Università degli Studi di Milano
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

PHD PROGRAM IN
COMPUTER SCIENCE

SIGNAL MODELS, ANALYSIS ALGORITHMS, AND SOFTWARE TOOLS FOR MODAL AUDIO RESYNTHESIS

Supervisor: Prof. Federico Avanzini
PhD Coordinator: Prof. Roberto Sassi

PhD Thesis by:
Marco Tiraboschi
Matr. No. R12964
SSD INF/01

Academic Year 2022-2023

“What I cannot create, I do not understand”

– Richard Feynman

Acknowledgements

I would like to acknowledge the invaluable guidance and support of my supervisor, Prof. Federico Avanzini, and of all other members of the *Laboratorio di Informatica Musicale*. Especially, I would like to thank Prof. Giorgio Presti and Prof. Luca Andrea Ludovico for all the stimulating conversations we have had since they were supervisors of my bachelor's thesis. I would also like to thank my colleague, Dr. Giulia Clerici for the encouragement she has given me throughout these years and for the collaboration we have carried on. Finally, I would like to thank Prof. Davide Rocchesso, Dr. Stefano Papetti, and Dr. Stefano Delle Monache for their continued help and feedback about my work on the Sound Design Toolkit.

Abstract

This thesis delves into topics related to modal sound synthesis, a technique that generates sounds by simulating the physical interactions within resonating objects. It proposes novel methods for analyzing audio recordings and extracting the modal parameters.

A central algorithm, SAMPLE, estimates these modal parameters by finding trajectories in the spectrogram of the input audio. However, SAMPLE encounters challenges with specific sounds, like acoustic beats, where two close frequencies interact and create a beating effect. To overcome this limitation, the thesis introduces BeatsDROP, an auxiliary algorithm that complements SAMPLE and models the trajectories in the spectrogram as the amplitude and frequency modulations of beats. This thesis also details the reasons why most signal analysis models fail with beats. Furthermore, the thesis presents the Generalized Mixture Space (GMS) model, which aids in representing sounds with multiple channels. GMS allows SAMPLE's simplified analysis to be applied while retaining the original channel distribution information for later resynthesis.

Beyond the theoretical framework, the thesis details the development of software tools to make these methods readily usable. SAMPLE is a Python package that implements the algorithms and models, along with additional functionalities previously defined in the audio DSP literature. The thesis also includes the re-engineering and expansion of the existing SDT (Sound Design Toolkit), written in C and available as an external library for Pure Data and Max. Functionalities are implemented within SDT to enable interoperability with other software, including the possibility to import modal analysis results from SAMPLE directly into SDT's modal synthesis models.

Contents

Chapter 1

Introduction

This thesis describes several contributions in the field of acoustic signal processing, especially in the context of digital sound analysis for modal synthesis. Modal synthesis approaches sound synthesis approximating the physical behaviour of sounding objects with linear models, while combining multiple objects by simulating the nonlinear interactions between them. These physically-based approaches to sound synthesis rely on a rich literature that spans decades, but have been gaining more and more relevance after the recent increases in the computational capabilities of consumer-grade computers. They can be used for different applications, from simulating the physics of real-life acoustic musical instruments to produce digital musical instruments to complementing interactive graphical experiences, like videogames and extended reality, with their auditory counterpart. With respect to using recorded samples of acoustic events, procedural sound generation, and physical models in particular, provide a more flexible and customizable sound generation process and better capable to match the user's actions with acoustic reactions.

The objective of this work is to provide an analytical framework for the estimate of modal sound synthesis parameter from real-life recordings of object impulse responses. Within this framework, novel acoustic signal models and algorithms were designed, following a problem-oriented mindset.

Firstly, and even before the start of my PhD studies, I investigated the core problem of estimating modal frequencies, gains, and decay

times from audio. This brought to the design of the SAMPLE (Spectral Analysis for Modal Parameters Linear Estimate) algorithm. This algorithm is based on previous works on analytical models for sinusoidal modelling synthesis, and is based on tracking the magnitude peaks in the Short-Time Fourier Transform of the sound.

Once this algorithm was designed, implemented, and evaluated, I noticed that it could not effectively reproduce acoustic beats. This is due to the fact that those beating components were not decomposed by the Short-Time Fourier Transform. While seeking a solution to the problem of analyzing acoustic beats, specifically in the context of modal systems, I investigated the effectiveness of alternative spectra analysis algorithm, such as all-pole models and empirical mode decomposition. I discovered that they also are unable to correctly analyze acoustic beats and I found theoretical and experimental arguments to their limitations. Then, I formulated a mathematical model for acoustic beats that allows to transform those signals from the sum of two oscillations to the product of an amplitude modulation component and a frequency-modulated sine wave. This was extended to the case of non-stationary components, especially to the case of exponentially-decaying amplitudes which characterizes modal systems' responses. After establishing this model, I developed a secondary algorithm called BeatsDROP (Beats Duality for the Resolution Of Partials) to complement SAMPLE. BeatsDROP fits the modal parameters of two components to the results of the peak tracking from SAMPLE. This algorithm was also computationally evaluated to assess its efficacy.

Finally, I addressed the problem of estimating modal parameters from multi-channel recordings. To this purpose, I extended the concepts of the Bivariate Mixture Space (BMS) model, designed for stereophonic signals, to the case of an arbitrary number of channels. This novel Generalized Mixture Space (GMS) allows the compact representation of the spectral content of multivariate signals as a single spectrum, called the Principal Spectrum. Although I have yet to evaluate the effects of integrating the GMS in the SAMPLE algorithmic pipeline, the aim is to find modes by tracking peaks in the Short-Time Principal Spectrum, while the distribution of the mode's energy can be estimated within the GMS model itself. In this thesis, the GMS is presented and its mathematical properties are formally investigated. I specifically

researched its ability to represent the spectral content of linear combinations of latent signal sources. Also, I investigated its equivalence to the BMS when applied on bivariate signals.

With the goal of making all this research accessible to end-users, I developed a software package for Python which includes the implementation of the previously described models. It also implements an interface for an optimization library, such that the hyperparameters of SAMPLE can be tuned by optimizing a loss function. For the purpose of defining perceptual loss functions, I also developed some efficient functions for computing time-frequency representations. Finally, a GUI makes the package accessible by users who are not programmers.

I also contributed to the development and the re-engineering of the Sound Design Toolkit. The SDT is a library for procedural sound generation written in C, that is also available as externals for Pure Data and Max. Notably, I implemented the support for centralized OSC communication towards the library, and the import and export of parameters with JSON, both for single objects and for projects involving multiple components. The SAMPLE Python package also exports modal parameters in a JSON syntax compliant with the SDT.

1.1 Publications

Some of the contributions of this thesis have been previously presented in the following publications.

[101] Marco Tiraboschi and Federico Avanzini. 'SAMPLE: a Python Package for the Spectral Analysis of Modal Sounds'. In: *Conference Proceedings of the 23rd Colloquium on Music Informatics*. "Aldo Piccialli" Award for the best contribution to the Scientific Programme. 2022, pp. 50–55. URL: https://hdl.handle.net/2434/945288

[104] Marco Tiraboschi, Federico Avanzini and Stavros Ntalampiras. 'Spectral Analysis for Modal Parameters Linear Estimate'. In: *Proceedings of the 17th Sound and Music Computing Conference*. Ed. by Simone Spagnol and Andrea Valle. SMC. Sound and Music Computing Network. Torino, Italy: Axea sas/SMC Network, June 2020, pp. 276–283. DOI: 10.5281/zenodo.3898795

[106] Marco Tiraboschi, Stefano Papetti and Federico Avanzini. 'Just a Sounding Object Notation: Sharing Objects for Sonic Interaction Design with JSON and OSC'. in: *Proceedings of the 4th International Symposium on the Internet of Sounds*. 2023. DOI: 10.1109/IEEECONF59510.2023.10335232

The following manuscripts, instead, are either under review or in preparation.

[102] Marco Tiraboschi and Federico Avanzini. 'Acoustic Beats and Where To Find Them: Uneven Beats Models and Modal Audio Inversion by Non-linear Least-Squares Optimization'. In: *Inverse Problems* (2024). [UNDER REVIEW]

[107] Marco Tiraboschi and Giorgio Presti. 'The Generalized Mixture Space: extending Compact Spectral Representations to Multichannel Signals'. In: *Journal of the Audio Engineering Society* (2024). [TO BE SUBMITTED]

During the period of my PhD, I published other articles, that are not part of this thesis, because they are about unrelated topics, although they belong to the field of Sound and Music Computing as well. First of all, I published an article based on my master's thesis.

[103] Marco Tiraboschi, Federico Avanzini and Giuseppe Boccignone. 'Listen to your Mind's (He)Art: A System for Affective Music Generation via Brain-Computer Interface'. In: *Proceedings of the 18th Sound and Music Computing Conference*. 2021. DOI: 10.5281/zenodo.5044984

I also collaborated with colleagues of the *Laboratorio di Informatica Musicale* (LIM) and working on the SONICOM project (funded by the European Commission's Horizon2020 programme) on the following work concerning the matching of reverb algorithm parameters to a given room impulse response.

[14] Riccardo Bona et al. 'Automatic Parameters Tuning of Late Reverberation Algorithms for Audio Augmented Reality'. In: *Proceedings of the 17th International Audio Mostly Conference*. AM '22. St. Pölten, Austria: Association for Computing Machinery, 2022, pp. 36–43. DOI: 10.1145/3561212.3561236

Finally, I worked with Giulia Clerici on a project for a PhD course on large graph analytics, which lead to the following contribution.

[22] Giulia Clerici and Marco Tiraboschi. 'Citation is not Collaboration: Music-Genre Dependence of Graph-Related Metrics in a Music Credits Network'. In: *Proceedings of the 20th Sound and Music Computing Conference*. (All authors contributed equally). 2023. DOI: 10.5281/zenodo.10061131

As a doctoral candidate in Computer Science, I would also like to acknowledge the open-source software I developed amongst my publications. I have always believed in the value and the impact of open source software on the technological advancement of the related communities, and this has also been investigated on behest of the European Commission [10], and confirmed. Mainly, my work contributed to the development of the following two projects.

[26] Stefano Delle Monache *et al.* and Marco Tiraboschi. *Sound Design Toolkit*. June 2022. DOI: 10.5281/zenodo.6628035

[99] Marco Tiraboschi. *SAMPLE – Python package*. LIM, University of Milan, 2021. DOI: 10.5281/zenodo.6536419

I also contributed to projects I was not affiliated to. Notably, I published some fixes for the JSON library that is used in the Sound Design Toolkit [26].

[59] James McLaughlin *et al.* and Marco Tiraboschi. *json-builder – The serializing counterpart to json-parser*. URL: https://github.com/json-parser/json-builder

[60] James McLaughlin *et al.* and Marco Tiraboschi. *json-parser – Very low footprint DOM-style JSON parser written in portable ANSI C*. URL: https://github.com/json-parser/json-parser

1.2 Structure

The thesis is divided in 7 chapters, grouped in two parts.

Chapter 1 **Introduction**. This chapter presents the scope and the structure of the thesis.

Chapter 2 **Background**.. This chapter provides an overview of preliminary concepts, which should make the rest of the thesis more accessible. These include the fundamentals ideas behind modal synthesis and some basic elements of Digital Signal Processing.

Part I **Models and Algorithms**. This part collects all contributions that involve mathematical modelling and algorithmic design for modal audio analysis.

Chapter 3 **Sinusoidal Analysis of Modal Audio**. This chapter describes the SAMPLE algorithm for the sinusoidal analysis modal impulse responses. This is the core analytical model, which can be complemented by the other methods. Finally, the algorithm is evaluated with both computational and subjective tests.

Chapter 4 **Modal Acoustic Beats**. This chapter details a model of acoustic beats as sinusoids modulated in amplitude and frequency, especially in the case of exponentially decaying oscillations. It also provides a theoretical discussion and experimental evidence of the frequency-resolution limits of most commonly used signal analysis approaches. Based, on this model, the analytical algorithm BeatsDROP is designed and tested computationally.

Chapter 5 **Generalized Mixture Space**. This chapter presents the GMS, a novel representation for multivariate signals. This model extends the Bivariate Mixture Space for stereo signals. Although general, it provides insightful information when the analyzed signal is a multivariate mixture of latent source signals. The features of the GMS are presented, along with their computational formulations, and they are compared to the features of the BMS, showing that the GMS can be considered a generalization of BMS. Some applications are proposed for modal audio analysis and for spectral visualization.

Part II **Software and Tools**. This part collects all contributions that consist in the software development of tools for modal audio analysis and sound design.

Chapter 6 **SAMPLE Python Package**. This chapter describes the SAMPLE package for Python. This package includes the implementation of SAMPLE and BeatsDROP. This estimation algorithms can be interfaced with hyperparameter optimization modules. Some efficient functions for computing time-frequency representations for audio are also implemented in the package for the purpose of defining psychoacoustically-based loss functions for the optimizers. A GUI complements the package as a more user-friendly way to use the methods.

Chapter 7 **Sound Design Toolkit**. This chapter details the work done on the Sound Design Toolkit. The SDT codebase was restructured and the package was also included in the official Max package manager. New features in SDT include the support of centralized OSC communication for the entire library and of importing and exporting the parameters of individual or groups of objects with JSON.

Chapter 2

Background

In this chapter, I provide some background for my thesis. This serves two different purposes. Firstly, I would like to acknowledge the intrinsically heterogeneous and multidisciplinary nature of the research community in Sound and Music Computing [9]. Researchers in this field have very diverse backgrounds, ranging from the hard sciences (physics, mathematics) to the soft sciences (psychology, sociology), and from technology and engineering (electronics, signal processing) even to the arts (music, design). This is a doctoral thesis in Computer Science, and most of the topics could be classified under the disciplines of signal processing or software development. Nevertheless, my wish is to make my work accessible to as broad an audience as I can by summarizing the most important prerequisite pieces of knowledge, as if this were a cheat sheet, even if this information will be redundant for some readers.

The second purpose is to add some context, highlighting the relevance and application domains of my research. This chapter should, therefore, clarify why I am working on the topics that I detail in the following chapters.

2.1 Acoustic Physical Models

In this section, I will refresh some of the elements of classical mechanics that are relevant to modal synthesis, a specific approach to physical-modelling sound synthesis. Physical models for sound synthesis generate audio by simulating the vibrations that mechanical systems produce. Usually, the interest is in modelling objects from the real world, but physical models can also be used to simulate the sound of objects that would be impossible or unfeasible to build. In physical sound modelling, "emphasis is placed on techniques which yield the highest *playability* and *sound quality* in real time at a reasonable computational expense" [91]. Keeping this in mind, most implementations can scale the sound quality with the computational expense, allowing the configuration of what has more recently been named the SLOD (Sound Level Of Detail) or the (Level Of Audio Detail) LOAD of the system [2]. Although physical models are often described continuous in time or also in space, implementations must rely on discretizations of such formulations. Approaches for the discretization and for the implementation of such models borrow from the literature in the related field of discrete-time Digital Signal Processing [114]. We also give a brief introduction to the relevant DSP fundamentals for in Section 2.2. Readers interested in gaining either a broader or a deeper understanding of physical audio signal processing may want to refer to Julius Smith's textbook [93].

2.1.1 Damped Harmonic Oscillator

Many physical modelling systems, even the earliest ones, such as the CORDIS system described by Claude Cadoz in his doctoral thesis [19], are built by approximating complex objects as a network of point-sized masses interacting with each other through springs. This abstract approach allows to model objects with arbitrary geometries and mass distribution, while working inside a mathematically tractable framework.

Classical linear mechanical equations can describe the temporal evolution of such a system. A force $f_m(t)$ applied to a mass m causes it to accelerate.

$$f_m(t) = m \cdot \frac{d^2x}{dt^2}(t) \tag{1}$$

Such force can be given as an external excitation $f(t)$, but it also includes the internal force from the spring that tends to bring back the mass to its equilibrium position. Such force can be expressed by Hooke's law, where $k \geq 0$ is the elastic coefficient.

$$f_k(t) = -k \cdot x(t) \tag{2}$$

It also includes friction, which is opposed to speed of the mass. The friction coefficient is c.

$$f_c(t) = -c \cdot \frac{dx}{dt}(t) \tag{3}$$

Pulling all the equations together, they form a second-order ordinary differential equation that describes a very common system known as "damped harmonic oscillator" in classical mechanics.

$$f_m(t) = f(t) + f_c(t) + f_k(t) \tag{4}$$

$$f(t) = m \cdot \frac{d^2x}{dt^2}(t) + c \cdot \frac{dx}{dt}(t) + k \cdot x(t) \tag{5}$$

The response of this system to an external impulsive force is a signal that exponentially decays to the equilibrium point. If the "damping ratio" $\zeta = c/2\sqrt{mk}$ is less than 1, the system is underdamped, and it oscillates while decaying.

2.1.2 Modal System

When the model is a network of springs and masses, the dynamics of the system can be expressed in a compact form using matrix and vector notation. For each node i of the network, the external force is the sum of the inertia of the mass and the elastic force and friction of all the springs to which it is attached to other nodes j with.

$$f_i(t) = m_i \frac{d^2x_i}{dt^2}(t) + \sum_j c_{i,j} \left(\frac{dx_i}{dt}(t) - \frac{dx_j}{dt}(t) \right) + \sum_j k_{i,j} \left(x_i(t) - x_j(t) \right) \tag{6}$$

Let M be the diagonal matrix of the masses ($M_{i,i} = m_i$ and $M_{i,j} = 0$ for $j \neq i$). A matrix C of the friction coefficients and a matrix K of the

elastic coefficients can be defined in order to simply express the system as the following equation [35].

$$f = Mx'' + Cx' + Kx \tag{7}$$

The diagonal elements in C and K balance the rest of the elements in the row so that, expanding the matrix-vector product, the previous formulation is recovered.

$$K_{i,j} = \begin{cases} -k_{i,j} & i \neq j \\ \sum_{j \neq i} k_{i,j} & i = j \end{cases} \qquad C_{i,j} = \begin{cases} -c_{i,j} & i \neq j \\ \sum_{j \neq i} c_{i,j} & i = j \end{cases} \tag{8}$$

$$(Kx)_i = \sum_j K_{i,j} x_j = \sum_{j \neq i} k_{i,j} x_i - \sum_{j \neq i} k_{i,j} x_j = \sum_j k_{i,j} \left(x_i - x_j\right) \tag{9}$$

With this model, the simulation can be very expensive for highly-connected networks. Diagonalizing the system, we obtain an equivalent expression where all rows are independent, so that they can be simulated in parallel. Each of such rows is one damped harmonic oscillator, as described in the previous section, and corresponds to a mode of oscillation of the spring-mass network. Even though this diagonalized system does not reflect the original geometry and mass distribution of the simulated object, it introduces no error and its parameters are more related to the simulated acoustic signal. Each mode has:

- a resonant frequency $\nu_i = \sqrt{4m_i k_i - c_i^2}/4\pi m_i$,
- an exponential decay coefficient $\lambda_i = c_i/2m_i$,
- an amplitude factor $a_i = 1/\sqrt{k_i m_i}$,
- a phase shift ϕ_i.

The impulse response of a single mode of oscillation is an exponentially decaying causal sine-wave function.

$$h_i(t) = u(t) a_i e^{-\lambda_i t} \cos\left(2\pi \nu_i t + \phi_i\right) \tag{10}$$

Here, $u(t)$ is the Heaviside step function.

$$u(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{11}$$

Modal approaches to physical-modelling work on this diagonalized system, either computing it from the spring-masses network or describing the modelled system directly with modal parameters.

A similar modal formulation can be derived from a continuous-space model. In this settings Partial Differential Equations describe the dynamics of the systems. The most important of such equations is d'Alembert's equation, often just the referred to as the *wave equation*: u is a function of both time (t) and space (x, y, z), and c is the wave propagation speed.

$$\nabla^2 u = \frac{1}{c^2}\frac{\partial^2}{dt^2}u \qquad \nabla^2 := \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{12}$$

2.2 Digital Signal Processing

In this section, I will collect some of the elements of Digital Signal Processing that are relevant for this thesis, especially for the analysis of signals in the frequency domain. This is by no means intended to be an exhaustive summary of DSP basics, but I will also cover some of the fundamentals to establish some notation conventions. People interested in learning more about this should refer to a Digital Signal Processing textbook like John Proakis's [70].

The frequency analysis of a signal consists in decomposing it into elementary oscillating components, such as sines and cosines. Each of these components represents the oscillation at a specific frequency. For compactness of representation, the usual choice for the basis functions are complex exponentials, which can be expressed with Euler's formula as the sum of a cosine function on the real axis and a sine function on the imaginary axis.

$$ae^{j(2\pi\nu t+\phi)} = a\cos(2\pi\nu t + \phi) + ja\sin(2\pi\nu t + \phi) \tag{13}$$

Please, note that I am using $j = \sqrt{-1}$ as the imaginary unit, to avoid confusion with the variable i often used as an index. The independent variable t is often time, especially for audio signals. The factor ν is called the *frequency* of the component, and it is the reciprocal of the oscillation period. Often, the angular frequency $\omega = 2\pi\nu$ is used, instead. The phase ϕ shifts the component in time. The amplitude a determines the amount of oscillation.

2.2.1 Discrete Fourier Transform

A discrete and finite signal $\vec{x}$ is a sequence of length $n \in \mathbb{N}$ of samples. Any such signal can be represented as the sum of n complex exponentials.

$$x_i = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} X_k \, e^{2j\pi ki/n} = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} a_k \, e^{j(2\pi ki/n + \phi_k)} \tag{14}$$

The vector of complex coefficients $X_k = a_k e^{j\phi_k}$ is the discrete frequency domain representation of the signal. The k-th component has a normalized frequency of k/n, an amplitude of $a_k = |X_k|$, and a phase of $\phi_k = \angle X_k$. If the sequence $\vec{x}$ has been sampled from a continuous-time signal $\hat{x}$ such that $x_i = \hat{x}\,(i/\nu_s)$, then ν_s is called the sampling frequency (or sample rate) of the signal and $\nu_k = k\nu_s/n$ is the frequency of the k-th component.

The function that computes the complex coefficients from the time-domain signal is called the *Discrete Fourier Transform* (DFT).

$$X_k = \frac{1}{\sqrt{n}} \sum_{i=0}^{n-1} x_i \, e^{-2j\pi ki/n} \tag{15}$$

Its inverse function is the *Inverse Discrete Fourier Transform* (IDFT), already introduced in Equation (14). The DFT can also be expressed as the product of the DFT matrix W with the signal.

$$X = W\vec{x} \qquad\qquad W := \left[\frac{1}{\sqrt{n}} e^{-2j\pi ki/n} \right]_{k,i=0}^{n-1} \tag{16}$$

2.2.2 Fourier Transform

Where a sum of only a finite number of components are needed for finite-length sequences, a continuous range of infinite components need to be integrated to represent continuous signals in the frequency domain. The Fourier Transform (FT) computes the complex coefficients for the continuous frequencies components.

$$X(\nu) = \mathcal{F}\{x\}(\nu) = \int_{t=-\infty}^{+\infty} x(t) e^{-2j\pi\nu t} dt \tag{17}$$

The Inverse Fourier Transform (IFT) computes the signal from the complex coefficients.

$$x(t) = \mathcal{F}^{-1}\{X\}(t) = \int_{\nu=-\infty}^{+\infty} X(\nu) e^{2j\pi\nu t} d\nu \tag{18}$$

2.2.3 Z-Transform

The z-transform (ZT), instead, is used to analyze discrete-time, possibly infinite, signals. It is particularly interesting for the analysis of the response of discrete-time Linear Time-Invariant systems. The z-transform $\mathcal{Z}$ is defined as a series.

$$X(z) = \mathcal{Z}\{x\}(z) := \sum_{i=-\infty}^{+\infty} x_i \, z^{-i} \tag{19}$$

Please, note that the DFT can be considered a special case of the z-transform, evaluated for values $z = \exp\{j\omega\}$ (where $\omega = 2\pi k/n$ is the normalized angular velocity) over a finite amount of samples. Using the same arguments, but an infinite number of samples, then the z-transform becomes the Discrete-Time Fourier Transform (DTFT).

Commonly, the z-transform is expressed in a rational form, which is often derived by the difference equation of a system. This form highlights the zeros (the roots of the numerator), for which the transform goes to zero, and the poles (the roots of the denominator), for which the transform diverges to infinity.

$$X(z) = \frac{N(z)}{D(z)} = \frac{\sum_{i=0}^{n_z} b_i \, z^{-i}}{\sum_{i=0}^{n_p} a_i \, z^{-i}} = \frac{b_0}{a_0} z^{n_p - n_z} \frac{\prod_{i=0}^{n_z} (z - z_i)}{\prod_{i=0}^{n_p} (z - p_i)} \tag{20}$$

The maximum exponent either at the numerator or denominator is called the order of the system. Many other properties of a system can be inspected with the z-transform, but they are out of the scope of this thesis.

2.2.4 Short-Time Fourier Transform

Fourier Transforms are able to represent a signal as the sum of oscillatory components, but they do not represent explicitly when such oscillations are happening in the signal. The Short-Time Fourier Transform

(STFT) divides the signal in windows and applies the FT to each window. The STFT is parametrized by a *window* function w, which is zero everywhere except in a limited range of its domain, where it is bound between zero and one. Multiplying the signal by the window function has the effect of isolating a temporal window of the signal and, for most practical window functions, smoothing the boundaries of the signal in the window.

$$\text{STFT}\{x\}(t, \nu|w) = \int_{\tau=-\infty}^{+\infty} x(\tau)w(\tau - t)e^{-2j\pi\nu\tau}d\tau \tag{21}$$

If the window values $w(t)$ are zero for all times below zero and above t_w, then the integration interval can be restrained.

$$\text{STFT}\{x\}(t, \nu|w) = \int_{\tau=t}^{t+t_w} x(\tau)w(\tau - t)e^{-2j\pi\nu\tau}d\tau \tag{22}$$

Using a similar formulation, the STFT is also defined for discrete signals.

$$\text{STFT}\{x\}_{i,k|w} = \sum_{m=-\infty}^{+\infty} x_m w_{m-i}\, e^{-2j\pi km/n} \tag{23}$$

The series become a finite sum if the length of the window n_w is explicit. Usually, the required sample rate for the STFT is much lower than the input's. Therefore, every successive window is not spaced one sample apart, but a number of samples h that is called *hop size*.

$$\text{STFT}\{x\}_{i,k|w,h} = \sum_{m=hi}^{hi+n_x-1} x_m w_{m-hi}\, e^{-2j\pi km/n} \tag{24}$$

2.2.5 Hilbert Transform

A real-valued time domain cosine function $a(t)\cos(\theta(t))$ can be transformed to a complex exponential by adding the corresponding imaginary sine function $ja(t)\sin(\theta(t))$ (see Equation (13)). The Hilbert transform shifts the phases of all frequency components by $\pi/2$, effectively mapping cosines to sines.

$$\mathcal{H}\{a\cos(2\pi\nu t + \phi)\} = a\sin(2\pi\nu t + \phi) \tag{25}$$

The *analytic signal* corresponding to the original signal can be built adding the original signal as the real part and its Hilbert transform as the imaginary part.

$$z(t) = x(t) + j \cdot \mathcal{H}\{x\}(t) \tag{26}$$

The analytic signal can also be computed with the FT [92] (as in Equation (11), u is the Heaviside step function).

$$z(t) = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{x\right\} \cdot u\right\}(t) \tag{27}$$

We can use the analytic signal to express any signal as a cosine function with time varying frequency $\nu(t)$ and amplitude $a(t)$.

$$x(t) = a(t)\cos\left(\theta(t)\right) = a(t)\cos\left(\phi + 2\pi\int_0^t \nu(t)dt\right) \tag{28}$$

$$a(t) = |z(t)| \qquad \theta(t) = \angle z(t)$$
$$\phi = \theta(0) \qquad \nu = \frac{d\theta}{2\pi dt}(t)$$

2.2.6 Auto-correlation

Given a signal x, its autocorrelation function $R_{xx}(\tau)$ expresses how much the signal is similar to itself after a delay of τ.

$$R_{xx}(\tau) = \int_{-\infty}^{+\infty} x(t)\overline{x}(t-\tau)dt \tag{29}$$

The complex conjugate of x, $\overline{x}$ is equal to x for real signals.

$$\begin{aligned} x &= a + jb & x &= \rho e^{j\theta} \\ \overline{x} &= a - jb & \overline{x} &= \rho e^{-j\theta} \end{aligned} \tag{30}$$

Complex conjugation is important for complex-valued signals, because the product of a complex number by its conjugate is their square magnitude.

$$x\overline{x} = \rho e^{j\theta}\ \rho e^{-j\theta} = \rho^2 \tag{31}$$

The correlation function is similar to a convolution.

$$(x * y)(\tau) = \int_{-\infty}^{+\infty} x(t)y(\tau - t)dt \tag{32}$$

In fact, the autocorrelation function of a real signal is equal to the convolution of the signal with a time-reversed copy of itself.

$$(x(t) * x(-t))(\tau) = \int_{-\infty}^{+\infty} x(t)x(-(\tau - t))dt = R_{xx}(\tau) \tag{33}$$

The autocorrelation function is also closely related to the Fourier Transform: it can be computed as the IFT of the square magnitude of the FT of the signal (sometimes, referred to as the Power Spectral Density). It can be proven using simple properties of the FT. First, the FT of a convolution is the product of the FTs.

$$\mathcal{F}\{x * y\}(\nu) = \mathcal{F}\{x\}(\nu) \cdot \mathcal{F}\{y\}(\nu) \tag{34}$$

Secondly, the FT of a time-reversed real signal is equal to the IFT of the original signal and to the complex conjugate of its FT.

$$\begin{aligned}
\mathcal{F}\{x(-t)\}(\nu) &= \int_{t=-\infty}^{+\infty} x(-t)e^{-2j\pi\nu t}dt = && (35)\\
&= \int_{t=-\infty}^{+\infty} x(\tau)e^{2j\pi\nu\tau}d\tau \overset{(18)}{=} \mathcal{F}^{-1}\{x\}(\nu) && (36)\\
&\overset{(30)}{=} \int_{t=-\infty}^{+\infty} \overline{x(\tau)e^{-2j\pi\nu\tau}}d\tau = \overline{\mathcal{F}\{x\}}(\nu) && (37)
\end{aligned}$$

This proves that the autocorrelation function is the IFT of the square magnitude of the FT of the signal.

$$\begin{aligned}
R_{xx}(\tau) &= (x(t) * x(-t))(\tau) = && (38)\\
&\overset{(34)}{=} \mathcal{F}^{-1}\{\mathcal{F}\{x(t)\} \cdot \mathcal{F}\{x(-t)\}\}(\tau) =\\
&\overset{(35)}{=} \mathcal{F}^{-1}\left\{\mathcal{F}\{x\} \cdot \overline{\mathcal{F}\{x\}}\right\}(\tau) =\\
&\overset{(31)}{=} \mathcal{F}^{-1}\left\{|\mathcal{F}\{x\}|^2\right\}(\tau)\\
\mathcal{F}\{R_{xx}\}(\nu) &= |\mathcal{F}\{x\}|^2(\nu)
\end{aligned}$$

Similar operations and properties can be derived for discrete signals.

2.3 Regression Analysis

In statistics, regression is a function that estimates the parameters of a model from a set of observations of inputs (independent variables) and outputs (dependent variables). Regression models range in complexity and expressive power, from the simple linear regression approaches to probabilistic processes and deep neural networks. In general, a regression model is a function r that, given a parametric function f_p and a set of paired observations for the independent variables and the dependent variables $(\vec{x}_i, \vec{y}_i)$, yields an estimate of the parameters $\hat{p}$ such that the function approximates $f_{\hat{p}}$ the relationship that exists in the data.

$$r\left(f, \left\{(\vec{x}_1, \vec{y}_1), \dots\right\}\right) = \hat{p} \;:\; f_{\hat{p}}(\vec{x}_i) \approx \vec{y}_i \quad \forall i \tag{39}$$

If this formulation seems too abstract, it is because it tries to encompass all the aforementioned variety of methods and contexts. Notably, what it means to "approximate" the relationship in the data really depends on the specific model.

