880 | 93.6376190 | Verse |
| 93.6376190 | 111.0641950 | Bridge |
| 111.0641950 | 128.9319500 | Refrain |
| 128.9319500 | 153.9514510 | Verse |
| 153.9514510 | 173.7116320 | Refrain |
| 173.7116320 | 193.3557140 | Refrain |
| 193.3557140 | 208.3451574 | Refrain |

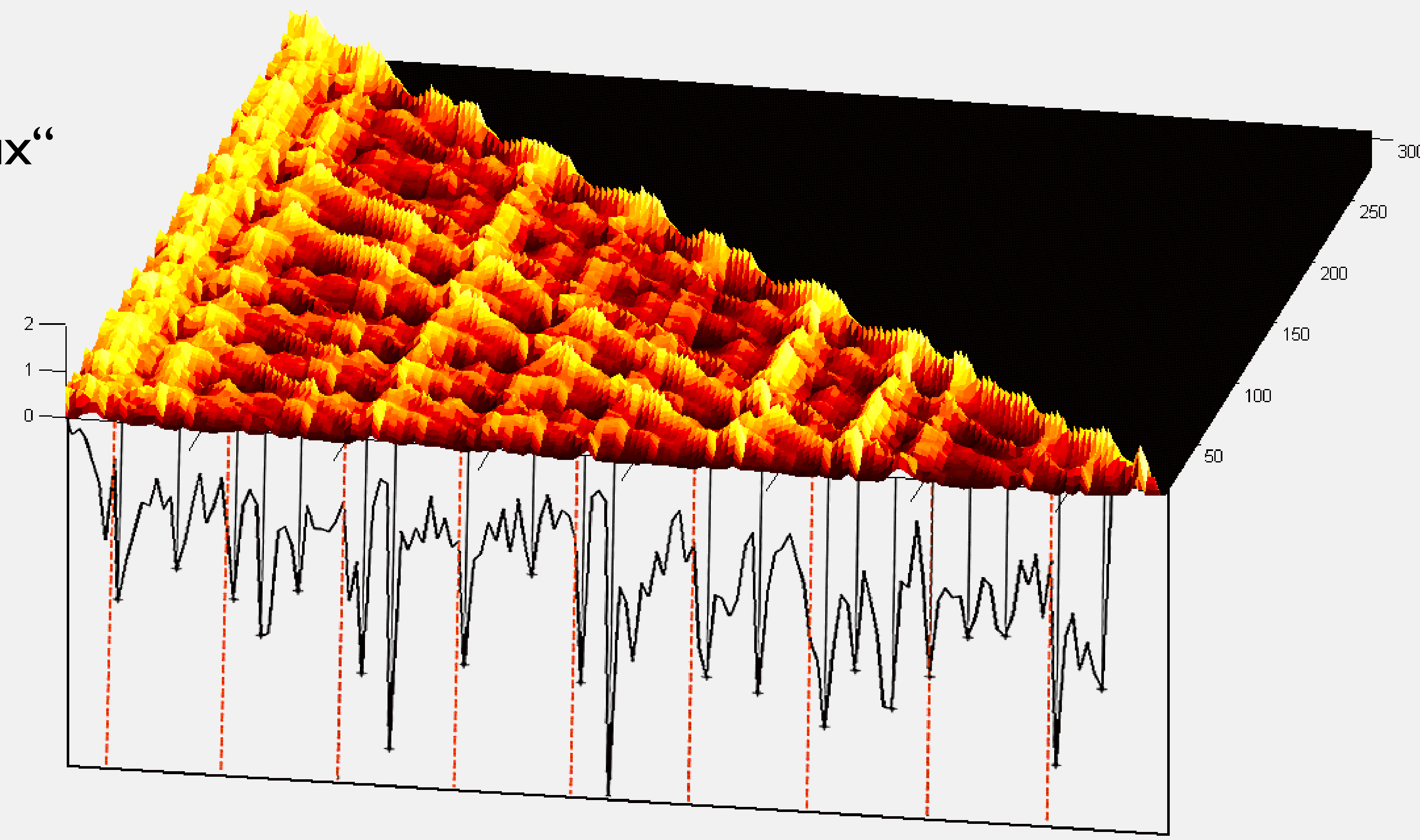
- Bad... :(



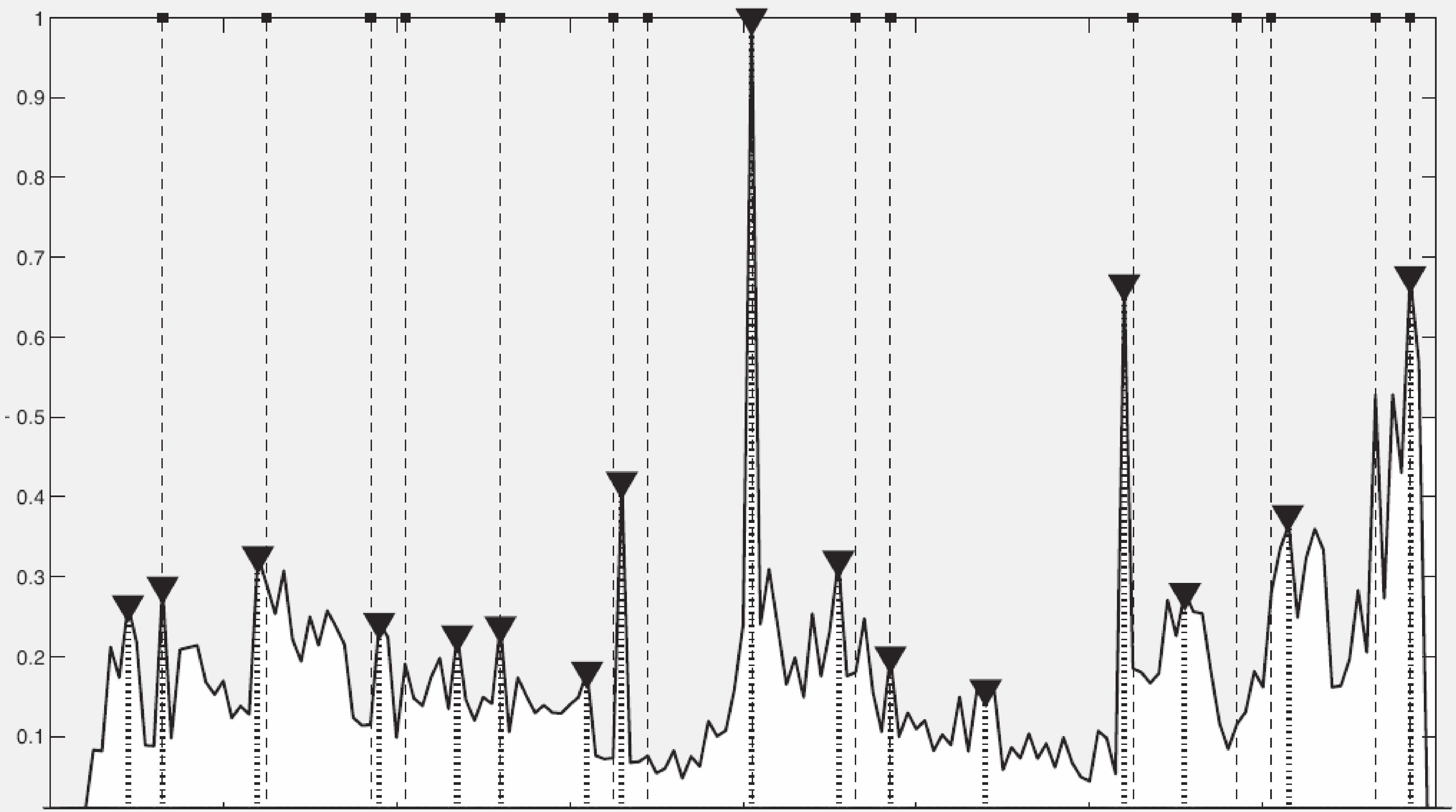
- block detection → no valley detection, no binary Matrix
  - global similarities
  - transitions between highly and less similar patterns

