MIRtoolbox 1.1
User’s Manual

Olivier Lartillot
Finnish Center of Excellence in Interdisciplinary Music Research
University of Jyväskylä, Finland
May, 4th, 2008
# TABLE OF CONTENTS

## 1. General Principles

- MIRtoolbox License
- MIRtoolbox Website
- MIRtoolbox History
- MIRtoolbox Objectives
- Related Works
- MIRtoolbox Features
- MIRtoolbox Reliances
- Installation
- Online documentation
- MIRtoolbox Interface

## 2. Basic Operators

- miraudio
- mirsave
- mirframe
- mirfilterbank
- mirspectrum
- mircepstrum
- mirautocor
- *
- mirflux
- mirsum
- mirsegment

2.12. mirplay
3. Feature Extractors

3.1. Dynamics

mirrms
mirlowenergy

3.2. Rhythm

mirfluctuation
mironsets
mirtempo

3.3. Timbre

mirattacktime
mirattackslope
mirzerocross
mirrolloff
mirbrightness
mirmfcc
mirroughness
mirregularity

3.4. Pitch

mirpitch
mirinharmonicity

3.5. Tonality

mirchromagram
mirkeystrength
mirkey
mirmode
mirkeysom
4. Post-Processing

4.1. Structure and form

4.2. Statistics

4.3. Classification

4.4. Exportation

5. Advanced use of MIRtoolbox
5.2. Memory management

Annex A
1. GENERAL PRINCIPLES

**MIRtoolbox License**
The Toolbox is free software; you can redistribute it and/or modify it under the terms of version 2 of **GNU General Public License** as published by the Free Software Foundation.

**MIRtoolbox Website**
The URL of MIRtoolbox website is [www.jyu.fi/music/coe/materials/mirtoolbox](http://www.jyu.fi/music/coe/materials/mirtoolbox)

A discussion list is also available:

- To subscribe, send an empty mail with ‘Subscribe’ as subject to mirtoolbox-request@freelists.org
- The archive is available [here](http://www.jyu.fi/music/coe/materials/mirtoolbox)

**MIRtoolbox History**

**ABOUT THE AUTHORS**
Olivier Lartillot, Petri Toiviainen and Tuomas Eerola are employed at the Department of Music of the University of Jyväskylä, Finland. They are members of the **Finnish Center of Excellence in Interdisciplinary Music Research**.

The development of the toolbox has benefitted from productive collaborations with the other partners of the project, in particular Marco Fabiani, Jose Fornari, students of our center of excellence (Vinoo Alluri, Rafael Ferrer, Marc Thompson, ...), external collaborators (Cyril Laurier, ...) and participants of the SMC Summer School 2007 and of ISSSM 2007.

**TUNING THE BRAIN FOR MUSIC**
MIRtoolbox has been developed within the context of a European Project called “Tuning the Brain for Music”, funded by the NEST (New and Emerging Science and Technology) program of the European Commission. The project, coordinated by Mari Tervaniemi from the Cognitive Brain Research Unit of the Department of Helsinki, is dedicated to the study of music and emotion, with collaboration between neurosciences, cognitive psychology and computer science. One particular question, studied in collaboration between the Music Cognition Team of the University of Jyväskylä and the Music Acoustics Group of the KTH in Stockholm, is related to the investigation of the relation between musical features and music-induced emotion.
In particular, we would like to know which musical parameters can be related to the induction
of particular emotion when playing or listening to music. For that purpose, we needed to ex-
tract a large set of musical features from large audio data-bases, in order to perform in a second
step a statistical mapping between the diverse musical parameters and musical materials with
listeners' emotional ratings. This requires in particular a management of the interdependencies
between the diverse features – in order to avoid having to recompute the same operations
again and again – and also a control of the memory costs while analyzing the databases.

Music, Mind, Technology
The Music Cognition Team has recently introduced a new master degree, called Music Mind
Technology (MMT). The Music Information Retrieval course, taught by Petri Toiviainen and my-
self, offers an overview of computer-based research for music analysis and in particular musical
feature extraction. For the hands-on sessions, we wanted the student to be able to try by them-
selves the different computational approaches using Matlab. As many of them did not have
much background in this programming environment, we decided to design a computational
environment for musical feature extraction aimed at both expert and non-expert of Matlab.

MIRtoolbox Objectives
Due to the context of development of this toolbox, we elaborated the following specifications:

General Framework
MIRtoolbox proposes a large set of musical feature extractors.

Modular Framework
MIRtoolbox is based on a set of building blocks that can be parametrized, reused, reordered,
etc.

Simple and Adaptive Syntax
Users can focus on the general design, MIRtoolbox takes care of the underlying laborious tasks.

Free Software, Open Source
The idea is to propose to capitalize the expertise of the research community, and to offer it
back to the community and the rest of us.
Related Works

Of course, **MIRtoolbox** is not the first and only musical feature extraction tool available “in the market”. Actually, there already exist very good frameworks. A comparative table (maybe not up to date ...) is shown in the following table:

<table>
<thead>
<tr>
<th>Related Solutions</th>
<th>Features</th>
<th>Interface</th>
<th>Output</th>
<th>Batch process.</th>
<th>Control of dependnc.</th>
<th>Distributed computing</th>
<th>Memory managmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marsyas (Tzanetakis)</td>
<td>dozen low + beats</td>
<td>scripting language</td>
<td>export</td>
<td>yes</td>
<td>manual</td>
<td>yes</td>
<td>real time</td>
</tr>
<tr>
<td>jAudio (McKay et al.)</td>
<td>~20 low + beats</td>
<td>GUI</td>
<td>export</td>
<td>yes</td>
<td>auto</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>CLAM</td>
<td>~12 spectr. + tonal visual program.</td>
<td>pre-visual. &amp; export</td>
<td>yes</td>
<td>manual</td>
<td>?</td>
<td>real time</td>
<td></td>
</tr>
<tr>
<td>ChucK (Wang, Cook)</td>
<td>dozen low-level program. language</td>
<td>numerical</td>
<td>yes</td>
<td>manual</td>
<td>yes</td>
<td>yes</td>
<td>real time</td>
</tr>
<tr>
<td>M2K (Downie et al.)</td>
<td>dozen low-level D2K visual program.</td>
<td>graphic &amp; export</td>
<td>yes</td>
<td>manual</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Pysound (Cabrera)</td>
<td>~30 low &amp; high</td>
<td>GUI</td>
<td>graphic &amp; export</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPEM toolbox</td>
<td>17 low &amp; high</td>
<td>Matlab functions</td>
<td>graphic &amp; numerical</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MA toolbox (Pampalk)</td>
<td>8 Matlab functions</td>
<td>numerical values</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MIRtoolbox</strong></td>
<td>~40 low &amp; high adaptive syntax</td>
<td>graphic &amp; export</td>
<td>yes</td>
<td>manual</td>
<td>yes</td>
<td>future work using DC tb</td>
<td></td>
</tr>
</tbody>
</table>

*Comparative list of audio feature extraction frameworks*
Synthetic overview of the features available in MIRtoolbox 1.1
**MIRtoolbox Features**

MIRtoolbox includes around 40 audio and music features extractors and statistical descriptors. A brief overview of most of the features can be seen in the previous figure.

**MIRtoolbox Reliances**

**Commercial Products to be Purchased**

*MIRtoolbox* requires the *Matlab* environment, version 7, and does not work very well with previous versions of *Matlab*. This is due in particular to the fact *MIRtoolbox* relies on multi-dimensional arrays and multiple outputs, which seem to be features introduced by version 7.

**WARNING:** *MIRtoolbox* has not been tested yet on the newest *Matlab* version 7.6. Some problem might arise with this new version, as it features a major revision of their Object-Oriented Programming environment, which is a key feature used in *MIRtoolbox*.

*MIRtoolbox* also requires that the *Signal Processing Toolbox*, one of the optional sub-packages of *Matlab*, be properly installed. But actually, a certain number of operators can adapt to the absence of this toolbox, and can produce more or less reliable results. But for serious use of *MIRtoolbox*, we strongly recommend a proper installation of the *Signal Processing Toolbox*.

**Free Softwares Included in the MIRtoolbox Distribution**

*MIRtoolbox* includes in its distribution several other freely available toolboxes, that are used for specific computations.

- The *Auditory Toolbox*, by Malcolm Slaney (1998), is used for Mel-band spectrum and MFCC computations, and Gammatone filterbank decomposition.

- The *Netlab* toolbox, by Ian Nabney (2002), where the routines for Gaussian Mixture Modeling (GMM) is used for classification (*mirclassify*).

- Finally, the SOM toolbox, by Esa Alhoniemi and colleagues (Vesanto, 1999), where only a routine for clustering based on $k$-means method is used, in the *mircluster* function.

**Code Integrated as part of GPL Project.**

*MIRtoolbox* license is based on GPL 2.0. As such, it can integrate codes from other GPL 2.0 projects, as long as their origins are explicitly stated.
• We have integrated so far codes from the Music Analysis Toolbox by Elias Pampalk (2004), related to the computation of Terhardt outer ear modeling, Bark band decomposition and masking effects.

Installation
To install MIRtoolbox in your Matlab environment, copy all the toolboxes folders (or only those that are not installed yet in your computer) into your Matlab “toolbox” folder. Then add each folder in your Matlab path.

UPDATE
If you replace an older version of MIRtoolbox with a new one, please update your Matlab path using the following command:

```
rehash toolboxcache
```

Update also the class structure of the toolbox, either by restarting Matlab, or by typing the following command:

```
clear classes
```

Online documentation
To get an overview of the functions available in the toolbox, type:

```
help mirtoolbox
```

A short documentation for each function is available using the same help command. For instance, type:

```
help miraudio
```

DEMOS
Examples of use of the toolbox are shown in the MIRToolboxDemos folder:

• mirdemo
• demo1basics
• demo2timbre
• demo3segmentation
• demo4tempo
• demo5export
• demo6curves
• demo7tonality
• demo8classification
**MIRtoolbox Interface**

**Basic Syntax**

All functions are preceded by the `mir` prefix in order to avoid conflicts with other Matlab functions. Each function is related to a particular data type: for instance, `miraudio` is related to the loading, transformation and display of audio waveform. An audio file, let’s say a WAV file of name `mysong.wav`, can be loaded simply by writing the command:

\[
\text{miraudio}(\text{mysong}, \text{wav})
\]

The extension of the file can be omitted:

\[
\text{miraudio}('\text{mysong}')
\]

Operations and options to be applied are indicated by particular keywords, expressed as arguments of the functions. For instance, the waveform can be centered using the ‘Center’ keyword:

\[
\text{miraudio}(\text{mysong}, '\text{Center}')
\]

which is equivalent to any of these parameters:

\[
\text{miraudio}(\text{mysong}, '\text{Center}', '\text{yes}')
\]

\[
\text{miraudio}(\text{mysong}, '\text{Center}', '\text{on}')
\]

\[
\text{miraudio}(\text{mysong}, '\text{Center}', 1)
\]

whereas the opposite set of parameters

\[
\text{miraudio}(\text{mysong}, '\text{Center}', '\text{no}')
\]

\[
\text{miraudio}(\text{mysong}, '\text{Center}', '\text{off}')
\]

\[
\text{miraudio}(\text{mysong}, '\text{Center}', 0)
\]

are not necessary in the case of the ‘Center’ options as it is toggle off by default in `miraudio`.

It should be noted also that keywords are not case-sensitive:

\[
\text{miraudio}(\text{mysong}, '\text{center}', '\text{YES}')
\]
Other options accept numerical particular parameters. For instance, an audio waveform can be resampled to any sampling rate, which is indicated by a value in Hertz (Hz.) indicated after the 'Sampling' keyword. For instance, to resample at 11025 Hz., we just write:

\[
\text{miraudio}('\text{mysong}', '\text{Sampling}', 11025)
\]

Finally the different options can be combined in one single command line:

\[
\text{miraudio}('\text{mysong}', '\text{Center}', '\text{Sampling}', 11025)
\]

**Batch Analysis**

Folder of files can be analyzed in exactly the same way. For that, the file name, which was initially the first argument of the functions, can be replaced by the 'Frame' keyword. For instance, a folder of audio files can be loaded like this:

\[
\text{miraudio}('\text{Folder}')
\]

Only audio files in the WAV and AU formats are taken into consideration, the other files are simply ignored:

```
Current Directory:

song1.wav
song2.wav
song3.au
non_audio.file

\text{miraudio}('\text{Folder}')
```

Automatic analysis of a batch of audio files using the 'Folder' keyword

**Output Format**

After entering one command, such as

\[
\text{miraudio}('\text{mysong}')
\]

the computation is carried out, and when it is completed, a text is written in the Command Window:

\text{ans is the Audio waveform related to file mysong.wav, of sampling rate 44100 Hz.}

Its content is displayed in Figure 1.
And a graphical representation of the result is displayed in a figure:

![Audio waveform](image)

*Display of a miraudio object.*

The display of the figures and the messages can be avoided, if necessary, by adding a semi-colon at the end of the command:

```
miraudio('mysong');
```

The actual output is stored in an object, hidden by default to the users, which contains all the information related to the data, such as the numerical values of the waveform amplitudes, the temporal dates of the bins, the sampling rate, the name of the file, etc. In this way we avoid the traditional interface in Matlab, not quite user-friendly in this respect, were results are directly displayed in the Command Window by a huge list of numbers.

**MULTIPLE FILE OUTPUT**

If we now analyze a folder of file:

```
miraudio('Folder')
```

the results related to each audio file is displayed in a different figure, and messages such as the following ones are displayed in the Command Window:
ans(1) is the Audio waveform related to file song1.wav, of sampling rate 44100 Hz.

Its content is displayed in Figure 1.

ans(2) is the Audio waveform related to file song2.wav, of sampling rate 22050 Hz.

Its content is displayed in Figure 2.

ans(3) is the Audio waveform related to file song3.au, of sampling rate 11025 Hz.

Its content is displayed in Figure 3.

and so on.

And the actual output is stored in one single object, that contains the information related to all the different audio files.

THREADING OF DATA FLOW

The result of one operation can be used for subsequent operations. For that purpose, it is better to store each result in a variable. For instance, the audio waveform(s) can be stored in one variable a:

\[ a = \text{miraudio}('mysong'); \]

Then the spectrum, for instance, related to the audio waveform can be computed by calling the function mirspectrum using simply the a variable as argument:

\[ s = \text{mirspectrum}(a) \]

In this way, all the information necessary for the computation of the spectrum can be retrieved from the hidden object, associated to the variable a, that contains the complex encapsulated data.

