MIRtoolbox PRIMER

for absolute beginners

~with concrete musical applications

Olivier Lartillot
Finnish Centre of Excellence in Interdisciplinary Music Research
University of Jyväskylä, Finland
February, 14th, 2012

1. BASICS

Matlab Environment

Launch *Matlab*

To start *Matlab* in Mac OS X, open the *Applications* folder and click on the *MATLAB* icon.

Matlab Command Window

Once *Matlab* is launched, you should see a new window with many panels. The most important part of this window is the large white area called '*Command Window*' where you can dialog with *Matlab* by writing some little commands.

Installing *MIRtoolbox*

If *MIRtoolbox* is already installed (cf. subsection below "*Check the Installation*"), you don't need to read this section.

Download *MIRtoolbox* at this address:

http://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox

You can save the unzipped folder wherever you like (on the desktop, or in your documents folder, etc.).

Add *MIRtoolbox* in the *MATLAB* path:

1. In the "*File*" menu, select "*Set Path...*".
2. In the new "*Set Path*" window, click on the second button "*Add with Subfolders...*"
3. In the file browser, select the folder you downloaded and unzipped (it should be called MIRToolbox1.4.something) and click "*Open*". You should see several new lines highlighted in blue in the "MATLAB search path" list in the "*Set Path*" window. These are the folder addresses of the content of the toolbox.
4. To avoid performing the operations 1-3 every time you launch *Matlab*, save MATLAB path by clicking on the "*Save*" button. You can then click on the "*Close*" button.

Check the Installation

Check that *MIRtoolbox* is properly installed by typing

ver mirtoolbox

in the *MATLAB Command Window*. You should see something like:

MATLAB Version 7.x.x.xxx (R20xx)

MATLAB License Number: xxxxx

Operating System: .....

Java VM Version: .....

MIRtoolbox **Version 1.x.x**

If you see that last line, that means that *MIRtoolbox* is properly installed.

mirplay: Playing an audio file

Example Audio Files

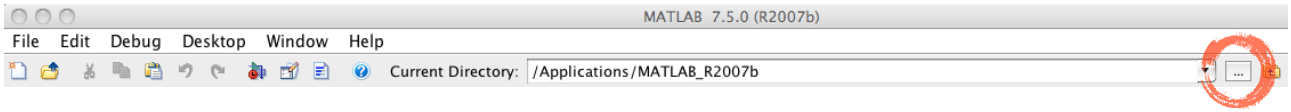
MIRtoolbox includes a set of musical examples that can be used to test the software. One of those files is a short excerpt of a piano ragtime, and is named *ragtime.wav*. To simply play that audio file, just write the following command in *Matlab* Command Window:

mirplay('ragtime')

and then press enter to execute that command.

Your Audio Files

Now let's try to load and play one of your own audio file. The file should be in WAV, MP3 or AU format, and should be stored in a folder accessible to your computer (for instance, your '*Desktop*' folder). First of all, you need to tell *Matlab* that the *Current Directory* you want to consider now is that particular folder where your file is located. To set the Current Directory, just click on the '*Browse for folder*' icon ('...' button next to the Current Directory field in the main *Matlab* toolbar:)



This opens a file browser where you can select that particular folder, and click '*Open*'.

Once you have selected that folder, you should now see the content of that folder on the '*Command Directory*' panel on the left of the *Matlab* window. In particular, the file you want to analyze should be indicated in that panel. Now you can simply play that file as before. If for instance your file is called *myfile.wav*, then just write the following command:

mirplay('myfile')

Playing a Folder of Files

You can also play a complete folder by writing the following command:

*mirplay('**Folder**')*

How to Abort a Process

If some particular process you have launched in *Matlab* takes too much time, you can abort it at any point. For instance, if you asked to play all the audio files of a folder, you can stop that playlist by pressing on both keys '*ctrl*' and '*c*' altogether. It might sometime takes some time to abort. If *Matlab* still does not react, try pressing this key combination several times.

miraudio: Audio waveform

MIRtoolbox offers a set of tools for the analysis of audio recordings, of music in particular. The simplest representation that can be obtained from an audio recording is the waveform representation of the audio signal.

To display the audio waveform of the *ragtime* file, just write the following:

***miraudio**('ragtime')*

This opens a new *Figure* window that contains the graphical representation of the audio waveform.

Saving Figures as Images

You can save that image by selecting '*Save*' in the '*File*' menu on the top of the Figure window. Matlab can save in various image format, that you can see in the File Format list. If for instance you want to save the image for further use in Microsoft Office, you can save it for instance using a file name with the PNG extension, such as *myimage.png*.