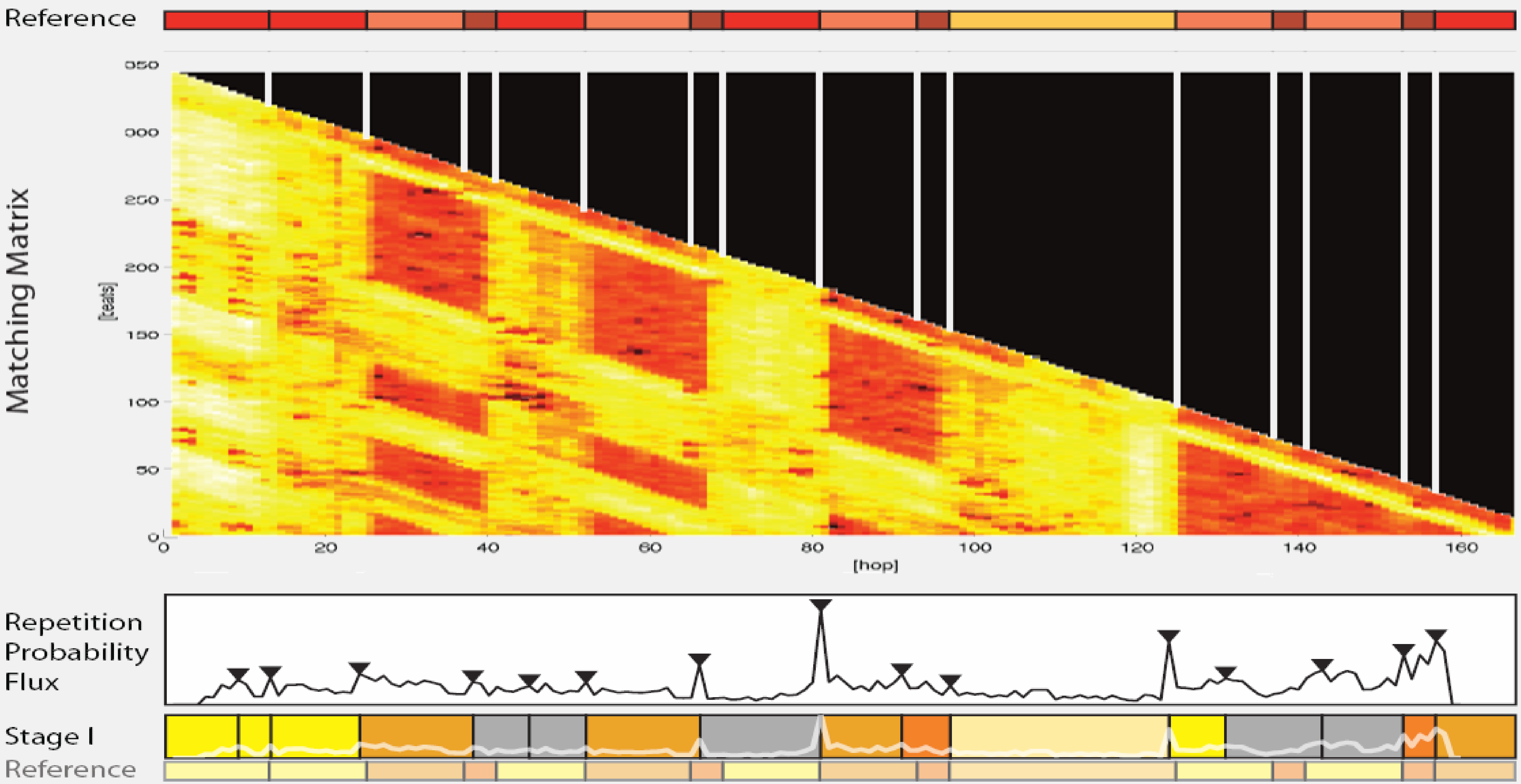


- find transitions between segments
  - squared difference of columns
    - $d[n] = |x[n] - x[n+1]|^2$
  - only use values larger than mean of column
    - $\hat{d}[i,r] = \begin{cases} d[i,r] & \text{if} \quad d[i,r] > \frac{1}{L}\sum_{r=1}^{L[i]} d[i,r] \\ 0 & else \end{cases}$
  - sum up to the „repetitive flux“
    - $\phi[i] = \sum_{r=1}^{L[i]} \hat{d}[i,r]$

