The AUTOTONNO tool

A Matlab implementation for pitch detection and correction

Author: Leonardo Fierro, Andrea Claudio

October 4, 2018

Università degli Studi di Brescia
Digital Audio Processing - CTMLM A.Y. 2017/18

Overview

- *Autotonno* is a simple tool to experiment and implement pitch analysis on a (short) audio signal in Matlab.
- Main functions:
  - pitch detection;
  - pitch correction;
  - vibrato.
- Input: recorded or imported *wave* file.
- Output:
  - graphically: instantaneous modifications;
  - exportable *wave* file.

Overview

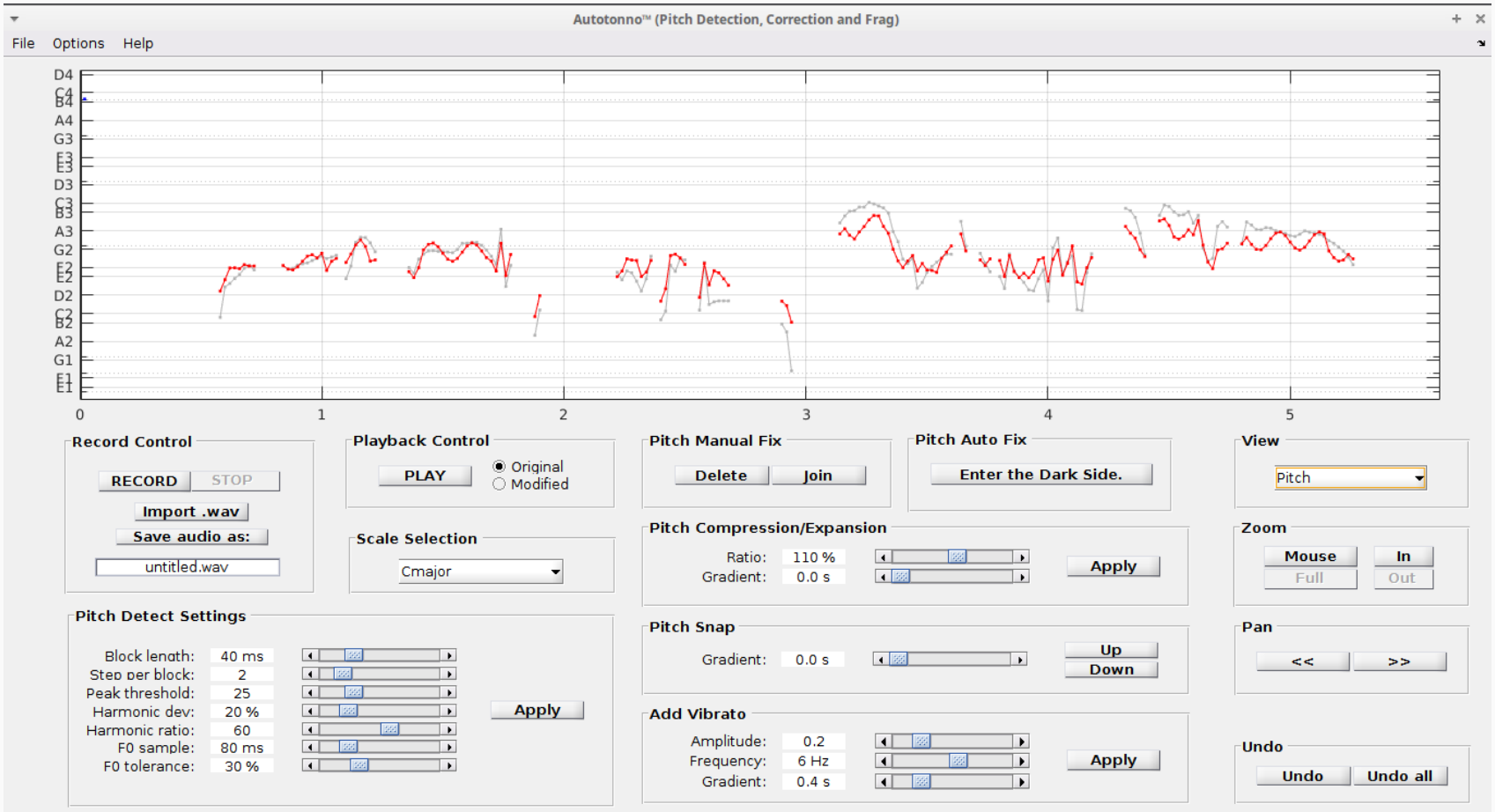


Figure 1: Scheme of the *Autotonno* tool.

