UNIVERSIDADE FEDERAL FLUMINENSE

ESCOLA DE ENGENHARIA

PROGRAMA DE PÓS-GRADUAÇÃO STRICTO SENSU EM ENGENHARIA DE PRODUÇÃO

CURSO DE DOUTORADO EM ENGENHARIA DE PRODUÇÃO

CARLOS HENRIQUE TARJANO SANTOS

A GENERAL ALGORITHM FOR THE REAL-TIME EMULATION OF PITCHED MUSICAL INSTRUMENTS AND THE SINGING VOICE

Dissertação apresentada ao Programa de Pós-Graduação Stricto Sensu em Engenharia de Produção da Universidade Federal Fluminense como requisito parcial para obtenção do Grau de Doutor em Engenharia de Produção.

Professor Orientador:

Valdecy Pereira, D.Sc.

NITERÓI

2022

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

T176g Tarjano santos, Carlos Henrique
A general algorithm for the real-time emulation of pitched musical instruments and the singing voice / Carlos Henrique Tarjano santos ; Valdecy Pereira, orientador. Niterói, 2022.
181 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2022.

DOI: http://dx.doi.org/10.22409/TPP.2022.d.11839251743

1. Rede neural artificial. 2. Inteligência artificial. 3. Aprendizado de máquina. 4. Processamento de sinais. 5. Produção intelectual. I. Pereira, Valdecy, orientador. II. Universidade Federal Fluminense. Escola de Engenharia. III. Título.

CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

CARLOS HENRIQUE TARJANO SANTOS

A GENERAL ALGORITHM FOR THE REAL-TIME EMULATION OF PITCHED MUSICAL INSTRUMENTS AND THE SINGING VOICE

Tese apresentada ao Curso de Doutorado em Engenharia de Produção da Universidade Federal Fluminense como requisito parcial para obtenção do Grau de Doutor em Engenharia de Produção.

Aprovada em 28 de julho de 2022

BANCA EXAMINADORA

Professor Orientador: Valdecy Pereira, D.Sc.

Universidade Federal Fluminense

Artur Alves Pessoa, D.Sc.

Universidade Federal Fluminense

Guilherme Lucio Abelha Mota, D.Sc.

Universidade do Estado do Rio de Janeiro

Miguel Arjona Ramírez, D.Sc.

Universidade de São Paulo

Vitor Acioly Barbosa, D.Sc.

Universidade Federal Fluminense

Dedico este trabalho à Maria e Nelson, meus progenitores, e ao meu tio-avô e irmão, Antonio. Que um dia eu venha a ser inteligente como eles achavam que eu fosse. Também à Samara: obrigado por caminhar comigo, já há tanto tempo, o caminho perigoso que é a vida. E à minha família: minha avó Helena, minha madrinha Maria e minha prima Larissa.

AGRADECIMENTOS

Agradeço à Universidade Federal do Rio de Janeiro, onde trabalho, e especialmente a seu corpo técnico-administrativo, por ter possibilitado dedicar-me aos meus arroubos intelectuais.

Ao professor Valdecy Pereira, meu orientador, por ter proporcionado o ambiente ideal para o desenvolvimento de minha pesquisa, tanto durante o mestrado quanto agora, no doutorado.

Ao professor Guilherme Lucio Abelha Mota, pela disciplina de Visão Computacional ministrada em 2019 na Universidade Estadual do Rio de Janeiro, uma das disciplinas mais bem estruturadas que tive o prazer de cursar, e que me proporcionou valiosa inspiração.

Ao professor Eduardo Miranda, por ter me acolhido em Plymouth, onde pude expandir meus horizontes sobre as possibilidades, teóricas e práticas, da minha área de pesquisa.

Ao doutor Maurício do Vale Madeira da Costa e ao professor Miguel Arjona Ramírez, pela leitura atenta da tese, e apontamentos importantes sobre termos e conceitos da área de DSP.

Ao professor Artur Alves Pessoa, pelas disciplinas estimulantes ministradas durante o Mestrado e Doutorado, e por ter me inspirado a investigar, além de outras coisas, a linguagem C++.

Ao professor Vitor Acioly Barbosa, amigo de longa data, por ter aceito o desafio de avaliar um trabalho fora de sua zona de conforto, em um momento em que estava bastante atarefado, e pelas valiosas contribuições oferecidas.

Problemas há, Liberális excelente, cuja pesquisa vale só pelo intelectual exercício, e que ficam sempre fora da vida; outros investigam-se com prazer e com proveito se resolvem.

Guimarães Rosa

Resumo

Instrumentos musicais digitais baseados em amostras representam o atual estado da arte da emulação de instrumentos musicais em tempo real. Embora os melhores instrumentos digitais baseados em amostras apresentem boa qualidade de som, eles apresentam várias desvantagens, como a falta de flexibilidade e as enormes bibliotecas de sons pré-gravados que demandam grandes espaços para armazenamento de arquivos digitais, por exemplo. Nos últimos anos, as abordagens baseadas em inteligência artificial vêm ganhando popularidade. Embora a qualidade e a eficiência dos modelos atuais estejam melhorando constantemente, eles tendem a consumir muitos recursos computacionais e atualmente não são capazes de competir, em termos de qualidade sonora, com instrumentos digitais baseados em amostras. No cerne deste problema está a falta de uma representação apropriada para sinais discretos com alto grau de periodicidade, formulada para aproveitar os resultados que as redes neurais estão demonstrando em áreas como processamento de linguagem natural e visão computacional, onde constituem o estado da arte. Este trabalho apresenta, portanto, tal representação e, a partir dela, desenvolve um conjunto de instrumentos musicais digitais capazes de emular instrumentos reais, além da voz cantada, em tempo real, com requisitos modestos de armazenamento e poder de processamento. Para tanto, é realizado um levantamento da literatura relacionada, especialmente no que diz respeito à área de processamento digital de sinais, e preenche as lacunas que impedem o desenvolvimento de tal representação. Especificamente, são introduzidos novos algoritmos de detecção de envelopes e de segmentação de sinais discretos, desenhado para identificar os pseudo ciclos individuais de sinais semi periódicos. Esses avanços teóricos são empregados na implementação de um framework para a emulação de instrumentos musicais em geral, onde os instrumentos digitais são treinados usando amostras pré-gravadas de instrumentos reais. A qualidade das amostras disponíveis gratuitamente para treinar o algoritmo é uma das limitações do presente trabalho. O trabalho também exemplifica como algoritmos baseados em redes neurais podem ser melhor integrados a áreas tradicionais relacionadas à síntese sonora, e como podem motivar avanços práticos e teóricos nessas áreas.

Palavras-chave: Síntese sonora em tempo real, processamento digital de sinais, redes neurais, representação discreta de sinais

Abstract

Sample-based digital musical instruments currently represent the state of the art in real-world instrument emulation. While the best sample-based digital instruments present good sound quality, they have several drawbacks, such as the lack of flexibility, and the huge libraries and storage requirements involved, to name a few. In the last few years, artificial intelligence-based approaches are gaining popularity. While the quality and efficiency of the most recent such models are constantly improving, they tend to be resource-intensive, and are not currently able to compete, quality-wise, with sample-based real-time instruments. At the heart of this problem lies the lack of an appropriate representation for quasi-periodic discrete signals, formulated to take advantage of the capabilities neural networks are demonstrating in areas such as natural language processing and computer vision, where they constitute the state of the art. This work introduces, therefore, such a representation, and develops a set of digital musical instruments capable of emulating real-world instruments and the singing voice in real-time, with modest storage and processing requirements. To do so, this work surveys the related literature, especially concerning digital signal processing, and fills the gaps that hinder the development of such a representation. Specifically, a novel envelope detection algorithm and a discrete signal segmentation algorithm, tailored to identify the individual pseudo cycles of quasi-periodic signals, are introduced. The theoretical advancements are employed in the implementation of a general algorithm for the emulation of real-world instruments, that are trained using samples from real-world instruments and the singing voice. The quality of the freely available samples used to train the algorithm is one of the limitations of the present work. The work also exemplifies how neural networks-based algorithms can be more integrated with traditional areas related to sound synthesis, and how they can motivate practical and theoretical advancements in those areas.

Keywords: Real-time sound synthesis, Digital Signal Processing, Neural Networks, Discrete Signal Representation

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAC Advanced Audio Coding. 128, 130, 132, 137

AAE average absolute error. 113, 114, 126, 134

AAX Avid Audio eXtension. 92

AI artificial intelligence. 14, 18, 19, 23, 24, 40–42, 142, 143

AIFF Audio Interchange File Format. 110

AM amplitude modulation. 55, 58

AU Audio Units. 92

CNN Convolutional Neural Network. 37, 128, 142

CSS Concatenative Sound Synthesis. 37

DAW digital audio workstation. 23, 91, 92, 94

DFT discrete Fourier transform. 28, 39, 51, 52, 69, 76, 85, 86, 122

DSP digital signal processing. 14–20, 23, 25, 29, 31, 32, 36, 46, 142, 144–146

DTFT discrete-time Fourier transform. 51

EDVAC Electronic Discrete Variable Automatic Computer. 27

ENIAC Electronic Numeric Integrator and Calculator. 25, 27, 40

FFT fast Fourier transform. 28, 39, 90, 102

FM frequency modulation. 56, 58

GAN Generative Adversarial Network. 42

GUI Graphical user interface. 92, 94, 98

HC Harmonic Compression. 79, 129–132, 135, 137

HILN Harmonic and Individual Lines plus Noise. 131

HVXC Harmonic Vector Excitation Coding. 131

IAS Institute for Advanced Study machine. 25

ICMAI 02 2nd International Conference on Music and Artificial Intelligence. 41

IDFT inverse discrete Fourier transform. 52, 86

IETF Internet Engineering Task Force. 130

ITU-R International Telecommunication Union's Radiocommunication. 135

kbit/s kilobits per second. 130, 132

MIDI Musical Instrument Digital Interface. 19, 43, 44, 91–94, 99, 102, 141

MIS University of Iowa Musical Instrument Samples. 110, 130

MMA MIDI Manufacturers Association. 44

MOS-LQO Mean Opinion Score — Listening Quality Objective. 136

MP3 MPEG-2 Audio Layer III. 110, 128, 130–133, 137

MSE mean squared error. 112–114, 134, 138–140

MUSHRA Multiple Stimuli with Hidden Reference and Anchor. 135

ONNX Open Neural Network Exchange. 93

PEAQ Perceptual Evaluation of Audio Quality. 136

PESQ Perceptual Evaluation of Speech Quality. 135

POLQA Perceptual Objective Listening Quality Assessment. 135, 136

PSOLA pitch-synchronous overlap-add. 38

PyPI Python Package Index. 55, 70

RNN Recurrent Neural Network. 37, 41, 101

RTAS Real-Time AudioSuite. 92

SDFT sliding discrete Fourier transform. 85

SDK Software development kit. 92

SGD Stochastic Gradient Descent. 105

SMS Spectral Modeling Synthesis. 40

STC Sinusoidal Transform Coding. 40

ViSQOL Virtual Speech Quality Objective Listener. 136

VoIP Voice over Internet Protocol. 136

VST virtual studio technology. 23, 92

WAV Waveform Audio File. 90, 110

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

The technological advances that took place in the XX century — phonography, electrification, and, more recently, digitalization — altered music and musical instruments profoundly (Bovermann et al., 2017). Not only the physical limitations of classical instruments were overcome by the introduction of electric or digital counterparts, but a landscape of new instruments and sounds came to be.

New technologies introduced in society in general often have an impact on the evolution of musical instruments, as luthiers and musicians incorporate the new technologies and knowledge in their craft. Following this trend, it is not surprising that the contemporary general musical landscape, including recording, production, and distribution, is largely digital, reflecting the omnipresence of digital technology in modern society (Tahiroğlu et al., 2021).

Following this trend, the recent popularity surge of neural networks naturally inspired a myriad of music-related applications, in areas such as sound synthesis, music classification, and algorithmic composition, to name a few.

A casual observer of the recent advancements in neural networks applied to audio processing would be led to believe that a strong synergy exists between both areas. Despite a profusion of audio-related works, the techniques employed suggested a disconnection with established digital signal processing (DSP) practices. While this trend in itself is not necessarily a problem, further analysis suggests there is room for improvement, mainly by the introduction of a better, more compact, neural-networks-friendly signal representation.

The lack of such an appropriate representation partly explains why, despite considerable advances in related areas such as computer vision, the field of machine learning is lagging in relation to sound synthesis and other audio-related tasks.

The main objective of this work is, then, to fill this gap at the intersection between the DSP and artificial intelligence (AI) theories, where a lack of an appropriate description of quasi-periodic discrete signals hinders the efficacy and efficiency of some applications, by proposing a novel, compact signal representation, and illustrates its effectiveness with an implementation of a digital instrument capable of emulating virtually any real-world pitched instrument, and the singing voice.

Ambitious as it may seem, it is also a tacit goal of this dissertation to motivate a change of approach in domains that, despite their modest intersections, share more attributes than is apparent at a superficial glance: both machine learning and digital

musical acoustics are areas that, despite their current relevance, are relatively new when compared to more established fields such as statistics, having their origin at around the 1950s, and a scrutiny of both promptly reveals the difficulties that arise from this state of affairs. A lack of their own terminology, with frequent borrows from other, more established areas, can be readily cited, as the pronounced prominence of few individual contributions.

Those shortcomings have, however, a bright side to them, in that they still expose those areas to reinterpretation, rendering them somewhat open to the influence of outside ideas.

The framework introduced in this work helps to illustrate the possibilities of this synergetic approach, exemplifying how such integration can be done, via the development of a new sound representation designed specifically to be used by neural networks in real-time sound synthesis tasks.

The embryo of the methodology underlying the present work was the realization, still during the development of my master's degree thesis (Tarjano, 2018), that even a heavily simplified description of characteristics of digital signals had the potential to be the basis for an elegant approach to neural networks-based sound synthesis.

To contextualize, in that work I was able to generate realistic, albeit in a limited way, acoustic instrument sounds using neural networks and a very crude model of the envelope of the underlying signals, one that only considered exponential decay. This led to severe limitations on the nature of the instruments that could be faithfully emulated, restricting its usability to those instruments exhibiting a prominent percussive nature in their modes of excitation. This included relevant instruments, such as pianos, acoustic stringed instruments, and drums, but excluded other important categories, such as the singing voice, wind instruments, and electric stringed instruments in general.

Despite those limitations, the work established the potential for improvement in neural-networks-based sound synthesis, as long as a fitting representation of the underlying signals became available, besides providing the experience of not only what was achievable in the following four years of Ph.D. work but, perhaps more importantly, how much experimentation and exploration could be done without compromising the established deadline.

This experience was crucial in the formulation of a research project that allowed focusing on two areas that were not adequately addressed in the DSP literature, and exploring unorthodox approaches to solve them.

The envelope detection problem was addressed first, as reported in Tarjano et al. (2022b). This paper forms the basis of Sections 3.3 and 5.1, and sets the general tone of this work: a pragmatic investigation, with a clear objective, that nevertheless allows itself to wander into ancillary topics, whenever those diversions have the potential to prove themselves relevant to future advancements in the area of DSP in general. This approach is reflected in the presence of unorthodox themes, such as discrete curvature estimation, and the sketch of a mathematically sound definition of an envelope, with some accompanying proofs.

The usefulness of the envelope detection algorithm introduced in the article motivated the thorough investigation of a novel segmentation algorithm as a means of addressing the second gap in the literature. The results of this investigation were first presented in Tarjano et al. (2022a), and can be seen, in the context of the present work, in Sections 3.3.2 and 3.4.

The main idea behind the algorithm is to segment a quasi-periodic signal into its smallest meaningful parts, dubbed pseudo cycles. Keeping the same exploratory ethos, a preliminary investigation of the use of the proposed theory in lossy sound compression is also presented, for example, in Section 5.2.1.

In order to maintain its general coherence, the research had, from the very beginning, a very clear goal of designing a general approach for the emulation of real-world acoustic musical instruments based on a compact but complete digital signal representation using those algorithms: compact in the sense that most redundancy is avoided in the proposed representation and complete since it must be possible to reconstruct the original signal from this representation.

1.1 GENERAL OBJECTIVE

This work has as its primary objective the development and implementation of a neural network-based real-time sound synthesis plugin, general enough to enable the realistic emulation of virtually any pitched instrument. After surveying the available relevant literature, this goal can be unfolded in the hierarchical steps described in the next section.

1.2.1 Formulation of a neural-networks-friendly representation

A neural-networks-friendly representation can be described as one that not only encodes, as succinctly as possible, all the information necessary to reconstruct a signal, but one that must be also efficiently learned by a neural network. Developing such a representation is the most general of the specific objectives of this work. Upon surveying the relevant DSP literature, it becomes clear that the existing alternative representations of discrete signals are tailored to specific applications, most of them with classification purposes. Hence, those representations tend to favor simplicity, ignoring important information essential to the reconstruction of the original signal.

The lack of tools — especially to envelope detection and signal segmentation — for a deeper understanding of the core characteristics of discrete signals contributes to the current situation and must be dealt with before one is able to formulate such a representation. The following specific objectives, therefore, consist in filling those gaps.

1.2.2 Formulation of an accurate envelope detection algorithm

Available envelope detection algorithms are not accurate enough to enable a separate understanding of the fine structure of a signal, particularly its evolution, and an analysis of the evolution of the envelope itself. This hinders a complete description of the signal, especially for sound reconstruction purposes, since this disentanglement is essential to pinpoint causality relations between loudness and other articulations from the timbre of instruments.

1.2.3 Formulation of a signal segmentation algorithm to divide a signal into its pseudo cycles

Discrete signals are commonly represented as a series of numbers representing the instantaneous intensity of the signal at regular time intervals. For digital signals in general, which often exhibit some degree of periodicity, this direct representation, while convenient, can be highly redundant. This problem becomes more pronounced in the case of discrete signals representing sound, where an elevated number of individual samples is needed: one second of audio, in CD quality, is represented by 44100 individual samples for each channel of a stereo format, for a total of 88200 individual samples. Representing this discrete signal in the frequency domain would involve 44100 complex numbers, whose

real and imaginary parts would be computationally equivalent to the 88200 numbers in the original, time domain representation.

The high dimensionality involved in both representations renders them unsuitable, efficiency-wise, for machine learning-based applications. For quasi-periodic signals, the intrinsic relations between neighboring samples are not emphasized by either of those representations, and identifying and taking advantage of them becomes, thus, the responsibility of algorithms that use one of those representations. A more high-level representation, encoding those relations organically, would alleviate the conceptual complexity of further processing algorithms.

1.3 LIMITATIONS

Given the breadth of the scope of the present work, one of its limitations has to do with the depth with which each area could be presented. This work adopts a multidisciplinary approach to solve the problem of neural networks-based real-time sound synthesis and demands familiarity with a wide gamut of fields that range from traditional DSP to contemporary AI. Solutions to some of the problems described in previous sections also take inspiration from related areas, such as computational geometry. All those areas mentioned have researchers and practitioners that devoted their whole careers to those individual fields and, naturally, the time constraints prevented an investigation as deep as the author would like.

A more pragmatic limitation was found when searching for quality, freely available sound libraries to train the final Omnes Sonos instruments. The libraries found, especially representing the singing voice, do not provide the ideal examples of articulations and are generally poorly documented, being generally the effort of individuals without institutional support. Besides, the quality of the recordings is often lacking, owing to the expensive infrastructure needed to record, in isolation, a great number of samples.

1.4 STRUCTURE OF THIS WORK

Given the wide scope of this work, Chapter 2 consists of a bibliographic review, aiming at familiarizing the reader with the topics that underline the work; the topics are presented with varying detail, depending on how unorthodox the topic is with respect to the mainstream DSP field.

Under this chapter, Section 2.1 presents a brief timeline of digital computers, from mainframes to current desktops, and how they impacted musical instruments, especially

with the advent and popularization of digital musical instruments.

In the following section , Section 2.2, dedicated to an overview of envelope detection techniques, a brief historical account, and the state of the art of the area are presented. Since they will be useful in Chapter 3, when a geometric algorithm for envelope extraction is introduced, topics such as how one defines the shape of a set of points, convex hulls, and others of geometric nature are introduced in more detail in subsection 2.2.1.

The segmentation section, Section 2.3, on the other hand, is more succinct, reflecting both the relatively lower volume of work concerning the subject found in the literature and its more pragmatic nature.

The chapter advances with Section 2.4 presenting an account of the intersection between the areas of AI and sound, helping to pinpoint the context in which this work was developed.

The bibliographic review chapter ends with a brief history of the MIDI specification, given its importance as a high-level music notation specification, and its centrality to the work here presented.

The first two sections of Chapter 3 are concerned with waves: entities central to this work, but that are seldom objectively defined in the literature.

In Section 3.1, about waves in general, instead of trying to propose a general definition for a wave, an endeavor that many more capable minds shied away from, this work aimed at providing an intuitive understanding of this entity algebraically grounded by the derivation of the wave equation introduced by d'Alembert as the solution for the problem of the vibrating string (Oliveira, 2020). This section also provides, intertwined with the development of the theory relevant to this work, a historical background of the theme.

The second section of this chapter — Section 3.2 — is more pragmatic in nature, and relates the conceptual continuous wave with its discrete counterpart, serving to introduce the nomenclature used throughout this dissertation.

The next two sections start the development of the main objective of this work by filling the relevant gaps in the DSP theory, to subsequently introduce a novel representation for pseudo periodic time series, of which sound representation is our primary interest.

Those two sections — Section 3.3 and Section 3.4 — build upon sections 2.2 and 2.3 of the bibliographic review, where envelope detection and segmentation algorithms, respectively, are commented: novel algorithms for each of those tasks are presented;

despite being developed as tools to enable the simplified representation envisaged in this work, those algorithms were chosen to be presented in this chapter given their generality.

Building upon those algorithms, subsection 3.4.1 presents the digital signal representation. Although much of the theory is drawn from conventional areas related to digital signal processing and image processing, among others, the representation was formulated from the beginning having in mind the strengths and limitations of neural networks: by representing the pseudo cycles in the frequency domain, redundancy in the representation is minimized, and the temporal dependency between entries of the same output eliminated, allowing the network to exclusively focus on learning the mapping between different inputs and outputs.

This representation and the underlying theory, initially described in general terms, are then applied in the implementation of a real-time audio plugin, capable of emulating virtually any real-world acoustic instrument as well as the singing voice.

In chapter 4, by contrast, methods specific to this work are introduced, in order to address more pragmatic problems in the realm of applied DSP, such as asynchronous programming and digital instruments libraries and standards. After a brief overview, in Section 4.1, of the ecosystem in which the plugin is implemented, Section 4.2 introduces the process of designing and implementing the plugin itself. Section 4.3 explains in detail the process of designing the neural networks that serve as engines to the plugin, while Section 4.4 comments on each of those trained networks, that constitute the core of the individual instruments.

Chapter 5 investigates the results of this work. In Section 5.1 the envelope extraction algorithm is analyzed, while Section 5.2 investigates the quality of the segmentation algorithm. The whole plugin is analyzed in Section 5.3.

Finally, chapter 6 comments on the whole work, pointing out its strengths, weaknesses, and further research directions.

2 BIBLIOGRAPHIC REVIEW

The digitalization of music, as first stated by Max Vernon Mathews (M. V. Mathews, 1963), brought limitless possibilities to timbres and sounds, a potential only limited by the available processing power and the capability of the high-level sound representation used.

As noted in Smith III (1991), the first limitation, whereas pronounced in the 1960s, is vastly overcome today, thanks to the advancements in hardware and general digital computing technology. The problem of the representation — algorithmically generating the millions of samples necessary to define a digital sound signal from a much smaller, manageable set of numbers — is still an open question, however.

While significant progress was made in this regard, with a proliferation of digital sound synthesis techniques, the understanding, and consequently the emulation of real-world acoustic instruments, is still lacking in many respects.

Considering the piano, perhaps the most important and symbolic instrument in Western culture (Aho, 2009; Bank et al., 2019), as an example, it is easy to see why. Piano sounds are generated by an intricate physical process that involves, for a single key, wooden hammers striking multiple strings that reverberate and interfere not only with other strings in the same course, but also, in varying measures, with all strings in the instrument. This complex pattern of vibrating string is propagated to the instrument's bridge, body, and so on. This intricate vibration can only be understood as sound after being transmitted by a fluid — usually the air — to a comparatively small target localized elsewhere in space: a microphone, or a listener, for example.

Given the importance of the instrument, this complex process was largely studied. Conklin, for example, published a series of articles in the 1990s investigating the various aspects of piano construction, and how they influence the instrument's sound (Conklin, 1996a; Conklin, 1996b; Conklin, 1996c).

More recently, Bank et al. (2019) wrote a review of physical modeling approaches to the piano, where the algorithms presented range from the more physically accurate models, intended to further the understanding of the acoustic phenomena that take place in the generation of the piano sounds, to simplified models suited for real-time synthesis.

Due to the challenges of owning an acoustic piano — from the price to space requirements, passing through maintenance costs — there is a strong demand for digital pianos. Despite this demand, the advancements in the understanding and modeling of the

instrument, and in general sound synthesis techniques, most digital pianos and virtually all the commercially successful ones, rely on pre-recorded samples (Bank et al., 2019): the sound of the digital version is generated by the playback of sounds pre-recorded from an acoustic piano.

Generally, each key is recorded at many velocities, and the results are quite acceptable. Nevertheless, phenomena such as sympathetic resonance are neglected in the emulated instrument.

Perhaps of more concern is the stalemate in which such an instrument, pleasing to western musical sensibilities, seems to encounter itself: Steinway & Sons, the most famous piano manufacturer, has demonstrated difficulties in improving its production process, declaring having reached a plateau (Aho, 2009). If digital pianos can do no more than imitate their acoustic counterparts, via the use of pre-recorded samples, we have little hope of further improving the sound of such instruments.

The concern that we might be sonically constrained is also expressed in Aho (2009): "The actual sounds of existing acoustic instruments might be much less pleasing musically than many other sounds that could be imagined: musical sounds we could hear, but which our acoustic instruments cannot produce because of the constraints set by their acoustic nature."

More prosaic problems exist, however, in the prevalence of sample-based approaches in the digital recreation of real-world acoustic musical instruments, such as the huge storage requirements, that prevent their implementation in cheaper, more accessible hardware. Another problem is the difficulty in incorporating continuous variables in the model. In the case of the piano, we have, for example, the tuning of the instrument, the position of the virtual recording device, and the hardness and mass of the hammers, to name a few (Aho, 2009). Those variables are not easy to change in a physical piano, and are generally not incorporated in their digital counterparts, with little compromise to their quality. Observing more flexible instruments, however, such as electric guitars, capable of bends, tremolos, hammer-ons, pull-offs, tapping, artificial harmonics, and a multitude of articulations to be accounted for in a sample-based, digital emulation, this problem becomes more prominent.

The consensus among the music production community is that the use of samples, and the consequent compromise in flexibility, is unavoidable when a realistic digital emulation of real-world instruments is the main goal: the general understanding is that the physical process underlying the sound of most instruments is so complex that, even

when reasonably understood, cannot be translated into an accurate real-time model on current hardware. The lack of realistic models for the (physical) modeling of most but the simplest plucked string instruments adds to this general understanding.

On the other side of the coin, the relatively recent re-emergence of AI, and the impact it caused in areas such as computer vision and natural language processing suggest another approach.