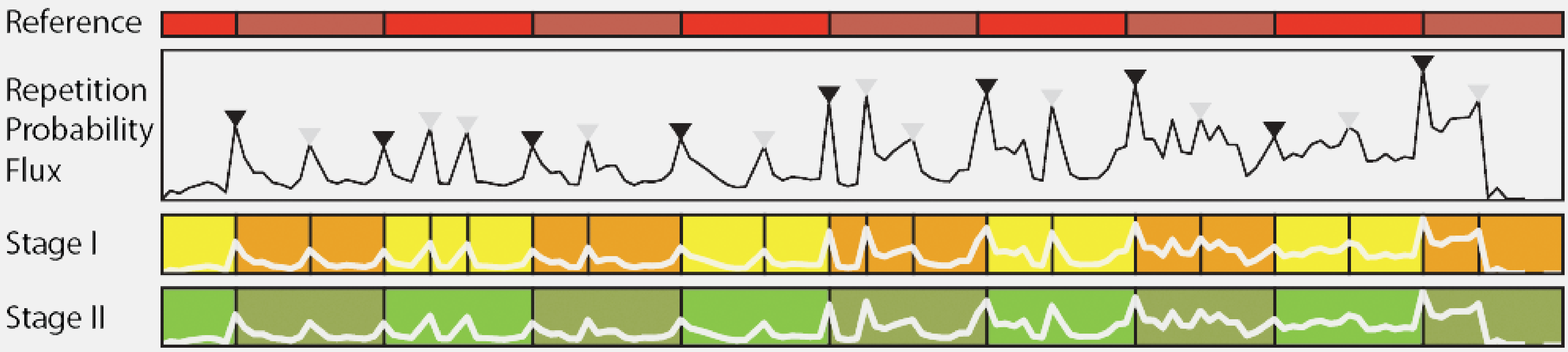


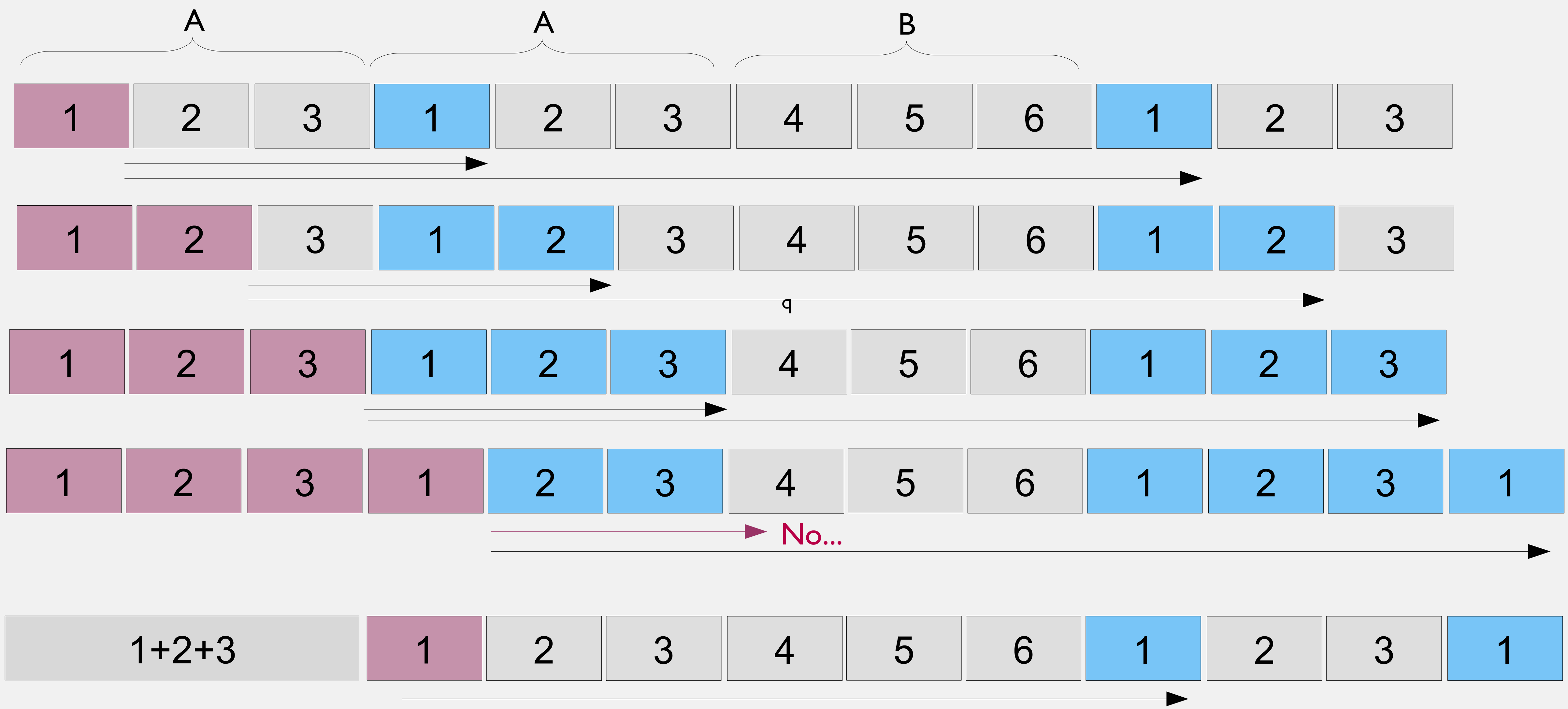
- peak picking
  - median window → sliding threshold
  - minimum distance of 8 beats