The GUI editor: GUIDE

A GUI (*Graphic User Interface*) offers point-and-click control on software applications, avoiding the need of learning a code language or digit commands to execute such apps.

GUIDE (*GUI + IDE*) is a Matlab Toolbox that allows the programmer to design user interfaces for personalized apps via Layout Editor. Many Matlab products, such as Signal Processing Toolbox or Curve Fitting Toolbox, have been made using GUIDE.

GUIDE implements all typical GUI controls: menus, toolbars, buttons, cursors, textboxes...

The GUI editor: GUIDE (2)

GUIDE automatically generates:

- a Matlab figure (.fig): GUIDE "project", contains the created GUI;
- a Matlab code (.m): automatically generated, contains initialization, callbacks and user-addable code.

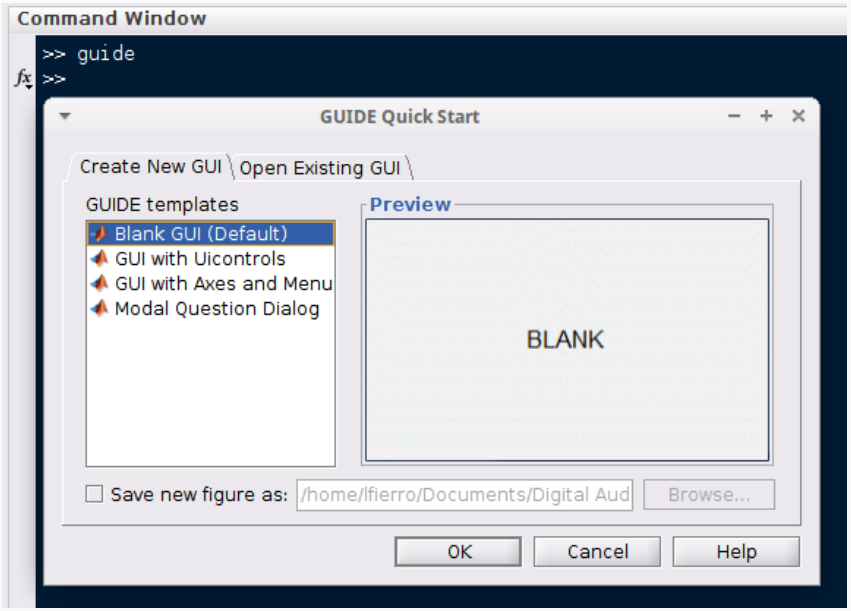


Figure 2: *Guide* tool.

The GUI editor: GUIDE (3)

