Proc. of the 12th Int. Conference on Digital Audio Effects (DAFx-09), Como, Italy, September 1-4, 2009

SIMPL: A PYTHON LIBRARY FOR SINUSOIDAL MODELLING

John Glover, Victor Lazzarini, Joseph Timoney

The Sound and Digital Music Research Group
National University of Ireland, Maynooth
Ireland

John.C.Glover@nuim.ie
Victor.Lazzarini@nuim.ie
JTimoney@cs.nuim.ie

ABSTRACT

This paper introduces Simpl, a new open source library for sinusoidal modelling written in Python. The library is presented as a resource for researchers in spectral signal processing, who might like to access existing methods and techniques. The text provides an overview of the design of the library, describing its data abstractions and integration with other systems. This is complemented by some brief examples exploring the functionality of the library.

1.INTRODUCTION

Simpl is an open source library for sinusoidal modelling [1] written in the Python programming language [2] and making use of Scientific Python (SciPy) [3]. The aim of this project is to tie together many of the existing sinusoidal modelling implementations into a single unified system with a consistent API, as well as providing implementations of some recently published sinusoidal modelling algorithms, many of which have yet to be released in software.

Simpl is primarily intended as a tool for other researchers in the field, allowing them to easily combine, compare and contrast many of the published analysis/synthesis algorithms. There are currently several open source software projects that either include or are dedicated solely to sinusoidal modelling such as PARSHL [4], the Sound Object Library [5], Csound [6], Loris [7], CLAM [8], libsms [9] and SAS [10]. All of these systems exist as separate entities, and due to their internal workings it can often be awkward to exchange analysis data between them for comparison. However, they generally share common ideas, terminology and abstractions (such as the concepts of spectral peaks and partial tracking). Simpl allows these abstract data types to be exchanged between different underlying implementations. For example, one might wish to compare the sinusoidal peaks detected with the SMS algorithm with those found by the Loris implementation. Due to the flexible, modular design of Simpl this sort of operation is straight-forward. Simpl analysis/synthesis is able to render audio files in non-real-time as well as operate in real-time streaming mode, as long as the underlying algorithms are able to do so.

1.1.Sinusoidal Modelling

Sinusoidal modelling is based on Fourier's theorem, which states that any periodic waveform can be modelled as the sum of sinusoids at various amplitudes and harmonic frequencies. For stationary pseudo-periodic sounds, these amplitudes and frequencies evolve slowly with time. They can be used as parameters to control pseudo-sinusoidal oscillators, commonly referred to as *partials*. The audio signal s can be calculated from the sum of the partials using:

$$s(t)=\sum_{1}^{N_p} A_{p(t)}\cos\left(\theta_{p(t)}\right) \tag{1}$$

$$\theta_{p(t)}=\theta_{p(0)}+2\pi\int_{0}^{t} f_{p(u)}\,du \tag{2}$$

where N_p is the number of partials and A_p, f_p and Θ_p are the amplitude, frequency and phase of the p-th partial respectively. Typically, the parameters are measured for every:

$$t=nH/F_s \tag{3}$$

where n is the sample number, H is the hop size and F_s is the sampling rate. To calculate the audio signal, the parameters must then be interpolated between measurements. Calculating these parameters for each frame is referred to in this document as *peak detection*, while the process of connecting these peaks between frames is called *partial tracking*.

In [11] McAulay and Quatieri proposed to represent a speech signal as a sum of sinusoids with time-varying amplitude, frequency and phase. While it is possible to model noisy signals with sinusoids, it is not very efficient, as large numbers of partials are often required. It is also not particularly meaningful, and does not seek to represent the underlying structure of the sound.

Serra and Smith extended this idea in [12], making a distinction between the pseudo-periodic or *deterministic* component of a sound and the more noise-like or *stochastic* component, modelling and synthesising the two components separately. Fitz and Haken keep this distinction in [13], but use *bandwidth-enhanced* oscillators to create a homogeneous additive sound model.

Later advances and refinements in the field have mostly been in the details of the analysis algorithms, in particular in the peak detection and partial tracking processes. A good overview of peak detection techniques can be found in [14]. In [15] Depalle

and Rodet use the Hidden Markov Model to improve partial tracking, while in [16] Lagrange et al achieve this using Linear Prediction.

1.2.SciPy

SciPy is a cross-platform, open source software package for mathematics, science and engineering. It depends on NumPy [17], which provides fast array processing. It has a syntax that is very similar to Matlab [18], with implementations of many of Matlab's functions: it contains packages for matrix manipulation, statistics, linear algebra as well as signal processing. SciPy also supports Matlab-style plotting and visualisation of data through the Matplotlib [19] language extension. The vast library of functions combined with the readability and power of the Python language make SciPy a great tool for quick prototyping as well as for the development of larger applications.