2.3.1 Linear Least-Squares

The most simple type of regression is linear regression. In this model, the relationship in the data is assumed to be a linear function.

$$A\vec{x}_i + \vec{b} \approx \vec{y}_i \quad \forall i \tag{40}$$

This formulation can also be expressed more compactly by adding a leading 1 to every input $\vec{x}_i$.

$$AX = Y \qquad A := \begin{bmatrix} b_1 & b_2 & \dots & b_m \\ a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n,1} & \dots & \dots & a_{n,m} \end{bmatrix} \quad X_{\bullet,i} := \begin{bmatrix} 1 \\ x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix} \tag{41}$$

If we multiply both sides by $X^+ = X^T \left(XX^T\right)^{-1}$ (the Moore-Penrose inverse of X), we derive an expression for matrix A.

$$\hat{A} = YX^T \left(XX^T\right)^{-1} \tag{42}$$

This can be proven to be the solution to the following optimization problem, called the Least-Squares problem.

$$\hat{p} = \arg\min_{p} \sum_{i} \left\| f_p(\vec{x}_i) - \vec{y}_i \right\|_2^2 \tag{43}$$

Since, in this instance, we have assumed a linear model, the specific problem is known as Linear Least-Squares.

$$\hat{A} = \arg\min_{A} \sum_{i} \| (AX - Y)_i \|_2^2 \tag{44}$$

The problem could also be framed as finding the parameters of the model that maximize the likelihood of the data, assuming that the approximation errors are independent, homoscedastic, and follow a Gaussian distribution with zero mean. In this formulation, the problem is called Ordinary Least-Squares.

$$\vec{y}_i = f_p(\vec{x}_i) + \epsilon \quad \epsilon \sim \mathcal{N}\left(\vec{0}, \mathbb{I}\right) \tag{45}$$

$$\hat{p} = \arg\max_{p} \prod_{i} \mathbb{P}_{\mathcal{N}}\left(f_p(\vec{x}_i) - \vec{y}_i \mid \vec{0}, \mathbb{I}\right) \tag{46}$$

Here, $\mathcal{N}(\vec{\mu}, \Sigma)$ is the Gaussian distribution with mean $\vec{\mu}$ and covariance Σ, and $\mathbb{P}_{\mathcal{N}}$ is its associated probability density function.

2.3.2 Nonlinear Least-Squares

The same framework can be extended to arbitrary nonlinear function. However, over this class of functions, there is no guarantee that the minimization algorithm will converge to the optimal solution.

Most approaches are based on either gradient descent (like most optimization algorithms for neural networks) or the Gauss-Newton algorithm, or on a combination of the two, like the Levenberg-Marquardt algorithm or Powell's dog leg method.

Part I

Models and Algorithms

Chapter 3

Sinusoidal Analysis of Modal Audio

Abstract

Modal synthesis models require a substantial number of parameters to achieve high-quality sounds. Defining such parameters is a lengthy and laborious process for sound designers if they do it manually. This chapter describes a method for practical modal parameter estimation, which is based on the spectral analysis of a single audio sample. The approach involves analyzing the sound spectrum using a sinusoidal model and fitting the modal parameters using linear and semi-linear techniques. The proposed method was tested on a dataset of impact sounds, taking into account objects of various shapes and materials. We assess the algorithm's performance by examining the quality of the resynthesized sounds. Resynthesis was performed using the Sound Design Toolkit (SDT) modal engine and compared to the sounds resynthesized from parameters extracted with the previous SDT's own

This chapter is based on [104] Marco Tiraboschi, Federico Avanzini and Stavros Ntalampiras. 'Spectral Analysis for Modal Parameters Linear Estimate'. In: *Proceedings of the 17th Sound and Music Computing Conference*. Ed. by Simone Spagnol and Andrea Valle. SMC. Sound and Music Computing Network. Torino, Italy: Axea sas/SMC Network, June 2020, pp. 276–283. DOI: 10.5281/zenodo.3898795

estimator. The proposed method was evaluated objectively, using perceptually relevant features, and subjectively following the MUSHRA protocol.

3.1 Introduction

Physically based and procedural methods for sound synthesis have historically struggled to gain widespread acceptance by the industry, despite their potential benefits in terms of interactivity and customizability. Böttcher [15] explored potential explanations for this. One such reason is the necessity for tools that simplify the tasks of sound designers, without burdening them with the low-level details of the specific models.

In the context of modal synthesis, a tool that is greatly needed by sound designers is one that enables automatic adjustment of the mode parameters to accurately resynthesize a target, possibly recorded, sound. In the restricted scenario of monophonic audio, each mode is completely defined by a triplet of scalars: the modal frequency, the decay coefficient, and the initial amplitude. This concept can be extended to multichannel audio by considering an array of initial magnitudes, each corresponding to a "pickup point" for the sound, or even to continuous spatial processes that describe the modal shapes as functions of spatial coordinates on the modal object. These parameters directly affect the spectral content of the synthesized sound, since each mode represent how much the modal object resonates at a specific frequency.

This chapter describes a method for the automatic estimation of modal parameters based on a target sound. The suggested method is called SAMPLE (*Spectral Analysis for Modal Parameters Linear Estimate*) as it uses a spectral modelling algorithm to monitor the energy fluctuations related to every sinusoidal component and subsequently applies linear regression to estimate the modal parameters corresponding to the spectral energy.

SAMPLE is assessed by resynthesizing the sounds with the Sound Design Toolkit (SDT), a software package that has been developed over several years (see Chapter 7), offering a collection of sound models for

Figure 1: The four object used for recording the dataset: a glass bottle, a metal saucepan, a porcelain mug and a small piece of wood. Recordings by Giulia Clerici [21].

the interactive creation of various acoustic phenomena, including interactions between solid objects via modal synthesis and physically-based interaction force models [4] (impacts and frictions). Figure 1 shows the objects used to compile the impact sounds objective evaluation dataset [105]. This evaluation set is a subset of the dataset used by Giulia Clerici for her master's thesis [21] (see Section 3.2.3).

3.1.1 Related work

Modal synthesis has been gaining popularity as a sound synthesis approach for interactive computer graphics applications. One of the earlier examples has been the work of van den Doel *et al.* [27]. The goal in this context is to produce a sound that corresponds to the geometry and the materials of virtual objects, as well as the user's interaction with these objects. Typically, the object's geometry is known in this setting, providing valuable insights into the object's modes.

If the geometry and material distribution of the object are both known, finite element methods (FEM) can be utilized to determine the modal parameters. Picard *et al.* [68] suggest employing finite element methods to calculate the masses and stiffnesses of the spring-mass network matrix, which can be decomposed to obtain the modal parameters

(see Section 2.1.2). This method can calculate the full modal shape for the modes and enables simulation of the vibrations at any pickup point on the object. However, the object's geometry and material distribution have to be known.

Michon *et al.* [61] adopted a similar approach, using FEM analysis from a volumetric mesh of a 3D object to generate the corresponding modal system for FAUST, a programming language for real-time audio processing [65].

Acoustic information can also be used as an alternative or in conjunction with object geometry. Ren *et al.* [79] developed a hybrid approach that combines FEM and audio analysis. An initial approximation of the modal parameters is derived from the geometrical model, with the assumption that the material is isotropic and homogeneous, along with several initial values for mass density, Young's modulus, and Poisson's ratio. Subsequently, convex optimization is applied to adjust the material parameters to the modes detected in the audio example, which are extracted from a series of spectrograms with varying resolution. This method also uses deterministic residual compensation to model the non-sinusoidal component. The advantage of this approach is that it can operate without any prior knowledge of the material parameters. Moreover, it can transfer the identified material parameters to virtual objects with different geometries.

Sterling *et al.* [95] developed their method based on the feature extraction algorithm suggested by Ren *et al.*, and incorporated a probabilistic model for the damping parameters to mitigate the influence of external factors and nonlinearities on the damping estimate. Such external factors include the object support (which introduces additional damping to the object's inherent damping), background noise, errors in feature extraction (for instance, spectrogram resolution and windowing sidelobes), and the relative emission and pickup patterns of the object and the microphone. The modal parameters are computed using maximum likelihood from an average of nearly 50 impact sounds samples per object, employing an exponentially modified Gaussian as the likelihood's probability density function. This method neither requires the material parameters nor the geometrical model of the object, and enhances resilience to certain external factors. However, it is still susceptible to reverberation and does not provide an estimate of the modal shape function. In addition, multiple audio examples are needed

to estimate the parameters.

Kereliuk *et al.* [53] suggested a modification of the ESPRIT [83] algorithm to determine the modal parameters of room impulse responses. Their method accommodated a variable number of modes, prioritising high quality for lower mode counts.

Some methods tackle modal parameter estimation through parametric estimate using autoregressive (AR) system fitting. Abel *et al.* [1] employed resonant filter parameters fitting to simulate the modal response of rooms and spring reverberators. Maestre *et al.* [57] applied a similar approach to simulate the response of the body of stringed instruments, like guitars and violins.

Deep Learning is also becoming an increasingly popular technique for sound design, as testified by the addition of the "Foley Sound Synthesis" task in the 2023 edition of the DCASE challenge [20]. Unlike physical models, they are usually black-box end-to-end models that generate sounds with the desired attributes. Their application for sonic interaction is still limited because of the computational power they require and because they often do not allow interaction at all. However, some approaches, such as Differentiable Digital Signal Processing [29] are addressing both of these issues.

3.2 Method

The proposed estimation algorithm is based on one single audio example, without any prior knowledge about the geometry or the material properties of the object. The audio example should be the recording of an impact onto the object of interest, i.e. its empirical impulse response. The modal parameters are found through linear or nonlinear regression on the spectral features obtained by sinusoidal analysis.

The SAMPLE Python package (see Chapter 6) includes an implementation of the proposed algorithm.

3.2.1 Sinusoidal Analysis

Serra [87, 88, 89] introduced Spectral Modelling Synthesis (SMS), an analysis and synthesis system for musical sounds based on the decomposition of the sound into a deterministic sinusoidal and a stochastic

component. The two main components of the sinusoidal analysis algorithm are the peak detection and the peak continuation subroutines.

The peak detection algorithm detects peaks in each STFT frame of the analysed sound as a local maximum in the magnitude spectrum. The DFTs are computed with zero-phase windowing [82], so that local flatness of the phase spectrum can be used to detect the peaks. Optionally, a minimum value can be specified to exclude low-volume peaks. The peak location is refined with Gaussian interpolation, which is implemented as a parabolic interpolation over logarithmic decibel values [37].

The peak continuation algorithm organizes the peaks into temporal trajectories, with every trajectory representing the time-varying behaviour of a sinusoidal component. For all peaks in the frame, in decreasing order of magnitude, the active trajectory with the closest frequency is selected as the candidate trajectory, to which the peak should be appended. The peak is appended only if its frequency is close enough to the last peak's frequency. The threshold for acceptance can be dependent on the frequency itself. Usually, it is set to be more tolerant on higher frequencies. If no peak is appended to an active trajectory for a frame, the trajectory becomes inactive. If a peak cannot be appended to any active trajectory, a new trajectory is initialized, if it does not exceed the maximum limit of concurrent trajectories. Remaining peaks will be disregarded.

The residual power-spectral density is estimated with a line segment approximation of the difference between the original spectrum and the sinusoidal component, and modelled as noise. However, this stochastic component is not of interest for modal parameters estimate and will not be further discussed.

In Serra's general purpose implementation of SMS, inactive trajectories were able to be recycled and continued. In the prototypical implementation of SAMPLE, based on SMS, there was a cleaning step that split trajectories when the SMS algorithm recycled them. After that, trajectories with similar frequencies were considered as belonging to the same sinusoidal component and merged. But this cleaning steps were more computationally expensive than the rest of the SAMPLE algorithm.

In the current implementation of SAMPLE, the sinusoidal tracking

algorithm is customized for modal impulse responses. First of all, trajectories cannot be recycled, so that the splitting step can be skipped. Secondly, some sanity checks are performed when tracks are deactivated. The average frequency in the trajectory should fall inside the auditory range of interest (e.g. the default bounds are 20 Hz and 16 kHz). Also, the trajectory should be longer than a minimum length and are expected to show a decreasing trend in the magnitude. The slope coefficient from a linear regression is used as a feature to check this property. Finally, when a trajectory is deactivated, it can be appended to a previous trajectory if they are close enough in frequency.

A common trick used in audio analysis for additive synthesis [63], and also used in MPEG Audio [74] codecs, is to feed the audio to the algorithm with the time axis reversed (i.e. starting from the end of the sound). This avoids the detection of spurious spectral peaks in the attack transient of the sound, while allowing it to pick up the longest-lasting partials more robustly. It doesn't impose any limitation to the approach, because processing is always assumed to be batch. Finally, since all components should start at the same onset, any trajectory that starts significantly later is discarded.

3.2.2 Parameter Regression

Sinusoidal components of a modal impact sound are characterized by exponentially decaying amplitudes. As anticipated in Section 2.1.2, for a diagonalized modal system, the oscillation related to the i-th mode of oscillation is defined as in Equation (10).

$$h_i(t) = u(t) a_i e^{-\lambda_i t} \cos\left(2\pi \nu_i t + \phi_i\right)$$

Its parameters are is the modal frequency ν_i, the exponential decay coefficient λ_i, and the amplitude a_i.

The sinusoidal trajectories carry information about instantaneous frequency and magnitude of the components. So, let τ_i be the function that associates to every frame index n the instantaneous amplitude $a_{i,n}$ and frequency $\nu_{i,n}$ of the i-th component and $N(i)$ the set of valid frame indices for τ_i.

$$\tau_i : N(i) \subset \mathbb{N} \longrightarrow \mathbb{R}^2, n \mapsto \left(a_{i,n}, \nu_{i,n}\right) \tag{47}$$

The regression problem consists in finding the parameters ($\hat{\nu}_i$, $\hat{\lambda}_i$, and $\hat{a}_i$) of each mode that fit the corresponding trajectory.

$$\hat{a}_i e^{-\hat{\lambda}_i t(n)} \approx a_{i,n} \qquad \hat{\nu}_i \approx \nu_{i,n} \tag{48}$$

Here, $t(n)$ is the time instant associated to frame n.

The modal frequency is simply estimated as the average frequency of the trajectory.

$$\hat{\nu}_i := \sum_{n \in N(i)} \frac{\nu_{i,n}}{\#N(i)} \tag{49}$$

Let's consider the expression for instantaneous amplitude in Equation (48) and take the logarithm of both sides.

$$\ln \hat{a}_i - \hat{\lambda}_i t(n) \approx \ln a_{i,n} \tag{50}$$

This is a linear regression problem, so $\hat{\lambda}_i$ and $\hat{a}_i$ can be estimated as the slope and intercept coefficients, respectively. In practice, amplitudes are expressed in decibel, so the previous expression should use a base-10 logarithm.

$$20 \log_{10} \hat{a}_i - 20 \log_{10}(e) \cdot \hat{\lambda}_i t(n) \approx 20 \log_{10} a_{i,n} \tag{51}$$

Linear regression solves the following quadratic optimization problem, called Linear Least-Squares (Section 2.3.1).

$$\hat{k}_i, \hat{q}_i = \underset{k_i, q_i \in \mathbb{R}^2}{\arg\min} \sum_{n \in N(i)} \left| 20 \log_{10} a_{i,n} - k_i t(n) - q_i \right|^2 \tag{52}$$

The estimated amplitude and decay coefficient can be computed from the linear regression coefficients.

$$\hat{a}_i = 10^{\hat{q}_i/20} \qquad \hat{\lambda}_i = -\hat{k}_i / 20 \log_{10}(e) \tag{53}$$

Often modal models are described using the *decay time* ($d = 2\lambda^{-1}$).

$$d_i = -40 \log_{10}(e) / \hat{k}_i \tag{54}$$

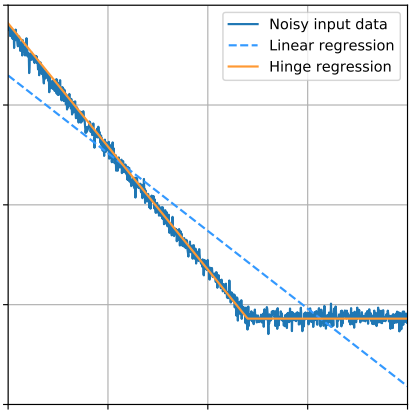

Figure 2: Comparison of fitting a linear function and a hinge function to noisy amplitude trajectory data when the noise floor stops the linear decay. The hinge regression allows for a better estimate of the linear parameters, while the linear regression is biased towards a lower value both the intercept and the slope.

Regression to a Hinge Function

When the noise floor is high, it could be detected by the sinusoidal tracking algorithm and its magnitude would contribute to the magnitude of the trajectory. This can happen especially if the input audio is not reversed. In other cases, reverberation changes the natural exponential decay of the signal.

In order to obtain a finer estimate of the linear parameters, the amplitude of the trajectory can be fitted to a hinge function using nonlinear least-squares estimate (as shown in Figure 2). A hinge function $h(t)$ is a function that is linear for $t < \alpha$ and then continues as a constant.

$$h_{k,q,\alpha}(t) := k \cdot \min(t, \alpha) + q \tag{55}$$

Nonlinear least-squares is not guaranteed to converge to a global optimum. Karjalainen *et al.* [51] included the noise in the model and defined a customized optimization method. In this approach most of the noise is excluded by the sinusoidal analysis and the optimizer is initialized with the parameter values found via linear least-squares, with α equal to half of the duration of the signal.

3.2.3 Dataset

A dataset was collected [105], containing 20 audio recordings from each of the 4 objects in Figure 1 (a bottle, saucepan, plank, and mug), for a total of 80 samples. They have been recorded in an acoustically isolated room at LIM with a Zoom H4 recorder. The sounds have been produced by manually hitting the objects with a wooden stick. These sounds were recorded by Giulia Clerici for her master thesis [21].

Additionally, five recordings of different bells sounds were used. They have been recorded by Daniel Simion and are available under Creative Commons Attribution 3.0 licence[1] on SoundBible[2]. Every recording has been cropped to a single repeat, starting right at the onset.

3.3 Results

The two datasets were analysed with both the proposed approach and SDTModalTracker, a modal parameters estimator that has been developed for the Sound Design Toolkit (Chapter 7). The sounds resynthesized using the parameters extracted with the two approaches were compared both objectively and subjectively.

All sounds have been analysed using the same hyperparameters. This serves as a reference for the baseline performance of the approach. An experienced sound-designer should tweak those hyperparameters based on the nature of the sound being analysed. The hyperparameters we used are the following: Hamming window (2048 points), FFT size 16384, hop size 256, magnitude threshold -80 dB, minimum sine duration 0.02 s, maximum number of sinusoidal components 64, frequency deviation offset 10 and slope 0.001, time delay threshold 0.1 s, initial magnitude threshold -60 dB (absolute), frequency boundaries 20 Hz and 18 kHz, using time-reversed audio, and a t_{60} threshold of 0 s (components with a negative decay time will be discarded). Please, note that the implementation under experiment was the prototype based on Xavier Serra's own Python implementation of SMS. Some variations from the reported results should be expected if testing the stable implementation in the SAMPLE Python package (Chapter 6).

[1] `https://creativecommons.org/licenses/by/3.0`
[2] `http://soundbible.com/tags-bell.html`

	μ **PCC**	σ	μ **NED**	σ
SDTModalTracker	0.225	0.293	0.494	0.077
SAMPLE	**0.643**	0.132	**0.351**	0.069

Table 1: Objective evaluation metrics macro-average and standard deviation. For both metrics, SAMPLE outperforms SDTModalTracker.

The original audio files and the outputs can be found on the GitLab Pages website of the development repository[3], as well as time-domain and time-frequency domain plots[4]. Plots for the in-between analysis steps are also available at the same page.

To resynthesize modal sounds, the Sound Design Toolkit's modal resonators were used. A Python interface for the relevant C API of SDT was developed with Cython [6], which made it possible to obtain the resynthesized modal audio in Python.

3.3.1 Objective Evaluation

To assess the resynthesis accuracy of the two methods, the resynthesized sounds have been compared to the original sounds using the mel-frequency cepstrogram (12 MFCCs are computed for every STFT frame, resulting in a time-cepstrum representation of the sound). For every MFCC, the Pearson correlation coefficient (PCC) is computed between the original and the resynthesized sound. Since the correlation is invariant to scale factors, another metric is employed to account for absolute differences, the normalized Euclidean dissimilarity (NED). This is a variant on the root mean squared error that is insensitive to bins for which both vectors are zero-valued and is always in the interval between zero and one.

$$\mathrm{NED}(A, B) = \frac{\|A - B\|_2}{\sqrt{2(\|A\|_2^2 + \|B\|_2^2)}}. \tag{56}$$

The first dataset (20 recordings for each of 4 objects) was used for this evaluation. Audio files by Daniel Simion (bell sounds) were not used because there were too few repetitions for each object. The values of

[3] `chromaticisobar.gitlab.io/pyaprsi2/audio`
[4] `chromaticisobar.gitlab.io/pyaprsi2/plots`

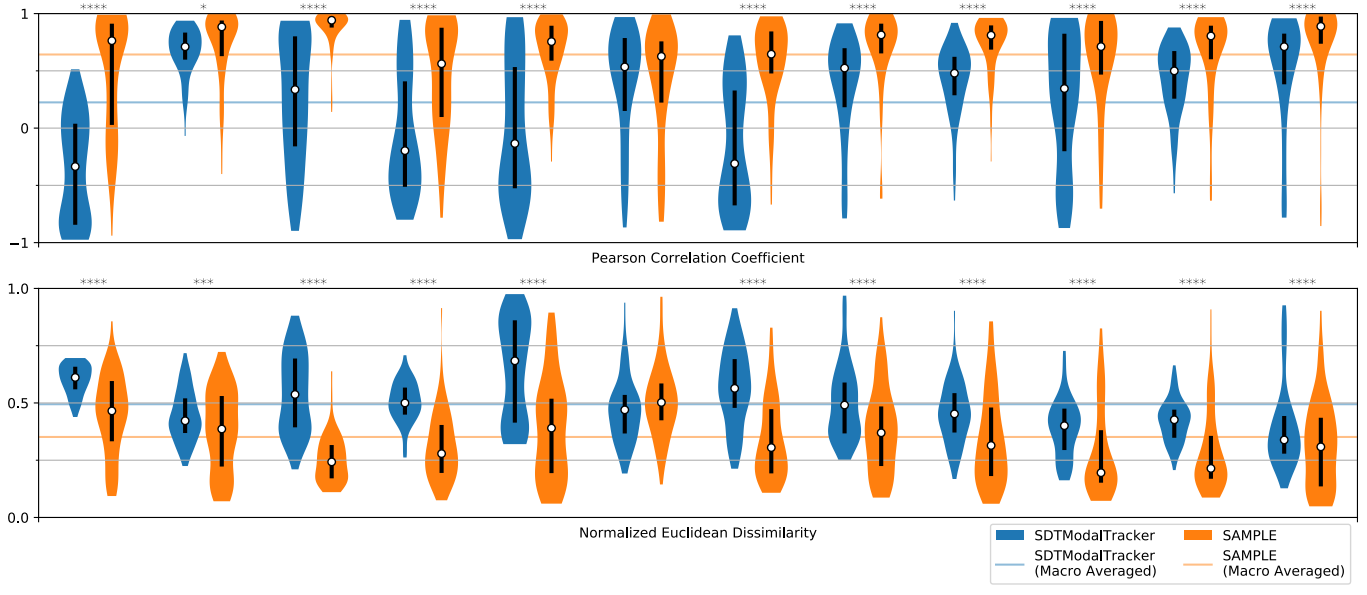

Figure 3: Violin plots for the Pearson correlation coefficient and the normalized Euclidean dissimilarity between the original sounds MFCCs and the resynthesized sounds for the first 12 MFCCs. Stars above the groups are indicative of the p-value of the paired-difference test.

each of the two metrics for both SAMPLE and SDTModalTracker are shown as violin plots in Figure 3: in the figure, stars indicate the level of significance for the p-value of the corresponding paired-difference test.

$$\begin{array}{llll} * & p \leq 0.05 & ** & p \leq 0.01 \\ *** & p \leq 0.005 & **** & p \leq 0.001 \end{array} \tag{57}$$

The Shapiro-Wilk test for normality [90] rejected the normality hypothesis for all metrics except the NED for MFCC-6. Because of this, the non-parametric Wilcoxon signed-rank test was used to test the significance of differences. For MFCC-6, the test did not show any significance neither for the PCC nor for the NED. But for every other MFCC, the difference is significant and in favour of the proposed method. The macro-average values are summarized in Table 1. It should be noted that the PCC is a measure of similarity (higher is better) and the NED is a measure of dissimilarity (lower is better).

3.3.2 Subjective Evaluation

A MUSHRA test [77] (*MUltiple Stimuli with Hidden Reference and Anchor*) has been set up to evaluate the subjective quality of the resynthesized sounds. It has been developed using webMUSHRA [85], a MUSHRA compliant web audio API based experiment software, and deployed using a webserver owned by LIM.

The listening test had 9 pages, one for each of the different sounds: one sample for each of the four objects in Figure 1, chosen at random, and the five bell sounds. In each page the listener is presented with one reference sound sample and 5 approximations to judge: the hidden reference itself, two anchors and two resynthesized sounds. The two anchors are computed from the reference sound by applying a low-pass filter at 7 kHz and 3.5 kHz, respectively. The two resynthesized sounds were generated with SDT using the modal parameters extracted with SAMPLE and SDTModalTracker, respectively.

The listener is asked to report on the quality of each stimulus with respect to the reference sound using a scale from 0 to 100. The listener is informed that the reference sound is hidden among the approximations. They are also encouraged to give a perfect score to a sample if

they think that it is, indeed, the reference. The listener can listen to the reference and to the approximations in any order and as many times as they like.

The listener is also asked to use headphones or studio monitors instead of their computer or phone built-in speakers, if possible. At the end of the test the listener must state what listening device they used (headphones, earphones, studio monitors or built-in speakers) and what is their audio background: no background, intermediate (e.g. student or amateur musician or audio engineer) or expert (e.g. musician or audio engineer). Optional information about gender and age can be input, along with a feedback message. All this information is saved anonymously.

We received 12 entries, two of which have been discarded according to the MUSHRA guideline because they rated the hidden reference below 90 MUSHRA points for more than 15 percent of all test items. One of them reported having no audio background, the other reported being an intermediate. The remaining 10 participants were so divided: 2 no-background, 3 intermediate and 5 experts. Only the discarded intermediate used studio monitors, and all other participants used either headphones or earphones. Although the test was not conducted in a controlled environment, these conditions and support have been considered sufficient to draw macroscopic conclusions.

The results for all the valid entries are summarized using violin plots in Figure 4. The sound resynthesized using the parameters extracted with SAMPLE have been considered as *Fair* in the average case. The distribution of the scores is also very similar to the 3.5 kHz low-pass filtered anchor and their differences are not significant ($p > 0.05$ for the Wilcoxon signed-rank test). The 7 kHz low-pass filtered anchor and the hidden reference were given better scores (*Good* and *Excellent*, respectively). The sound resynthesized using the parameters extracted with SDTModalTracker have been considered as *Poor* in the average case, that is, significantly worse than SAMPLE ($p \leq 0.001$ for the Wilcoxon signed-rank test).

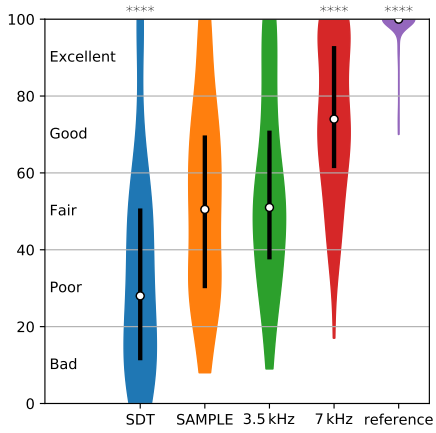

Figure 4: Violin plots for the MUSHRA test scores. Stars above the groups are indicative of the p-value of the paired-difference test, where significant. The audio samples resynthesized with parameters extracted with the proposed method have been judged similar to the 3.5 kHz anchor and better than SDT.

3.4 Conclusion

This chapter described an approach for estimating modal parameters, which uses only one audio recording as a sample. The method outperformed a publicly available open-source method in the task of parameter estimation for modal synthesis. The two methods were compared with computational and subjective tests.

The proposed method does not require any previous knowledge of the geometry or materials of the object and also requires no more than one audio recording of the sound of the impacted object. However, it is only applicable to rigid resonators and it is not robust to external factors such as reverberation. Furthermore, the system that contributes to the sound is modelled as a whole: the contribution of the resonator, the impacting object and any support are not disentangled. In addition, the method only models the contribution of the mode when it is impacted at one impact point and recorded at one pickup point.

This approach does not model nonlinear phenomena, such as coupling, and non-stationary modes. Modal systems that have a significantly nonlinear behaviour should be analysed by addressing the specific nonlinearities. Those nonlinearities should also be implemented in the synthesis model. The parametrization of transients is not addressed and left to the sound-designers. The main motivation of this choice is that it would require a complementary approach (e.g., it could be addressed using information from the residual component). Also, in modal synthesisers such as SDT's there is a small number of transient parameters, that can easily be tweaked by the sound-designer.

Assumptions of modal analysis which were not implemented in the software prototype, such as stationarity of modes and monotonic decreasing amplitude, have been successfully integrated directly in the sinusoidal model. Also, a GUI was then developed to allow for the fine-tuning of hyperparameters and modal parameters and ease the access for sound-designers who are not familiar with Python (see Chapter 6).

Beats can be included in the model, allowing for the resolution of partials that are grouped together in the STFT. This is performed as an alternative fitting step after sinusoidal analysis (see Chapter 4). The model could be generalized to the case in which more than one example is available, to improve robustness. Similarly, a generalized model could accept optional prior information about the geometry and the material distribution of the object, to extend the model to a space-time process, or multichannel samples (see Chapter 5).

Chapter 4

Modal Acoustic Beats

Abstract

Acoustic "beats" are one of the most extensively studied interference patterns. A beat arises from two pure sinusoidal tones with the same volume that oscillate at two similar frequencies, but is heard it as one tone, whose volume periodically changes with time. Beats represent a challenge for current state-of-the-art signal analysis methods, especially for the inverse problem of estimating "modal" parameters from audio signals. A baseline algorithm was designed to exploits the information from the amplitude modulation of a beat to estimate the parameters of two beating modal tones. But when the two tones have different amplitudes, then the frequency of the tone also changes with time. Building on this fact, the algorithm was, then, refined by also considering the frequency modulation pattern. The performance of the two algorithms was evaluated on a synthetic dataset of exponentially-decaying beating sine-waves, finding found statistical evidence that the proposed method better estimates the modal parameters.

This chapter is based on [102] Marco Tiraboschi and Federico Avanzini. 'Acoustic Beats and Where To Find Them: Uneven Beats Models and Modal Audio Inversion by Non-linear Least-Squares Optimization'. In: *Inverse Problems* (2024). [UNDER REVIEW]

4.1 Introduction

Most people who took an introductory physics class will remember a lecture about interference patterns and beats. The subject is a teacher's favourite, because the maths is simple enough that a high-school student can keep up, and the phenomenon can be easily shown: you can play the same note on two different strings on a guitar, one of which is slightly out-of-tune, and you can hear the beat. Beats can be also commonly found in sounds made from a variety of real objects. Because of their imperfections, they often have resonant modes the frequencies of which do not coincide, but are very close to each other.

With the aim of achieving more believable and more realistic sound synthesis models that accurately replicate the beats real-life objects produce, the problem investigated in this chapter extends the work described in Chapter 3. Here, an expression for beats is discussed, focussing on the case where the two amplitudes are different or time-varying, especially the case of exponential decay. This model represents the sum of two such sinusoidal components as a single component whose volume and frequency are modulated with the same modulation frequency.

In different contexts, such as telecommunications or civil engineering, researchers have been using models based on amplitude and frequency modulated partials to represent many kinds of non-stationary or non-sinusoidal signals. The literature on Empirical Mode Decomposition (EMD) is rich of adaptive and nonlinear algorithms for the analysis of such signals. While starting on similar theoretical grounds, the proposed model is simpler because it exploits some knowledge about the signals' behaviour in the domain of application. After establishing such theoretical framework, some of the most used methods for performing spectral decomposition are discussed: Fourier analysis, autoregressive modelling, and EMD. Their limitations in the context of separating beating components are shown.

Then, two variants of a modal parameter estimation algorithm are defined. SAMPLE (see Chapter 3) is applied for sinusoidal analysis, to track the instantaneous frequency and amplitude of beating components. Then, nonlinear least-squares regression fits the parameters of the beat model to the trajectories. One variant fits the parameters only accounting for the amplitude modulation, the other also accounts for

the frequency modulation. The proposed algorithm was named Beats-DROP (*Beats Duality for the Resolution Of Partials*).

A synthetic dataset of beating sounds was generated for regression performance evaluation. Every beat instance is the sum of two sinusoidal components with exponential decay (see Equation (10) in Chapter 2). Finally, it is shown that the algorithm intrinsically provides a decision criterion for determining if the tracked instantaneous frequencies and amplitudes belong to a single component or to a pair of beating components. A second synthetic dataset, constructed similarly to the first, was used to quantitatively assess the classification performance of such criterion.