Reference
Matching Matrix
[beats]
0
50
100
150
200
250
300
350
0
20
40
60
80
100
120
140
160
[hop]
Repetition
Probability
Flux
Stage I
Reference

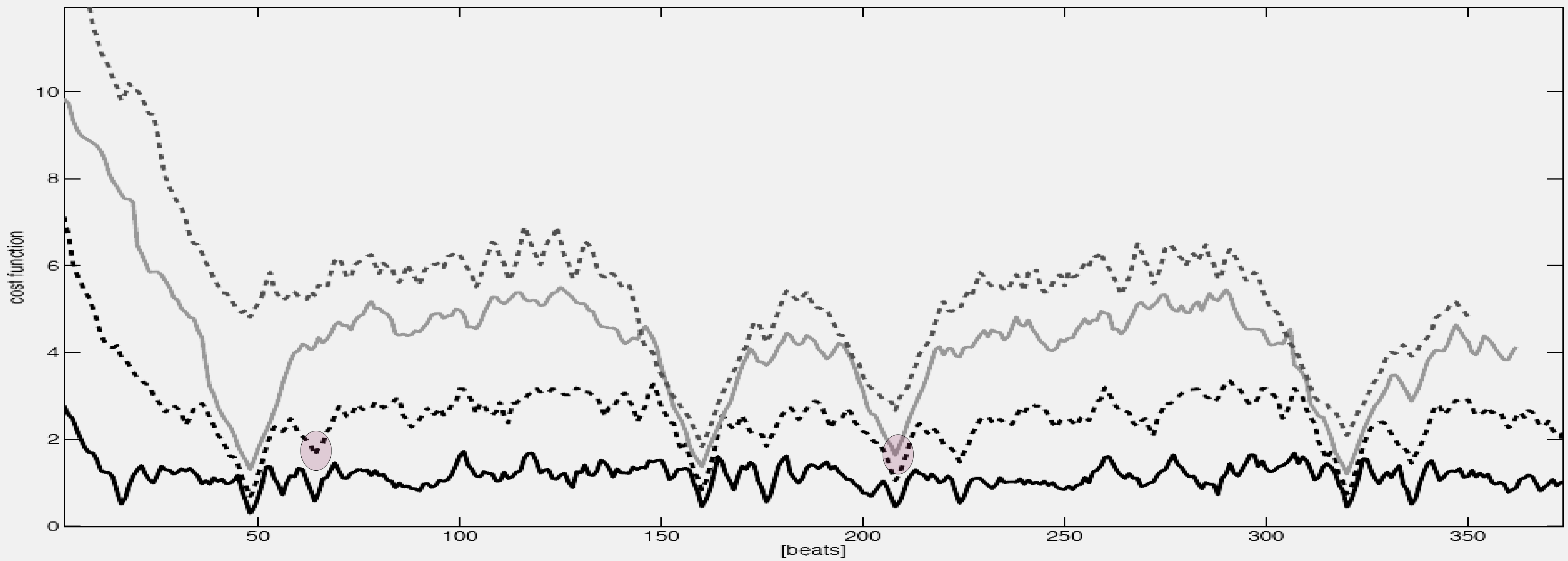
- feature sequence → beat averaged chroma vectors
  - new info (not directly used)
  - spectral and timbral information
  - beat → re-alignment possible (border correction)