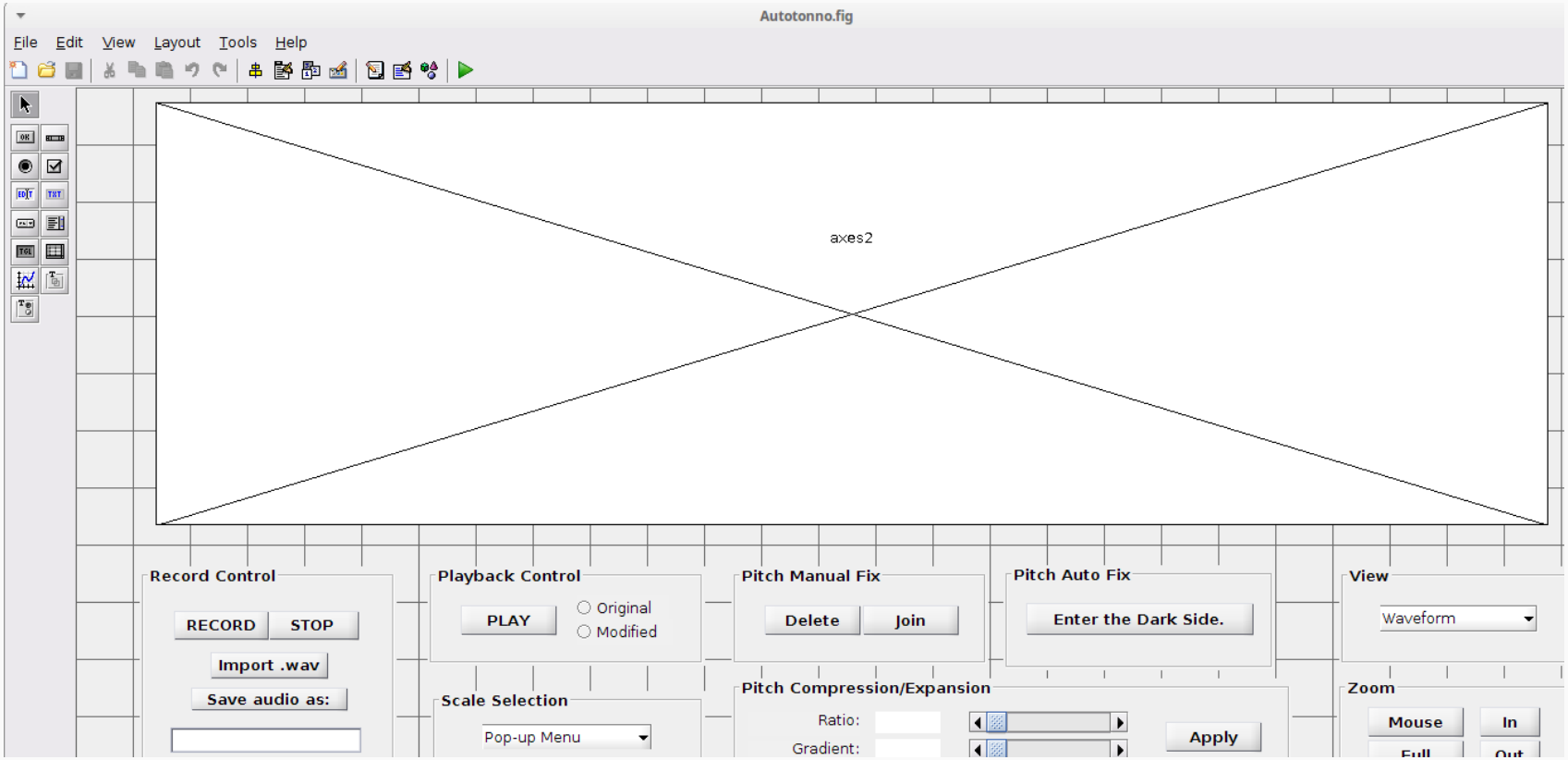


Figure 3: *Autotonno.fig* created with GUIDE

Initialization: input files

Input files allowed:

- imported audio file:
  - only *.wav* files;
  - sample frequency from file properties;
- recorded audio file:
  - recording device is the active one on computer;
  - sample frequency from default settings.

Both kinds undergo the same algorithm for pitch detection based on common user-defined settings (see later).

Pitch detection: main idea

Implemented pitch detection algorithm uses a modified auto-correlation routine in the time domain, operating on small blocks of the recorded signal at a time:

1. First N samples are extracted and a copy of it is made.
2. The copy is slid across the original block of N points to determine the "likeness".

Since to a particular pitch corresponds a periodic signal, when the copy block is slid of one period it should match the original block. The period gives the frequency, that is, the pitch.

Process is repeated for all of the remaining blocks of samples that comprise the entire signal to find local pitches.

Pitch detection: algorithm

1. Extract a block of the signal and make a copy of it.
   - OLA (*Overlap-Add*): it is possible to overlap part of the blocks to increase the frequency points. 50% overlap is a good call.
2. Define minimum and maximum shifts (F_{min}, F_{max}).
3. Find "auto-correlation":

$$X_{corr}(\tau) = 1 - \frac{\Sigma|A_{copy}(n) - A_{block}(n-\tau)|}{2 \cdot \Sigma|A_{copy}(n)|}$$

4. Level auto-correlation with a polynomial function.
   - Minimum value is subtracted, so that lower bound is equal to zero.