This work can be seen, therefore, as an endeavor to contribute a solution to the second limitation stated by Mathews (M. V. Mathews, 1963), by leveraging the recent advancements in machine learning, in the form of a general methodology for the real-time emulation of acoustic musical instruments and the singing voice. This methodology bypasses the problem of fully understanding the inner workings of specific instruments with the use of a novel digital signal representation tailored for neural networks and, for this very reason, achieves its generality.

While, in so doing, the method presented here doesn't directly address the more abstract problem of digitally advancing the timbres of conventional instruments, the machinery devised during its development can be used, in conjunction with other methods, to gain a better understanding of those timbres. As will be seen, the method here proposed demands accurate envelope detection and sound segmentation techniques, the likes of which weren't found in the current DSP literature, and had to be developed. Those tools can be used to better understand the nature of musical sounds and signals in general, with immediate applications to areas such as lossy sound compression.

All the theory explored throughout this work is ultimately applied in a framework for the emulation of acoustic instruments in general, where instruments are trained using a collection of samples exemplifying the timbre and the articulations of interest, giving rise to a set of production-ready digital instruments that can be used directly or as a virtual studio technology (VST) plugin hosted in any compatible digital audio workstation (DAW).

In order to provide context, the next section of this chapter is comprised of an introduction to the history of digital computers and their uses in musical applications in general. Given the interdisciplinary nature of this work, the following two sections, in conjunction with Chapter 3, are intended to present the theory necessary for the understanding of the approaches developed in Chapter 4, with just enough background information for their full appreciation. This chapter also comments on the state of the art of the pertinent literature, pointing out relevant applications when appropriate.

The tone changes in the subsequent section to a less pragmatic approach, when the state of the applications of AI in general, and neural networks in particular, to sound generation, is investigated. The general idea of the third section of the present chapter is to highlight the gaps in the interface between those areas that this work intends to fill.

2.1 MUSICAL INSTRUMENTS AND DIGITAL COMPUTERS

Music is a staple of humanity's creativity. Montagu (2007) defines music as sounds generated deliberately with the aim of provoking emotion. Musical instruments, consequently, can broadly be defined as any tool used for that purpose. In this context, music can be traced back to immemorial times, with its origins probably motivated by ritual practices performed by early humans. Whereas the very first musical instrument was most probably the voice, it might be shortly followed by other rudimentary musical instruments, likely of percussive nature (Montagu, 2007): Evidence of the use of simple instruments, such as drums and bone flutes, can be traced back to the Paleolithic era, more than 30000 years ago (Bovermann et al., 2017).

During the course of human history, musical instruments became more elaborate, incorporating technological advancements besides responding and also helping to evolve the predominant aesthetic of music in each society. Despite reaching, along this long period of evolution, a high degree of sophistication in their design, before the XX century, musical instruments necessarily relied on mechanical vibrations to generate their sounds.

In a comprehensive timeline of the technology associated with electronic music, Stubbs (2018) marks the year 1876, with the invention of the Musical Telegraph by Elisha Gray — a device that broadcasted single notes along the telegraph lines — as the beginning of the electronic instruments' era.

Regarding stringed instruments, in 1890, George Breed, a United States Naval Officer, filed a patent for an electrified guitar. Electricity, however, wasn't used for amplification purposes, serving instead to continually vibrate the guitar strings, leaving the instrument's sound acoustic in nature. The patent also suggests the use of the method for an electric piano, albeit in a less detailed way. The guitar's design, in addition to differing considerably from what we now consider an electric guitar, presented some idiosyncrasies that probably prevented its commercial success (Hill, 2008).

Other authors place the origin of this new electronic era for musical instruments at the invention of the Telharmonium, patented in 1897 by Thaddeus Cahill, due to the influence the instrument eventually exercised. The Telharmonium, also known as

Dynamophone, was a machine weighing around two hundred tons, intended to synthesize music, with the use of sinusoidal tones of different frequencies generated with modified dynamos, to be broadcasted in real-time over the telephone lines (Collins et al., 2007).

Others still, such as Montagu (2007), attribute the beginning of the electronic instruments to the invention of the Theremin by Lev Termen, at around 1920, on the grounds of the popularity accrued by the instrument at the time, both in its original USSR and in the USA.

About a decade later, the first commercially successful guitars started to emerge. Notorious among them was the Gibson ES-150, in 1936, with a single pickup responsible for capturing the vibration of the metallic strings (French, 2012). Electric guitars, due to the mechanical origins of the sound generated, aren't regarded as pure electronic instruments.

Be that as it may, the Moog synthesizer was the instrument responsible for popularizing electronic instruments. Implementing the ideas published by Robert Moog (Moog, 1965), it is largely considered the first analog synthesizer, and the first commercial synthesizer.

The 1968 Wendy Carlos's album Switched-On Bach, featuring Bach compositions arranged for the Moog synthesizer, achieved critical and popular acclaim (Collins et al., 2007), helping popularize the instrument, that would also be used by bands such as The Beatles, Rolling Stones, and Yes.

The theoretical foundation underlying the developments introduced by the process of electrification, especially concerning the area that came to be known as digital signal processing, can be traced back to a few centuries before those events, however.

The work of d'Alembert and the introduction of differential equations (Oliveira, 2020), Euler, with the invention of integral transforms (Dominguez, 2016), Bernoulli and most notably Fourier formed the theoretical basis for the birth of DSP (Alessio, 2016): how this process happened is described in more detail in Chapter 3.

For the theory to be applied in the form it is known nowadays, however, computers were needed, and it would be only after the second world war that such machines would first emerge, with examples such as the Electronic Numeric Integrator and Calculator (ENIAC) and the Institute for Advanced Study machine (IAS), dating from the mid to late 1940s. With growing access to computers, the area of DSP started to consolidate itself in subsequent decades.

Fortunately, parallel to that revolution, another one was taking place, one that

would eventually lead to the development of modern digital computers. The need to process large amounts of data, both in Europe and the US, lead to an increasing interest in calculating machines. A punched card tabulating system was developed for the US government to be used in the 1890 census, and became so popular in the first half of the twentieth century that originated IBM. A few decades later, the company consolidated itself as a business machine manufacturer, with an income of 20 million dollars by 1928.

This quest for computing machines, and the rudiments of modern computer science, can be traced back to at least a century earlier, however, as described in Kelly (2014), to the efforts of Charles Babbage.

Motivated by the desire to improve the process of manufacturing navigational tables for the English government, Babbage designed his first famous computation machine, the Difference Engine, in the early 1820s. Despite securing funding, and completing a working prototype, Babbage never completed a full-scale Difference Engine. Technical problems encountered along the way led him to investigate the state of the art of English mechanical systems and eventually conceptualize, about a decade later, the Analytical Engine, first mentioned in a statement for the English government in 1834.

By suggesting that funding should be shifted to this new machine without having completed the former and the tables it was supposed to produce, Babbage undermined the government's confidence in him. Despite this, Babbage continued to work on the Analytical Engine for the rest of his life, and secured the interest of people such as Ada Lovelace, responsible for the most extensive account of the machine at the time.

Ultimately, however, his efforts were insufficient to ensure the construction of the machine, with Babbage's failure motivating scientists to momentarily pursue another path to computing machines: analog computing. This was unfortunate, since Babbage's design for the Analytical Engine exhibited all the characteristics encountered in a modern digital computer, such as separation of arithmetic and storage, and would likely be Turing-complete (Graham-Cumming, 2010).

Nevertheless, the beginning of the XX century saw the ascension of analog computing, with the term analog coming from the word analogy. Analog computers were single-purpose machines built as scale models of the problems they were intended to solve, such as dam building and electrical grid design.

In 1937, inspired in part by Babbage's concepts, Aiken proposed to IBM the first designs of the Mark I, to be finalized, after a series of delays, in January 1943, to become the first fully automatic computation machine. It was electro-mechanic in principle,

with an abundance of moving parts, and was eventually superseded by strictly electronic designs.

Parallel to those developments, in the field of mathematics, Turing and Church were making advancements in the theory of logic, independently arriving at similar results in the mid-1930s. Turing's approach, however, involved a conceptual computer, later known as the Turing Machine, and a proof, described in his article "On Computable Numbers with an Application to the Entscheidungsproblem" (Turing, 1937), that this idealized machine could compute any function.

While staying at Princeton University, Turing met John von Neumann, who would go on to play a prominent role in the invention of the modern electronic computer, inspired in part by Turing's work.

John von Neumann would see himself involved in the computer development scene when he became aware of the ongoing efforts for the construction of the ENIAC, around 1944, a computer that used thousands of valves, and was built to use the decimal system. Neumann started to work as a consultant for the team and soon identified many potential problems with the machine, leading him to start designing a successor capable of addressing those problems.

His report "A First Draft of a Report on the EDVAC" (Neumann, 1993), originally from June 30, 1945, laid the basis for modern digital computers. When finally operational, at the beginning of 1950, the Electronic Discrete Variable Automatic Computer (EDVAC) brought with it the new era of digital computers. From this point on, the process of popularization of the computer, from a mathematical machine to a general-purpose appliance, would continue, eventually culminating in the creation and widespread adoption of personal computers.

Since the early days of digital computers, even when mainframes were the only available machines, restricted to organizations with considerable resources, people found ways and motivation to overcome technical limitations and develop artistic applications, like video games, in their — and the machine's — spare time.

At the 1940 New York World's Fair, for example, the Nimatron was first exhibited (Condon, 1942). It was a single-purpose, electromechanical machine designed to play the game of Nim, described in Redheffer (1948). Another example is a computer chess game for programmable machines, written by Turing in 1947, that couldn't be implemented due to the limitations of available computers at the time (Donovan, 2010). In the following year the patent for the cathode-ray tube amusement device, considered the first known

example of an interactive electronic game, was granted (Wolf, 2021).

Regarding musical application, in 1957, while working at the Bell Telephone Laboratories, Max Mathews created the MUSIC I, a programming environment intended to simulate the circuitry for analog musical synthesizers in digital computers (Collins et al., 2007). His efforts are described in Max V. Mathews (1961), and inaugurate the quest for digital sound synthesis.

In the seminal paper "The Digital Computer as a Musical Instrument" (M. V. Mathews, 1963), Max Vernon Mathews, after years of using computers to analyze sounds at Bell Labs, argued that digital computers could be used not only to aid in composition and other high-level music-related tasks, but to generate sounds, that could be reproduced by loudspeaker

The paper highlights that computers are theoretically capable of synthesizing any sound, proceeding to describe the process in which discrete numbers can be transformed into sounds and, in so doing, introducing the concept of discrete numbers as samples from the instantaneous pressure of a sound wave.

To convert those numbers to sound, a digital to analog device, capable of generating pulses proportional to the magnitude of each number, is to be used, and those pulses smoothed a posteriori. The paper also highlights the practical necessity of compact sound representations.

Mathews went on to implement, in 1968, the successor of MUSIC I-IV family, Music V, in Fortran, the first music programming language to be implemented in a portable programming language (Collins et al., 2007).

Around that time, in 1965, Cooley and Tukey introduced what is now known as the fast Fourier transform (FFT) algorithm, in the paper "An algorithm for the machine calculation of complex Fourier series" (Cooley et al., 1965), one of the 10 most influential algorithms of the 20th century (Dongarra et al., 2000).

The FFT algorithm, by exploiting symmetries, reduced the complexity of the calculation of the discrete Fourier transform (DFT) from $O(N^2)$, where N is the length of the original discrete signal, to $O(N \log_2(N))$, for cases when N is a power of 2. Although this algorithm was originally discovered by Gauss (Cooley, 1987), and rediscovered at least a handful of times in the approximately 100 years between Gauss and Cooley (Heideman et al., 1985; Ceccherini-Silberstein et al., 2018), it became famous in the form proposed by the latter.

2.2 ENVELOPE DETECTION

Envelope detection, also known as envelope tracking, following, or demodulation, has applications in medicine, sound classification, sound synthesis, seismology, and speech recognition. Despite its importance, there is no general method to envelope detection, with most approaches involving manual intervention (L. Yang et al., 2014), for example in the form of filter design.

The practical interest in the envelope of a signal started with the widespread adoption of radio communications, even before the advent of digital signals. In this setting, the problem is usually restricted to the investigation of methods for retrieving the envelope of narrowband, artificial signals, generally with well-defined frequency ranges (Turner et al., 2011).

In 1946, Gabor (Gabor, 1946) made use of the then relatively new mathematical machinery of quantum mechanics to introduce a new representation of a real signal, using the Hilbert transform to unify its time and frequency-domain representations in an equivalent complex signal (Hahn, 2007).

The resulting complex signal, which became known as the analytic signal, has the form $A(t) = S(t) + \mathcal{H}(S(t))\mathrm{i}$ (He et al., 2016), where $S(t)$ is the original real signal and $\mathcal{H}(S(t))$, the imaginary part, its Hilbert transform. The envelope of a signal can be obtained from its representation as an analytic signal simply by computing the complex modulus of the latter.

This approach for envelope detection is mathematically rigorous, and works well for narrowband signals, characteristics that helped solidify the Hilbert transform as one of the most used tools for envelope detection. For broadband signals, however, envelope detection techniques based on the Hilbert transform yield poor results (Dau, 2012; Jia et al., 2019) that are even, in some instances, incoherent from a physical standpoint (Loughlin et al., 1996; L. Yang et al., 2014).

Despite the absence of a general mathematical definition of an envelope for broadband signals (Hlawatsch et al., 2008; X. Hu et al., 2012; L. Yang et al., 2014; L. Yang et al., 2015; Jia et al., 2019), a category that comprises most real-world signals of interest, most DSP practitioners and researchers agree that an adequate envelope should exhibit some characteristics, such as smoothness. It is also widely accepted that, in general, besides being a smooth curve, an envelope should ideally be situated as close as possible to the original signal (L. Yang et al., 2014; L. Yang et al., 2015; Jia et al., 2019), without

ever intersecting it.

A formal set of requirements, with the objective of guaranteeing that an envelope will behave in accordance with physical expectations, is introduced in Section 3.3.6, when this work discusses approaches to assess the quality of envelopes.

Figure 1 presents an example of a Hilbert envelope, that helps to visualize the intuitive differences in envelope quality in the absence of Gaussian noise, and in its presence. The envelope is extracted from a simple sinusoid, artificially generated, and modulated by a polynomial.

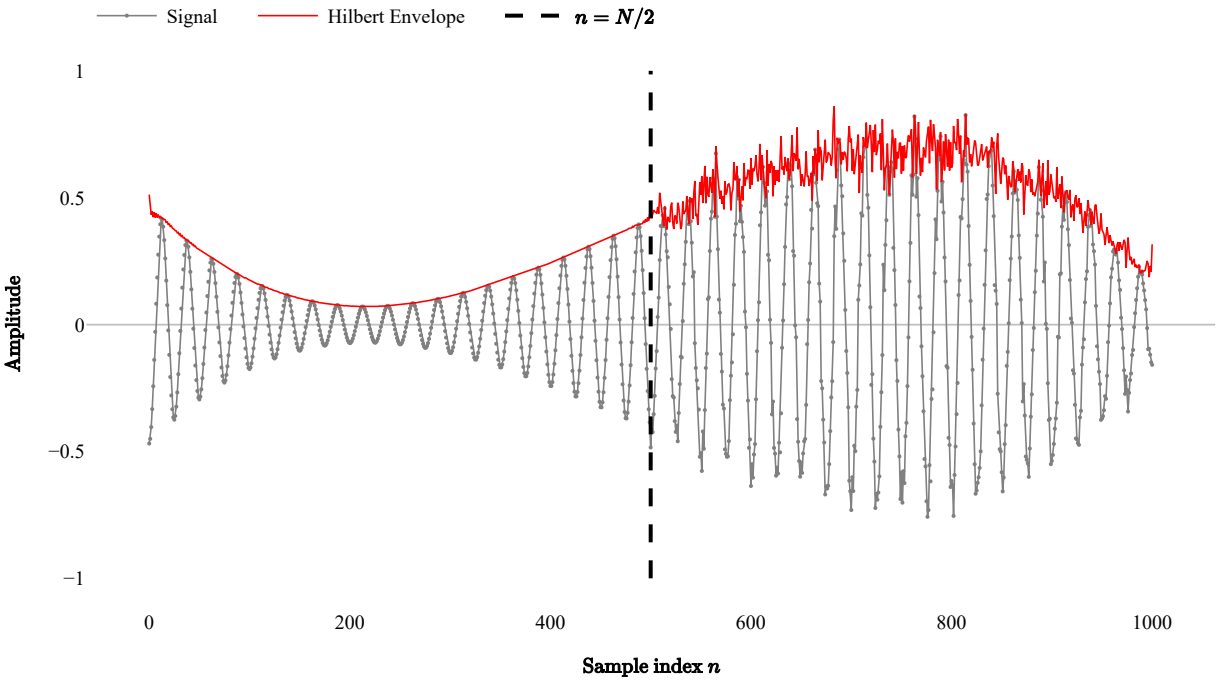

Figure 1: Envelope of a pure sinusoid with a local frequency of 40 cycles per its N samples, modulated by a polynomial of degree 3, obtained by the Hilbert transform approach. In the first half of the samples, the sinusoid is free of noise. In the second half, Gaussian noise was added to the signal.

As seen, the intuitive definition of an envelope suggests that it should be a smooth curve; this smoothness requirement implies that the envelope should ideally be comprised of lower frequencies than the original signal. For real-world signals, such as the one of an alto singer sustaining a steady note, partially shown in Figure 2, the Hilbert transform alone provides a poor envelope, since it retains most of the frequencies of the underlying wave.

Applying a low pass filter, we can transform the result of the Hilbert transform in order to comply with the smoothness requirement, by removing high frequencies from the original result. This will potentially lead to undershoots, however, proportional to the

power of the high-frequency content of the transform removed without compensation.

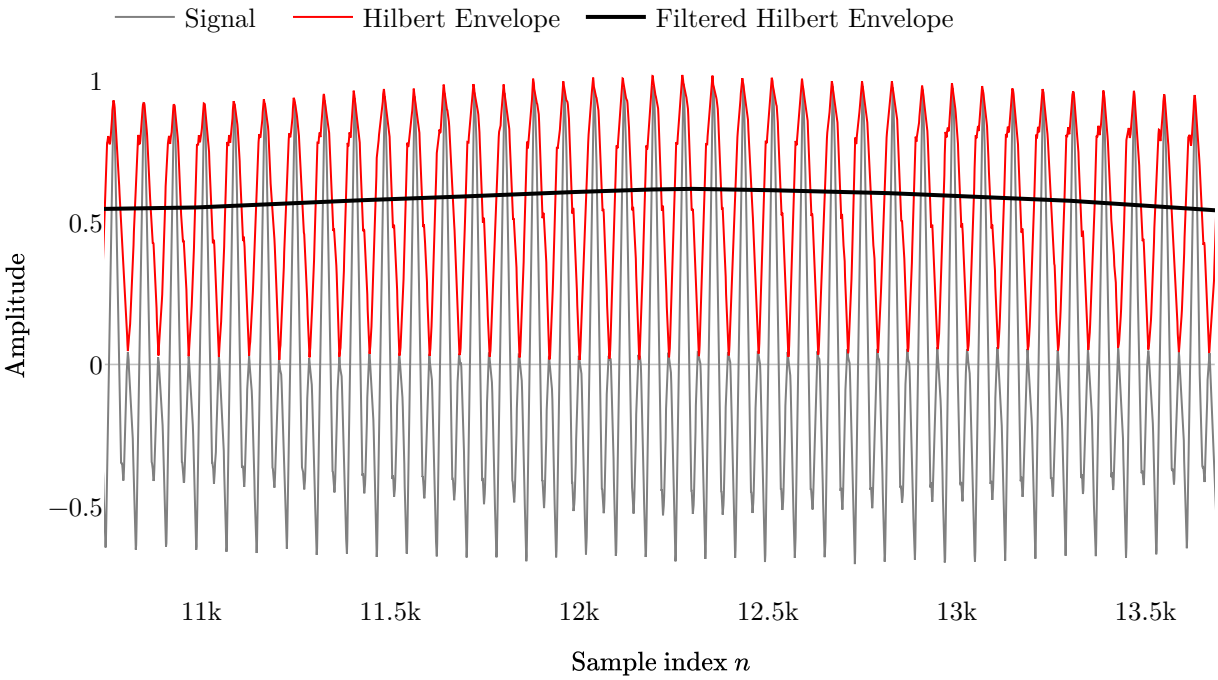

Figure 2: Comparison of the envelopes obtained by the Hilbert transform, with and without further filtering, for a section of the representation of an alto singer sustaining a steady note. The filtered envelope, while smooth, significantly undershoots the original signal.

In the context of DSP, the demand for envelope detection algorithms is high (Caetano et al., 2011). Since the temporal envelope of a signal is of primary importance in a considerable number of applications, various specialized algorithms that take advantage of prior knowledge about the nature of the signal of interest exist. The algorithm presented by W. Yang et al. (2014) for the distributed monitoring of fiber optic, or the one formulated by Assef et al. (2018) in the context of medical ultrasound imaging, are some examples.

Regarding the psychoacoustic importance of envelopes, on the subject of intelligibility of Mandarin speech, the envelope is as important as the spectral content of the underlying signal (Qi et al., 2017); in the case of the English language, Shannon et al. (1995) work illustrates that envelopes modulating a signal consisting mostly of noise were still able to convey meaning.

Important characteristics of both voice (Zhu et al., 2018) and music (Lokki et al., 2011), such as emotions and identity, are also conveyed by the envelope modulation of the underlying signal.

The importance of the envelope in a wealth of applications on one hand, and the

absence of a general approach to envelope detection, on the other, lead to a fragmentation in the literature about envelope detection (Richard Lyons, 2017), that was aggravated by a lack of a strict mathematical definition of a temporal envelope (L. Yang et al., 2014; L. Yang et al., 2015; Jia et al., 2019).

2.2.1 The shape of a set of points in two dimensions

Despite being such an elusive concept, the accurate identification of the temporal envelope of a broadband signal is central to the representation introduced in this work, as will become clear in Section 5.2. The absence of a fitting algorithm motivated a search for alternative, less orthodox approaches to envelope detection.

Powerful algorithms exist in the area of computer graphics to transform a cloud of points in three-dimensional space, a common result of 3D scanning processes, into a 3D surface.

Considering the most common visual representation of a discrete wave — where each sample is plotted in a two-dimensional graph where the horizontal axis is usually proportional to time and the vertical axis proportional to the amplitude of the wave at each sampled point — it is not difficult to think of each sample of a discrete signal as a point in the cartesian plane.

What is necessary to enable the use of geometric methods to this problem is a mapping from this space, where the abscissa and ordinate have different units, to the cartesian plane; once this mapping is available the DSP problem of extracting the envelope of a discrete signal can be converted to the geometric problem of defining the shape of a set of points in $\mathbb{R}^2$.

Before, however, it is necessary to settle on a definition of what constitutes the shape of a set of points, one of the most widely used such definition being the concept of a convex hull. The term convex hull appeared first, in English, in Birkhoff (1935), and was consolidated in the literature shortly after (Dines, 1938). Berg et al. (2008) defines the convex hull of a set of points in $\mathbb{R}^2$ as the smallest convex subset of the plane that contains all the points in the set. The term convex hull is, as noted in Dines (1938), ill chosen, since the definition refers to the whole region encompassed by the points, and not only the frontier, as the term “hull” implies.

Intuitively, one can imagine each of the points that are part of the set as the head of nails in a wooden board, around which a rubber band is stretched. The regions delimited by the polygon formed by the rubber band constitute the unique convex hull of the set.

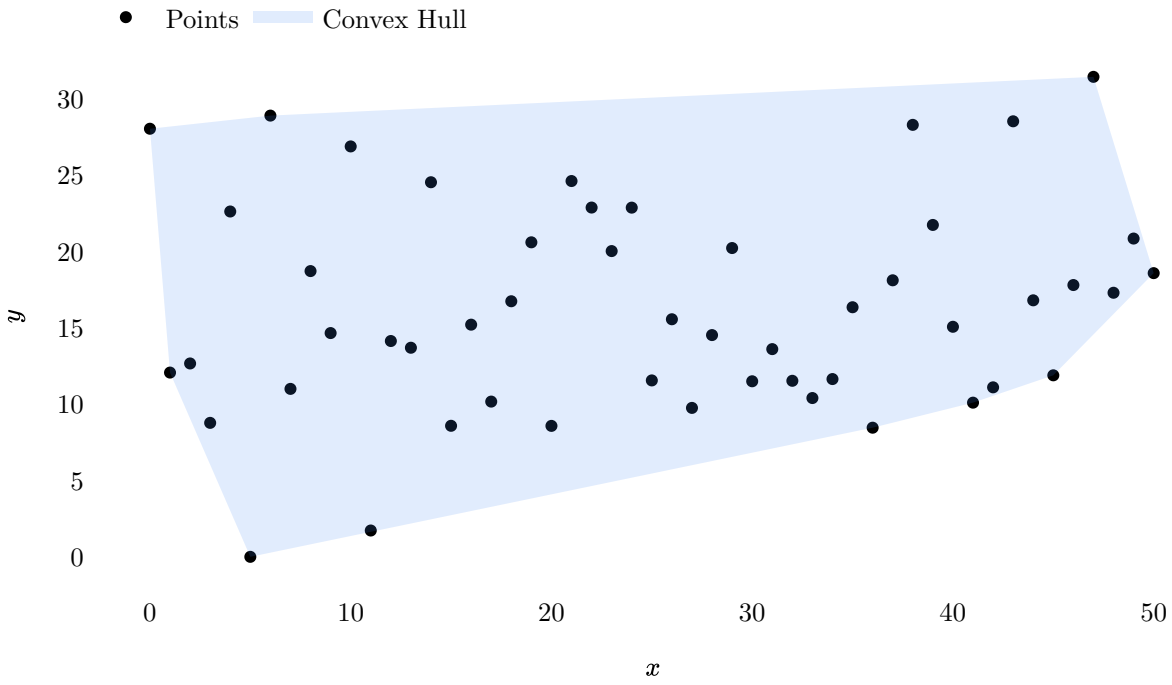

Figure 3: A set of points and its corresponding convex hull.

The region corresponding to the convex hull of a set of points in two-dimensional space will always be convex, as the name implies. For sets of points with relatively extensive "concavities", this leads to a very overestimated convex hull, and consequently, boundary.

The concept of alpha shapes, also known as concave hulls, was introduced by H. Edelsbrunner et al. (1983) to address this limitation. Alpha shapes are a mathematically well-defined generalization of the convex hull of a finite set of points, closely related to the Delaunay triangulation and Voronoi diagrams of those points.

Intuitively, one can imagine that, instead of the rubber band in the convex hull example, one has a wooden disk, lying on the wooden board where the nails are fixed. Without lifting the disk, one can move it freely around the set of nails. The nails that can be touched by the disk in those settings are the ones that are part of the concave hull of the set of nails.

In the two-dimensional case, a more formal definition is as follows: given a fixed radius α, we take all possible pairs of the points in the set, and try to construct a circle passing through those pairs that define an open circumference not containing any other point of the set. If we succeed, the edge defined by those two points is part of the boundary of the set.

Figure 4 illustrates an alpha shape, with $\alpha = 10$, of a set of points. Note that two

points of the set lie on the circle illustrated, on the left-hand side of the circle, but the open disc it defines contains no points of the set. A similar construction is possible for every point at the frontier. Conversely, any circle of radius $\alpha = 10$ containing at least one of the interior points defines an open disc that also contains at least another point of the set.

