AMFM decompy Documentation

Release 1.0.5

Bernardo J. B. Schmitt

Sep 22, 2016

CONTENTS

Contents:

CHAPTER

ONE

PYAAPT

This a ported version for Python from the YAAPT (Yet Another Algorithm for Pitch Tracking) algorithm. The original MATLAB program was written by Hongbing Hu and Stephen A. Zahorian.

The YAAPT program, designed for fundamental frequency tracking, is extremely robust for both high quality and telephone speech. The YAAPT program was created by the Speech Communication Laboratory of the state university of New York at Binghamton. The original program is available at http://www.ws.binghamton.edu/zahorian as free software. Further information about the program could be found at *[ref1]*.

It must be noticed that, although this ported version is almost equal to the original, some few changes were made in order to make the program more "pythonic" and improve its performance. Nevertheless, the results obtained with both algorithms were similar.

Quick start

The pYAAPT basically contains the whole set of functions to extract the pitch track from a speech signal. These functions, in their turn, are independent from the pyQHM package. Therefore, pYAAPT can be used in any other speech processing application, not only in the AM-FM decomposition.

USAGE:

`amfm_decompy.pYAAPT.`**`yaapt`**(*signal*[, *options*])

Parameters

- **`signal`** (*`signal object`*) – created with amfm_decompy.basic_tools.
- **`options`** (Must be formatted as follow: **{'option_name1' : value1, 'option_name2' : value2, ...}) – the default configuration values for all of them are the same as in the original version. A short description about them is presented in the next subitem. For more information about them, please refer to *[ref1]*.

Return type pitch object

OPTIONS:

- 'frame_length' - length of each analysis frame (default: 25 ms)
- 'frame_space' - spacing between analysis frames (default: 10 ms)
- 'f0_min' - minimum pitch searched (default: 60 Hz)
- 'f0_max' - maximum pitch searched (default: 400 Hz)
- 'fft_length' - FFT length (default: 8192 samples)
- 'bp_forder' - order of band-pass filter (default: 150)

- 'bp_low' - low frequency of filter passband (default: 50 Hz)
- 'bp_high' - high frequency of filter passband (default: 1500 Hz)
- 'nlfer_thresh1' - NLFER (Normalized Low Frequency Energy Ratio) boundary for voiced/unvoiced decisions (default: 0.75)
- 'nlfer_thresh2' - threshold for NLFER definitely unvoiced (default: 0.1)
- 'shc_numharms' - number of harmonics in SHC (Spectral Harmonics Correlation) calculation (default: 3)
- 'shc_window' - SHC window length (default: 40 Hz)
- 'shc_maxpeaks' - maximum number of SHC peaks to be found (default: 4)
- 'shc_pwidth' - window width in SHC peak picking (default: 50 Hz)
- 'shc_thresh1' - threshold 1 for SHC peak picking (default: 5)
- 'shc_thresh2' - threshold 2 for SHC peak picking (default: 1.25)
- 'f0_double'- pitch doubling decision threshold (default: 150 Hz)
- 'f0_half' - pitch halving decision threshold (default: 150 Hz)
- 'dp5_k1' - weight used in dynamic program (default: 11)
- 'dec_factor' - factor for signal resampling (default: 1)
- 'nccf_thresh1' - threshold for considering a peak in NCCF (Normalized Cross Correlation Function) (default: 0.25)
- 'nccf_thresh2' - threshold for terminating search in NCCF (default: 0.9)
- 'nccf_maxcands' - maximum number of candidates found (default: 3)
- 'nccf_pwidth' - window width in NCCF peak picking (default: 5)
- 'merit_boost' - boost merit (default. 0.20)
- 'merit_pivot' - merit assigned to unvoiced candidates in definitely unvoiced frames (default: 0.99)
- 'merit_extra' - merit assigned to extra candidates in reducing pitch doubling/halving errors (default: 0.4)
- 'median_value' - order of medial filter (default: 7)
- 'dp_w1' - DP (Dynamic Programming) weight factor for voiced-voiced transitions (default: 0.15)
- 'dp_w2' - DP weight factor for voiced-unvoiced or unvoiced-voiced transitions (default: 0.5)
- 'dp_w3' - DP weight factor of unvoiced-unvoiced transitions (default: 0.1)
- 'dp_w4' - Weight factor for local costs (default: 0.9)

EXAMPLES:

Example 1 - extract the pitch track from a signal using the default configurations:

```
import amfm_decompy.pYAAPT as pYAAPT
import amfm_decompy.basic_tools as basic

signal = basic.SignalObj('path_to_sample.wav')
pitch = pYAAPT.yaapt(signal)
```

Example 2 - extract the pitch track from a signal with the minimum pitch set to 150 Hz, the frame length to 15 ms and the frame jump to 5 ms:

```
import amfm_decompy.pYAAPT as pYAAPT
import amfm_decompy.basic_tools as basic

signal = basic.SignalObj('path_to_sample.wav')
pitch = pYAAPT.yaapt(signal, **{'f0_min' : 150.0, 'frame_length' : 15.0, 'frame_space
↪' : 5.0})
```

Classes

PitchObj Class

The PitchObj Class stores the extracted pitch and all the parameters related to it. A pitch object is necessary for the QHM algorithms. However, the pitch class structure was built in a way that it can be used by any other pitch tracker, not only the YAAPT.