The implementation of these methods is included in the SAMPLE Python package (see Chapter 6). The repository on Zenodo [99] also includes the evaluation protocols. Experimental data for this chapter are also available on Zenodo [100].

4.2 Interference Patterns

In this section, the core model is discussed. This is an expression for the interference of two pure sinusoidal tones with different stationary frequencies and different, possibly modulated, amplitudes. This pattern will be referred to as the *uneven beat*, to distinguish it from the more commonly studied beat pattern, the *perfect beat*, in which the two tones have the same amplitude.

This interference is well known in the literature. The expression by Feldman [32] (equations 10 and 11) is very similar to the one considered here, but it requires $a_1 > a_2$ and $\omega_2 > \omega_1$. Boashash [11] (equations 28–30) had previously derived another similar set of equations, but for complex partials. The specific expression used in this chapter is proven in Section 4.2.1 (Theorem 1).

$$\begin{aligned} f(t) &= a_1 \cos\left(\omega_1 t + \phi_1\right) + a_2 \cos\left(\omega_2 t + \phi_2\right) = \\ &= 2\alpha(t) \cos\left(\phi_0 + \int_0^t \omega(x)\, dx\right) \end{aligned} \tag{58}$$

The instantaneous amplitude is $2\alpha(t)$.

$$\alpha^2(t) := \left(\overline{a}^2 - \hat{a}^2\right) \cos^2\left(\hat{\omega} t + \hat{\phi}\right) + \hat{a}^2 \tag{59}$$

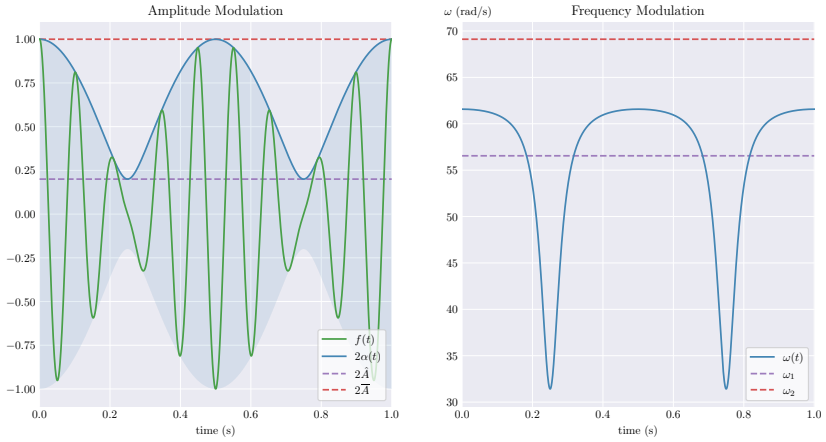

Figure 5: Sum of two sinusoidal functions: one has amplitude $a_1 = 0.6$ and angular velocity $\omega_1 = 18\pi$ rad/s, the other has amplitude $a_2 = 0.4$ and angular velocity $\omega_2 = 22\pi$ rad/s.

The instantaneous angular velocity is $\omega(t)$.

$$\omega(t) := \overline{\omega} + \frac{\overline{a}\hat{a}}{\alpha^2(t)}\hat{\omega} \tag{60}$$

The initial phase is ϕ_0.

$$\phi_0 := \overline{\phi} + \operatorname{arctan2}\left(\hat{a}\sin\hat{\phi},\ \overline{a}\cos\hat{\phi}\right) \tag{61}$$

As a convention, the letter ω is used for angular velocities, the letter a for amplitudes and the letter ϕ for phases. Later in the paper, the letter ν will be used for frequencies in hertz ($\omega = 2\pi\nu$), if they are more convenient to use. Letters with an overline indicate averages and letters with a hat indicate semi-differences.

$$\begin{aligned} \overline{a} &:= \frac{a_1 + a_2}{2} & \overline{\omega} &:= \frac{\omega_1 + \omega_2}{2} & \overline{\phi} &:= \frac{\phi_1 + \phi_2}{2} \\ \hat{a} &:= \frac{a_1 - a_2}{2} & \hat{\omega} &:= \frac{\omega_1 - \omega_2}{2} & \hat{\phi} &:= \frac{\phi_1 - \phi_2}{2} \end{aligned} \tag{62}$$

Figure 5 displays an example of the amplitude and frequency modulations happening in uneven beats. Let us pause a moment to appreciate

the duality of the pattern: the interference of two tones with different frequency determines an amplitude modulation and, if the two tones have different amplitudes, it also yields a frequency modulation.

Section 4.2.1 also proves the expression for amplitude-modulated components (Theorem 2). In this case the instantaneous angular velocity is $\omega(t)$.

$$\omega(t) = \overline{\omega} + \alpha^{-2}(t) \left[\overline{a}(t)\hat{a}(t)\hat{\omega} + \right. \\ \left. + \left(\frac{d\hat{a}(t)}{dt}\overline{a}(t) - \frac{d\overline{a}(t)}{dt}\hat{a}(t) \right) \frac{\sin\left(2\hat{\omega}t + 2\hat{\phi}\right)}{2} \right] \quad (63)$$

The initial phase is ϕ_0.

$$\phi_0 = \overline{\phi} + \arctan2\left(\hat{a}(0)\sin\hat{\phi},\ \overline{a}(0)\cos\hat{\phi}\right) \quad (64)$$

4.2.1 Proof

This section proves the expression for the interference of two pure sinusoidal components with different frequencies and amplitudes (uneven beats). The proof exploits some known facts about two simpler interference patterns, the perfect beat and the out-of-phase interference (we omit the proof of these lemmas). The perfect beat interference pattern can be expressed by the following equality:

$$a\cos(\omega_1 t + \phi_1) + a\cos(\omega_2 t + \phi_2) = \\ = 2a\cos\left(\overline{\omega}t + \overline{\phi}\right)\cos\left(\hat{\omega}t + \hat{\phi}\right) \quad (65)$$

The symbols are explained in Equation (62). The out-of-phase interference can be expressed with phasor addition.

$$a_1\cos(\omega t + \phi_1) + a_2\cos(\omega t + \phi_2) = a_3\cos(\omega t + \phi_3) \quad (66)$$

$$a_3 = \left|a_1 e^{j\phi_1} + a_2 e^{j\phi_2}\right| \qquad \phi_3 = \angle\left(a_1 e^{j\phi_1} + a_2 e^{j\phi_2}\right)$$

Uneven Beats Interference Pattern

These building blocks can be used to find an expression for the interference pattern of uneven beats: when two pure sinusoidal tones with difference amplitudes interfere, they are equivalent to a single tone that changes volume and frequency with time. Expressions for the amplitude modulation are very common and, for a simple proof, we suggest the Feynman lectures [33] (equations 48.1–48.8). But this expression will also take into account the frequency modulation component.

Theorem 1. *The sum of two sinusoidal functions with different, but constant, amplitudes and frequencies can be expressed as one sinusoidal function periodically modulated in amplitude and frequency. The frequency of both modulation signals is equal to the difference of the two initial frequencies. The amplitude values range between the difference and the sum of the two initial amplitude values. The carrier frequency is the average of the two initial frequencies.*

$$a_1 \cos(\omega_1 t + \phi_1) + a_2 \cos(\omega_2 t + \phi_2) = \\ = 2\alpha(t) \cos\left(\phi_0 + \int_0^t \omega(x)dx\right)$$

where

$$\alpha^2(t) := \left(\overline{a}^2 - \hat{a}^2\right) \cos^2\left(\hat{\omega}t + \hat{\phi}\right) + \hat{a}^2$$

$$\omega(t) := \overline{\omega} + \frac{\overline{a}\hat{a}}{\alpha^2(t)}\hat{\omega}$$

$$\phi_0 := \overline{\phi} + \arctan2\left(\hat{a}\sin\hat{\phi},\ \overline{a}\cos\hat{\phi}\right)$$

$$\overline{a} := \frac{a_1 + a_2}{2} \qquad \overline{\omega} := \frac{\omega_1 + \omega_2}{2} \qquad \overline{\phi} := \frac{\phi_1 + \phi_2}{2}$$

$$\hat{a} := \frac{a_1 - a_2}{2} \qquad \hat{\omega} := \frac{\omega_1 - \omega_2}{2} \qquad \hat{\phi} := \frac{\phi_1 - \phi_2}{2}$$

Proof. The whole function can be expressed as the sum of two *perfect beat* patterns.

$$f(t) = \quad a_1 \cos(\omega_1 t + \phi_1) + \quad a_2 \cos(\omega_2 t + \phi_2) = \tag{67}$$

$$= (\overline{a} + \hat{a}) \cos(\omega_1 t + \phi_1) + (\overline{a} - \hat{a}) \cos(\omega_2 t + \phi_2) \tag{68}$$

$$f(t) = \overline{a}\Big[\cos(\omega_1 t + \phi_1) + \cos(\omega_2 t + \phi_2) \quad\Big] + \\ + \hat{a}\Big[\cos(\omega_1 t + \phi_1) + \cos(\omega_2 t + \phi_2 + \pi)\Big] \tag{69}$$

$$f(t) \overset{(65)}{=} 2\Bigg\{\overline{a}\cos\Big(\hat{\omega}t + \hat{\phi}\Big) \quad \cos\Big(\overline{\omega}t + \overline{\phi}\Big) + \\ + \hat{a}\cos\Big(\hat{\omega}t + \hat{\phi} - \frac{\pi}{2}\Big)\cos\Big(\overline{\omega}t + \overline{\phi} + \frac{\pi}{2}\Big)\Bigg\} \tag{70}$$

This is the sum of two out-of-phase sine-waves with different amplitudes, even though the amplitudes themselves happen to be sine-waves. So, using Equation (66).

$$f(t) = 2\alpha(t)\cos\Big(\overline{\omega}t + \overline{\phi} + \tau(t)\Big) \tag{71}$$

$$\alpha(t)\, e^{j\left[\tau(t)+\overline{\phi}\right]} = \overline{a}\cos\Big(\hat{\omega}t + \hat{\phi}\Big)\, e^{j\overline{\phi}} + \\ + \hat{a}\cos\Big(\hat{\omega}t + \hat{\phi} - \pi/2\Big)\, e^{j\left(\overline{\phi}+\pi/2\right)} \tag{72}$$

This equation can be simplified removing the phase shift of $\overline{\phi}$ from both sides of the equation, rewriting $\exp(j\pi/2)$ as j and applying the trigonometric identity $\cos(x - \pi/2) = \sin(x)$.

$$\alpha(t)e^{j\tau(t)} = \overline{a}\cos\Big(\hat{\omega}t + \hat{\phi}\Big) + j\hat{a}\sin\Big(\hat{\omega}t + \hat{\phi}\Big) \tag{73}$$

The modulant signal for the amplitude modulation is $\alpha(t)$.

$$\alpha^2(t) = \Re^2\Big[\alpha(t)e^{j\tau(t)}\Big] + \Im^2\Big[\alpha(t)e^{j\tau(t)}\Big] \quad = \tag{74}$$

$$= \overline{a}^2\cos^2\Big(\hat{\omega}t + \hat{\phi}\Big) + \hat{a}^2\sin^2\Big(\hat{\omega}t + \hat{\phi}\Big) = \tag{75}$$

$$= \Big(\overline{a}^2 - \hat{a}^2\Big)\cos^2\Big(\hat{\omega}t + \hat{\phi}\Big) + \hat{a}^2 \tag{76}$$

The modulant signal for the phase modulation is $\tau(t)$.

$$\tau(t) = \operatorname{arctan2}\Big(\Im\Big[\alpha(t)e^{j\tau(t)}\Big], \Re\Big[\alpha(t)e^{j\tau(t)}\Big]\Big) = \tag{77}$$

$$= \operatorname{arctan2}\Big(\hat{a}\sin\Big(\hat{\omega}t + \hat{\phi}\Big), \overline{a}\cos\Big(\hat{\omega}t + \hat{\phi}\Big)\Big) \tag{78}$$

The instantaneous angular velocity is the phase derivative.

$$\omega(t) := \frac{d}{dt}\left[\overline{\omega}t + \overline{\phi} + \tau(t)\right] = \overline{\omega} + \frac{d\tau}{dt}(t) \tag{79}$$

The function arctan2 (y, x) is equal up to a constant to arctan (y/x) for all y and for all $x \neq 0$. So, we will take the derivative of $\tilde{\tau}(t)$.

$$\tilde{\tau}(t) := \arctan\left(\frac{\hat{a}\sin\left(\hat{\omega}t + \hat{\phi}\right)}{\overline{a}\cos\left(\hat{\omega}t + \hat{\phi}\right)}\right) = \arctan\left(\frac{\hat{a}}{\overline{a}}\tan\left(\hat{\omega}t + \hat{\phi}\right)\right) \tag{80}$$

Where it exists, the derivative of $\tilde{\tau}$ is equal to the derivative of τ. The argument of the tan function is $\theta(t) := \hat{\omega}t + \hat{\phi}$, and its derivative is just $\hat{\omega}$. The argument of arctan is $\xi(t) := \hat{a}/\overline{a} \cdot \tan\theta(t)$.

$$\frac{d\xi}{dt}(t) = \frac{\hat{a}}{\overline{a}}\frac{d\tan}{d\theta}(\theta(t))\frac{d\theta}{dt}(t) = \frac{\hat{a}\hat{\omega}}{\overline{a}\cos^2\left(\hat{\omega}t + \hat{\phi}\right)} \tag{81}$$

The derivative of $\tilde{\tau}$ can be built using the chain rule.

$$\begin{aligned}
\frac{d\tilde{\tau}}{dt}(t) &= \frac{d\arctan}{d\xi}(\xi(t))\frac{d\xi}{dt}(t) && = && (82)\\
&= \frac{1}{1+\xi^2(t)}\frac{\hat{a}\hat{\omega}}{\overline{a}\cos^2\left(\hat{\omega}t + \hat{\phi}\right)} && = && (83)\\
&= \left\{1 + \left[\frac{\hat{a}}{\overline{a}}\tan\left(\hat{\omega}t + \hat{\phi}\right)\right]^2\right\}^{-1}\frac{\hat{a}\hat{\omega}}{\overline{a}\cos^2\left(\hat{\omega}t + \hat{\phi}\right)} && = && (84)\\
&= \frac{\overline{a}\hat{a}\hat{\omega}}{\overline{a}^2\cos^2\left(\hat{\omega}t + \hat{\phi}\right) + \hat{a}^2\sin^2\left(\hat{\omega}t + \hat{\phi}\right)} \overset{(75)}{=} \frac{\overline{a}\hat{a}\hat{\omega}}{\alpha^2(t)} && && (85)
\end{aligned}$$

So, the frequency modulation is $\omega(t)$.

$$\omega(t) = \overline{\omega} + \frac{d\tau}{dt}(t) = \overline{\omega} + \frac{d\tilde{\tau}}{dt}(t) = \overline{\omega} + \frac{\overline{a}\hat{a}}{\alpha^2(t)}\hat{\omega} \tag{86}$$

Finally, the value of the phase modulation function at the time instant $t = 0$ is needed to compute ϕ_0.

$$\tau(0) = \text{arctan2}\left(\hat{a}\sin\hat{\phi}, \overline{a}\cos\hat{\phi}\right) \tag{87}$$

Quod erat demonstrandum.

$$a_1 \cos(\omega_1 t + \phi_1) + a_2 \cos(\omega_2 t + \phi_2) = 2\alpha(t) \cos \phi(t) \tag{88}$$

$$\alpha^2(t) = \left(\overline{a}^2 - \hat{a}^2\right) \cos^2\left(\hat{\omega} t + \hat{\phi}\right) + \hat{a}^2 \tag{89}$$

$$\phi(t) = \phi_0 + \int_0^t \left[\overline{\omega} + \overline{a}\hat{a}\hat{\omega}/\alpha^2(x)\right] dx \tag{90}$$

$$\phi_0 = \overline{\phi} + \arctan2\left(\hat{a} \sin \hat{\phi}, \overline{a} \cos \hat{\phi}\right) \tag{91}$$

□

Modulated Beats Interference Pattern

This expression can be modified to describe the case when the amplitudes of the two interfering waves change with time.

Theorem 2. *The sum of two sinusoidal functions with different, but constant, frequencies, and time-varying amplitudes can be expressed as one sinusoidal function modulated in amplitude and frequency.*

$$a_1(t) \cos(\omega_1 t + \phi_1) + a_2(t) \cos(\omega_2 t + \phi_2) = \\ = 2\alpha(t) \cos\left(\phi_0 + \int_0^t \omega(x) dx\right)$$

where

$$\alpha^2(t) := \left(\overline{a}^2(t) - \hat{a}^2(t)\right) \cos^2\left(\hat{\omega} t + \hat{\phi}\right) + \hat{a}^2(t)$$

$$\omega(t) := \overline{\omega} + \frac{1}{\alpha^2(t)} \left[\overline{a}(t)\hat{a}(t)\hat{\omega} + \left(\frac{d\hat{a}(t)}{dt}\overline{a}(t) + \right.\right. \\ \left.\left. - \frac{d\overline{a}(t)}{dt}\hat{a}(t)\right) \frac{\sin\left(2\hat{\omega} t + 2\hat{\phi}\right)}{2}\right]$$

$$\phi_0 := \overline{\phi} + \arctan2\left(\hat{a}(0) \sin \hat{\phi},\ \overline{a}(0) \cos \hat{\phi}\right)$$

$$\overline{a}(t) := \frac{a_1(t) + a_2(t)}{2} \qquad \overline{\omega} := \frac{\omega_1 + \omega_2}{2} \qquad \overline{\phi} := \frac{\phi_1 + \phi_2}{2}$$

$$\hat{a}(t) := \frac{a_1(t) - a_2(t)}{2} \qquad \hat{\omega} := \frac{\omega_1 - \omega_2}{2} \qquad \hat{\phi} := \frac{\phi_1 - \phi_2}{2}$$

Proof. The proof for Theorem 1 holds for all steps until we compute the derivative of the phase modulation in Equation (81). In the case of amplitude-modulated partials, the derivative of the components' amplitude affects the derivative of ξ.

$$\frac{d\xi}{dt}(t) = \frac{d}{dt}\left\{\frac{\hat{a}(t)}{\overline{a}(t)}\tan\left(\theta(t)\right)\right\} \tag{92}$$

$$\frac{d\xi}{dt}(t) = \frac{\hat{a}(t)}{\overline{a}(t)}\frac{d\tan}{dt}\left(\theta(t)\right) + \frac{d\left(\hat{a}/\overline{a}\right)}{dt}(t)\tan\left(\theta(t)\right) \tag{93}$$

$$\frac{d\xi}{dt}(t) = \frac{\hat{a}(t)\hat{\omega}}{\overline{a}(t)\cos^2\left(\theta(t)\right)} + \\ + \frac{\frac{d\hat{a}(t)}{dt}\overline{a}(t) - \frac{d\overline{a}(t)}{dt}\hat{a}(t)}{\overline{a}^2(t)}\tan\left(\theta(t)\right) \tag{94}$$

Consequently, the derivative of $\tilde{\tau}$ also changes.

$$\frac{d\tilde{\tau}}{dt}(t) = \frac{d\arctan}{d\xi}(\xi(t))\frac{d\xi}{dt}(t) \tag{95}$$

$$= \frac{\overline{a}^2(t)\cos^2\left(\theta(t)\right)d\xi(t)/dt}{\overline{a}^2(t)\cos^2\left(\theta(t)\right) + \hat{a}^2(t)\sin^2\left(\theta(t)\right)} \tag{96}$$

$$= \frac{\overline{a}(t)\hat{a}(t)\hat{\omega} + \left(\frac{d\hat{a}(t)}{dt}\overline{a}(t) - \frac{d\overline{a}(t)}{dt}\hat{a}(t)\right)\frac{\sin(2\theta(t))}{2}}{\alpha^2(t)} \tag{97}$$

Quod erat demonstrandum.

$$\omega(t) = \overline{\omega} + \frac{1}{\alpha^2(t)}\left[\overline{a}(t)\hat{a}(t)\hat{\omega}\left(\frac{d\hat{a}(t)}{dt}\overline{a}(t) + \right.\right. \\ \left.\left. - \frac{d\overline{a}(t)}{dt}\hat{a}(t)\right)\frac{\sin\left(2\hat{\omega}t + 2\hat{\phi}\right)}{2}\right] \tag{98}$$

$$\phi_0 = \overline{\phi} + \tau(0) = \overline{\phi} + \text{arctan2}\left(\hat{a}(0)\sin\hat{\phi},\ \overline{a}(0)\cos\hat{\phi}\right) \tag{99}$$

□

4.3 State-of-the-Art

In this section, several methods that address the problem of separating partials with an arbitrarily small frequency difference are discussed. A theoretical discussion and experimental results motivate the final choice of developing a specialized algorithm.

4.3.1 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is arguably the most common spectral analysis method (see Section 2.2.1 for a summary on the DFT). Given a discrete signal with n samples, sampled at a rate of ν_s, its *duration* is $T := n/\nu_s$. The frequency resolution of the DFT, i.e. the width of the DFT bins, is the reciprocal of the signal duration $\Delta\nu = \nu_s/n = T^{-1}$.

Under the assumptions that the two oscillations are stationary, if the frequency difference between the two components $2\,|\hat{\nu}|$ (where $2\hat{\nu} = \nu_1 - \nu_2$) is larger than $\Delta\nu$, then the energy peak of the two spectra will belong to two different DFT bins. In order for the two peaks to be detected as local maxima in the DFT, there must be at least one bin in-between. This means that the frequency difference should be more than double the resolution $2\,|\hat{\nu}| > 2\Delta\nu$, which implies $T > |\hat{\nu}|^{-1}$. In other words, the signal must include at least two full cycles of the amplitude modulation (the frequency of the amplitude modulation is $2\,|\hat{\nu}|$).

Even in this simple setting, it becomes evident that very large DFT sizes are needed. To resolve a beat at 2 Hz the DFT size should be at least 1 second, which is already much larger than common window sizes: with a sample rate of 44.1 kHz, the lowest sufficient power of two would be 65 536 samples. For some audio software, like AudioSculpt [13], this is already the DFT size limit. But beats can also be slower, which would require a larger DFT.

Fourier Transform of Damped Sine Waves

However, this is only true for stationary sounds. The signals of interest are causal and exponentially-decaying (see Equation (10)). They are also known as *damped* oscillations, because they describe the behaviour of a mechanical oscillator subject to energy dissipation (see

Section 2.1.1). A causal exponential function is

$$f(t) = u(t)e^{-\lambda t} \tag{100}$$

The Fourier Transform and its square magnitude are known.

$$\mathcal{F}\left\{u(t)e^{-\lambda t}\right\}(\nu) = \int_0^{+\infty} e^{-\lambda t}e^{-2j\pi\nu t}dt = \frac{1}{\lambda + 2j\pi\nu} \tag{101}$$

$$\left|\mathcal{F}\left\{u(t)e^{-kt}\right\}(\nu)\right|^2 = \frac{1}{\lambda^2 + 4\pi^2\nu^2} \tag{102}$$

The magnitude peak occurs at $\nu = 0$.

$$\max_{\nu}\left|\mathcal{F}\left\{u(t)e^{-\lambda t}\right\}(\nu)\right| = \left|\mathcal{F}\left\{u(t)e^{-\lambda t}\right\}(0)\right| = \frac{1}{\lambda} \tag{103}$$

The half-power (3 dB) bandwidth β is double of the distance from the peak where the magnitude drops by half.

$$\beta = \frac{\sqrt{3}\lambda}{\pi} \tag{104}$$

$$\left|\mathcal{F}\left\{u(t)e^{-\lambda t}\right\}\left(\frac{\sqrt{3}\lambda}{2\pi}\right)\right| = \sqrt{\lambda^2 + 4\pi^2\left(\frac{\sqrt{3}\lambda}{2\pi}\right)^2}^{-1} = \frac{1}{2\lambda} \tag{105}$$

Let $f_{a,d,\nu_0,\phi}$ be a causal, exponentially-decaying cosine function with initial amplitude a, decay time $d = 2/\lambda$, frequency ν_0, and initial phase ϕ.

$$f_{a,d,\nu_0,\phi}(t) := u(t)ae^{-2t/d}\cos\left(2\pi\nu_0 t + \phi\right) \tag{106}$$

Its Fourier Transform is $F_{a,d,\nu_0,\phi} := \mathcal{F}\left\{f_{a,d,\nu_0,\phi}(t)\right\}$.

$$F_{a,d,\nu_0,\phi}(\nu) = \frac{a}{4}\left\{\frac{e^{-j\phi}}{d^{-1} + j\pi(\nu - \nu_0)} + \frac{e^{j\phi}}{d^{-1} + j\pi(\nu + \nu_0)}\right\} \tag{107}$$

Fourier Transform of Damped Beats

The sum of two damped sine-waves will have eight parameters.

$$\mathcal{F}\left\{f_{a_1,d_1,\nu_1,\phi_1} + f_{a_2,d_2,\nu_2,\phi_2}\right\}(\nu) = F_{a_1,d_1,\nu_1,\phi_1}(\nu) + F_{a_2,d_2,\nu_2,\phi_2}(\nu) \tag{108}$$

To simplify the discussion, let there be some assumptions. In the simplest case, the two components are in-phase with each other ($\phi_1 = \phi_2 =$

0). Let the beat be *perfect*, such that the two amplitudes and the two decay times are the same ($a_1 = a_2 = 1/2$ and $d_1 = d_2 = d$). Now, only three parameters are left: the common decay time d, the frequency average $\overline{\nu} = (\nu_1 + \nu_2)/2$ and the frequency semi-difference $\hat{\nu} = (\nu_1 - \nu_2)/2$.

$$B_{d,\overline{\nu},\hat{\nu}}(\nu) = F_{1/2,d,\overline{\nu}-\hat{\nu},0}(\nu) + F_{1/2,d,\overline{\nu}+\hat{\nu},0}(\nu) \quad = \quad (109)$$

$$= \frac{1}{8}\left\{ \frac{1}{d^{-1} + j\pi(\nu - \overline{\nu} + \hat{\nu})} + \frac{1}{d^{-1} + j\pi(\nu + \overline{\nu} - \hat{\nu})} \quad + \quad (110)\right.$$

$$\left. + \frac{1}{d^{-1} + j\pi(\nu - \overline{\nu} - \hat{\nu})} + \frac{1}{d^{-1} + j\pi(\nu + \overline{\nu} + \hat{\nu})} \right\}$$

This is the sum of four complex-valued bell-shaped curves, centred at $\overline{\nu} + \hat{\nu}$, $\overline{\nu} - \hat{\nu}$, $\hat{\nu} - \overline{\nu}$, and $-\overline{\nu} - \hat{\nu}$. The peak magnitude value of each component is $d/8$. From Equation (104), the half-power bandwidth is $\beta = 2\sqrt{3}/\pi d$. The bands centred at $\overline{\nu} + \hat{\nu}$ and at $\overline{\nu} - \hat{\nu}$ overlap if the frequency difference is less than the bandwidth.

$$2\hat{\nu} < \beta = \frac{2\sqrt{3}}{\pi d} \quad \Rightarrow \quad d\hat{\nu} < \frac{\sqrt{3}}{\pi} \tag{111}$$

If the difference is less than half the bandwidth, then each of the two peaks is inside the other component's band.

$$2\hat{\nu} < \frac{\beta}{2} = \frac{\sqrt{3}}{\pi d} \quad \Rightarrow \quad d\hat{\nu} < \frac{\sqrt{3}}{2\pi} \tag{112}$$

Figure 6 shows a comparison between Fourier-Transform (FT) analysis and SAMPLE on synthetic examples. Each example sound is composed of two modes with the same amplitude, decay, and phase, but two different frequencies (the average frequency is $\overline{\nu} = 50\,\text{Hz}$ and the difference is $2\hat{\nu} = 1\,\text{Hz}$). Each example has a different decay time (150, 300, 450, and 600 ms). In the first plot, the two FT peaks for low decay times are less distinct or even not distinguishable at all. From Equation (111) and Equation (112) we know that the two half-power bands overlap for decay times less than $2\sqrt{3}/\pi \approx 1.1\,\text{s}$ and that each peak is inside the other component's band for decay times less than 551 ms. The third plot shows that SAMPLE can follow the amplitude and frequency envelopes. From these envelopes we can infer the frequency difference as the frequency of the beat.

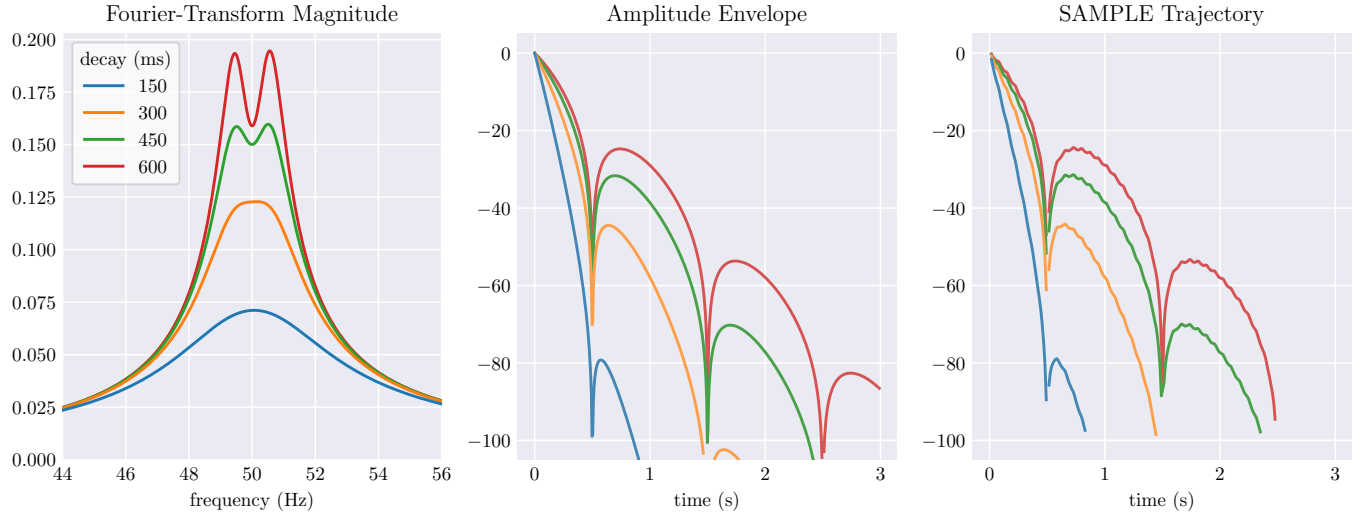

Figure 6: Comparison between Fourier-Transform (FT) analysis and SAMPLE on synthetic examples. For low decay times the two FT peaks are less distinct or even not distinguishable at all. Contrarily, SAMPLE can follow all these signals' envelopes.

This highlights how the FT cannot disentangle causal and exponentially-decaying beating sinusoids for short decay times and small frequency differences. For a peak detection algorithm to be successful in discriminating the two components in the FT, at least $d\hat{\nu} > \sqrt{3}/\pi$ should hold, or preferably $d\hat{\nu} > \sqrt{3}/2\pi$. This limit also affects the Discrete Fourier Transform, but it is not because it is discrete or finite, but because it inherits this intrinsic limit of the FT.

On the other hand, the peak tracking algorithm in SAMPLE (see Section 3.2) can follow the amplitude envelope of even the most extreme examples shown above.

4.3.2 All-Pole Modelling

If methods based on the DFT are the most common non-parametric approaches to spectral analysis, *all-pole models* are popular choices for parametric spectral analysis. They are based on the idea of modelling a discrete-time signal as the response of a discrete-time Linear Time-Invariant (LTI) system when the input is an impulse (for deterministic signals) or white noise (for stochastic processes) and computing an estimate of the Power Spectral Density (PSD) from the z-transform (see Section 2.2.3) of the system. Although discrete-time LTI systems have both poles and zeros, most approaches only estimate the poles and are called *autoregressive* (AR) or *all-pole* models [52].