2.THE SIMPL LIBRARY

Simpl is an object-orientated Python library for sinusoidal modelling. Spectral data is represented by two main object types: Peak (represents a spectral peak) and Partial. A Partial is basically just an ordered collection of Peak objects.

Simpl includes a module with plotting functions that use Matplotlib to plot analysis data from the peak detection and partial tracking analysis phases, but generating additional plots is trivial using Matplotlib's Matlab-like interface.

All audio in Simpl is stored in NumPy arrays. This means that SciPy functions can be used for basic tasks such as reading and writing audio files, as well as more complex procedures such as performing additional processing, analysis or visualisation of the data.

Each supported analysis/synthesis method has associated wrapper objects that allows it to be used with Simpl Peak and Partial data, which facilitates the exchange of information between what were originally unrelated sinusoidal modelling systems. The implementations that are currently supported are the Sound Object Library, Spectral Modelling Synthesis (SMS, using libsms) and Loris. Additionally, the following algorithms are included: McAulay-Quatieri (MQ) analysis and synthesis as given in [11], partial tracking using the Hidden Markov Model (HMM) as detailed in [15] and partial tracking using Linear Prediction (LP) as detailed in [16].

Currently in Simpl the sinusoidal modelling process is broken down into three distinct steps: peak detection, partial tracking and sound synthesis. Python objects exist for each step, which all of the analysis/synthesis wrapper objects derive from. Each object has a method for real-time interaction as well as non-real-time or batch mode processing, as long as these modes are supported by the underlying algorithm. For any given step, every analysis/synthesis object returns data in the same format, irrespective of its underlying implementation. This allows analysis/synthesis networks to be created in which the algorithm that is used for a particular step can be changed without effecting the rest of the network. The process is summarised in figure 1.

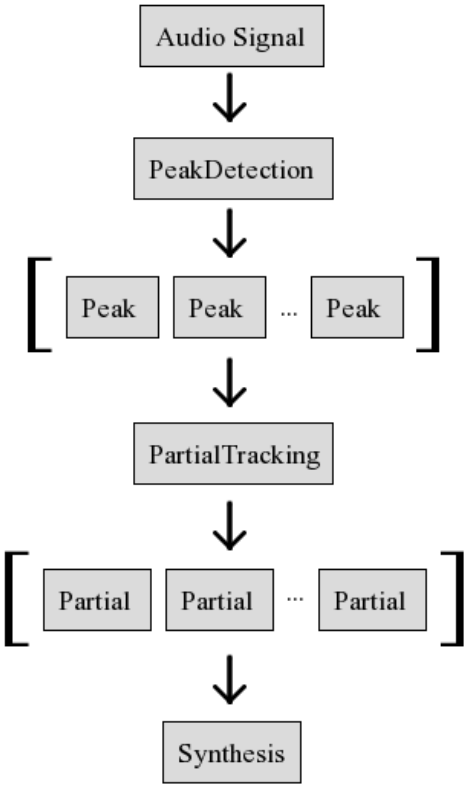

Figure 1: *Simpl analysis-synthesis process*

2.1.Peak Detection

PeakDetection objects take a NumPy array of audio samples as input. This can be just a single frame of audio, or a longer signal of arbitrary length that will be cut up into frames for further processing internally and zero padded if necessary. For each frame, spectral peaks are calculated. If the input was a single audio frame, then a single list of Peak objects is returned. If it was a longer signal, a separate list of Peaks is returned for each audio frame.

2.2.Partial Tracking

The input to PartialTracking objects is either a list of Peaks or an arbitrary number of lists of Peaks. This information is used by the partial tracking algorithm to form Partial objects, which are ordered lists of Peaks. The return value is always a list of Partials.

2.3.Sound Synthesis

SoundSynthesis objects take a list of Partials and a NumPy array of audio samples (the original signal) as input. They use this data in various ways depending on the synthesis algorithm, but the general process is to use the Partial data to synthesise the harmonic (deterministic) sound component, then subtract this from the original signal in order to obtain the residual (stochastic) component. All derived objects can return a fully synthesised signal, as well as these two components in isolation if supported. For example, the MQ algorithm does not make this distinction

and between components and so the MQSoundSynthesis object returns a synthesised signal based only on the Partial data. SMS-SoundSynthesis on the other hand can return all three signal types. Audio signals are returned as NumPy arrays.

3.EXAMPLES

In this section we will present three examples, demonstrating the system. The first deals with the basic manipulation of audio data using SciPy. The next two provide examples of the Simpl library proper for two basic tasks of analysis-synthesis and spectral display.