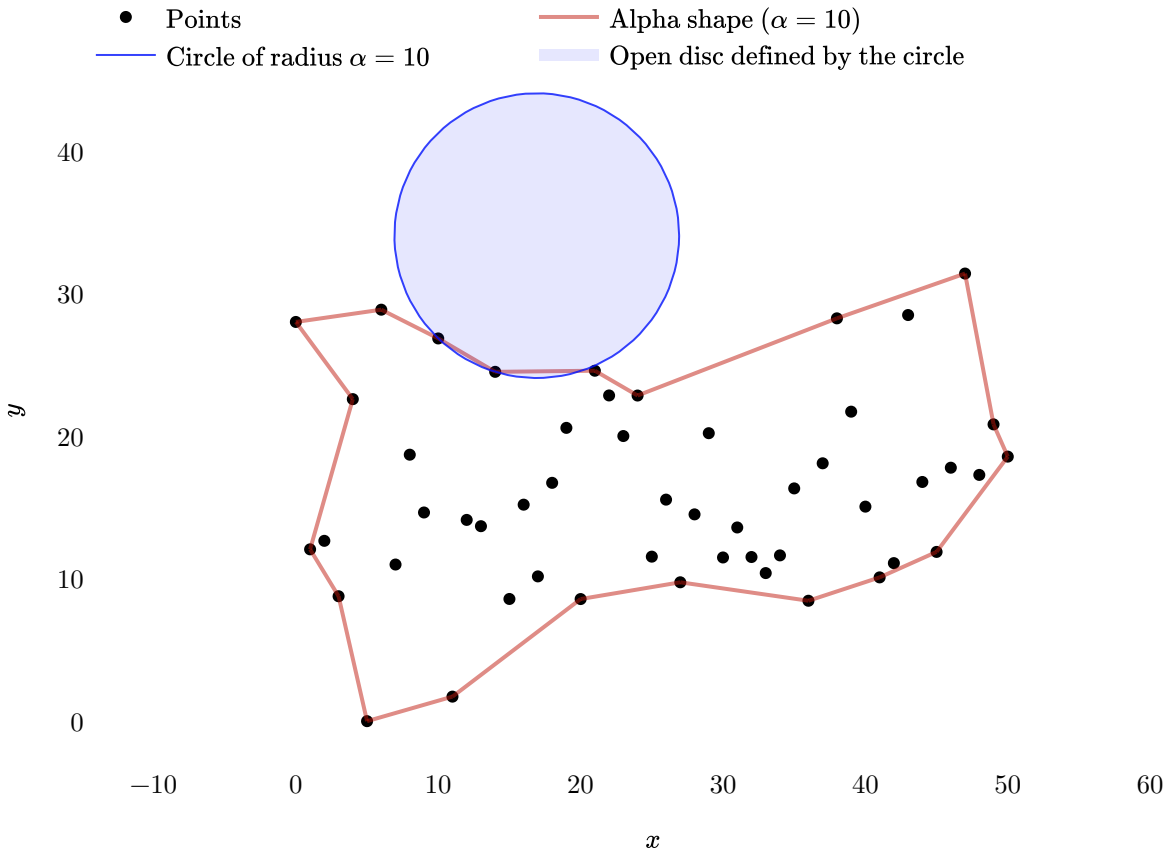

Figure 4: A set of points and its corresponding alpha shape.

Figure 5 presents a comparison between the two aforementioned definitions of the shape of a set of points, as obtained by the convex hull and the alpha shapes. In this example, the shape obtained with the alpha shapes definition, using a radius of 150 units, conforms better with the intuitive notion of shape.

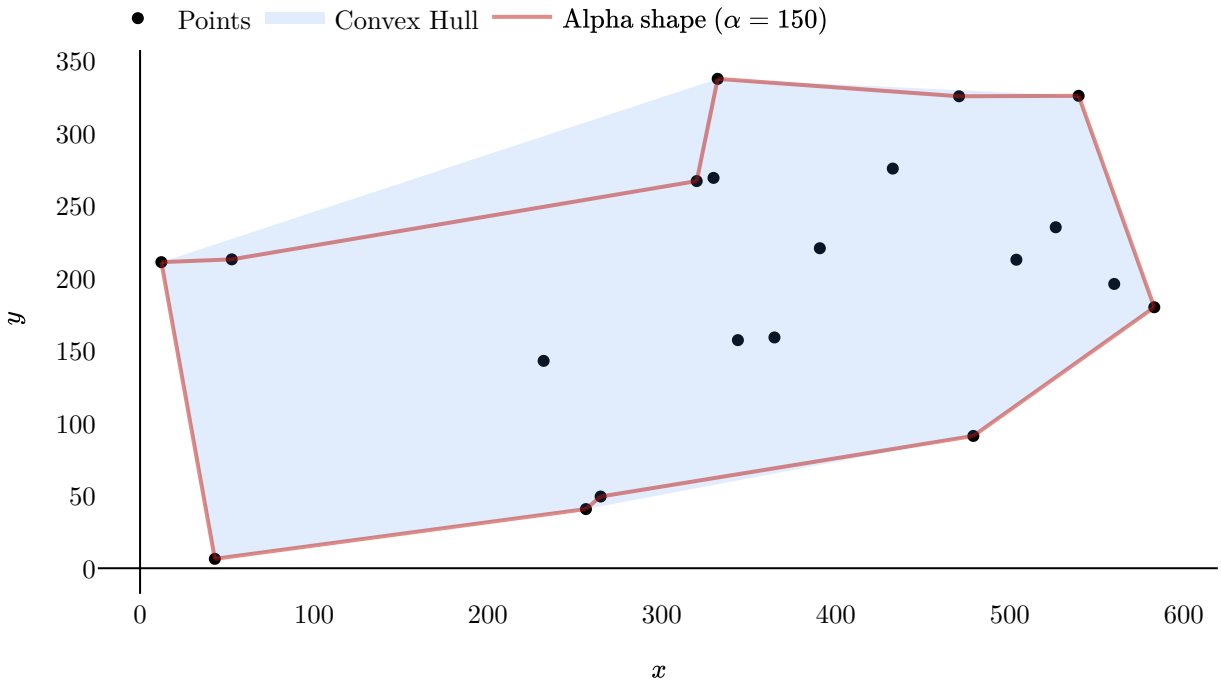

Figure 5: Comparison between the convex hull and an alpha shape of a set of points.

The alpha shape for each specific radius α is unique. Alpha shapes can also be considered as a generalization of convex hulls, since, when the radius alpha tends to infinity, the alpha shape of a set tends to its convex hull.

Consider, for example, the artificially generated discrete wave in Figure 6, where the links between consecutive samples were de-emphasized to highlight the "set of points" nature of a signal: two alpha shapes, with different values of α, and the convex hull are illustrated. As α increases, the contour of the set of points becomes coarser, and closer to the convex hull.

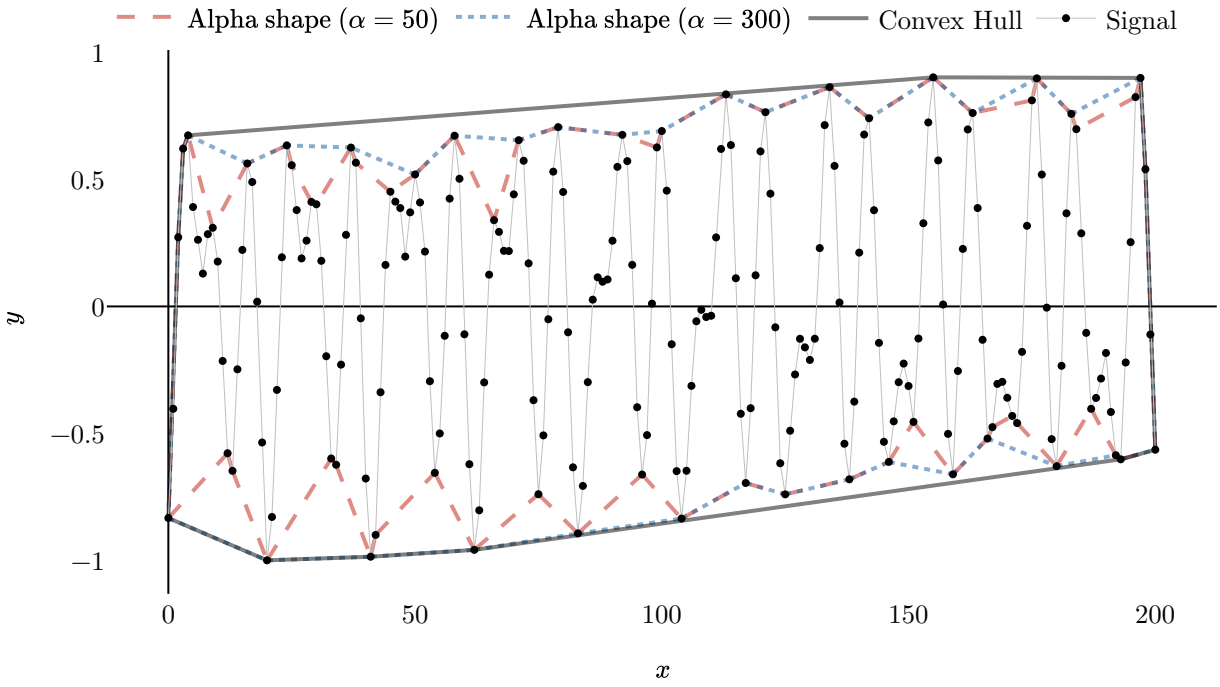

Figure 6: A discrete signal interpreted as a set of points in the Cartesian plane, two alpha shapes, and the convex hull defined by it.

Alpha shapes are used in areas such as detection of features in images (Varytimidis et al., 2016), reconstruction of surfaces from clouds of points (Wu et al., 2015) and spectroscopy (Xu et al., 2019).

This last work, which involves the estimation and removal of the Blaze function — a kind of envelope — of an echelle spectrograph, is particularly illustrative of the potential synergy between geometric and DSP approaches.

Other steps in the direction of applying geometric algorithms to envelope detection were made by C. Yang et al. (2015b), with an algorithm based on the construction of a skeleton underlying the digital wave of interest, and by C. Yang et al. (2015a), via the direct application of computer vision methods to the task of envelope detection.

2.3 SIGNAL SEGMENTATION

Signal segmentation is used in a wide range of applications, such as seismic signals analysis (Popescu, 2014), condition monitoring of industrial processes (Aiordachioaie et al., 2019), bioacoustical animal recognition (Colonna et al., 2015), heart sound segmentation (Moukadem et al., 2013) and electrocardiogram signals analysis (Andreão et al., 2006), to name a few.

Segmenting a signal in time is also a crucial pre-processing step in many audio

applications, a step that can substantially impact the quality of further processing tasks (Rybach et al., 2009). The approaches to signal segmentation are diverse, and generally tailored to specific applications: Dessein et al. (2013), for example, proposes a real-time method to segment an audio signal based on the monitoring of the information rate of the incoming signal; the approach accommodates various homogeneity criteria in order to divide the signal into the segments of interest.

An algorithm specific for music signals segmentation, based on the variance of the spectral domain, is presented in Krymova et al. (2017). Hubert et al. (2018) presents a distance-based maximum entropy Bayesian approach to the segmentation of subaquatic audio recording.

More recently, machine learning approaches were presented in Jingzhou et al. (2019), with the use of a Convolutional Neural Network (CNN) to segment audio from a radio station broadcasting, and Y.-C. Chen et al. (2019), where a Recurrent Neural Network (RNN) was employed to jointly segment and classify words in utterances.

Most methods divide a signal into homogeneous segments (Dessein et al., 2013), according to homogeneity criteria tailored to the specific needs of the application. In this context, the technique of Concatenative Sound Synthesis (CSS), a relatively young research field started in the early 2000s (Schwarz, 2005), is one prominent example.

In Zils et al. (2001), one of the first papers dedicated to the theme, the technique is dubbed musaicing, a combination of the words musical and mosaicing. The paper is focused on the problem of automating sample selection and retrieval.

The basic approach for CSS consists of segmenting the original sound files into meaningful atomic signals, that are saved, alongside their descriptions, in a database. During the synthesis process, those "atoms" are then retrieved, according to a symbolic description, and are concatenated together (Schwarz, 2005).

Vocaloid, a singing voice synthesis software vastly successful in Japan (Kobayashi et al., 2019), uses CSS (Kenmochi et al., 2007). Each "singer" consists of a database of phonetic sounds that are glued together according to the instructions inputted by the user in a virtual piano roll, to generate the sang phrases.

The original signals used in CSS need not be recorded specifically for this purpose, however: in Maestre et al. (2009), for example, the authors use recordings of jazz saxophone to generate the database used to synthesize new melodies.

Many concatenative synthesis algorithms, especially those related to speech, rely on a pitch-synchronous overlap-add (PSOLA) approach for modifying pitch, duration,

and other characteristics of sounds during concatenation; those modifications can be performed either in the frequency domain (FD-PSOLA) or in the time domain (TD-PSOLA) (Moulines et al., 1990).

An explanation of the PSOLA method is given in Valbret et al. (1992), in the context of speech: The original signal is first decomposed into short-term signals proportional to the local pitch-period and synchronized with them. These short-term signals are thus placed at a pitch-synchronous rate on the voiced segments of the original signal and at a constant rate on the unvoiced segments. During the synthesis step, a new set of pitch-marks is designed in accordance with the desired prosodic modifications. Overlap-add is then used to join the segments, that are finally filtered to produce the final result.

Segmenting a signal into pseudo cycles, on the other hand, is an overlooked area in the literature, and can conceptually be seen as dividing a signal into fundamental building blocks.

Such segmentation can also be interpreted as the foundation for an alternative representation of a signal, one that describes the signal as a basic waveform evolving as a function of an independent variable, usually time. An example of this interpretation is illustrated in Figure 7. In the figure, each pseudo cycle is shown, in order, from front to back: pseudo cycle number 0 would be the first pseudo cycle obtained using the segmentation algorithm. From left to right, the figure shows the indices of each pseudo cycle. The pseudo cycles were zero-padded to the length of the longest pseudo cycle, a fact that can be seen in the mostly white and plane region to the right of the figure.

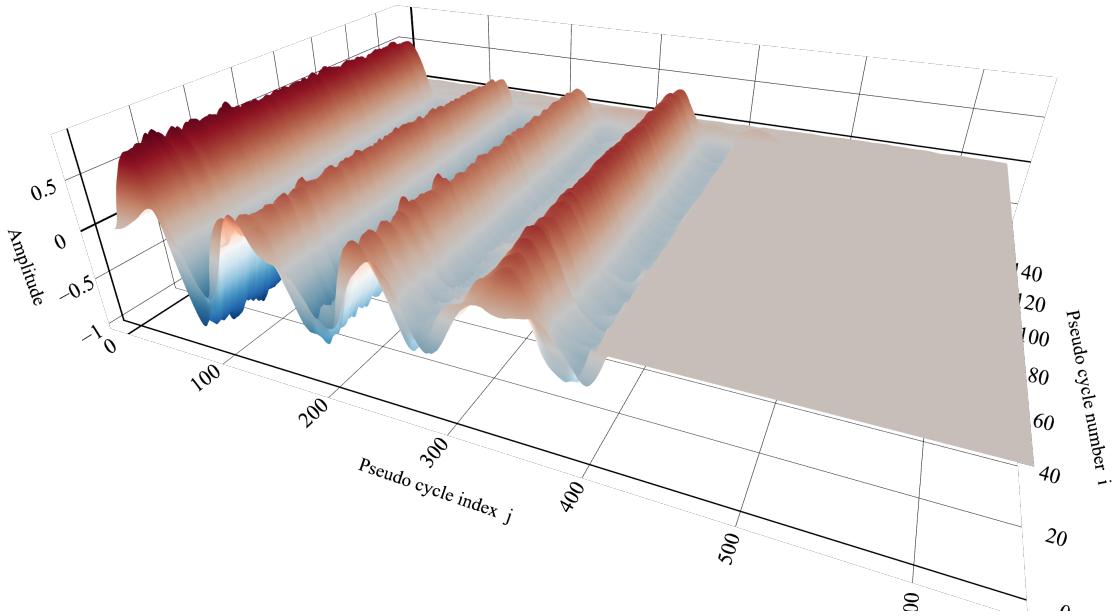

Figure 7: Discrete representation of the sound of a bassoon, segmented into its pseudo cycles.

Traditionally, however, discrete signals are represented in the time or frequency domains. Those representations are deemed equivalent, in the sense that a given signal can be unequivocally transformed from one to another via a well-defined mathematical algorithm, the DFT, presented with more detail in Section 3.1. The existence of an efficient implementation, in the form of the FFT, helped popularize this convention of interchangeably representing a discrete signal in the time or frequency domains, according to convenience.

Alternative representations for digital signals are normally not designed to be reversible, but are meant to encode, in a compact manner, characteristics of the original signal, generally for classification purposes.

Vocoders, created with the aim of encoding and resynthesizing speech (Mills, 2012), are an example of an exception. Based on the work presented in Fujimura (1968), where the multi-band representation of speech, and its advantages, started to be studied, Griffin et al. (1988) introduced the Multiband Excitation Vocoders, expanding vocoders capabilities to high-quality speech generation.

The representation introduced in Levine et al. (1998), which divides a given signal into sinusoids, transients, and noise, with the aim of facilitating pitch-shifting, between other manipulations, is another example of reversible representation.

A more general treatment of this problem can be seen in Serra et al. (1990), where the Spectral Modeling Synthesis (SMS) approach to sound synthesis is introduced. The basic methodology extends the research presented by Serra in his Ph.D. thesis (Serra, 1989) and consists of extracting the "deterministic" part of a signal, which is thus represented as a series of time-varying sinusoids. This part is removed from the original signal, leaving what is to be considered the stochastic part, or "noise floor". This residual noise is then modeled using white noise filtered through a time-varying filter.

Since its inception, this approach gained popularity, in part due to the possibility of extracting information from records of real sounds, that can be reproduced and altered afterwards (Serra et al., 1997). This potential for sound transformation is explored in Serra et al. (1997) and, ultimately, in Bonada et al. (2011), where techniques for manipulating high-level attributes of the representation, in order to make the transformation process more intuitive, are introduced. An entry on SMS can be found in `https://www.upf.edu/web/mtg/sms-tools`, where the main publications on the method, besides an extensive list of related papers, can be found.

In Mcaulay et al. (1988), the technique of Sinusoidal Transform Coding (STC) is introduced. It consists, like in Serra et al. (1990), of representing a sound signal as a series of sinusoids, whose amplitudes, frequencies, and phases vary in time. Unlike in Serra et al. (1990), however, the algorithm is exclusively designed to process voice signals, allowing for increased robustness against acoustic background noise.

The present work introduces, in Section 3.4, a novel representation that, in a manner similar to the STC, can be used to decompose a signal in time-varying sinusoids, albeit ignoring background noise. Since this representation operates at the level of individual pseudo cycles of signals with high harmonic content, it is able to organically incorporate the stochastic part of the sound.

Although not meant to be completely reversible, the representation can in the future be extended to account for the discarded noise, in a similar manner to Levine's (Levine et al., 1998) representation.

2.4 ARTIFICIAL INTELLIGENCE AND SOUND SYNTHESIS

Since ancient times, humanity has dreamed of AI. The dawn of the first computing machines, such as the ENIAC in 1946, brought with it the promise that what we now call general AI was close (Donovan, 2010). Although this proved not to be the case, artistic applications blending early implementations of AI are abundant, and music was one of

the most prominent objects of interest: as early as 1957, Illiac Suite (Hiller et al., 1959), the first AI generated composition (Miranda, 2021), was presented.

The interest in the topic, and the approaches proposed, continued to increase throughout the decades, as can be seen in the following compilations: the one presented by Miranda (2000) offers an early picture of the area, that had an exploratory nature at the time. A couple of years later, Carbonell et al. (2002) present the annals of the 2nd International Conference on Music and Artificial Intelligence (ICMAI 02), where a predominance of interest in analyzing various aspects of music can be seen.

The contents of Miranda (2021), compiled 19 years later, show that the area evolved enormously, shifting to a more applied paradigm, with an abundance of practical applications.

The work in Miranda et al. (2015) also provides insight into this progress, by analyzing the theme as it evolved from the point of view of publications in the "Organised Sound" journal, one of the most prominent publications in the area of technology applied to music.

While high-level approaches have been seen since the beginning of computers, AI based applications focusing on sound synthesis at raw audio level emerged more recently, due to computational constraints (Hawley, 2020), and are rapidly gaining space, especially with approaches based on deep neural networks (Kiefer, 2019).

One of the first such applications was presented in Oord et al. (2016a) and took inspiration from advancements obtained with deep neural networks in the areas of computer vision and text processing. It introduces the WaveNet model, based on the PixelCNN (Oord et al., 2016b) architecture. The WaveNet uses dilated causal convolutions, an architecture where each output is related to every previous output, but that avoids the recurrent nature of RNNs, achieving greater efficiency. The original paper evaluates the model on the generative tasks of multi-speaker speech generation, text-to-speech, and music generation. Besides, the model was shown to perform well on speech recognition classification tasks, with small modifications. Results can be heard at WaveNet's website `deepmind.com/blog/wavenet-a-generative-model-for-raw-audio`

Using RNN, Mehri et al. (2016) present another architecture dedicated to the generation of raw sound. The work introduces an end-to-end audio synthesis model based on RNNs, that avoids the efficiency problems generally associated with such architectures by employing different modules operating at different clock rates. The higher levels of the proposed architecture operate in sets of frames of variable size, offering adaptability

in face of the available processing power. The model is evaluated on the task of speech synthesis, the synthesis of onomatopoeic sounds — sounds such as screaming, coughing, and heavy breathing — and the synthesis of piano sounds, using as a basis a database of Beethoven's piano sonatas, besides being compared with a reimplementation of the WaveNet architecture.

Also inspired by advancements in computer vision, Engel et al. (2017) introduce a WaveNet-style model and the NSynth dataset of musical notes. The WaveNet inspired architecture presented is shown to learn a semantically meaningful hidden representation that can be tuned to change timbre and other dynamics of sounds during playback. In contrast with architectures such as WaveNet and SampleRNN, this paper proposes the use of an autoencoder, removing the need for external conditioning for longer-term dependencies. The tasks of note reconstruction and instrument and pitch interpolation are used to evaluate the architecture proposed.

Engel et al. (2019) investigate the performance of Generative Adversarial Networks (GANs) in the task of raw sound generation, and suggest that this architecture has the potential to outperform WaveNet based architectures. The proposed model, dubbed GANSynth, tries to improve the efficiency of previous autoregressive AI based architectures, mitigating the problem of the inefficient ancestral sampling approach used in those models.

Again inspired by advancements in the computer vision area, the paper proposes an architecture that learns the instantaneous angular frequency of a signal and is able to generate examples approximately 54,000 times faster than the WaveNet architecture.

Autoregressive models, such as WaveNet, model local structure but have slow iterative sampling and lack global latent structure. In contrast, GANs have global latent conditioning and efficient parallel sampling, but struggle to generate locally-coherent audio waveforms.

A similar approach is presented in Donahue et al. (2019), where the WaveGAN architecture is introduced. The work investigates the generation of one second of audio with the use of waveform and spectrogram representations. Examples, generated on demand, can be heard at `chrisdonahue.com/wavegan`.

A very comprehensive investigation of applications of deep generative models for sound synthesis can be found at Huzaifah et al. (2021), while an account focused on the sound representations used in those algorithms is available at Natsiou et al. (2021).

2.5 THE MIDI SPECIFICATION

The MIDI format is a high-level music notation, that abstracts the timbre and other instrument-specific characteristics, and focuses on representing the notes, their duration and velocity, besides other information about the performance dynamics. In other words, the goal of the standard is to encode Human gestural control information, such as keypresses and knob turns (Loy, 1985). With the popularity of synthesizers at the beginning of the 1980s, the necessity for a standard to assure compatibility between products from different manufacturers became evident.

In 1981 a paper was published (Smith et al., 1981) where a proposal for a Universal Synthesizer Interface was presented; this recommended format, after incorporating suggestions from the leading synthesizer manufacturers at the time, served as the basis for the MIDI standard, announced to the public in 1982 (The MIDI Association, 2022).

The goals behind the development of the MIDI standard were the possibility of connecting hardware from different manufacturers together, the possibility of interfacing them with digital computers, and protection of hardware from obsolescence (Loy, 1985).

Version 1.0 of the MIDI standard was conceived as a two-layer specification, in the sense that it standardized the hardware used to link music devices and the format of the data that was to be exchanged through those physical channels (Loy, 1985).

From the hardware perspective, devices implemented following the MIDI standard have a MIDI input, to receive information, and a MIDI output, to send signals. They can also have a MIDI THRU, that passes the received information forward. In all those channels, asynchronous serial data flows in a single direction (Loy, 1985). The software part of the specification is based on the following entities: mode, of which there are three; channel, of which there are sixteen; and commands.

Commands are the entities that reflect the musician's gestures, such as *note on* and *note off*; the commands can be labeled with a specific channel, enabling the receiver of the commands to route those messages accordingly.

The three modes are omni, which enables the commands to navigate through all channels; the poly mode, which causes devices to use a single channel, and causes notes activated at the same time to form chords; the mono mode, which functions similarly, but causes a portamento effect, where the command to activate a note deactivates the anterior sounding note (Loy, 1985).

Despite having become, as early as 3 years after its publication, the *de facto* stan-

dard for the communication between musical instruments, the format is not without its flaws (Loy, 1985). From the beginning, it was clear that the standard favored piano-like instruments, for example. Also, when interfacing with digital computers, the commands need to be time stamped, something that wasn't foreseen by the original specification (Loy, 1985).

Six years after the publication of the MIDI specification, and after its widespread adoption, Moore (1988) pointed out in his publications some of the limitations of the format, from a musical perspective. The most problematic aspects were the uncertainty in the amount of delay, especially in live performance, and the event-driven orientation of the protocol, detrimental to the adoption of a continuous approach that would be more suited for some non-keyboard-based instruments.

Recently, in 2020, the MIDI Manufacturers Association (MMA), a non-profit organization established in 1985 to supervise the MIDI specification, agreed on the version 2.0 specifications for the format. This new version addresses some of the deficiencies of the original format, most notably with the introduction of continuous commands, and the possibility of embedding annotations in commands.

Regarding the MIDI format, the plugin implemented in this work listens for *note on* and *note off* commands, representing instructions to start and stop the sound associated with a particular frequency; these commands can be equated with the press and release, respectively, of the keys of a keyboard, with the difference that a typical electronic keyboard, like a grand piano, has 88 keys, while the MIDI specification has a total of 128 notes, represented by the integers from 0 to 127. Each note command is associated with a velocity that, in the keyboard analogy, represents the force with which the key was pressed. The values of the velocity are also represented by integers from 0 to 127, with zero representing the lightest intensity and 127 the maximum intensity of activation.

Considering the standard piano tuning, where the note A4 represents a frequency of 440 Hz, the MIDI note number and the frequency it represents are linked via Equation 1.

$$f = 4402^{\frac{\text{key}-69}{12}} \tag{1}$$

Following the MIDI standard, the plugin also listens to information not bundled in note commands, that is, information that can change at any time, and in the case of the implemented plugin, must be incorporated in the sounding note in real-time. One

of those instructions is the commands related to manipulation of the Mod Wheel, which also is represented by an integer from 0 to 127, and is used by the plugin to change the vibrato dynamics of the synthesized sound.

The movements of the Pitch Wheel of the controller, likewise, are used to change the pitch of the currently sounding note, as is customary, emulating a bend effect. This information comes in the form of an integer from 0 to 16383, where the integer 8191 represents the neutral position.