USAGE:

`amfm_decompy.pYAAPT.`**`PitchObj`** (*frame_size*, *frame_jump*[, *nfft=8192*])

Parameters

- **`frame_size`** (*int*) – analysis frame length.
- **`frame_jump`** (*int*) – distance between the center of a extracting frame and the center of its adjacent neighbours.
- **`nfft`** (*int*) – FFT length.

Return type pitch object.

PITCH CLASS VARIABLES:

These variables not related with the YAAPT algorithm itself, but with a post-processing where the data is smoothed and halving/doubling errors corrected.

`PitchObj.`**`PITCH_HALF`**

This variable is a flag. When its value is equal to 1, the halving detector set the half pitch values to 0. If PITCH_HALF is equal to 2, the half pitch values are multiplied by 2. For other PITCH_HALF values, the halving detector is not employed (default: 0).

`PitchObj.`**`PITCH_HALF_SENS`**

Set the halving detector sensibility. A pitch sample is considered half valued if it is not zero and lower than:

mean(pitch) - PITCH_HALF_SENS*std(pitch)

(default: 2.9).

`PitchObj.`**`PITCH_DOUBLE`**

This variable is a flag. When its value is equal to 1, the doubling detector set the double pitch values to 0. If PITCH_DOUBLE is equal to 2, the double pitch values are divided by 2. For other PITCH_DOUBLE values, the doubling detector is not employed (default: 0).

`PitchObj.`**`PITCH_DOUBLE_SENS`**

Set the doubling detector sensibility. A pitch sample is considered double valued if it is not zero and higher than:

mean(pitch) + PITCH_DOUBLE_SENS*std(pitch) (default: 2.9).

PitchObj.**SMOOTH_FACTOR**
Determines the median filter length used to smooth the interpolated pitch values (default: 5). [1]

PitchObj.**SMOOTH**
This variable is a flag. When its value is not equal to 0, the interpolated pitch is smoothed by a median filter (default: 5). [1]

PitchObj.**PTCH_TYP**
If there are less than 2 voiced frames in the file, the PTCH_TYP value is used in the interpolation (default: 100 Hz). [1]

EXAMPLE:

Example 1 - the pitch is extracted from sample.wav with different smoothing and interpolation configurations:

```
import amfm_decompy.pYAAPT as pYAAPT
import amfm_decompy.basic_tools as basic

signal = basic.SignalObj('path_to_sample.wav')

pYAAPT.PitchObj.PITCH_DOUBLE = 2      # set new values
pYAAPT.PitchObj.PITCH_HALF = 2
pYAAPT.PitchObj.SMOOTH_FACTOR = 3

pitch = pYAAPT.yaapt(signal) # calculate the pitch track
```

PITCH OBJECT ATTRIBUTES:

PitchObj.**nfft**
Length in samples from the FFT used by the pitch tracker. It is set during the object's initialization.

PitchObj.**frame_size**
Length in samples from the frames used by the pitch tracker. It is set during the object's initialization.

PitchObj.**frame_jump**
Distance in samples between the center of a extracting frame and the center of its adjacent neighbours. It is set during the object's initialization.

PitchObj.**noverlap**
It's the difference between the frame size and the frame jump. Represents the number of samples that two adjacent frames share in common, i.e, how much they overlap each other. It is set during the object's initialization.

PitchObj.**mean_energy**
Signal's low frequency band mean energy. It is set by the PitchObj.set_energy method.

PitchObj.**energy**
Array that contains the low frequency band energy from each frame, normalized by PitchObj.mean_energy. It is set by the PitchObj.set_energy method.

PitchObj.**vuv**
Boolean vector that indicates if each speech frame was classified as voiced (represented as 'True') or unvoiced (represented as 'False'). It is set by the PitchObj.set_energy method.

PitchObj.**frames_pos**
A numpy array that contains the samples where the center of the frames were placed during the extraction. It is set by the PitchObj.set_frame_pos method.

[1] don't mistake this interpolation with the one performed by the pYAAPT.upsample method. For more explanation, please refer to the pYAAPT.samp_interp and pYAAPT.values_interp attributes.

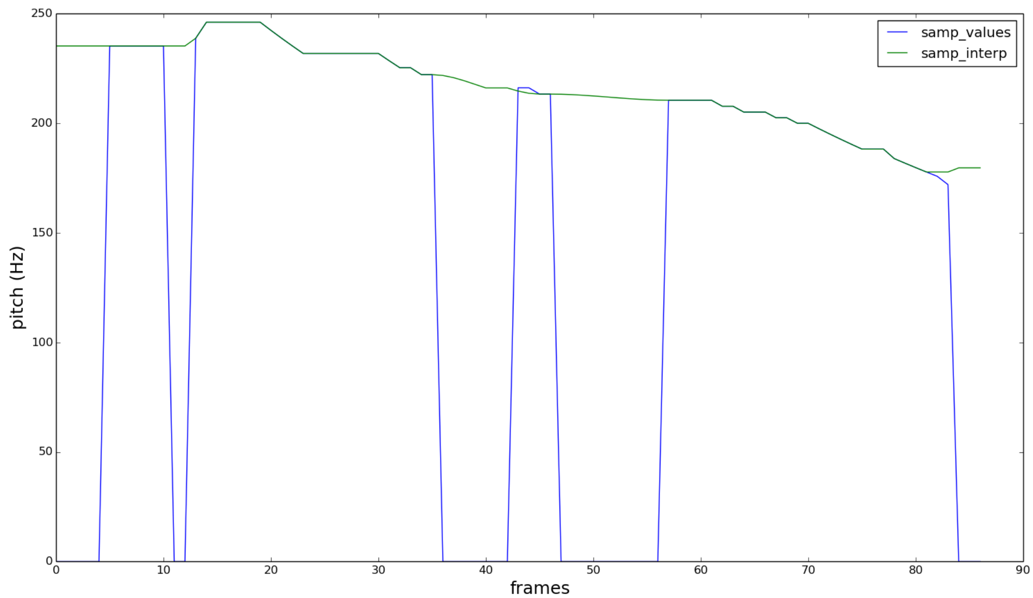
`PitchObj.`**`nframes`**

Number of frames. It is set by the PitchObj.set_frame_pos method.