Your Audio File

To display the audio waveform of a file called *myfile.wav* located in the Current Directory, just write the following:

miraudio('myfile')

Analyzing a Folder of Files

To display the audio waveforms of all the audio files located in the Current Directory, just write the following:

*miraudio('**Folder**')*

Extracting a Part

You can focus on one particular part of your audio file. If for instance, you want to see only the excerpt starting at t = 20 seconds and ending at t = 30 seconds, just write the following:

*miraudio('myfile', **'Extract',** 20, 30)*

Playing an Excerpt

You can store your excerpt in a variable – let's call it *a*, for instance:

***a** = miraudio('myfile', 'Extract', 20, 30);*

TIP: If you don't want to see the graphical output of this command, just add '**;**' at the end of the line, as in the previous example.

In this way you can then play the result (i.e., that particular excerpt) by simply writing:

*mirplay(**a**)*

mirrms: Global energy

Global Representation

One very simple description of sound is its global energy computed using an operator called RMS (Root Mean Square). To get the global energy for the *ragtime* example, just write the following:

***mirrms**('ragtime')*

This first representation simply gives one number indicating the amount of energy for the whole piece of music. You should get this text in the Command Window:

The RMS energy related to file ragtime is 0.017932

This could be used to compare different audio files, for instance.

*mirrms('**Folder**')*

Temporal Evolution

Alternatively, we can also see the evolution over time of this energy, by using the '*Frame*' keyword:

*mirrms('ragtime', **'Frame'**)*

Play the Result **(NEW!)**

You can also visualise the curve while playing the audio at the same time, using the new interface developed by Pasi Saari. Please note that this requires *MIRtoolbox* version 1.3.4.4.

r = mirrms('ragtime', 'Frame')

***mirplayer**(r)*

In the new '*MIRplayer*' window, you have under the left column '*SELECT FEATURES AND PEAKS*' the name of the feature we just computed: '*RMS energy*'. Click on the triangle on its left to unroll the list of colors, and click on the first black icon. You should see the RMS curve appearing on the graph. Now click on the play button...

Analyzing an Excerpt

You can get the global energy of the excerpt you have selected:

a = miraudio('myfile', 'Extract', 20, 30)

mirrms(a)

And similarly, for the temporal evolution of energy of that excerpt:

mirrms(a, 'Frame')

2. SPECTRAL ANALYSIS

mirspectrum: Spectral decomposition

Global Representation

One common representation of sound is by displaying the repartition of energy along the different frequencies. To get this spectral representation for the *ragtime* example, just write the following:

***mirspectrum**('ragtime')*

This first representation simply gives the global repartition for the whole piece of music. You can use the magnifier button in the toolbar to zoom in in the picture, or alternatively you can select a particular frequency region, for instance below 3000 Hz:

*mirspectrum('ragtime', **'Max'**, 3000)*

Temporal Evolution

Alternatively, we can also see the evolution over time of this spectral representation, by using the '*Frame*' keyword:

*mirspectrum('ragtime'**, 'Frame'**)*

Now the different frequencies are represented on the vertical axis, the horizontal axis corresponds to the temporal evolution of the music. Similarly, you can zoom in, or select a frequency region, for instance:

mirspectrum('ragtime', 'Frame', 'Max', 3000)

Analyzing an Excerpt

You can get the spectral representation of the excerpt you have selected:

a = miraudio('myfile', 'Extract', 20, 30)

*mirspectrum(**a**)*

And similarly, for the temporal spectral evolution of that excerpt:

mirspectrum(a, 'Frame')

Spectral Flux

You can compare the spectrum between each successive frame, that gives a temporal curve, called *spectral flux*:

```
s = mirspectrum('ragtime', 'Frame')

mirflux(s)
```

The peaks in this curve shows where there are important changes in the spectrum.

Play the Result **(NEW!)**

You can also visualise the curve while playing the audio at the same time:

```
f = mirflux(s)

mirplayer(f)
```

mirbrightness: How bright is the sound?

```
b = mirbrightness('myfile', 'Frame')

mirplayer(b)
```

The *brightness* curve shows the evolution of brightness throughout the piece of music. High values indicates moments in the music where most of the sound energy is on the high-frequency register, whereas low values indicates moments where most of the sound energy is on the low-frequency register.

You can also compute the curve for a given excerpt of your audio file, for instance between t_1 = 20 seconds and t_2 = 30 seconds:

```
a = miraudio('myfile', 'Extract', 20, 30)

mirbrightness(a, 'Frame')
```

You can also compute a global brightness value for each audio files in your Current Directory. We suggest for instance to select the folder *ShortClips* as Current Directory. Write then:

```
b = mirbrightness('Folder')
```

So that you can now play the audio files in increasing order of brightness:

```
mirplay('Folder', 'Increasing', b)
```

mircentroid: Spectral centroid

```
c = mircentroid('ragtime', 'Frame')

mirplayer(c)
```

The *spectral centroid* curve is quite similar to the *brightness* curve. It shows, for each successive instant of the music, around which frequencies the sound energy is centered. High values indicates moments in the music where most of the sound energy is on the high-frequency register, whereas low values indicates moments where most of the sound energy is on the low-frequency register.

