Stationary and non-stationary sinusoidal model synthesis with phase vocoder and FFT^{-1}

Clément Cazorla, Vincent Chrun, Bastien Fundaro, Clément Maliet
ENSEEIHT - 3EN TSI 2016-2017

March 6, 2017

Abstract

The present document is to serve as both a technical report and a documentation for the code produced during the long project. As such the first part will explain the theoretical framework and the state-of-the-art of the field. The second part will give some insight about the code structure and the conventions that were adopted and last but not least, the third part will serve as an actual documentation and details every class, methods and attributes.

Contents

Part I
Theoretical framework

1 Sound synthesis

1.1 Frequency domain

1.2 Sinusoidal model

1.3 Phase Vocoder

2 Theoretical synthesis with phase vocoder and FFT^{-1}

2.1 Stationary case

2.2 Non-stationary case

Part II

Code structure and conventions

3 Conventions

In this section we remind the reader of a few coding convention necessary to ensure a seamless work flow and a bug free program as much as possible. Files should contains an entire module (as described in 4.1) not just a single class to limit the number of files and ease the bug tracking. Imports in files should be kept to a minimum and left in namespaces (do not use the **from** module **import** * syntax). It is preferable to import a whole module if more than three elements from the module are needed in the file, otherwise consider the **from** module **import** element1 , element2 syntax to avoid unnecessary memory flooding. If conflicts exists, notwithstanding the number of elements needed, the whole modules are to be loaded with a namespace. Namespaces may be abbreviated to the programmer's convenience however some abbreviation are to be universally respected :

(i) `numpy` should always be imported as `np`

(ii) `matplotlib.pyplot` should always be imported as `plt`

Finally math functions should always come from the numpy module and not python's math module to guarantee a universal behaviour across the program.

3.1 Naming conventions

Naming conventions are freely adapted from Python recommended conventions defined in PEP8 [1], as such :

(i) *Class* should be named in `CapitalizedWord`

(ii) *Methods* and *functions* should be named in `lower_case_with_underscores`

(iii) *Attributes* and *variables* should be names in `lower_case_with_underscores`

(iv) *Instantiation* following the fact that everything is an object in python should be named as *variables*.

Moreover during class declaration, the following principles should be adopted :

(i) Non-public methods and attributes should use one leading underscore.

(ii) Elements that conflicts with python reserved name should use one trailing underscore rather that simplification or a misspelling.

(iii) Accessors or mutators using one leading underscore should be interpreted as properties of their associated attribute. As such it should be guaranteed that they induce a low computational cost.

(iv) Non-public elements that should not be inherited or may cause conflicts during inheritance should use two leading underscore and make use of Python name-mangling.

To seamlessly manipulate both *stationary* and *non stationary* models, class that are inherited in two versions are preceded with either `Stationary` are `NonStationary` respectively.

3.2 Spectrum and sinusoids parameters classes

Because many spectra, main lobes and sinusoidal model parameters have to be traded between modules we created two classes, respectively `Spectrum` and `Parameters`. They mainly serve as containers, holding the data and returning them in a point wise fashion. This way we can prevent conflicts and errors that would come from a non-uniform data sharing protocol and as well ensure that every operation performed on either spectra or parameters are made following the same principles and algorithms.

3.2.1 Spectrum

```
Spectrum
---------------------------------------------------------------
_amplitude : np.array
_phase : np.array
_nfft : int
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
__init__(self, amplitude, phase)
__add__(self, other)
__iadd__(self, other)
__mul__(self, other)
__imul__(self, other)
from_complex_spectrum(cls, complex_spectrum)  @classmethod
void_spectrum(cls)                            @classmethod
set_spectrum(self, amplitude, phase,
      start_bin=None, stop_bin=None)
set_complex_spectrum(self, complex_spectrum,
      start_bin=None, stop_bin=None)
get_amplitude(self, k)
get_phase(self, k)
get_nfft(self)
```

The Spectrum class stores a spectrum in amplitude and phase, however it may be created or changed from a complex `np.array` respectively with the class method `from_complex_spectrum` and the method `set_complex_spectrum`. Those two methods may take optional parameters start_bin and stop_bin if one need to update only a part of the spectrum, for example a single lobe. The class checks that the given data are consistent upon instantiation. The $+$ operation as well as the $+=$ operation have been defined between two `Spectrum` objects and between a `Spectrum` object and an array of complex numbers.

The $\times$ operation as well as the $\times =$ operation have been defined between a `Spectrum` object and an array of complex numbers. Addition and multiplication attempts between other data type will result in a `NotImplementedError` exception.

3.2.2 Parameters

```
Parameters
-----------------------------------------------------
_amplitudes :  np.array
_frequencies :  np.array
_phases :  np.array
_number_sinuses :  int
- - - - - - - - - - - - - - - - - - - - - - - - - - -
__init__(self, amplitudes, frequencies, phases)
get_amplitude(self, k)
get_frequency(self, k)
get_phase(self, k)
get_number_sinuses(self)
```

The Parameters class is more of a structure than a class and only contains the stationary sinusoidal model parameters and their respective accessors. It also stores the number of sinuses and checks that the given data are consistent upon instantiation.
In the stationary sinusoidal model the signal $s(t)$ is defined as follow[1] :

$$s(n) = \sum_{i=1}^{N_{sinus}} \alpha_i \sin(2\pi \bar{f}_i n + \phi_i)$$

with $\bar{f}_i = \frac{f_i}{f_s}$ the normalised frequency.
We then store the parameters as follows :

_amplitudes stores the α_i

_frequencies stores the $\bar{f}_i$

_phases stores the ϕ_i

3.2.3 NonStationaryParameters

```
NonStationaryParameters(Parameters)
-----------------------------------------------------
_acrs :  np.array
_fcrs :  np.array
- - - - - - - - - - - - - - - - - - - - - - - - - - -
__init__(self, amplitudes, frequencies, phases, acrs, fcrs)
get_acr(self, k)
get_fcr(self, k)
```

The stationary sinusoidal model can be extended to the first order development to better model fast amplitude and frequency change over time. The signal $s(t)$ can then be expressed as a sum of linearly varying chirps[1] :

$$s(n) = \sum_{i=1}^{N_sinus} (\alpha_i + \mu_i \cdot nT_s) \sin(2\pi \bar{f}_i n + \frac{\psi_i}{2}(nT_s)^2 + \phi_i)$$

Where we define the **Amplitude Change Rate** μ and the **Frequency Change Rate** ψ.
Thus we inherit the `Parameters` class to add the two additional parameters as follow :

_acrs stores the μ_i

_fcrs stores the ψ_i

4 Code structure

4.1 General structure

4.2 Class structure

[1]Please look up section 1.2 page 3 for more details

Part III

Documentation

5 Core module

5.1 Synthesizer

5.2 StationarySynthesizer

5.3 NonStationarySynthesizer

6 Spectrum generation module

6.1 SpectrumGenerator

6.2 StaionarySpectrumGenerator

6.3 NonStationarySpectrumGenerator

6.4 StationnaryLobe

6.5 NonStationaryLUT

7 Phase Vocoder module

7.1 PhaseVocoder

7.2 StationaryPhaseVocoder

7.3 NonStationaryPhaseVocoder

Conclusion

References

[1] N. C. Guido van Rossum, Barry Warsaw, “Style guide for python code,” Aug. 2013.