One key issue of parametric modelling is the selection of the model *order*, i.e. the number of poles. In our case we know the nature of the signal under analysis and the correct order. A causal cosine can be expressed as the infinite impulse response of a discrete-time LTI with one zero and two poles.

$$x_i = u(i)\cos\left(\omega i/\nu_s + \phi\right) \tag{113}$$

$$X(z) = \mathcal{Z}\left\{x\right\}(z) = \frac{\cos\left(\phi\right) - \cos\left(\omega/\nu_s - \phi\right)z^{-1}}{\left(1 - e^{j\omega/\nu_s}z^{-1}\right)\left(1 - e^{-j\omega/\nu_s}z^{-1}\right)} \tag{114}$$

The poles correspond to the two (positive and negative) frequency components of the cosine function. A sum of two cosine functions will have

three zeros and four poles.

$$\mathcal{Z}\{x_1 + x_2\}(z) = \mathcal{Z}\{x_1\}(z) + \mathcal{Z}\{x_2\}(z) = \tag{115}$$
$$= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{\left(1 - e^{j\omega_1/\nu_s} z^{-1}\right)\left(1 - e^{-j\omega_1/\nu_s} z^{-1}\right)\left(1 - e^{j\omega_2/\nu_s} z^{-1}\right)\left(1 - e^{-j\omega_2/\nu_s} z^{-1}\right)}$$

So, an all-pole model for beating signals should be of order four: then, the two frequencies can be determined by the angle of the pole positions. Oppenheim and Schafer [64] give a more in-depth discussion of this case. The same considerations apply to exponentially-decaying sinusoidal signals, where we introduce the initial amplitude a and the decay time d.

$$x_i = u(i)\, a\, e^{-\lambda i/\nu_s} \cos(\omega i/\nu_s + \phi) \tag{116}$$

$$\mathcal{Z}\{x\}(z) = a \frac{\cos\phi - e^{-\lambda/\nu_s} \cos(\omega/\nu_s - \phi) z^{-1}}{\left(1 - e^{-\lambda/\nu_s + j\omega/\nu_s} z^{-1}\right)\left(1 - e^{-\lambda/\nu_s - j\omega/\nu_s} z^{-1}\right)} \tag{117}$$

The poles are still two and their angles are determined by the signal frequency, but they are not on the unit circle (their distance from the origin is $\exp\{-\lambda/\nu_s\}$). As before, an exponentially-decaying beating signal can be modelled as the response of a fourth-order model.

$$\mathcal{Z}\{x_1 + x_2\}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{\left(1 - e^{-\lambda_1/\nu_s + j\omega_1/\nu_s} z^{-1}\right)\left(1 - e^{-\lambda_1/\nu_s - j\omega_1/\nu_s} z^{-1}\right)} \cdot$$
$$\cdot \frac{1}{\left(1 - e^{-\lambda_2/\nu_s + j\omega_2/\nu_s} z^{-1}\right)\left(1 - e^{-\lambda_2/\nu_s - j\omega_2/\nu_s} z^{-1}\right)} \tag{118}$$

In the case where the number of effectively collected samples is low, all-pole models can be much more effective than the DFT for disentangling beating partials (see again Oppenheim and Schafer [64]). In the case of the problem analyzed in this chapter, the signal is recorded until it decays below the threshold of hearing, so that the limiting factor is the intrinsic duration of the signal determined by its starting amplitude and decay time (see Section 4.3.1).

We compared several parametric models [58] implemented in the Python package *Spectrum* [23]: Yule-Walker, Covariance and Modified Covariance. When the limiting factor is the signal duration and not the number of collected samples, all-pole models do not improve the frequency resolution of the PSD estimate with respect to the FT. Figure 7

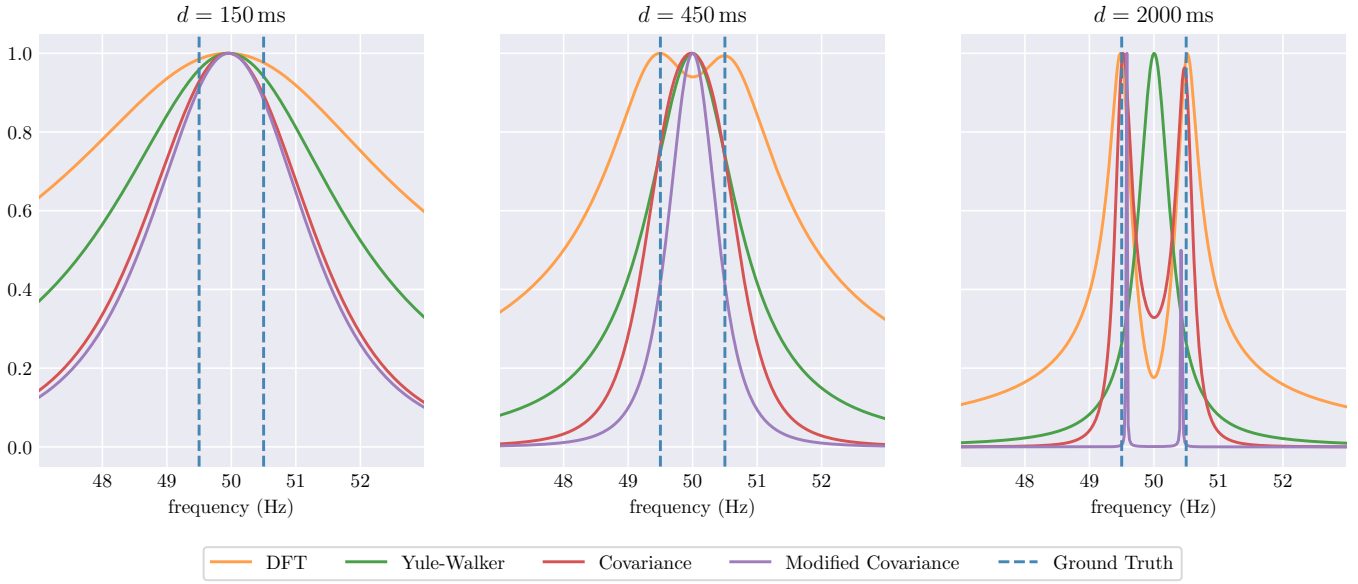

Figure 7: Comparison between Discrete Fourier-Transform (DFT) and parametric methods for Power Spectral Density (PSD) estimate on three example sounds.

shows three examples with three different decay times. Each example sound is composed of two modes with the same amplitude, decay, and phase, but two different frequencies, displayed with dashed lines (the average frequency is $\overline{\nu} = 50\,\text{Hz}$ and the difference is $2\hat{\nu} = 1\,\text{Hz}$). Each example has a different decay time (150, 450, and 2000 ms). Sampling frequency is 1 kHz and model order is 4. None of the methods could find the two different peaks for the shortest decay time. For the middle decay time, the DFT shows the two peaks but none of the parametric methods do. For the longest decay time, the DFT and covariance-based models could find the two peaks, but Yule-Walker (autocorrelation-based) could not.

4.3.3 Empirical Mode Decomposition

Empirical Mode Decomposition (EMD) is a nonlinear non-stationary signal analysis technique [111]. The goal of EMD algorithms is to obtain some components of a signal with well-behaved instantaneous amplitude and angular velocity, often to compute the Hilbert-Huang Transform or the Holospectrum [49]. Such features can be computed, respectively, as the absolute value and the derivative of the angle of an *analytic signal* (see Section 2.2.5).

Unfortunately, the Hilbert Transform of a real-world signal is usually not well-behaved and it may include a superposition of different oscillatory modes. EMD addresses the task of extracting components from a signal, called *Intrinsic Mode Functions* (IMFs), whose Hilbert Transform is well-behaved. An IMF is a signal that satisfies these two conditions [48]:

IMF.1 the number of local extrema (minima and maxima) and the number of zero-crossings (the number of signal sign changes) must differ at most by one;

IMF.2 the sum "of the envelope defined by the local maxima and the envelope defined by the local minima is zero" everywhere.

The core of EMD is the *sift* algorithm. It extracts IMFs iteratively: it computes the envelope of the local maxima and the envelope of the local minima, takes the average of the two and subtracts this average from the original signal until the average is close enough to zero. What

remains is an IMF and is subtracted out of the signal, which is analyzed again to find the next IMF, until the specified number of IMFs is found. The last component is not an IMF, but the residual signal after removing the IMFs. By construction, the sum of the IMFs reconstructs exactly the signal.

There are other variations on the original sift algorithm. The *ensemble* sifting algorithm (or EEMD) [108, 54] consists of sifting the signal multiple times, each time adding a different white noise signal, and then averaging the resulting IMFs to obtain the final output. For a sufficiently large ensemble size, the white noise should average out.

The *masked* sift [25] algorithm targets the problems that the original sift presents when dealing with intermitting signals. When extracting IMFs, the masked sift algorithm adds a *mask* signal, which is a sinusoid with a user-defined frequency. This frequency should be close to the frequency of the target IMF, so the frequency decreases for each successive IMF. The *iterated-masked* sifting algorithm (itEMD) [30] includes an automatic iterative mask-selection procedure.

Although the general concept of extracting IMFs from the signal seems promising for the aim of disentangling beats, EMD is insufficient for the task. EMD does not separate beating components because the sum of two beating sine-waves is actually an IMF, as it satisfies the two criteria above. This is not always true, but it is for the case of interest, acoustic beats. That case could be defined as the case in which the amplitude modulation modulant signal $\alpha(t)$ is much slower than the sinusoidal carrier wave $\cos(\phi(t))$. An exponentially-decaying beat signal $x(t)$ can be written using similar notation as in Theorem 2.

$$x(t) = a_1 e^{-2t/d_1} \cos(\omega_1 t + \phi_i) + a_2 e^{-2t/d_2} \cos(\omega_2 t + \phi_i) \tag{119}$$

Then, the fact that the amplitude modulation signal is much slower than the sinusoidal carrier can be expressed by assuming that the frequency average $\overline{\omega}$ is much bigger than other time constants.

$$\overline{\omega} \gg \max\left\{\hat{\omega}, \frac{2\pi}{d_1}, \frac{2\pi}{d_2}\right\} \tag{120}$$

Figure 8 shows an example and a counter-example. Figure 9 displays the output of different EMD methods on an example beat sound. We used the implementation in the Python package *EMD* [75]. The desired output would be a set of two IMFs with a constant frequency and

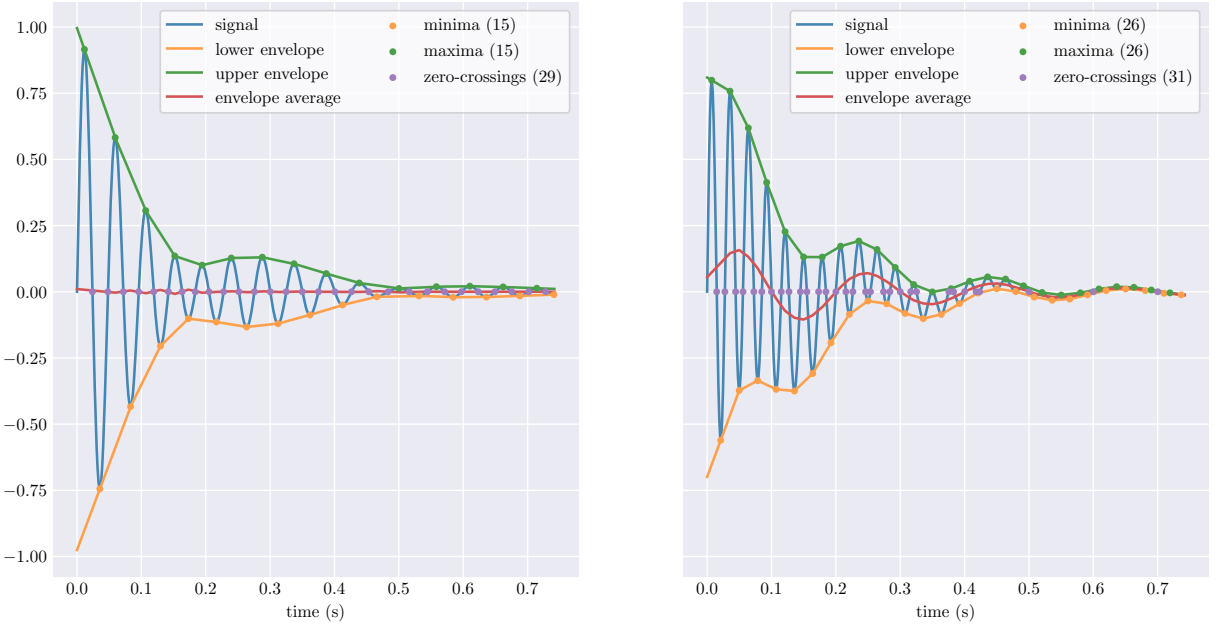

Figure 8: Two examples of sums of damped sinusoids. Frequency differences are $\hat{\nu} = 1.5\,\mathrm{Hz}$ ($\ll \overline{\nu}$) and $\hat{\nu} = 15\,\mathrm{Hz}$ ($\not\ll \overline{\nu}$). In the first example, the number of local extrema and the number of zero-crossings differ by only one (property IMF.1). Also, the envelope average is close enough to zero and we can consider property IMF.2 satisfied as well. The second example violates both conditions.

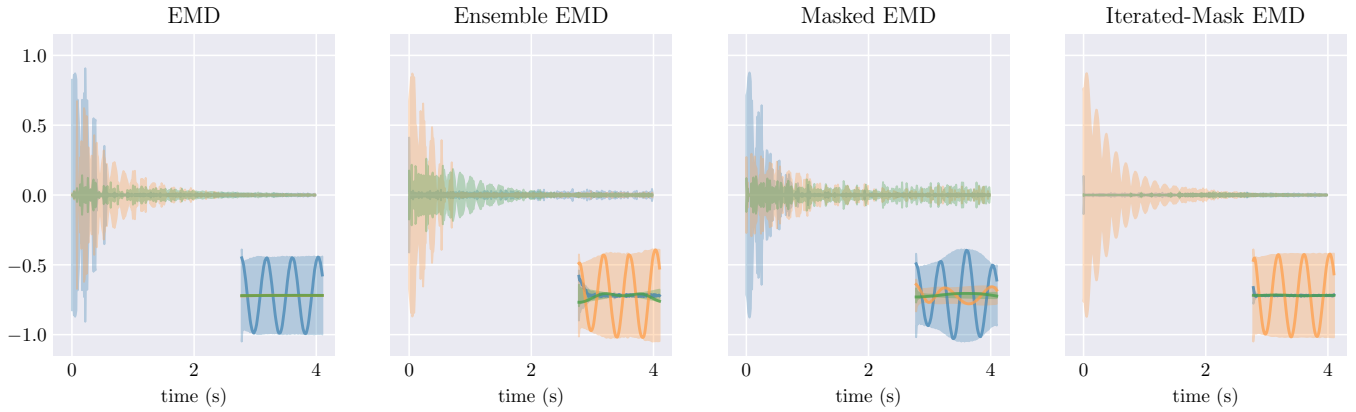

Figure 9: Comparison between Empirical Mode Decomposition (EMD), Ensemble EMD, Masked EMD, and Iterative-Mask EMD on a synthetic example. The input signal is the sum of two damped oscillations ($\nu_1 \approx 1103\,\text{Hz}$, $\nu_2 \approx 1097\,\text{Hz}$, $a_1 = 0.8$, $a_2 = 0.2$, $d_1 = 0.75\,\text{s}$, $d_2 = 2\,\text{s}$). The amplitude envelopes for all extracted modes are plotted as areas. Additionally, in the bottom-right corner of each plot shows the detail of the first 126 samples of each mode of oscillation: instantaneous values as full lines and envelopes as areas. Blue and orange are the first two IMFs, while green is the residual component.

exponential decay, but no approach could disentangle the two partials. The first three methods are not able to decompose the signal into any meaningful representation. On the other hand, itEMD assigns all of the signal to one IMF. Since, as observed, the sum of two beating partials is an IMF, this is the best result that can be expected from any EMD-based method.

4.4 Method

BeatsDROP (*Beats Duality for the Resolution Of Partials*) is an algorithm for estimating modal parameters. It fits the parameters of two interfering sinusoids to the time series of amplitude and frequency values, which we refer to as *trajectories*. The BeatsDROP algorithm does not perform the extraction of amplitude and frequency trajectories, which in turn can be achieved with Hilbert analysis or sinusoidal modelling techniques, such as SMS [87, 88, 89] or SAMPLE (see Chapter 3).

Estimating the parameters of two beating oscillations can help modal parameters estimates based on spectral analysis. When handling time-frequency representations, the audio analyst must always make a trade-off between time and frequency resolution, as implied by Heisenberg's uncertainty principle [45] in the form of the Gabor limit [36]. When using short windows for the Short-Time Fourier Transform analysis, if two sinusoids are not resolved because of the lower frequency resolution, BeatsDROP can be applied to estimate their parameters.

Empirical Mode Decomposition approaches could also benefit from the proposed algorithm. As we have shown in Section 4.3.3, the EMD sift algorithm can aggregate two beating partials in the same IMF, but BeatsDROP could be subsequently applied to such IMFs to separate the two tones.

BeatsDROP employs a nonlinear least-squares algorithm to fit the amplitude and frequency modulation expressions in Equation (59) and in Equation (63) to the amplitude and frequency trajectories. Two variants of the algorithm have been evaluated: a baseline, which only considers the amplitude modulation, and the proposed algorithm, which also exploits the frequency modulation in the beat. The evaluation shows that the frequency modulation pattern brings information that significantly improves the estimate.

4.4.1 Modal Model

A *partial* is a sinusoidal component of a signal. In modal synthesis, the amplitude of freely-oscillating partials is usually modelled through an exponential decay, which has two parameters: a *starting amplitude* (a) and a *decay time* (d). The partial also has a *modal frequency* (ν) and an *initial phase* (ϕ). A simple direct model for a modal partial is Equation (10).

$$x(t) = ae^{-2t/d} \cos(2\pi\nu t + \phi) \tag{121}$$

The problem addressed by this algorithm is to estimate the modal parameters of the two partials that are interfering in an input signal: a_1, a_2, d_1, d_2, ν_1, ν_2, ϕ_1, and ϕ_2. This is a special case of theorem 2, where the amplitude functions are exponentials.

$$A_1(t) = a_1\, e^{-2t/d_1} \qquad\qquad A_2(t) = a_2\, e^{-2t/d_2} \tag{122}$$

Since the amplitude envelopes are exponential, taking their derivative has the effect of just multiplying by a constant.

$$\frac{dA_i}{dt}(t) = a_i \frac{d\exp}{dt}\left(-\frac{2t}{d_i}\right) = -\frac{2a_i}{d_i} e^{-2t/d_i} = -\frac{2}{d_i} A_i(t) \tag{123}$$

The derivatives can be plugged into Equation (63) for the frequency modulation. In the following equations, the dependency on time is omitted for brevity.

$$\frac{d\hat{A}}{dt}\overline{A} - \frac{d\overline{A}}{dt}\hat{A} = \tag{124}$$

$$= \frac{1}{4}\left[\frac{dA_1 - dA_2}{dt}(A_1 + A_2) - \frac{dA_1 + dA_2}{dt}(A_1 - A_2)\right] = \tag{125}$$

$$= \frac{1}{2}\left(\frac{dA_1}{dt}A_2 - \frac{dA_2}{dt}A_1\right) = \frac{d_1 - d_2}{d_1 d_2} A_1 A_2 \tag{126}$$

So, the frequency modulation signal is

$$\omega(t) = \overline{\omega} + \frac{1}{\alpha^2(t)}\Bigg[\overline{A}(t)\hat{A}(t)\hat{\omega} + \frac{(d_1 - d_2)\sin\left(2\hat{\omega}t + 2\hat{\phi}\right)}{2d_1 d_2} A_1(t) A_2(t)\Bigg] \tag{127}$$

4.4.2 Sinusoidal Analysis

The frequency and amplitude trajectories are extracted with SAMPLE (see Chapter 3). They are the time series of estimated amplitude and frequency values. The amplitude and frequency at time $t(i)$ are a_i and v_i, respectively, where i is the frame index.

SAMPLE estimates the trajectories from the Short-Time Fourier Transform (STFT) of the signal, which is the series of the DFT of successive segments of the signal (see Section 2.2.4). So, the temporal resolution of the trajectories is determined by the segment size, while their sampling frequency is determined by the hop size, i.e. the number of samples in between successive frames. The frequency resolution of the STFT is limited by the segment size (see Section 4.3.1), but the estimation of the peak values is refined with parabolic interpolation.

4.4.3 Parameter Initialization

BeatsDROP initializes amplitude and decay values to be the same for the two partials, using a linear or semi-linear fit over the amplitude values in decibel, as it is done in SAMPLE. For the experiments, linear least-squares estimation [40] was used.

The average frequency of the two partials ($\overline{v}$) is initialized as the average frequency of the corresponding trajectory. The frequency difference ($2\hat{v}$) is initialized analyzing the periodicity in the amplitude trajectory. From Theorem 2, the frequency of the amplitude modulation is approximately $2\hat{v}$, because taking the square of a sinusoidal function doubles its frequency. BeatsDROP estimates the period $1/2\hat{v}$ of the amplitude trajectory as the first peak of its smoothed autocorrelation vector, following the example of classical pitch detection algorithms [76].

The input trajectories may have missing samples, because of how the SAMPLE peak continuation algorithm attaches new trajectory segments to previous ones (see Section 3.2.1). The Lomb-Scargle periodogram [55, 84] can estimate the Power Spectral Density (PSD) from unevenly sampled data. The PSD is estimated only for low frequencies (in the experiments, below 12 Hz), setting the PSD for higher frequencies to zero. This has the effect of a low-pass filter and the autocorrelation

vector will be smoother. The smoothed autocorrelation vector is computed as the Inverse Discrete Fourier Transform of the PSD vector (see Equation (38) in Section 2.2.6).

Finally, BeatsDROP initializes the phase difference as the phase value of the DFT of the amplitude trajectory at the amplitude modulation frequency ($2\hat{\nu}$). The phase of the first partial is assumed to be zero. This simplifies the initialization and has no effects on the quality of the fit, because both the amplitude and the frequency trajectories only depend on the phase difference, and not on the absolute values.

4.4.4 Optimization

Two residual functions are defined for the least-squares optimizer (Section 2.3): one over amplitude values and one over frequency values. The residual function for amplitude is the difference between the trajectory amplitude value a_i and the amplitude modulation predictions α, computed as defined in equation (59), at the corresponding time instant $t(i)$. The baseline algorithm minimizes these residuals only.

$$\Delta_i^{(a)} := a_i - 2\alpha(t(i)) \tag{128}$$

The proposed algorithm also includes a residual function over frequency values, computed as in equation (127). Frequency differences are computed on the Mel scale, which is a scale related to the human perception of pitch [96].

$$\Delta_i^{(\nu)} := \text{hz2mel}(\nu_i) - \text{hz2mel}\left(\frac{\omega(t(i))}{2\pi}\right) \tag{129}$$

$$\text{hz2mel}(\nu) := 2595 \log_{10}\left(1 + \frac{\nu}{700}\right) \tag{130}$$

The residuals have two very different ranges. Amplitude values range between 0 and 1. Frequency values range between 0 and 3800 mel (between 0 and 20 kHz). A weighting parameter w multiplies the frequency residuals to manage the relative importance of the two residual vectors. By default, it is set to $w = 1/3800$, so that the maximum residual for both amplitude and frequency is 1. In summary, the least-squares problem is:

$$\arg\min \sum_{i=1}^{n} \rho\left(\left[\Delta_i^{(a)}\right]^2\right) + \rho\left(\left[w\,\Delta_i^{(\nu)}\right]^2\right) \tag{131}$$

The function ρ is the loss function. The experiments were performed using the implementation in *SciPy* [116], with the Rectangular Trust Region Dogleg Approach (DogBox) [117] and the Cauchy loss function [110] $\rho(z) = \ln(1 + z)$.

4.4.5 Decision Criterion

Not all trajectories belong to beats. When this is the case, the best fit for BeatsDROP is to either set both frequencies to the same value or to set the amplitude of one of the partials to zero. Given a sensitivity threshold ϵ, a decision function can be defined to discriminate between beating trajectories and non-beating trajectories.

$$D_{\text{beat}}(a_1, a_2, \nu_1, \nu_2) = |\nu_1 - \nu_2| > \epsilon \quad \wedge \quad a_1 > \epsilon \quad \wedge \quad a_2 > \epsilon \tag{132}$$

Although simple, it is also effective, as the experiments show (see Section 4.5.2).

4.5 Evaluation

The performance of both BeatsDROP and of the baseline algorithm was evaluated on a synthetic dataset of 1024 pairs of interfering modal partials (experimental data is available on Zenodo [100]). Figure 10 displays an example of applying the two algorithms to a synthetic audio example. In this instance, the baseline could not disentangle the two parameter sets and the a_1 and a_2 functions are superimposed. On the other hand, BeatsDROP successfully estimates different amplitudes and decay times for the two partials. As we expected, since BeatsDROP fits the parameters to the frequency trajectory as well, the estimate of the modal frequencies is more accurate.

4.5.1 Synthetic Dataset

Each of the 1024 pairs of interfering partials was saved to an audio file of 3 seconds, at a 44.1 kHz sampling rate. White Gaussian noise, with a signal-to-noise ratio of 45 dB, simulates background noise.

For each pair, the first frequency is chosen randomly between 200 Hz and 2 kHz. The distribution of random values is a Beta distribution with parameters $\alpha = \beta = 2$ over the Bark scale. The Beta distribution

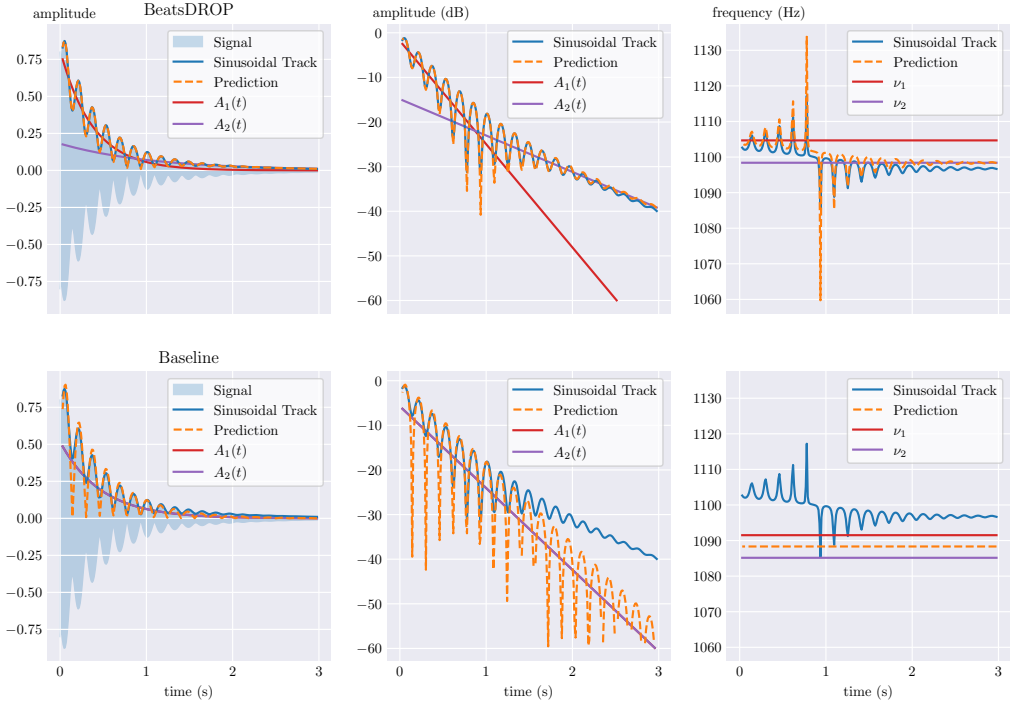

Figure 10: Example of running BeatsDROP on synthetic data. The input is the sum of two partials: $\nu_1 \approx 1103\,\text{Hz}$, $\nu_2 \approx 1097\,\text{Hz}$, $a_1 = 0.8$, $a_2 = 0.2$, $d_1 = 0.75\,\text{s}$, $d_2 = 2\,\text{s}$.

is used, as opposed to a uniform distribution, in order to reduce the probability of extreme values. The Bark scale is a scale for frequencies related to critical bands and the perception of beats [109].

The difference between the first and the second frequency is chosen randomly between 1.8 Hz and 18 Hz. By doing so, it will be big enough to produce at least five cycles of the modulation functions, but that it will be small enough not to be resolved as two different partials in the Short-Time Fourier Transform (the frequency resolution for 2048 samples per window at 44.1 kHz is 21 Hz). Since the frequency difference from the amplitude modulation is perceived as a beat in time, and not as a pitch, its reciprocal, the modulation period, is sampled from a Beta distribution.

Starting amplitude values are sampled uniformly in the range between -10 dB and 0 dB, then they are converted to linear amplitude values and normalized for each pair in order to avoid clipping when saving the synthetic data to an audio file.

Decay times have an exponential distribution with minimum value 0.5 s and expected value 1 s. With these constraints, the sounds will persist for at least three cycles of the amplitude modulation: in the worst case, they will take 1.73 s to decay by 60 dB, which is 3.11 periods at the minimum frequency difference of 1.8 Hz.

Finally, initial phase values are generated uniformly in the range between 0 and 2π. The prediction errors on these parameters will not be assesses, as estimating the initial phase values is outside of the aim of the proposed algorithm, since they are not perceptually relevant.

4.5.2 Results

The prediction errors for the frequency, amplitude, and decay parameters of the two interfering partials for both the baseline and the proposed algorithm are tested.

Errors on the modal frequencies are computed in mel [96] and the errors on the starting amplitudes in decibel. These units are related to the human perception of these quantities more than hertz and linear amplitude values.

The Bayesian signed-rank test [7] implementation in *Baycomp* [8] was used through *Autorank* [46] to determine the significance of differences between the absolute error values of the two algorithms. The

	BeatsDROP		Baseline	
	Med.	95% CI	Med.	95% CI
v_1 (mel)	0.536	[0.380, 0.798]	2.834	[2.321, 3.389]
v_2 (mel)	0.300	[0.226, 0.398]	1.949	[1.623, 2.286]
a_1 (dB)	0.913	[0.818, 1.003]	1.434	[1.159, 1.712]
a_2 (dB)	0.460	[0.395, 0.543]	2.075	[1.805, 2.330]
d_1 (s)	0.008	[0.005, 0.011]	0.257	[0.205, 0.316]
d_2 (s)	0.007	[0.005, 0.011]	0.109	[0.089, 0.131]

Table 2: Summary of test results: medians and the 95% confidence intervals of the absolute errors for each of the two variants and for each of the six parameters. BeatsDROP achieves smaller median errors than the baseline and the differences are significant and practically relevant for all parameters.

family-wise significance level of the tests is $\alpha = 0.05$ and the width of the region of practical equivalence (ROPE) was dynamically determined depending on the effect size. The ROPE width is set to 0.1γ, where γ is the pooled median absolute deviation (Akinshin's γ). The medians and the 95% confidence intervals for the errors on all six parameters are summarized in Table 2.

Discussion

When comparing the two algorithms, all median errors of BeatsDROP are lower than the baseline's. Moreover, for all six parameters the differences are significant and practically relevant. This can also be easily seen in Table 2 from the fact that, for all variables, the two 95% confidence intervals do not overlap.

The difference is very promising in general. BeatsDROP better fits not only the frequency trajectory, which could be expected, but also the tail of the amplitude envelope. This results in a one-order-of-magnitude decrease of the median errors on the decay times, from above 100 ms to below 10 ms.

Additional Results

One might think that increasing the FFT size could help in resolving beats in the STFT, without using the BeatsDROP algorithm at all. But running SAMPLE with an increased FFT size gives very poor results: two tracks were detected in 8 out of the 1024 examples in the synthetic dataset (0.78%) using an FFT size of 2^{14} samples (0.37 seconds at 44.1 kHz), and only in 6 examples (0.59%) using an FFT size of 2^{16} samples (1.49 s). This is the effect of the resolution limit in the FT of exponentially-decaying partials discussed in Section 4.3.1.

Another dataset, similar to the first, was generated to test the decision criterion. In every example there were not only the two beating partials, but also a third non-beating partial. The frequency of the third partial is randomized at a distance between 1.5 and 4 Barks from the average frequency of the beating partials, following a Beta distribution with parameters $\alpha = 2$ and $\beta = 4$, which favours lower values. The probability of being higher or lower than the beat average frequency is the same. The classification accuracy of the simple decision rule described in Section 4.4.5 was found to be 88% on this dataset. Most false negatives, i.e. unrecognized beats, happen when one of the two beating partials has a much higher decay time or amplitude than the other. Most false positives happen for short decay times of the third partial.

4.6 Conclusion

A set of theoretical results about beats was presented. The most important is the expression of the sum of two partials with different amplitudes and frequencies as one partial, which is modulated both in amplitude and in frequency, and the special case of partials with exponential decay. This formulation is of practical interest in the context of sinusoidal analysis, as it allows separating two partials that were not resolved within a STFT. This, in turn, can compensate for the loss of frequency resolution when using smaller windows for a higher time resolution in the STFT. It was shown that pre-existing approaches for spectral analysis, namely the Fourier Transform, all-pole models, and Empirical Mode Decomposition, are insufficient for resolving beats.