Alternatively, the spectrum can be directly computed from a given audio file by indicating the file name as argument of the mirspectrum function:

\[ s = \text{mirspectrum}('mysong') \]

SUCCESSIVE OPERATIONS ON ONE SAME DATA FORMAT

When some data has been computed on a given format, let’s say an audio waveform using the miraudio function:
\[ a = \text{miraudio}('mysong'); \]

it is possible to apply options related to that format in successive step. For instance, we can center the audio waveform in a second step:

\[ a = \text{miraudio}(a, 'Center'); \]

which could have been also written in one line:

\[ a = \text{miraudio}('mysong', 'Center'); \]

**N U M E R I C A L  D A T A  R E C U P E R A T I O N**

The numerical data encapsulated in the output objects can be recuperated if necessary. In particular, the main numerical data (such as the amplitudes of the audio waveform) are obtained using the `mirgetdata` command:

\[ \text{mirgetdata}(a) \]

the other related informations are obtained using the generic `get` method. For instance, the sampling rate of the waveform `a` is obtained using the command:

\[ \text{get}(a, 'Sampling') \]
2. BASIC OPERATORS

MIRtoolbox basic operators concern the management of audio waveforms (miraudio, mirsave), frame-based analysis (mirframe, mirflux), periodicity estimation (mirautocor, mirspectrum, mircepstrum), operations related more or less to auditory modeling (mirenvelope, mirfilterbank, mirauditory), peak picking (mirpeaks) and sonification of the results (mirplay).

miraudio

AUDIO WAVEFORM
As explained previously, this operator basically loads audio files, displays and performs operations on the waveform.

ACCEPTED INPUT FORMATS
- file name: The accepted file formats are WAV and AU formats, as the loading operations are based on the Matlab wavread and auread functions. Stereo signals are converted to mono.
- miraudio object: for further transformations.
- Matlab array: It is possible to import an audio waveform encoded into a Matlab array, by using the following syntax:

\[
\text{miraudio}(v, sr)
\]

where \(v\) is the array and \(sr\) is the sampling rate of the signal, in Hz. The default value for \(sr\) is 44100 Hz.

TRANSFORMATION OPTIONS
- miraudio(..., 'Center') centers the waveform.
- miraudio(..., 'Sampling', \(r\)) resamples at sampling rate \(r\) (in Hz). It uses the resample function from Signal Processing Toolbox.
- miraudio(..., 'Normal') normalizes with respect to RMS energy (cf. § 3.1).

EXTRACTION OPTIONS
- miraudio(..., 'Extract', \(t1, t2, u\)) extracts the signal between the dates \(t1\) and \(t2\), expressed in the unit \(u\).
  - Possible units \(u = 's'\) (seconds, by default) or \(u = 'sp'\) (sample index, starting from 1).

MIR toolbox 1.1 User’s Manual 18
• **miraudio(..., *Trim*)** trims the pseudo-silence beginning and end off the audio file.

• **miraudio(..., *TrimThreshold*, t)** specifies the trimming threshold t. Silent frames are frames with RMS energy below t times the medium RMS of the whole audio file. Default value: \( t = 0.06 \).

• Instead of ‘Trim’, ‘TrimStart’ only trims the beginning of the audio file, whereas ‘TrimEnd’ only trims the end.

**Labeling Option**

miraudio(..., *Label*, lb) labels the audio signals following the name of their respective audio files. lb is one number, or an array of numbers, and the audio signals are labelled using the substring of their respective file name of index lb. If \( lb = 0 \), the audio signal(s) are labelled using the whole file name.

<table>
<thead>
<tr>
<th>miraudio('Folder', 'Label', lb)</th>
<th>song1g.wav</th>
<th>song2g.wav</th>
<th>song3b.au</th>
</tr>
</thead>
<tbody>
<tr>
<td>( lb = 6 )</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>( lb = [5, 6] )</td>
<td>1g</td>
<td>2g</td>
<td>3b</td>
</tr>
</tbody>
</table>

*Example of labelling of a folder of audio files*

The labeling is used for classification purposes (cf. mirclassify).

**Summation**

Audio signals can be superposed using the basic Matlab summation operators (+). For instance let’s say we have two sequences:

\[
a1 = \text{miraudio('melody.wav')};
\]

\[
a2 = \text{miraudio('accompaniment.wav')};
\]

Then the two sequences can be superposed using the command:

\[
a = a1 + a2
\]

When superposing miraudio objects, the longest audio are no more truncated, but on the contrary the shortest one are prolonged by silence. When audio have different sampling rates, all are converted to the highest one.
**mirsave**

**SAVING AUDIO WAVEFORM INTO FILES**

Audio waveforms, loaded, produced or/and transformed in MIRtoolbox, can be saved as audio files.

**FLOWCHART INTERCONNECTIONS**

miraudio → mirsave

**mirsave** accepts as input data type:

- miraudio objects,
- file names or the ‘Folder’ keyword.

**OPTIONS**

The name and extension of the saved file can be specified in different ways, as shown in the tables below.

- By default, the files are saved in WAV format, using the extension ‘.mir.sav’ in order to lower the risk of overwriting original audio files.

- If the string ‘.au’ is indicated as second argument of mirsave, the audio will be saved in AU format.

- A string can be indicated as second argument of mirsave.
  
  - If the miraudio object to be saved contains only one audio file, the specified string will be used as the name of the new audio file.
  
  - If the miraudio object to be saved contains several audio files, the specified string will be concatenated to the original name of each audio file.

- If the second argument of mirsave is a string ended by ‘.au’, the file name will follow the convention explained in the previous point, and the files will be saved in AU format.

<table>
<thead>
<tr>
<th>miraudio(mysong.au)</th>
<th>mysong.au</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>mirsave(a)</strong></td>
<td><strong>mysong.mir.wav</strong></td>
</tr>
<tr>
<td><strong>mirsave(a, 'new')</strong></td>
<td>new.wav</td>
</tr>
<tr>
<td><strong>mirsave(a, '.au')</strong></td>
<td><strong>mysong.mir.au</strong></td>
</tr>
<tr>
<td><strong>mirsave(a, 'new.au')</strong></td>
<td>new.au</td>
</tr>
</tbody>
</table>
Diverse ways of saving an audio file.

<table>
<thead>
<tr>
<th><code>a = miraudio('Folder')</code></th>
<th><code>song1.wav</code></th>
<th><code>song2.wav</code></th>
<th><code>song3.au</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mirsave(a)</code></td>
<td><code>song1.mir.wav</code></td>
<td><code>song2.mir.wav</code></td>
<td><code>song3.mir.wav</code></td>
</tr>
<tr>
<td><code>mirsave(a,'new')</code></td>
<td><code>song1new.wav</code></td>
<td><code>song2new.wav</code></td>
<td><code>song3new.wav</code></td>
</tr>
<tr>
<td><code>mirsave(a,'.au')</code></td>
<td><code>song1.mir.au</code></td>
<td><code>song2.mir.au</code></td>
<td><code>song3.mir.au</code></td>
</tr>
<tr>
<td><code>mirsave(a,'new.au')</code></td>
<td><code>song1new.au</code></td>
<td><code>song2new.au</code></td>
<td><code>song3new.au</code></td>
</tr>
</tbody>
</table>

Diverse ways of saving a batch of audio files.
**Frame Decomposition**

The analysis of a whole temporal signal (such as an audio waveform in particular) leads to a global description of the average value of the feature under study. In order to take into account the dynamic evolution of the feature, the analysis has to be carried out on a short-term window that moves chronologically along the temporal signal. Each position of the window is called a frame.

**Flowchart Interconnections**

`mirframe` accepts as input any temporal object:

- an audio waveform `miraudio`,
- file names or the 'Folder' keyword,
- an envelope `mirenvelope`,
- the temporal evolution of a scalar data, such as fluxes in particular (`mirflux`),
- in particular, onset detection curves (`mironsets`) can be decomposed into frames as well.

**Syntax**

The frame decomposition can be performed using the `mirframe` command. The frames can be specified as follows:

```plaintext
mirframe(x,..., 'Length', w, wu):
```

- \( w \) is the length of the window in seconds (default: .05 seconds);
- \( u \) is the unit, either
  - 's' (seconds, default unit),
  - or 'sp' (number of samples).

```plaintext
mirframe(x,..., 'Hop', h, hu):
```

- \( b \) is the hop factor, or distance between successive frames (default: half overlapping: each frame begins at the middle of the previous frame)
- \( u \) is the unit, either
Frame decomposition of an audio waveform, with frame length \( l \) and hop factor \( h \) (represented here, following the default unit, as a ratio with respect to the frame length).

These arguments can also be written as follows (where some of these parameters can be omitted):

\[
\text{mirframe}(x, w, wu, h, hu)
\]

**Chainming of operations**

Suppose we load an audio file:

\[
a = \text{miraudio}(\text{mysong})
\]

then we decompose into frames

\[
f = \text{mirframe}(a)
\]

then we can perform any computation on each of the successive frame easily. For instance, the computation of the spectrum in each frame (or spectrogram), can be written as:

\[
s = \text{mirspecturm}(f)
\]
THE ‘FRAME’ OPTION

The two first previous commands can be condensed into one line, using the ‘Frame’ option.

\[ f = \text{miraudio}(\text{mysong}, \text{‘Frame’}) \]

and the three commands can be condensed into one line also using the ‘Frame’ option.

\[ s = \text{mirspectrum}(\text{mysong}, \text{‘Frame’}) \]

The frame specifications can be expressed in the following way:

\[ \text{mirspectrum}(..., \text{‘Frame’}, l, \text{‘s’}, b, \text{‘/x’}) \]

This ‘Frame’ option is available to most operators. Each operator uses specific default values for the ‘Frame’ parameters. Each operator can perform the frame decomposition where it is most suitable. For instance, as can be seen in the feature map (§), the ‘Frame’ option related to the \text{mironsets} operator will lead to a frame decomposition after the actual computation of the onset detection curve (produced by \text{mironsets}).
FILTERBANK DECOMPOSITION

It is often interesting to decompose the audio signal into a series of audio signals of different frequency register, from low frequency channels to high frequency channels. This enables thus to study each of these channels separately. The decomposition is performed by a bank of filters, each one selecting a particular range of frequency values. This transformation models an actual process of human perception, corresponding to the distribution of frequencies into critical bands in the cochlea.

FLOWCHART INTERCONNECTIONS

mirfilterbank accepts as input data type either:

• miraudio objects, where the audio waveform can be segmented (using mirsegment),

• file names or the ‘Folder’ keyword.

FILTERBANK SELECTION

Two filterbanks are proposed in MIRtoolbox so far:

• mirfilterbank(...) , ‘Gammatone’ carries out a Gammatone filterbank decomposition (Patterson et al, 1992). It is known to simulate well the response of the basilar membrane. It is based on a Equivalent Rectangular Bandwidth (ERB) filterbank, meaning that the width of each band is determined by a particular psychoacoustical law. For Gammatone filterbanks, mirfilterbank calls the Auditory Toolbox routines MakeERBFilters and ERBFilterbank. This is the default choice when calling mirfilterbank.
Ten ERB filters between 100 and 8000Hz (Slaney, 1998)

If the number of channels exceeds 20, the audio waveform decomposition is represented as a single image bitmap, where each line of pixel represents each successive channel:

```
mirfilterbank('ragtime')
```
mirfilterbank('ragtime', 'NbChannels', 40)

- `mirfilterbank(..., '2Channels')` performs a computational simplification of the filterbank using just two channels, one for low-frequencies, below 1000 Hz, and one for high-frequencies, over 1000 Hz (Tolonen and Karjalainen, 2000). On the high-frequency channel is performed an envelope extraction using a half-wave rectification and the same low-pass filter used for the low-frequency channel. This filterbank is mainly used for multi-pitch extraction (cf. `mirpitch`).

```

Diagram of the two-channel filterbank proposed in (Tolonen and Karjalainen, 2000)
```

**Options**

- `mirfilterbank(..., 'NbChannels', N)` specifies the number of channels in the bank. By default: $N = 10$. This option is useless for '2Channels'.

- `mirfilterbank(..., 'Channel', c) – or mirfilterbank(..., 'Channels', c)` – only output the channels whose ranks are indicated in the array c. (default: $c = (1:N)$)
**Amplitude Envelope**

From an audio waveform can be computed the envelope, which shows the global outer shape of the signal. It is particularly useful in order to show the long term evolution of the signal, and has application in particular to the detection of musical events such as notes.

Here is an example of audio file with its envelope:

![Audio waveform of ragtime excerpt](image1)

![Corresponding envelope of the ragtime excerpt](image2)

**Flowchart Interconnections**

```
mirfilterbank
```

```
mirenvelope
```

```
mirframe
```

```
miraudio
```

```
mirsegment
```
mirenvelope accepts as input data type either:

- **miraudio** objects, where the audio waveform can be segmented (using **mirsegment**) and/or decomposed into channels (using **mirfilterbank**),

- **file names** or the `Folder` keyword.

Besides, **mirenvelope(..., 'Frame', ...)** directly performs a frame decomposition on the resulting envelope\(^1\). Default value: window length of 50 ms and half overlapping.

**PARAMETERS SPECIFICATION**

The envelope is computed in the following way (Tzanetakis, Essl and Cook, 2001):

- First the signal can be converted from the real domain to the complex domain using a Hilbert transform. In this way the envelope is estimated in a three-dimensional space defined by the product of the complex domain and the temporal axis. Indeed in this representation the signal looks like a “spring” of varying width, and the envelope would correspond to that varying width. In the real domain, on the other hand, the constant crossing of the signal with the zero axis may sometime give erroneous results.

An Hilbert transform can be performed in **mirenvelope**, based on the Matlab function **hilbert**. In order to toggle on the Hilbert transform, the following keyword should be added:

\[ \text{mirenvelope(..., 'Hilbert')} \]

Beware however that, although sometimes the use of the Hilbert transform seems to improve somewhat the results, and might in particular show clearer burst of energy, we noticed some problematic behavior, in particular at the beginning and the end of the signal, and after some

\(^1\) The frame decomposition should not be performed before the envelope extraction, as it would induce significant redundancy in the computation and arouse problems related to the transitory phases at the beginning of each frame.
particular bursts of energy. This becomes all the more problematic when chunk decompositions are used (cf. section...), since the continuity between chunk cannot be ensured any more. For that reason, since version 1.1 of MIRtoolbox, the use of Hilbert transform is toggled off by default.

- If the signal is in the real domain, the next step consists in a full-wave rectification, reflecting all the negative lobes of the signal into the positive domain, leading to a series of positive half-wave lobes. The further smoothing of the signal (in the next step) will leads to an estimation of the envelope. If on the contrary the signal is in the complex domain, a direct estimation of the envelope can be obtained by computing the modulus, i.e., the width of the “string”. These two operations, either from the real or the complex domains, although apparently different, relate to the same Matlab command abs.

- The next step consists in a low-pass filter than retain from the signal only the long-term evolution, by removing all the more rapid oscillations. This is performed by a one-pole filter. The range of frequencies to be filtered can be controlled by selecting a proper value for the $a$ parameter. Another way of expressing this parameter is by considering its time constant. If we feed the filter with a step function (i.e. 0 before time 0, and 1 after time 0), the time constant will correspond to the time it will take for the output to reach 63% of the input. Hence higher time constant means smoother filtering. The default time constant is set to .02 seconds and can be changed using the option:

\[ \text{mirenvelope}(..., \text{'Tau'}, t) \]

Remarks:

1. As low-pass filters actually lead to a shifting of the phases of the signal. This is counteracted using a second filtering of the reverse signal. The time constant $t$ is the time constant of each separate filter, therefore the resulting time constant is around twice bigger.