3.1 Using SciPy

The following example shows how SciPy can be used to read in an audio file called piano.wav and plot it using Matplotlib. The resulting plot is displayed in figure 2.

```
from scipy.io.wavfile import read
from pylab import plot, xlabel, ylabel, \
                  title, show

input_data = read('piano.wav')

# store samples as floats between -1 and 1
audio_samples = input_data[1] / 32768.0

# plot the first 4096 samples
plot(audio_samples[0:4096])
ylabel('Amplitude')
xlabel('Time (samples)')
title('piano.wav')
show()
```

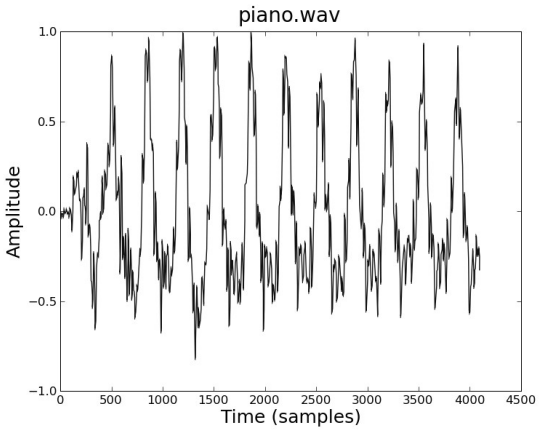

Figure 2: *Resulting waveform plot*

3.2.Using Simpl

This data can now be passed directly to the Simpl analysis objects. In the following example, peak detection is performed using the Sound Object Library, followed by partial tracking from the MQ algorithm before finally the sound is resynthesised using SMS. All operations are performed in non-real-time.

```
from scipy.io.wavfile import read
from scipy import asarray, float32
from SimplSndObj import SndObjPeakDetection
from SimplMQ import MQPartialTracking
from SimplSMS import SMSSynthesis
input_data = read('piano.wav')
# store audio samples as 32-bit floats,
# with values between -1 and 1
audio_samples = asarray(input_data[1], \
   float32) / 32768.0

# This detects peaks using the SndObj lib
# and stores them in a numpy array
pd = SndObjPeakDetection()
peaks = pd.find_peaks(audio_samples)

# Here we have partial tracking using
# McAulay-Quatieri method
pt = MQPartialTracking()
partials = pt.find_partials(peaks)

# finally we synthesise the audio
# using SMS
synth = SMSSynthesis()
# our detected partials will be used to
# form the harmonic component, and the
# original audio signal will be used when
# calculating the residual
audio_out = synth.synth(partials, \
   audio_samples)
```

3.3 Simpl Data Visualisation

Partial tracking data can be displayed using the Simpl plotting module. The plot produced by this example is shown in figure 3.

```
from scipy.io.wavfile import read
from scipy import asarray, float32
from SimplSndObj import SndObjPeakDetection
from SimplMQ import MQPartialTracking
from SimplPlots import plot_partials

# read audio data
input_data = read('piano.wav')
audio_samples = asarray(input_data[1], \
   float32) / 32768.0

# detect up to a maximum of 20 peaks. If
# there are more, the 20 with the largest
# amplitudes will be selected
pd = SndObjPeakDetection()
pd.max_peaks = 20
peaks = pd.find_peaks(audio_samples)

# track peaks
pt = MQPartialTracking()
partials = pt.find_partials(peaks)
# display them
plot_partials(partials)
show()
```

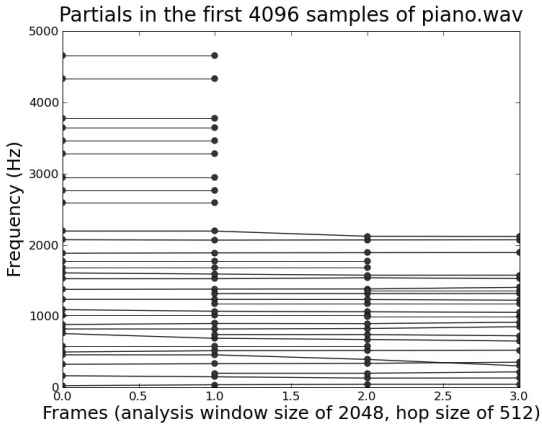

Figure 3: *Plot of partial data. Circles represent peaks, lines show the resulting partials. Some extra peaks were created by the MQ partial tracking algorithm, for partial 'birth' and 'death'.*

4. FUTURE WORK

More visualisation functions will be added to the library. In particular, we want to add the ability to display data during real-time analysis. The current plotting functions are not efficient enough to achieve this. It is expected that more analysis/synthesis algorithms will be added to the library, adapted from the many published papers in the field. We would also like to add the ability to control algorithm parameters in real-time using OpenSound Control [20]. Developers are encouraged to contribute to the project, and can contact the authors via email.