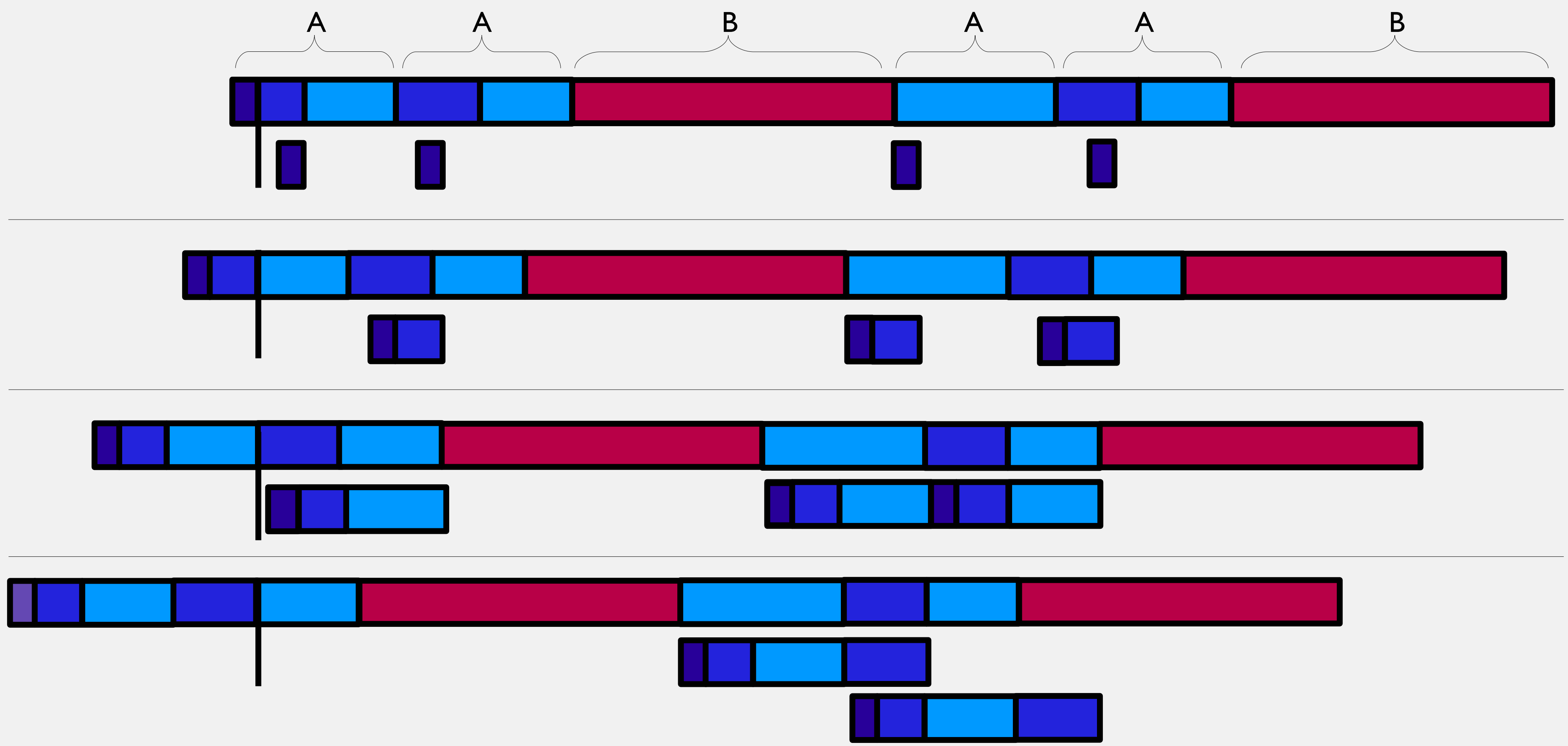




A
A
B
1 2 3 1 2 3 4 5 6 1 2 3
1 2 3 1 2 3 4 5 6 1 2 3
q
1 2 3 1 2 3 4 5 6 1 2 3
1 2 3 1 2 3 4 5 6 1 2 3 1
No...
1+2+3 1 2 3 4 5 6 1 2 3 1

- valley detection
- max. merging length
- hard/soft borders



- 32 songs
  - 16 pop songs
    - e.g. Alanis Morisette, Beastie Boys, Britney Spears, Eminem, …
  - 16 Beatles songs
    - „With the Beatles“ (full album)
    - other songs
- reference segmentations by members of the MPEG-7 working group
    - used by other authors (e.g. Levy and Sandler)

- ground truth problem
  - Levy et al.