2. The reverse filtering is not performed using Matlab `filtfilt` function since version 1.1 of MIRtoolbox – because this would not work in the case of chunk decomposition (cf.) – but has been partly re-implemented. In particular, contrary to `filtfilt`, care is not yet taken to minimize startup and ending transients by matching initial conditions.

- Once the signal has been smoothed, as there is a lot of redundancy between the successive samples, the signal can be down-sampled. The default parameter related to down-sampling is the down-sampling rate $N$, i.e. the integer ratio between the old and the new sampling rate. $N$ is set by default to 16, and can be changed using the option:

\[ \text{mirenvelope}(..., \text{'Down'}, N) \]
Alternatively, any sampling rate $r$ (in Hz) can be specified using the option:

$mirenvelope(..., 'Sampling', r)$

**POST-PROCESSING OPTIONS**

Different operations can be performed on the envelope curve:

- $mirenvelope(..., 'Center')$ centers the extracted envelope.

![Envelope](image1)

- $mirenvelope(..., 'HalfwaveCenter')$ performs a half-wave rectification on the centered envelope.

![Envelop (half-wave rectified)](image2)

- $mirenvelope(..., 'Diff')$ computes the differentiation of the envelope, i.e., the differences between successive samples.

![Differentiated envelope](image3)

- $mirenvelope(..., 'HalfwaveDiff')$ performs a half-wave rectification on the differentiated envelope.
• `mirenvelope(..., 'Normal')` normalizes the values of the envelope by fixing the maximum value to 1.

• `mirenvelope(..., 'Smooth', o)` smooths the envelope using a moving average of order o. The default value when the option is toggled on: $o=30$

• `mirenvelope(..., 'Gauss', o)` smooths the envelope using a gaussian of standard deviation o samples. The default value when the option is toggled on: $o=30$
**mirspecrum**

**FOURIER TRANSFORM**

A decomposition of the energy of a signal (be it an audio waveform, or an envelope, etc.) along frequencies can be performed using a Discrete Fourier Transform, which, for an audio signal \( x \) has for equation:

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

This decomposition is performed using a Fast Fourier Transform by the `mirspecrum` function by calling Matlab `fft` function. The graph returned by the function highlights the repartition of the amplitude of the frequencies (i.e., the modulus of \( X_k \) for all \( k \)), such as the following:

We can also obtain for each frequency the actual phase position (i.e., the phase of \( X_k \)), which indicates the exact position of each frequency component at the instant \( t = 0 \). If the result of the spectrum decomposition is \( s \), the phase spectrum is obtained by using the command:

\[
\text{get}(s, \text{'Phase'})
\]

**FLOWCHART INTERCONNECTIONS**

![Flowchart Diagram]

`mirspecrum` accepts as input data type either:
• **miraudio** objects, where the audio waveform can be segmented (using **mirsegment**), decomposed into channels (using **mirfilterbank**), and/or decomposed into frames (using **mirframe** or the ‘**Frame**’ option, with by default a frame length of 50 ms and half overlapping);

• **file names** or the ‘**Folder**’ keyword;

• data in the onset detection curve category (cf. **mironsets**):
  
  • **mirenvelope** objects, frame-decomposed or not,
  
  • fluxes (cf. **mirflux**), frame-decomposed or not;

• **mirspectrump** frame-decomposed objects: by calling again **mirspectrump** with the ‘**Along-Bands**’ option, Fourier transforms are computed this time on each temporal signal related to each separate frequency bin (or frequency band, cf. below).

**PARAMETERS SPECIFICATION**

The range of frequencies, in Hz, can be specified by the options:

• **mirspectrump(..., ‘Min’, mi)** indicates the lowest frequency taken into consideration, expressed in Hz. Default value: 0 Hz.

• **mirspectrump(..., ‘Max’, ma)** indicates the highest frequency taken into consideration, expressed in Hz. Default value: the maximal possible frequency, corresponding to the sampling rate divided by 2.

• **mirspectrump(..., ‘Window’, w)** specifies the windowing method. Windows are used to avoid the problems due to the discontinuities provoked by finite signals. Indeed, an audio sequence is not infinite, and the application of the Fourier Transform requires to replace the infinite time before and after the sequence by zeroes, leading to possible discontinuities at the borders. Windows are used to counteract those discontinuities. Possible values for *w* are either *w* = 0 (no windowing) or any windowing function proposed in the **Signal Processing Toolbox**. Default value: *w* = ‘**hamming**’, the Hamming window being a particular good window for Fourier Transform.

• **mirspectrump(...,’NormalInput’)** normalizes the waveform between 0 and 1 before computing the Fourier Transform.

---

2 The list of possible window arguments can be found in the **window** documentation (*help window*).
RESOLUTION SPECIFICATION

The frequency resolution of the spectrum directly depends on the size of the audio waveform: the longer the waveform, the better the frequency resolution. It is possible, however, to increase the frequency resolution of a given audio waveform by simply adding a series of zeros at the end of the sequence, which is called zero-padding. Besides, an optimized version of the Discrete Fourier Transform, called Fast Fourier Transform (FFT) can be performed if the length of the audio waveform (including the zero-padding) is a power of 2. For this reason, by default, a zero-padding is performed by default in order to ensure that the length of the audio waveform is a power of 2. But these operations can be tuned individually:

- *mirspecrum(....,’MirRes’, mr)* adds a constraint related to the a minimal frequency resolution, fixed to the value *mr* (in Hz). The audio waveform is automatically zero-padded to the lowest power of 2 ensuring the required frequency resolution.

- *mirspecrum(....,’Res’, r)* specifies the frequency resolution *r* (in Hz) that will be secured as closely as possible, through an automated zero-padding. The length of the resulting audio waveform will not necessarily be a power of 2, therefore the FFT routine will rarely be used.

- *mirspecrum(....,’Length’, l)* specifies the length of the audio waveform after zero-padding. If the length is not a power of 2, the FFT routine will not be used.

- *mirspecrum(....,’ZeroPad’, s)* performs a zero-padding of *s* samples. If the total length is not a power of 2, the FFT routine will not be used.

POST-PRESENTATION OPTIONS

- *mirspecrum(....,’Normal‘)* normalizes with respect to energy: each magnitude is divided by the euclidian norm (root sum of the squared magnitude).

- *mirspecrum(....,’Power‘)* squares the energy: each magnitude is squared.

- *mirspecrum(....,’dB‘)* represents the spectrum energy in decibel scale. For the previous example we obtain the following spectrum:

![Spectrum graph]

- *mirspecrum(....,’dB’, th)* keeps only the highest energy over a range of *th* dB. For example if we take only the 20 most highest dB in the previous example we obtain:
• \textit{mirspectrum}(\ldots, \textit{Resonance}, r) multiplies the spectrum curve with a resonance curve that emphasizes pulsations that are more easily perceived. Two resonance curves are available:
  
  - \( r = \textit{ToiviainenSnyder} \) (Toiviainen & Snyder 2003), default choice for onset detection curve (cf. \textit{mirtempo}),
  
  - \( r = \textit{Fluctuation} \): fluctuation strength (Fastl 1982), default choice for frame-decomposed \textit{mirspectrum} objects (cf. \textit{mirfluctuation}).

• \textit{mirspectrum}(\ldots, \textit{Smooth}, o) smooths the envelope using a moving average of order \( o \). Default value when the option is toggled on: \( o=10 \)

• \textit{mirspectrum}(\ldots, \textit{Gauss}, o) smooths the envelope using a gaussian of standard deviation \( o \) samples. Default value when the option is toggled on: \( o=10 \)

**AUDITORY MODELS**

Different auditory modeling can be applied directly to the spectrum representation. Let’s suppose our initial spectral representation is the following:

• \textit{mirspectrum}(\ldots, \textit{Terhardt}) modulates the energy following (Terhardt, 1979) outer ear model. The function is mainly characterized by an attenuation in the lower and higher registers of the spectrum, and an emphasis around 2–5 KHz, where much of the speech information is carried. (Code based on Pampalk’s MA toolbox). In our example, the output of the model is:
mirspectrum(\ldots,\texttt{Cents}) redistributes the frequencies along cents. Each octave is decomposed into 1200 bins equally distant in the logarithmic representation. The frequency axis is hence expressed in MIDI-cents unit: to each pitch of the equal temperament is associated the corresponding MIDI pitch standard value multiply by 100 (69*100=6900 for A4=440Hz, 70*100=7000 for B4, etc.).

It has to be noticed that this decomposition requires a frequency resolution that gets higher for lower frequencies: a cent-distribution starting from infinitely low frequency (near 0 Hz would require an infinite frequency resolution). Hence by default, the cent-decomposition is defined only for the frequency range suitable for the frequency resolution initially associated to the given spectrum representation. Two levers are available here:

- If a minimal frequency range for the spectrum representation has been set (using the 'Min' parameter), the frequency resolution of the spectrum is automatically set in order to meet that particular requirement.

- By increasing the frequency resolution of the spectrum (for instance by using the 'Res' or 'MinRes' parameters), the frequency range will be increased accordingly.
• *mirscale*(..., *'Collapsed'*) collapses the cent-spectrum into one octave. In the resulting spectrum, the abscissa contains in total 1200 bins, representing the 1200 cents of one octave, and each bin contains the energy related to one position of one octave and of all the multiple of this octave.

![Spectrum](image)

*mirscale*(ragtime,'Cents','Min',100,'Collapsed')

• *mirscale*(..., *'Mel'*) redistributes the frequencies along Mel bands. The Mel-scale of auditory pitch was established on the basis of listening experiments with simple tones (Stevens and Volkman, 1940). The Mel scale is now mainly used for the reason of its historical priority only. It is closely related to the Bark scale. It requires the *Auditory Toolbox*. In our example we obtain the following:

![Mel-Spectrum](image)

• *mirscale*(..., *'Bark'*) redistributes the frequencies along critical band rates (in Bark). Measurement of the classical “critical bandwidth” typically involves loudness summation experiments (Zwicker et al., 1957). The critical band rate scale differs from Mel-scale mainly in that it uses the critical band as a natural scale unit. The code is based on the *MA* toolbox.

• *mirscale*(..., *'Mask'*) models masking phenomena in each band: when a certain energy appears at a given frequency, lower frequencies in the same frequency region may be unheard, following particular equations. By modeling these masking effects, the unheard periodicities are removed from the spectrum. The code is based on the *MA* toolbox. In our example this will lead to:
HARMONIC SPECTRAL ANALYSIS

A lot of natural sounds, especially musical ones, are harmonic: each sound consists of a series of frequencies at a multiple ratio of the one of lowest frequency, called fundamental. Techniques have been developed in signal processing to reduce each harmonic series to its fundamental, in order to simplify the representation. MIRtoolbox includes for the moment two closed techniques (Alonso et al, 2003):

- \texttt{mirspectrum}(..., \texttt{Prod}, m) Enhances components that have harmonics located at multiples of range(s) m of the signal’s fundamental frequency. Computed by compressing the signal by the list of factors m, and by multiplying all the results with the original signal. Default value is \( m = 1:6 \). Hence for this initial spectrum:

\[ \begin{array}{c|c|c|c|c|c|c|c} 
\text{Spectrum frequency (Hz)} & 0 & 1000 & 2000 & 3000 & 4000 & 5000 & 6000 \\
\hline 
\text{magnitude} & 0 & 200 & 400 & 600 & 800 & 1000 & \end{array} \]

we obtain this reduced spectrum:

- \texttt{mirspectrum}(..., \texttt{Sum}, m) Similar idea using addition of the multiples instead of multiplication.

\[ \begin{array}{c|c|c|c|c|c|c|c} 
\text{Spectrum frequency (Hz)} & 0 & 1000 & 2000 & 3000 & 4000 & 5000 & 6000 \\
\hline 
\text{magnitude} & 0 & 1 & 2 & 3 & 4 & 5 & \times 10^3 \end{array} \]
Spectral Analysis of Spectrum

The harmonic sequence can also be used for the detection of the fundamental frequency itself. One idea is to look at the spectrum representation, and try to automatically detect these periodic sequences. And one simple idea consists in performing a Fourier Transform of the Fourier Transform itself, leading to a so-called cepstrum (Bogert et al., 1963).

So if we take the complex spectrum ($X_k$ in the equation defining $\text{mir spectrum}$), we can operate the following chain of operations:

1. First a logarithm is performed in order to allow an additive separability of product components of the original spectrum. For instance, for the voice in particular, the spectrum is composed of a product of a vocal cord elementary burst, their echoes, and the vocal track. In the logarithm representations, these components are now added one to each other, and we will then be able to detect the periodic signal as one of the components.

2. Then because the logarithm provokes some modification of the phase, it is important to ensure that the phase remains continuous.

3. Finally the second Fourier transform is performed in order to find the periodic sequences. As it is sometime a little difficult to conceive what a Fourier transform of Fourier transform is really about, we can simply say, as most say, that it is in fact an Inverse Fourier Transform (as it is the same thing, after all), and the results can then be expressed in a kind of temporal domain, with unit called “quefrency”.

For instance for this spectrum:
we obtain the following cepstrum:

The cepstrum can also be computed from the spectrum amplitude only, by simply taking the logarithm, and directly computing the Fourier transform.

In this case, the phase of the spectrum is not computed.
**Flowchart Interconnections**

`mircepstrum` accepts either:

- `mirspectrurm` objects, or
- `miraudio` objects (same as for `mirspectrurm`),
- `file names` or the `Folder` keyword.

**Parameter Specifications**

- `mircepstrum(..., 'Freq')`: The results can be represented, instead of using the quefrency domain (in seconds), back to the frequency domain (in Hz) by taking the inverse of each abscissae value. In this frequency representation, each peak is located on a position that directly indicates the associated fundamental frequency.

- `mircepstrum(..., 'Min', min)` specifies the lowest delay taken into consideration, in seconds. Default value: 0 s

- `mircepstrum(..., 'Max', max)` specifies the highest delay taken into consideration, in seconds. Default value: 0.05 s (corresponding to a minimum frequency of 20 Hz).
A U T O C O R R E L A T I O N  F U N C T I O N

Another way to evaluate periodicities in signals (be it an audio waveform, a spectrum, an envelope, etc.) consists in looking at local correlation between samples. If we take a signal $x$, such as for instance this trumpet sound:

the autocorrelation function is computed as follows:

$$R_{xx}(j) = \sum_{n} x_n \bar{x}_{n-j}.$$ 

For a given lag $j$, the autocorrelation $Rxx(j)$ is computed by multiplying point par point the signal with a shifted version of it of $j$ samples. We obtain this curve:

Hence when the lag $j$ corresponds to a period of the signal, the signal is shifted to one period ahead, and therefore is exactly superposed to the original signal. Hence the summation gives very high value, as the two signals are highly correlated.
FLOWCHART INTERCONNECTIONS

`mirautocor` usually accepts either:

- **file names** or the `'Folder'` keyword,

- `miraudio` objects, where the audio waveform can be segmented (using `mirsegment`), decomposed into channels (using `mirfilterbank`), and/or decomposed into frames (using `mirframe` or the `'Frame'` option, with by default a frame length of 50 ms and half overlapping),

- `mirspectrum` objects,

- data in the onset detection curve category (cf. `mironsets`):
  - `mirenvelope` objects, frame-decomposed or not,
  - fluxes (cf. `mirflux`), frame-decomposed or not.