3 PROPOSED TECHNIQUES

The first two sections of this chapter are concerned solely with waves, due to their conceptual importance in the work proposed. The first section presents waves in general, pointing out the difficulties in their precise definition, and introducing theoretical results that help to solidify the intuition of what a wave is.

The section that follows is concerned with discrete signals, their relationship with their continuous counterparts, and the nomenclature used in the rest of the work. Those chapters prepare the reader for the introduction, in Section 5.2, of a new signal representation, and particularly the interpretation of that representation.

The two algorithms that serve as basis for the representation introduced in Section 5.2 are developed in the last two sections of this chapter.

The third section illustrates the development of the envelope detection algorithm. The problem is formalized in Section 3.3.1; Section 3.3.2 introduces some simplifications to the task of envelope detection that arise from the envelope definition adopted in this work. Section 3.3.3 explores the geometric characteristics of envelope detection and formalizes how this classic DSP problem can be transformed into the geometric problem of finding the shape of a set of points. In Section 3.3.4 the concept of discrete curvature is defined, and a new discrete curvature measurement is introduced. This measurement is used in Section 3.3.5 to discriminate which samples of a discrete signal belong to the envelope, circumventing the problem of accounting for the inherent uncertainty of the sampling process that generated the discrete signal from an unknown continuous one.

The fourth section of this chapter introduces the signal segmentation algorithm. After the algorithm is presented, Section 3.4.1 investigates how a compact representation can be derived from the output of the segmentation algorithm, and explores its potential for compression purposes.

3.1 WAVES

In Mandel et al. (1975), a mechanical wave is described simply as a perturbation that propagates itself. Whitham (1999) opts for an intuitive definition of a wave, refined by the mathematical development of the theme.

It is not necessary, however, in the context of the present work, to investigate the most general definition of waves. As is clear when thinking about electromagnetic waves, for instance, such a general definition would have to account for waves that don't

need a medium to propagate. We are, instead, interested in mechanical waves, and, more specifically, how they behave in solids. Although ultimately, all sounds come to us propagated through the air — a fluid — for our immediate purposes, we will assume that the movement in the vibration source is proportional to the changes in air pressure that will ultimately stimulate the listener's auditory system.

Deriving d'Alembert's wave equation, as will be done below, besides mathematically grounding the idea of a wave, helps to give a sense of what is commonly excluded from physical modeling theory. All simplifications and assumptions will become explicit in this exercise, helping to highlight, later in the work, how the model produced reincorporates them using neural networks, bypassing difficult tasks such as modeling nonlinearities both in the source and the propagation medium.

For the derivation, we follow the path presented in Clelland et al. (2013), that is a refinement of a common approach found in the literature (e.g. Coulson, 1977; Fletcher, 1998; Garrett, 2017), where little more than Newton's second law and rudimentary calculus knowledge is needed; for a more involved derivation, based on strength of materials theory, that considers, for example, Young's modulus and the shear force acting on the string, we refer the reader to Mandel et al. (1975).

The derivation presented here arises from the observation of the behavior of transverse waves in a string. It is also interesting to note that, as shown in Morse et al. (1987) and in much more detail in Bruneau (2006), the same result can be obtained using the theory of fluid mechanics, starting from the analysis of plane, longitudinal waves, traveling in the same direction of the propagation in a tube with a uniform cross-section in the first case, and the theory of wave propagation in the second case.

Considering the infinitesimal segment of an uniform string with linear density ρ shown in Figure 8, and the forces acting on it to restore the balance after a perturbation, and assuming that each point of the string moves only in the vertical direction, one can therefore interpret $y(x, t)$ as the scalar, perpendicular distance from the horizontal direction to a point on the string as a function of the point's abscissa x and the time t.

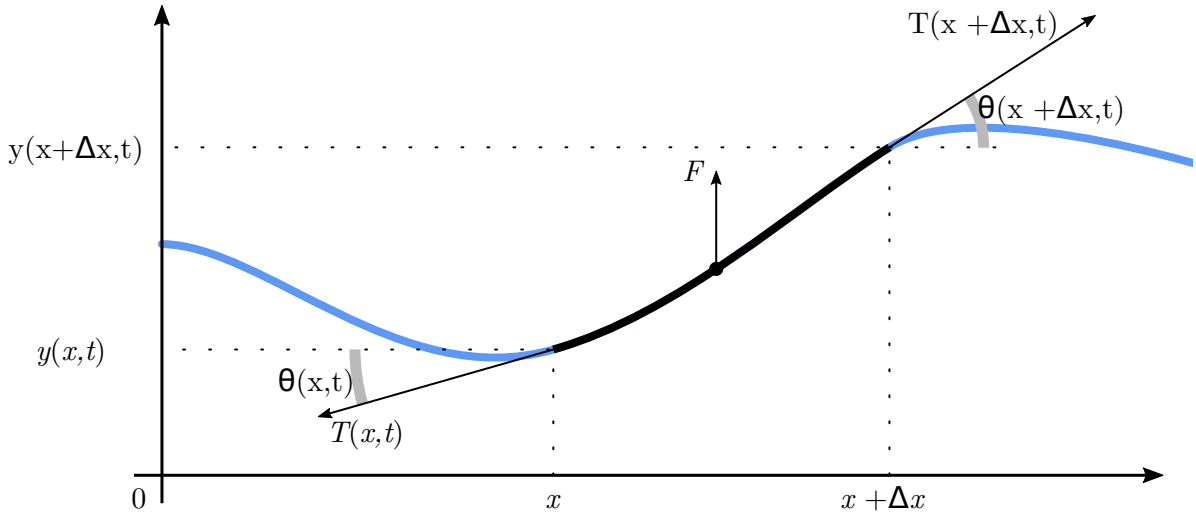

Figure 8: An infinitesimal segment of a long string with constant density.

From Newton's second law, the resulting (vertical) force F acting on the string segment between x and $x + \Delta x$ can be described as the mass $\rho\Delta x$ of the segment times its acceleration $\frac{\partial^2 y(x+\Delta/2,t)}{\partial t^2}$ and must be equal to the resultant of the tension, that is, $T(x + \Delta x, t)\sin(\theta(x + \Delta x, t)) - T(x, t)\sin(\theta(x, t))$, enabling one to write Equation 2. Additionally, the resulting horizontal force acting on the segment must be zero and we can thus write Equation 3.

$$T(x + \Delta x, t)\sin\left(\theta(x + \Delta x, t)\right) - T(x, t)\sin\left(\theta(x, t)\right) = \rho\Delta x\frac{\partial^2 y(x + \frac{\Delta x}{2}, t)}{\partial t^2} \tag{2}$$

$$T(x + \Delta x, t)\cos\left(\theta(x + \Delta x, t)\right) - T(x, t)\cos\left(\theta(x, t)\right) = 0 \tag{3}$$

For small values of Δx, due to stiffness, we can assume that the string segment is a straight line and say that $\cos\left(\theta(x, t)\right) = \cos\left(\theta(x + \Delta x, t)\right) = (\Delta x/2)/(\text{hypotenuse}/2) = \Delta x/\text{hypotenuse}$, $\sin\left(\theta(x, t)\right) = \left(y(x + \Delta x/2, t) - y(x, t)\right)/(\text{hypotenuse}/2)$ and that $\sin\left(\theta(x + \Delta x, t)\right)$ $\left(y(x + \Delta x, t) - y(x + \Delta x/2, t)\right)/(\text{hypotenuse}/2)$, with hypotenuse $\approx \Delta x/2$.

Subtracting Equation 3 from Equation 2, performing the above substitutions and dividing the resulting equation by Δx yields Equation 4:

$$\frac{2T(x,t) - 2T(x+\Delta x,t) + \frac{4T(x+\Delta x,t)\left(y(x+\Delta x,t) - y(x+\frac{\Delta x}{2},t)\right) - 4T(x,t)\left(y(x+\frac{\Delta x}{2},t) - y(x,t)\right)}{\Delta x}}{\Delta x} = \rho\frac{\partial^2 y(x+\frac{\Delta x}{2},t)}{\partial t^2} \tag{4}$$

After taking the limit of Equation 4 when Δx tends to zero we have Equation 5.

$$T(x,t)\frac{\partial^2 y(x,t)}{\partial x^2} + 2\frac{\partial T(x,t)}{\partial x} - 2\frac{\partial T(x,t)}{\partial x}\frac{\partial y(x,t)}{\partial x} = \rho\frac{\partial^2 y(x,t)}{\partial t^2} \tag{5}$$

Assuming constant tension, such that $T(x,t) = T$, both in time and along the string, it is possible to rewrite Equation 5 in the form shown in Equation 6, since all the partial derivatives of the tension are now zero.

$$\frac{\partial^2 y(x,t)}{\partial t^2} = \frac{T}{\rho}\frac{\partial^2 y(x,t)}{\partial x^2} = c^2\frac{\partial^2 y(x,t)}{\partial x^2} \tag{6}$$

Equation 6 is a second-order linear partial differential equation known as the wave equation, responsible for describing transverse waves in a string using the framework of classical mechanics. The quantity $c = \sqrt{\frac{T}{\rho}}$ represents the speed of the wave propagation, as it moves along the string.

d'Alembert was responsible not only for this formulation, setting the physical units in a way that makes $c = 1$, but for the introduction of the concept of differential equations as a new field inside calculus, in order to deal with oscillatory problems (Oliveira, 2020).

He also proposed a general solution for this equation of the form $y(x,t) = f(x+t) + g(x-t)$, and applied the boundary conditions $y(0,t) = y(L,t)$, where L is the length of the string, to arrive at $y(x,t) = f(x+t) + f(x-t)$, where f is a periodic, odd, and everywhere differentiable arbitrary function (Wheeler et al., 1987).

The concept of differential equations attracted the interest of Euler who, besides expanding the general theory of differential equations, offered a different take on Equation 6 (Demidov, 1982), explicitly introducing c in the equation, and becoming thus responsible for its most recognizable form, as shown in Equation 7, as well as its association with the plucked string.

$$\frac{\partial^2 y(x,t)}{\partial t^2} = c^2\frac{\partial^2 y(x,t)}{\partial x^2} \tag{7}$$

With the introduction of c, the general solution proposed by Euler takes the form $y(x,t) = f(x+ct)+g(x-ct)$ and, applying the same boundary conditions $y(0,t) = y(L,t)$, gives rise to Equation 8.

$$y(x,t) = f(x+ct) + f(x-ct) \tag{8}$$

Received with controversy, however, was the new characterization of f: Euler argued that f could be any freeform curve in the interval between 0 and the length L of the string, provided this curve extends itself, with odd periodicity, along the whole real line (Wheeler et al., 1987).

We are tributary to Bernoulli and, later, Lagrange, for the rudiments of the solution in the form of an infinite sum of sinusoids (Wheeler et al., 1987). Although this approach introduced by Bernoulli would, about half a century later, be corroborated by Fourier via the principle of superposition, it was received with skepticism by both Euler, that argued that this approach couldn't be general enough to represent arbitrary curves, and d'Alembert, under the argument that only one mode of vibration could physically exist.

It is interesting to note that, from a mathematical perspective, the Fourier theory can be seen as a natural development of this discussion, it was revolutionary from a physical and, ultimately, acoustical perspective.

Until this point, we have followed a very long and incremental path of investigation, initiated in antiquity by Euclid with the conceptualization of the monochord, an instrument designed mostly as a thought experiment consisting of a single string and a movable bridge. This idealized instrument served as the theoretical platform for the investigation of sound for a couple of millennia, and around the time of d'ALambert, Euler and Bernoulli, its weaknesses were becoming more evident (Lostanlen et al., 2019).

The term timbre, for example, first appeared around that time, coined by Jean-Jacques Rousseau in his entry about the three main characteristics of sound in Diderot's Encyclopédie (Diderot et al., 2021), published in 1765. It was clear to him that the timbre wasn't directly related to the pitch or the intensity of a sound, the aspects the monochord was designed to explain, but was nevertheless important, as it permitted to distinguish, for example, the same note, at the same intensity, played on a flute or on a guitar (Dolan, 2013).

The Encyclopédie had d'Alembert as other of its editors, and Rousseau, both in his entry and directly, communicated to him the necessity of improvements in the theory of

sound in order to account for the timbre. This motivated the publication of the Elements of Music (d'Alembert, 1752), where d'Alembert attempts to move away from the classical, monochord-based, Greek paradigm, to establish a more complete description of sound.

This updated description was to begin to take shape, albeit indirectly, with the publication of Fourier's Analytical Theory of Heat (Fourier, 1822), and the birth of harmonic analysis. The work of Fourier, when interpreted through the lens of acoustics, enables the decomposition of an arbitrary sound signal into a sum of sinusoids (Lostanlen et al., 2019). Consider a continuous, possibly complex, signal $y(t)$ as a function of time $t \in \mathbb{R}$, with $-\infty \leq t \leq \infty$. We have that:

$$Y(f) = \mathcal{F}\{y(t)\} = \int_{-\infty}^{\infty} y(t)\mathrm{e}^{-2\pi ft\mathrm{i}}\mathrm{d}t \tag{9}$$

$$y(t) = \mathcal{F}^{-1}\{Y(f)\} = \int_{-\infty}^{\infty} Y(f)\mathrm{e}^{2\pi ft\mathrm{i}}\mathrm{d}f. \tag{10}$$

$Y(f)$ and $y(t)$ are often referred to as a Fourier pair, in the sense that they represent the same entity in two different forms. A discussion of the conditions under which this identity holds can be found in Stein et al. (2003), while its derivation and proof are available at D'Angelo (2013).

The discrete-time Fourier transform (DTFT), conceptually, consists of applying the Fourier transform to a discrete signal, generating a function continuous in frequency. Mathematically, it is an operator that maps a discrete signal to a function that takes a real value, representing a frequency, and returns a complex value, that can be interpreted as this frequency's amplitude and phase (Staff, 2008), in the context of the discrete signal. Sampling the DTFT at equally spaced intervals in frequency corresponds to the DFT (Oppenheim et al., 2013), the discrete version of the Fourier transform in both domains.

In the context of this work we are more interested in the DFT, the transformation shown in Equation 11, and its inverse, shown in Equation 12. In those equations n and k are discrete quantities belonging to the set of positive integers, analogous, respectively, to the continuous variables t and f in Equations 9 and 10.

Considering $x_0, x_1, \cdots, x_{N-1}$, a finite sequence of N possibly complex numbers that could, for example, be obtained by sampling a continuous signal, such as $y(t)$, at regular intervals. Its frequency-domain representation can be obtained using Equation 11.

$$X_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n \mathrm{e}^{-2\pi kni/N} \tag{11}$$

Conversely, the original signal can be retrieved from any sequence of the form $X_{iN}, X_{iN+1}, \cdots, X_{iN+N-1}$, with $i \in \mathbb{N}$, since the frequency domain representation is periodic with period N. The sequence $X_0, X_1, \cdots, X_{N-1}$ of N complex numbers, obtained when $i = 0$, will be considered in the present work the default frequency-domain representation, and its conversion back to the time domain is shown in Equation 12.

Equations 11 and 12 are almost identical, except for the sign of the term $2\pi kni/N$ and the factor $\frac{1}{N}$ multiplying, in this case, Equation 11. While this factor is more commonly found multiplying the inverse discrete Fourier transform (IDFT) some authors, such as Bracewell (2000) and Kammler (2008), prefer this presentation (Amidror, 2013). This work adopts this less popular convention since it enables the amplitude of the individual sinusoids that form the signal to be derived directly from the absolute value of the respective entry in the frequency-domain representation.

$$x_n = \sum_{k=0}^{N-1} X_k \mathrm{e}^{2\pi kni/N} \tag{12}$$

Figure 9 illustrates the DFT concept: the red signal in the figure, representing a complex signal, is obtained by the summation of the multi-colored sinusoids, each one with a particular frequency and phase, whose individual amplitudes can be seen in the blue line on the figure.

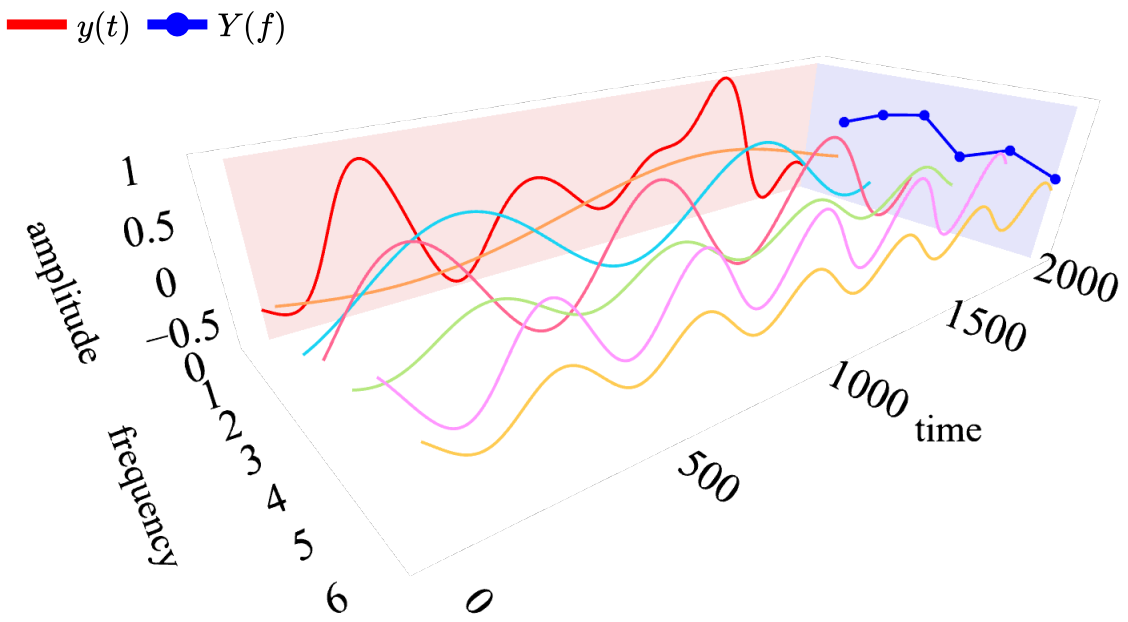

Figure 9: A complex signal, in red, decomposed into its sinusoidal components. The amplitude of each component can be seen in the blue line.

Since we are concerned, in this work, with purely real signals, it is appropriate to introduce some modifications to the aforementioned representations, to put the transform in a more straightforward form, for the reason that implementations designed to deal exclusively with real signals can offer better efficiency (Ersoy et al., 1988).

Noting that $\mathrm{e}^{2\pi kn\mathrm{i}/N} = \cos(2\pi kn/N) + \mathrm{i}\sin(2\pi kn/N)$, and that $X_k \in \mathbb{C}$ is a complex number that can, thus, be written as $R_k + \mathrm{i}I_k$, with $R_k, I_k \in \mathbb{R}$, Equation 12 can be rewritten as

$$x_n = \sum_{k=0}^{N-1} \left(R_k + \mathrm{i}I_k\right)\left(\cos(2\pi kn/N) + \mathrm{i}\sin(2\pi kn/N)\right)$$

and subdivided into two summations:

$$x_n = \sum_{k=0}^{N-1}\left(R_k\cos\left(\frac{2\pi kn}{N}\right) - I_k\sin\left(\frac{2\pi kn}{N}\right)\right) + \mathrm{i}\sum_{k=0}^{N-1}\left(I_k\cos\left(\frac{2\pi kn}{N}\right) + R_k\sin\left(\frac{2\pi kn}{N}\right)\right).$$

If $x_n \in \mathbb{R}\ \forall\ n$, the imaginary part of the equation must be zero, and the equation can be reduced to a purely real summation, such as

$$x_n = \sum_{k=0}^{N-1}\left(R_k\cos\left(\frac{2\pi kn}{N}\right) - I_k\sin\left(\frac{2\pi kn}{N}\right)\right).$$

This equation can be represented more compactly, however, if we consider $a_k = \sqrt{{R_k}^2 + {I_k}^2} = |X_k|$ and $\phi_k = \tan^{-1}(I_k/R_k) = \arg(X_k)$, where arg is a function that returns the argument of a complex number:

$$x_n = \sum_{k=0}^{N-1} a_k \cos\left(\phi_k + \frac{2\pi k n}{N}\right). \tag{13}$$

This representation more closely matches the intuition presented in Figure 9 and, as will be seen, the synthesis algorithm presented in Section 5.2.

3.2 RELATION BETWEEN CONTINUOUS AND DISCRETE WAVES

It is important at this point to establish a correspondence between a continuous and a discrete wave, as the latter will be the object of most of this work. To do that we will define a simple continuous sinusoidal wave of the form $s(t) = a\cos(\phi + 2\pi f t)$, where $a \in \mathbb{R}_{\geq 0}$ is the amplitude, $\phi \in \mathbb{R}_{\geq 0}, 0 \leq \phi < 2\pi$ is the phase, $f \in \mathbb{R}_{\geq 0}$ is the absolute frequency of the wave in Hz, and $t \in \mathbb{R}$ is time in seconds.

The cosine function is used for twofold reasons: its use helps to underline the periodic nature of the signal and establishes a parallel with Equation 13. It is important to keep in mind, however, that this discussion applies to any arbitrary periodic function of the form $g(x) = g(x+p)\ \forall\ x, x+p \in \mathrm{dom}(g)$, where dom is the domain of the function g and $p \in \mathbb{R}$. This will be further illustrated in the discrete treatment of the problem, below.

If we were to sample $s(t)$ at regular intervals for a definite period of time, that is, to measure the value of the function with a constant sampling rate of f_s Hz, neglecting errors of measurement, we would have a finite discrete representation of part of the continuous wave of the form $x_n = a\cos\left(\phi + 2\pi k \frac{n}{N}\right)$ where, alongside the quantities introduced before, we would have $n \in \mathbb{N}$ as the sample index; $N \in \mathbb{N}$ as the total number of discrete samples obtained and $k \in \mathbb{R}_{\geq 0}$ as the local frequency of our discrete wave, in cycles per the total number of samples N.

The relative, local frequency k of the discrete representation, which denotes how many cycles the discrete signal went through in its total N samples, is related to the absolute frequency f, in Hz, of the continuous signal via the Equation $k = \frac{N}{f_s} f$ while the absolute time in the continuous case can be obtained from the sample number n of the discrete signal via $n = t f_s$. We are now in a position to conceptually use the continuous

or discrete version of a wave, having pointed the way in which they are related.

3.3 A GEOMETRIC APPROACH TO ENVELOPE ESTIMATION

In this section, we address the difficulties exposed in Section 2.2, by formulating a general envelope detection algorithm that uses intrinsic characteristics of a signal to estimate its envelope, completely automating this process and thus eliminating the necessity of parameter tuning. The contents of this section were published, with some modifications, in Tarjano et al. (2022b).

The method draws inspiration from geometric concepts: the Individual samples comprising the discrete signal are mapped to a set of points in the Cartesian plane, where a new measure of discrete curvature is used to identify the samples that are part of the envelope. Due to its geometric nature, the method can optionally generate the superior and inferior envelopes, or frontiers, of a given wave.

As will be seen in Section 5.1.2, the algorithm compares favorably with classic envelope detection techniques based on smoothing, filtering, and the Hilbert transform, besides being physically plausible, as proved in Section 3.3.6.

We provide visualizations of the envelope extracted for various real-world signals with diverse characteristics, such as voice — spoken and sung — and pitched and unpitched musical instruments, and discuss some approaches to objectively assess the quality of the obtained envelopes. An optimized implementation is available via the Python Package Index (PyPI), while interactive visualizations and source code are available from the accompanying GitHub repository (Tarjano, 2020).

The Python module implementing the algorithm can be installed with the command “pip install signal-envelope”. Those working under Windows 64-bit machines can benefit from a specialized dynamic library implemented in C++, automatically used within the module, when applicable. This library can also be obtained separately via the repository’s release section.

3.3.1 Formalizing the problem of envelope detection

In general, the problem of envelope detection can be interpreted as the task of, given a continuous wave $w(t)$, decomposing it in two waves $e(t)$ and $c(t)$, such that $w(t) = e(t)c(t)$ (Turner et al., 2011).

$e(t)$ represents the slow varying part of the wave, also known as the (temporal) envelope, modulator component, or amplitude modulation (AM) of $w(t)$, while $c(t)$ models

its fast-varying part, called throughout the literature as its (temporal) fine structure, carrier wave, or frequency modulation (FM).

In the case of broadband signals, the problem of envelope detection of an arbitrary wave is ill-posed, in the sense that an infinite number of pairs of $e(t)$ and $c(t)$ can result in the same $w(t)$ (Turner et al., 2011; Loughlin et al., 1996).

This work addresses this problem by assuming $c(t)$ to be normalized between $\{-1, 1\}$ such that $|\max(c(t))| = 1$. This assumption does not cause any loss of generality, since, given an arbitrary $c(t)$, we can obtain its normalization by dividing the function by its absolute global maximum, that is, $\hat{c}(t) = c(t)/\max(|c(t)|)$, provided that $\max(|c(t)|) \neq 0$.

In this work we are concerned with the discrete version of the envelope detection problem, that is, given a finite discrete wave represented by the vector $\mathbf{w} \in \mathbb{R}^N$, obtaining the temporal envelope $\mathbf{e} \in \mathbb{R}^N$ of $\mathbf{w}$.

The continuous definitions can be translated to this discrete scenario assuming that the discrete quantities arise from observing the continuous ones at regular time intervals, for a finite period of time. In that case, as noted in Section 3.2, the equality $n = tf_s$ can be used to link both settings, where n is the sample index of the discrete signal, t stands for the time in seconds, and f_s is the sampling rate, or the number of observations of the continuous signal made in one second. We can thus define:

Given:

$$\mathbf{w} = (w_0, w_1, \cdots, w_{N-1}), \mathbf{w} \in \mathbb{R}^N$$

We define:

$$\begin{aligned} &\mathbf{e}, \mathbf{c} \in \mathbb{R}^N \\ &\mathbf{e} \triangleq (e_0, e_1, \cdots, e_{N-1}), e_n \geq 0 \ \forall \ n, \ 0 \leq n \leq N-1 \\ &\mathbf{c} \triangleq (c_0, c_1, \cdots, c_{N-1}), |c_n| \leq 1 \ \forall \ n, \ 0 \leq n \leq N-1, |\max(\mathbf{c})| = 1 \end{aligned} \tag{14}$$

Such as:

$$\mathbf{w} = \mathbf{e} \odot \mathbf{c} \ \ \therefore \ \ w_n = e_n c_n \ \forall \ n, \ 0 \leq n \leq N-1.$$

The $\odot$ operator in Equation 14 stands for the Hadamard product, and denotes elementwise multiplication of two vectors. Figure 10 provides an example of the vectors just defined: $\mathbf{w}$ is obtained by elementwise multiplication between the carrier wave $\mathbf{c}$ and the envelope $\mathbf{e}$.

Equation 14 connects the three vectors **e**, **w**, and **c** in the sense that each one can be derived from the other two. In most real-world scenarios, however, only the original signal **w** is known, and the envelope **e** is to be retrieved without a priori knowledge of the carrier wave **c**.