BeatsDROP was proposed, a least-squares regression algorithm for

the estimate of modal parameters, that mainly builds on top of previous algorithms for spectral analysis of modal sounds, but could also be applied in the context of Empirical Mode Decomposition and Hilbert analysis. Its performance was evaluated against a baseline on a synthetic dataset. The baseline algorithm only accounts for the well-known amplitude modulation in beats, but not for the frequency modulation. Accounting for the frequency modulation results in a more accurate estimate of all modal parameters on the synthetic dataset. Also, the parameters extracted with the BeatsDROP algorithm can be used to determine if the analyzed data belong to a beating sound or to a single partial with a simple decision function.

The implementations of both the algorithms and of the evaluation protocols are included in the open source *SAMPLE* Python package (see Chapter 6).

In the future, more sophisticated decision functions should be developed for determining whether a trajectory should be decomposed in two partials or not. Also a subjective evaluation should be performed on the resynthesis quality over real sounds (as in Section 3.3.2).

Chapter 5

Generalized Mixture Space

Abstract

The Generalized Mixture Space (GMS) is presented as a novel model for spectral analysis of multi-channel signals. Traditional methods often analyze each channel independently, neglecting the relationships between them. GMS overcomes this limitation by representing the spectrum as a continuous interpolation space. This allows a compact representation in terms of spectral content (magnitude and phase) and inter-channel relationships (correlation and energy distribution).

GMS is meant as a generalization of the Bivariate Mixture Space (BMS) for bivariate signals. It extends the BMS concept to an arbitrary number of channels while preserving its core functionalities. This chapter formally defines the GMS and demonstrates where it is equivalent or compatible to the BMS for bivariate signals. After defining the Principal Spectrum through a maximization problem, an eigendecomposition of the channel cross-correlation matrix is demonstrated to provide a solution to that problem. This eigenvector formulation is similar to performing Principal Component Analysis across channels for each frequency. Finally, some potential applications are discussed,

This chapter is based on [107] Marco Tiraboschi and Giorgio Presti. 'The Generalized Mixture Space: extending Compact Spectral Representations to Multi-channel Signals'. In: *Journal of the Audio Engineering Society* (2024). [TO BE SUBMITTED]

including spectrogram visualization and modal audio analysis, highlighting areas for future investigation.

5.1 Introduction

Spectral analysis models are often defined on single-channel signals, especially for audio. However, when dealing with multi-channel signals, analyzing them independently might not be the best approach, since this would not highlight the relationships between them. Furthermore, representing the spectrum of a multi-channel signal as multiple spectral can be extremely redundant, as common components are repeated in many channels, and difficult to display visually, e.g. producing spectrograms.

Many approaches have been developed for couples of signals, like stereophonic audio. Some approaches treat these *bivariate* signals as complex valued signals [98], onto which the standard Fourier Transform is defined. There are also specialized variants of the FT specifically designed for bivariate signals [34]. Hamon and Hannan [43], instead, analyze bivariate signals from their cross-channel delays, while Briand *et al.* [17] use Principal Component Analysis [66] (PCA) in the time-domain.

The Bivariate Mixture Space [69] (BMS), instead, represents bivariate spectra as a continuous interpolation space on which the two signals lie. This model allows for a compact representation not only of a principal spectral content of the bivariate signal (its magnitude and phase), but also of its relational information (correlation and channel distribution). This representation can be used for many applications, like source separation, feature engineering, and compact visualization.

In this chapter, the Generalized Mixture Space (GMS) is proposed. This model extends the concept of the BMS to signals with an arbitrary number of channels. It preserves the core properties of representing the principal spectral content of the signal as a single spectrum and also modelling information about the channel correlation and energy distribution.

The GMS analysis algorithm is based on eigen-decomposition over the channel cross-correlation matrix, and it could be associated with performing PCA over the channels for each frequency. Also, the GMS

can be shown to be equivalent to the BMS for bivariate inputs.

5.2 Multivariate model

In this section, the direct model that is the conceptual core of the Generalized Mixture Space is presented. It directly extends the mixing model defined in the context of the Bivariate Mixture Space [69] for multiple channels.

5.2.1 Multivariate Mixture

A multivariate signal can be equivalently defined either as a vector of c signals or as a signal with c values.

$$\vec{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_c(t) \end{bmatrix} \in \mathbb{R}^c \tag{133}$$

Each of these channels could be the linear combination (a *mixture*) of a number s of latent source signals $z_j(t)$.

$$x_i(t) = \sum_{j=1}^{s} a_{i,j}\, z_j(t) \qquad\qquad \vec{x}(t) = A\vec{z}(t) \tag{134}$$

Matrix $A \in \mathbb{R}^{c \times s}$ contains the mixing coefficients for all channels and sources. Without loss of generality, the sum of the squares of the mixing coefficients for each source (every column of the matrix) can be constrained to be 1.

$$\left\| a_{\bullet,j} \right\|_2^2 = \sum_{i=1}^{c} a_{i,j}^2 = 1 \quad \forall j \in \{1, \ldots, s\} \tag{135}$$

$$a_{\bullet,j} = \begin{bmatrix} a_{1,j} \\ a_{2,j} \\ \vdots \\ a_{c,j} \end{bmatrix} \tag{136}$$

If a column is zero, then the corresponding source is discarded, as it is not in the mixture. Otherwise, any column of coefficients can be normalized by applying an appropriate gain to the latent source, obtaining the same mixture.

$$A\vec{z}(t) = \begin{bmatrix} \frac{a_{1,1}}{\|a_{\bullet,1}\|_2} & \frac{a_{1,2}}{\|a_{\bullet,2}\|_2} & \cdots & \frac{a_{1,s}}{\|a_{\bullet,s}\|_2} \\ \frac{a_{2,1}}{\|a_{\bullet,1}\|_2} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{a_{c,1}}{\|a_{\bullet,1}\|_2} & \cdots & \cdots & \frac{a_{c,s}}{\|a_{\bullet,s}\|_2} \end{bmatrix} \begin{bmatrix} \|a_{\bullet,1}\|_2 \, z_1(t) \\ \|a_{\bullet,2}\|_2 \, z_2(t) \\ \vdots \\ \|a_{\bullet,s}\|_2 \, z_s(t) \end{bmatrix} \tag{137}$$

A vector in $\mathbb{R}^c$ can be expressed either with c Cartesian coordinates or with one radius and $c - 1$ angles, in hyperspherical coordinates (more on this in Section 5.2.3). Since the columns of A have unitary L^2-norm, they are all vectors that lie on the c-dimensional unitary-radius hypersphere. So, they only need the $c - 1$ angular values to be defined. In the special case of bivariate signals, this falls back to the cosine panlaw [28], which only needs one angular parameter to "move" the source between the two channels, like in the BMS.

$$c = 2 \quad \Rightarrow \quad A = \begin{bmatrix} \cos(\sigma_1) & \cos(\sigma_2) & \dots & \cos(\sigma_s) \\ \sin(\sigma_1) & \sin(\sigma_2) & \dots & \sin(\sigma_s) \end{bmatrix} \tag{138}$$

This formulation also holds for digital signals. In this case, the latent sources will be s rows of n samples $Z \in \mathbb{R}^{s \times n}$, and the multivariate signal will be the matrix $X \in \mathbb{R}^{c \times n}$.

$$x_{i,k} = \sum_{j=1}^{s} a_{i,j} \, z_{j,k} \qquad\qquad X = AZ \tag{139}$$

5.2.2 Mixture of Spectra

The same model can be applied to the Fourier Transform of mixtures.

$$\mathcal{F}\{x_i\}(\nu) = \sum_{j=1}^{s} \mathcal{F}\{a_{i,j} \, z_j\}(\nu) = \sum_{j=1}^{s} a_{i,j} \mathcal{F}\{z_j\}(\nu) \tag{140}$$

$$\mathcal{F}\{\vec{x}\}(\nu) := \begin{bmatrix} \mathcal{F}\{x_1\}(\nu) \\ \mathcal{F}\{x_2\}(\nu) \\ \vdots \\ \mathcal{F}\{x_c\}(\nu) \end{bmatrix} = A \begin{bmatrix} \mathcal{F}\{z_1\}(\nu) \\ \mathcal{F}\{z_2\}(\nu) \\ \vdots \\ \mathcal{F}\{z_s\}(\nu) \end{bmatrix} = A\mathcal{F}\{\vec{z}\}(\nu) \tag{141}$$

And it is true, also, for the Discrete Fourier Transform ($W \in \mathbb{C}^{n\times n}$ is the DFT matrix defined in Equation (16), and it is symmetric). Note that a channel is a row $x_{i,\bullet}$ in matrix X and a source is a row $z_{j,\bullet}$ in matrix Z. Let $\xi_{i,\bullet}$ be the DFT of channel $x_{i,\bullet}$ and let $\Xi \in \mathbb{C}^{c\times n}$ be the matrix of the DFT's of all channels, which has rows $\xi_{i,\bullet}$.

$$\xi_{i,\bullet} = \left(W\left(x_{i,\bullet}\right)^T\right)^T = x_{i,\bullet}W^T = x_{i,\bullet}W \tag{142}$$

$$\Xi = \begin{bmatrix} \xi_{1,1} & \xi_{1,2} & \cdots & \xi_{1,n} \\ \xi_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \xi_{c,1} & \cdots & \cdots & \xi_{c,n} \end{bmatrix} = XW \tag{143}$$

By associativity, the rows of Ξ can be expressed as the linear combinations of the DFTs of the latent sources, with the same coefficients A.

$$\Xi = XW = (AZ)\,W = A\,(ZW) \tag{144}$$

5.2.3 Coordinate Systems

As it was anticipated in Section 5.2.1, vectors in $\mathbb{R}^c$ can be expressed either with c Cartesian coordinates or in hyperspherical coordinates with one radius and c - 1 angles. Given a radius ρ and c - 1 angles α_j, the vector of cartesian coordinates $\vec{y}$ can be computed iteratively.

$$\vec{y} = \mathrm{h2c}\,(\rho, \vec{\alpha}) := \left[\rho\left(\prod_{j=1}^{i-1}\sin\alpha_j\right)\cdot\begin{cases}\cos\alpha_i & i < c \\ 1 & i = c\end{cases}\right]_{i=1}^{c} \tag{145}$$

The radius ρ is the L^2-norm of $\vec{y}$.

$$\|\vec{y}\|_2^2 = \sum_{i=1}^{c}\left[\rho\left(\prod_{j=1}^{i-1}\sin\alpha_j\right)\cdot\begin{cases}\cos\alpha_i & i < c \\ 1 & i = c\end{cases}\right]^2 = \tag{146}$$

$$= \rho^2\sum_{i=1}^{c}\left(\prod_{j=1}^{i-1}\sin^2\alpha_j\right)\cdot\begin{cases}\cos^2\alpha_i & i < c \\ 1 & i = c\end{cases} = \tag{147}$$

$$= \rho^2 \left[\left(\prod_{j=1}^{c-2} \sin^2 \alpha_j \right) \left(\cos^2 \alpha_{c-1} + \sin^2 \alpha_{c-1} \right) + \right.$$

$$\left. + \sum_{i=1}^{c-2} \left(\prod_{j=1}^{i-1} \sin^2 \alpha_j \right) \cos^2 \alpha_i \right] = \quad (148)$$

$$= \rho^2 \sum_{i=1}^{c-1} \left(\prod_{j=1}^{i-1} \sin^2 \alpha_j \right) \cdot \begin{cases} \cos^2 \alpha_i & i < c - 1 \\ 1 & i = c - 1 \end{cases} \quad (149)$$

The sum can be simplified telescopically.

$$\|\vec{y}\|_2^2 = \rho^2 \sum_{i=1}^{1} \left(\prod_{j=1}^{i-1} \sin^2 \alpha_j \right) \cdot \begin{cases} \cos^2 \alpha_i & i < 0 \\ 1 & i = 1 \end{cases} = \rho^2 \quad (150)$$

The angles can be inverted to be α_j. In the ambiguous cases in which $y_k = 0$ for all $k \geq j$, let α_j be $\pi/2$.

$$\alpha_j = \begin{cases} \operatorname{arctan2}\left(\sqrt{\sum_{k=j+1}^{c} y_k^2}, y_j \right) & j < c - 1 \\ \operatorname{arctan2}\left(y_c, y_{c-1} \right) & j = c - 1 \end{cases} \quad (151)$$

5.2.4 Generalized Mixture Space

Similarly to the definition of the Bivariate Mixture Space, the individual discrete channels can be interpreted as samples of an underlying continuous signal. Formally, let $\vec{\delta}^{(c,i)}$ be the c-dimensional Kronecker delta versor, which is equal to the first c entries of the i-th row or column of an identity matrix.

$$\vec{\delta}^{(c,i)} = \left[\mathbb{I}_{i,j} \right]_{j=1}^{c} \quad (152)$$

Each channel (either in the time or in the frequency domain) can be sampled from the mixture by a product with a Kronecker delta, i.e. its corresponding row in the identity matrix.

$$x_i(t) = \left(\mathbb{I}\vec{x}(t) \right)_i = \vec{\delta}^{(c,i)^T} \vec{x}(t) \quad (153)$$

$$\mathcal{F}\{x_i\}(\nu) = \mathcal{F}\{(\mathbb{I}\vec{x})\}_i(\nu) = \vec{\delta}^{(c,i)^T} \mathcal{F}\{\vec{x}\}(\nu) \quad (154)$$

The L^2-norm of these coefficient vectors is 1, so they can be represented in hyperspherical coordinates with $c - 1$ angles. The angles for channel

i will all be $\pi/2$ except for angle α_i, which will be zero (all angles will be $\pi/2$ for the last channel).

$$\vec{\delta}^{(c,i)} = \begin{cases} \text{h2c}\left(1, \pi/2 \cdot \left(\vec{\delta}^{(c-1,i)} - \vec{1}\right)\right) & i < c \\ \text{h2c}\left(1, \pi/2 \cdot \vec{\delta}^{(c-1,i)}\right) & i = c \end{cases} \tag{155}$$

The Generalized Mixture Space is a function of one frequency value and c - 1 angles that interpolates the original channels' spectra.

$$\text{GMS}\left\{\vec{x}\right\}(\nu, \vec{\alpha}) := \text{h2c}\,(1, \vec{\alpha})^T \mathcal{F}\{x\}(\nu) \tag{156}$$

This formulation does not depend on any assumptions about $\vec{x}$ being a multivariate mixture, but the next section will show some properties of the GMS for such signals.

5.3 Principal Spectral Analysis

The *Principal Spectrum* (PS) of a multivariate signal is the linear combination across channels with maximal square magnitude with unitary L^2-norm coefficients, for each frequency. The PS can be equivalently defined as the GMS value with maximum magnitude for each frequency.

$$\text{PS}\left\{\vec{x}\right\}(\nu) := \text{GMS}\left\{\vec{x}\right\}(\nu, \vec{\sigma}(\nu)) \tag{157}$$

$$\vec{\sigma}(\nu) := \underset{\vec{\alpha} \in [0, 2\pi]^{c-1}}{\arg\max} \left\|\text{GMS}\left\{\vec{x}\right\}(\nu, \vec{\alpha})\right\|_2 \tag{158}$$

$$\vec{h}(\nu) := \text{h2c}\,(1, \vec{\sigma}(\nu)) = \underset{\vec{a} \in \mathbb{R}^c \,:\, \|\vec{a}\|_2 = 1}{\arg\max} \left\|\vec{a}^T \mathcal{F}\{x\}(\nu)\right\|_2 \tag{159}$$

5.3.1 Cross-correlation Eigendecomposition

The maximization formulation in Equation (159) admits a closed-form solution. The Principal Spectrum can be found by decomposing the channel cross-correlation matrix $\mathcal{R}_{\vec{x}}(\nu)$ into its eigenvectors. The coefficient vector for the Principal Spectrum is the unitary L^2-norm eigenvector $\vec{h}(\nu)$ with maximal eigenvalue λ of such matrix. Since the cross-correlation matrix is real and symmetric, the eigenvector can be chosen to be real and orthonormal [97].

Theorem 3. *The closed-form solution for the optimal coefficient array at one frequency ν is the eigenvector with maximal eigenvalue of the sum of the cross-correlation matrix of the real and imaginary parts of the channels $\mathcal{F}\{\vec{x}\}(\nu)$.*

$$\vec{h}(\nu) = \underset{\vec{a}\in\mathbb{R}^c \,:\, \|\vec{a}\|_2=1}{\arg\max} \left\| \vec{a}^T \mathcal{F}\{\vec{x}\}(\nu) \right\|_2 = \underset{\substack{\vec{a}\in\mathbb{R}^c \,:\, \|\vec{a}\|_2=1, \\ \mathcal{R}_{\vec{x}}(\nu)\vec{a}=\lambda\vec{a}}}{\arg\max} \lambda$$

$$\mathcal{R}_{\vec{x}}(\nu) := \Re\vec{X}(\nu) \left(\Re\vec{X}(\nu)\right)^T + \Im\vec{X}(\nu) \left(\Im\vec{X}(\nu)\right)^T$$

$$\vec{X}(\nu) := \mathcal{F}\{\vec{x}\}(\nu)$$

Proof. The square magnitude of a real linear combination of the channel spectra can be expressed with a quadratic form involving the cross-correlation matrix $\mathcal{R}_{\vec{x}}(\nu)$.

$$\left|\vec{a}^T\vec{X}(\nu)\right|^2 = \Re^2\left(\vec{a}^T\vec{X}(\nu)\right) + \Im^2\left(\vec{a}^T\vec{X}(\nu)\right) = \tag{160}$$

$$= \left(\vec{a}^T\Re\vec{X}(\nu)\right)^2 + \left(\vec{a}^T\Im\vec{X}(\nu)\right)^2 = \tag{161}$$

$$= \vec{a}^T\Re\vec{X}(\nu)\left(\Re\vec{X}(\nu)\right)^T\vec{a} \quad + \tag{162}$$

$$+ \vec{a}^T\,\Im\vec{X}(\nu)\,\left(\Im\vec{X}(\nu)\right)^T\vec{a} \quad = \tag{163}$$

$$= \vec{a}^T\mathcal{R}_{\vec{x}}(\nu)\vec{a} \tag{164}$$

This auto-correlation matrix sum is a real symmetric matrix ($\mathcal{R}_{\vec{x}}(\nu) = \mathcal{R}_{\vec{x}}^T(\nu)$). Any real symmetric matrix can be decomposed as the product $U\Lambda U^T$, where U is the ortho-normal matrix of its eigenvectors and Λ is the diagonal matrix of its eigenvalues, in decreasing order of absolute value. For an ortho-normal matrix, the inverse is the transpose: $UU^T = U^TU = \mathbb{I}$, and both the set of the rows and the set of the columns are a ortho-normal basis for the vector space. The eigendecomposition $\mathcal{R}_{\vec{x}}(\nu) = U\Lambda U^T$ can be plugged in the expression for the magnitude.

$$\left|\vec{a}^T\vec{X}(\nu)\right|^2 = \vec{a}^TU\Lambda U^T\vec{a} \tag{165}$$

Let $\vec{\gamma} := U^T\,\vec{a}$ be the diagonalized projection of $\vec{a}$.

$$\left|\vec{a}^T\vec{X}(\nu)\right|^2 = \vec{\gamma}^T\Lambda\vec{\gamma} = \sum_{i=1}^{c}\overline{\gamma_i}\lambda_i\gamma_i = \sum_{i=1}^{c}|\gamma_i|^2\,\lambda_i \tag{166}$$

Since U is a ortho-normal matrix, the vector $\vec{\gamma}$ also has the same unitary L^2-norm as $\vec{a}$.

$$\|\vec{\gamma}\|_2^2 = \vec{\gamma}^T\vec{\gamma} = \vec{a}^T U U^T \vec{a} = \vec{a}^T\vec{a} = \|\vec{a}\|_2^2 = 1 \tag{167}$$

It follows that $\vec{\gamma}^{(2)} := \left[\gamma_j^2\right]_{j=1}^{C}$ can be seen as the coefficient vector of a convex combination (all values are non-negative and they sum up to one). So, the square magnitude of a linear combination with unitary L^2-norm coefficients can only be a convex combination of the eigenvalues.

$$\left|\vec{a}^T\vec{X}(\nu)\right|^2 = \sum_{j=1}^{C} \left|\gamma_j\right|^2 \lambda_j = \vec{\gamma}^{(2)\,T}\vec{\lambda} \tag{168}$$

The maximum convex combination occurs if we set to 1 the coefficient of the maximum eigenvalue and to 0 all the others. So, the optimal $\vec{\gamma}$ is the Kronecker delta vector with the non-null entry corresponding to the maximal eigenvalue (if the eigenvalues are ordered in descending order, then it will be the first entry). This means that the optimal coefficient vector is the corresponding eigenvector.

$$\vec{\gamma}^* = \vec{\delta}^{(c,1)} = \begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}^T \in \mathbb{R}^c \tag{169}$$

$$\vec{h}(\nu) = U\vec{\gamma}^* = U_{\bullet,1} \tag{170}$$

Quod erat demonstrandum. □

5.3.2 Mixture of non-overlapping Sources

A new condition is now added to the latent source mixture model described in Section 5.2.1. In this restricted model, latent sources are not allowed to overlap in the frequency domain, i.e., only one channel's FT can be non-zero for each frequency value.

$$\exists j, \nu \;:\; \mathcal{F}\left\{z_j\right\}(\nu) \neq 0 \Rightarrow \mathcal{F}\{z_i\}(\nu) = 0 \quad \forall i \neq j \tag{171}$$

Equivalently, the product of the FTs of any pair of different channels must be zero.

$$\mathcal{F}\{z_i\}(\nu)\mathcal{F}\left\{z_j\right\}(\nu) = 0 \quad \forall \nu\, \forall i\, \forall j \neq i \tag{172}$$

If this condition holds, then the PS is the sum of the spectra, or, equivalently the spectrum of the non-zero source for each frequency.

Theorem 4. *The Principal Spectrum of a multivariate mixture of non-overlapping sources at a frequency value is the value of the non-zero source signal's spectrum at that frequency, if any, otherwise zero. The polarity of the PS might be inverted with respect to the source signal's spectrum. The PS coefficients are equal to the non-zero source's mixing coefficients or to their opposite.*

$$PS\left\{A\vec{z}\right\}(\nu) = \begin{cases} \pm\mathcal{F}\left\{z_j\right\}(\nu) & \exists! j \ : \ \mathcal{F}\left\{z_j\right\}(\nu) \neq 0 \\ 0 & \mathcal{F}\left\{z_j\right\}(\nu) = 0 \quad \forall j \end{cases}$$

Proof. The Principal Spectrum is computed independently for each frequency value. For any specific ν, if all sources spectra are zero, then any linear combination of such spectra will be zero.

$$\mathcal{F}\{\vec{z}\}(\nu) = \vec{0} \Rightarrow \vec{a}^T A\mathcal{F}\{\vec{z}\}(\nu) = 0 \quad \forall \vec{a} \in \mathbb{R}^c \tag{173}$$

If one source spectrum $z_j(\nu)$ is non-zero, then the mixture signal is the mixing coefficients vector multiplied by the FT value.

$$A\mathcal{F}\{\vec{z}\}(\nu) = \vec{a}_{\bullet,j}\mathcal{F}\left\{z_j\right\}(\nu) \tag{174}$$

Any coefficient vector candidate $\vec{c}$ for the PS must have unitary L^2-norm, like the mixing coefficients column vector $\vec{a}_{\bullet,j}$. So, their inner product is bound between -1 and 1.

$$\vec{c}^T\vec{a}_{\bullet,j} = \|\vec{c}\|_2 \left\|\vec{a}_{\bullet,j}\right\|_2 \cos\left(\angle\left(\vec{c}, \vec{a}_{\bullet,j}\right)\right) = \cos\left(\angle\left(\vec{c}, \vec{a}_{\bullet,j}\right)\right) \tag{175}$$

When $\cos\left(\angle\left(\vec{c}, \vec{a}_{\bullet,j}\right)\right) = \pm 1$, then the squared magnitude is maximal. So, $\vec{h}(\nu)$ is equal to either $\vec{a}_{\bullet,j}$ or to its opposite.

$$\vec{h}(\nu) = \pm\vec{a}_{\bullet,j} \tag{176}$$

The PS is either the original FT value or its opposite.

$$\mathrm{PS}\left\{A\vec{z}\right\}(\nu) = \pm\vec{a}_{\bullet,j}^T\vec{a}_{\bullet,j}\mathcal{F}\left\{z_j\right\}(\nu) = \pm\mathcal{F}\left\{z_j\right\}(\nu) \tag{177}$$

Quod erat demonstrandum. □

This gives an insight into what the Principal Spectrum represents. It is able to recover the latent sources spectra when they are linearly

mixed and do not overlap in frequency. Contrarily to taking a linear combination of the time-domain signals, this exploits the potential separation in the frequency domain. Taking a different combination for each frequency allows the PS to recover the full energy of the latent sources. When analyzing multivariate mixtures, the PS magnitude can express the latent source's energy for that frequency, while coefficient vectors can be interpreted as mixing coefficients for that source.

In the case of bivariate signals, the coefficient vector at one frequency can be represented in polar coordinates (hyperspherical coordinates in two dimensions) with only one angle. This angle is equal to the $\sigma(\nu)$ angle defined for the BMS, or to $\sigma(\nu) + \pi$ (which gives the mixing coefficients with opposite polarity).

5.3.3 Orthogonal Spectra

The eigendecomposition formulation of the Principal Spectrum can be regarded as a Principal Component Analysis [66] over the channels for every frequency. As in PCA, this formulation does not only define one principal spectrum, using the eigenvector with maximal eigenvalue as the coefficient vector, but also a set of components that correspond to the other eigenvectors, called *orthogonal* spectra.

This results in a model with parametric compactness, since the user can decide how many components to keep, or how much of the signal's energy to preserve (similarly to the explained variance in PCA). In general, there can be $c - 1$ orthogonal spectra. So, in the case of bivariate signals, there is only one orthogonal spectrum, which is the value of the BMS at the angle $\sigma(\nu) + \pi/2$.

5.4 Correlation

The *Correlation* of a multivariate signal is the circular concentration of phases across channels, disregarding the polarity, as a function of frequency.

$$\mathrm{R}_{\vec{x}}(\nu) = \frac{\left|\sum_{i=1}^{c} |X_i(\nu)|\, e^{2j\angle X_i(\nu)}\right|}{\sum_{i=1}^{c} |X_i(\nu)|} \qquad (178)$$
$$\vec{X}(\nu) = \mathcal{F}\{\vec{x}\}(\nu)$$

This average is weighted by each channel's magnitude, but it can also be computed as an unweighted average.

$$\mathrm{R}_{\vec{x}}(\nu) = \frac{1}{c}\left|\sum_{i=1}^{c} e^{2j\angle X_i(\nu)}\right| \tag{179}$$

Correlation provides additional relational information about the channels, other than the energy distribution inferred with the PS coefficients. For linear mixtures of non-overlapping sources (Section 5.3.2) the correlation will be 1, since all non-zero channels have the same phase value.

Lower correlation values are indicative of violations of the assumed signal model, such as the presence of overlapping sources. It can also be the effect of convolutional or nonlinear mixing models. In general, it can be regarded as an index of the reliability of the estimate of the Principal Spectrum mixing coefficients.

5.4.1 Angular average

The correlation was defined as an angular average. The angular average of an array of angles α_i is computed from a complex number obtained by averaging the values $\exp\{j\alpha_i\}$. The values can be weighted by some convex combination coefficients $\vec{\rho}$.

$$\mathrm{AVG}_{\angle}\left(\vec{\alpha} \mid \vec{\rho}\right) = \sum_i \rho_i e^{j\alpha_i} \tag{180}$$

The angular average is the phase of this sum, while its absolute value is a measure of concentration ($1 - |\mathrm{AVG}_{\angle}|$ is the variance).

In the case of axial data, where any direction α_i is considered equivalent to its opposite $\alpha_i + \pi$, the circular average can be computed by doubling the angles [50] (or, for an arbitrary period p, the data should multiplied by $2\pi/p$).

$$\mathrm{AVG}_{\angle}\left(\vec{\alpha} \mid \vec{\rho}, p\right) = \sum_i \rho_i e^{2j\pi\alpha_i/p} \tag{181}$$

In this case, the phase of the sum should be divided by $2\pi/p$.

5.4.2 Pearson Correlation Coefficient

In the context of the BMS, correlation is expressed as Pearson's Correlation Coefficient (PCC) in the time-domain, and computed in the frequency domain as a function of the phase difference. In the case of two channels, the Correlation here defined is equivalent to the absolute value of the same PCC.

Theorem 5. *The Correlation defined as the absolute value of the unweighted circular average of the phases is equivalent, for bivariate signals, to the absolute value of Pearson's linear Correlation Coefficient computed in the time-domain.*

$$R_{\vec{x}}(\nu) = |\rho\left(a_1 \cos\left(2\pi\nu t + \phi_1\right), a_2 \cos\left(2\pi\nu t + \phi_2\right)\right)|$$

$$\begin{aligned} a_1 &:= |\mathcal{F}\{x_1\}(\nu)| & \phi_1 &:= \angle\mathcal{F}\{x_1\}(\nu) \\ a_2 &:= |\mathcal{F}\{x_2\}(\nu)| & \phi_2 &:= \angle\mathcal{F}\{x_2\}(\nu) \end{aligned}$$

Proof. As proven in the case of the Bivariate Mixture Space [69], the PCC computed in the time-domain is equal to the cosine of the phase difference.

$$\rho\left(a_1 \cos\left(2\pi\nu t + \phi_1\right), a_2 \cos\left(2\pi\nu t + \phi_2\right)\right) = \cos\left(\phi_1 - \phi_2\right) \tag{182}$$

The complex angular average of the doubled phases is ζ, and its modulo is the correlation $\mathrm{R}_{\vec{x}}(\nu)$ defined as angular average.

$$\begin{aligned} \zeta = \frac{e^{2j\phi_1} + e^{2j\phi_2}}{2} = \frac{e^{j(\phi_1-\phi_2)} + e^{-j(\phi_1-\phi_2)}}{2} e^{j(\phi_1+\phi_2)} = \\ = \cos\left(\phi_1 - \phi_2\right) e^{j(\phi_1+\phi_2)} \end{aligned} \tag{183}$$

So, its modulo is equal to the absolute value of the PCC.

$$\mathrm{R}_{\vec{x}}(\nu) = |\zeta| = |\cos\left(\phi_1 - \phi_2\right)| \tag{184}$$

Quod erat demonstrandum. □

5.5 Applications

5.5.1 GMS Enhanced Spectrogram

The same GMS model can be applied to the Short-Time Fourier Transform of multivariate signals. This has the added benefit of relaxing the constraint that the mixing coefficients should be static in time for the same frequency, and allows them to be dynamic, albeit stationary inside every STFT window. To each time and frequency cell of the STFT, the GMS can associate both the Princpal Spectrum (with its associated coefficients) and the correlation value. This allows compact time-frequency analysis of multivariate signals.

Spectrograms are often used to display the spectral content of a sound, mapping the magnitude values of the STFT to greyscale or RGB values. Along with the BMS, BS Enhanced Spectrograms were defined to compactly display the STFT of two channels as one plot. Adopting an HSV (Hue, Saturation, and Value) color model, the magnitudes of the bivariate spectrum were mapped to the value channel, the absolute value of the correlation to saturation, and mixing angles to hue. Figure 11 shows a bivariate signal in the time domain and Figure 12 shows its enhanced spectrogram.