`PitchObj.`**`samp_values`**

`PitchObj.`**`samp_interp`**

Both arrays contain the pitch values from each of the nframes. The only difference is that, in PitchObj.samp_interp the unvoiced segments are substituted by the interpolation from the adjacent voiced segments edges. This provides a non-zero version from the pitch track, which can be necessary for some applications. The figure below presents both arrays from the sample.wav file:

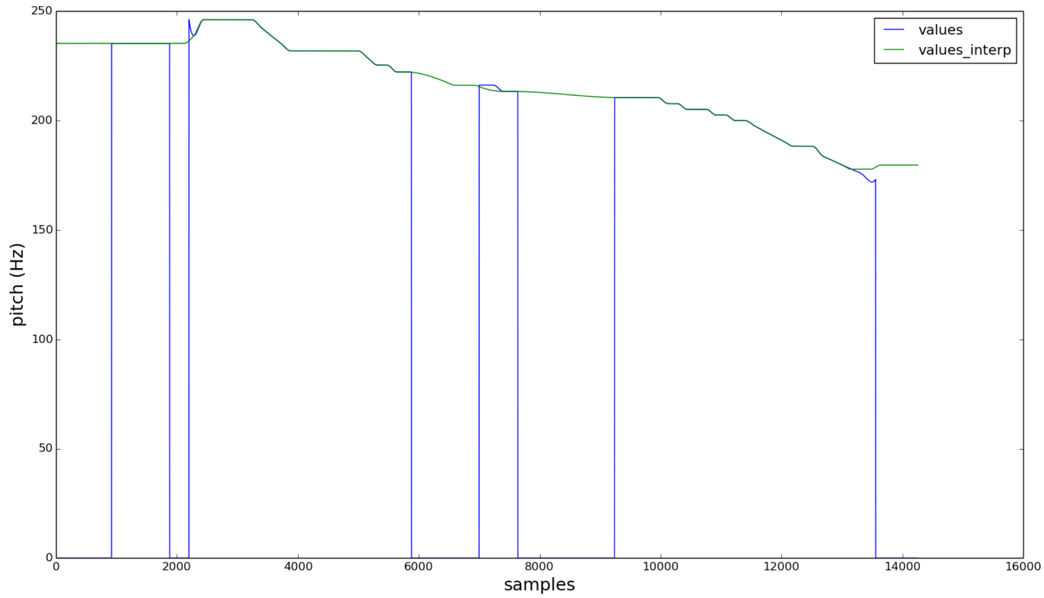


Both attributes are set by the PitchObj.set_values method.

`PitchObj.`**`values`**

`PitchObj.`**`values_interp`**

PitchObj.values and PitchObj.values_interp are the upsampled versions from PitchObj.samp_values and PitchObj.samp_interp respectively. Therefore, their length is equal to the original file length (for more information, check the PitchObj.upsample() method). The figure below presents both arrays from the sample.wav file:



Both attributes are set by the PitchObj.set_values method.

PitchObj.**edges**

A list that contains the index where occur the transitions between unvoiced-voiced and voiced-unvoiced in PitchObj.values. It is set by the PitchObj.set_values method, which employs internally the PitchObj.edges_finder method.

PITCH OBJECT METHODS:

PitchObj.**set_energy**(*energy*, *threshold*)

Parameters

- **energy** (*numpy array*) – contains the low frequency energy for each frame.
- **threshold** – normalized threshold.

Set the normalized low frequency energy by taking the input array and dividing it by its mean value. Normalized values above the threshold are considered voiced frames, while the ones below it are unvoiced frames.

PitchObj.**set_frames_pos**(*frames_pos*)

Parameters **frames_pos** – index with the sample positions.

Set the position from the center of the extraction frames.

PitchObj.**set_values**(*samp_values*, *file_size*[, *interp_tech='spline'*])