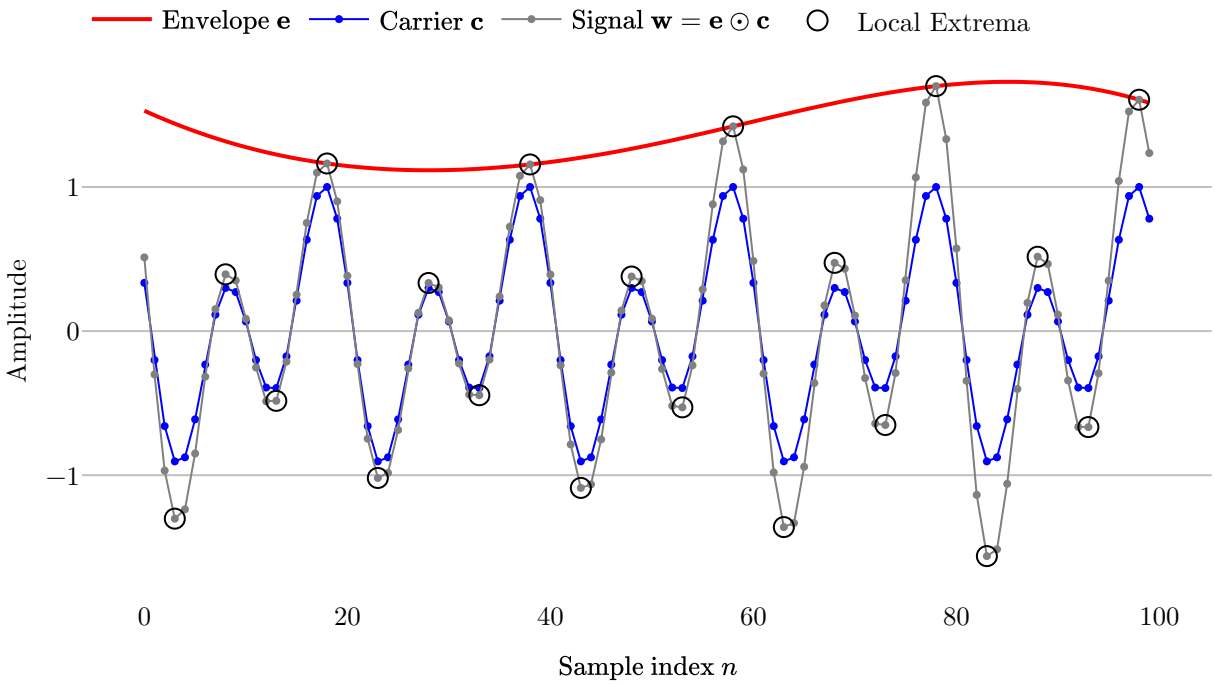

Figure 10: Example of a discrete wave **w** arising from the elementwise multiplication of an envelope **e** and a carrier **c**, both previously known. The local extrema of **w** are highlighted with a circle.

In this context, Figure 10 helps to illustrate a key concept: only at some local extrema the envelope can be inferred without prior knowledge of the underlying carrier.

Consider the point $w_n = c_n e_n$, with c_n as a local maximum of **c**, and assume $c_n, e_n > 0$. A point close to w_n can be written as $w_n + \Delta w = (c_n - \Delta c)(e_n + \Delta e)$. We can assume $\Delta e \approx 0$, since the rate of change of the envelope is considerably smaller than that of the carrier. Making $\Delta c > 0$ guarantees the assumption that c_n is a local maximum.

Expanding this equation gives $w_n + \Delta w = -\Delta c e_n + c_n e_n$, where the term $c_n e_n$ is equal to w_n and can be canceled. Since $\Delta c, e_n > 0$, we conclude that $\Delta w < 0$, and w_n must be a local maximum of **w**.

Hence, the local extrema of the carrier wave **c** and the original signal **w** coincide. Some of those local extrema are also global extrema and in those the absolute value of the carrier wave, by definition, is 1, and we have that $w_n = e_n$, meaning that, at those points, the envelope is equal to the instantaneous amplitude of the signal.

This realization that the envelope coincides with the instantaneous amplitude of

the signal only at some of the signal's local extrema is important, as it can be used to generate a simplified representation of the signal, making the algorithm more efficient.

Note that, in Figure 10, the carrier, the original signal, and even the envelope are representations of similar entities: signals, in the same coordinate system. This is in line with the work of Caetano et al. (2011), Turner et al. (2011), and Richard Lyons (2017), but in contrast with some representations found in the literature of amplitude modulation (e.g. Loughlin et al., 1996), where the FM and AM of a signal are represented in different coordinate systems: frequency versus time and amplitude versus time, respectively.

3.3.2 Simplified representation of a signal for envelope detection

In order to introduce this representation, it is appropriate to define the term pulse as it will be used in the present work: each time the value of w_n changes sign, that is, every time the instantaneous amplitude of the discrete wave $\mathbf{w}$ crosses the horizontal axis, the beginning of a pulse is defined, with the next crossing defining its end. This interpretation of a pulse is in line with the one presented in Division et al. (1996), where a pulse is defined as a rapid change in the amplitude of a signal, followed by a fast return to the baseline value; zero, in our case.

From Figure 10 and the discussion in Section 3.3.1, we can see that, if the envelope touches a pulse, it must pass through the maximum point of a positive pulse or the minimum point of a negative pulse, making those our main points of interest. We can then proceed, for the rest of the method, considering only those points, to a considerable computational economy.

For this reason, we define P as the set of the points $\{P_0, P_1, \cdots, P_{M-1}\}$, each point associated with a pulse, where $P_i = (n_i, |w_{n_i}|)$, that is, $|w_{n_i}|$, the absolute value of the extremum of each pulse, becomes the ordinate of each point and n_i, its original sample index, becomes the abscissa. This is illustrated in Figure 11.

While the exact relationship between N, the number of samples of the original signal, and M, the cardinality of the set P, is dependent on factors such as the frequency and sampling rate of the original wave, the inequality $M \leq N$ holds for all discrete signals. In practice, $M \ll N$ for most real-world signals.

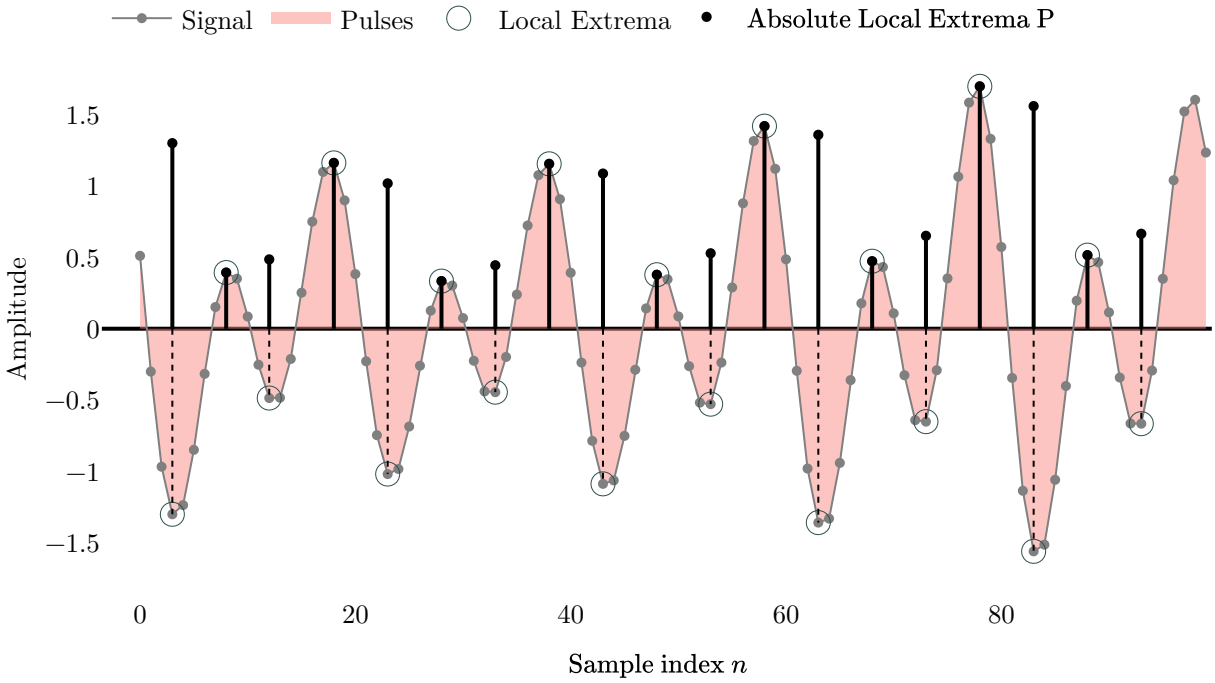

Figure 11: Example of a discrete wave $\mathbf{w}$ divided into pulses, of which the extrema are highlighted with a circle. P is the set of the points $P_i = (n_i, |w_{n_i}|)$ representing the absolute value of those extrema.

3.3.3 Mapping to the Cartesian coordinate system

The points in P are not fit for geometric interpretation yet, since the abscissa and ordinate of their orthogonal coordinate system have different units: in the vertical axis, we have a unit related to the instantaneous amplitude of the wave, while in the horizontal axis we have the index at which each extremum occurs, ultimately a time-related unit.

We can think of these points as lying in a vector space with a non-standard, orthogonal basis vector $\mathbf{b} = \{\mathbf{b}_x, \mathbf{b}_y\}$ with $\mathbf{b}_x = (\Delta n, 0), \mathbf{b}_y = (0, \Delta a)$, where Δa is a pseudo unit related to the amplitude of a sample and Δn a pseudo unit related to the sample index. The exact magnitude of those pseudo units is not important here.

We are interested in representing the points in P in the Cartesian coordinate system, and for that we need a transformation that allows those points to be described in the standard basis $\mathbf{s} = \{\mathbf{s}_x, \mathbf{s}_y\} = \{(1, 0), (0, 1)\}$ of this vector space.

The general idea is to represent each coordinate as a fraction of the average of all coordinates in the same direction. To achieve that, we divide the ordinate of each point by the average of all ordinates, effectively canceling the unit of the vertical components of the points. If we were to proceed similarly regarding the horizontal components, the direct numerical link between P and $\mathbf{w}$ would be lost.

It is best to leave the abscissas intact and multiply the now unitless vertical coordinates of each point by the average of the difference between the abscissa of a point and the abscissa of the immediately posterior point.

In this way, we can consider both axes as having unit Δn. By dividing both of them by Δn, we obtain the effect of representing the points on the standard basis. The relationship is preserved, since the abscissas of the points thus transformed are numerically equivalent to the indices n of $\mathbf{w}$, making it straightforward to recover the envelope points in the original coordinate system.

Considering $P_{\mathbf{b}}$ a point of the set P represented in the original coordinate system and $P_{\mathbf{s}}$ the same point represented in the Cartesian coordinate system, we formalize this transformation in Equation 15:

$$\begin{aligned} P &= (x, y) \\ \beta &= \frac{\left(\frac{n_{M-1}-n_0}{M-1}\right)}{\left(\frac{\sum_{i=0}^{M-1}|w_i|}{M}\right)} \\ P_{\mathbf{b}} &= x\ \mathbf{b}_x + y\ \mathbf{b}_y = x(\Delta n, 0) + y(0, \Delta a) = (x\Delta n, y\Delta a) \\ P_{\mathbf{s}} &= x\ \mathbf{s}_x + \beta\ y\ \mathbf{s}_y = x(1,0) + \beta y(0,1) = (x, \beta y) \\ &\forall\ P \in \mathrm{P} \end{aligned} \tag{15}$$

The effect of this transformation can be seen in the points represented in Figure 12. Despite the two bases being different, the horizontal relationship is preserved, since one unit along the horizontal axis in Cartesian coordinates corresponds to one unit of the sample index n in the original coordinate system.

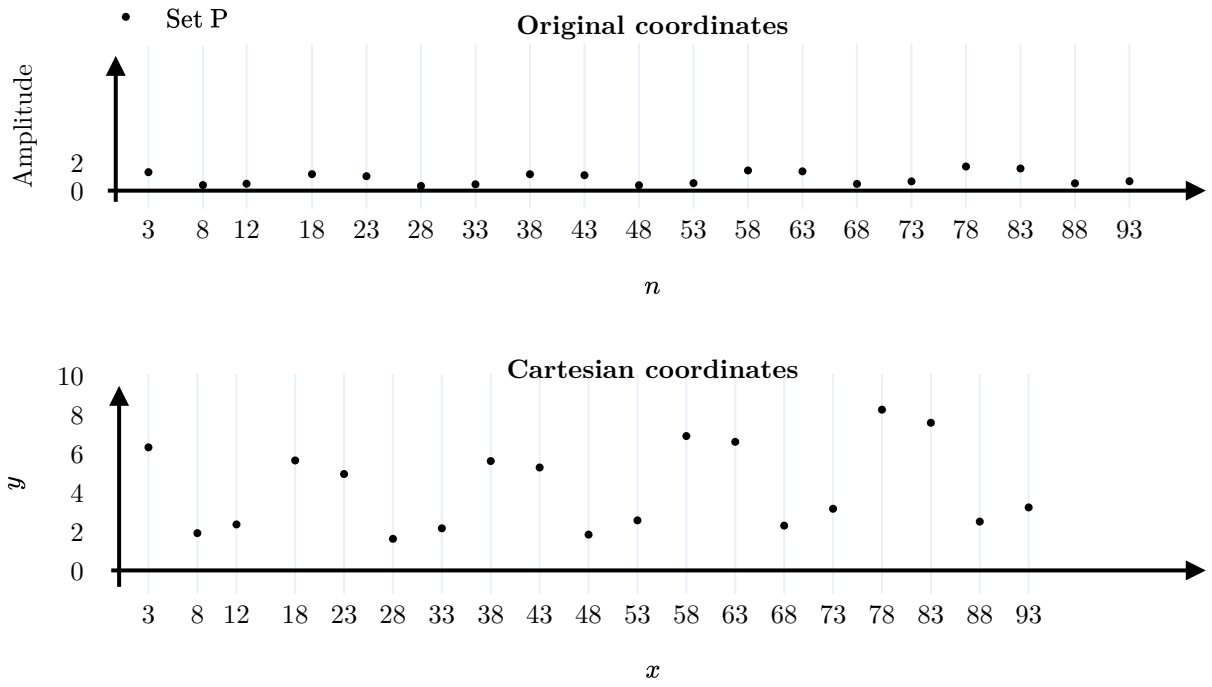

Figure 12: The set P of points, in both the original (top) and Cartesian (bottom) coordinate systems, shown in scale. Correspondence is maintained between both horizontal axes, in order to simplify the process of retrieving the algorithm results.

3.3.4 Discrete curvature estimation

Having established a transformation from the original space of the two-dimensional representation of a discrete wave to the Cartesian coordinate system, we are now able to apply the geometric concept of the shape of a set of points.

Algorithms such as the convex hull and the alpha shapes solve this problem by discriminating which points of a set are part of its frontier.

A formal definition of alpha shapes is given in H. Edelsbrunner et al. (1983). In the context of this work, however, gaining intuition about the algorithm is more important. For that the explanation presented in Herbert Edelsbrunner et al. (1994) can be more insightful.

The explanation, translated to our context, invites one to think of the plane where the points in the set P are represented as a soft material, such as Styrofoam. The points themselves are made of a harder material, such as rock, and occupy a fixed position in this plane. The alpha shapes algorithm can be seen as carving this plane from the outside of those points with a circular tool of fixed radius α. At the end of the process, the remaining Styrofoam that could not be removed due to the impeding rocks is the alpha shape of this set of points.

Alternatively, one can picture a circle of fixed radius being rolled above the points in the set P, marking the points it touches as part of the envelope. A video illustrating this is available at `envelope.netlify.app`.

To employ an alpha shape inspired algorithm, however, the radius r of such a circle must be estimated, and for that, a measure of discrete curvature is needed.

Discrete curvature estimation is an important task in image processing (Fleischmann et al., 2010), for which no default algorithm exists.

The two general approaches are the derivation of direct methods that use characteristics of the discrete curve to calculate the curvature, or the use of the curvature of a smooth, continuous curve fitted to the original discrete curve (Coeurjolly et al., 2001). Since we are dealing with a potentially large set of points, it is important for this measure to act locally, to assure $O(N)$ complexity; this is generally not the case for approaches involving curve fitting.

Concerning direct methods, Carroll et al. (2014) presents three such definitions, based on the approximation of a circle by an inscribed, centered, and circumscribed polygon. In the context of three-dimensional meshes, Libor Váša et al. (2016) evaluate a range of existing estimators, from a multivariate point of view.

Those approaches define the discrete curvature in relation to the vertices of a discrete curve (or mesh, in the three-dimensional case). Since we are interested in the change of direction — what the term curvature ultimately means — between two adjacent points, a definition of discrete curvature over the edges of a discrete curve would be preferable.

Thus, the set V of the vectors from each point of P to the next, formalized in Equation 16 and illustrated in Figure 13, will be used to estimate the average curvature of P: we will calculate the equivalent circle for each vector, and average the values obtained to infer the curvature of the set.

$$\mathrm{V} = \{v_0, v_1, \cdots, v_{M-2}\} \quad \text{where} \quad v_i = P_{i+1} - P_i \;\forall\; 0 \leq i \leq M-2 \tag{16}$$

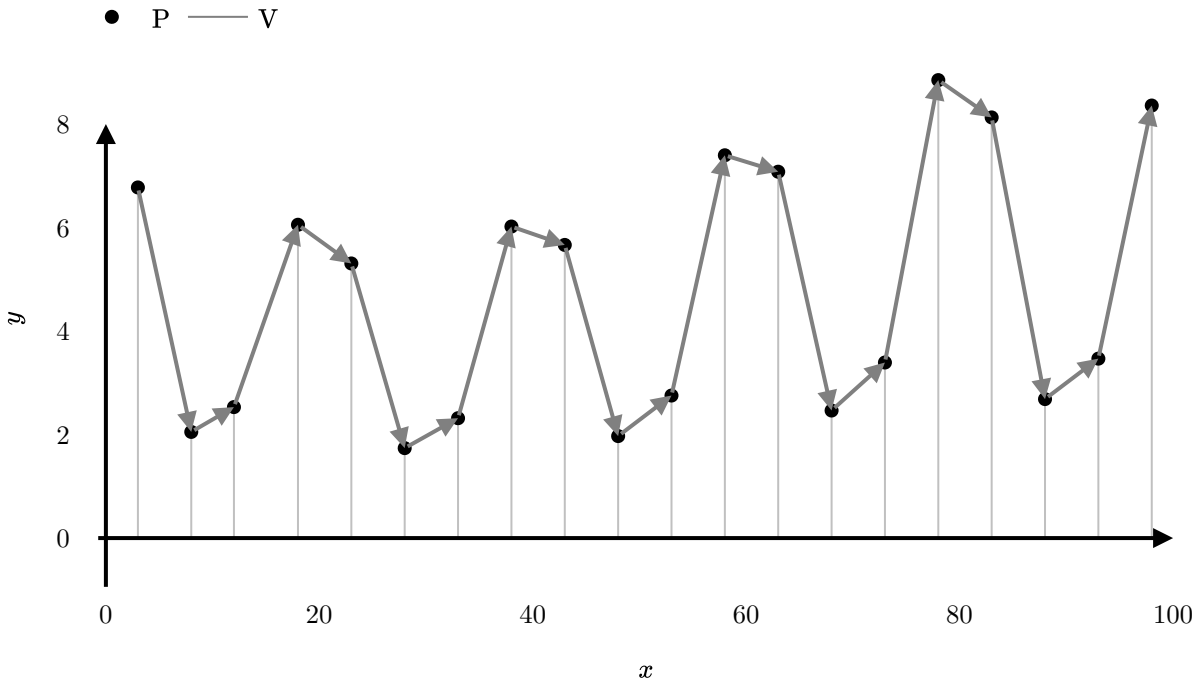

Figure 13: The set P of points and the set V of vectors between two adjacent points of P, in the Cartesian coordinate system.

3.3.4.1 The Equivalent Circle Approach to discrete curvature estimation

We thus proceed to define a discrete curvature measure over the edges of a discrete curve. To that end, we are going to apply the definition of smooth curvature as the rate of change of the unit tangent to a curve. This definition is convenient, since it is equivalent to that of the osculating circle (L. Váša et al., 2016), the value we are interested in finding.

One of those edges, in the form of the vector $v_i \in \mathrm{V}$, is singled out in Figure 14, and represents the envelope's displacement from the extrema $P_i = (x_i, y_i)$ to $P_{i+1} = (x_{i+1}, y_{i+1})$.

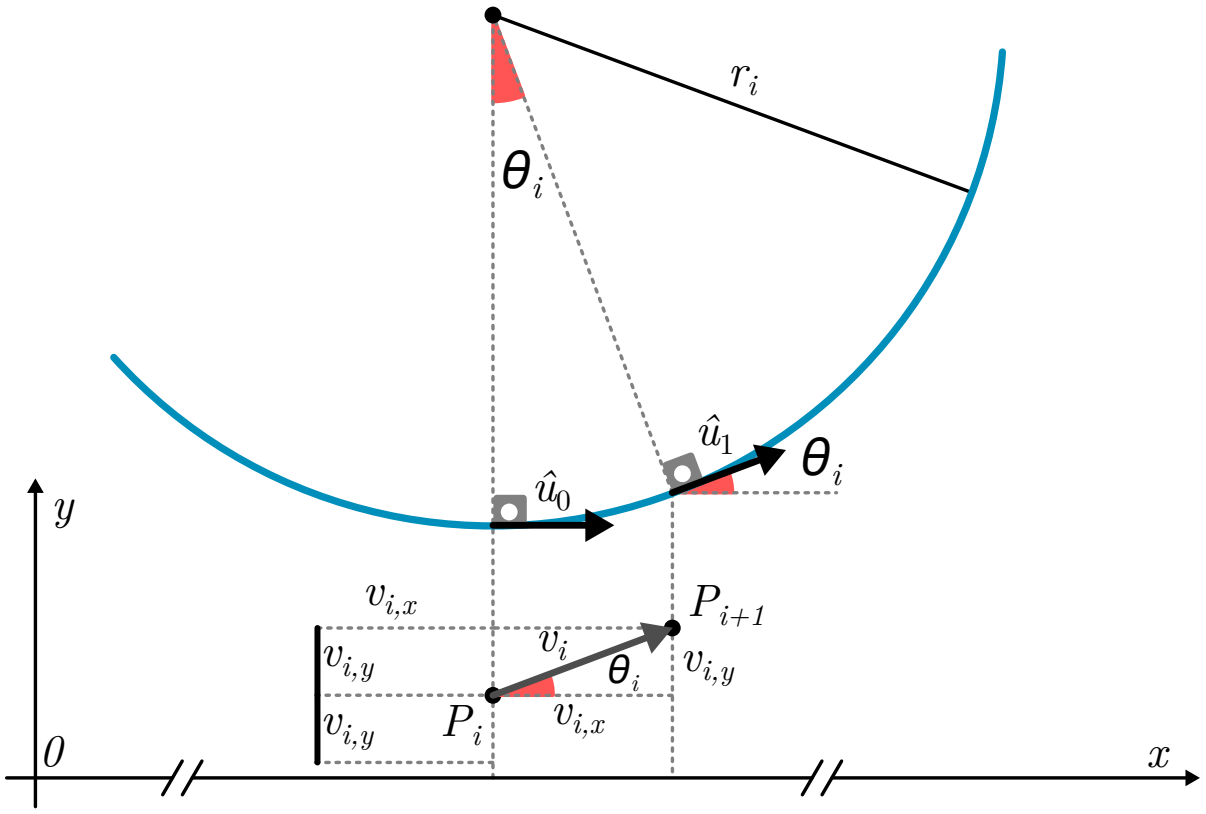

Figure 14: The unit vector tangent to the circle changes from the horizontal direction in $\hat{u}_0$ to an inclination of θ_i in $\hat{u}_1$, θ_i being the angle that vector v_i makes with the horizontal direction.

We assume that, near vector v_i, the envelope defines a vertical displacement $v_{i,y} = y_{i+1} - y_i$ and a horizontal displacement $v_{i,x} = x_{i+1} - x_i$. Since we do not yet know which points of P are part of the envelope, we cannot be certain that the envelope passes through P_{i-1} before reaching P_i, or even if it ever reaches P_i.

Nevertheless, assuming it passes through P_i, and is limited to the displacements just observed, the envelope might have passed anywhere in the line segment from points $(x_i - (x_{i+1} - x_i), y_i - (y_{i+1} - y_i))$ to $(x_i - (x_{i+1} - x_i), y_i + (y_{i+1} - y_i))$, the bold line segment before v_i in Figure 14, before reaching P_i.

Generalizing those observations for all vectors $v_i \in$ V, we can consider that, on average, the envelope approaches P_i horizontally and smoothly changes from the horizontal direction to the direction of the vector v_i.

The rationale then is to find, for each vector v_i, the radius r_i of the equivalent circle whose unit tangent performs the same change in direction, in the same horizontal distance. The average curvature of P will be obtained by the average of all radii r_i.

From Figure 14 one can see that $r_i = v_{i,x}/\sin(\theta_i)$, where $v_{i,x}$ is the component of v_i in the horizontal direction; observing v_i in the figure we see that $\sin(\theta_i) = v_{i,y}/|v_i| =$

$v_{i,y}/\sqrt{v_{i,x}^2 + v_{i,y}^2}$. Joining both equations, we can write Equation 17:

$$r_i = \frac{v_{i,x}\sqrt{v_{i,x}^2 + v_{i,y}^2}}{v_{i,y}}. \tag{17}$$

The radius r_i of the circle equivalent to each v_i is thus defined in Equation 17. This equation has the desirable property $v_{i,y} \to 0 \implies r_i \to \infty$, that is, the radius tends to infinity when there is no vertical displacement, effectively turning the circle into a line. The radius is also directly proportional to $v_{i,x}$, becoming larger in proportion to the horizontal distance between two adjacent points.

For implementation purposes, however, the formulation in Equation 17 poses a problem, as a zero could make its way into the denominator of a fraction; in the actual algorithm, the quantity calculated for each vector is the curvature itself, that is, the inverse of each radius r_i. In this way, the problem is avoided, since $v_{i,x}$ is guaranteed to be greater than zero. The average of the curvatures is then inverted in order to obtain the average radius $\overline{r}$, as illustrated in Equation 18.

$$\overline{r} = \left(\frac{\sum_{i=0}^{M-2} \left(\frac{v_{i,y}}{v_{i,x}\sqrt{v_{i,x}^2 + v_{i,y}^2}} \right)}{M-2} \right)^{-1} \tag{18}$$

3.3.5 Identifying the envelope

We need to construct an algorithm, using the radius just obtained, to identify the points that belong to the envelope. Generally, alpha shape algorithms resort to the construction of the Delaunay triangulation of the set of points, which is later filtered to contain only the outer edges. Using Equation 19, we can find the edges directly.

Let $x_o, y_o \in \mathbb{R}, y_o \geq 0$ be the coordinates of the center of a circle with radius r, that can be placed anywhere in the upper half of the coordinate system, and $P_a, P_b \in \mathrm{P}$ be two points of the set P.