Pitch detection: algorithm (2)

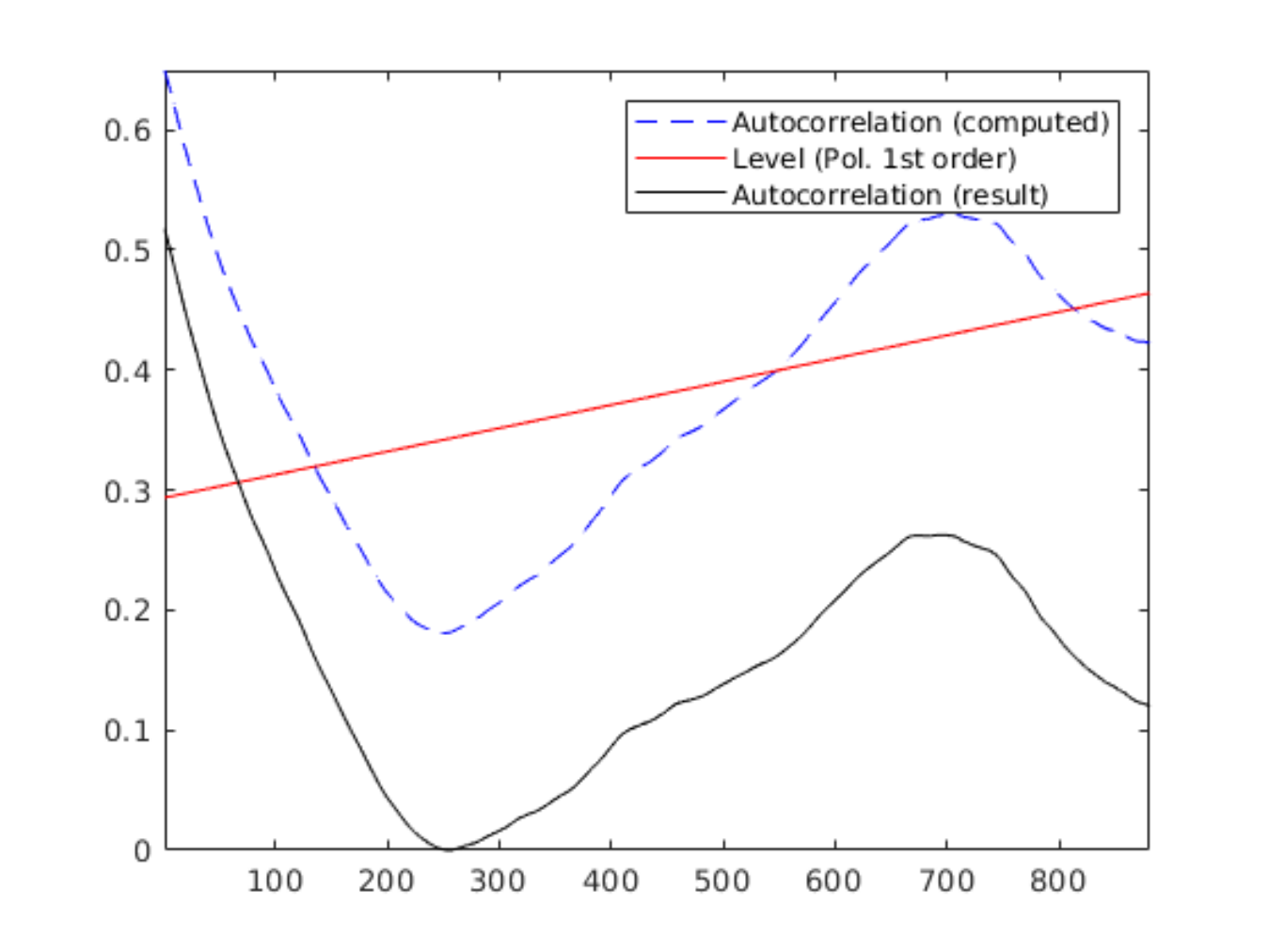


Figure 4: Auto-correlation function.

Pitch detection: algorithm (3)

5. Identify valid peaks (above thresholds):
   - Find min/max points w/ differential;
   - Remove peaks under a user-defined/default *peak value*;
   - Remove peaks closer than *minimum allowed separation*.
6. Keep only those harmonics whose frequency is less than a user-defined/default *harmonic deviation* around a multiple of the fundamental.
7. Choose the ultimate pitch:
   - Most dominant harmonic, or
   - Closest to the desired harmonic (*target frequency*);

Pitch detection: settings

- **Block length**: dimension (in time) of the block, and therefore how many samples are in each block ($N = \textit{block length} \cdot \textit{sample rate}$) *[Default: 40 ms]*.
- **Step per block**: degree of overlap, that is, how many samples to skip when extracting the next block *[Default: 2]*.