Parameters

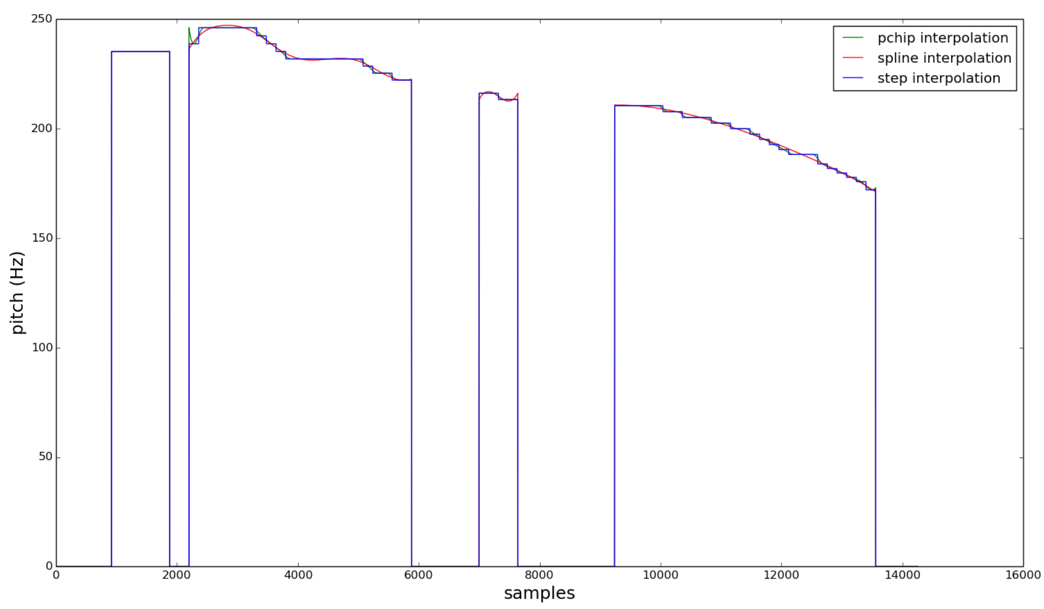
- **samp_values** (*numpy array*) – pitch value for each frame.
- **file_size** (*int*) – length of the speech signal.
- **interp_tech** (*string*) – interpolation method employed to upsample the data. Can be 'pchip' (default), 'spline' and 'step'.

Set the pitch values and also calculates its interpolated version (for more information, check the PitchObj.samp_values and PitchObj.samp_interp attributes). A post-process is employed then using the PitchObj class attributes. After that, both arrays are upsampled, so that the output arrays have the same length as the original speech signal. In this process, a second interpolation is necessary. The interpolation technique employed is indicated by the parameter interp_tech.

Example:

```
import amfm_decompy.pYAAPT as pYAAPT
import amfm_decompy.basic_tools as basic
from matplotlib import pyplot as plt

signal = basic.SignalObj('path_to_sample.wav')
pitch = pYAAPT.yaapt(signal)

plt.plot(pitch.values, label='pchip interpolation', color='green')

pitch.set_values(pitch.samp_values, len(pitch.values), interp_tech='spline')
plt.plot(pitch.values, label='spline interpolation', color='red')

pitch.set_values(pitch.samp_values, len(pitch.values), interp_tech='step')
plt.plot(pitch.values, label='step interpolation', color='blue')

plt.xlabel('samples', fontsize=18)
plt.ylabel('pitch (Hz)', fontsize=18)
plt.legend(loc='upper right')
```

The output is presented below:



PitchObj.**edges_finder**(*values*)

Parameters **values** (*numpy array*) – contains the low frequency energy for each frame.

Return type list.

Returns the index of the samples where occur the transitions between unvoiced-voiced and voiced-unvoiced.

BandpassFilter Class

Creates a bandpass filter necessary for the YAAPT algorithm.

USAGE:

`amfm_decompy.pYAAPT.`**`BandpassFilter`**(*fs*, *parameters*)

Parameters

- **`fs`** (*float*) – signal's fundamental frequency
- **`parameters`** (*dictionary*) – contains the parameters options from the YAAPT algorithm.

Return type bandpass filter object.

BANDPASS FILTER ATTRIBUTES:

`BandpassFilter.`**`b`**

Bandpass filter zeros coefficients. It is set during the object's initialization.

`BandpassFilter.`**`a`**

Bandpass filter poles coefficients. It is set during the object's initialization.

`BandpassFilter.`**`dec_factor`**

Decimation factor used for downsampling the data. It is set during the object's initialization.

CHAPTER

TWO

PYQHM

The algorithms here implemented were the QHM (Quasi-Harmonic Model), and its upgrades, aQHM (adaptive Quasi-Harmonic Model) and eaQHM (extended adaptive Quasi-Harmonic Model). Their formulation can be found at references *[ref2]*, *[ref3]* and *[ref4]*.

These algorithms perform the so-called AM-FM decomposition. This designation is used due the fact that, in this method, the signal is modeled as a sum of amplitude- and frequency-modulated components. The goal is to overcome the drawbacks from Fourier-alike techniques, e.g. SFFT, wavelets, etc, which are limited in the time-frequency analysis by the so-called Heisenberg-Gabor inequality.

Quick start

The pyQHM module provides a function for each of the QHM family algorithms:

USAGE:

`amfm_decompy.pyQHM.`**`qhm`**(*signal*, *pitch*, *window*[, *samp_jump=None*, *N_iter=1*, *phase_tech='phase'*])

`amfm_decompy.pyQHM.`**`aqhm`**(*signal*, *previous_HM*, *pitch*, *window*[, *samp_jump=None*, *N_iter=1*, *N_runs=float('Inf')*, *phase_tech='phase'*])

`amfm_decompy.pyQHM.`**`eaqhm`**(*signal*, *previous_HM*, *pitch*, *window*[, *samp_jump=None*, *N_iter=1*, *N_runs=float('Inf')*, *phase_tech='phase'*])

Parameters

- **`signal`** (*`signal object`*) – contains the signal data and its parameters.
- **`pitch`** (*`pitch object`*) – contains the pitch track and its parameters.
- **`window`** (*`window object`*) – contains the sample window and some reference arrays.
- **`samp_jump`** (*`float`*) – distance in seconds between the center of a extracting frame and the center of its adjacent neighbours (default: sample by sample).
- **`N_iter`** (*`int`*) – number of iterations for each frame estimation (default: 1).
- **`phase_tech`** (*`str`*) – has two options: 'phase' (default) and 'freq'. The objective is to choose the smoother base for further aQHM and eaQHM calculations in order to avoid the degradation of their performance due the phase wild behaviour. Normally when a sample jump is employed, the 'phase' option it's enough, since that the interpolation process already smooths the phase signal. However, in a sample by sample analysis, the use of 'freq' (cumulative frequency) is favoured.
- **`previous_HM`** (*`modulated signal object`*) – previously extracted AM-FM signal, used as base for the aQHM and eaQHM calculations.