PARAMETERS SPECIFICATION

- `mirautocor(..., 'Min', mi)` indicates the lowest delay taken into consideration. Default value: 0 s. The unit can be precised:
  - `mirautocor(..., 'Min', mi, 's')` (default unit)
  - `mirautocor(..., 'Min', mi, 'Hz')`
- `mirautocor(..., 'Max', ma)` indicates the highest delay taken into consideration. The unit can be specified as for `'Min'`. Default value:
  - if the input is an audio waveform, the highest delay is 0.05 s (corresponding to a minimum frequency of 20 Hz).
  - if the input is an envelope, the highest delay is 2 s.
• `mirautocor(..., 'Normal', n)` specifies a normalization option for the cross-correlation (`biased`, `unbiased`, `coeff`, `none`). This corresponds exactly to the normalization options in Matlab `xcorr` function, as `mirautocor` actually calls `xcorr` for the actual computation. The default value is `coeff`, corresponding to a normalization so that the autocorrelation at zero lag is identically 1. Note however that this routine is not used when the compression (`Compres`) factor $k$ is not equal to 2 (see below).

**POST-PROCESSING OPTIONS**

• `mirautocor(..., 'Freq')` represents the autocorrelation function in the frequency domain: the periods are expressed in Hz instead of seconds (see the last curve in the figure below for an illustration).

• `mirautocor(..., 'NormalWindow')` divides the autocorrelation by the autocorrelation of the window. Boersma (1993) shows that by default the autocorrelation function gives higher coefficients for small lags, since the summation is done on more samples. Thus by dividing by the autocorrelation of the window, we normalize all coefficients in such a way that this default is completely resolved. At first sight, the window should simply be a simple rectangular window. But Boersma (1993) shows that it is better to use `hanning` window in particular, in order to obtain better harmonic to noise ratio.

• `mirautocor(..., 'NormalWindow', w)` specifies the window to be used, which can be any window available in the Signal Processing Toolbox. Besides $w = 'rectangular'$ will not perform any particular windowing (corresponding to a rectangular (“invisible”) window), but the normalization of the autocorrelation by the autocorrelation of the invisible window will be performed nonetheless. The default value is $w = 'hanning'$.

• `mirautocor(..., 'NormalWindow', 'off')` toggles off this normalization (which is `on` by default).

• `mirautocor(..., 'Resonance')` multiplies the spectrum curve with a resonance curve that emphasizes pulsations that are more easily perceived (Toiviainen & Snyder 2003).

• `mirautocor(..., 'Halfwave')` performs a half-wave rectification on the result, in order to just show the positive autocorrelation coefficients.

**GENERALIZED AUTOCORRELATION**

`mirautocor(..., 'Compres', k)` – or equivalently `mirautocor(..., 'Generalized', k)` – computes the autocorrelation in the frequency domain and includes a magnitude compression of the spectral representation. Indeed an autocorrelation can be expressed using Discrete Fourier Transform as

$$ y = \text{IDFT}(|\text{DFT}(x)|^2), $$
which can be generalized as:

\[ y = IDFT(DFT(x) |^k) \]

Compression of the autocorrelation (i.e., setting a value of \( k \) lower than 2) are recommended in (Tolonen & Karjalainen, 2000) because this decreases the width of the peaks in the autocorrelation curve, at the risk however of increasing the sensitivity to noise. According to this study, a good compromise seems to be achieved using value \( k = .67 \). By default, no compression is performed (hence \( k = 2 \)), whereas if the 'Compress' keyword is used, value \( k = .67 \) is set by default if no other value is indicated.

**Enhanced Autocorrelation**

In the autocorrelation function, for each periodicity in the signal, peaks will be shown not only at the lag corresponding to that periodicity, but also to all the multiples of that periodicity. In order to avoid such redundancy of information, techniques have been proposed that automatically remove these harmonics. In the frequency domain, this corresponds to sub-harmonics of the peaks.

`mirautocor(..., 'Enhanced', a)`: The original autocorrelation function is half-wave rectified, time-scaled by factor \( a \) (which can be a factor list as well), and subtracted from the original clipped function (Tolonen & Karjalainen, 2000). If the 'Enhanced' option is not followed by any value, the default value is \( a = 2:10 \), i.e., from 2 to 10. See the figure below for an example of enhanced autocorrelation from a piano Amin3 chord, with the successive step of the enhancement:

`mirautocor('Amaj3', 'Enhanced')`
fig 1: Waveform autocorrelation of a piano chord Amaj3 (blue), and scaled autocorrelation of factor 2 (red);
fig 2: subtraction of the autocorrelation by the previous scaled autocorrelation (blue), scaled autocorrelation of factor 3 (red); fig 3: resulting subtraction (blue), scaled autocorrelation of factor 4 (red); fig 4: idem for factor 5; fig 5: idem for factor 6; fig 6: idem for factor 7; fig 7: resulting autocorrelation curve in the frequency domain and peak picking
COMBINING REPRESENTATIONS

It is also possible to multiple points by points diverse spectral representations and autocorrelation functions, the latter being automatically translated to the spectrum domain (Peeters, 2006).

For instance given the autocorrelation of a waveform $a$:

$$ac = \text{mirautocor}(a)$$

given the autocorrelation of the spectrum of $a$:

$$s = \text{mirspectrum}(a)$$

$$as = \text{mirautocor}(s)$$

and given the cepstrum of $a$:

$$cp = \text{mircepstrum}(a)$$
we can multiply the three representations one with each other:

\[ ac \ast as \]

\[ ac \ast cp \]

\[ as \ast cp \]
**DISTANCE BETWEEN SUCCESSIVE FRAMES**

Given a spectrogram:

\[ s = \text{mirspectrum}(a, \text{Frame}) \]

we can compute the spectral flux as being the distance between the spectrum of each successive frames.

\[ \text{mirflux}(s) \]

The peaks in the curve indicate the temporal position of important contrast in the spectrogram.

In *MIRtoolbox* fluxes are generalized to any kind of frame-decomposed representation, for instance a cepstral flux:

\[ c = \text{mircepstrum}(a, \text{Frame}) \]
`mirflux(c)`

Cepstrum flux

Flowchart Interconnections

`mirflux` usually accepts either:

- `mirspectrum` frame-decomposed objects.
- `miraudio` objects, where the audio waveform can be segmented (using `mirsegment`), decomposed into channels (using `mirfilterbank`). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 200 ms and a hop factor of 1.3 – can be changed using the ‘Frame’ option. If the input is a `miraudio`...
dio object, the default flux is a spectral flux: i.e., the audio waveform is passed to the mirspec-
trum operator before being fed into mirflux.

- **file names** or the 'Folder' keyword: same behavior than for miraudio objects;
- **mirautocor** frame-decomposed objects;
- **mirecepstrum** frame-decomposed objects;
- **mirmfcc** frame-decomposed objects;
- **mirchromagram** frame-decomposed objects;
- **mirkeystrength** frame-decomposed objects.

**PARAMETERS SPECIFICATION**
- **mirflux(x, 'Dist', d)** specifies the distance between successive frames:
  - \( d = '\text{Euclidian}' \): Euclidian distance (Default)
  - \( d = '\text{City}' \): City-block distance
  - \( d = '\text{Cosine}' \): Cosine distance (or normalized correlation)

- **mirflux(..., 'Inc')**: Only positive difference between frames are summed, in order to focus on increase of energy solely.

- **mirflux(..., 'Complex')**, for spectral flux, combines the use of both energy and phase information (Bello et al, 2004).

**POST-PROCESSING**
- **mirflux(..., 'Halfwave')**: performs a half-wave rectification on the result.

- **mirflux(..., 'Median', l, C)**: removes small spurious peaks by subtracting to the result its me-
dian filtering. The median filter computes the point-wise median inside a window of length \( l \)
(in seconds), that includes a same number of previous and next samples. \( C \) is a scaling factor
whose purpose is to artificially rise the curve slightly above the steady state of the signal. If
no parameters are given, the default values are: \( l = 0.2 \) s. and \( C = 1.3 \)

- **mirflux(..., 'Median', l, C, 'Halfwave')**: The scaled median filtering is designed to be suc-
cceeded by the half-wave rectification process in order to select peaks above the dynamic
threshold calculated with the help of the median filter. The resulting signal is called “detection_
function” (Alonso, David, Richard, 2004). To ensure accurate detection, the length \( l \) of the
median filter must be longer than the average width of the peaks of the detection function.
SUMMATION OF FILTERBANK CHANNELS

Once an audio waveform is decomposed into channels using a filterbank:

\[ f = \text{mirfilterbank}(a) \]

An envelope extraction, for instance, can be computed using this very simple syntax:

\[ e = \text{mirenvelope}(f) \]

Then the channels can be summed back using the \textit{mirsum} command:

\[ s = \text{mirsum}(e) \]
The summation can be centered using the command:

\[ s = \text{mirsum}(e, \text{\textquoteleft}Center\textquoteright) \]

**SUMMARY OF FILTERBANK CHANNELS**

If we compute for instance an autocorrelation from the envelopes:

\[ ac = \text{mirautocor}(e) \]

Then we can sum all the autocorrelation using exactly the same `mirsum` command:

\[ s = \text{mirsum}(e) \]

This summation of non-temporal signals across channels is usually called `summary`.
**mirpeaks**

**Peak picking**

Peaks (or important local maxima) can be detected automatically from any data \( x \) produced in MIRtoolbox using the command

\[
\text{mirpeaks}(x)
\]

If \( x \) is a curve, peaks are represented by red circles:

![Graph showing peaks on a curve]

If \( x \) is a frame-decomposed matrix, peaks are represented by white crosses:

![Graph showing peaks on a frame-decomposed matrix]

**Parameters specification**

- \textit{mirpeaks}(..., \textit{Total}, \( m \)): only the \( m \) highest peaks are selected. If \( m = \text{Inf} \), no limitation of number of peaks. Default value: \( m = \text{Inf} \)

- Border effects can be specified:
  - \textit{mirpeaks}(..., \textit{NoBegin}) does not consider the first sample as a possible peak candidate.
  - \textit{mirpeaks}(..., \textit{NoEnd}) does not consider the last sample as a possible peak candidate.

- \textit{mirpeaks}(..., \textit{Order}, \( o \)) specifies the ordering of the peaks.
  - \( o = \text{Amplitude} \) orders the peaks from highest to lowest (Default choice.)
• $o = \text{‘Abscissa’}$ orders the peaks along the abscissa axis.

• $\text{mirpeaks(..., Valleys)}$ detect valleys (local minima) instead of peaks.

• $\text{mirpeaks(..., Contrast, cthr)}$: A given local maximum will be considered as a peak if its distance with the previous and successive local minima (if any) is higher than this threshold $\text{cthr}$. This distance is expressed with respect to the total amplitude of the input signal: a distance of 1, for instance, is equivalent to the distance between the maximum and the minimum of the input signal. Default value: $\text{cthr} = 0.1$

• $\text{mirpeaks(..., SelectFirst, ftbr)}$: If the ‘Contrast’ selection has been chosen, this additional option specifies that when one peak has to be chosen out of two candidates, and if the difference of their amplitude is below the threshold $\text{tbr}$, then the most ancien one is selected. Option toggled off by default. Default value if toggled on: $\text{ftbr} = \text{cthr}/2$

• $\text{mirpeaks(..., Threshold, thr)}$: A given local maximum will be considered as a peak if its normalized amplitude is higher than this threshold $\text{tbr}$. A given local minimum will be considered as a valley if its normalized amplitude is lower than this threshold. The normalized amplitude can have value between 0 (the minimum of the signal in each frame) and 1 (the maximum in each frame). Default value: $\text{thr} = 0$ for peaks, $\text{thr} = 1$ for valleys.

• $\text{mirpeaks(..., Interpol, i)}$ estimates more precisely the peak position and amplitude using interpolation. Performed only on data with numerical abscissae axis.

  • $i = \text{‘no’, ‘off’, 0}$: no interpolation
• $i = \textit{Quadratic}$: quadratic interpolation. (default value).

• $\textit{mirpeaks}(\ldots, \text{\texttt{Reso}}, r)$ removes peaks whose distance to one or several higher peaks is lower than a given threshold. Possible value for the threshold: $r = \textit{SemiTone}$: ratio between the two peak positions equal to $2^{(\pi/12)}$

• $\textit{mirpeaks}(\ldots, \text{\texttt{Pref}}, c, \text{\texttt{std}})$ indicates a region of preference for the peak picking, centered on the abscissa value $c$, with a standard deviation of $\text{\texttt{std}}$.

• $\textit{mirpeaks}(\ldots, \text{\texttt{Nearest}}, t, s)$ takes the peak nearest a given abscissa values $t$. The distance is computed either on a linear scale ($s = \textit{Lin}$) or logarithmic scale ($s = \textit{Log}$). When using the \textit{Nearest} option, only one peak is extracted (overriding hence the \textit{Total} parameter).

• $\textit{mirpeaks}(\ldots, \text{\texttt{Normalize}}, n)$ specifies whether frames are normalized globally or individually.
  - $n = \textit{Global}$ normalizes the whole frames altogether from 0 to 1 (default choice).
  - $n = \textit{Local}$: normalizes each frame from 0 to 1.

• $\textit{mirpeaks}(\ldots, \text{\texttt{Extract}}$) extracts from the curves all the positive continuous segments (or “curve portions”) where peaks are located. First, a low-pass filtered version of the curve is computed, on which the temporal span of the positive lobes containing each peak are stored. The output consists of the part of the original non-filtered curve corresponding to the same temporal span. For instance:

$$ac = \textit{mirautocor}(\text{\texttt{ragtime}})$$

```plaintext
--!
```

![Waveform autocorrelation](image)

```
\textit{mirpeaks}(ac, \text{\texttt{Extract}})
```

MIRtoolbox 1.1 User’s Manual
**mirsegment**

**SEGMENTATION**

- An audio waveform \( a \) can be segmented using the output \( p \) of a peak picking from data resulting from \( a \) itself, using the following syntax:

\[
sg = \text{mirsegment}(a, p)
\]

- An audio waveform \( a \) can also be segmented manually, based on temporal position directly given by the user, in the form:

\[
sg = \text{mirsegment}(a, v)
\]

where \( v \) is an array of number(s) corresponding to time positions in seconds.

- Automated segmentation methods are provided as well, that can be called using the syntax:

\[
sg = \text{mirsegment}(a, \ldots)
\]

and by adding, if necessary, some further options related to the segmentation heuristics (cf. section).