For all circles of center (x_o, y_o) and radius r that can be constructed to pass through any two points of P, that is, all circles such that $(P_{a,x} - x_o)^2 + (P_{a,y} - y_o)^2 = (P_{b,x} - x_o)^2 + (P_{b,y} - y_o)^2 = r^2$, the edge from P_a to P_b will be part of the envelope, and the points P_a and P_b will be part of the set E of the points belonging to the envelope, if and only if exists one such circle defining a disc that does not contain any other point of P. Equation

19 summarizes this.

$$\begin{aligned} &\text{Given:} \\ &\qquad P_a, P_b, P_c \in \mathrm{P}, \quad P_a \neq P_c, P_b \neq P_c \quad \text{and} \quad x_o, y_o \in \mathbb{R},\ y_o \geq 0 \\ &\text{Such as:} \\ &\qquad (P_{a,x} - x_o)^2 + (P_{a,y} - y_o)^2 = (P_{b,x} - x_o)^2 + (P_{b,y} - y_o)^2 = r^2 \end{aligned} \tag{19}$$

We have that:

$$P_a, P_b \in \mathrm{E} \leftrightarrow (P_{c,x} - x_o)^2 + (P_{c,y} - y_o)^2 > r^2 \ \forall\ P_c \in \mathrm{P} - \{P_a, P_b\}$$

Algorithm 1 adopts a more straightforward approach, since our points have a sequential structure, and starts searching for a new edge belonging to the envelope from the rightmost point of the last edge identified as part of the envelope.

The algorithm follows directly from the definition in Equation 19, after noting that the first and last points of P will always be part of the envelope: since P_a can be equal to P_b, it is always possible to construct a circle containing the first, leftmost point of P, by placing the center of the circle at $(P_{0,x} - r, P_{0,y})$. The disc defined by this circle cannot contain any other point of P, since all other points have abscissas greater than $P_{0,x}$. The same is true for P_{M-1}, the last and rightmost point of P.

From P_0, our first pivot point, we construct circles passing through $P_1, P_2, \cdots, P_d$ until a circle that does not contain any point is found; P_d then becomes the new pivot point and is included in E, and this process continues until P_{M-1} is reached.

The procedure is formalized in Algorithm 1. We do not provide similar algorithmic descriptions of the preceding steps since the Python implementation available at `github.com/tesserato/envelope/blob/master/envelopePythonSource.py` can easily serve as pseudocode.

Algorithm 1 Retrieve Envelope

Input:
P $= \{P_0, P_1, \cdots, P_{M-1}\}$
$r \in \mathbb{R}, r > 0$
Define:
circle ▷ function, returns the center of a circle of a given radius passing through two points
distance ▷ function, returns the Euclidean distance between two points
P_c = point
empty = boolean
procedure RetrieveEnvelope(P, r)
    E $\leftarrow \{P_0\}$
    id1 $\leftarrow 0$
    id2 $\leftarrow 1$
    while id2 $< M$ **do**
        $P_c \leftarrow$ circle$(r, P_{\text{id1}}, P_{\text{id2}})$
        empty $\leftarrow$ **True**
        for $i =$ id2 $+ 1, i < M, i+ = 1$ **do**
            if distance$(P_c, P_i) < r$ **then**
                empty $\leftarrow$ **False**
                id2 $\leftarrow$ id2 $+ 1$
                break
            end if
        end for
        if empty **then**
            E $\leftarrow$ E $\cup$ $\{P_{\text{id2}}\}$
            id1 $\leftarrow$ id2
            id2 $\leftarrow$ id2 $+ 1$
        end if
    end while
    return E
end procedure

The set E obtained by the Algorithm 1 is a subset of P, and thus also a set of points in the Euclidean space. Recall from Section 3.3.3 that the coordinates of these points are currently scaled, in relation to the original signal, by the transformation described in Equation 15. We could invert the transformation, but since we were careful to maintain a link between the horizontal coordinates in both coordinate systems, we can simply consider the horizontal coordinates of P as the indices of the samples of the original discrete signal, and retrieve the envelope at each point directly.

The most straightforward way to transform the points of the set E into the vector $\mathbf{e} \in \mathbb{R}^N$ is via linear interpolation, the approach used in this work. It is worth noting, however, that many other procedures are available, both for the interpolation and the

smooth approximation of points. Depending on the intended use of the envelope, k-curves interpolation (Yan et al., 2017) is a specially interesting approach, since it guarantees that the maximum curvature occurs at the interpolated points.

3.3.6 Theoretical guarantees

In an effort to formalize the assessment of the quality of the amplitude and frequency modulations of a signal, Loughlin et al. (1996) proposed some theoretical conditions necessary, but not sufficient, to ensure the physical plausibility of an envelope.

We comment below on how those conditions translate to the discrete setting and subsequently prove that the algorithm proposed satisfies all the four conditions presented.

The first one states that, if a signal is bounded in magnitude, its envelope should also be bounded. Intuitively, it is easy to see that the algorithm complies with this requirement, as the envelope is composed of selected samples from the original signal itself.

More formally, we have the set P of the absolute values of the local extrema of the original signal $\mathbf{w} = (w_0, w_1, \cdots, w_{N-1})$ that contains, by definition, the global extrema of $\mathbf{w}$. The set E of the points that belong to the envelope is a subset of P. We can thus write Equation 20, with P represented in the original coordinate system:

$$\begin{aligned} &\mathrm{P} = \{P_0, P_1, \cdots, P_{M-1}\} \quad \text{where} \quad P_i = (n_i, |w_{n_i}|) \;\forall\; 0 \leq i \leq M-1 \\ &\mathrm{E} \subset \mathrm{P} \implies \max(\mathrm{E}) \leq \max(\mathrm{P}) = \max(|w_0|, |w_1|, \cdots, |w_{N-1}|) \end{aligned} \tag{20}$$

If the amplitude of the original signal is bounded, that is $\max(|w_0|, |w_1|, \cdots, |w_{N-1}|) = b \in \mathbb{R}$, not only the amplitude of the envelope is bounded, as required, but is guaranteed to have the same bound b.

The second statement dictates that if a signal has a finite frequency range, that frequency range must not be exceeded by its carrier wave. To see that this condition holds for the proposed method, consider a discrete signal $\mathbf{w} = (w_0, w_1, \cdots, w_{N-1})$, where all the absolute values of its local extrema are greater than one.

This requirement does not imply loss of generality since, for an arbitrary signal, we can assure this condition by dividing all samples by the value of the minimum absolute local extrema.

The relation $\min(\mathrm{E}) \geq 1$ thus holds, since $\mathrm{E} \subset \mathrm{P}$. Provided that one uses a method that preserves the original bounds when interpolating the points in E, such as linear inter-

polation, we can be sure that all the points in the envelope vector $\mathbf{e} = (e_0, e_1, \cdots, e_{N-1})$ are also greater than one.

Recalling Equation 14, where the relation between the original signal, the carrier and the envelope was defined as $\mathbf{w} = \mathbf{e} \odot \mathbf{c}$, we can write the magnitude $m_{\mathbf{w}}(k)$ of an arbitrary frequency of the original signal and, likewise, the magnitude $m_{\mathbf{c}}(k)$ of an arbitrary frequency of the carrier, in terms of the DFT, as in Equation 21:

$$\begin{aligned} m_{\mathbf{w}}(k) &= \sum_{n=0}^{N-1} \sqrt{(w_n \cos(2\pi kn/N))^2 + (w_n \sin(2\pi kn/N))^2} \\ m_{\mathbf{c}}(k) &= \sum_{n=0}^{N-1} \sqrt{\left(\frac{w_n}{e_n} \cos(2\pi kn/N)\right)^2 + \left(\frac{w_n}{e_n} \sin(2\pi kn/N)\right)^2}. \end{aligned} \tag{21}$$

For any given frequency k, both $m_{\mathbf{w}}(k)$ and $m_{\mathbf{c}}(k)$ are nonnegative. Also, since $e_n \geq 1 \ \forall \ n, 0 \leq n \leq N-1$, we establish that $0 \leq m_{\mathbf{c}}(k) \leq m_{\mathbf{w}}(k) \ \forall \ k$, which is a more restrictive condition than the original. If $\mathbf{w}$ has a finite frequency range, the magnitude of the frequencies outside that range are zero, as are those of the carrier $\mathbf{c}$.

The third condition states that a physically plausible algorithm should, for a signal $\mathbf{w}$ of constant amplitude and constant frequency — that can thus be defined as $a\cos(\kappa n + \phi)$ with $a \in \mathbb{R},\ a > 0$ and $\kappa \in N$ — generate a constant envelope $\mathbf{e}$ of magnitude a and a carrier wave $\mathbf{c}$ of the form $\cos(\kappa n + \phi)$. An example of such behavior can be observed in Section 5.1.2, Figure 48, where the envelope of a simple sinusoid is shown.

To see that this is always the case, note that all the extrema of a sinusoid are of the form $a(\pm 1)$ and $P_i = (n_i, |a(\pm 1)|) = (n_i, a) \ \forall \ P_i \in \mathrm{P}$ in the original coordinate system, provided that the sampling frequency is a multiple of the sinusoid's frequency. In practice, given a standard sampling rate of 44100 Hz, all extrema can be considered approximately equal to a throughout the audible spectra. As $\mathrm{E} \subset \mathrm{P}$, we have that the envelope $\mathbf{e} = (a, a, \cdots, a)$ and is, thus, constant. Likewise, since $\mathbf{w} = \mathbf{e} \odot \mathbf{c}$, we have $c_n = a\cos(\kappa n + \phi)/a = \cos(\kappa n + \phi) \ \forall \ n, 0 \leq n \leq N-1$.

The fourth and last condition states that if a wave is multiplied by a constant, its envelope should also be multiplied by the same constant. Recall from Section 3.3.3 that, before the extraction of the envelope, the original points undergo a transformation to the Cartesian coordinate system. After the extraction, the inverse operation can be performed on the points of the envelope.

Consider a discrete signal $\mathbf{w} = (w_0, w_1, \cdots, w_{N-1})$ and the set P of its absolute extrema. Represented in the Cartesian coordinate system, $P_i = (n_i, \beta|w_{n_i}|)$, where β is

obtained from Equation 15.

Given a constant $\alpha \in \mathbb{R},\ \alpha > 0$, we can define the scaled signal as $\mathbf{w}' = (\alpha w_0, \alpha w_1, \cdots, \alpha w_{N-1})$. Observing, from Equation 15, that α can be factored out of denominator of the summation, we have $\beta' = \beta/\alpha$ and the new set $\mathrm{P}' = \{P'_0, P'_1, \cdots, P'_{M-1}\}$, with $P'_i = (n_i, \frac{\beta}{\alpha}|\alpha w_{n_i}|)$, is equal to the original set P.

This guarantees that the same subset of points E will be identified as part of the envelope in both cases. In their respective original basis system, applying the inverse transformation, those points are $P_i = (n_i, \frac{\beta}{\beta}|w_{n_i}|) = (n_i, |w_{n_i}|)\ \forall\ P \in \mathrm{E}$, for the original signal, and $P'_i = (n_i, \frac{\alpha}{\beta}\frac{\beta}{\alpha}|\alpha w_{n_i}|) = (n_i, |\alpha w_{n_i}|)\ \forall\ P' \in \mathrm{E}'$ for the scaled signal.

3.3.7 Extensions

This section comments briefly on some characteristics and extensions of the envelope extraction algorithm, in order to suggest possible applications beyond envelope identification. One of the less obvious is, perhaps, its use in the segmentation of a signal into pseudo cycles, a theme deeply explored in Section 3.4.

3.3.7.1 Superior and inferior envelopes

In practice, it is not uncommon for a discrete wave, especially in the case of sound, to present somewhat different superior and inferior contours. In those cases, the algorithm can be easily modified to generate two independent envelopes, that can be called the superior and inferior frontiers of a signal.

If P, defined in Section 3.3.2, is divided into P^+, the set of the local maxima of the original signal, and P^-, the set of its local minima, one can independently apply the remaining steps of the algorithm in those two sets, arriving at a superior and an inferior frontier, respectively.

Figure 15 illustrates these frontiers for six diverse discrete sound waves, as well as an in-detail view of the highlighted segment for each signal. All waves are records of physical sounds, chosen to represent the applicability of the algorithm in a real-world scenario. The implementation available at the PyPI repository provides an option to extract the frontiers.

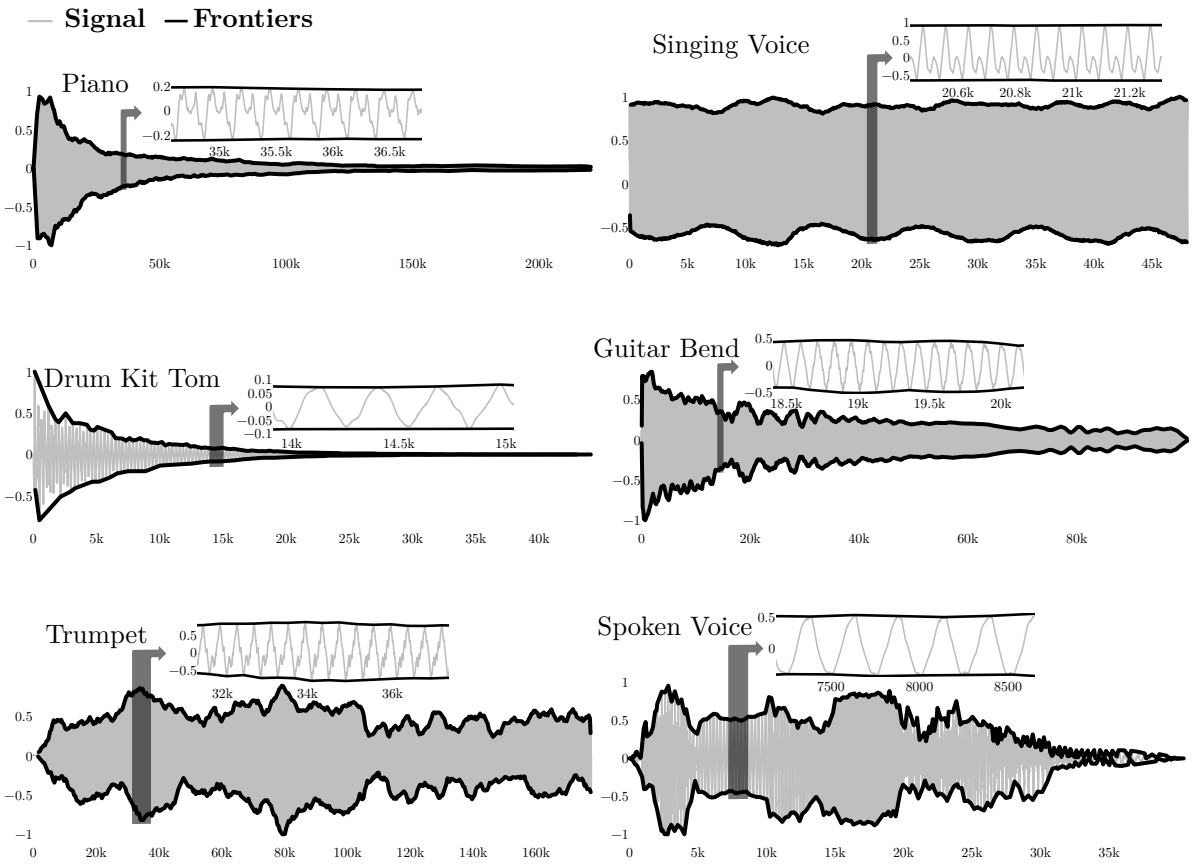

Figure 15: Superior and inferior frontiers of six discrete signals, extracted by the proposed algorithm. For each wave, the region highlighted in black is shown in detail. The horizontal axis of each subplot represents the sample index n, while the vertical axis represents the normalized amplitude.

The frontiers were satisfactorily detected in the case of harmonic and inharmonic sounds, and are robust in relation to the number of samples and the frequencies of the waves. It is worth noting that the positive and negative frontiers are generally smoother than the unified envelope presented in Figures 43, 44 and 45, and conforms better to the contours of the underlying discrete signal.

3.3.7.2 Simplified spectral representation

In general, the temporal envelope adds complexity to the spectral representation of a wave (Tarjano et al., 2019). As demonstrated in Section 5.1, the carrier wave obtained with the proposed envelope detection algorithm is guaranteed to have a narrower frequency range than the original signal.

Using the carrier wave in place of the original signal would simplify algorithms based on spectral analysis, without considerable drawbacks, since the envelope is fully known and can be applied back to the signal after eventual modifications.

Figure 16 shows the frequency-domain power spectrum for the wave and the carrier presented in Figure 10. For the carrier, the power spectrum presents two nonzero values, while the power spectrum for the wave, composed of the carrier modulated by the envelope, is more complex.

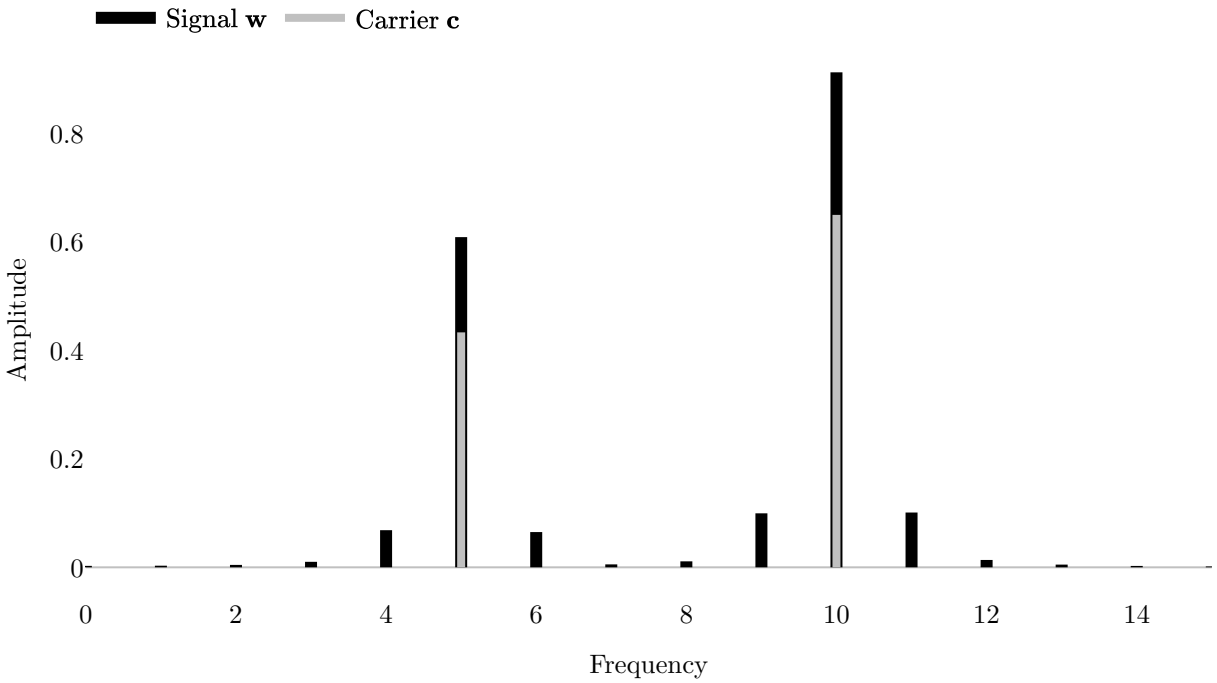

Figure 16: Fourier power spectrum for the wave and carrier shown in Figure 10.

Figure 17 shows part of the power spectra for both the original wave and the carrier obtained by the present algorithm, for the guitar bend sound in Figure 45, for frequencies around its fundamental frequency. Two prominent frequencies can be seen, corresponding to the initial and final notes of the bend. In the carrier, the other frequencies, introduced by the envelope, are less pronounced.

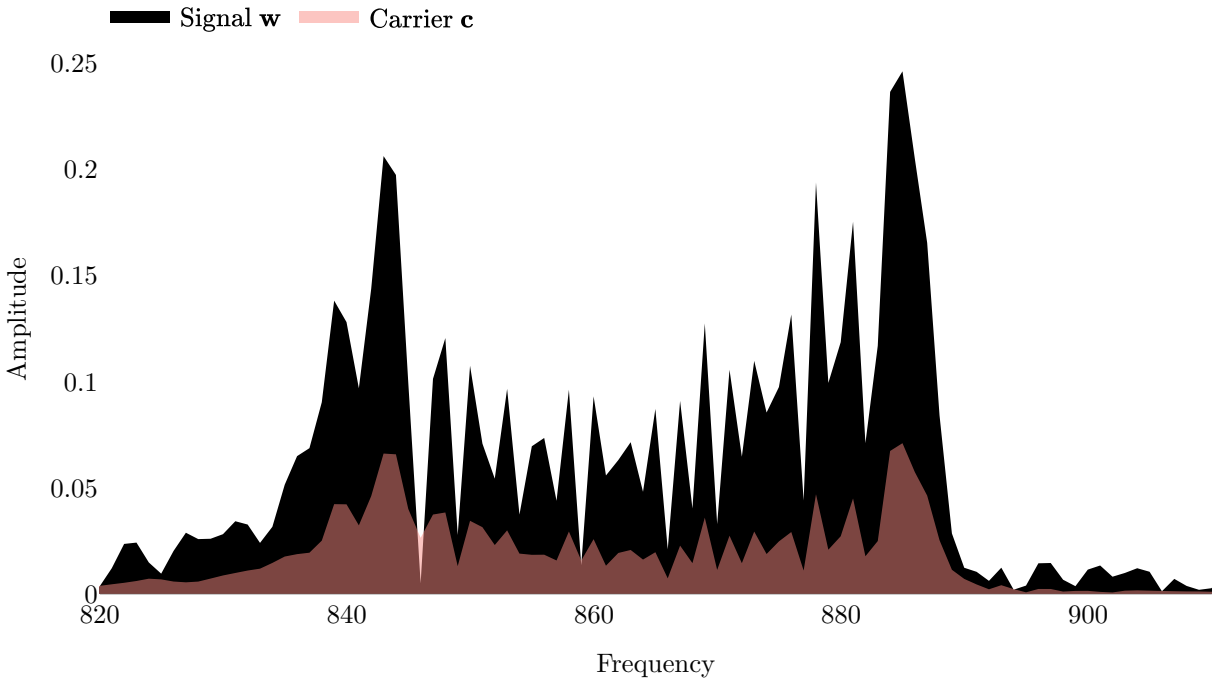

Figure 17: Fourier power spectrum for wave and carrier of the sound of a guitar bend shown in Figure 45.

3.3.7.3 Approximated location of pseudo cycles

The algorithm developed for envelope detection naturally divides a signal into its pseudo cycles, pinpointing them in the time domain. Figure 18 exemplifies this, for the sound of an alto singer singing a sustained note seen in Figure 44.

The vertical lines mark the indices of the samples that belong to the positive frontier of the original signal. The section between each vertical line can be seen as a pseudo cycle of the wave. Although two local maxima exist in each pseudo cycle, only the largest was marked by the algorithm.

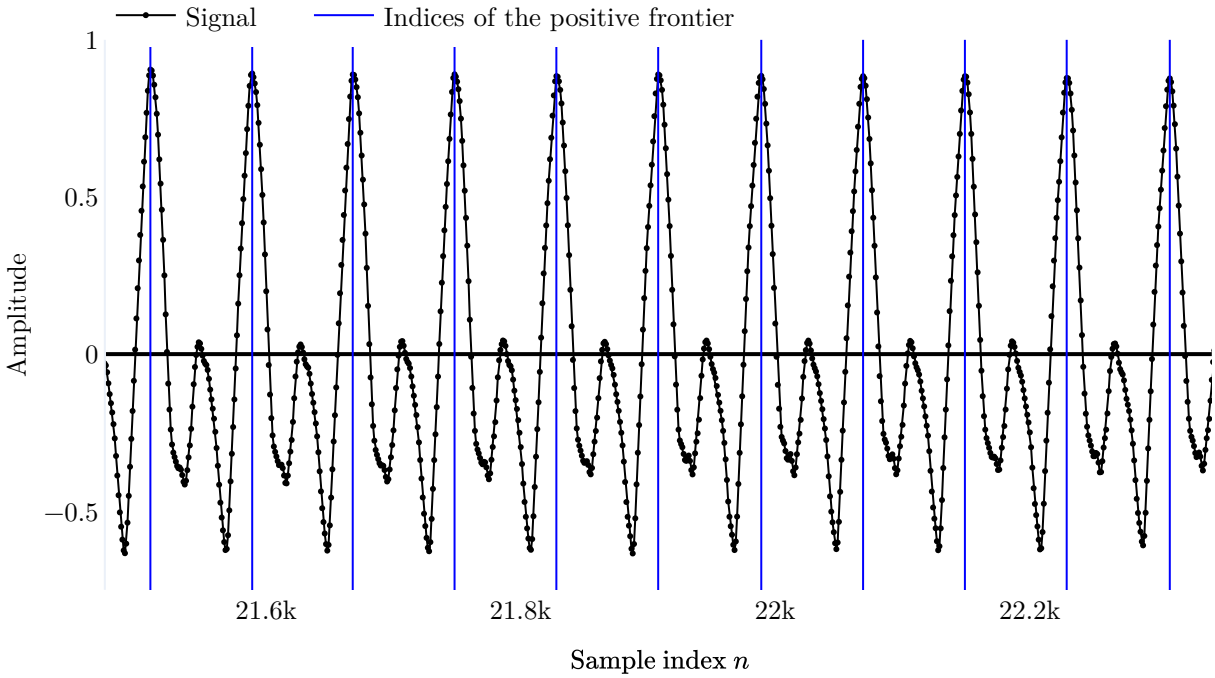

Figure 18: Part of the digital wave illustrated in Figure 44, with vertical lines at the indices of the samples that belong to the positive frontier.

Therefore, using those indices, one can split a discrete wave into its pseudo cycles. One application of that is illustrated in Figure 19: it shows the superposition of the extracted waveforms, after being normalized by length, and their average.

This same average is shown shifted by one average wavelength, to exemplify that approximate C^0 and C^1 continuity is achieved; in other words, in the point where they join, the curves share a common point (C^0 continuity) and are smooth around that point (C^1 continuity). This can be seen as the average waveform of the original discrete signal, and can be used, along with the description of the signal's envelope, to generate an alternative representation of the wave, useful, for example, for compression purposes; this idea will be further explored, albeit using an alternative approach, in Section 5.2.1, when a lossy compression codec is introduced.

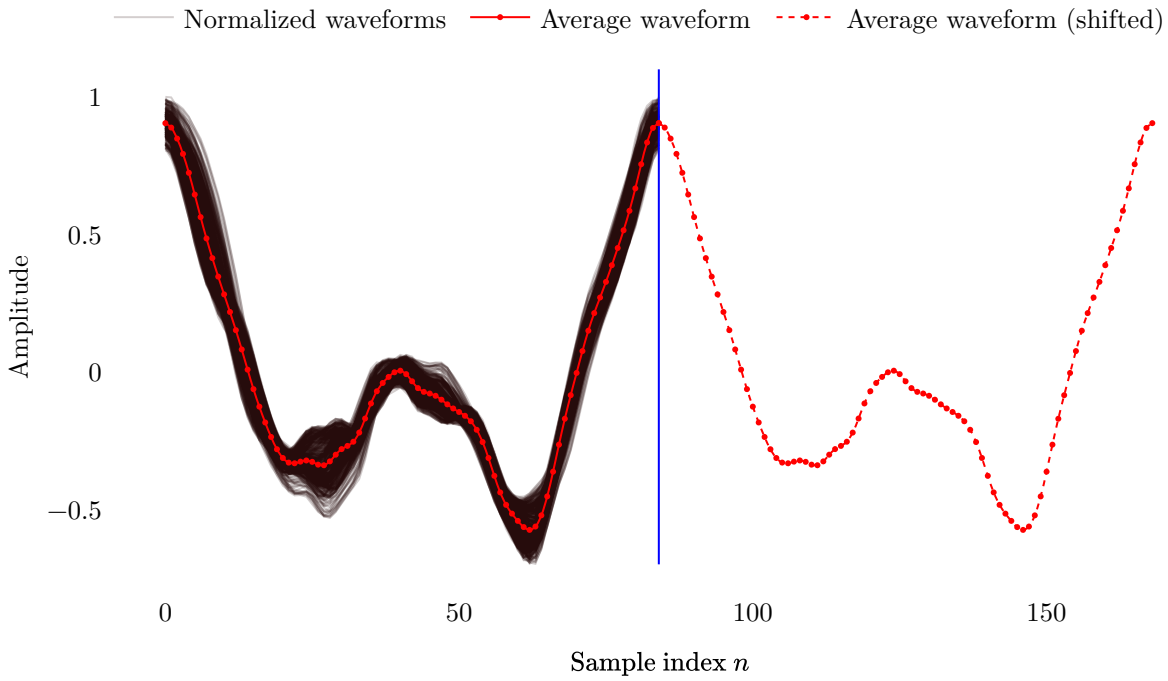

Figure 19: Waveform of each pseudo cycle of the wave illustrated in Figure 44, interpolated to the same length and superposed. The average in its original position, and shifted, is also shown.