- **N_runs** (*int*) – after the aQHM/eaQHM algorithm has been applied on the whole signal, the function takes the output modulated signal object as new input and restart the aQHM/eaQHM until N_runs are performed OR until the output SRER (Signal-to-Reconstruction Error Ratio) stops growing. The goal is to refine the results. (default: keeps restarting the algorithm infinitely until the maximum SRER).

Return type modulated signal object

EXAMPLES:

Example 1 - the parameters of a speech signal are extracted sample by sample through QHM. After that, its output is used as input for the first of two aQHM runs with 1 ms sample jump. Finally, the result is used to start one run of the eaQHM with a 1 ms sample jump again. The three algorithms perform 3 iterations per frame extraction.:

```
import amfm_decompy.pYAAPT as pyaapt
import amfm_decompy.pyQHM as pyqhm
import amfm_decompy.basic_tools as basic

# Declare the variables.
window_duration = 0.015
nharm_max = 25

# Create the signal object.
signal = basic.SignalObj('path_to_sample.wav')

# Create the window object.
window = pyqhm.SampleWindow(window_duration, signal.fs)

# Create the pitch object and calculate its attributes.
pitch = pyaapt.yaapt(signal)

# Use the pitch track to set the number of modulated components.
signal.set_nharm(pitch.values, nharm_max)

# Perform the QHM extraction.
QHM = pyqhm.qhm(signal, pitch, window, N_iter = 3, phase_tech = 'freq')

# Perform the aQHM extraction.
aQHM = pyqhm.aqhm(signal, QHM, pitch, window, 0.001, N_iter = 3, N_runs = 2)

# Perform the eaQHM extraction.
eaQHM = pyqhm.eaqhm(signal, aQHM, pitch, window, 0.001, N_iter=3, N_runs=1)
```

Classes

ModulatedSign Class

The ModulatedSign Class stores the extracted modulated signal and all the parameters related to it. The data structure provided by this class is used by all the QHM algorithms, since that the model for a modulated signal is basically the same for all of them.

USAGE:

`amfm_decompy.pyQHM.`**`ModulatedSign`**(*n_harm*, *file_size*, *fs*[, *phase_tech='phase'*])

Parameters

- **n_harm** (*int*) – number of modulated components that form the signal.

- **file_size** (*int*) – length of the speech signal in samples.
- **fs** (*float*) – sampling frequency in Hz.
- **phase_tech** (*str*) – has two options: 'phase' (default) and 'freq'. The objective is to choose the smoother base for further aQHM and eaQHM calculations in order to avoid the degradation of their performance due the phase wild behaviour. Normally when a sample jump is employed, the 'phase' option it's enough, since that the interpolation process already smooths the phase signal. However, in a sample by sample analysis, the use of 'freq' (cumulative frequency) is favoured.

Return type modulated signal object.

MODULATED SIGNAL ATTRIBUTES:

`ModulatedSign.`**`n_harm`**
Number of modulated components that form the signal. It is set during the object's initialization.

`ModulatedSign.`**`size`**
Length of the speech signal in samples. It is set during the object's initialization.

`ModulatedSign.`**`fs`**
Sampling frequency in Hz. It is set during the object's initialization.

`ModulatedSign.`**`H`**
3-dimension numpy array (n_harm, 3, file_size), which stores the magnitude, phase and frequency values from all components. Its first dimension refers to the n_harm components, the second to the three composing parameters (where 0 stands for the magnitude, 1 for the phase and 2 for the frequency) and the third dimension to the temporal axis. It is created during the object's initialization.

`ModulatedSign.`**`harmonics`**
List where each element is a modulated component. Read more about it in the ComponentObj Class section. It is created during the object's initialization.

`ModulatedSign.`**`error`**
Numpy array where each element is the mean squared error between the original signal frame and its synthesized version. It is created during the object's initialization.

`ModulatedSign.`**`phase_tech`**
Name of the phase smoothing method used to create a reference for future aQHM/eaQHM calculations. Can be 'phase' or 'freq'. It is set during the object's initialization.

`ModulatedSign.`**`signal`**
Final signal synthesized with the extracted parameters. It is created by the ModulatedSign.synthesize method.

`ModulatedSign.`**`SRER`**
Signal-to-Reconstruction Error Ratio, measures the similarity between the original signal and its synthesized version. The bigger its value, the better the reconstruction. It is calculated by the ModulatedSign.srer method.

`ModulatedSign.`**`extrap_phase`**
2-dimension numpy array (n_harm, file_size) which contains a modified version of the extracted phase track from each component. The signals are smoothed (check the ModulatedSign.phase_tech attribute) and their edge values are extrapolated for future aQHM/eaQHM runs. It is calculated by the ModulatedSign.phase_edges method.