Seeking such a direct mapping for the GMS is harder, because of the higher dimensionality of the mixing coefficients. Still, the magnitude of the principal spectrum can be mapped to value, and the correlation to saturation. There are several strategies for mapping the mixing coefficients distribution. One of such strategies is to apply PCA to the PS mixing coefficients of all STFT cells, weighted by the PS magnitude.

$$\vec{e} = \text{PCA}\left(\left\{\left\|\text{PS}\left\{\vec{x}\right\}(t,\nu)\right\|_2 \cdot \vec{h}(t,\nu) \mid \forall t,\nu\right\}\right) \tag{185}$$

The dot product of each vector $\vec{h}$ with the global PCA principal eigenvector will give a value between 1 (if they are equal) and -1 (if they are opposite), and the arccosine of such value will be the angle between them (in the range from 0 to π).

$$\vec{e}^T\vec{h}(t,\nu) = \cos\left(\angle\left(\vec{e},\vec{h}(t,\nu)\right)\right) \tag{186}$$

$$\arccos\left(\vec{e}^T\vec{h}(t,\nu)\right) = \angle\left(\vec{e},\vec{h}(t,\nu)\right) \tag{187}$$

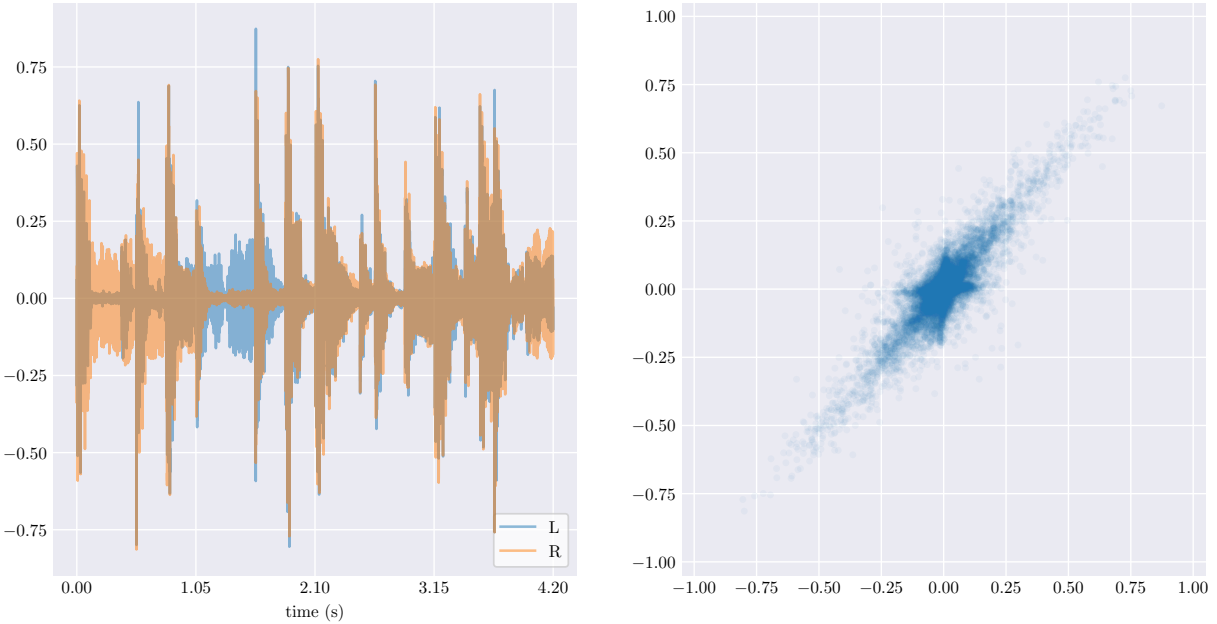

Figure 11: Time-domain signal for an excerpt of the song “Sing with Me” by Side Effects Project [71] starting at timestamp 4.475 s, decimated by 16 times for visualization. On the left the two channels in time: note that in the first second the right channel is louder, then the left channel is louder, and finally the two signals are mostly superimposed. On the right, the scatterplot shows the mixture of three distributions: one horizontal, one vertical, and one diagonal.

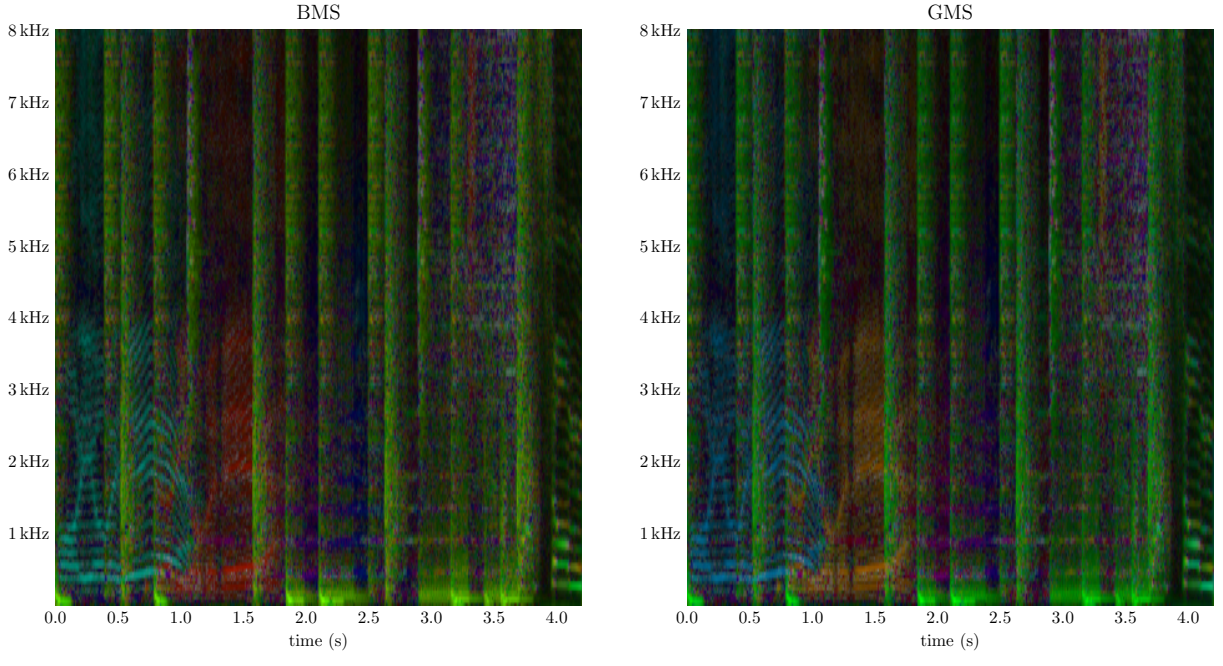

Figure 12: Enhanced spectrograms, using features from the BMS (left) and the GMS (right) for the signal shown in Figure 11. Although the colours are slightly different, the two plots are substantially equivalent. Note how three distinct colours highlight the spectra of the three components seen in Figure 11 (left, right, and middle).

Polarity is of no interest here, so the angles can be doubled to obtain an hue value between 0 and 2π. STFT cells aligned with the principal component will be plotted with the same hue, and STFT cells orthogonal to the principal component will be plotted with opposite hue.

Figure 12 shows a comparison of spectrograms enhanced with the BMS and the GMS, while Figure 13 shows the enhanced spectrogram for a multivariate signal, for which is the BMS enhanced spectrogram cannot be defined, and other ways to visualize the spectrogram of a multichannel signal. At the top left, the eight spectrograms of each individual channel show only a partial view of the signal. On the bottom left, the spectrogram of the average of all channels suffers from phase interference. E.g., the low-frequency component around half time from channels 5 and 6 is partially degraded. On the bottom center, the spectrograms computed averaging the magnitudes of all channels solves the phase interference. Still, some components are hard to make out, like the high-frequency component from channels 7 and 8, which is hidden in the middle of the broadband sounds from channels 5 and 6. On the right, the enhanced spectrogram also shows relational information. Although not as effective as in the bivariate case, it succeeds in highlighting components from channels 7 and 8 in green, and all others in magenta.

5.5.2 Multivariate Modal Audio

In a multivariate signal generated by a modal system, each channel is a mixture of the same modes. Each row in the modal system is associated with a different oscillation frequency. Such components make up the final vibration at each pickup point depending on their pickup gain. Given m modes and p pickup points, the model for the multivariate modal signal requires m distinct frequencies ν_i, m decay times d_i, m amplitudes a_i, m phases ϕ_i, and a $p \times m$ matrix of pickup gains G. Let $\vec{h}(t)$ be the vector of modes.

$$\vec{h}(t) = \begin{bmatrix} h_2(t) & h_2(t) & \dots & h_m(t) \end{bmatrix}^T \tag{188}$$
$$h_i(t) \overset{(10)}{=} u(t) a_i e^{-2t/d_i} \cos\left(2\pi \nu_i t + \phi_i\right)$$

The multivariate modal signal is a linear mix of such modes.

$$\vec{x}(t) = G\vec{h}(t) \tag{189}$$

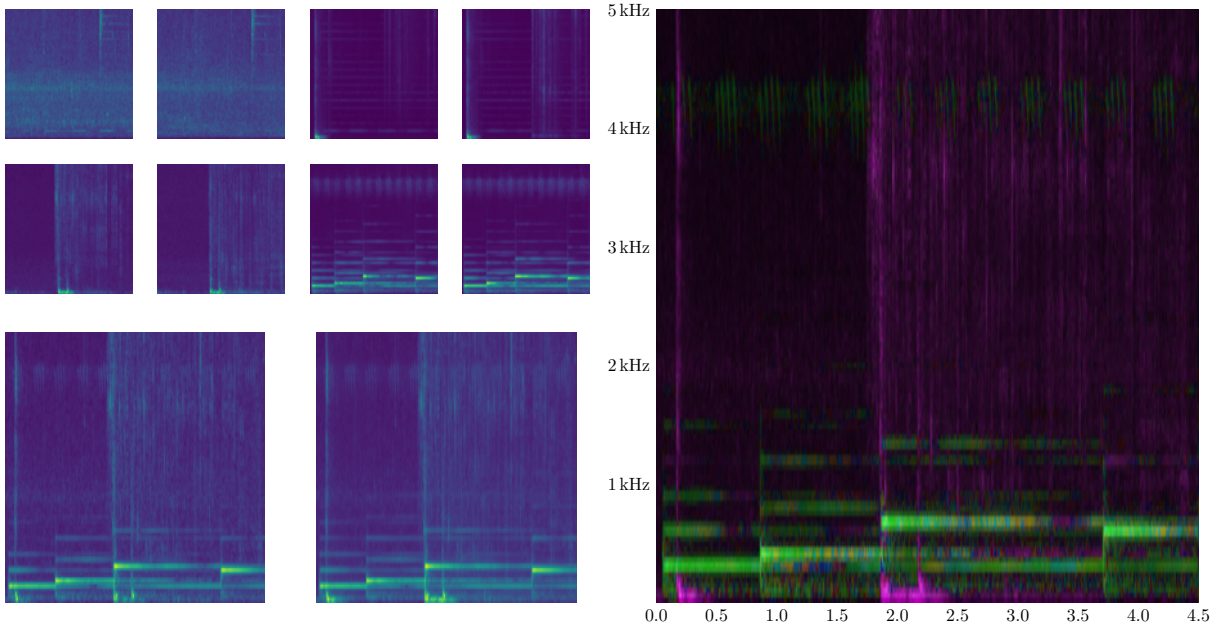

Figure 13: Excerpt of the 8-channel composition “Bosco” by Giorgio Presti starting at timestamp 10.25 s. At the top left, the 8 spectrograms of each indivdual channel. On the bottom left, the spectrogram of the average of all channels. On the bottom center, the average of the spectrograms of all channels. On the right, the enhanced spectrogram.

Since the spectrum of each mode is a bell curve, which is larger the shorter the decay time is (see Section 4.3.1), this mixture does not have the non-overlapping property. Despite this, the Principal Spectrum can still be an estimate of the modes energy in time.

Peak Detection

For every time frame of the input signal, the vanilla SAMPLE algorithm computes the FFT. In this setting, the FFT is computed for each channel and used to compute the Principal Spectrum. The magnitude of the PS can estimate the modal energy over frequency and time and is used for the standard peak detection and tracking algorithms. In addition, the peaks in the PS are annotated with the coefficient vector corresponding to their frequency.

So, at the end of the peak tracking, each track will also include a coefficient vector for each time frame. The average of the coefficients could be taken as an estimate of the modal energy distribution across channels. There are several techniques to approach the problem of averaging data points that lie on an hypersphere [81]. In this setting, there is the additional complication that a coefficient vector and its opposite should be considered as equivalent. Also, the average function it should allow non-uniform weighting of the data point.

As a baseline algorithm, the implemented average function was developed starting from the concept of spherical linear interpolation (slerp). In terms of linear interpolation (lerp), the ordinary average between two points is just their middle point.

$$\text{lerp}\,(\vec{a}_1, \vec{a}_2; t) = (1-t)\vec{a}_1 + t\vec{a}_2 \quad \vec{a}_i \in \mathbb{R}^n \;\forall i,\; t \in [0,1] \tag{190}$$

$$\text{avg}\,(\vec{a}_1, \vec{a}_2) = \text{lerp}\,(\vec{a}_1, \vec{a}_2; 0.5) = (\vec{a}_1 + \vec{a}_2)/2 \tag{191}$$

The weighted average between two points can also be expressed as a linear interpolation.

$$\text{avg}\,(\vec{a}_1, \vec{a}_2; w_1, w_2) = \text{lerp}\left(\vec{a}_1, \vec{a}_2; \frac{w_2}{w_1 + w_2}\right) = \frac{w_1\vec{a}_1 + w_2\vec{a}_2}{w_1 + w_2} \tag{192}$$
$$w_i \in \mathbb{R}_{>0}\;\forall i$$

The average between more points can be expressed recursively, where

the base case is the average of two points

$$\begin{aligned} &\text{avg}\left(\vec{a}_1, \ldots, \vec{a}_n; w_1, \ldots, w_n\right) = \\ &= \text{avg}\left(\text{avg}\left(\vec{a}_1, \ldots, \vec{a}_{n-1}; w_1, \ldots, w_{n-1}\right), \vec{a}_n; \sum_{i=1}^{n-1} w_i, w_n\right) = \\ &= \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i \vec{a}_i \end{aligned} \tag{193}$$

Defining $\vec{\mu}_i$ as the average of the first i points and ω_i its weight, the following update formula can be used to compute the average iteratively adding the difference between the current value and the current mean, multiplied by a scalar.

$$\vec{\mu}_{i+1} = \frac{\omega_i \vec{\mu}_i + w_{i+1}\vec{a}_{i+1}}{\omega_i + w_{i+1}} = \vec{\mu}_i + \frac{w_{i+1}}{\omega_i + w_{i+1}}\left(\vec{a}_{i+1} - \vec{\mu}_i\right) \tag{194}$$

$$\omega_{i+1} = \omega_i + w_{i+1} \tag{195}$$

$$\mu_0 := \vec{0} \quad \omega_0 = 0$$

Analogously, a spherical average (savg) can be defined using spherical linear interpolation. As linear interpolation returns points along the segment between the two points, spherical linear interpolation returns points along the circle arc between the two points.

$$\begin{aligned} &\text{slerp}\left(\vec{a}_1, \vec{a}_2; t\right) = \text{h2c}\left(1, (1-t)\,\angle\vec{a}_1 + t\,\angle\vec{a}_2\right) \\ &\vec{a}_i \in \mathbb{R}^n \wedge \left\|\vec{a}_i\right\|_2 = 1\ \forall i,\ t \in [0,1] \end{aligned} \tag{196}$$

Where $\angle$ is the function that computes the angles vector for the hyperspherical coordinates.

$$\vec{a} = \text{h2c}\left(\left\|\vec{a}\right\|_2, \angle\vec{a}\right) \tag{197}$$

Since polarity is indifferent, if the two vectors are pointing in opposite directions, one of the two is multiplied by -1.

$$\text{savg}\left(\vec{a}_1, \vec{a}_2; w_1, w_2\right) = \text{slerp}\left(\vec{a}_1, \text{sign}\left(\left\langle\vec{a}_1, \vec{a}_2\right\rangle\right)\vec{a}_2; \frac{w_2}{w_1 + w_2}\right) \tag{198}$$

As with linear interpolation, the spherical average between more points

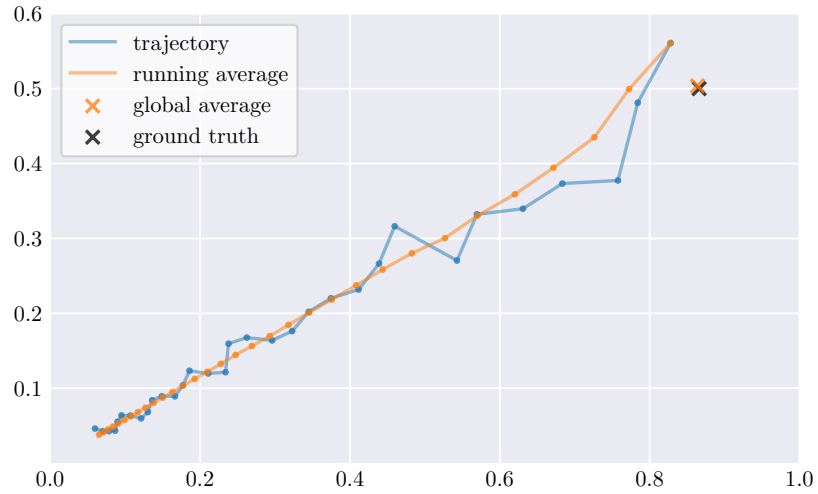

Figure 14: Example of application of spherical interpolation average to a coefficient trajectory. Coefficients are generated at angles of $\pi/6$ with random noise and weighted by the corresponding magnitude value (exponentially decreasing). The average is run in order of time, final average is displayed as a cross mark.

can be expressed recursively.

$$\begin{aligned} &\operatorname{savg}\left(\vec{a}_1, \ldots, \vec{a}_n; w_1, \ldots, w_n\right) = \\ &= \operatorname{savg}\left(\operatorname{savg}\left(\vec{a}_1, \ldots, \vec{a}_{n-1}; w_1, \ldots, w_{n-1}\right), \vec{a}_n; \sum_{i=1}^{n-1} w_i, w_n\right) \end{aligned} \tag{199}$$

And the following is the update formula.

$$\vec{\mu}_{i+1} = \operatorname{h2c}\left(1, \vec{\mu}_i + \frac{w_{i+1}}{\omega_i + w_{i+1}}\left(\angle\vec{a}_{i+1} - \angle\vec{\mu}_i\right)\right) \tag{200}$$

$$\omega_{i+1} = \omega_i + w_{i+1} \tag{201}$$

$$\mu_0 := \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \quad \omega_0 = 0$$

Figure 14 shows the application of this spherical average over a noisy coefficient trajectory.

5.6 Conclusion

The Generalized Mixture Space for multivariate signal spectral analysis was formalized. This model successfully extends the Bivariate Mixture Space as a compact frequency-dependent representation of both the energy distribution and the relationship between channels.

For the Principal Spectrum a computationally practical formulation was proven to be equivalent to the definition. This formulation exploits the eigendecomposition of a channel-wise cross-correlation matrix.

This formulation was shown to be largely equivalent to the BMS in the special case of bivariate signals, since the Principal Spectrum of the GMS is equal to the Principal Spectral Content from BMS (disregarding polarity) and the correlation computed from the angular average is equal to the modulo of Pearson's Correlation Coefficient used in BMS.

Like BMS, GMS is particularly insightful in the case of mixture signals. Interpretations for the PS coefficients are given in the case of mixture signals of non-overlapping sources and correlation was shown to be an index of validity of this assumed mixing model.

Two applications scenarios were presented: spectrogram visualization and modal audio analysis. In the future, such application scenarios should be thoroughly investigated. Also, the code for GMS analysis should be finalized and included in the SAMPLE Python package (Chapter 6).

Part II

Software and Tools

Chapter 6

SAMPLE Python Package

Abstract

This chapter introduces SAMPLE, a Python package designed for spectral analysis and modal parameter estimation. The core functionality centers around the implementation of the "Spectral Analysis for Modal Parameters Linear Estimate" (SAMPLE) algorithm. This includes a custom implementation of a Sinusoidal Analysis algorithm specifically tailored for modal tracking, based on Spectral Modelling Synthesis. Furthermore, the package offers utilities for automatic parameter tuning via a Bayesian optimization approach based on Gaussian Processes. To facilitate this process, SAMPLE includes efficient routines for computing perceptual time-frequency audio representations suitable for loss functions, including multiscale spectrograms, mel-spectrograms, and cochleagrams. Finally, SAMPLE provides a Graphical User Interface (GUI) that enables users to load and trim audio inputs, configure algorithm parameters, execute the algorithm, listen to a resynthesized

This chapter is based on [101] Marco Tiraboschi and Federico Avanzini. 'SAMPLE: a Python Package for the Spectral Analysis of Modal Sounds'. In: *Conference Proceedings of the 23rd Colloquium on Music Informatics*. "Aldo Piccialli" Award for the best contribution to the Scientific Programme. 2022, pp. 50–55. URL: https://hdl.handle.net/2434/945288

version of the input, and export the obtained results. The GUI is optionally available in the Python package and as a standalone executable.

6.1 Introduction

In the field of audio analysis, as in many others in computer science, research projects often produce software implementations of the proposed algorithms or models. Unfortunately, these implementations frequently lack proper packaging, e.g. code fragmented across numerous Jupyter notebooks or scripts. This practises hinder reproducibility, collaboration among researchers, and ultimately, accessibility by users.

The SAMPLE Python package addresses this issue for the methods presented in previous chapters. It includes implementations of several algorithms for audio analysis and the automatic estimation of modal parameters from audio examples, including SAMPLE (Chapter 3) and BeatsDROP (Chapter 4).

SAMPLE is available via the Python Package Index, on GitHub, and on Zenodo [99]. It can be installed as a Python package or used through a graphical user interface (GUI) as a standalone executable.

All public classes, methods and functions are documented in the source code with Google-style docstrings and converted into an HTML documentation website with the Sphinx [113] package. The package is comprehensively tested with unit tests using the Python built-in module *unittest* [42]. Code coverage is 100% for the entire codebase, excluding the GUI module. The GitHub and Zenodo repositories also include code for the evaluations protocols for the different methods and for figure generation. A set of interactive tutorials, in the form of Jupyter notebooks, complement the package.

6.2 SAMPLE

This section details the implementation of the Spectral Analysis for Modal Parameters Linear Estimate (SAMPLE) algorithm (Chapter 3). The primary objective behind the creation of this package was to provide users with a readily available, distributable, and user-friendly software for this method.

The core component of the package is the `SAMPLE` class. This class adheres to the design patterns established by the scikit-learn library API [18, 67]. Each instance of SAMPLE encapsulates a sinusoidal model and a regressor, which can be either linear or semi-linear.

By fitting these SAMPLE class instances to audio samples, users can estimate the modal parameters (amplitudes, frequencies, and decay times) of the objects being recorded. It is important to note that the input audio should be the sound generated by striking the object of interest, i.e. a recording of its impulse response.

6.2.1 Sinusoidal Model

The `SinusoidalModel` class implements the analysis algorithm used in Sinusoidal Modelling Synthesis [87, 88, 89] (SMS). It prepares the input Short-Time Fourier Transform (see Section 2.2.4) and runs a `SineTracker` which detects and organizes the peaks in the STFT. Additionally, the class allows for processing audio in reverse order, a technique commonly used in analysis methods for additive synthesis [63].

The `ModalModel` and `ModalTracker` classes inherit from `SinusoidalModel` and `SineTracker`, respectively, and introduce functionalities specifically tailored for modal sounds. During the analysis phase, a `ModalTracker` instance can enforce a set of constraints. It can discard trajectories where the identified modal frequencies fall outside the desired range (e.g., infrasounds or ultrasounds). It can also eliminate ill-behaved trajectories, whose magnitude exhibits an increase over time rather than the expected decrease. Furthermore, the class can estimate the starting amplitude of a trajectory that has stopped and discard it if it is deemed too quiet. Finally, a stopped trajectory can be merged with a preceding one if their frequencies are sufficiently similar.

6.2.2 Hinge Regression

The `HingeRegression` class implements the regression algorithm used to estimate modal amplitudes and decay times. This algorithm is based on the Rectangular Trust Region Dogleg approach [117] (DogBox) for nonlinear least-squares optimization, as implemented in SciPy [116].

HingeRegression works by fitting an amplitude trajectory obtained from the sinusoidal model to a hinge function, denoted as $h_{k,q,\alpha}(t)$.

This function is linear for values of t less than α and remains constant thereafter.

$$h_{k,q,\alpha}(t) \overset{(55)}{:=} k \cdot \min(t, \alpha) + q \tag{202}$$

The utilization of hinge regression offers an advantage over linear least-squares regression in scenarios with high noise floors. In such cases, linear regression exhibits a bias towards underestimating both the amplitude and decay time values (see Section 3.2.2).

6.2.3 BeatsDROP

The package also includes the implementation of the "Beats Duality for the Resolution Of Partials" (BeatsDROP) algorithm (Chapter 4). The `DualBeatRegression` class implements a regression model for two modes of oscillation with very similar frequencies. In this case, the sinusoidal model would not find two different trajectories, but it would output a single trajectory, in which the two partials interfere.

`DualBeatRegression` fits the modal frequencies, amplitudes and decay times of both partials to the sinusoidal model trajectory. The loss function includes the differences for both the amplitude and the frequency of the trajectory. This algorithm exploits the fact that beats of uneven amplitudes produce modulations of both amplitude and frequency. Its parent class `BeatRegression` only considers the amplitude for the loss function and it is the baseline algorithm used for evaluation in Section 4.5. This regessor can be used instead of `HingeRegression`, to find two sets of modal parameters from one trajectory.

The `Beat` and `ModalBeat` classes define all the intermediate and final signals used to model beats as sinusoids with modulated amplitude and frequency (see Section 4.2). This signals are defined using a computational graph with the Paragraph package [16], this ensures that, depending on the desired output, unused intermediate variables are not computed and that each variable in only computed once.

The `SAMPLEBeatsDROP` class inherits from `SAMPLE` and applies the `DualBeatRegression` to the trajectories, inferring whether the trajectory is a beat or not with the decision criterion defined in Equation (4.4.5), implemented by the `AlmostNotABeatDecisor` class. Beat regression can be performed concurrently in multiprocessing, since the regression problems of different trajectories are independent.

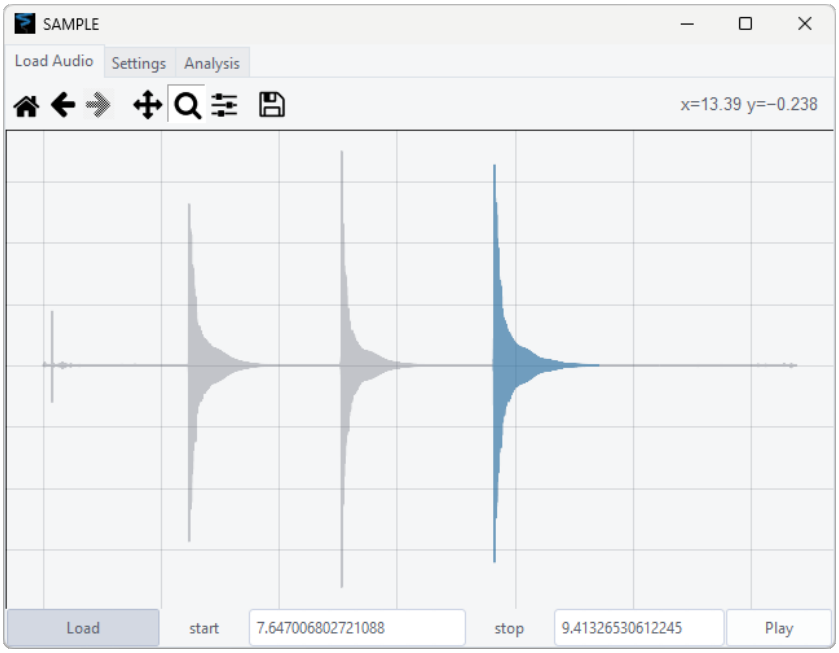

Figure 15: The *"Load Audio"* pane of the GUI, as it renders on Windows 11 with the *arc* theme. In the this pane, the user can load, trim and play the audio input.

6.3 Graphical User Interface

A Graphical User Interface (GUI) was designed to facilitate access to the presented methods for users without programming experience. The GUI enables users to load an audio file, apply the SAMPLE model, and export the results in a JSON format. A series of screenshots depicting the GUI in action are presented in Figures 15, 16, and 17.

6.3.1 Audio Loading

The first pane of the GUI, labelled "Load Audio", is dedicated to audio loading and manipulation (Figure 15). This pane allows the user to load and optionally trim a target audio file. The "Load" button triggers a file browser window, enabling the user to locate and choose the desired

audio file for processing. Upon successful loading, a visual representation of the audio waveform is displayed within the GUI. Additionally, an algorithm based on onset detection [12] is employed to automatically initialize a region of interest (ROI) within the waveform, which is then highlighted with a distinct color.

The SAMPLE model operates exclusively on the chosen ROI of the audio data. Users are provided with the ability to modify this ROI by specifying their preferred start and stop timestamps within dedicated text input fields. Alternatively, a they can use the waveform display. Clicking on the waveform allows the user to adjust the start or stop timestamp (depending on which is closer to the click location), and the value is finalized on mouse release. Furthermore, users can zoom in and out on the waveform display for a finer selection of the ROI.

A “Play” button is included within the interface, enabling the user to preview the currently selected ROI of the audio file. Pygame [118] is used to ensure compatibility of audio playback across different operating systems.

6.3.2 Settings

The second pane of the GUI, labeled "Settings", is dedicated to configuring various parameters of the SAMPLE (and, eventually, BeatsDROP) algorithms (Figure 16).

Resynthesis

This parameter solely affects the resynthesis stage and does not influence the analysis phase.

- `n modes`
  This parameter controls the maximum number of modal components that will be reconstructed by the model during post-processing. The model prioritizes modes with the highest energies for resynthesis. Users can adjust this value to control the level of audio detail captured in the final output. It’s important to note that this parameter solely affects the resynthesis stage and doesn’t influence the analysis phase.

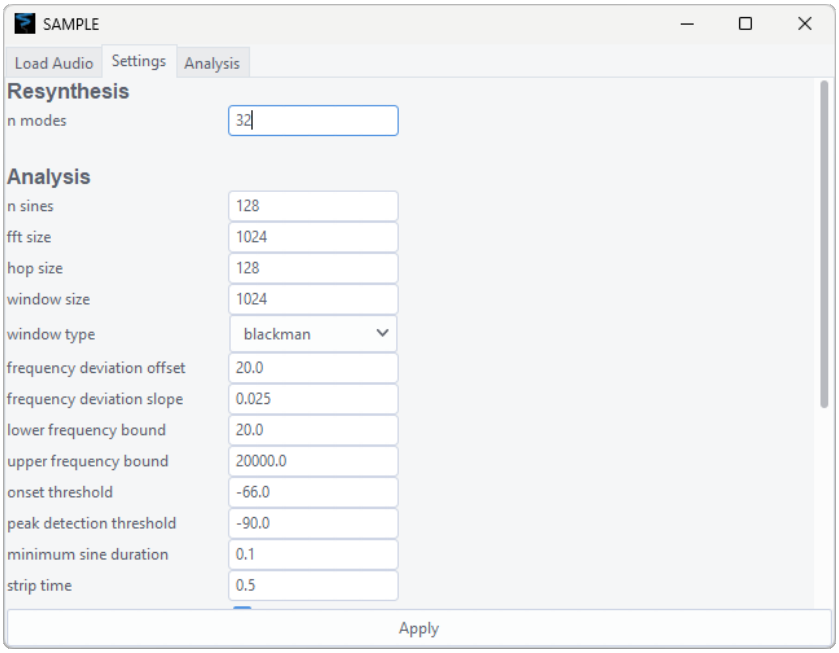

Figure 16: The *"Settings"* pane of the GUI, as it renders on Windows 11 with the *arc* theme. In the this pane, the user can tweak the algorithm parameters.

Analysis

These parameters affect the SAMPLE sinusoidal analysis algorithm.

- `n sines`
  This parameter specifies the maximum number of sinusoidal peaks the SAMPLE algorithm will track within each Short-Time Fourier Transform (STFT) frame.

- `fft size`
  This setting determines the size of the Fast Fourier Transform (FFT) applied to each frame. User input for this parameter is automatically rounded up to the nearest power of 2 for efficient computation.

- `hop size`

This parameter defines the distance (in samples) between consecutive STFT frames along the time axis. It controls the temporal granularity of the analysis.

- `window size`
  This setting specifies the size (dimension) of the window function used during STFT analysis, provided in samples. Unlike FFT size, the window size doesn't have to be a power of 2. However, it's limited by the current value set for the FFT size.

- `window type`
  This parameter allows users to select the desired window function from those supported by the SciPy library [116]. The chosen window function should not require any additional arguments besides the window size.

- `frequency deviation offset`
  This parameter sets the threshold for considering peaks as continuations of the same trajectory at 0 Hz. Successive peaks with a frequency difference below this threshold will be organized in the same trajectory.

- `frequency deviation slope`
  This parameter controls the rate of change for the peak continuation threshold with respect to frequency. The threshold at frequency ν is $\tau(\nu) = \tau_{\text{offset}} + \tau_{\text{slope}}\,\nu$.

- `lower frequency bound`
  This parameter defines the minimum acceptable frequency (in hertz) for including a trajectory in the final analysis. Trajectories below this threshold will be discarded.

- `upper frequency bound`
  This parameter defines the maximum acceptable frequency (in hertz) for including a trajectory in the final analysis. Trajectories above this threshold will be discarded.

- `onset threshold`
  This parameter sets the minimum threshold for initial amplitude values (in dBFS). If the intercept of the linear regression of a trajectory falls below this threshold, the trajectory is discarded.