3.4 SEGMENTING QUASI-PERIODIC SIGNALS INTO PSEUDO CYCLES

The contents of this section were published, with some modifications, in Tarjano et al. (2022a). Consider a sequence $\mathbf{s} = (s_1, s_2, \cdots, s_N)$ in $\mathbb{R}^N$, representing a finite discrete signal obtained by sampling part of a quasi-periodic continuous wave at equal time intervals.

From the predominant discrete frequency $k_{\text{pred}} \in \mathbb{N}$ of the discrete signal, obtained from the equation $k_{\text{pred}} = \text{argmax}(\text{DFT}(\mathbf{s}))$, where argmax is a function that returns the index of the maximum value of a sequence, we can obtain an initial approximation $T_0 \in \mathbb{R}$ of the period of the signal from the equation $T_0 = N/k_{\text{pred}}$.

This approximation, however, potentially underestimates the length of pseudo cycles with complex waveforms involving various zero crossings, introducing in the algorithm a tendency to divide the original signal into non-homogeneous segments.

An updated approximate period T can be obtained by $T = \text{argmax}(\text{r}_{\mathbf{ss}}(\lceil T_0/2 \rceil), \text{r}_{\mathbf{ss}}(\lceil T_0/2 \rceil + 1), \cdots, \text{r}_{\mathbf{ss}}(l_{\max} T_0))$, where $\text{r}_{\mathbf{ss}}(l)$ is the autocorrelation of $\mathbf{s}$ with itself at lag $l \in \mathbb{N}$. $l_{\max} \in \mathbb{N}, l_{\max} \geq 1$ fixes the maximum lag, in terms of T_0. In this work $l_{\max}$ is set empirically to 6, reflecting the maximum complexity expected for the pseudo cycles' waveforms, in terms of zero crossings.

Starting at the autocorrelation lag $l = \lceil T_0/2 \rceil$ addresses the problem that the autocorrelation tends to be high, regardless of the signal, for lags close to zero (Sikora et al., 2005).

By sliding a window of size T over $\mathbf{s}$, sample by sample, and performing a DFT at each step, we can localize the behavior of the basis waveform in time, observing the behavior of the predominant phase of each DFT.

This procedure is analogous to the sliding discrete Fourier transform (Rabiner, 1975). It is also very close to performing a circular time shift in the underlying basis waveform, as long as this signal is reasonably harmonic; one can thus expect an approximately linear behavior for the values obtained using this procedure (R. Lyons, 2003). Since those values are bound between $-\pi$ and π, this linearity generates a succession of approximately parallel diagonal lines, as seen in Figure 20.

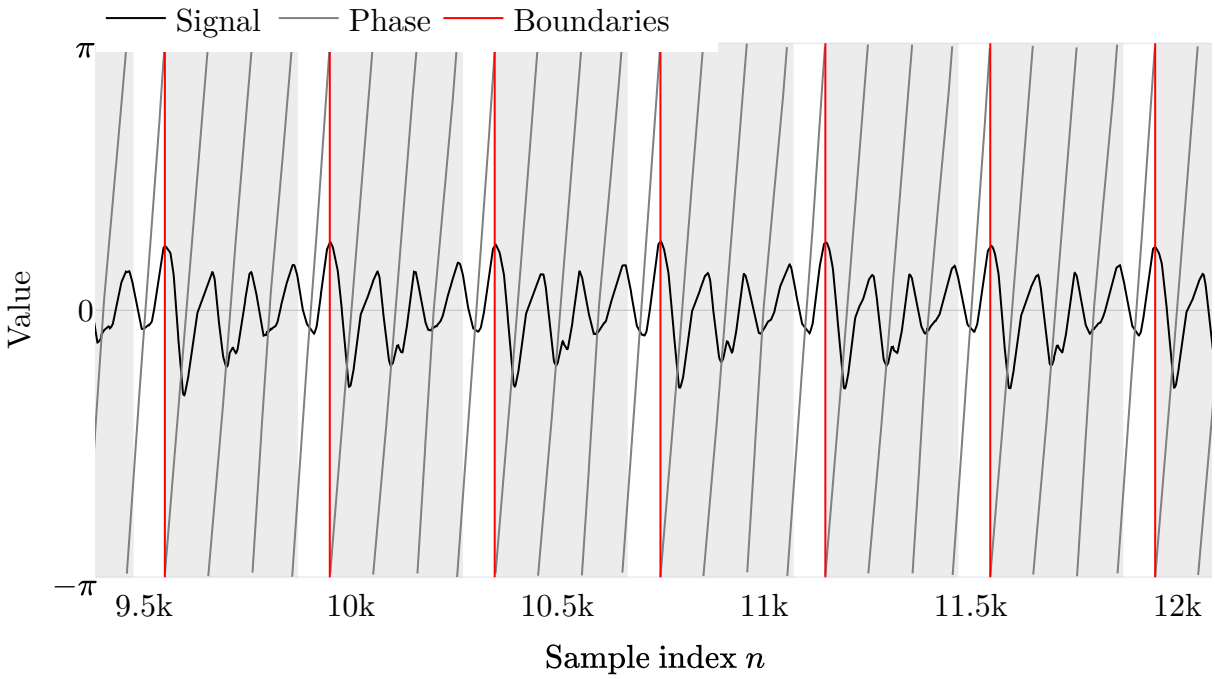

Figure 20: Signal and predominant phases for part of the soprano A signal shown in Table 6. Where the phase reaches π outside the grayed-out area representing the jumps, a boundary is defined.

Any horizontal line, intercepting each of those parallel lines at the same value, would define a set of potential boundaries of the underlying signal. In the actual implementation, presented in Algorithm 2, this line is set at π for convenience.

Since the approximate period T of the signal is known, each time a boundary is found, the algorithm can jump forward a fraction d of T, in the interest of efficiency.

Those jumps are represented in Figure 20 as the grayed-out areas in the plot.

The location of the boundaries, in the form of indices of the original signal, is stored in the vector of positive integers $\mathbf{n}$ and is the output of Algorithm 2. A representation of Algorithm 2 in the form of a flow diagram can be seen at the root of the repository prepared for the algorithm (Tarjano, 2021), under the name `SegmentationAlgorithm.pdf`. A Python implementation can be found at `github.com/tesserato/compression/blob/main/000PythonImplementation.py`.

If we let $P \in \mathbb{N},\ P > 0$ be the number of pseudo cycles in which the original signal $\mathbf{s}$ is divided, $\mathbf{n}$ can be properly defined as $\mathbf{n} = (n_1, n_2, \cdots, n_{P+1}),\ \mathbf{n} \in \mathbb{N}^{P+1}$. This is so because we do not consider the region between the beginning of the signal and the first boundary n_1, or between the last boundary n_{P+1} and the end of the signal as pseudo cycles since they are, in general, incomplete pseudo cycles.

3.4.0.1 Deriving an envelope from the results of the segmentation algorithm

As noted in Section 2.2, envelopes are important components in the characterization of signals (Tarjano et al., 2022b), being largely responsible for the intelligibility (Qi et al., 2017) and emotion (Zhu et al., 2018) of the spoken voice, for example. Nevertheless, the identification and even the definition of a temporal envelope, in the general case of broadband signals, is still an open question (Jia et al., 2019).

While an envelope detection algorithm was already introduced in Section 3.3.5, the segmentation procedure illustrated in Algorithm 2 can serve as a starting point for a more accurate definition of temporal envelopes, and for a novel algorithm to extract them, more fitting in the case of harmonic signals. Even for signals with very little harmonic content, as a crash cymbal, the waveform of the reconstructed signal, obtained as presented in Section 3.4.1, is very close to that of the original signal, as Figure 21 illustrates.

Algorithm 2 Segment Signal

Input:
$\mathbf{s} \in \mathbb{R}^N, \mathbf{s} = (s_1, s_2, \cdots, s_N)$ ▷ original discrete signal representation
$l_{\max} \in \mathbb{R}, l_{\max} \geq 1$ ▷ max. expected complexity of basis waveform
$d \in \mathbb{R}, 0 < d < 1$ ▷ fraction of T to be skipped once a frontier is found
$t \in \mathbb{R}, -\pi \leq t \leq \pi$ ▷ threshold above which the
$\mathrm{r}_{\mathbf{ss}}(l), l \in \mathbb{N}$ ▷ given a delay (or lag) l, returns the autocorrelation of the signal $\mathbf{s}$
procedure SEGMENT($\mathbf{s}, l_{\max}, t, d$)
    $k_{\text{pred}} \leftarrow \text{argmax}(\text{DFT}(\mathbf{s}))$
    $T_0 \leftarrow N/k_{\text{pred}}$
    $T \leftarrow \lceil T_0/2 \rceil + \text{argmax}(\mathrm{r}_{\mathbf{ss}}(\lceil T_0/2 \rceil), \mathrm{r}_{\mathbf{ss}}(\lceil T_0/2 \rceil + 1), \cdots, \mathrm{r}_{\mathbf{ss}}(l_{\max} T_0))$
    $\mathbf{n} \leftarrow \{\}$
    $n \leftarrow 0$
    $\phi' \leftarrow 0$
    $n' \leftarrow 0$
    insideThreshold $\leftarrow$ **False**
    while $n + T < N$ **do**
        $\phi \leftarrow \arg(\max(\text{DFT}_T(s_n, s_{n+1}, \cdots, s_{n+T})))$
        if $\phi > t$ **then**
            insideThreshold $\leftarrow$ **True**
            if $\phi > \phi'$ **then**
                $\phi' \leftarrow \phi$
                $n' \leftarrow n$
            end if
        else
            if insideThreshold **then**
                $\phi' \leftarrow 0$
                $n \leftarrow n' + \lceil dT \rceil$
                $\mathbf{n} \leftarrow \mathbf{n} \cup \{n'\}$
            end if
            insideThreshold $\leftarrow$ **False**
        end if
        $n \leftarrow n + 1$
    end while
    return n ▷ indices of $\mathbf{s}$ that represent the boundaries between pseudo cycles
end procedure

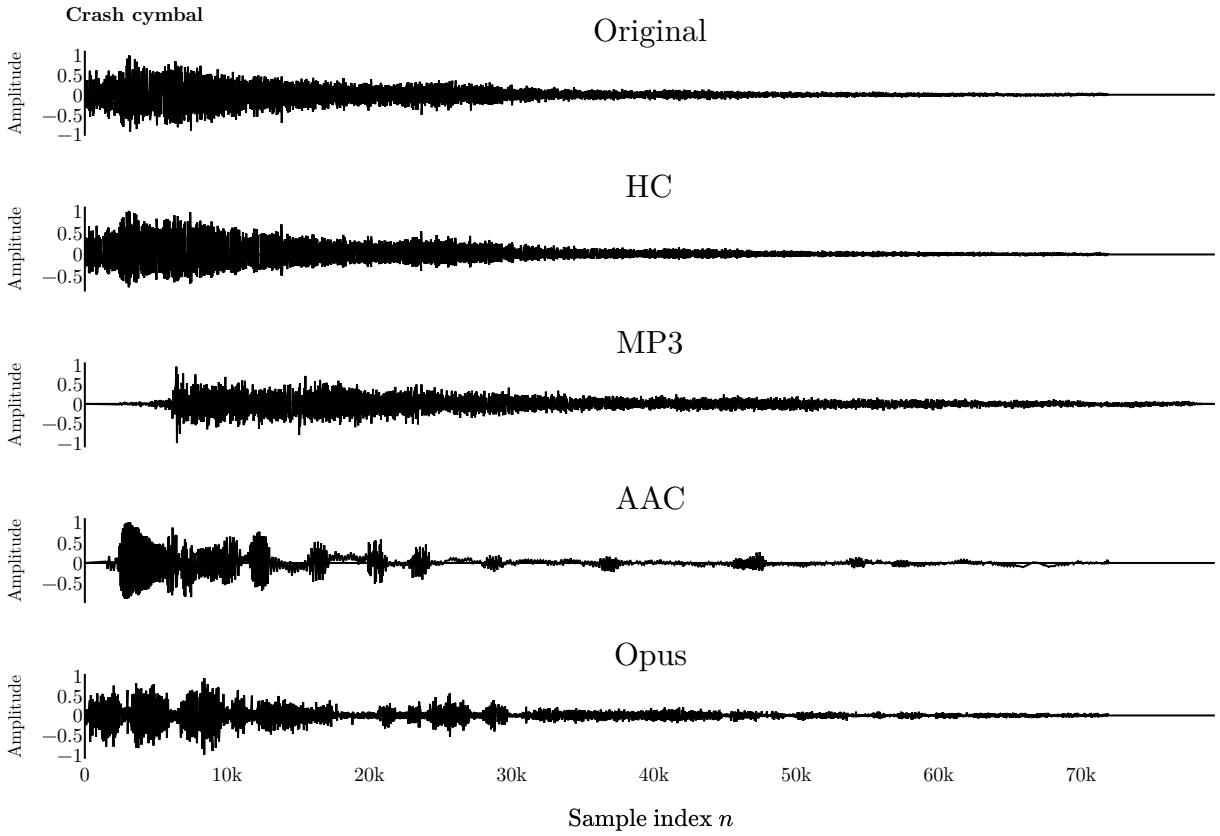

Figure 21: Waveform of the original signal and the 4 compressed signals. HC is the compression schema presented in Section 3.3.2. The traditional codecs are operating at extreme compression settings fully described in Section 5.2.2.1. The cymbal, being predominantly inharmonic, is used to illustrate that the algorithm exhibits reasonable performance even in worst-case scenario situations.

From the output of the segmentation Algorithm 2 we can straightforwardly obtain two sets of points: the set $F_{>0}$ of the points with the maximum amplitude of each pseudo cycle as their ordinates and the index of the sample in which it occurs as their abscissa and, analogously, the set $F_{<0}$ of the points with the index and the value of the samples with minimum amplitude in each pseudo cycle.

Those sets can be seen in Figure 22, for the soprano A signal, compared with the envelope obtained by the low pass filtered result of the Hilbert transform, one of the most common algorithms for envelope extraction.

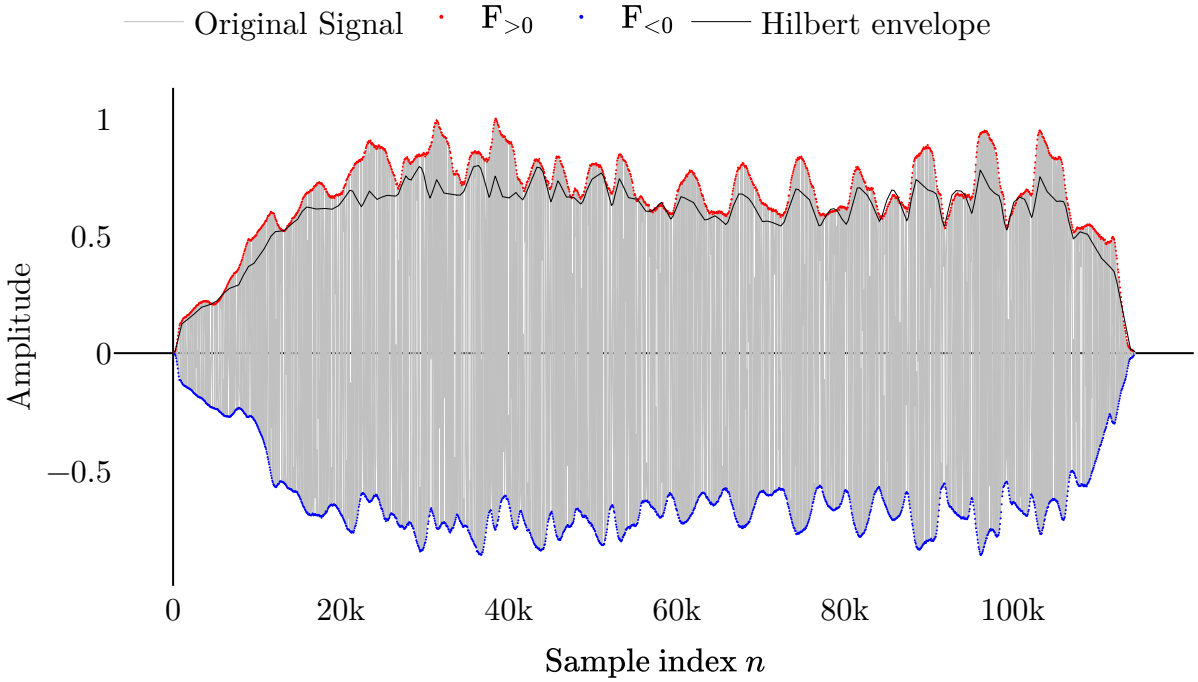

Figure 22: Positive and negative frontiers of the original soprano A signal, obtained using the segmentation presented in Algorithm 2, compared to the envelope obtained using the Hilbert transform.

Those sets can be transformed into proper envelopes by interpolation. Examples for all signals, using linear interpolation, are available at the repository prepared for the segmentation algorithm (Tarjano, 2021).

3.4.1 Representing a signal as an evolving waveform

Segmented into P pseudo cycles, the sequence $\mathbf{s} = (s_1, s_2, \cdots, s_N)$, introduced in Section 3.4 to represent a finite discrete signal, can be interpreted as a succession of shorter signals, or waveforms, each one representing a (pseudo) cycle of the original wave, not unlike how a video emerges from a sequence of images.

Mathematically, the instantaneous amplitude s_n of the original signal, which was represented as a function of the sample index n of the discrete representation of the signal, would now be represented as a function of two variables: we can define the new representation as $z(i, j)$, where $i \in \{1, 2, \cdots, P\}$ indicates the index of each pseudo cycle among the P pseudo cycles in which the wave was divided, and $j \in \mathbb{N}$ is each pseudo cycle's sample index. This is exemplified in Figure 23.

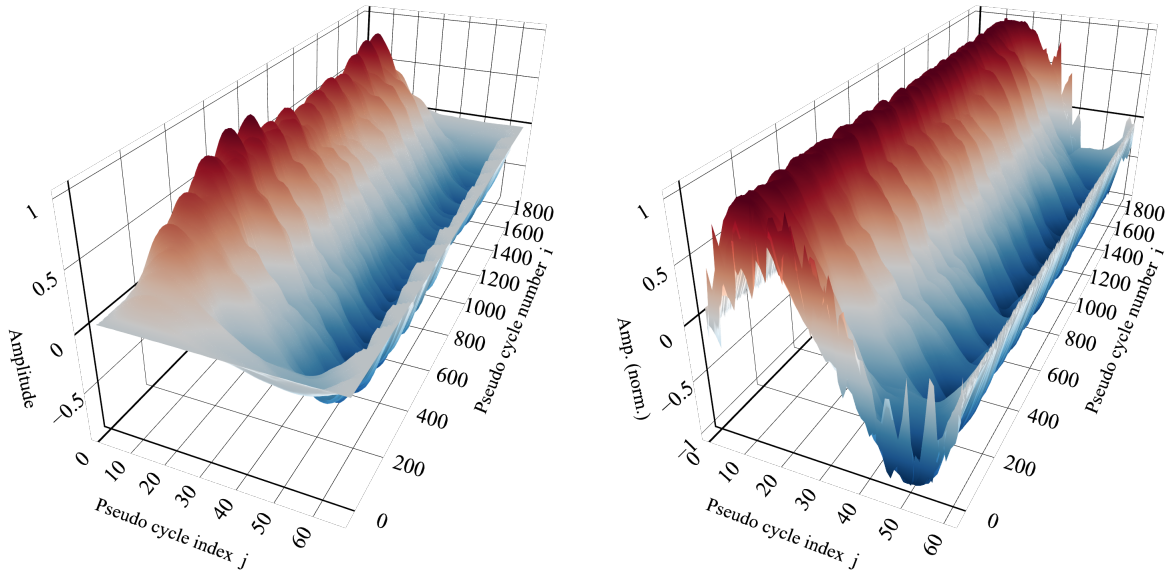

Figure 23: The original Soprano A signal is shown, segmented into its pseudo cycles. In the left subplot, the effect of the temporal envelope and the variation in pseudo cycle lengths can be seen, in the form of a white, approximately plane area of the surface on the left-hand side of the graph. This is so because the waveforms of the pseudo cycles were zero-padded. Those effects were normalized in the plot on the right, to highlight the similarity between adjacent waveforms.

In the case of perfectly periodic signals, the lengths and the waveforms of those pseudo cycles would be identical, and the identification of a single waveform would characterize the whole signal. In most real-world signals, on the other hand, the waveform changes, often subtly, between pseudo cycles, as changes the length of each pseudo cycle. To accommodate that, L itself must be a function of i, such that $j \in \{1, 2, \cdots, L_i\}$. In other words, the range of the index j, that represents the indices of the instantaneous amplitude of each pseudo cycle, is generally different for each pseudo cycle, reflecting their different lengths.

Figure 23 illustrates, for quasi-periodic signals, how those changes can be subtle. Both subplots show the index i of each pseudo cycle in the longest horizontal axis, representing the waveform's evolution. In the shortest horizontal axis, from left to right, the index j, that reflects the position inside each pseudo cycle, is shown.

On the left subplot in Figure 23, the effect of the envelope is present, as can be seen by the differences in maximum amplitude between the waveforms of the different pseudo cycles. The length L_i of each pseudo cycle is also slightly different: this can be seen as the white space on the right side of the graph.

On the right subplot, both effects were normalized. The waveforms of every pseudo cycle were interpolated to have a length equal to the maximum length. Likewise, their amplitude was normalized, to remove the effect of the temporal envelope. It can be seen that, after those steps, the variation between normalized waveforms is very small.

If we assume that the basis waveform is approximately constant, as is the case for tuned instruments and singing voice, for example, we can introduce a simplified representation for digital waves. Let this basis waveform be defined as $\mathbf{w} = (w_1, w_2, \cdots, w_{\overline{L}})$, where $\overline{L}$ is the average of the pseudo cycle's lengths.

We consider, as a simplification, that this basis waveform will be modified only by the effect of the temporal envelope modulating its amplitude and small deviations from the predominant period modulating its length. Besides the basis waveform, then, those quantities must be stored.

Using $\mathbf{n}$, the vector of the indices of the boundaries between pseudo cycles that is the output of Algorithm 2, one can directly obtain the length, or period, of each pseudo cycle, by observing the distance between two successive boundaries, as formalized by Equation 22.

$$\mathbf{t} = (n_2 - n_1, n_3 - n_2, \cdots, n_{L+1} - n_L) \tag{22}$$

Since $0 \leq n_i \leq N$, where N is the size of the original signal, that can be arbitrarily large, storing the information about the length of each pseudo cycle in the form of the values of $\mathbf{t}$ has a practical advantage: those values, generally much smaller than N, can be stored in fewer bytes.

Likewise, the vector of pseudo cycles' amplitudes can be defined as the maximum absolute amplitude of each pseudo cycle, as in Equation 23. This definition is similar to the definition of the frontiers in Section 3.4.1.

$$\begin{aligned} &\mathbf{a} = (a_1, a_2, \cdots, a_L); \\ &a_i = \max(|s_{n_i}|, |s_{n_i+1}|, \cdots, |s_{n_{i+1}-1}|) \ \forall \ i; \ 1 \leq i \leq P \end{aligned} \tag{23}$$

Lastly, the basis waveform can be obtained as the average of the normalized waveforms of each pseudo cycle, as defined in Equation 24. An example is shown in Figure 24.

$$\mathbf{w} = \text{IDFT}\left(\frac{\sum_{i=1}^{L} \text{DFT}_{\overline{L}}\left(\frac{s_{n_i}}{a_i}, \frac{s_{n_i+1}}{a_i}, \cdots, \frac{s_{n_{i+1}-1}}{a_i}\right)}{L}\right) \tag{24}$$

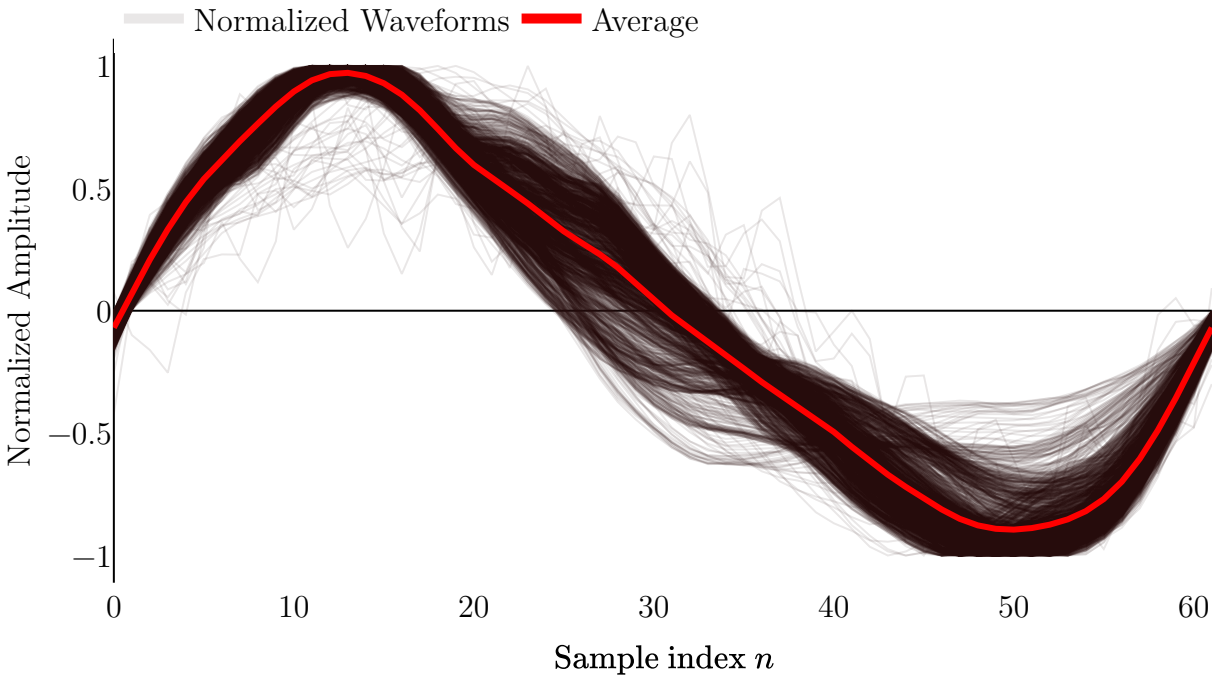

Figure 24: The various waveforms that compose the original soprano A signal are shown superimposed, after being scaled to the same length, and having their amplitudes normalized. The average of those individual waveforms is shown in red.