MODULATED SIGNAL METHODS:

`ModulatedSign.`**`update_values`**(*a*, *freq*, *frame*)

Parameters

- **a** (*numpy array*) – contains the extracted complex coefficients from the harmonic model (for more information about them, please check the references).
- **freq** (*numpy array*) – instantaneous frequency from each of the components.
- **frame** (*int*) – sample where the center of the moving sample window is located.

Updates the values of magnitude, phase and instantaneous frequency in the H matrix.

ModulatedSign.**interpolate_samp**(*samp_frames*, *pitch*)

Parameters

- **samp_frames** (*numpy array*) – contains the sample locations where the algorithm was employed.
- **pitch** (*pitch object*) – pitch information.

Interpolate the parameters values when the extraction is not performed sample-by-sample.

ModulatedSign.**synthesize**([*N=None*])

Parameters **N** – select which of the components are going to be synthesized (default: all of them).

Runs the ComponentObj.synthesize method for each of the n_harm components, and after that, sum them to construct the final synthesized signal.

ModulatedSign.**srer**(*orig_signal*, *pitch_track*)

Parameters

- **orig_signal** (*numpy array*) – original signal.
- **pitch_track** (*numpy array*) – pitch values for each sample.

Calculates the SRER (Signal-to-Reconstruction Error Ratio) for the synthesized signal. It is defined mathematically as

20*log10(std(orig_signal) / std(orig_signal - synth_signal)).

ModulatedSign.**phase_edges**(*edges*, *window*)

Parameters

- **edges** – index where occur the pitch transitions between unvoiced-voiced and voiced-unvoiced.
- **window** (*window object*) – sample window and its parameters.

Extrapolates the phase at the border of the voiced frames by integrating the edge frequency value. This procedure is necessary for posterior aQHM calculations. Additionally, the method allows the replacement of the extracted phase by the cumulative frequency. The objective is to provide smoother bases for further aQHM and eaQHM calculations. Normally this is not necessary, since that the interpolation process already smooths the phase vector. But in a sample-by-sample extraction case, this substitution is very helpful to avoid the degradation of aQHM and eaQHM performance due the phase wild behaviour.

ComponentObj Class

Creates a single component object, whose data is stored in the ModulatedSign.H matrix. The ComponentObj Class provides thus an alternative interface to separately access and manipulate each component.

USAGE:

amfm_decompy.pyQHM.**ComponentObj**(*H*, *harm*)

Parameters

- **H** (*numpy array*) – 3-dimensional array where the component data is stored (for more information, check the ModulatedSign.H attribute).
- **harm** (*int*) – the component index.

Return type component object.

MODULATED COMPONENT ATTRIBUTES:

ComponentObj.**mag**
Magnitude envelope of the component. It is set during the object's initialization.

ComponentObj.**phase**
Phase angle track of the component in radians. It is set during the object's initialization.

ComponentObj.**freq**
Instantaneous normalized frequency track of the component. To get the value in Hz just multiply this array by the sample frequency. It is set during the object's initialization.

ComponentObj.**signal**
Component signal synthesized with the extracted parameters. It is created by the ComponentObj.synthesize method.

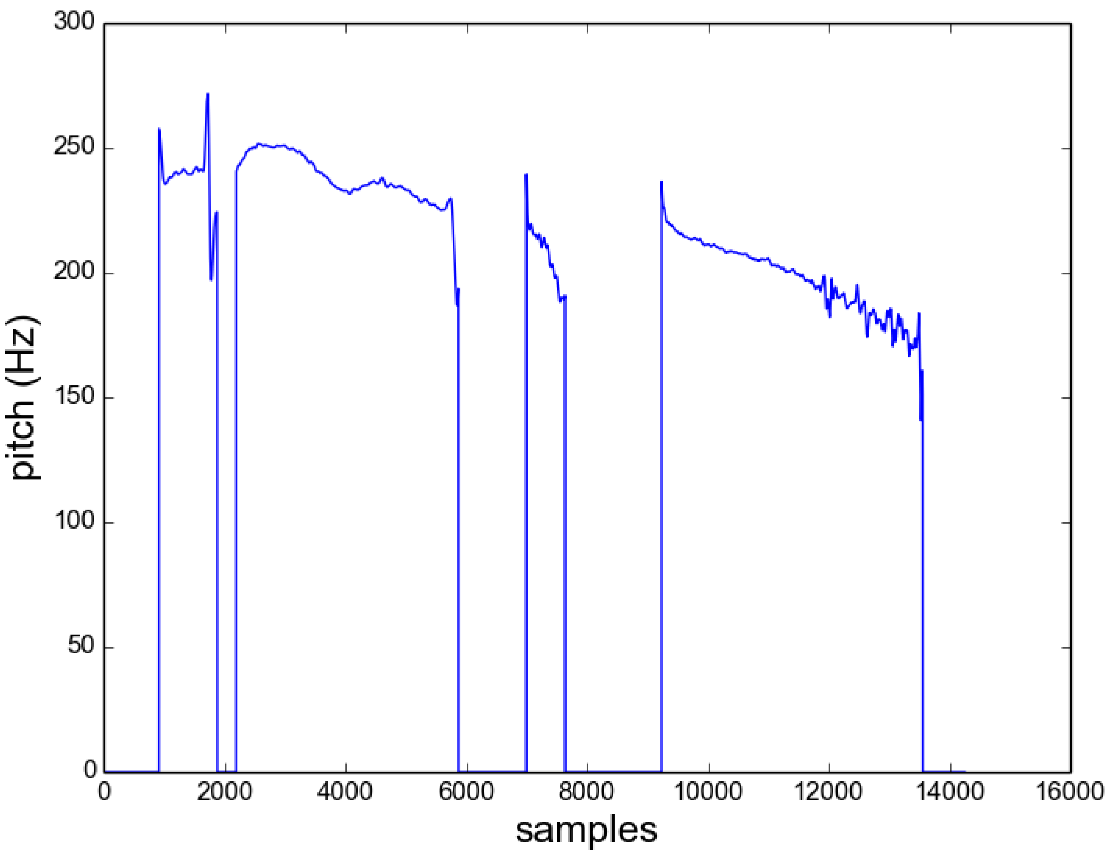
EXAMPLES:

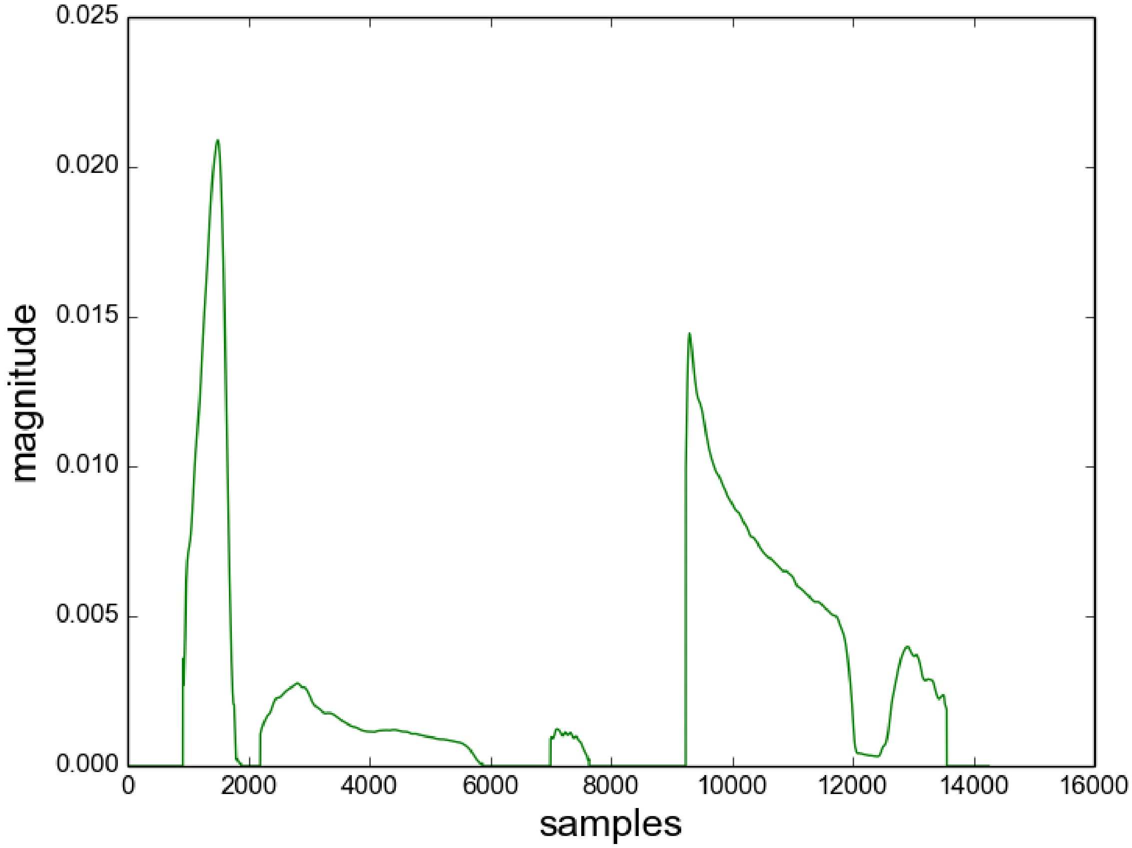
Example 1 - Shows how to to access the component data of a specific component:

```
import amfm_decompy.pYAAPT as pyaapt
import amfm_decompy.pyQHM as pyqhm
import amfm_decompy.basic_tools as basic
from matplotlib import pyplot as plt

# Declare the variables.
window_duration = 0.015
nharm_max = 25

# Create the signal object.
signal = basic.SignalObj('path_to_sample.wav')

# Create the window object.
window = pyqhm.SampleWindow(window_duration, signal.fs)

# Create the pitch object and calculate its attributes.
pitch = pyaapt.yaapt(signal)

# Use the pitch track to set the number of modulated components.
signal.set_nharm(pitch.values, nharm_max)

# Perform the QHM extraction.
QHM = pyqhm.qhm(signal, pitch, window, 0.001, N_iter = 3)

fig1 = plt.figure()

# Plot the instaneous frequency of the fundamental harmonic.
# The ComponentObj objects are stored inside the harmonics list.
# For more information, please check the ModulatedSign.harmonics attribute.
plt.plot(QHM.harmonics[0].freq*signal.fs)
```

```
plt.xlabel('samples', fontsize=18)
plt.ylabel('pitch (Hz)', fontsize=18)

fig2 = plt.figure()

# Plot the envelope magnitude of the third harmonic.
# The ComponentObj objects are stored inside the harmonics list.
# For more information, please check the ModulatedSign.harmonics attribute.
plt.plot(QHM.harmonics[2].mag, color='green')

plt.xlabel('samples', fontsize=18)
plt.ylabel('magnitude', fontsize=18)
```