**EXAMPLE**

\[
sg = \text{mirsegment}('\text{ragtime}', '\text{KernelSize}', 32)
\]
The output can be sent to any further analysis, for instance:

\[ sp = \text{mirspectrum}(sg, \text{'dB'}) \]
SONIFICATION OF THE RESULT
Certain classes of temporal data can be sonified:

• **miraudio** objects: the waveform is directly played, and
  • if the audio waveform is segmented (using **mirsegment**), segments are played successively with a short burst of noise in-between;
  • if the audio waveform is decomposed into channels (using **mirfilterbank**), channels are played successively from low to high register;
  • if the audio is decomposed into frames (using **mirframe** or the **Frame** option, with by default a frame length of 50 ms and half overlapping), frames are played successively;
• **file names** or the **Folder** keyword: same behavior than for **miraudio** objects;
• **mirenvelope** objects (frame-decomposed or not) are sonified using a white noise modulated in amplitude by the envelope,

![Diagram of MIR Toolbox](image)

OPTIONS
• **mirplay(..., ‘Channel’, i)** plays the channel(s) of rank(s) indicated by the array i.
• **mirplay(..., ‘Frame’, j)** plays the frame(s) of rank(s) indicated by the array j.
• **mirplay(..., ‘Segment’, k)** plays the segment(s) of rank(s) indicated by the array k.
• **mirplay(..., ‘Sequence’, l)** plays the sequence(s) of rank(s) indicated by the array l.
3. FEATURE EXTRACTORS

The musical feature extractors can be organized along main musical dimensions: dynamics, rhythm, timbre, pitch and tonality.

3.1. Dynamics

**mirrms**

**ROOT-MEAN-SQUARE ENERGY**

The global energy of the signal $x$ can be computed simply by taking the root average of the square of the amplitude, also called root-mean-square (RMS):

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

**FLOWCHART INTERCONNECTIONS**

`miraudio` → `mirfilterbank` → `mirsegment` → `mirframe` → `mirrms`

`mirrms` accepts as input data type either:

- `miraudio` objects, where the audio waveform can be segmented (using `mirsegment`), decomposed into channels (using `mirfilterbank`), and/or decomposed into frames (using `mirframe` or the 'Frame' option, with by default a frame length of 50 ms and half overlapping),
- `file names` or the 'Folder' keyword.

The following command orders the computation of the RMS related to a given audio file:

```
mirrms('ragtime')
```

which produce the resulting message in the Command Window:
The RMS energy related to file ragtime is 0.017932

If we know ask for a frame-decomposed computation of RMS:

\( \text{mirrms('} \text{ragtime'}, \text{'Frame'}) \)

we obtain a temporal evolution of the energy:

We can note that this energy curve is very close to the envelope:

\textit{Option}

- \( \text{mirrms(..., 'Root', 'no'}) \) does not compute the square-root after the averaging.
**mirlowenergy**

**Description**
The energy curve can be used to get an assessment of the temporal distribution of energy, in order to see if its remains constant throughout the signal, or if some frames are more contrastive than others. One way to estimate this consists in computing the low energy rate, i.e. the percentage of frames showing less-than-average energy (Tzanetakis and Cook, 2002).

**Flowchart Interconnections**

The flowchart shows the interconnections between the different functions involved in the computation of the low energy rate.

*mirlowenergy* accepts as input data type either:

- **mirmms frame-decomposed** data,
- **miraudio** objects, where the audio waveform can be segmented (using *mirsegment*), decomposed into channels (using *mirfilterbank*). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 50 ms and half overlapping – can be changed using the *Frame* option.
- **file names** or the *Folder* keyword: same behavior than for **miraudio** objects.

*mirlowenergy* can return several outputs:

1. the low-energy rate itself and
2. the **mirmms frame-decomposed** data.

**Examples**
If we take for instance this energy curve:

\[ \texttt{r1 = mirmms('a1', 'Frame')} \]
We can see that due to some rare frames containing particularly high energy, most of the frames are below the average RMS. And indeed if we compute the low-energy rate

$$mirlowenergy(r_1)$$

we obtain the value 0.71317.

For this opposite example:

$$r_2 = mirrms(a_2, 'Frame')$$

there are two kind of frames basically, those that have quite constant high energy, and fewer that have very low energy. Hence most of the frames are over the average energy, leading to a low low-energy rate:

$$mirlowenergy(r_2)$$

equal to 0.42398

**Options**

- *mirlowenergy(..., *Threshold*, *t)* expressed as a ratio to the average energy over the frames. Default value: $t = 1$
- *mirlowenergy(..., *Root*, ...) specifies the ‘Root’ option used in *mirrms*. Toggled on by default.
- *mirlowenergy(..., *ASR*) computes the Average Silence Ratio, which corresponds in fact to
mirlowenergy(...,'Root', 'no', 'Threshold', t)

where $t$ is fixed here by default to a smaller value $t = .5$
3.2. Rhythm

The estimation of rhythmicity in the audio signal can be performed using the basic operators we introduced previously.

\textit{mirfluctuation}

\textbf{RHYTHMIC PERIODICITY ALONG AUDITORY CHANNELS}

One way of estimating the rhythmic is based on spectrogram computation transformed by auditory modeling and then a spectrum estimation in each band (Pampalk et al., 2002). The process can be detailed as follows:

- First the spectrogram is computed on frames of 23 ms and half overlapping, then the Terhardt outer ear modeling is computed, with Bark-band redistribution of the energy, and estimation of the masking effects, and finally the amplitudes are computed in dB scale:

\[
s = \text{mirspectrum}(..., 'Frame', .023, .5, 'Terhardt', 'Bark', 'Mask', 'dB')
\]

- Then a FFT is computed on each Bark band, from 0 to 10 Hz. The amplitude modulation coefficients are weighted based on the psychoacoustic model of the fluctuation strength (Fastl, 1982). We can see in the matrix the rhythmic periodicities for each different Bark band.

\[
f = \text{mirspectrum}(s, 'AlongBands', 'Max', 10, 'Window', 0, 'Resonance', 'Fluctuation')
\]
\textbullet{} \texttt{mirfluctuation(..., 'Summary')} subsequently sums the resulting spectrum across bands, leading to a spectrum summary, showing the global repartition of rhythmic periodicities:

\begin{equation*}
\texttt{mirsum}(f)
\end{equation*}

\textbf{Flow Chart Interconnections}

mirfluctuation accepts as input data type either:

\textbullet{} \texttt{mirspectrum frame-decomposed} objects (i.e., spectrograms),

\textbullet{} \texttt{miraudio} objects, where the audio waveform can be segmented (using \texttt{mirsegment}). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 23 ms and half overlapping – can be changed using the \texttt{Frame} option.

\textbullet{} file names or the \texttt{Folder} keyword: same behavior than for \texttt{miraudio} objects.
**Estimation of Notes Onset Time**

Another way of determining the tempo is based on first the computation of an onset detection curve, showing the successive bursts of energy corresponding to the successive pulses. A peak picking is automatically performed on the onset detection curve, in order to show the estimated positions of the notes.

The onset detection curve can be computed in various ways:


- *mironsets(…, 'Filterbank', N)* specifies the number of channels for the filterbank decomposition (*mirfilterbank*): the default value being $N = 40$. $N = 0$ toggles off the filterbank decomposition.

- *mironsets(…, 'Sum', off)* toggles off the channel summation (*mirsum*) that is performed by default.
• *mironsets(..., 'SpectralFlux')* computes a spectral flux. Options related to *mirflux* can be passed here as well:
  - ‘Inc’ (toggled on by default here),
  - ‘Halfwave’ (toggled on by default here),
  - ‘Complex’ (toggled off by default as usual),
  - ‘Median’ (toggled on by default here, with same default parameters than in *mirflux*).

• *mironsets(..., 'Pitch')* computes a frame-decomposed autocorrelation function (*mirautocor*), of exactly same default characteristics than those returned by *mirpitch*, and subsequently computes the novelty curve of the resulting similatrix matrix.

*mironsets* accepts as input data type either:

• envelope curves (resulting from *mirenvelope*),

• any scalar object, in particular:
  - fluxes (resulting from *mirflux*)
  - novelty (resulting from *mirnovelty*)

• similatrix matrices (resulting from *mirsimatrix*): its novelty will be automatically computed;

• *miraudio* objects, where the audio waveform can be:
  - segmented (using *mirsegment*),
  - when the onset detection function is computed using an amplitude envelope (‘Envelope’ strategy), the audio waveform is by default first decomposed into channels (cf. the *Filterbank* option below),
  - decomposed into frames or not (using *mirframe*):
    - if the audio waveform is decomposed into frames, the onset curve will be based on the spectral flux;
    - if the audio waveform is not decomposed into frames, the default onset curve will be based on the envelope;

• file names or the ‘Folder’ keyword: same behavior than for *miraudio* objects,
• any other object: it is decomposed into frames (if not already decomposed) using the parameters specified by the ‘Frame’ option; the flux will be automatically computed by default, or the novelty (if the ‘Pitch’ option has been chosen).

**EXAMPLE**

Differentiating the envelope using the ‘Diff’ option highlights the difference of energy. By subsequently applying a halfwave rectification of the result (‘HalfwaveDiff’), bursts of energy are emphasized:

For the previous example (cf. figure above) we obtain now for the differentiated envelopes the following representation:

\[ o = \text{mironsets('ragtime', 'Diff', 'Sum', 'no', 'Filterbank', 5, 'HalfwaveDiff', 'Detect', 'no')} \]

And once the enveloped are summed:
**ONSET DETECTION**

- `mironsets(..., 'Detect', d)` specifies options related to the peak picking from the onset detection curve:
  - `d = 'Peaks'` (default choice): local maxima are chosen as onset positions;
  - `d = 'Valleys'`: local minima are chosen as onset positions;
  - `d = 0`, or `no`, or `off`: no peak picking is performed.

Options associated to the `mirpeaks` function can be specified as well. In particular:

- `mironsets(..., 'Contrast', c)` with default value here `c = .001`,
- `mironsets(..., 'SelectFirst', s)`, with default value here `s = 'on'`.

**ATTACK AND RELEASE**

The maxima of the onset detection curve show the positions of the note onsets, but more precisely the end of the attack phase. The `Attack` and `Release` options estimate the beginning of the attack phase and the end of the release phase of each note by searching for the local minimum before and after each peak.

- `mironsets(..., 'Attack', a)` (or `Attacks`) detects attack phases, using a gaussian envelope smoothing of order `a`. Default value when `Attack` is called: `a = 20`.

- `mironsets(..., 'Release', r)` (or `Releases`) detects release phases, using a gaussian envelope smoothing of order `r`. Default value when `Release` is called: `r = 20`.

---

*MIRtoolbox 1.1 User’s Manual*
**FRAME DECOMPOSITION**

If the onset detection curve is not a scalar object (i.e., basically, if the output is an envelope), it can be further decomposed into frames if the ‘Frame’ option has been specified, with default frame length 3 seconds and hop factor .1.
**mirtempo**

**DESCRIPTION**
Estimates the tempo by detecting periodicities from the onset detection curve.

**FLOWCHART INTERCONNECTIONS**

The tempo can be estimated in various ways:

- `mirtempo(,,, 'Autocor')` computes an autocorrelation function of the onset detection curve, using `mirautocor` (default choice). Options related to `mirautocor` can be specified:
  - `'Enhanced'` (toggled on by default here),
  - `'Resonance'` (set by default to ‘ToiviainenSnyder’),
  - `'NormalWindow'` (same default value).

- `mirtempo(,,, 'Spectrum')` computes a spectral decomposition of the onset detection curve, using `mirspectrum`. Options related to `mirspectrum` can be passed here as well:
  - `'ZeroPad'` (set by default here to 10000 samples),
  - `'Prod'` (same default as for `mirspectrum`),
  - `'Resonance'` either ‘ToiviainenSnyder’ (default value) or 0, ‘off’, or ‘no’.

- `mirtempo(,,, 'Autocor', 'Spectrum')` combines both strategies: the autocorrelation function is translated into the frequency domain in order to be compared to the spectrum curve, and the two curves are subsequently multiplied.

Then a peak picking is applied to the autocorrelation function or to the spectrum representation. The parameters of the peak picking can be tuned.

- `mirtempo(,,, 'Total', m)` selects not only the best tempo, but the $m$ best tempos.
- `mirtempo(,,, 'Min', mi)` indicates the lowest tempo taken into consideration, expressed in bpm. Default value: 40 bpm.
- `mirtempo(,,, 'Max', ma)` indicates the highest tempo taken into consideration, expressed in bpm. Default value: 200 bpm.
• \texttt{mirtempo}(..., \texttt{Contrast}, c) specifies the contrast factor for the peak picking. Default value: $c = 0.1$

\texttt{mirtempo} accepts as input data type either:

• \texttt{mirautocor} objects,

• \texttt{mirspectrument} objects,

• onset detection curve (resulting from \texttt{mironsets}), frame-decomposed or not, channel-decomposed or not,

• and all the input data accepted by \texttt{mironsets}.

Options related to \texttt{mironsets} can be specified:

• \texttt{Filterbank}', with same default value than for \texttt{mironsets},

• \texttt{Frame}', with same default value than for \texttt{mironsets},

• \texttt{mironsets(..., \texttt{Sum}, w)} specifies when to sum the channels. Possible values:
  • \texttt{w = 'Before'}: sum before the autocorrelation or spectrum computation.
  • \texttt{w = 'After'}: autocorrelation or spectrum computed for each band, and summed into a “summary”.

• \texttt{Envelope}': Options related to \texttt{mirenvelope} can be specified: \texttt{'HalfwaveCenter}, \texttt{'Diff} (toggled on by default here), \texttt{'HalfwaveDiff}, \texttt{'Center}, \texttt{'Smooth}, \texttt{'Sampling}', with same default value than for \texttt{mirenvelope}.

• \texttt{Flux}': Options related to \texttt{mirflux} can be specified: \texttt{'Inc}, \texttt{'Halfwave}, \texttt{'Complex'}, \texttt{'Median}', with same default value than for \texttt{mirflux}.

• \texttt{Pitch}'.

\texttt{mirtempo} can return several outputs:

1. the tempo itself (or set of tempi) and

2. the \texttt{mirspectrument} or \texttt{mirautocor} data, where is highlighted the (set of) peak(s) corresponding to the estimated tempo (or set of tempi).
EXAMPLE

The tempo estimation related to the *ragtime* example

\[
[t_{ac}] = \text{mirtempo}('\text{ragtime}')
\]

leads to a tempo \( t = 129.1832 \) bpm and to the following autocorrelation curve \( ac \):

![Envelope autocorrelation](image1)

The frame-decomposed tempo estimation related to the *czardas* example

\[
[t_{ac}] = \text{mirtempo}('\text{czardas}', 'Frame')
\]

leads to the following tempo curve \( t \):

![Tempo](image2)

and the following autocorrelation frame decomposition \( ac \):

![Envelope autocorrelation](image3)
3.3. Timbre

mirattacktime

Description
The attack phase detected using the ‘Attacks’ option in mironsets can offer some timbral characterizations. One simple way of describing the attack phase, proposed in mirattacktime, consists in estimating its temporal duration.

Flowchart Interconnections

mirattacktime accepts as input data type either:

- onset detection curves (resulting from mironsets), already including peaks or not,
- and all the input data accepted by mironsets.

mirattacktime can return several outputs:

1. the attack time itself and
2. the onset detection curve returned by mironsets, including the detected onsets.