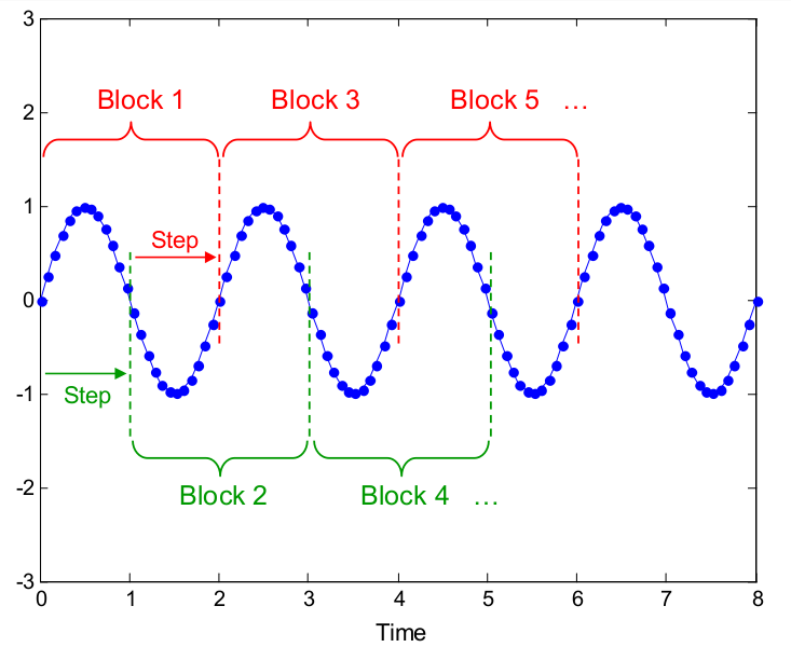


Figure 5: Example of a sampled waveform with 50% overlap.

Pitch detection: settings (2)

- **Peak threshold**: minimum correlation value required to determine the frequency of the signal *[Default: 25]*.
  - PT $= 100 \Rightarrow$ perfect correlation
  - PT $= 0 \Rightarrow$ no similarity

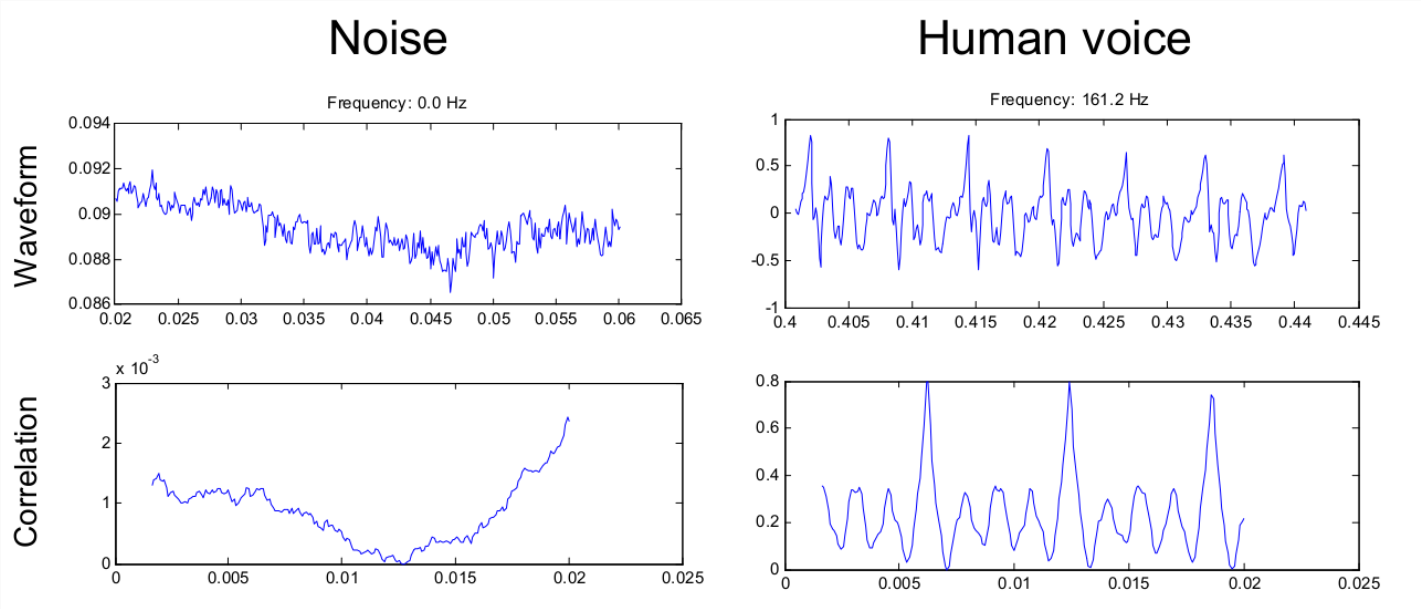


Figure 6: Example of correlation shown by noise and human voice.