5.CONCLUSION

Simpl provides a new environment for developing sinusoidal modelling applications, unifying several of the existing solutions in addition to implementing some of the most important advances in the field. Together with the flexibility of Python and the extensive range of SciPy functions, Simpl should be a valuable tool for other researchers and developers.

Simpl is free software, available under the terms of the GNU GPL. To download it or for more information go to: http://simplsound.sourceforge.net

6.ACKNOWLEDGEMENTS

The authors would like to acknowledge the generous support of An Foras Feasa, who funded this research.

7.REFERENCES

[1] X. Amatriain, J. Bonada, A. Loscos, X. Serra, "DAFX - Digital Audio Effects", chapter Spectral Processing, pp 373-438, Udo Zölzer Ed, John Wiley & Sons, Chichester, UK, 2002.

[2] G. Van Rossum, F. Drake, "The Python Language Reference Manual", Network Theory, Bristol, UK, 2006.

[3] E. Jones, T. Oliphant, P. Peterson and others, "SciPy: Open Source Scientific Tools for Python", http://www.scipy.org, accessed April 6, 2009.

[4] J. Smith, X. Serra, "PARSHL: An Analysis/Synthesis Program for Non-Harmonic Sounds Based on a Sinusoidal Representation." Proceedings of the International Computer Music Conference (ICMC), San Francisco, USA, 1987.

[5] V. Lazzarini, "The Sound Object Library", Organised Sound 5 (1), pp 35-49, Cambridge University Press, Cambridge, UK, 2000.

[6] J. Ffitch, "On the Design of Csound5", Proceedings of the 3rd Linux Audio Conference (LAC), pp. 37-42, ZKM, Karlsruhe, Germany, 2005.

[7] K. Fitz, L. Haken, S. Lefvert, M. O'Donnel, "Sound Morphing using Loris and the Reassigned Bandwdith-Enhanced Additive Sound Model: Practice and Applications", Proceedings of the International Computer Music Conference, Gotenborg, Sweden, 2002.

[8] X. Amatriain, P. Arumi, D. Garcia,"CLAM: A Framework for Efficient and Rapid Development of Cross-platform Audio Applications", Proceedings of ACM Multimedia, Santa Barbara, California, USA, 2006.

[9] R. Eakin, X. Serra, "libsms Library for Spectral Modeling Synthesis" http://www.mtg.upf.edu/static/libsms/, accessed April 06, 2009.

[10] M. Desainte-Catherine, S. Marchand, "Structured Additive Synthesis: Towards a Model of Sound Timbre and Electroacoustic Music Forms", Proceedings of the International Computer Music Conference (ICMC), pp. 260-263, Beijing, China, 1999.

[11] R. McAulay, T. Quatieri, "Speech Analysis/Synthesis Based on a Sinusoidal Representation", IEEE Transaction on Acoustics, Speech and Signal Processing, vol. 34, no. 4, pp. 744–754, 1986.

[12] X. Serra, J. Smith, "Spectral Modeling Synthesis A Sound Analysis/Synthesis Based on a Deterministic plus Stochastic Decomposition", Computer Music Journal, Vol. 14, No. 4 (Winter), 12-24, 1990.

[13] K. Fitz, "The Reassigned Bandwidth-Enhanced Method of Additive Synthesis", Ph. D. dissertation, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, USA, 1999.

[14] F. Keiler, S. Marchand, "Survey on Extraction of Sinusoids in Stationary Sounds", Proceedings of the 5th International Conference on Digital Audio Effects (DAFx), Hamburg, Germany, 2002.

[15] P. Depalle, G. Garcia, X. Rodet, "Tracking of Partials for Additive Sound Synthesis Using Hidden Markov Models", Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Minneapolis, Minnesota, USA, 1993.

[16] M. Lagrange, S. Marchand, M. Raspaud, J. Rault, "Enhanced Partial Tracking Using Linear Prediction ", Proceedings of the 6th International Conference on Digital Audio Effects (DAFx), London, UK, 2003.

[17] T. Oliphant, "Guide to NumPy", http://numpy.scipy.org/numpybook.pdf, accessed April 06, 2009.

[18] The MathWorks, "MATLAB - The Language of Technical Computing", http://www.mathworks.com/products/matlab, accessed April 06, 2009.

[19] J. Hunter and others, "Matplotlib - Python Plotting", http://matplotlib.sourceforge.net/, accessed April 06, 2009.

[20] M. Wright, A. Freed, A. Momeni, "OpenSound Control: State of the Art 2003", Proceedings of the Conference on New Interfaces for Musical Expression (NIME), Montreal, Canada, 2003.