- `peak detection threshold`
  This parameter defines the minimum amplitude (in dBFS) for detecting a peak within the STFT.

- `minimum sine duration`
  This parameter sets the minimum required length (in seconds) for a trajectory to be considered valid. Trajectories shorter than this duration will be discarded.

- `strip time`
  This parameter defines the maximum allowed delay (in seconds) for the onset of a trajectory. Any trajectories whose onset falls after this limit will be discarded.

- `reverse`
  This parameter allows users to control whether the audio signal is processed forward or backwards.

- `zero-padding`
  This parameter toggles the initial zero-padding of the signal. If zero-padding is on, the signal will start in the middle of the first STFT frame. If it is off, the signal will start at the beginning of the first STFT frame.

BeatsDROP

These parameters control the BeatsDROP regression algorithm.

- `BeatsDROP`
  This toggle allows the user to include or remove BeatsDROP from the analysis pipeline. If this is off, all other parameters in this section have no effect.

- `multiprocessing`
  This parameter allows users to control whether the regression for all trajectories is performed concurrently or not.

- `n jobs`
  This parameter determines the number of processes in the pool for running different BeatsDROP instances in parallel, if multiprocessing is on.

- `threshold`
  This parameter controls the decision rule threshold, in machine epsilons. Values below this threshold will be considered as functionally equivalent to zero and will result in a negative test result.

- `lpf`
  This setting is the cutoff frequency for the autocorrelation vector computation. The Power Spectral Density will be assumed to be zero above this frequency.

GUI

This parameter solely affects the GUI appearance.

- `gui theme`
  This parameter enables users to select the theme applied to the graphical user interface. The theme options are defined within the ttkthemes library [78]. The chosen theme will be applied upon the next application launch. When switching themes, users will be prompted to confirm whether they want to immediately reload the GUI with the new theme.

6.3.3 Analysis

The last pane of the GUI is the *"Analysis"* pane (Figure 17). The "Analyze" button starts the analysis of the target audio file with the SAMPLE algorithm. A progress bar visually indicates the analysis progress.

Upon completion of the analysis, the GUI populates the visual display areas. The top-left subplot presents a greyscale spectrogram of the target audio. Overlaid on this spectrogram, the frequency trajectories are plotted as coloured lines. The top-right subplot visualizes the amplitude trajectories as coloured line plots, with colours matching the corresponding frequency trajectories plots. Finally, the bottom subplot displays the waveform of both the target audio and of an audio resynthesized with the estimated modal parameters. The resynthesized audio is generated through a straightforward additive synthesis approach,

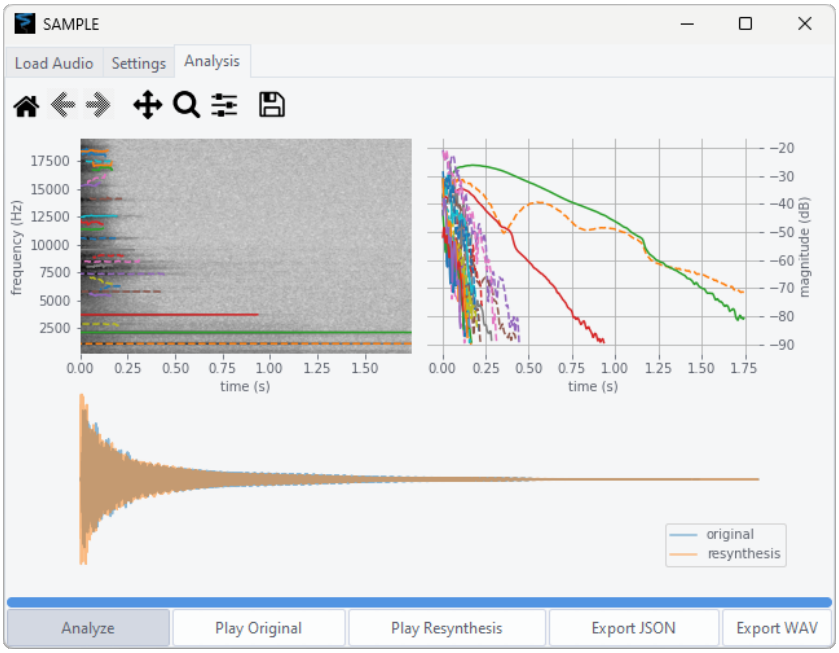

Figure 17: The *"Analysis"* pane of the GUI, as it renders on Windows 11 with the *arc* theme. In the this pane, the user can run the algorithm, play an audio resynthesis and export the estimated modal parameters.

where all sinusoidal components have exponentially-decaying envelopes.

$$\hat{x}(t) = \sum_{i=1}^{\text{n\_modes}} \hat{h}_i(t) \overset{(10)}{=} \sum_{i=1}^{\text{n\_modes}} a_i e^{-\frac{2t}{d_i}} \cos\left(2\pi\nu_i t + \phi_i\right) \tag{203}$$

The user has the ability to compare the target audio with the resynthesized version through dedicated buttons. Clicking on the "Play Original"' button triggers playback of the target audio, while clicking on the "Play Resynthesis" button starts playback of the resynthesized audio.

The GUI also offers functionalities for exporting the inferred modal parameters. Clicking on the "Export JSON" button saves these parameters as a JSON file. Similarly, the "Export WAV" button allows the user to save the resynthesized audio to a WAV file. In both cases, a file

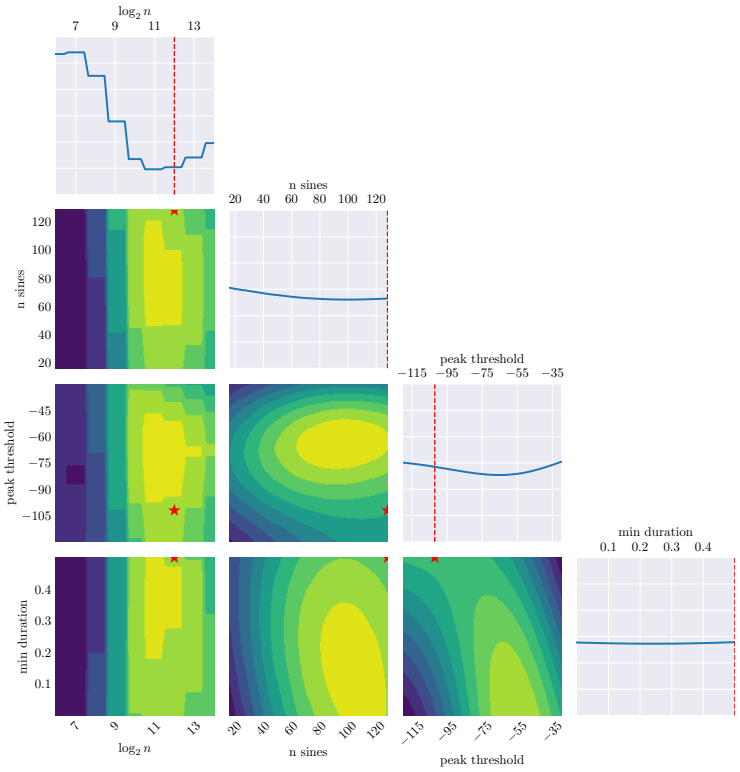

Figure 18: Partial Dependence plots of the cochleagram loss for SAMPLE hyperparameter optimization on a synthetic modal sound.

browser window will appear, enabling the user to specify the desired save location.

6.4 Automatic Optimization

The SAMPLE algorithm has many hyperparameters and it is not always obvious a-priori what the best values might be for a specific target sound. For this reason, an interface module was developed to apply automatic hyperparameter optimization to SAMPLE. Figure 18 shows the loss function values estimated by the minimization process while optimizing the analysis window size, the number of peaks, the peak

threshold and the minimum trajectory duration. A partial dependence plot (PDP) shows "the dependence between the target response and a set of input features of interest, marginalizing over the values of all other input features" [67]. Since visualizing a function of more than two inputs would be impossible, PDPs visualize different *views* of the same function. The line plots on the diagonal show the dependence of the loss function on one hyperparameter at a time. The contour plots in the lower triangle show the dependence of the loss function on two hyperparameters at a time.

The `SAMPLEOptimizer` class provides a flexible framework for defining the optimization problem. Developers can specify a set of hyperparameters that will be excluded from optimization. A recommended practice is to predetermine a maximum number of modes for resynthesis to prevent overfitting. Utilizing a large number of modes can lead to modelling noise as sinusoids, even with minimal improvement in the loss function. The optimal value depends on the target sound's complexity, but satisfactory results are typically achieved with 16 to 128 modes. Alternatively, the developer can specify hyperparameters as Scikit-Optimize [44] dimensions. These parameters will be optimized using `skopt.gp_minimize`, a Gaussian Process-based Bayesian optimization function. Certain arguments of the `SAMPLE` class are not suited for direct optimization. For instance, the analysis window is a high-dimensional vector, and optimizing each individual value would be impractical. Instead, the goal usually is to identify the optimal window size and type. A remap function can be employed to transform hyperparameters into valid arguments for the constructor. The default remap function offers functionalities such as specifying the logarithm of the FFT size, defining the window using a name and size relative to the FFT size, and defining the window overlap as a fraction of the size, instead specifying the hop size in samples.

The optimization objective is defined by the loss function. This function takes two arrays (one containing the original samples and another containing the resynthesized samples) and returns a score, with lower values indicating better performance. The default loss function is a multi-scale spectral loss.

6.4.1 Loss Functions

This section describes several time-frequency audio representations that are implementation in the SAMPLE package. They can be used to define perceptually-based loss functions.

Multi-scale Spectral Loss

The first implemented loss function is the multi-scale spectral loss [29]. This function calculates the weighted sum of the distances between the spectrograms of two sounds, represented in both linear amplitude and decibel scales, for various window sizes, that can be chosen by the user. A multi-scale spectral loss function is defined by set of spectrogram functions S, by the Minkowski distance order p, and by a scaling parameter α for the distance of logarithmic-magnitude spectra.

$$\mathcal{L}_{S,\alpha,p} := \sum_{s \in S} \left[\|s(x) - s(y)\|_p + \alpha \left\|\log_{10} s(x) - \log_{10} s(y)\right\|_p \right] \quad (204)$$

For improved efficiency, the implementation allows developers to specify a number of jobs or a process pool to perform computations of different spectrograms in parallel. Similar loss functions can also be found in other packages, like the "auraloss" [94] package for PyTorch.

Cochleagram Loss

A cochleagram is another time-frequency representation that mimics human auditory perception by adapting its time and frequency resolution with respect to frequency [119]. This characteristic eliminates the need for computing cochleagrams at various resolutions for defining a perceptually-based loss function.

Other approaches that try to get closer to human perception than spectrograms often use the mel scale [96]. Mel-spectrograms are just spectrograms whose frequency axis has been rescaled according to the mel scale. Mel-Frequency Cepstral Coefficients (MFCCs) are the Discrete Cosine Transform of the magnitudes of mel-frequency spectra.

Instead, the cochleagram function is implemented as a set of convolutions of the audio signal with gammatone filter impulse responses (IRs). Traditionally, this convolution is followed by a simple nonlinearity like half-wave rectification. The implementation offers the option

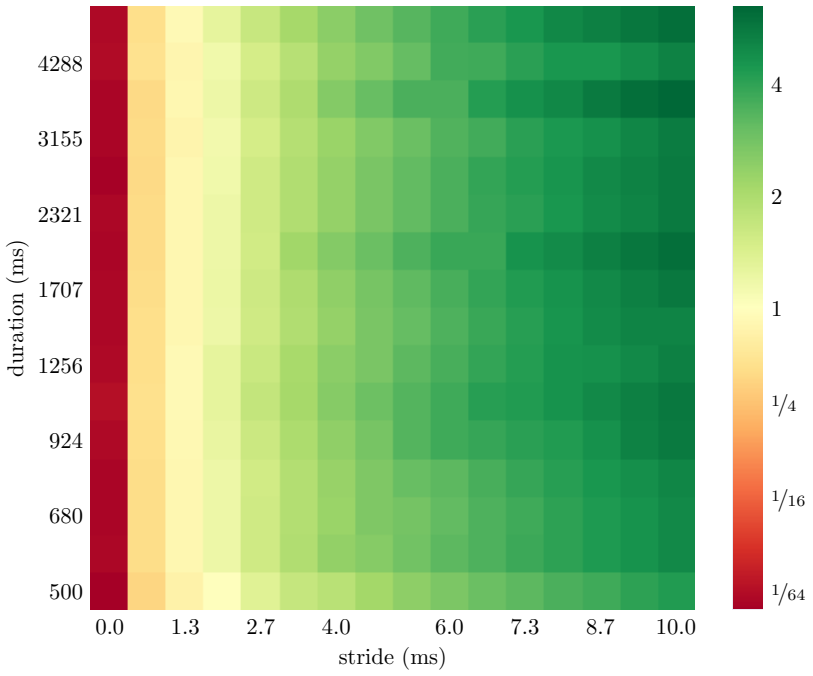

Figure 19: Speed-up of cochleagram method using the custom implementation for strided convolution. Speed-up is the ratio between the run-time of the custom implementation and the run-time of the best of SciPy convolution methods. Speedup is evaluated for different input sizes and stride lengths (0 ms means 1 sample of stride) at 44.1 kHz averaging the run-times of 32 trials per case.

to perform the convolution with the analytic signals of the IRs. This approach is equivalent to, but computationally faster than, explicitly calculating the analytic signal of the output [92]. In this case, the resulting cochleagram is complex-valued, and the default nonlinearity is the absolute value function to obtain a real-valued output.

This cochleagram has the same sampling rate as the input signal. This can be redundant, especially for defining a loss function. Therefore, the cochleagram is downsampled to a desired rate. However, this approach can be wasteful as it involves computing a high-resolution representation followed by discarding a significant portion of the data.

To address this inefficiency, a custom function for strided convolution was implemented. In a strided convolution with stride $s \in \mathbb{N}^+$, the output at sample i is equivalent to the output of a standard convolution at sample $s \cdot i$.

$$(x *_s y)_i := (x * y)_{si} \tag{205}$$

This implementation offers significant speedup compared to the full convolution methods available within the SciPy library [116]: direct, FFT, auto (direct or FFT depending on input size) and overlap-and-add [56]. Figure 19 illustrates the average speedup achieved with different strides and input sizes.

6.5 Conclusion

This chapter presented the SAMPLE package for Python and its core components. The package leverages unit tests with comprehensive code coverage to ensure its functionality.

A general overview of the classes which implement the SAMPLE (Chapter 3) and BeatsDROP (Chapter 4) algorithms for modal sound analysis is provided. These classes offer programmatic access to the functionalities of these algorithms.

The Graphical User Interface (GUI) distributed with the package is described. This GUI is a self-contained environment for users to:

- Select the target audio regions from a file
- Configure hyperparameters for SAMPLE and BeatsDROP
- Execute the algorithm
- Visually and auditorily inspect, and export the generated results

Finally, an interface module for Bayesian hyperparameter optimization function based on Gaussian Processes is introduced. This module allows for automated optimization of the algorithm's hyperparameters. For this purpose, loss functions such as multi-scale spectral losses and cochleagram-based losses are also implemented in the package.

Future development plans for the SAMPLE package include the incorporation of any additional methods related to SAMPLE that may be developed, such as the GMS (Chapter 5). Additionally, the package will

transition to a different hyperparameter optimization library due to the discontinued maintenance of Scikit-Optimize.

Chapter 7

Sound Design Toolkit

Abstract

This chapter delves into the advancements made on the Sound Design Toolkit (SDT). The work presented herein encompasses both the re-engineering of the library and the introduction of novel functionalities. It is noteworthy that since the undertaking of its re-engineering, the SDT has achieved inclusion within the official Max package manager, signifying its broader recognition and adoption. The chapter also describes the adoption of a standardized approach for representation and communication of SDT parameters. This approach centers on the utilization of JSON as the representation format, and OSC as the communication protocol. JSON allows to represent in the same format the parameters for SDT objects, irrespective of the programming language (C, Pure Data, or Max) and of the operating system (macOS, Windows, or Linux). The chapter further elaborates on the integration of an OSC server. This integration serves to significantly enhance the capabilities

This chapter is based on [106] Marco Tiraboschi, Stefano Papetti and Federico Avanzini. 'Just a Sounding Object Notation: Sharing Objects for Sonic Interaction Design with JSON and OSC'. in: *Proceedings of the 4th International Symposium on the Internet of Sounds*. 2023. DOI: 10.1109/IEEECONF59510.2023.10335232

of the SDT in terms of interoperability with a multitude of other software platforms. A concrete use case is presented. This use case exemplifies the seamless integration between the SAMPLE Python package, employed for modal analysis, and SDT.

7.1 Introduction

The Sound Design Toolkit (SDT) is an open-source software package, released under GPLv3 license, and available online on GitHub and Zenodo [26]. It offers a collection of "perceptually founded sound models" [5] that enable the interactive generation of various acoustic phenomena, including interactions between solid objects, liquids or gases, and machines.

7.1.1 The Toolkit

SDT can be considered a "virtual Foley box", providing a rich library of software sound generators for sound designers to sketch and prototype sonic interactive behaviors [62]. The core elements of SDT are physics-based procedural audio models [31], designed with acoustic principles in mind and whose parameters have clear physical meaning. The primary application domain lies in sound design for multisensory interactive systems, like videogames or extended reality (XR), where the focus is on modelling a realistic relationship between action and sound, rather than the sound quality out of context.

This approach is the result of over two decades of research in sound synthesis and design [80]. The initial set of sound models were developed within the EU project Sounding Object (SOb) as externals and patches for Pure Data [73], a popular open-source visual programming language for sound. The entire package was also subsequently ported to the Max [72] environment, a commercial visual programming language by Cycling '74. Max is based on the same paradigm as Pure Data, but offers a nicer graphical user interface and more built-in features, although there have been several distributions of Pure Data that improved on these points, like the recently developed Plug Data [86]. This porting was made keeping the core implementation in a common C library, which can also be used in any programming language able to import C libraries (even Python, as was done in Section 3.3). The

Pure Data and Max packages are simply implemented as the wrapping of these core functions for the respective frameworks' APIs.

Within the scope of the recent EU project SkAT-VG (Sketching Audio Technologies using Vocalizations and Gestures), the SDT software architecture underwent a significant revision, and the library was further expanded with more sound models. The goal was to provide a palette of sounding objects encompassing a significant range of acoustic phenomena relevant to major sound design applications. These objects are based on an ecological approach to everyday listening [38, 39], organized to reflect current knowledge on the perception and categorization of everyday sounds [47].

7.1.2 Contribution

When dealing with the SDT, or any other sound design library, sonic interaction designers must navigate a high-dimensional space defined by the multitude of parameters of the sounding objects they are handling. This high dimensionality, while fostering creative exploration, presents challenges due to two primary factors.

Firstly, the absence of a standardized exchange format hinders the transition between different visual environments, like Pure Data and Max, and production software, such as VST plugins developed in C++ with JUCE or XR experiences, often programmed in C# with Unity or in C++ with Unreal Engine. This not only impedes the design workflow but also burdens SDT's own developers and designers, who must develop from scratch programs for each distribution (Pure Data and Max), maintaining different copies of the same patches. Furthermore, the lack of a shared representation format exposes the existing gap between sound analysis software, used for parameter estimation and fitting, and the synthesis software of SDT.

Secondly, real-time interoperability with other software relies on platform-specific utilities. This often results in over-complicated *spaghetti code*, especially in Pure Data and Max, where a patch with a large number of control connections fits the idiom particularly well. The SDT needed to adopt an effective communication protocol.

This work proposes a solution that addresses these limitations by adopting JSON as the representational syntax and exchange format for the parameters of virtual sounding objects. Additionally, an OSC [121]

server is implemented to enable seamless interoperability and centralized control. Finally, a preliminary example showcasing interoperability is presented, where the SAMPLE software for modal analysis (Chapter 6) can already generate JSON files compliant with the SDT specifications.

7.2 Development

This section delves into the integration of JSON and OSC within the SDT codebase, and in the development of related features accessible in any distribution of SDT.

7.2.1 JSON

The JavaScript Object Notation (JSON) format has emerged as a prevalent method for data interchange due to its inherent advantages of simplicity, flexibility, and seamless integration across diverse platforms and programming languages. JSON provides a practical and efficient mechanism for structuring and transmitting data between server-side and client-side applications. JSON has also seen widespread adoption across various domains within Web and Internet of Things (IoT), including the Internet of Sounds [112]. In contrast, XML has seen a decline in preference due to its verbose nature.

At its core, JSON leverages two fundamental data structures: objects and arrays. These structures offer a versatile approach to representing a wide range of data types and hierarchical relationships within the data. An object, denoted by curly braces (`{ }`), comprises key-value pairs. Each key is a string enclosed in double quotation marks (`" "`), followed by a colon (`:`), and linked to its corresponding value. These values can be of various data types, including strings, numbers, arrays, or even nested objects themselves. The key-value pairs within an object are separated by commas.

Arrays, represented by square brackets (`[ ]`), serve as ordered collections of values. Similar to object values, elements within JSON arrays can be of any data type, including objects and nested arrays, enabling the representation of complex data structures.

The integration with SDT leverages the JSON implementation by

James McLaughlin *et al.* [59, 60], to which this work has also contributed. This specific implementation aligns perfectly with the project's requirements due to its low memory footprint and its portability, since it is compliant to the ANSI C standard.

Any object implemented as a C `struct` within the SDT library is represented as a JSON object. Keys in this JSON object correspond to the SDT object's attributes, while the corresponding attribute values serve as the values. The JSON dump and load functions, respectively, call the corresponding getter and setter functions for each attribute.

Some SDT attributes hold multiple values. These attributes are represented as JSON arrays. For instance, modal resonators have two attributes defining the size of other attributes: the number of modes and the number of pickup points. The `SDTResonator` structure stores a modal frequency, decay time, and modal weight value for each resonance mode to be simulated. Consequently, modal frequencies, decay times, and modal weights are represented as three separate arrays. Since each mode has its own gain factor at each pickup point, modal gains are represented as a nested array. Listing 1 showcases an example JSON representation of a resonator.

Considering the potential need for users to call the load function at audio rate, a safety flag argument exists within each JSON load function. When it is false, all attributes can be set without limitations. Conversely, with the flag set to true, the JSON loader refrains from modifying any attribute that would necessitate memory allocation or deallocation, opting instead to display a warning message. Attributes requiring particular attention include the sizes of memory buffers for analysis objects and the cardinality of array-valued attributes (e.g., the number of modes in a resonator). By design choice, the load functions exposed via the OSC server always set the safety flag to true.

Within this context, a collection of SDT objects is referred to as a *project*. These projects can encompass, for example, the SDT objects loaded within a Pure Data patch or all the components necessary for a VST plugin. A project is represented as a JSON object containing a single key-value pair for each SDT object type. The key identifies the type (e.g., `resonators`, `frictions`, `bubbles`, etc.), and the value is itself a JSON object. Inside this nested object, there exists a key-value pair for each instance of that type within the project. The key acts as the object's identifier, while the value is the JSON representation of the

```
{
  "nModes": 20,
  "nPickups": 1,
  "activeModes": 20,
  "fragmentSize": 1.0,
  "freqs": [
    8460.87, 8452.44, 2079.75, 7361.63, 8467.28,
    3678.45, 1065.26, 12572.1, 1063.06, 9533.82,
    5762.19, 12596.3, 10555.4, 17511.9, 5769.25,
    16935.2, 16943.8, 10561.7, 2848.88, 2834.24
  ],
  "decays": [
    0.067, 0.079, 0.558, 0.103, 0.106,
    0.284, 1.100, 0.066, 0.793, 0.078,
    0.128, 0.125, 0.101, 0.048, 0.130,
    0.062, 0.049, 0.197, 0.047, 0.100
  ],
  "weights": [
    1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0
  ],
  "gains": [
    [
      1.39, 1.20, 0.23, 0.30, 0.24,
      0.08, 0.04, 0.13, 0.03, 0.09,
      0.06, 0.06, 0.05, 0.07, 0.03,
      0.04, 0.04, 0.01, 0.03, 0.02
    ]
  ]
}
```

Listing 1: Example of a JSON file with parameters for a modal resonator estimated from a glass sound using SAMPLE. This JSON object has four scalar attributes, and three array-valued attributes (`freqs`, `decays`, and `weights`) of size `nModes`. Finally, `gains` is a nested array of `nPickups` arrays of size `nModes`.

SDT object itself. Projects, therefore, provide a convenient mechanism for saving and loading the parameters for all relevant SDT objects in a single operation. To be accessible in projects, the objects' keys must be registered within a hash map. In Pure Data and Max, any object instantiated specifying the identifier argument is automatically registered in the appropriate hash map for the type.

7.2.2 Open Sound Control

The Open Sound Control [121] (OSC) protocol, operates at the application layer and can be transmitted over either TCP or UDP. OSC is primarily designed to facilitate communication between computers and multimedia devices, with a particular focus on digital musical instruments.

An OSC client refers to any software application that transmits OSC packets, which are the fundamental units of data exchange within the OSC protocol. Conversely, an OSC server is any application capable of receiving OSC packets. Each OSC server offers a defined set of OSC methods. These methods represent potential destinations for incoming OSC messages and usually correspond to the controllable aspects of a sound processing application. All OSC methods are organized hierarchically within an OSC address space. This structure resembles a tree, where OSC methods occupy the terminal nodes (leaves), and the intermediate branching nodes are called OSC containers. Notably, the OSC address space is not static; it can dynamically evolve over time.

Every OSC method and container is uniquely identified by a symbolic name expressed as an ASCII string. An OSC address, associated with a specific OSC method, represents the complete path leading to that method within the OSC address space. This path starts with the root symbol `/`, followed by the names of all the preceding containers in a pre-order traversal, and concludes with the name of the target OSC method itself. Forward slashes (`/`) serve as separators between these names. It is worth mentioning that the syntax employed for OSC addresses closely resembles the syntax used for URLs.

The choice of communication protocol for the Sound Description Toolbox (SDT) was informed by two key considerations: prevalence and flexibility. OSC met these requirements effectively. By leveraging OSC, the SDT benefits from a widely adopted standard, ensuring efficient message exchange and enabling interoperability with various software environments. This approach avoids the need to reinvent low-level networking functionalities, allowing development efforts to focus on the core functionalities of the SDT.

Several existing projects implemented in Pure Data demonstrate the successful use of OSC libraries for communication. These projects often rely on the libraries developed by Martin Peach [41] or the

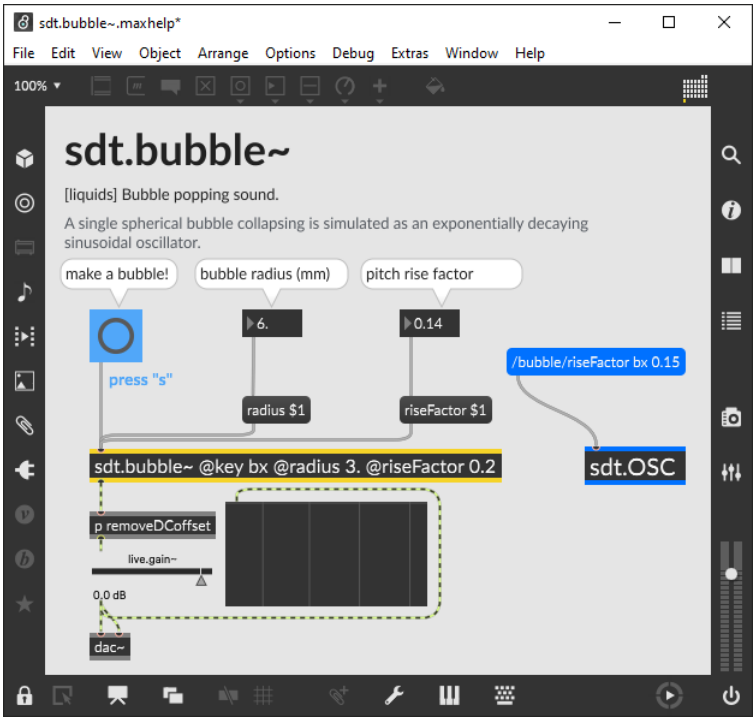

Figure 20: Example of the Max help patcher for `sdt.bubble~` objects (in yellow), with the addition of the `sdt.OSC` object and an OSC message (both in blue) to set the "rise factor" parameter.

built-in networking objects (`netreceive` and `netsend`), message parsing objects (`oscparse`), and message formatting objects (`oscformat`). Conversely, Max provides native support for network I/O with UDP, simplifying communication for its users. Programmers opting for traditional text-based languages have the freedom to select their preferred networking library from the vast array of available options.

To facilitate OSC communication within the SDT, a new function named `SDTOSC` was developed. This function is responsible for routing and processing OSC messages. It is accessible in both the Max (Figure 20) and Pure Data (Figure 21) distributions of the SDT as new objects, named `sdt.OSC` and `sdtOSC`, respectively.

The topmost level container of OSC addresses for SDT corresponds

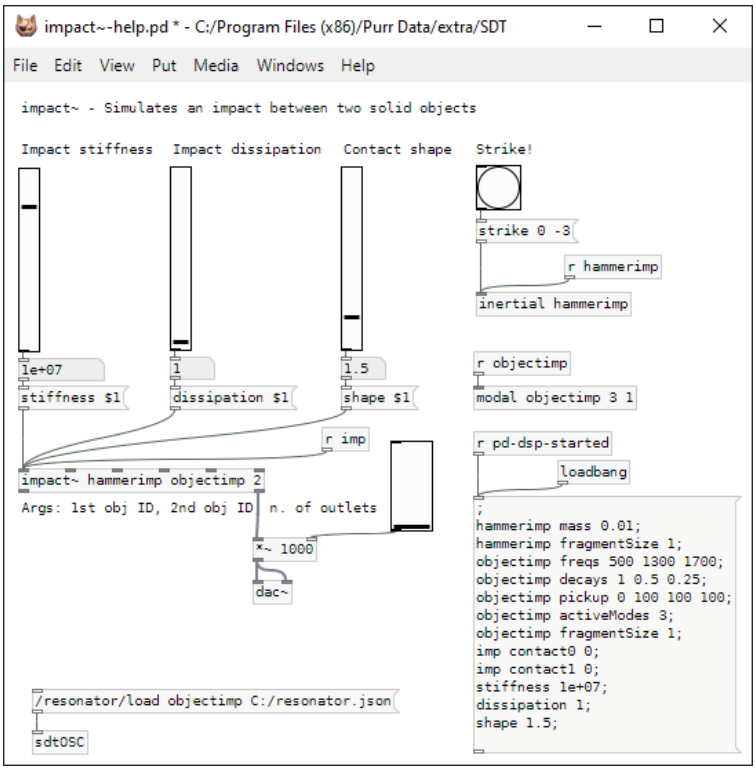

Figure 21: Example of the Pure Data help patch for `impact~` objects, with the addition of the `sdtOSC` object and an OSC message to load the parameters of a modal resonator from a JSON file.

to the name of the SDT object type (e.g., `resonator`, `friction`, `bubble`, etc.). Nested within each type container, there exists a setter method for every attribute of the corresponding object type. These setter methods take two arguments: the identifier of the target object and the value to be set. For instance, the OSC message `/myo/threshold mx 0.001` modifies the "threshold" attribute of the myoelastic feature extractor object identified by the key "mx" and sets its value to 0.001. Notably, setter methods for attributes that hold arrays of values include an additional argument to specify the index of the element being modified.

In addition to individual parameter control, all objects within the

SDT support four methods for data exchange in JSON format:

- `load` loads the object's parameters from a JSON file.
- `loads` loads the object's parameters from a JSON string.
- `save` saves the object's parameters to a JSON file.
- `log` logs the object's parameters as a JSON string.

A special container named `project` does not correspond to a specific object type. This container is designed for the management of multiple objects simultaneously. The `load` and `loads` methods of the *project* container accept the file path or JSON string to be loaded, respectively. The `save` and `log` methods take identifiers of the objects to be included in the project as arguments. When using the `save` method, the path of the destination file must also be provided.

7.2.3 Interoperability Example

The open-source SAMPLE Python package (Chapter 6) implementing the SAMPLE algorithm has been integrated into the SDT ecosystem.

The primary function of SAMPLE is the analysis of audio samples containing modal resonator impulse responses. Notably, the software's "JSON export" functionality allows users to permanently store the estimated modal parameters in a structured format: Listing 1 shows an example of such output files. It is important to acknowledge a current limitation of the analysis algorithm: at present, SAMPLE can only estimate the modal parameters for a single pickup point.

7.3 Software Re-engineering

In conjunction with the development of the previously described new functionalities, the SDT underwent significant maintenance and restructuring efforts.