Pitch detection: settings (3)

- **Harmonic deviation**: allowable tolerance that a peak can be off from the predicted harmonic frequency to be still considered a peak *[Default: 20%]*.
- **Harmonic ratio**: allowable ratio in peak height wrt main peak to be classified as a harmonic. *[Default: 60]*.
- **F0 sample**: number of previous samples used for prediction of the next block pitch *[Default: 80 ms]*.
- **F0 tolerance**: maximum deviation allowed from frequency predicted by the previous few samples *[Default: 30%]*.

Pitch detection: scales

Pitch graph on screen is a sort of "spectrogram": it shows the variation of the pitch of the audio signal in time.

- Scale range (dynamic): all notes from A1 (55 Hz) to A5 (880 Hz).
- Both major/minor scales.
- Corresponding pitch is computed as:

$$f_0 = 55 \cdot 2^{\frac{index}{12}}$$

  where *index* is a vector of integers indicating the progression of notes ($0 \Rightarrow$ A1, $48 \Rightarrow$ A5).
- Scale selection can be made by the user at any time.

Pitch correction: main idea

Major modifications appliable to the signal are:

- Pitch auto-fix ("*Auto-Tune* EFX").
- Pitch compression/expansion.
- Pitch snap (shift up/down).
- Pitch manual connections.

All of these functions modify the spectral and amplitude behaviour of the signal, so an underlying common interpolation algorithm is implemented to compensate for each modification.

Pitch correction: quantization

Based on the current scale selection, the tool automatically finds the note of the scale which is closest in pitch to the selected point and forces it to that scale pitch value.

- Commonly known as Auto-tune EFX/pitch quantization.
- No gradient, 1st order interpolation transition.

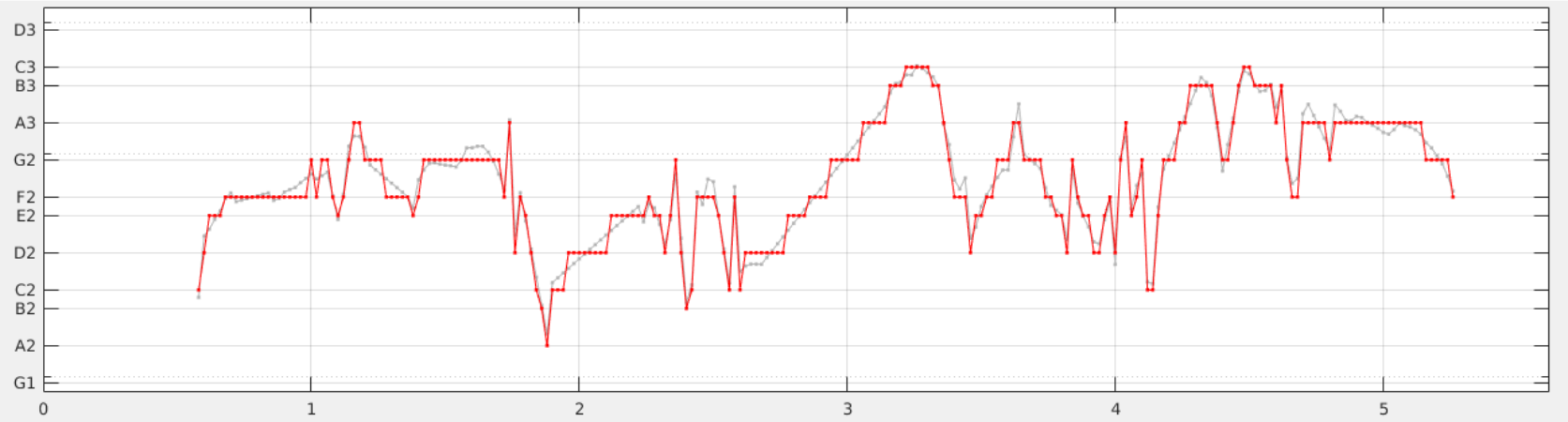


Figure 7: Quantized (red) vs original (grey)

Pitch correction: compression/expansion

Dynamic range compression (DRC) is an audio signal processing operation that reduces the volume of loud sounds or amplifies quiet sounds thus reducing or compressing dynamic range of an audio signal. Compressors are commonly used in sound recording and reproduction, broadcasting, live sound reinforcement and in some instrument amplifiers.