  - Paualus et al.


  - Levy et al.


# evaluation

./testing corpus > ground truth > **results**

- evaluation measures
  - precision

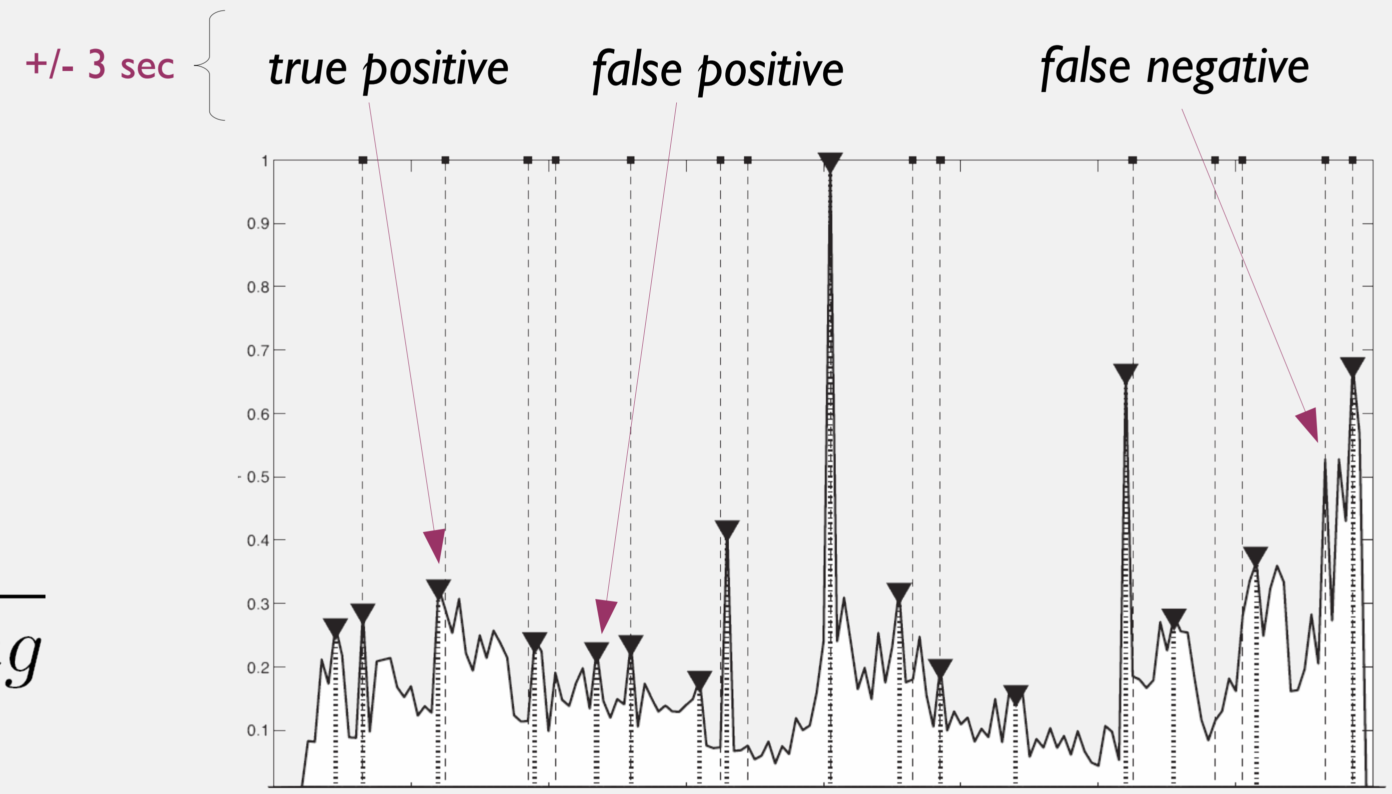
$$p = \frac{truePos}{truePos + falsePos}$$

  - recall

$$r = \frac{truePos}{truePos + falseNeg}$$

  - f-measure

$$f = \frac{2pr}{p + r}$$




| Corpus | precision $p$ | recall $r$ | $f$ |
|---|---|---|---|
| Beatles | 0.50 | 0.83 | 0.61 |
| Recent | 0.70 | 0.73 | 0.70 |
| **Overall** | **0.62** | **0.77** | **0.65** |

## thank you!

THX!

Q?