A crucial improvement involved the establishment of a Continuous Integration and Continuous Deployment (CI/CD) workflow on GitHub. CI represents a software development approach where the software "is constructed in such a way as to facilitate the incorporation of new requirements and thus achieve higher adaptability" [24]. CD refers to

the fact that all of these incremental versions are automatically built and deployed. This workflow facilitates the creation of reproducible builds for all supported SDT distributions (C library, Pure Data, and Max) across various operating systems (macOS, Windows, and Linux). Notably, Max for Linux systems is excluded due to the platform's unavailability from Cycling '74. The CI/CD workflow additionally compiles the accompanying documentation using Doxygen [115].

Furthermore, a complete overhaul of the build system was undertaken. This involved updating both the Pure Data and Max SDKs, enabling support for Pure Data on 64-bit Windows operating systems and for both distributions on Apple Silicon (arm64) architectures within macOS. Consequently, support for 32-bit architectures was discontinued due to incompatibility with the new SDKs. To enhance build system organization, a single Makefile now manages all build automation processes. Prior to this change, separate Makefiles were employed, which frequently resulted in errors. Additionally, the new build automation follows an out-of-tree approach, ensuring that build artifacts are stored in separate directories from source files. Whenever feasible, external dependencies were externalized by linking them as Git submodules. This approach eliminates the need to version control large binary objects, such as the Max SDK.

A logging system was implemented within the SDT, allowing the flexible display of log messages. This system offers configuration options at three distinct levels. During compilation, all messages below a specified log level (default: `DEBUG`) are excluded from the final object code. This facilitates development with verbose builds containing messages that are not intended for the released library. In the most detailed setting (`VERBOSE`), a memory tracking system becomes active, aiding in the identification of memory leaks. The second level of configuration occurs at runtime, where the SDT checks the environment variable `SDT_LOG_LEVEL` upon loading (default: `INFO`). Messages corresponding to lower log levels are not recorded. Finally, the log functions for each level are entirely customizable. This feature is utilized within the Pure Data and Max packages to display log messages in their respective consoles, instead of the command line, which is typically hidden when running the GUI of these environments.

Comprehensively, the SDT underwent a review process to ensure proper const-correctness. Additionally, clang-format was adopted to

achieve standardized code formatting conventions. To enhance the appearance of the HTML documentation pages, doxygen-awesome was integrated.

Specific objects, such as `SDTMotor`, necessitated the use of a high-pass filter to remove DC components. The previously included filters were inadequate for this task. To address this requirement, a custom one-pole, one-zero filter was implemented as a new type, named `SDTDCFilter`.

Many externals in Max duplicated information about parameters of SDT objects, adding a copy of their values in the wrapper structures that interface SDT with the Max API. This was redundant and problematic for the adoption of OSC, since the internal values of SDT objects could be changed without notifying the wrapper structure. All wrappers have been refactored to read and write parameters directly from and to the wrapped SDT objects, ensuring synchronization between their actual values and the values seen by the Max runtime.

Initiation of a unit-test suite using CuTest marked another significant development. This suite is planned for future expansion and has already been set up in the CI/CD workflow. As a result of these efforts, the SDT is now available on Cycling '74's official Max Package Manager[1].

7.4 Conclusion

This chapter addressed the challenges of developing and maintaining the Sound Design Toolkit (SDT), with a particular focus on sharing and communicating the parameters of the sounding objects. The adoption of JSON as a representational syntax and exchange format was proposed and implemented. This approach is complemented by the integration of an Open Sound Control (OSC) server, facilitating enhanced interoperability and centralized control.

JSON offers a standardized methodology for representing comprehensive information pertaining to sounding objects. This streamlines the transition from the prototyping stage to production software and fosters efficient communication between analysis and synthesis tools.

[1] *"Announcing a new package: Sound Design Toolkit"* `https://cycling74.com/forums/announcing-a-new-package-sound-design-toolkit`

Moreover, the inherent readability of JSON [120] empowers end-users, including those without development expertise, to readily edit it in accordance with their specific requirements.

Furthermore, the integration of an OSC server significantly bolsters real-time interoperability with a multitude of software platforms. A practical use case was presented to demonstrate the effectiveness and viability of integrating JSON and OSC. This use case involved the combination of the SAMPLE Python package (Chapter 6), employed for modal analysis, with the SDT.

The SDT library itself underwent substantial improvements and restructuring efforts. A CI/CD pipeline was established on the GitHub platform, guaranteeing consistent builds across various operating systems and enabling automatic documentation generation. The build system itself was comprehensively revamped, incorporating support for 64-bit Windows and Apple Silicon architectures. Build organization was streamlined through the implementation of a single Makefile and an out-of-tree approach. External dependencies were effectively managed using Git submodules. A versatile logging system was introduced, offering three distinct levels of detail and customization options. Code quality was demonstrably enhanced through a const-correctness review process, standardized formatting practices, and improved visual elements within the documentation. Additionally, targeted improvements were made to specific objects within the library, such as the inclusion of a custom filter designed for the removal of DC components. The culmination of these efforts led to the successful inclusion of the SDT within Cycling '74's official Max Package Manager, thereby expanding its reach and accessibility to a wider user base.

Future developments could involve usability tests to quantitatively assess the impact of integrating JSON and OSC on the efficiency of the sound design workflow. In consideration of the inherent flexibility of the MIDI 2.0 standard, which permits messages containing JSON payloads [3], the exploration of incorporating support for this protocol as an alternative to OSC could be a worthwhile pursuit. Furthermore, the CuTest unit-testing suite should be expanded to cover the entirety of the SDT library.

Bibliography

[1] Jonathan S. Abel, Sean Coffin and Kyle Spratt. 'A Modal Architecture for Artificial Reverberation with Application to Room Acoustics Modeling'. In: *Audio Engineering Society Convention 137*. New York, NY, USA: Audio Engineering Society, Oct. 2014.

[2] Federico Avanzini. 'Procedural Modeling of Interactive Sound Sources in Virtual Reality'. In: *Sonic Interactions in Virtual Environments*. Ed. by Michele Geronazzo and Stefania Serafin. Cham: Springer International Publishing, 2023, pp. 49–76. ISBN: 978-3-031-04021-4. DOI: 10.1007/978-3-031-04021-4_2.

[3] Federico Avanzini, Vanessa Faschi and Luca Andrea Ludovico. 'A Web-Based MIDI 2.0 Monitor'. In: *Proceedings of the 20th Sound and Music Computing Conference*. Stockholm, Sweden, 2023, pp. 148–153. ISBN: 978-91-527-7372-7.

[4] Federico Avanzini, Matthias Rath and Davide Rocchesso. 'Physically-based Audio Rendering of contact'. In: *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME 2002)*. Vol. 2. Lausanne, Switzerland: IEEE, 2002, pp. 445–448. DOI: 10.1109/ICME.2002.1035636.

[5] Stefano Baldan, Stefano Delle Monache and Davide Rocchesso. 'The Sound Design Toolkit'. In: *SoftwareX* 6 (2017), pp. 255–260. ISSN: 2352-7110. DOI: 10.1016/j.softx.2017.06.003.

[6] Stefan Behnel et al. 'Cython: The best of both worlds'. In: *Computing in Science & Engineering* 13.2 (2011), p. 31.

[7] Alessio Benavoli et al. 'A Bayesian Wilcoxon signed-rank test based on the Dirichlet process'. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing

and Tony Jebara. Vol. 32(2). Proceedings of Machine Learning Research. Bejing, China: PMLR, June 2014, pp. 1026–1034.

[8] Alessio Benavoli et al. 'Time for a Change: a Tutorial for Comparing Multiple Classifiers Through Bayesian Analysis'. In: *Journal of Machine Learning Research* 18.77 (2017), pp. 1–36.

[9] Nicola Bernardini and Giovanni De Poli. 'The Sound and Music Computing Field: Present and Future'. In: *Journal of New Music Research* 36.3 (2007). DOI: 10.1080/09298210701862432.

[10] Knut Blind et al. *The impact of Open Source Software and Hardware on technological independence, competitiveness and innovation in the EU economy*. Final Study Report. Brussels, Belgium: Directorate-General for Communications Networks, Content and Technology (European Commission), 2021. DOI: 10.2759/430161.

[11] Boualem Boashash. 'Estimating and interpreting the instantaneous frequency of a signal. – Part 1: Fundamentals'. In: *Proceedings of the IEEE* 80.4 (1992), pp. 520–538. DOI: 10.1109/5.135376.

[12] Sebastian Bock and Gerhard Widmer. 'Maximum Filter Vibrato Suppression for Onset Detection'. In: *Proceedings of the 16th International Conference on Digital Audio Effects*. Maynooth, Ireland, 2013.

[13] Niels Bogaards, Axel Roebel and Xavier Rodet. 'Sound Analysis and Processing with AudioSculpt 2'. In: *International Computer Music Conference (ICMC)*. Miami, United States, Nov. 2004. URL: hal.science/hal-01161198.

[14] Riccardo Bona et al. 'Automatic Parameters Tuning of Late Reverberation Algorithms for Audio Augmented Reality'. In: *Proceedings of the 17th International Audio Mostly Conference*. AM '22. St. Pölten, Austria: Association for Computing Machinery, 2022, pp. 36–43. DOI: 10.1145/3561212.3561236.

[15] Niels Böttcher. 'Current problems and future possibilities of procedural audio in computer games'. In: *Journal of Gaming & Virtual Worlds* 5.3 (2013), pp. 215–234. ISSN: 1757-191X. DOI: 10.1386/jgvw.5.3.215_1.

[16] Pierre-Yves Bourguignon and Philipp Wiesner. *Pargraph – A pure Python micro-framework supporting seamless lazy and concurrent evaluation of computation graphs.* © Copyright Othoz GmbH. 2019.

[17] Manuel Briand, David Virette and Nadine Martin. 'Parametric representation of multichannel audio based on Principal Component Analysis'. In: *Proceedings of the 120th AES Convention*. Paris, France, May 2006, p. 13. URL: https://hal.science/hal-00381051.

[18] Lars Buitinck et al. 'API design for machine learning software: experiences from the scikit-learn project'. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[19] Claude Cadoz. 'Synthèse sonore par simulation de mécanismes vibratoires. Applications aux sons musicaux'. Theses. Institut national polytechnique de Grenoble, Oct. 1979. URL: https://hal.science/tel-01260499.

[20] Keunwoo Choi et al. 'Foley Sound Synthesis at the DCASE 2023 Challenge'. In: *arXiv e-prints: 2304.12521* (2023). DOI: 10.48550/arXiv.2304.12521.

[21] Giulia Clerici. 'Real-time audio synthesis based on classification of environmental sounds'. Supervisor: Stavros Ntalampiras. Master's thesis in Computer Science. University of Milan, June 2020. URL: https://unimi.primo.exlibrisgroup.com/permalink/39UMI_INST/i9q3jt/alma991017853374106031.

[22] Giulia Clerici and Marco Tiraboschi. 'Citation is not Collaboration: Music-Genre Dependence of Graph-Related Metrics in a Music Credits Network'. In: *Proceedings of the 20th Sound and Music Computing Conference*. (All authors contributed equally). 2023. DOI: 10.5281/zenodo.10061131.

[23] Thomas Cokelaer and Juergen Hasch. ''Spectrum': Spectral Analysis in Python'. In: *Journal of Open Source Software* 2.18 (2017), p. 348. DOI: 10.21105/joss.00348.

[24] A. M. Davis, E. H. Bersoff and E. R. Comer. 'A strategy for comparing alternative software development life cycle models'. In: *IEEE Transactions on Software Engineering* 14.10 (1988), pp. 1453–1461. DOI: 10.1109/32.6190.

[25] R. Deering and J.F. Kaiser. 'The use of a masking signal to improve empirical mode decomposition'. In: *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* Vol. 4. 2005, iv/485–iv/488 Vol. 4. DOI: 10.1109/ICASSP.2005.1416051.

[26] Stefano Delle Monache *et al.* and Marco Tiraboschi. *Sound Design Toolkit*. June 2022. DOI: 10.5281/zenodo.6628035.

[27] Kees van den Doel, Paul G. Kry and Dinesh K. Pai. 'FoleyAutomatic: Physically-Based Sound Effects for Interactive Simulation and Animation'. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 537–544. ISBN: 158113374X. DOI: 10.1145/383259.383322.

[28] John M. Eargle. 'Stereo/Mono Disc Compatibility: A Survey of the Problems'. In: *Audio Engineering Society Convention 35*. Oct. 1968. URL: https://www.aes.org/e-lib/browse.cfm?elib=1407.

[29] Jesse Engel et al. 'DDSP: Differentiable digital signal processing'. In: *International Conference on Learning Representations*. 2020.

[30] Marco S. Fabus et al. 'Automatic decomposition of electrophysiological data into distinct nonsinusoidal oscillatory modes'. In: *Journal of Neurophysiology* 126.5 (2021), pp. 1670–1684. DOI: 10.1152/jn.00315.2021.

[31] Andy Farnell. *Designing sound*. Mit Press, 2010.

[32] Michael Feldman. 'Analytical basics of the EMD: Two harmonics decomposition'. In: *Mechanical Systems and Signal Processing* 23.7 (2009), pp. 2059–2071. ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2009.04.002.

[33] Richard P. Feynman, Robert B. Leighton and Matthew Sands. 'Beats'. In: *The Feynman Lectures on Physics, Vol. I: Mainly Mechanics, Radiation and Heat*. New York, NY, USA: Basic Books, 1963. Chap. 48, pp. 48-1 –48-18.

[34] Julien Flamant, Nicolas Le Bihan and Pierre Chainais. 'Spectral Analysis of Stationary Random Bivariate Signals'. In: *IEEE Transactions on Signal Processing* 65.23 (2017), pp. 6135–6145. DOI: 10.1109/TSP.2017.2736494.

[35] Jean-Loup Florens and Claude Cadoz. 'The physical model: modeling and simulating the instrumental universe'. In: *Representations of Musical Signals*. Cambridge, MA, USA: MIT Press, 1991. Chap. 7, pp. 227–268. ISBN: 0262041138.

[36] D. Gabor. 'Theory of communication. Part 1: The analysis of information'. In: *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering* 93 (26 Nov. 1946), pp. 429–441. ISSN: 2054-0604. DOI: 10.1049/ji-3-2.1946.0074.

[37] M. Gasior and J. L. Gonzalez. 'Improving FFT Frequency Measurement Resolution by Parabolic and Gaussian Spectrum Interpolation'. In: *AIP Conference Proceedings* 732.1 (Nov. 2004), pp. 276–285. ISSN: 0094-243X. DOI: 10.1063/1.1831158.

[38] William W. Gaver. 'How Do We Hear in the World? Explorations in Ecological Acoustics'. In: *Ecological Psychology* 5.4 (1993), pp. 285–313. DOI: 10.1207/s15326969eco0504_2.

[39] William W. Gaver. 'What in the World Do We Hear?: An Ecological Approach to Auditory Event Perception'. In: *Ecological Psychology* 5.1 (1993). DOI: 10.1207/s15326969eco0501_1.

[40] Arthur S. Goldberger. 'Classical linear regression'. In: *Econometric theory*. New York, NY, USA: Wiley, 1964. Chap. 4, pp. 156–212. DOI: 10.1002/nav.3800110213.

[41] Roman Haefeli. 'netpd - A Collaborative Realtime Networked Music Making Environment written in Pure Data'. In: *Linux Audio Conference 2013*. Vol. 1. Institute of Electronic Music, Acoustics. University for Music and Performing Arts Graz, Austria. 2013.

[42] Paul Hamill. *Unit test frameworks: tools for high-quality software development.* "O'Reilly Media, Inc.", 2004.

[43] B. V. Hamon and E. J. Hannan. 'Spectral Estimation of Time Delay for Dispersive and Non-Dispersive Systems'. In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 23.2 (Dec. 2018), pp. 134–142. ISSN: 0035-9254. DOI: 10.2307/2346994.

[44] Tim Head et al. *Scikit-Optimize.* Version v0.9.0. Oct. 2021. DOI: 10.5281/zenodo.5565057.

[45] W. Heisenberg. 'Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik'. In: *Zeitschrift für Physik* 43.3 (Mar. 1927), pp. 172–198. ISSN: 0044-3328. DOI: 10.1007/BF01397280.

[46] Steffen Herbold. 'Autorank: A Python package for automated ranking of classifiers'. In: *Journal of Open Source Software* 5.48 (2020), p. 2173. DOI: 10.21105/joss.02173.

[47] Olivier Houix et al. 'A lexical analysis of environmental sound categories.' In: *Journal of Experimental Psychology: Applied* 18.1 (2012), pp. 52–80. DOI: 10.1037/a0026240.

[48] Norden E. Huang et al. 'The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis'. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 454 (1998), pp. 903–995. DOI: 10.1098/rspa.1998.0193.

[49] Norden E. Huang et al. 'On Holo-Hilbert spectral analysis: a full informational spectral representation for nonlinear and non-stationary data'. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2016). DOI: 10.1098/rsta.2015.0206.

[50] S Rao Jammalamadaka and A Sengupta. *Topics in circular statistics.* Vol. 5. World Scientific, 2001.

[51] Matti Karjalainen et al. 'Estimation of Modal Decay Parameters from Noisy Response Measurements'. In: *Proceedings of the AES 110th Convention.* Audio Engineering Society, 2002, pp. 867–878.

[52] Steven M. Kay. *Modern Spectral Estimation: Theory and Application.* Prentice-Hall, NJ, USA, 1988.

[53] Corey Kereliuk et al. 'Modal analysis of room impulse responses using subband esprit'. In: *Proceedings of the International Conference on Digital Audio Effects*. 2018.

[54] Xun Lang et al. 'Median ensemble empirical mode decomposition'. In: *Signal Processing* 176 (2020). ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2020.107686.

[55] Nicholas R. Lomb. 'Least-squares frequency analysis of unequally spaced data'. In: *Astrophysics and Space Science* 39.2 (Feb. 1976), pp. 447–462. ISSN: 1572-946X. DOI: 10.1007/BF00648343.

[56] Richard G. Lyons. 'Understanding Digital Signal Processing'. In: Third. Pearson, 2011. Chap. 13.10. ISBN: 978-0137-02741-5.

[57] E. Maestre, G. P. Scavone and J. O. Smith. 'Joint Modeling of Bridge Admittance and Body Radiativity for Efficient Synthesis of String Instrument Sound by Digital Waveguides'. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 25.5 (May 2017), pp. 1128–1139. ISSN: 2329-9290. DOI: 10.1109/TASLP.2017.2689241.

[58] S. Lawrence Jr. Marple. *Digital Spectral Analysis*. Englewood Cliffs, NJ: Prentice Hall, 1987.

[59] James McLaughlin *et al.* and Marco Tiraboschi. *json-builder – The serializing counterpart to json-parser*. URL: https://github.com/json-parser/json-builder.

[60] James McLaughlin *et al.* and Marco Tiraboschi. *json-parser – Very low footprint DOM-style JSON parser written in portable ANSI C*. URL: https://github.com/json-parser/json-parser.

[61] Romain Michon, Sara R Martin and Julius O. Smith. 'MESH2-FAUST: a Modal Physical Model Generator for the Faust Programming Language - Application to Bell Modeling'. In: *Proceedings of the 2017 International Computer Music Conference, ICMC*. Ann Arbor, MI, USA: Michigan Publishing, 2017, pp. 99–103. DOI: 2027/spo.bbp2372.2017.014.

[62] Stefano Delle Monache, Pietro Polotti and Davide Rocchesso. 'A Toolkit for Explorations in Sonic Interaction Design'. In: *Proceedings of the 5th Audio Mostly Conference: A Conference on Interaction with Sound*. AM '10. Piteå, Sweden: Association for Computing Machinery, 2010, pp. 1–7. ISBN: 9781450300469. DOI: 10.1145/1859799.1859800.

[63] James A. Moorer. 'Signal Processing Aspects of Computer Music — A Survey'. In: *CMJ* 1.1 (1977), pp. 4–37.

[64] Alan V Oppenheim and Ronald W Schafer. 'All-Pole Modeling of Sinusoidal Signals'. In: *Discrete-Time Signal Processing*. Third. Prentice Hall, 2010. Chap. 11.5.3, pp. 913–915.

[65] Yann Orlarey, Dominique Fober and Stéphane Letz. 'FAUST: an Efficient Functional Approach to DSP Programming'. In: *New Computational Paradigms for Computer Music*. Ed. by Editions Delatour France. 2009, pp. 65–96. URL: https://hal.science/hal-02159014.

[66] Karl Pearson. 'On lines and planes of closest fit to systems of points in space'. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720.

[67] F. Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[68] Cécile Picard et al. 'A Robust and Multi-Scale Modal Analysis For Sound Synthesis'. In: *Proceedings of the 12th International Conference on Digital Audio Effects*. Como, Italy: DAFx, 2009, pp. 1–7.

[69] Giorgio Presti. 'The Bivariate Mixture Space: A Compact Spectral Representation of Bivariate Signals'. In: *J. Audio Eng. Soc* 71.7/8 (2023), pp. 481–491. DOI: 10.17743/jaes.2022.0090.

[70] John G Proakis and Dimitris G Manolakis, eds. *Digital Signal Processing: Principles, Algorithms, and Applications*. 5th ed. Pearson, July 2021.

[71] Side Effects Project. *Sing With Me*. As featured in: *Sound on Sound*. 2010. URL: https://cambridge-mt.com/ms/mtk/#SideEffectsProject.

[72] Miller Puckette. 'Combining event and signal processing in the MAX graphical programming environment'. In: *Computer Music Journal* 15.3 (1991), pp. 68–77. ISSN: 01489267, 15315169. DOI: 10.2307/3680767.

[73] Miller Puckette. 'Pure data'. In: *Proceedings of the 1997 International Computer Music Conference*. Thessaloniki, Greece: Michigan Publishing, 1997. URL: http://quod.lib.umich.edu/i/icmc/bbp2372.1997.

[74] Heiko Purnhagen and Nikolaus Meine. 'HILN - The MPEG-4 parametric audio coding tools'. In: *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 3. IEEE. 2000, pp. 201–204.

[75] Andrew J. Quinn et al. 'EMD: Empirical Mode Decomposition and Hilbert-Huang Spectral Analyses in Python'. In: *Journal of Open Source Software* 6.59 (2021), p. 2977. DOI: 10.21105/joss.02977.

[76] L. Rabiner et al. 'A comparative performance study of several pitch detection algorithms'. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24.5 (1976), pp. 399–418. DOI: 10.1109/TASSP.1976.1162846.

[77] ITUR Recommendation. '1534-1, "Method for the Subjective Assessment of Intermediate Sound Quality (MUSHRA)"'. In: *International Telecommunications Union, Geneva, Switzerland* (2001).

[78] RedFantom et al. *Ttkthemes - A group of themes for the ttk extensions of Tcl*. Version 3.2.2. URL: https://pypi.org/project/ttkthemes.

[79] Zhimin Ren, Hengchin Yeh and Ming C. Lin. 'Example-Guided Physically Based Modal Sound Synthesis'. In: *ACM Transactions on Graphics (TOG)* 32.1 (Feb. 2013). ISSN: 0730-0301. DOI: 10.1145/2421636.2421637.

[80] Davide Rocchesso. 'Sounding objects in Europe'. In: *The New Soundtrack* 4.2 (2014), pp. 157–164. DOI: 10.3366/sound.2014.0060.

[81] Kai Rothaus, Xiaoyi Jiang and Martin Lambers. 'Comparison of Methods for Hyperspherical Data Averaging and Parameter Estimation'. In: *18th International Conference on Pattern Recognition (ICPR'06)*. Vol. 3. 2006, pp. 395–399. DOI: 10.1109/ICPR.2006.391.

[82] S. Roucos and A. Wilgus. 'High quality time-scale modification for speech'. In: *ICASSP '85. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 10. 1985, pp. 493–496. DOI: 10.1109/ICASSP.1985.1168381.

[83] Richard Roy and Thomas Kailath. 'ESPRIT - Estimation of Signal Parameters via Rotational Invariance Techniques'. In: *IEEE Transactions on acoustics, speech, and signal processing* (1989), pp. 984–995.

[84] Jeffrey D. Scargle. 'Studies in astronomical time series analysis. II. Statistical aspects of spectral analysis of unevenly spaced data.' In: *Astrophysical Journal* 263 (Dec. 1982). DOI: 10.1086/160554.

[85] Michael Schoeffler et al. 'webMUSHRA — A Comprehensive Framework for Web-based Listening Tests'. In: *Journal of Open Research Software* (Feb. 2018). DOI: 10.5334/jors.187.

[86] Timothy Schoen. *Plug Data*. 2023. URL: https://plugdata.org.

[87] Xavier Serra. 'A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition'. PhD thesis. CCRMA Department of Music, Stanford University, 1989.

[88] Xavier Serra. 'Musical sound modeling with sinusoids plus noise'. In: *Musical signal processing*. Lisse, Netherlands: Swets & Zeitlinger Publishers, 1997. Chap. 3, pp. 91–122. DOI: 10.4324/9781315078120.

[89] Xavier Serra and Julius O. Smith. 'Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition'. In: *Computer Music Journal* (1990), pp. 12–24. DOI: 10.2307/3680788.

[90] Samuel Sanford Shapiro and Martin B Wilk. 'An analysis of variance test for normality (complete samples)'. In: *Biometrika* 52.3/4 (1965), pp. 591–611.

[91] Julius O. Smith. 'Virtual Acoustic Musical Instruments: Review and Update'. In: *Journal of New Music Research* 33.3 (2004). DOI: 10.1080/0929821042000317859.

[92] Julius O. Smith. 'Analytic Signals and Hilbert Transform Filters'. In: *Mathematics of the Discrete Fourier Transform (DFT) with Audio Applications*. Second. W3K Publishing, 2007. ISBN: 978-0-9745607-4-8. URL: https://ccrma.stanford.edu/~jos/st/Analytic_Signals_Hilbert_Transform.html.

[93] Julius O. Smith. *Physical audio signal processing: for virtual musical instruments and audio effects*. W3K Publishing, 2010. URL: https://ccrma.stanford.edu/~jos/pasp.

[94] Christian J. Steinmetz and Joshua D. Reiss. 'auraloss: Audio focused loss functions in PyTorch'. In: *Digital Music Research Network One-day Workshop (DMRN+15)*. London, UK, Dec. 2020, pp. 14–14. URL: https://www.qmul.ac.uk/dmrn/media/dmrn/DMRN-15_proceedings.pdf.

[95] Auston Sterling et al. 'Audio-Material Reconstruction for Virtualized Reality Using a Probabilistic Damping Model'. In: *IEEE Transactions on Visualization and Computer Graphics* 25.5 (2019), pp. 1855–1864. DOI: 10.1109/TVCG.2019.2898822.

[96] S. S. Stevens, J. Volkmann and E. B. Newman. 'A Scale for the Measurement of the Psychological Magnitude Pitch'. In: *The Journal of the Acoustical Society of America* 8.3 (1937), pp. 185–190. DOI: 10.1121/1.1915893.

[97] Gilbert Strang. *Linear algebra and its applications*. Brooks/Cole Pub Co, 2012.

[98] Adam M. Sykulski et al. 'Frequency-Domain Stochastic Modeling of Stationary Bivariate or Complex-Valued Signals'. In: *IEEE Transactions on Signal Processing* 65.12 (2017), pp. 3136–3151. DOI: 10.1109/TSP.2017.2686334.

[99] Marco Tiraboschi. *SAMPLE – Python package*. LIM, University of Milan, 2021. DOI: 10.5281/zenodo.6536419.

[100] Marco Tiraboschi. *Chapter 4 – Experimental Data*. 2024. DOI: 10.5281/zenodo.10684364.

[101] Marco Tiraboschi and Federico Avanzini. 'SAMPLE: a Python Package for the Spectral Analysis of Modal Sounds'. In: *Conference Proceedings of the 23rd Colloquium on Music Informatics*. "Aldo Piccialli" Award for the best contribution to the Scientific Programme. 2022, pp. 50–55. URL: https://hdl.handle.net/2434/945288.

[102] Marco Tiraboschi and Federico Avanzini. 'Acoustic Beats and Where To Find Them: Uneven Beats Models and Modal Audio Inversion by Non-linear Least-Squares Optimization'. In: *Inverse Problems* (2024). [UNDER REVIEW].

[103] Marco Tiraboschi, Federico Avanzini and Giuseppe Boccignone. 'Listen to your Mind's (He)Art: A System for Affective Music Generation via Brain-Computer Interface'. In: *Proceedings of the 18th Sound and Music Computing Conference*. 2021. DOI: 10.5281/zenodo.5044984.

[104] Marco Tiraboschi, Federico Avanzini and Stavros Ntalampiras. 'Spectral Analysis for Modal Parameters Linear Estimate'. In: *Proceedings of the 17th Sound and Music Computing Conference*. Ed. by Simone Spagnol and Andrea Valle. SMC. Sound and Music Computing Network. Torino, Italy: Axea sas/SMC Network, June 2020, pp. 276–283. DOI: 10.5281/zenodo.3898795.

[105] Marco Tiraboschi and Giulia Clerici. *Chapter 3 – Evaluation Data*. 2024. DOI: 10.5281/zenodo.12179981.

[106] Marco Tiraboschi, Stefano Papetti and Federico Avanzini. 'Just a Sounding Object Notation: Sharing Objects for Sonic Interaction Design with JSON and OSC'. In: *Proceedings of the 4th*

International Symposium on the Internet of Sounds. 2023. DOI: 10.1109/IEEECONF59510.2023.10335232.

[107] Marco Tiraboschi and Giorgio Presti. 'The Generalized Mixture Space: extending Compact Spectral Representations to Multichannel Signals'. In: *Journal of the Audio Engineering Society* (2024). [TO BE SUBMITTED].

[108] María E. Torres et al. 'A complete ensemble empirical mode decomposition with adaptive noise'. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2011, pp. 4144–4147. DOI: 10.1109/ICASSP.2011.5947265.

[109] Hartmut Traunmüller. 'Analytical expressions for the tonotopic sensory scale'. In: *The Journal of the Acoustical Society of America* 88.1 (1990), pp. 97–100. DOI: 10.1121/1.399849.

[110] Bill Triggs et al. 'Bundle Adjustment - A Modern Synthesis'. In: *Vision Algorithms: Theory and Practice*. Ed. by Bill Triggs, Andrew Zisserman and Richard Szeliski. Heidelberg, Germany: Springer, 2000, pp. 298–372. ISBN: 978-3-540-44480-0. DOI: 10.1007/3-540-44480-7_21.

[111] Nikolaos Tsakalozos, Konstantinos Drakakis and Scott Rickard. 'A formal study of the nonlinearity and consistency of the Empirical Mode Decomposition'. In: *Signal Processing* 92.9 (2012), pp. 1961–1969. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2011.09.014.

[112] Luca Turchet et al. 'The Internet of Sounds: Convergent Trends, Insights, and Future Directions'. In: *IEEE Internet of Things Journal* 10.13 (2023), pp. 11264–11292. DOI: 10.1109/JIOT.2023.3253602.

[113] Adam Turner et al. *Sphinx documentation*. 2007. URL: https://www.sphinx-doc.org.

[114] Vesa Välimäki et al. 'Discrete-time modelling of musical instruments'. In: *Reports on Progress in Physics* 69.1 (Oct. 2005), p. 1. DOI: 10.1088/0034-4885/69/1/R01.

[115] Dimitri van Heesch. *doxygen – Code Documentation. Automated.* 1997. URL: https://www.doxygen.nl.

[116] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[117] C. Voglis and I.E. Lagaris. 'A rectangular trust region dogleg approach for unconstrained and bound constrained nonlinear optimization'. In: *WSEAS International Conference on Applied Mathematics*. Vol. 7. Boca Raton, FL, USA: CRC Press, 2004, p. 4.

[118] Marcus Von Appen et al. *Pygame - A Free and Open Source python programming language library for making multimedia applications*. Version 2.1.2. URL: https://pypi.org/project/pygame.

[119] DeLiang Wang and Guy J Brown. *Computational auditory scene analysis: Principles, algorithms, and applications*. Wiley - IEEE press, 2006.

[120] Philipp Wehner, Christina Piberger and Diana Göhringer. 'Using JSON to manage communication between services in the Internet of Things'. In: *2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*. 2014, pp. 1–4. DOI: 10.1109/ReCoSoC.2014.6861361. URL: https://doi.org/10.1109/ReCoSoC.2014.6861361.

[121] Matthew Wright and Adrian Freed. 'Open sound control: A new protocol for communicating with sound synthesizers'. In: *Proceedings of the 23rd International Computer Music Conference*. Thessaloniki, Greece: Michigan Publishing, 1997. URL: http://quod.lib.umich.edu/i/icmc/bbp2372.1997.

Project developed at the Laboratorio di Informatica Musicale (LIM)
https://www.lim.di.unimi.it