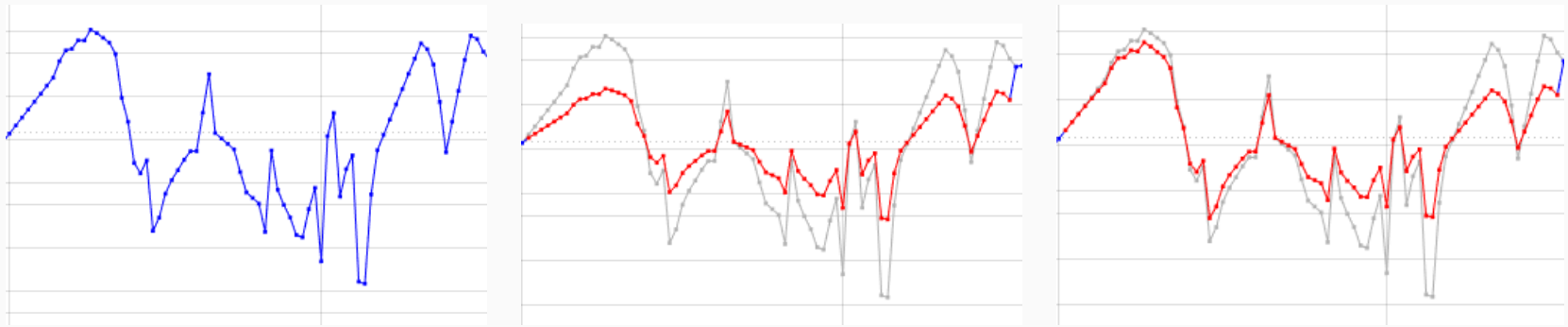
Viceversa, expanders increase the dynamic range of the audio signal and are generally used to make quiet sounds even quieter by reducing the level of an audio signal that falls below a set threshold level.

The method impremented involves scaling the deviations around the mean in a log sense with respect to the compression/expansion ratio:

$$f_0 = 10^{\,mean[\log f_0]} + S_{cmp/exp} \cdot (\log f_0 - mean[\log f_0])$$

Pitch correction: compression/expansion (2)

- **Ratio**: determines how much the pitch variation is decreased/increased each time process is applied. A ratio of 50% means that the overall pitch variation of the selected points will decrease by 50% with each iteration.
- **Gradient**: determines the time over which the pitch is ramped from no compression to the desired compression ration. This is used to "smooth" out the transition to the compressed/expanded pitch.



(a) Original (b) 50% compression (c) 50% w/ 1 s gradient

Figure 8: Compression with and without gradient.

Pitch correction: snap

There are two "main" modes that affect the way in which a note will snap to a certain pitch:

- Absolute snapping: each pitch is shifted to the next semitone.
- Relative snapping: segment is shifted keeping its current deviation in cents.

In this tool, relative snapping is implemented, using an algorithm similiar to the one for compression:

$$f_{0_m} = 10^{mean[\log f_0]} \qquad f_{0_s} = \begin{cases} \max\left(f_{scale} < f_{0_m}\right) & (down) \\ \min\left(f_{scale} > f_{0_m}\right) & (up) \end{cases}$$

$$f_{0_{snap}} = \frac{f_{0_s}}{f_{0_m}} \cdot f_0$$

Pitch correction: snap (2)

If only one point is selected on the pitch graph, what happens is the same as absolute snapping, as the pitch will be shifted to the closest next semitone above/below.