Options
- mirattacktime(..., ‘Lin’) returns the duration in a linear scale (in seconds).
• `mirattacktime(..., 'Log')` returns the duration in a log scale (Krimphoff et al., 1994).
**mirattackslope**

**Description**
Another good description of the attack phase is related to its average slope.

**Flowchart Interconnections**

mirattackslope accepts as input data type either:

- onset detection curves (resulting from mironsets), already including peaks or not,
- and all the input data accepted by mironsets.

mirattackslope can return several outputs:

1. the attack slope itself and
2. the onset detection curve returned by mironsets, including the detected onsets.

**Options**

- **mirattackslope(x, `Diff`)** computes the slope as a ratio between the magnitude difference at the beginning and the ending of the attack period, and the corresponding time difference.

- **mirattackslope(x, `Gauss`)** computes the average of the slope, weighted by a gaussian curve that emphasizes values at the middle of the attack period (similar to Peeters, 2004).
**mirzerocross**

**WAVEFORM SIGN-CHANGE RATE**

A simple indicator of noisiness consists in counting the number of times the signal crosses the X-axis (or, in other words, changes sign).

*Flowchart Interconnections*

*mirzerocross* actually accepts any input data type (cf.).

**Options**

- *mirzerocross(…, 'Per', p)* precises the temporal reference for the rate computation. Possible values:
  - *p* = ‘Second’: number of sign-changes per second (Default).
  - *p* = ‘Sample’: number of sign-changes divided by the total number of samples. The ‘Second’ option returns a result equal to the one returned by the ‘Sample’ option multiplied by the sampling rate.

- *mirzerocross(…, 'Dir', d)* precises the definition of sign change. Possible values:
  - *d* = ‘One’: number of sign-changes from negative to positive only (or, equivalently, from positive to negative only). (Default)
• $d = 'Both'$: number of sign-changes in both ways. The $'Both'$ option returns a result equal to twice the one returned by the $'One'$ option.
**mirrolloff**

**HIGH-FREQUENCY ENERGY (I)**

One way to estimate the amount of high frequency in the signal consists in finding the frequency such that a certain fraction of the total energy is contained below that frequency. This ratio is fixed by default to .85 (following Tzanetakis and Cook, 2002), other have proposed .95 (Pohle, Pampalk and Widmer, 2005).

![Spectrum graph](image)

**FLOWCHART INTERCONNECTIONS**

```
miraudio  mirfilterbank

mirsegment

mirspectrum

mirrolloff
```

`mirrolloff` accepts either:

- *mirspectrum* objects, or
- *miraudio* objects (same as for *mirspectrum*),
- *file names* or the *Folder* keyword.

**OPTION**

`mirrolloff(..., 'Threshold', p)` specifies the energy threshold, as a percentage. Default value: .85
**mirbrightness**

**HIGH-FREQUENCY ENERGY (II)**

A dual method consists in fixing this time the cut-off frequency, and measuring the amount of energy above that frequency (Juslin, 2000). The result is expressed as a number between 0 and 1.

![Graph showing frequency spectrum and 53.96% of the energy at 1500 Hz.](image)

**Flowchart Interconnections**

![Flowchart diagram](image)

**mirbrightness** accepts either:

- **mirspecturm** objects, or
- **miraudio** objects (same as for **mirspecturm**),
- **file names** or the ‘Folder’ keyword.

**Options**

**mirrolloff(…, ‘CutOff’, f)** specifies the frequency cut-off, in Hz. Default value: 1500 Hz. The value 1000 Hz has been proposed in (Laukka, Juslin and Bresin, 2005), and the value of 3000 Hz has been proposed in (Juslin, 2000).
**mirmfcc**

**MEL-FREQUENCY CEPSRAL COEFFICIENTS**

MFCC offers a description of the spectral shape of the sound. We recall that the computation of the cepstrum followed the following scheme:

```
audio → mirspectrum. (Fourier transform) → Abs → Log → (“Inverse”) Fourier transform
```

The computation of mel-frequency cepstral coefficients is highly similar:

```
mirspectrum (Fourier transform) → Abs → Log → (Mel) → Discrete cosine transform
```

Here the frequency bands are positioned logarithmically (on the Mel scale) which approximates the human auditory system's response more closely than the linearly-spaced frequency bands. And the Fourier Transform is replaced by a Discrete Cosine Transform. A discrete cosine transform (DCT) is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. It has a strong “energy compaction” property: most of the signal information tends to be concentrated in a few low-frequency components of the DCT. That is why by default only the first 13 components are returned.

**FLOWCHART INTERCONNECTIONS**

```
miraudio
```

```
misegment
```

```
mirframe
```

```
mirfilterbank
```

```
mirspectrum
```

```
mirmfcc
```

*mirmfcc* accepts either:
• \textit{mirspectrum} objects, or
• \textit{miraudio} objects (same as for \textit{mirspectrum}),
• \textit{file names} or the \textit{Folder}’ keyword.

\textbf{OPTIONS}

• \textit{mirmfcc}(..., \textit{Rank}, N) computes the coefficients of rank(s) \(N\). The default value is \(N = 1:13\).

• If the output is frame-decomposed, showing the temporal evolution of the MFCC along the successive frames, the temporal differentiation can be computed:
  
  • \textit{mirmfcc}(..., \textit{Delta}, d) performs temporal differentiations of order \(d\) of the coefficients, also called delta-MFCC (for \(d = 1\)) or delta-delta-MFCC (for \(d = 2\)). By default, \(d = 1\).

  • \textit{mirmfcc}(..., \textit{Radius}, r) specifies, for each frame, the number of successive and previous neighbouring frames taken into consideration for the least-square approximation used for the derivation. Usually the radius is equal to 1 or 2. Default value: \(r = 2\).
**mirroughness**

**Sensory Dissonance**

Plomp and Levelt (1965) has proposed an estimation of the sensory dissonance, or roughness, related to the beating phenomenon whenever pair of sinusoids are closed in frequency. The authors propose as a result an estimation of roughness depending on the frequency ratio of each pair of sinusoids represented as follows:

An estimation of the total roughness is available in `mirroughness` by computing the peaks of the spectrum, and taking the average of all the dissonance between all possible pairs of peaks (Sethares, 1998).

**Flowchart Interconnections**

The ‘Contrast’ parameter associated to `mirpeaks` can be specified, and is set by default to .01

`mirroughness` accepts either:

- **mirspecrum** objects, where peaks have already been picked or not,

- **miraudio** objects: same as for `mirspecrum`, except that a **frame decomposition** is automatically performed, with default frame length 50 ms and half overlapping. This default frame decomposition is due to the fact that roughness can only be associated to a spectral representation association to a short-term sound excerpt: there is no sensory dissonance provoked by a pair of sinusoid significantly distant in time.

- **file names** or the ‘Folder’ keyword.
mirroughness can return several outputs:

1. the roughness value itself and

2. the spectral representation (output of mirspectrum) showing the picked peaks (returned by mirpeaks).
**mirregularity**

**Spectral Peaks Variability**

The irregularity of a spectrum is the degree of variation of the successive peaks of the spectrum.

**Flowchart Interconnections**

The ‘Contrast’ parameter associated to `mirpeaks` can be specified, and is set by default to .01.

`mirregularity` accepts either:

- `mirspectrumb` objects, where peaks have already been picked or not,
- `miraudio` objects (same as for `mirspectrumb`),
- file names or the ‘Folder’ keyword.

**Options**

- `mirregularity(..., Jensen)` is based on (Jensen, 1999), where the irregularity is the sum of the square of the difference in amplitude between adjoining partials. (Default approach)

\[
\left( \sum_{k=1}^{N} (a_k - a_{k+1})^2 \right)/ \sum_{k=1}^{N} a_k^2
\]

- `mirregularity(..., Krimphoff)` is based on (Krimphoff et al., 1994), where the irregularity is the sum of the amplitude minus the mean of the preceding, same and next amplitude.

\[
\sum_{k=2}^{N-1} \left| a_k - \frac{a_{k-1} + a_k + a_{k+1}}{3} \right|
\]
3.4. Pitch

**mirpitch**

**Pitch Estimation**
Extract pitches and return their frequencies in Hz.

**Flowchart Interconnections**

The pitch content can be estimated in various ways:

- `mirpitch(..., 'Autocor')` computes an autocorrelation function of the audio waveform, using `mirautocor`. This is the default strategy. Options related to `mirautocor` can be specified:
  - `'Enhanced'`, toggled on by default here,
  - `'Compress'`, set by default to .5,
  - filterbank configuration can be specified: either `'2Channels'`, `'Gammatone'` or `'NoFilter-bank'`.

- `mirpitch(..., 'AutocorSpectrum')` computes the autocorrelation (`mirautocor`) of the FFT spectrum (`mirspectrum`).

- `mirpitch(..., 'Cepstrum')` computes the cepstrum (`mircepstrum`).

- These methods can be combined. In this case, the resulting representations (autocorrelation function or cepstral representations) are all expressed in the frequency domain and multiplied altogether.

Then a peak picking is applied to the autocorrelation function or to the cepstral representation. The parameters of the peak picking can be tuned.

- `mirpitch(..., 'Total', m)` selects only the m best pitches.

- `mirpitch(..., 'Mono')` only select the best pitch, corresponding hence to `mirpitch(..., 'Total', 1)`.
• \textit{mirpitch}(\ldots, \textit{Min}, \textit{mi}) indicates the lowest pitch taken into consideration, in Hz. Default value: 75 Hz, following a convention in the \textit{Praat} software (Boersma & Weenink, 2005).

• \textit{mirpitch}(\ldots, \textit{Max}, \textit{ma}) indicates the highest tempo taken into consideration, expressed in bpm. Default value: 2400 Hz, because there seem to be some problems with higher frequency, due probably to the absence of pre-whitening in our implementation of Tolonen and Karjalainen autocorrelation approach (used by default).

• \textit{mirpitch}(\ldots, \textit{Contrast}, \epsilon) specifies the contrast factor for the peak picking. Default value: \(\epsilon = 0.1\).

\textit{mirpitch} accepts as input data type either:

• \textit{mirautocor} objects,

• \textit{mircepstrum} objects,

• \textit{mirspectrum} objects,

• \textit{miraudio} objects, where the audio waveform can be:
  • segmented (using \textit{mirsegment}),
  • when pitch is estimated by autocorrelating the audio waveform ("Autocor" strategy), the audio waveform is be default first decomposed into channels (cf. the \textit{Filterbank} option below),
  • decomposed into frames or not, using \textit{mirframe}, or the "Frame" option, where the default frame length is .464 ms and the default hop length is 10 ms (Tolonen & Karjalainen 2000);

• file names or the "Folder" keyword: same behavior than for \textit{miraudio} objects.

\textit{mirpitch} can return several outputs:

1. the pitch frequencies themselves, and

2. the \textit{mirautocor} or \textit{mircepstrum} data, where is highlighted the (set of) peak(s) corresponding to the estimated pitch (or set of pitches).

\textsc{Options}

• \textit{mirpitch}(\ldots, \textit{Median}, \textit{l}) performs a median filtering of the pitch curve. The length of the median filter is given by \(l\) (in s.). Its default value is .1 s. The median filtering can only be applied
to mono-pitch curve. If several pitches were extracted in each frame, a mono-pitch curve is first computed by selecting the best peak of each successive frame.

- `mirpitch(..., 'Stable', th, n)` remove pitch values when the difference (or more precisely absolute logarithmic quotient) with the `n` precedent frames exceeds the threshold `th`.
  - if `th` is not specified, the default value .1 is used.
  - if `n` is not specified, the default value 3 is used.

- `mirpitch(..., 'Reso', 'SemiTone')` removes peaks whose distance to one or several higher peaks is lower than a given threshold $2^{\frac{1}{12}}$ (corresponding to a semitone).

- `mirpitch(..., 'Tolonen')` implements (part of) the model proposed in (Tolonen & Karjalainen, 2000). It is equivalent to

\[
\text{mirpitch}(..., 'Enhanced', 2:10, 'Generalized', .67, '2Channels')
\]

**EXAMPLE**

\[
[p ac] = \text{mirpitch('ragtime', 'Frame')}
\]
**mirinharmonicity**

**PARTIALS NON-MULTIPLE OF FUNDAMENTALS**
mirinharmonicity(\(x\)) estimates the inharmonicity, i.e., the amount of partials that are not multiples of the fundamental frequency, as a value between 0 and 1. For that purpose, we use a simple function estimating the inharmonicity of each frequency given the fundamental frequency \(f_0\):

\[
\begin{align*}
&0 \\
&f_0 \\
&2f_0 \\
&3f_0 \\
&4f_0 \\
&5f_0
\end{align*}
\]

**WARNING:**
1. This simple model presupposes that there is only one fundamental frequency.
2. The model should be improved in future versions of MIRtoolbox.

**FLOWCHART INTERCONNECTIONS**

*mirinharmonicity* accepts as main input either:

- *mirspectrum* objects,
- *miraudio* objects (same as for *mirspectrum*),

**WARNING:** The ‘Frame’ option does not work correctly because it is performed first, before calling *mirspectrum* and *mirpitch*: the filterbank decomposition used in *mirpitch* is therefore computed after the frame decomposition, which is not recommended. This problem should be solved in later version of MIRtoolbox.

- *file names* or the ‘Folder’ keyword.
mirinharmonicity can return several outputs:

1. the inharmonicity rate itself,
2. the mirspectrum data, and
3. the fundamental frequency \( f_0 \).

**OPTION**

`mirinharmonicity(...) \( f_0 \)`, \( f \)` bases the computation of the inharmonicity on the fundamental frequency indicated by \( f \). The frequency data should be represented in the standard format (cf.). By default, the fundamental frequency is computed using the command:

\[
f = mirpitch(...) \text{, } \text{'Mono'}}
\]
3.5. Tonality

Energy Distribution Along Pitches

The chromagram, also called Harmonic Pitch Class Profile, shows the distribution of energy along the pitches or pitch classes.

- First the spectrum is computed in the logarithmic scale, with selection of the 20 highest dB, and restriction to the frequency range between 100 and 5000 Hz, as proposed in (Gomez, 2006), and normalization of the audio waveform before computation of the FFT, and a final frequency resolution of 5 Hz:

\[ s = \text{mirspectrum}(\ldots, \text{`dB'}, 20, \text{`Min'}, 100, \text{`Max'}, 5000, \text{`NormalInput'}, \text{`Res'}, 5) \]

- we can redistribute the energy along the different pitches:

\[ c = \text{mirchromagram}(s, \text{`Wrap'}, \text{`no'}) \]

- and then compute the distribution of energy along the 12 pitch classes:

\[ c = \text{mirchromagram}(c, \text{`Wrap'}, \text{`yes'}) \]
The simple algorithm proposed in \textit{mirchromagram} will be progressively refined in future versions of MIRtoolbox.

\textbf{Flowchart Interconnections}

For the moment, only the ‘dB’ threshold value used in \textit{mirspectrogram} can be tuned directly in \textit{mirchromagram}.