You can also compute the curve for a given excerpt of your audio file, for instance between t_1 = 20 seconds and t_2 = 30 seconds:

```
a = miraudio('myfile', 'Extract', 20, 30)

mircentroid(a, 'Frame')
```

You can also compute a global spectral centroid value for each audio files in your Current Directory:

```
c = mircentroid('Folder')
```

So that you can now play the audio files in increasing order of spectral centroid:

```
mirplay('Folder', 'Increasing', c)
```

mirroughness: Sensory dissonance

```
r = mirroughness('ragtime')

mirplayer(r)
```

The *roughness* curve shows the amount of sensory dissonance at each successive moments throughout the piece of music. This sensory dissonance corresponds to the "beating" phenomenon when several sound are heard with nearly the same frequency, but with just a few Hz of

difference. When roughness is high, the sounds feels more harsh, containing more strange oscillations.

You can also compute the curve for a given excerpt of your audio file, for instance between t_1 = 20 seconds and t_2 = 30 seconds:

```
a = miraudio('myfile', 'Extract', 20, 30)
```

```
mirroughness(a)
```

You can also compute a roughness curve for each audio files in your Current Directory:

```
r = mirroughness('Folder')
```

Then get the average of roughness for each separate file, so that we have now one global value for each file:

```
m = mirmean(r)
```

So that you can now play the audio files in increasing order of roughness:

```
mirplay('Folder', 'Increasing', m)
```

3. TEMPO

mirtempo: Tempo estimation

You can get a tentative estimation of the tempo of a given recording by writing for instance:

mirtempo*('myfile')*

A temporal evolution of tempo, or *tempo curve*, can also be obtained:

t = mirtempo('myfile', ***'Frame'****)*

mirplayer(t)

You can also compute a global tempo value for each audio files in your Current Directory. We suggest for instance to select the folder *LongClips* as Current Directory. Write then:

*t = mirtempo(****'Folder'****)*

So that you can now play the audio files in increasing order of tempo:

mirplay('Folder', ***'Increasing'****, t)*

mirpulseclarity: Clarity of the pulsation

It is possible to get an estimation of the strength of the beat – i.e., the relative importance of the regular pulsation – by using the command:

mirpulseclarity*('myfile')*

p = mirpulseclarity('myfile', ***'Frame'****)*

mirplayer(p)

You can also compute a global pulse clarity value for each audio files in your Current Directory:

*p = mirpulseclarity(****'Folder'****)*

So that you can now play the audio files in increasing order of pulse clarity:

mirplay('Folder', ***'Increasing'****, p)*

4. TRANSCRIPTION

mironsets: Detection of the successive notes

In order to detect the notes that are contained in a recording, we first need to find where in time the notes starts. For that, a *onset curve* can be computed like this:

mironsets*('myfile')*

The peaks of the curve indicates the point where the energy is the highest. We can suppose that these peaks corresponds the successive notes in the music. These peaks are highlighted with red circles.

Before the peak of each note, there is the *attack phase* where the energy progressively increase. We can see the attack phase of each note:

mironsets('myfile', ***'Attacks'****)*

Similarly, we can also see the *release phase*, where the energy of the note progressively decreases after the peak:

mironsets('myfile', 'Attacks', ***'Releases'****)*

mirsegment: Segmentation

Then you can segment the audio files at the position of those onsets, by writing:

o = mironsets('myfile', 'Attacks', ***'Releases'****)*

s = ***mirsegment****('myfile', o)*

Then play the result of this segmentation:

mirplay(s)

mirattackslope: Attack of notes

These attack phases show how each note is attacked: if a note has a steep attack (short attack but with a very fast increase), it sounds more aggressive than if a note has a more gradual attack. We can display the attack slope of each successive note, which indicates whether each note has a steep attack (high attack slope value) or gradual attack (low attack slope value).

mirattackslope*('myfile')*

mireventdensity: Density of notes

You can display the density of notes over time, like this:

mireventdensity*('myfile', 'Frame')*

High values in the curve indicates moment where there is a lot of notes, low values shows moments when there is not a lot of notes.

mirpitch: Pitch height

We saw in section 2 that we can have a first simple representation of the successive notes with their pitch by computing the spectrogram, for instance using:

mirspectrum(a, 'Frame')

We can see some kind of music score emerging from this picture, but if we need more explicit information about these pitch height, we need to perform more complex operations. Hopefully, you don't need to do these operations yourself, you can use advanced operators, like *mirpitch*, to get these advanced operations, for instance:

p = ***mirpitch****('myfile',* ***'Frame'****)*

mirplayer(p)

This shows the progressive pitch height discovered throughout the audio recording.