The results are presented in the next two pictures:





NOTE: It must noticed that the ComponentObj can be normally sliced. For example:

```
QHM.harmonics[0].freq[920:1000]
```

will return a array containing only the segment of the fundamental frequency between the samples from 920 to 999, while:

```
QHM.harmonics[2].mag[950]
```

will return the magnitude of the third harmonic at the 950th sample. However, due the way that the python language is internally built, unfortunately it's not possible to slice the harmonics list. For example:

```
QHM.harmonics[0:3].freq[920:1000]
QHM.harmonics[0:2].mag[950]
```

will raise an ERROR message. Therefore, the only way to simultaneously get the data of a group of components is by directly accessing the ModulatedSign.H matrix (or using a foor loop, but this option is slower):

```
QHM.H[0:3, 2, 920:1000]
QHM.H[0:2, 0, 950]
```

MODULATED COMPONENT METHODS:

`ComponentObj.synthesize()`
Synthsize the modulated component by using the extracted magnitude and phase.

SampleWindow Class

Creates the sample hamming window object and some related index arrays.

USAGE:

`amfm_decompy.pyQHM.SampleWindow(window_duration, fs)`

Parameters

- **window_duration** (*float*) – window duration in seconds.
- **fs** (*float*) – sample frequency in Hz.

Return type sample window object.

SAMPLE WINDOW ATTRIBUTES:

`SampleWindow.dur`
Window duration in seconds. It is set during the object's initialization.

`SampleWindow.length`
Window length in samples. It is set during the object's initialization.

`SampleWindow.data`
Array containing the hamming window data. It is set during the object's initialization.

`SampleWindow.data2`
Array containing the hamming window data with each element raised to the 2 power. It is set during the object's initialization.

`SampleWindow.N`
Half-window length, i.e., SampleWindow.length/2 -1. It is set during the object's initialization.

`SampleWindow.half_len_vec`
Numpy array that contains the indexes from zero to N, i.e, [0, 1 ... N]. It is set during the object's initialization.

`SampleWindow.len_vec`
Numpy array that contains the indexes from -N to N, i.e, [-N, -N+1 ... N-1, N]. It is set during the object's initialization.

CHAPTER

THREE

BASIC_TOOLS

This module contains a set of basic classes and functions that are commonly used by the other modules of the package.

Classes

SignalObj Class

The SignalObj Class stores the speech signal and all the parameters related to it.

USAGE:

`amfm_decompy.basic_tools.`**`SignalObj`**`(`*args*`)`

Parameters **`args`** – the input argument can be a string with the wav file path OR a tuple containing the speech signal data and its fundamental frequency in Hz.

Return type speech signal object.

SIGNAL OBJECT ATTRIBUTES:

`SignalObj.`**`data`**
Numpy array containing the speech signal data. It is set during the object's initialization.

`SignalObj.`**`fs`**
Sample frequency in Hz. It is set during the object's initialization.

`SignalObj.`**`size`**
Speech signal length. It is set during the object's initialization.

`SignalObj.`**`filtered`**
Bandpassed version from the speech data. It is set by the SignalObj.filtered_version method.

`SignalObj.`**`new_fs`**
Downsampled fundamental frequency from the speech data. It is set by the SignalObj.filtered_version method.

`SignalObj.`**`clean`**
When the SignalObj.noiser method is called, this attribute is created and used to store a clean copy from the original signal.

SIGNAL OBJECT METHODS:

`SignalObj.`**`filtered_version`**`(`*bp_filter*`)`

Parameters **`bp_filter`** – BandpassFilter object.

Filters the signal data by a bandpass filter.

SignalObj.**set_nharm**(*pitch_track*, *n_harm_max*)

Parameters

- **pitch_track** (*numpy array*) – pitch extracted values for each signal sample.
- **n_harm_max** (*int*) – represents the maximum number of components that can be extracted from the signal.

Uses the pitch values to estimate the number of modulated components in the signal.

SignalObj.**noiser**(*pitch_track*, *SNR*)

Parameters

- **pitch_track** (*numpy array*) – pitch extracted values for each signal sample.
- **SNR** (*float*) – desired signal-to-noise ratio from the output signal.

Adds a zero-mean gaussian noise to the signal.

Functions

pcm2float

USAGE:

amfm_decompy.basic_tools.**pcm2float**(*sig*[, *dtype=numpy.float64*])

Parameters

- **sig** (*numpy array*) – PCM speech signal data.
- **dtype** (*float*) – data type from the elements of the output array (default: numpy.float64).

Return type numpy array.

Transform a PCM raw signal into a float one, with values limited between -1 and 1.

[ref1] Stephen A. Zahorian, and Hongbing Hu, "A spectral/temporal method for robust fundamental frequency tracking," J. Acosut. Soc. Am. 123(6), June 2008.

[ref2] Y.Pantazis, "Decomposition of AM-FM signals with applications in speech processing", PhD Thesis, University of Creta, 2010.

[ref3] Y.Pantazis, O. Rosec and Y. Stylianou, "Adaptive AM-FM signal decomposition with application to speech analysis", IEEE Transactions on Audio, Speech and Language Processing, vol. 19, n 2, 2011.

[ref4] G.P. Kafentzis, Y. Pantazis, O. Rosec and Y. Stylianou, "An extension of the adaptive quasi-harmonic model", em IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2012.

U

V

Y