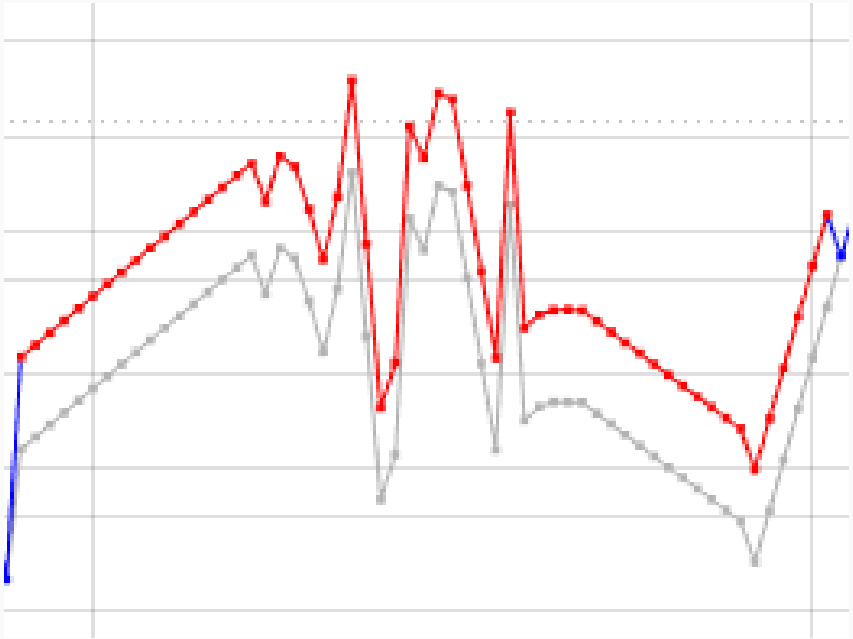


Figure 9: Snapping upwards.

There is also a gradient parameter, as for compression, to determine the transient of snapping.

Pitch correction: manual connections

Some detected pitch points may be erroneous, or at least separated from the rest. For this reason, the tool implements the possibility to delete any of the detected points or to link together two points, that are initially separated, via interpolation, generating new "fake" pitch points.

(a) Original (b) Points deleted (c) Points connected

Figure 10: Manual join/delete of pitch points.

Vibrato

Vibrato is a musical effect consisting of a regular, pulsating change of pitch. It is used to add expression to vocal and instrumental music. Vibrato is typically characterised in terms of two factors: the amount of pitch variation ("extent of vibrato") and the speed with which the pitch is varied ("rate of vibrato").

Vibrato settings:

- **Amplitude**: magnitude of the pitch variation of the vibrato; a value of 1.0 corresponds to a variation of one semitone from nominal frequency.
- **Frequency**: pitch variation per second of the vibrato.
- **Gradient**: How quickly the vibrato grows to the peak amplitude.

Vibrato

Vibrato is added in a simple way, as follows:

$$V = \frac{A_{vib}}{12} \cdot \sin(2\pi f_{vib}\ t) \qquad f_{0_{vib}} = 2^V \cdot f_0$$

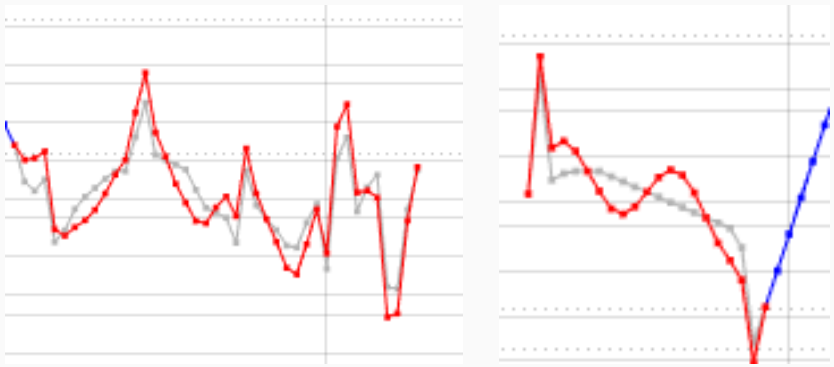


Figure 11: Vibrato added to the input signal.

Limits and possible improvements

- Given that the tool uses memory to store the audio data, performances are not optimized, and can be compromised by longer audio files.
- When working on audio whose pitches comprise different octave, some graphical errors in the scale priting arise (unknown reason up-to-now).
- Algorithms for pitch detection should be improved, also in terms of code optimization.

+ Import/export audio in MP3 or other formats.
+ Drag-and-drop functionalities.
+ DAFX $\Rightarrow$ create a real audio suite.
+ Work in real-time.
  $\Rightarrow$ Hard, but possible using Audio System Toolbox and Playrec.

Special thanks

- Mathworks community, that came up with most of the algorithms used in this tool.
- Steinberg community, for the understanding of pitch compression and snapping.
- Antares Software, for the “trailing idea” for this project