5. TONALITY

mirchromagram: Pitch class profile

mirchromagram*('myfile')*

The *chromagram* curve shows the distribution of energy along the 12 pitch classes.

You can also see the evolution of this distribution over time, using the *'Frame'* option:

mirchromagram*('myfile', 'Frame')*

mirkeystrength: Tonal analysis

mirkeystrength*('myfile')*

The *key strength* curve shows the probability for each different key, major (in blue) and minor (in red). The most probable key is the one with the highest value.

You can also see the evolution of this key estimation over time, using the *'Frame'* option:

mirkeystrength*('myfile', 'Frame')*

As usual, the horizontal axis represents the time, and here the different key candidates are the different lines. Red colors shows the most probable keys at each particular instant.

mirkey: Tonal analysis

mirkey*('myfile')*

This simply returns the most probable key found.

You can also see the evolution of this key estimation over time, using the *'Frame'* option:

mirkey*('myfile', 'Frame')*

mirmode: Major vs. minor

mirmode*('myfile')*

mirmode tries to estimate if the piece is in major, or in minor mode. A high positive value indicates majorness and a high negative value indicates minorness. A value close to 0 indicates an ambiguity between major and minor.

You can also see the evolution of this mode estimation over time, using the *'Frame'* option:

```
m = mirmode('myfile', 'Frame')

mirplayer(m)
```

mirkeysom: Key map

```
mirkeysom('myfile')
```

This *keysom* maps locates the tonality of the piece in a 2D maps where all the tonalities are represented with suitable perceptive distance between them.

You can also see the evolution of this *keysom* estimation over time, using the *'Frame'* option:

```
mirkeysom('myfile', 'Frame')
```

6. STRUCTURE

mirsimatrix: Similarity matrix

mirsimatrix*('myfile')*

The *similarity matrix* shows the similarity between each instant of the piece of music and each other instant of that same piece. Time is displayed both in the horizontal and in the vertical axis. By default it is computed based on the spectral representation (*mirspectrum*).

You can also compute the similarity matrix for other representations, for instance:

c = mirchromagram('myfile', 'Frame')

mirsimatrix(c)

Try both methods (the default with *mirspectrum* and the one with *mirchromagram*) and observe the results. You can try with the different pieces in the *LongClips* folder.

There is also a timbral feature we haven't look at yet, called MFCC, whose actual definition is quite technical. One interest of it is that it is useful to show temporal evolutions in the timbral domain:

c = ***mirmfcc****('myfile', 'Frame')*

mirsimatrix(c)

mirnovelty: Novelty curve

n = ***mirnovelty****('myfile')*

mirplayer(n)

The *novelty curve* shows the moments in time where there is the most important contrasts. By default, it is based, like *mirsimatrix*, on *mirspectrum*.

You can also compute the similarity matrix for other representations, for instance:

c = mirchromagram('myfile', 'Frame')

```
s = mirsimatrix(c)

n = mirnovelty(s)
```

Try both methods (the default with *mirspectrum* and the one with *mirchromagram*) and observe the results. You can try with the different pieces in the *LongClips* folder.

You can then automatically highlight the peaks in the novelty curve:

```
p = mirpeaks(n)
```

mirsegment: Segmentation

Then you can segment the audio files at the position of those peaks, by writing:

```
s = mirsegment('myfile', p)
```

Then play the result of this segmentation:

```
mirplay(s)
```

You can then perform any analysis you like on this segmentation, for instance:

```
p = mirpitch(s)

mirplay(p)
```

7. WHAT CAN I DO NOW?

Now there are plenty of different kind of analyses you can perform with all the building blocks you just discovered. And other tools are available as well in *MIRtoolbox*. More details in the *User's Guide*.

You can also get a synthetic list of commands available in *MIRtoolbox* by typing:

help mirtoolbox

Each *MIRtoolbox* command, even all those we saw in this short tutorial, offers a large set of possible options. You can see more details about these command by looking at the online help for each command by typing, for instance:

***help** miraudio*

The best documentation is in the *User's Guide*, with explanations, figures, etc.

If you need the actual numerical values of the output of your computations, use *mirgetdata*:

s = mirspectrum('ragtime');

***mirgetdata**(s)*

You can also get basic statistics from your analyses, for instance:

t = mirtempo('myfile', 'Frame')

***mirstat**(t)*

You can export your analyses to text files, that you can use in other programs, such as SPSS:

t = mirtempo('myfile', 'Frame')

b = mirbrightness('myfile', 'Frame')

***mirexport**('myresults.txt', t, b)*

Again, don't forget to have a look at the extensive documentation offered in the *User's Guide*.

Have fun!