\textit{mirchromagram} accepts either:

\begin{itemize}
  \item \textit{mirspectrogram} objects,
  \item \textit{miraudio} objects, where the audio waveform can be segmented (using \textit{mirsegment}), decomposed into channels (using \textit{mirfilterbank}), and/or decomposed into frames (using \textit{mirframe} or the ‘Frame’ option, with by default a frame length of 100 ms and a hop factor of \(1.125\)),
  \item file names or the ‘Folder’ keyword.
\end{itemize}

\textbf{Options}

\begin{itemize}
  \item \(c = \text{mirchromagram}(..., \text{'Triangle'})\) weights the contribution of each frequency with respect to the distance with the actual frequency of the corresponding chroma.
  \item \(c = \text{mirchromagram}(..., \text{'Weight'}, o)\) specifies the relative radius of the weighting window, with respect to the distance between frequencies of successive chromas.
\end{itemize}
• $o = 1$: each window begins at the centers of the previous one.
• $o = .5$: each window begins at the end of the previous one. (default value)

**Post-processing Operations**

• $c = \text{mirchromagram}(...,'\text{Wrap}', w)$ specifies whether the chromagram is wrapped or not.
  • $w = 'yes'$: groups all the pitches belonging to same pitch classes (default value)
  • $w = 'no'$: pitches are considered as absolute values.

• $c = \text{mirchromagram}(...,'\text{Center}')$ centers the result.

• $c = \text{mirchromagram}(...,'\text{Normal}')$ normalizes the result.

• $c = \text{mirchromagram}(...,'\text{Pitch}', p)$ specifies how to label chromas in the figures.
  • $p = 'yes'$: chromas are labeled using pitch names (default)
  • $p = 'no'$: chromas are labeled using MIDI pitch numbers.

**Example**

![Chromagram example](image-url)
\textit{mirkeystrength}

**Probability of Key Candidates**

\textit{mirkeystrength} computes the key strength, i.e., the probability associated with each possible key candidate, through a cross-correlation of the chromagram returned by \textit{mirchromagram}, wrapped and normalized, with similar profiles representing all the possible tonality candidates (Krumhansl, 1990; Gomez, 2006).

![Cross-Correlations](image)

The resulting graph indicate the cross-correlation score for each different tonality candidate.

![Flowchart Interconnections](image)

For the moment, only the ‘Weight’ and ‘Triangle’ options used in \textit{mirchromagram} can be tuned directly in \textit{mirkeystrength}. 

\textit{MIRtoolbox 1.1 User’s Manual}
mirkeystrength accepts either:

- *mirchromagram* objects,
- *mirspectrumin* objects,
- *miraudio* objects (same as for *mirchromagram*),
- *file names* or the *Folder* keyword.

mirkeystrength can return several outputs:

1. the key strength itself, and
2. the *mirchromagram* data.

**EXAMPLE**

\[
\text{mirkeystrength('ragtime', 'Frame')}\
\]
mirkey

DESCRIPTION
Gives a broad estimation of tonal center positions and their respective clarity.

FLOWCHART INTERCONNECTIONS

It consists simply of a peak picking in the mirkeystrength curve(s). Two options of mirpeaks are accessible from mirkey:

- ‘Total’, set to 1
- ‘Contrast’, set to .1

For the moment, only the ‘Weight’ and ‘Triangle’ options used in mirchromagram can be changed directly in mirkeystrength.

mirkey accepts either:

- mirkeystrength objects, where peaks have been already extracted or not,
- mirchromagram objects,
- mirspectrum objects,
- miraudio objects, where the audio waveform can be segmented (using mirsegment), decomposed into channels (using mirfilterbank), and/or decomposed into frames (using mirframe or the ‘Frame’ option, with by default a frame length of 1 s and half overlapping),
- file names or the ‘Folder’ keyword.

mirkey can return several outputs:

1. the best key(s), i.e., the peak abscissa(e);
2. the key clarity: i.e., the key strength associated to the best key(s), i.e., the peak ordinate(s);
3. the `mirkeystrength` data including the picked peaks (`mirpeaks`).

**Example**

\[
[kc] = \text{mirkey('ragtime', 'Frame')}
\]
**mirmode**

**Description**
Estimate the modality, i.e. major vs. minor, returned as a numerical value: the more it is higher than 0, the more major the given excerpt is supposed to be, the more the value is lower than 0, the more minor the excerpt should be.

**Flowchart Interconnections**

*mirmode* accepts either:

- *mirkeystrength* objects, where peaks have been already extracted or not,
- *mirchromagram* objects,
- *mirspectrum* objects,
- *miraudio* objects (same as for *mirspectrum*) or
- file names or the ‘Folder’ keyword.

*mirmode* can return several outputs:

1. modality itself, and
2. the *mirkeystrength* result used for the computation of modality.

**Strategies**
- *mirkeystrength*(..., ‘Best’) computes the key strength difference between the best major key and the best minor key. (default choice)
mirkeystrength(...) computes the key strength difference between all the major keys and their relative minor keys.
**mirkeysom**

**Description**
Projects the chromagram (normalized) into a self-organizing map trained with the Krumhansl-Kessler profiles (modified for chromagrams) (Toiviainen and Krumhansl, 2003; Krumhansl, 1990).

The result is displayed as a pseudo-color map, where colors correspond to Pearson correlation values. In case of frame decomposition, the projection maps are shown one after the other in an animated figure.

**Flowchart Interconnections**

```
mirkeysom
```

`mirkeysom` accepts either:

- **mirchromagram** objects,
- **mirspectrum** objects,
- **miraudio** objects, where the audio waveform can be segmented (using `mirsegment`), decomposed into channels (using `mirfilterbank`), and/or decomposed into frames (using `mirframe` or the 'Frame' option, with by default a frame length of 1 s and half overlapping),
- **file names** or the 'Folder' keyword.

**Example**

`mirkeysom('ragtime')`
**mirtonalcentroid**

**Description**
Calculates the 6-dimensional tonal centroid vector from the chromagram. It corresponds to a projection of the chords along circles of fifths, of minor thirds, and of major thirds (Harte and Sandler, 2006).

**Flowchart Interconnections**

mirtonalcentroid accepts either:

- **mirchromagram** objects,
- **mirspectrum** objects,
- **miraudio** objects, where the audio waveform can be segmented (using **mirsegment**), decomposed into channels (using **mirfilterbank**), and/or decomposed into frames (using **mirframe** or the 'Frame' option, with by default a frame length of 1 s and a hop factor of .1),
- **file names** or the 'Folder' keyword.

mirtonalcentroid can return several outputs:

1. the tonal centroid itself, and
2. the **mirchromagram** data.
**mirhcdf**

**DESCRIPTION**
The Harmonic Change Detection Function (HCDF) is the flux of the tonal centroid (Harte and Sandler, 2006).

**FLOWCHART INTERCONNECTIONS**

`mirhcdf` accepts either:

- **mirtonalcentroid** frame-decomposed objects,
- **mirchromagram** frame-decomposed objects,
- **mirspectrum** frame-decomposed objects,
- **miraudio** objects, where the audio waveform can be segmented (using **mirsegment**), decomposed into channels (using **mirfilterbank**). If not decomposed yet, it is decomposed into frames (using the 'Frame' option, with by default a frame length of 1 s and half overlapping),
- **file names** or the 'Folder' keyword.

**mirtonalcentroid** can return several outputs:

1. the tonal centroid itself, and
2. the **mirchromagram** data.
\textit{mirsegment}(\ldots, \textit{HCDF})

Peak detection applied to the HCDF returns the temporal position of tonal discontinuities that can be used for the actual segmentation of the audio sequence.

\textbf{Flowchart Interconnections}

\textit{mirsegment} accepts uniquely as main input a \textit{miraudio} objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, \textbf{file names} or the \textit{Folder} keyword can be used as well.

\textit{mirsegment}(\ldots, \textit{HCDF}) can return several outputs:

1. the segmented audio waveform itself,
2. the HCDF (\textit{mirdc}) after peak picking (\textit{mirpeaks}),
3. the tonal centroid (\textit{mirtonalcentroid}), and
4. the chromagram (\textit{mirchromagram}).
4. POST-PROCESSING

4.1. Structure and form

More elaborate tools have also been implemented that can carry out higher-level analyses and transformations. In particular, audio files can be automatically segmented into a series of homogeneous sections, through the estimation of temporal discontinuities along diverse alternative features such as timbre in particular (Foote and Cooper, 2003).

**mirsimatrix**

**Description**

A similarity matrix shows the similarity between all possible pairs of frames from the input data.
**Flowchart Interconnections**

`mirsimatrix` usually accepts either:

- **`mirspectrum`** frame-decomposed objects.
- **`miraudio`** objects: in this case, the (dis)similarity matrix will be based on the spectrogram (**`mirspectrum`**). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 50 ms and no overlapping – can be changed using the **Frame** option.

If the audio waveform is segmented (with **`mirsegment`**), the similarities are not computed between frames but on the contrary between segments, using Kullback-Leibler distance (Foote and Cooper, 2003).

- **file names** or the **Folder** keyword: same behavior than for **`miraudio`** objects;
- **`mirautocor`** frame-decomposed objects;
- **`mircepstrum`** frame-decomposed objects;
- **`mirmfcc`** frame-decomposed objects;
- **`mirchromagram`** frame-decomposed objects;
- **`mirkeystrength`** frame-decomposed objects.

**Options**

- `mirsimatrix(..., 'Distance', f)` specifies the name of a distance function, from those proposed in the **Statistics Toolbox** (help `pdist`). default value: `f = 'cosine'`
• `mirsimatrix(..., 'Width', w)` specifies the size of the diagonal bandwidth, in samples, outside which the dissimilarity will not be computed. If \( w = \text{inf} \) (default value), all the matrix will be computed.

• `mirsimatrix(..., 'Dissimilarity')` return the dissimilarity matrix, which is the intermediary result before the computation of the similarity matrix. It shows the distance between each possible frame.

![Dissimilarity matrix](image)

• `mirsimatrix(..., 'Similarity', f)` indicates the function \( f \) specifying the way the distance values in the dissimilarity matrix are transformed into the values of the similarity matrix. default value: \( f = '\text{exponential}' \) corresponding to

\[
f(x) = \exp(-x)
\]
• `mirsimatrix(..., 'Horizontal')` rotates the matrix $45^\circ$ in order to make the first diagonal horizontal, and to restrict on the diagonal bandwidth only.
**mirnovelty**

**Novelty Curve**
Convolution along the main diagonal of the similarity matrix using a Gaussian checkerboard kernel yields a novelty curve that indicates the temporal locations of significant textural changes.

![Image of novelty curve](image)

**Flowchart Interconnections**

Some parameters related to `mirsimatrix` are accessible in `mirnovelty`:

- **‘Distance’,**
- **‘Similarity’** and
- **‘Width’** (can also be called ‘**KernelSize**’), with the default value here 128 samples.

`mirnovelty` usually accepts either:
• **mirsimatrix** objects,
• **mirspectr** frame-decomposed objects,
• **miraudio** objects (same as for **mirsimatrix**).
• **file names** or the ‘Folder’ keyword: same behavior than for **miraudio** objects;
• **mirautocor** frame-decomposed objects;
• **mircepstrum** frame-decomposed objects;
• **mirmfcc** frame-decomposed objects;
• **mirchromagram** frame-decomposed objects;
• **mirkeystrength** frame-decomposed objects.

**mirnovelty** can return several outputs:

1. the novelty curve itself, and
2. the similarity matrix (**mirsimatrix**), horizontal.

**POST-PROCESSING OPERATION**

• **mirnovelty**(…, ‘Normal’, n) toggles on/off the normalization of the novelty curve between the values 0 and 1. Toggled on by default.

**EXAMPLE**

Novelty curve computed using increasing kernel size ‘KernelSize’:
**mirsegment(..., ‘Novelty’)**

Peak detection applied to the novelty curve returns the temporal position of feature discontinuities that can be used for the actual segmentation of the audio sequence.

The ‘Novelty’ keyword is actually not necessary, as this strategy is chosen by default in mirsegment.

**Flowchart Interconnections**

Some parameters related to **mirnovelty** are accessible in **mirsegment**: ‘Distance’, ‘Similarity’ and ‘KernelSize’.

The choice of the feature used for the similarity matrix computation can be specified:

- **mirsegment(..., ‘Spectrum’)** will compute a **mirspectrum**, where some parameters can be specified: ‘Min’, ‘Max’, ‘Normal’, ‘Window’, ‘Prod’. The default frame length is 50 ms and no overlapping.

- **mirsegment(..., ‘MFCC’)** will compute a **mirmfcc**, where the ‘Rank’ parameter can be specified. Same default frame parameters than for ‘Spectrum’.

- **mirsegment(..., ‘KeyStrength’)** will compute a **mirkeystrength**, where the ‘Weight’ parameter can be specified. The default frame length is 500 ms and a hop factor of .2

- **mirsegment(..., ‘Pitch’)** will compute a **mirpitch** operation and use its second output, i.e., the **mirautocor**, for the computation of the similarity matrix. The default frame length is 50 ms and a hop factor of .01

- If no feature is specified, the default feature used in **mirsimatrix** will be chosen, i.e., the spectrum (**mirspectrum**).

The default frame parameters can be changed using the ‘WinLength’ option (in second) and the ‘Hop’ option (a value between 0 and 1).
mirsegment accepts uniquely as main input a miraudio objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, file names or the 'Folder' keyword can be used as well.

`mirsegment(..., 'Novelty')` can return several outputs:

1. the segmented audio waveform itself,
2. the novelty curve (`mirnovelty`) after peak picking (`mirpeaks`),
3. the similarity matrix (`mirsimatrix`), and
4. the features entered into the `mirsimatrix` operator.
4.2. Statistics

\texttt{mirstat}.

\textbf{DESCRIPTION}

\texttt{mirstat} can be applied to any object and will return its statistics in a structure array.

**WARNING:** \texttt{mirstat} does not work with multi-channel data.

- If the object is frame-decomposed, the fields of the output are:
  - \textit{Mean}: the average along frames
  - \textit{Std}: the standard deviation along frames
  - \textit{Slope}: the slope of the trend along frames, in a least square sense
  - \textit{PeriodFreq}: the frequency of the main periodicity detected in the frame-by-frame evolution of the values, estimated through autocorrelation. If no periodicity is detected, \texttt{NaN} is returned.
  - \textit{PeriodAmp}: the amplitude of that main periodicity
  - \textit{PeriodEntropy}: the entropy of the autocorrelation function

- If the object is not frame-decomposed, the data itself is returned directly in the single field \textit{Mean}.

\textbf{N A N - F I L T E R}

\texttt{mirstat} automatically discard any \texttt{NaN} value contained in the input data.

\textbf{M A N A G E M E N T O F S T R U C T U R E A R R A Y S}

If the input is already a structure array, the output will follows the same field decomposition (and all subfields as well) of the structure array, and will replace each final node with its corresponding \texttt{mirstat} structure array result.
**mirhisto**

**DESCRIPTION**

`mirhisto` can be applied to any object and will return its corresponding histogram. The data is binned into equally spaced containers.

![Histogram of Audio waveform](image)

**OPTIONS**

- `mirhisto(..., 'Number', n)` specifies the number of containers. Default value: `n = 10`.
- `mirhisto(..., 'Ampli')` adds the amplitude of the elements, instead of simply counting them.
\textit{mirzerocross}

\textbf{DESCRIPTION}

\textit{mirzerocross} counts the number of times the signal crosses the X-axis (or, in other words, changes sign).

This function has already defined in as: applied directly to audio waveform, \textit{mirzerocross} is an indicator of noisiness.

But actually \textit{mirzerocross} accepts any input data type.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram.png}
\caption{Diagram showing the \textit{mirzerocross} function with signal crossings marked.}
\end{figure}

\textbf{OPTIONS}

- \textit{mirzerocross(..., 'Per', \textit{p})} precises the temporal reference for the rate computation. Possible values:
  - \textit{p = 'Second'}: number of sign-changes per second (Default).
  - \textit{p = 'Sample'}: number of sign-changes divided by the total number of samples. The 'Second' option returns a result equal to the one returned by the 'Sample' option multiplied by the sampling rate.

- \textit{mirzerocross(..., 'Dir', \textit{d})} precises the definition of sign change. Possible values:
  - \textit{d = 'One'}: number of sign-changes from negative to positive only (or, equivalently, from positive to negative only). (Default)
  - \textit{d = 'Both'}: number of sign-changes in both ways. The 'Both' option returns a result equal to twice the one returned by the 'One' option.
**mircentroid**

**Description**

*mircentroid* returns the centroid of the data.

**Explanation**

An important and useful description of the shape of a distribution can be obtained through the use of its moments. The first moment, called the mean, is the geometric center (centroid) of the distribution and is a measure of central tendency for the random variable.

\[ \mu_1 = \int x f(x) \, dx \]

**Inputs**

Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the centroid is computed on the spectrum (spectral centroid).

If the input is a series of peak lobes produced by *mirpeaks(..., Extract)*, the centroid will be computed for each of these lobes separately.

**Option**

- When the input contains peaks (using *mirpeaks*), *mircentroid(..., Peaks, i)* will compute the centroids of the distribution of peaks. The argument *i* accepts two arguments:
  - *i = 'NoInterpol'*: the centroid is computed on the non-interpolated peaks (default choice),
  - *i = 'Interpol'*: the centroid is computed on the interpolated peaks (cf. ‘Interpol’ option in *mirpeaks*).
**mirspread**

**Description**
mirspread returns the standard deviation of the data.

**Explanation**
The second central moment, called the variance, is usually given the symbol sigma squared and is defined as:

\[ \sigma^2 = \mu_2 = \int (x - \mu_1)^2 f(x) dx \]

Being the squared deviation of the random variable from its mean value, the variance is always positive and is a measure of the dispersion or spread of the distribution. The square root of the variance is called the standard deviation, and is more useful in describing the nature of the distribution since it has the same units as the random variable. (Koch)

**Inputs**
Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the spread is computed on the spectrum (spectral spread).

If the input is a series of peak lobes produced by mirpeaks(..., ‘Extract’), the spread will be computed for each of these lobes separately.
mirskewness

DESCRIPTION
mirspread returns the coefficient of skewness of the data.

EXPLANATION
The third central moment is called the skewness and is a measure of the symmetry of the distribution. The skewness can have a positive value in which case the distribution is said to be positively skewed with a few values much larger than the mean and therefore a long tail to the right. A negatively skewed distribution has a longer tail to the left. A symmetrical distribution has a skewness of zero. (Koch)

\[ \mu_3 = \int (x - \mu_1)^3 f(x) \, dx \]

The coefficient of skewness is the ratio of the skewness to the standard deviation raised to the third power.

\[ \frac{\mu_3}{\sigma^3} \]

The coefficient of skewness has more convenient units than does the skewness and often ranges from -3.0 to 3.0 for data from natural systems. Again, a symmetrical distribution has a coefficient of skewness of zero. A positive coefficient of skewness often indicates that the distribution exhibits a concentration of mass toward the left and a long tail to the right whereas a negative value generally indicates the opposite. (Koch)

\[ \mu_1 \]

INPUTS
Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the skewness is computed on the spectrum (spectral skewness).
If the input is a series of peak lobes produced by `mirpeaks(..., 'Extract')`, the skewness will be computed for each of these lobes separately.
**mirkurtosis**

**Description**

*mirkurtosis* returns the (excess) *kurtosis*, of the data.

**Explanation**

The *fourth standardized moment* is defined as,

\[
\frac{\mu_4}{\sigma^4}
\]

*Kurtosis* is more commonly defined as the fourth cumulant divided by the square of the variance of the probability distribution, equivalent to:

\[
\frac{\mu_4}{\sigma^4} - 3
\]

which is known as *excess kurtosis*. The “minus 3” at the end of this formula is often explained as a correction to make the kurtosis of the normal distribution equal to zero. Another reason can be seen by looking at the formula for the kurtosis of the sum of random variables. Because of the use of the cumulant, if \( Y \) is the sum of \( n \) independent random variables, all with the same distribution as \( X \), then \( \text{Kurt}[Y] = \text{Kurt}[X]/n \), while the formula would be more complicated if kurtosis were simply defined as *fourth standardized moment*. (Wikipedia)

![Kurtosis Types](image)

**Inputs**

Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the kurtosis is computed on the spectrum (spectral kurtosis).

If the input is a series of peak lobes produced by *mirpeaks(…, ‘Extract*)*, the kurtosis will be computed for each of these lobes separately.
**mirflatness**

**DESCRIPTION**
mirflatness returns the flatness of the data.

**EXPLANATION**
The flatness indicates whether the distribution is smooth or spiky, and results from the simple ratio between the geometric mean and the arithmetic mean:

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

**INPUTS**
Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the flatness is computed on the spectrum (spectral flatness).

**mirentropy**

**DESCRIPTION**
mirentropy returns the relative entropy of the curve.

**INPUTS**
Any data can be used as input.

If the input is an audio waveform, a file name, or the ‘Folder’ keyword, the entropy is computed on the spectrum (spectral entropy).
mirfeatures

Description
mirfeatures computes a large set of features, and returns them in a structure array organized as follows:

- in a dynamics field,
  - a rms field: the frame-based RMS (mirrms);
- in a rhythm field,
  - a fluctuation field, containing:
    - a peak field: a fluctuation summary (mirfluctuation) and its highest peak (mirpeak),
    - a centroid field: the centroid (mircentroid) of the fluctuation summary;
  - a tempo field: a frame-based tempo estimation (mirtempo),
- an attack field, containing:
  - a time field: the attack times (mirattacktime) of the onsets (mironsets),
  - a slope field: the attack slopes (mirattackslope) of the onsets,
- in a timbre field,
  - a zerocross field: the frame-decomposed zero-crossing rate (mirzerocross),
  - a centroid field: the frame-decomposed spectral centroid (mircentroid),
  - a brightness field: the frame-decomposed brightness (mirbrightness),
  - a spread field: the frame-decomposed spectral spread (mirspread),
  - a skewness field: the frame-decomposed spectral skewness (mirskewness),
  - a kurtosis field: the frame-decomposed spectral kurtosis (mirkurtosis),
  - a rolloff95 field: the frame-decomposed roll-off (mirrolloff), using a 95 % threshold,
  - a rolloff85 field: the frame-decomposed roll-off, using a 85 % threshold,
  - a spectentropy field: the frame-decomposed spectral entropy (mirentropy),
  - a flatness field: the frame-decomposed spectral flatness (mirflatness),
  - a roughness field: the frame-decomposed roughness (mirroughness),
• an irregularity field: the frame-decomposed irregularity (mirirregularity),
• an inharmonicity field: the frame-decomposed inharmonicity (mirinharmonicity),
• an mfcc field: the frame-decomposed MFCCs (mirmfcc),
• a dmfcc field: the frame-decomposed delta-MFCCs,
• a ddmfcc field: the frame-decomposed delta-delta-MFCCs,
• a lowenergy field: the frame-decomposed low energy rate (mirlowenergy),
• a spectralflux field: the frame-decomposed spectral flux (mirflux);
• in a pitch field,
  • a salient field: the frame-decomposed pitches (mirpitch),
  • a chromagram field, containing:
    • a peak field: an unwrapped chromagram (mirchromagram) and its highest peak,
    • a centroid field: the centroid of the chromagram;
• in a tonal field,
  • a keyclarity field: the frame-decomposed key clarity (second output of mirkey),
  • a mode field: the frame-decomposed mode (mirmode),
  • a hcdf field: the frame-decomposed HCDF (mirhcdf).

O P T I O N
• mirfeatures(..., ‘Stat’) returns the statistics of the features instead of the complete features themselves.
4.3. Classification

**mirclassify**

**DESCRIPTION**

`mirclassify(test, features_test, train, features_train)` classifies the audio sequence(s) contained in the audio object `test`, along the analytic feature(s) `features_test`, following the supervised learning of a training set defined by the audio object `train` and the corresponding analytic feature(s) `features_train`. Multiple analytic features have to be grouped into one array of cells.

Requires the `Netlab` toolbox.

**EXAMPLE**

```plaintext
mirclassify(test, mfcc(test), train, mfcc(train))

mirclassify(test, {mfcc(test), centroid(test)}, train, {mfcc(train), centroid(train)})
```

**OPTIONAL ARGUMENTS**

- `mirclassify(..., 'Nearest')` uses the minimum distance strategy. (by default)

- `mirclassify(..., 'Nearest', k)` uses the $k$-nearest-neighbour strategy. Default value: $k = 1$, corresponding to the minimum distance strategy.

- `mirclassify(..., 'GMM', ng)` uses a gaussian mixture model. Each class is modeled by at most $ng$ gaussians. Default value: $ng = 1$.

  - Additionally, the type of mixture model can be specified, using the set of value proposed in Netlab’s `gmm` function: i.e., ‘spherical’, ‘diag’, ‘full’ (default value) and ‘ppca’. (cf. help `gmm`)
**mircluster**

**DESCRIPTION**

mircluster(a, f) clusters the segments in the audio sequence(s) contained in the audio object a, along the analytic feature(s) f, using the k-means strategy. Multiple analytic features have to be grouped into one array of cells.

**EXAMPLE**

```plaintext
sg = mirsegment(a);

mircluster(sg, mirmfcc(sg))

mircluster(sg, {mirmfcc(sg), mircentroid(sg)})
```

**OPTIONAL ARGUMENT:**

- `mircluster(..., n)` indicates the maximal number of clusters. Default value: `n = 2`.

- `mircluster(..., 'Runs', r)` indicates the maximal number of runs. Default value: `r = 5`.
4.4. Exportation

mirgetdata

DESCRIPTION

mirgetdata return the data contained in the input in a structure that can be used for further computation outside MIRtoolbox.

OUTPUT

If the input is based on one non-segmented audio sequence, the result is returned as a matrix. The columns of the matrix usually correspond to the successive frames of the audio signal. The third dimension of the matrix corresponds to the different channels of a filterbank.

If the input corresponds to a set of audio sequences, and if each sequence has same number of frames, the corresponding resulting matrices are concatenated columnwise one after the other. If the number of rows of the elementary matrices varies, the missing values are replaced by NaN in the final matrix. On the contrary, if the number of columns (i.e., frames) differs, then the result remains a cell array of matrices.

Idem if the input corresponds to one or several segmented audio sequence(s).

If the input is a key strength curve, the fourth dimension distinguishes between major and minor keys. i.e.:

- \( d(:, :, :, 1) \) is the keystrength for the major keys, and
- \( d(:, :, :, 2) \) is the keystrength for the minor keys.

If the input is a key estimation, two output are returned:

1. the keys (from 1 to 12) and
2. the modes (1 for major, 2 for minor).

If the input is the result of a peak detection, two output are returned:

1. the position of the peaks and
2. the value corresponding to these peaks, in the units predefined for this data.
mirexport

mirexport(filename, ...) exports statistical information related to diverse data into a text file called filename.

mirexport('Workspace', ...) instead directly output the statistical information in a structure array saved in the Matlab workspace. This structure contains three fields:

- filenames: the name of the original audio files,
- types: the name of the features,
- data: the data.

The exported data should be related to the same initial audio file or the same ordered set of audio files.

The data listed after the first arguments can be:

- any feature computed in MIRtoolbox. What will be exported is the statistical description of the feature (using the mirstat function)
- any structure array of such features. Such as the output of the mirstat function.
- any cell array of such features.
- the name of a text file. The text file is imported with the Matlab importdata command. Each line of the file should contains a fixed number of data delimited by tabulations. The first line, or ‘header’, indicates the name of each of these columns.

The file format of the output can be either:

- a text file. It follows the same text file representation as for the input text files. The first column of the matrix indicates the name of the audio files. The text file can be opened in Matlab, or in a spreadsheet program, such as Microsoft Excel, where the data matrix can be automatically reconstructed.
- an attribute-relation file. It follows the ARFF standard, used in particular in the WEKA data mining environment.
5. ADVANCED USE OF MIRTOOLBOX

5.1. get

get returns fields of MIRtoolbox objects. More details in next update of MIRtoolbox.

5.2. Memory management

There are important things to know in order to take benefit of the memory management mechanism offered by MIRtoolbox. More details in next update of MIRtoolbox.

USE FILE NAME AS INPUT

The memory management mechanism is automatically launched when using file name or ‘Folder’ keyword as input for most MIRtoolbox operators.

FLOWCHART DESIGN

If you write this command:

\[
a = \text{miraudio('myhugefile')}
\]
\[
s = \text{mirspectrum}(a, 'Frame')
\]
\[
c = \text{mircentroid}(s)
\]

there may be memory problem when computing the spectrogram. In order to benefit from MIRtoolbox memory management mechanisms, you should write instead:

\[
a = \text{miraudio('Design')}
\]
\[
s = \text{mirspectrum}(a, 'Frame')
\]
\[
c = \text{mircentroid}(s)
\]
\[
\text{mireval}(c, 'myhugefile')
\]

COMPLEX FLOWCHART DESIGN

But if you want to get several output in your flowchart, for instance:
\[ s = \text{mirspectrum}(\text{mysong}); \]
\[ \text{cent} = \text{mircentroid}(s); \]
\[ \text{ceps} = \text{mircepstrum}(s); \]

You should use a \textit{mirstruct} object, which store all the outputs of your flowchart into fields and write as follows:

\[ \text{myflow} = \text{mirstruct}; \]

All the temporary data used in your flowchart should be stored into a \textit{tmp} field:

\[ \text{myflow}.\text{tmp}.s = \text{mirspectrum}(\text{Design}); \]
\[ \text{myflow.cent} = \text{mircentroid}(|\text{myflow}.\text{tmp}.s|); \]
\[ \text{myflow.ceps} = \text{mircepstrum}(|\text{myflow}.\text{tmp}.s|); \]
\[ \text{output} = \text{mireval}(\text{myflow}, \text{`myhugefile'}); \]

I know this may look quite complex now. More documentation is necessary...
ANNEX A

References


Peeters. G. (2004). A large set of audio features for sound description (similarity and classification) in the CUIDADO project. version 1.0