From the periods $\mathbf{t} \in \mathbb{N}^L$ (see Equation 22), the amplitudes $\mathbf{a} \in \mathbb{R}^L_{\geq 0}$ (see Equation 23) and the basis waveform $\mathbf{w} \in \mathbb{R}^{\overline{L}}$ (see Equation 24) the original signal can be approximately reconstructed. We can formalize the recreated signal as a concatenation of its pseudo cycles, such as defined in Equation 25.

$$\begin{aligned} &\mathbf{s}' = (\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_L); \\ &\mathbf{p}_i = a_i \ \text{IDFT}_{t_i}\Big(\text{DFT}(\mathbf{w})\Big) \ \forall \ i; \ 1 \leq i \leq L \end{aligned} \tag{25}$$

This representation guards some similarities with the technique of wavetable synthesis, where a cycle of a longer signal is stored, to be read later at different speeds (Franck et al., 2013), allowing different pitches to be derived from a single wavetable. Although the focus of wavetable synthesis has recently shifted to sound design, the use of wavetable synthesis for lossy compression is investigated, for example, in Maher (2005).

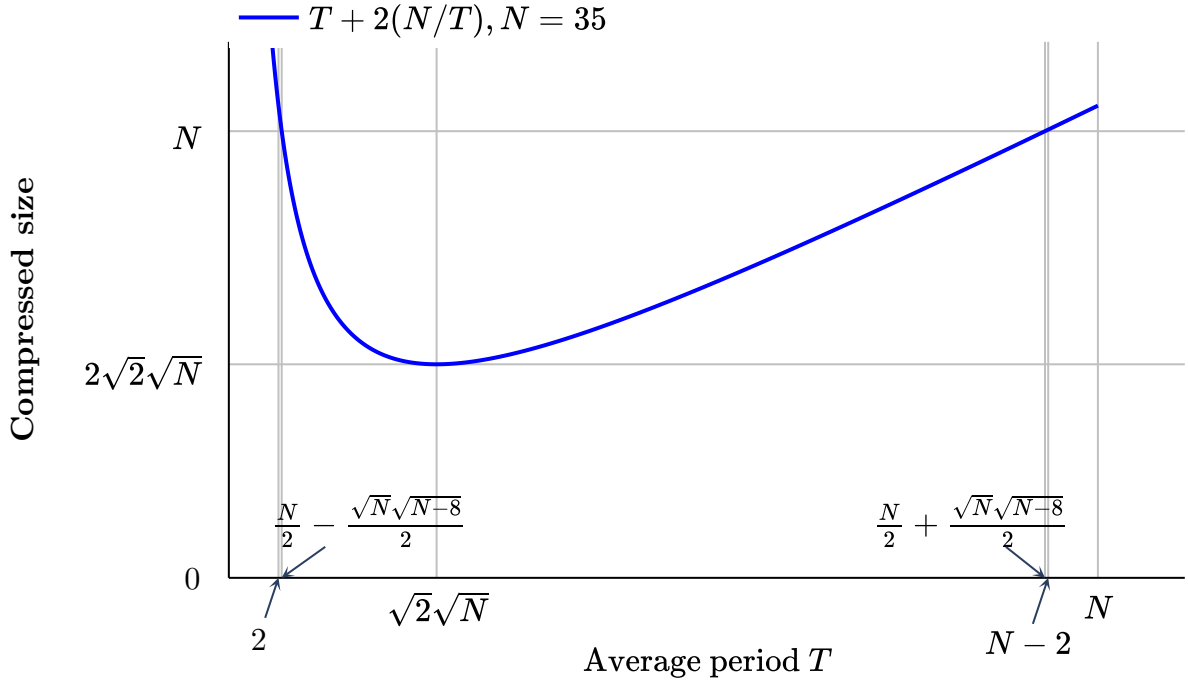

Figure 25: The theoretical compressed size as a function of T, the average period of the original signal.

3.4.2 Theoretical analysis of the segmentation algorithm

Insight into the compression potential of the representation introduced in Section 3.4 can be obtained from the following approximation: the original signal has approximately N/T pseudo cycles, and the alternative representation can be seen as a vector of approximately $T+2(N/T)$ real numbers. Transforming the N real numbers that describe the time domain representation of the original signal into the $T + 2(N/T)$ numbers necessary for the alternative representation results in compression if $\frac{N}{2} - \frac{\sqrt{N}\sqrt{N-8}}{2} \leq T \leq \frac{N}{2} + \frac{\sqrt{N}\sqrt{N-8}}{2}$.

As N grows, the leftmost side of this inequality tends to 2, while the rightmost side tends to $N - 2$, making the limits inside which this representation is advantageous comfortable. For sufficiently long signals, virtually every predominant period T results in compression.

Maximum compression is achieved when $T = \sqrt{2}\sqrt{N}$ and the original N real numbers that represent the original signal are encoded as $2\sqrt{2}\sqrt{N}$ real numbers under the alternative representation. This puts a theoretical limit of $\frac{\sqrt{2}\sqrt{N}}{4}$ on the compression rate attainable with the algorithm. As will be seen in Section 3.3.2, optimizations in the way the alternative representation is stored will lead to higher compression rates in practice for all tested signals.

Algorithm 2 forms the core of the compression algorithm, where the most important and resource-intensive processes take place and, as will be seen shortly, dominates the complexity of the compression algorithm.

The DFT of the whole signal, as seen in line 8 of the algorithm, has $O(N\log(N))$ complexity. In line 10, $l_{\max}T_0 - \lceil T_0/2 \rceil$ autocorrelation steps are performed, at the (maximum) complexity of $O(N)$ for each lag, plus a maximum $O(N)$ complexity in determining the lag with the highest correlation value among the evaluated lags. Considering $k = l_{\max}T_0 - \lceil T_0/2 \rceil$, one can write the complexity of the fixed part of the algorithm as $O(N\log(N)) + O(kN) + O(N)$, which can be simplified to $O(N\log(N))$.

The iterative part of Algorithm 2, under the `while` loop, is repeated $(N - T)$ times. There are two options to implement the DFT indicated in line 17: we can perform it directly, at the cost of $T\log(T)$ operations per iteration, or we can use a sliding discrete Fourier transform (SDFT), that demands $T\log(T)$ operations for the first iteration (R. Lyons, 2003) and $3T$ operations, a complex multiplication and two additions per time instant (Orallo et al., 2017), for each subsequent iteration. Regardless of the approach, additional T operations are necessary, at each iteration, to find the maximum value of the result. All remaining operations in each cycle are not a function of T.

Using the direct DFT approach results in a total of $(N-T)(T\log(T)+T)$ operations in the iterative part of the algorithm. After expansion, we have $TN\log(T) + TN - T^2\log(T) - T^2$; the complexity is dominated by the $TN\log(T)$. For $N, T \geq 1$, the function is monotonically increasing in relation to both N and T and, since $T \leq N$, it reaches its maximum when $N = T$, conferring $O(N^2\log(N))$ complexity for the direct DFT approach.

The SDFT method demands a total of $T\log(T) + (N - T - 1)3T + (N - T)T$, or $4TN + T\log(T) - 3T - 4T^2$ operations. When $N = T$, the number of operations necessary is dominated by $4N^2$, giving a complexity of $O(N^2)$.

For large values of N, we have that $O(N^2) < O(N^2\log(N))$, and can say, provided we choose the SDFT implementation, that the iterative part of Algorithm 2 has $O(N^2)$ complexity. Since this dominates the complexity of $O(N\log(N))$ found for the initial part of the algorithm, we can say that the segmentation algorithm as a whole has complexity $O(N^2)$.

To determine the complexity of the complete compression algorithm, however, we need to analyze the additional steps necessary for generating and storing the new representation of the original signal, not described in Algorithm 2. From the vector $\mathbf{n}$,

that contains the indices of the samples that represent frontiers between adjacent pseudo cycles, the output of the segmentation algorithm, we first divide the original signal into segments.

The maximum absolute value of each segment has to be found, as shown in Equation 23, and each segment normalized, as per Equation 24, for a total of $3N$ operations.

Converting the P normalized segments to the frequency domain via DFT, as in Equation 24, demands $(n_{i+1}-n_i)\log(n_{i+1}-n_i)$ operations for each segment, where $n_{i+1}-n_i$ is the length of each segment and $(n_2-n_1)+(n_3-n_2)+\cdots+(n_{P+1}-n_P)=N$; assuming, as a simplification, pseudo cycles of equal length, such that $n_2-n_1=n_3-n_2=\cdots=n_{P+1}-n_P=T$, we have the case where the original signal is divided into $P=N/T$ pseudo cycles of length T each. Hence, we have a total of $PT\log(T)=N\log(T)$ for the steps described.

We need then to average P segments of length T along their indices, for a total of N computations, plus $T\log(T)$ computations, regarding the IDFT, to bring back the average to the frequency domain. This totalizes $N+T\log(T)$ operations.

Finally, we generate vector $\mathbf{t}$ from vector $\mathbf{n}$, as seen in Equation 22, at the cost of P computations, and store vectors $\mathbf{t}$, $\mathbf{a}$, and $\mathbf{w}$ with $P+P+T$ operations.

We have then, for the additional steps of the compression, a total of $4N+N\log(T)+T\log(T)+3N/T+T$ computations. For $N,T\geq 1$ the terms with $\log(T)$ are monotonically increasing and dominate the complexity in the worst-case scenario when $T=N$ and $N\to\infty$; the additional compression steps, therefore, have complexity $O(N\log(N))$. The complexity of the whole compression algorithm is thus dominated by the $O(N^2)$ complexity of the segmentation part of the algorithm, and can be said to be $O(N^2)$.

The decompression step was designed to be simpler, to ensure performance compatible with real-time usage. In practice, more efficient approaches than the DFT-IDFT pair in Equation 25 can be drawn from the vast wavetable literature to resample the base waveform. Although formally described in Equation 25 as a DFT followed by an IDFT, domain conversions to scale the basis waveform were substituted by cubic Hermite interpolation in the implementation.

After reading the vectors $\mathbf{t}$, $\mathbf{a}$, and $\mathbf{w}$ at the cost of $P+P+T$ operations, an interpolation model is constructed in approximately T operations. Each evaluation of a new sample demands additional $\log(T)$ plus 1 operation to multiply it for the appropriate amplitude. We have, then, a total of $P+P+T+T+N(\log(T)+1)$ for the decompression phase, which gives a maximum complexity of $O(N\log(N))$ for the decompression

algorithm.

This difference in complexity is reflected in the times presented in Figure 52, where the compression and decompression performance of the implementation is tested in a practical scenario.

4 THE GENERAL ALGORITHM

After laying the theoretical groundwork for a better understanding of discrete quasi-periodic signals — and, consequently, the sound of pitched musical instruments — in Sections 3.3 and 3.4, and introducing an improved representation for such sounds in Section 3.4.1, it is now possible to design and implement the digital musical instrument that is the main objective of the present work.

While the algorithms presented in Sections 3.3 and 3.4 were developed with the ultimate goal of enabling the representation introduced in Section 3.4.1, that serves as the basis for the plugin, they were published as independent papers and ultimately placed in the Proposed Techniques chapter, of a more general scope, on the grounds that their usefulness is not restricted to the present context.

This chapter concerns itself, thus, mainly with the process of actually describing the general framework that enables the real-time emulation of pitched musical instruments and the singing voice.

For the accurate description of such a framework, however, two key components are yet to be introduced: the neural network responsible for learning the mappings from the high-level descriptions of the desired sounds to the neural-networks-friendly representation, and the plugin that transforms this representation into the actual sound signals. Those entities are introduced, respectively, in Sections 4.3 and 4.2 of this chapter, after a brief commentary, in Section 4.1 on the ecosystem in which they reside.

A diagram representing an overview of the general workflow for the emulation of a particular instrument is shown in Figure 26.

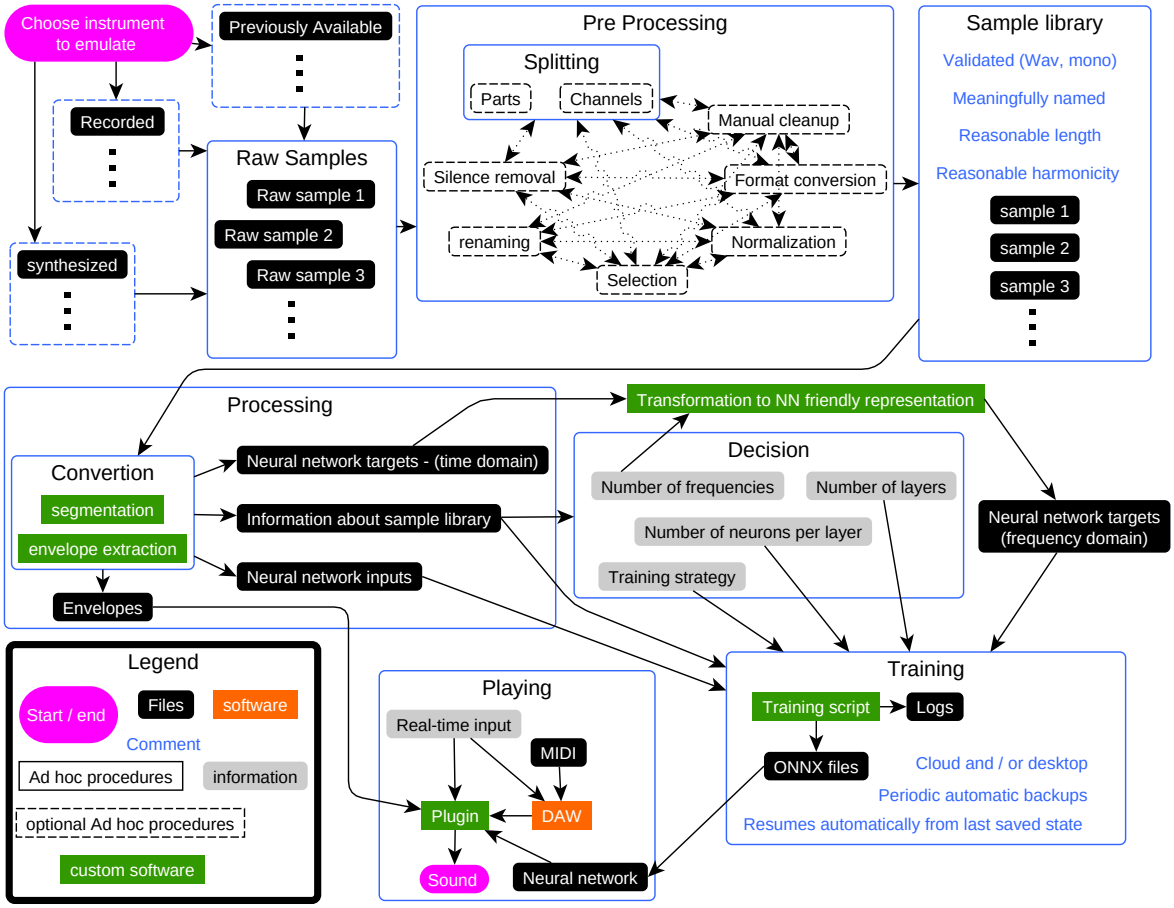

Figure 26: Workflow for the emulation of an instrument, starting with the process of acquiring the samples and culminating in the generation of a trained neural network to serve as the plugin's engine.

As Figure 26 illustrates, the process starts with the acquisition of samples that must represent, as best as possible, all the dynamics one would like to reproduce in a particular instrument. In the case of a piano, for example, one would use a set of samples comprised of recordings of each of its keys played at different velocity levels. An analogous set, with the piano in its dumped state, as well as sets representing multiple microphone positions, can also be used. Ideally, multiple redundant samples of each of those sounds, known as round-robins, would be present.

As for the origin of those samples, they could be recorded from a real instrument or generated by (physical) simulation, most probably using a resource-intensive simulation algorithm that doesn't lend itself well for real-time synthesis. In the case of recorded samples, they could be pre-recorded, and made available, free or commercially, by third parties, or specifically recorded for the purpose of the algorithm's implementation. The last case, while preferable, is sometimes cost-prohibitive.

The next step of the process consists of the pre-treatment of the samples. It can involve the normalization of the collected samples, to ensure that samples representing a particular sound intensity have, in fact, consistent volume. Silence removal is also a common task in this step, as is the conversion of the files from the original format in which they were recorded to the mono Waveform Audio File (WAV) format.

It is possible also, in this step, to split the channels of stereo files into two individual mono samples, artificially generating round-robins. Sometimes, available libraries encode many individual samples sequentially in the same file, in which case they must also be split.

It is also common in this step, especially in the case of using previously available samples, to discard samples that don't conform to the specifications, for various reasons: they can be corrupted by external noise, be too long, or don't represent their nominal pitch, for example. Although the conversion algorithm infers the necessary characteristics of each sample, such as pitch and intensity, in order to generate the appropriate neural network input, it is common to rename files in this step, to a more convenient naming convention.

The output of the previous steps is a consistent and validated set of samples exhibiting reasonable length and harmonicity, as shown in "Sample library", in Figure 26. The prerequisite of reasonable length is imposed by the necessity of performing the FFT in each sample, to assess its predominant frequency, and compare it to the predominant frequency inferred by the "Conversion algorithm", the algorithm that also performs the envelope extraction and segmentation of the samples.

This requisite is intentionally left vague since it must be fixed by the user, according to the hardware and time available for the task. Based on this comparison, samples with discrepancies in their pitch, as inferred by the algorithm and the FFT, corresponding to distances greater than two semitones, are automatically excluded by the algorithm and marked as discarded in its log. Hence, the "Reasonable harmonicity" requirement. This cautionary check is done to prevent subtle interferences from making their way into the training of the network, generally considerably more resource and time-intensive, where they would degrade the results of the emulation, in an opaque way.

The "Processing" step in Figure 26 transforms this set of samples into two sequences of vectors, encoded as CSV files, and saves them to the disk. Each sample is segmented into its pseudo cycles. For each pseudo cycle, input information — such as the predominant pitch and volume of the original sample it belonged to, as well as the

pseudo cycle index — are stored in the input sequence. The corresponding pseudo cycle, still in its time domain representation, is stored in the corresponding entry in the output sequence, as described in more detail in Section 4.3.

Information about the envelope of each sample is also extracted and saved, to be used later by the plugin. This step additionally outputs general information about the samples used, in the form of a CSV file, such as the maximum amplitude of each original sample, their inferred pitch, minimum, maximum, and average periods, length and size in disk, to name a few, that can be used to inform the training step.

This information, besides being used to inform the training step, must be used to establish the number of frequencies to be used in order to emulate the original instrument, as well as the number of parameters of the network. Both topics are discussed more profoundly in Section 4.2 and Section 4.3, where baseline values are derived and can be used as-is in the vast majority of cases.

To account for potential failures, a log is saved periodically during the training process. Upon resuming training, the script checks for preexistent logs, and automatically resumes from the last saved position, if such information is available.

Training can be performed both in cloud environments, or on a local desktop, and can sequentially (but not concomitantly) be performed in both, given the convenience: training can proceed for a period of time on a local desktop, be resumed on a cloud environment, and resume locally, for example.

Once properly trained, the networks function as the plugin's engine and can be used to generate sounds. This is done either directly, by using a real-time controller such as a musical keyboard with MIDI-controller capabilities, or hosting the plugin in a DAW, where it can also be controlled in real-time or can be used to interpret preexistent MIDI files.

4.1 THE ECOSYSTEM

While audio applications are, at their core, no different from any other pieces of software, an understanding of the current ecosystem regarding audio software, especially in relation to tacit industry standards and consolidated best practices can be beneficial for the efficiency and compatibility of the final product. Besides rendering the development process more streamlined, this knowledge guarantees that consistent design decisions can be made at the early steps of development, minimizing the necessity of refactoring and reimplementation.

The first important practical peculiarity to be noted is that, although many audio plugins run directly in a gamut of operational systems, it is more common for them to be used as plugins, hosted in pieces of software called DAWs. This is so because sound applications are seldom used in isolation. Generally, in a production environment, one or more generator plugins, representing instruments such as keyboards and drum kits, are combined with effect plugins, such as reverb and equalizers, to generate the desired final sound.

Those generator plugins can be controlled in real-time or be fed a preexistent set of commands representing the performer's actions, generally in a high-level music description language such as the MIDI format (more details about the format can be found in Section 2.5). DAWs, in this setting, are responsible for receiving and processing the inputs and forwarding them to its hosted plugins, besides providing the interface for the exchange of relevant information between the various plugins. It is also common for other conveniences, such as sound recording, parameter automation, and macro capabilities, to be offered by DAWs.

Although the VST format, introduced by Steinberg in 1996, is the most common format for sound plugins, they are also available as Audio Units (AU), an equivalent of the VST format focusing on the Apple ecosystem; Real-Time AudioSuite (RTAS) and its successor Avid Audio eXtension (AAX) formats used by Avid's Pro Tools DAW, as well as many other, less popular, formats.

Development of plugins in those formats involves the use of a Software development kit (SDK), provided by the respective vendors, that exposes classes and functions, usually in the C++ language. Given the architectural similarities between the most common plugin formats, the use of higher-level libraries that unify the development process became the norm.

One such framework is the JUCE (Raw Material Software Limited, 2022) library. Besides enabling the unified development for the VST, AU, RTAS and AAX plugin formats, applications using this library can be compiled to software that runs natively in Windows, macOS and Linux desktops, as well as IOs and Android devices.

The library also provides cross-platform classes that can be used to implement the application's Graphical user interface (GUI). The framework is also very mature and widely used for the development of both commercial and open-source plugins. For these reasons, it was chosen for the implementation of the plugin object of this work.

As software running sandboxed in a host environment, the plugin must exhibit a

reactive nature, responding mostly to MIDI messages, delivered by the host, emulating the playing dynamics of the instrument. Those messages can correspond to real-time inputs produced, for example, by a physical MIDI controller instrument such as a keyboard. They can also be read from an existing MIDI file that can be either a record of a performance or a manually crafted file.

The other technical pillar of the plugin introduced here is the available neural networks ecosystem that is, in some important aspects, very different from the digital musical instrument's ecosystem. While C++ is the *de facto* programming language for the development of music software, neural networks research tends to rely extensively on more high-level, interpreted languages, such as Python.

The usual architecture of frameworks used in neural networks research involves the implementation, in C++ or other low-level compiled languages, of individual modules, responsible for the more intensive calculations.

Frameworks such as the Pytorch (FAIR, 2016) library, bundle a collection of such modules, besides providing a layer of abstraction over the functions exposed by those modules. This enables the language in which the models are programmed by the final user to be different from the languages in which the modules are implemented; usually a high-level, interpreted, dynamically-typed language such as Python is used as the interface language offered to the researcher.

This approach has the advantage of simplifying research while adding little overhead to the process of training and inferencing with neural networks, since most of the critical, resource-intensive tasks are performed by optimized, low-level code.

On the other hand, however, the customization of individual modules can be challenging, since it must be done in a language with which the researcher has potentially no familiarity. Another problem is the implementation of the final models in a production environment, where any improvement in efficiency is desirable. In those situations, as well as in situations where speed is one of the main concerns, such as in the context of digital musical instruments, it is desirable that the whole ecosystem is coded in compiled languages, such as C++.

As a strategy to extract the best aspects of both approaches, the Open Neural Network Exchange (ONNX) (The Linux Foundation, 2019) format was introduced, to enable the compatibility of trained neural network models between different libraries, potentially coded in different programming languages.

The format was formalized in 2017 by a joint initiative between Microsoft (Boyd,

2017) and Facebook (Bird et al., 2017), and nowadays includes the support of companies such as IBM, Huawei, Intel, AMD, Arm, and Qualcomm (Shah, 2017). The format defines a syntax to describe computation graphs, and specific operators to be used in those graphs, that are guaranteed to be implemented by libraries that adopt the standard (The Linux Foundation, 2019).

Outside of the strict MIDI specification, the JUCE library allows one to define other controllers, that function in a similar manner to the Mod Wheel, and can be controlled and automated via the GUI associated with the instrument.

We define three such controllers: one to control the decay of the sounds produced — useful in the emulation of instruments that consist of an initial excitation followed by a reverberation phase, such as pianos and acoustic guitars — and two more to control different aspects of the various instruments that the plugin will be able to emulate.

In the case of the singing voice, for example, one of those controllers is used to control the gender of the voice, while the other controls the vowel that is being emulated.

This summarizes the high-level interface exposed by the instrument to the performer, in case of real-time usage, or to the high-level notation language.

4.2 OMNES SONOS: A REAL-TIME AUDIO PLUGIN APPLICATION

The plugin implemented in this work is divided into two entities: the plugin per se, a piece of software that handles the logic of translating the representation of the user interaction into the signal that constitutes the final output of the model, and trained neural networks, that function as a set of engines encoding the logic specific to each instrument that can be emulated using the plugin. Details of the sample libraries that each of those engines are meant to emulate can be found in Table 2.

As mentioned, audio plugins are generally hosted by a DAW, operating in a sandboxed environment, with little information about the actual hardware in which they are running, apart from the information conveyed by their host. In addition to the performer's information, the host provides the expected sampling rate, the number of audio channels that should be filled, and the number of samples needed at each time. The goal of the plugin is to use this input to efficiently generate the actual samples.

A plugin can't enforce a pre-defined sampling rate, but must instead provide meaningful samples at the sampling rate requested by the host. Although this sampling rate in which a plugin must operate is normally constant, it can in fact change at any given instant, to respond, for example, to a lack of available hardware resources, or to a settings

change on the part of the user. For those reasons, the implementation must be as efficient and flexible as possible.

Likewise, it is also impossible for a plugin to enforce, or even be certain of, the size of the buffers used to accumulate its samples and, hence, the number of samples it needs to generate. Those characteristics translate, in the context of the present work, to strong indications that a frequency-domain approach is preferable. Another argument in this direction is that sound signals, owing to their oscillating nature, exhibit redundancies that are best accounted for using intrinsic cyclic models.

Using the algorithms developed earlier in this work, we are currently able to remove the influence of the temporal envelope on a quasi-periodic wave and segment it in its pseudo cycles. From the inputs to the plugin — more specifically, the expected frequency and sampling rate — we can derive the pseudo cycle the plugin is supposed to reproduce at any given time.

While one could train a network to directly learn this mapping, this direct approach is potentially problematic, for the main reason that, although multi-rate frameworks exist at least since the 1980s, the model would be biased towards a particular sampling rate.

The direct approach also makes it very difficult to control the processing requirements of the model, and gracefully degrade the quality of the emulation to use less processing power and conform to lower-end hardware, for example.

A third drawback is that the direct approach forcefully encodes the phase information of the signal, burdening the model with the task of learning superfluous information, as the phase is not used by the final model to generate the sound.

On the other hand, since the algorithm's focus is on real-time sound synthesis, we would ideally want to model the problem in such a way that domain conversions are employed primarily during the training step, leaving the evaluation as streamlined as possible.

In other words, it isn't necessary to visually emulate the waves in each of the pseudo cycles since "sound" is not a univariate function, in the psychoacoustic sense: a multitude of different signals can have a similar sound, as long as they exhibit similar underlying characteristics. The most relevant characteristics of each pseudo cycle can be more compactly represented in a time-frequency domain, for example.

Figure 27 shows individual normalized pseudo cycles isolated by the algorithm presented in Section 3.4 both in the frequency domain — which would correspond to the direct approach mentioned above — and in the frequency domain, in the form of their

power spectra.

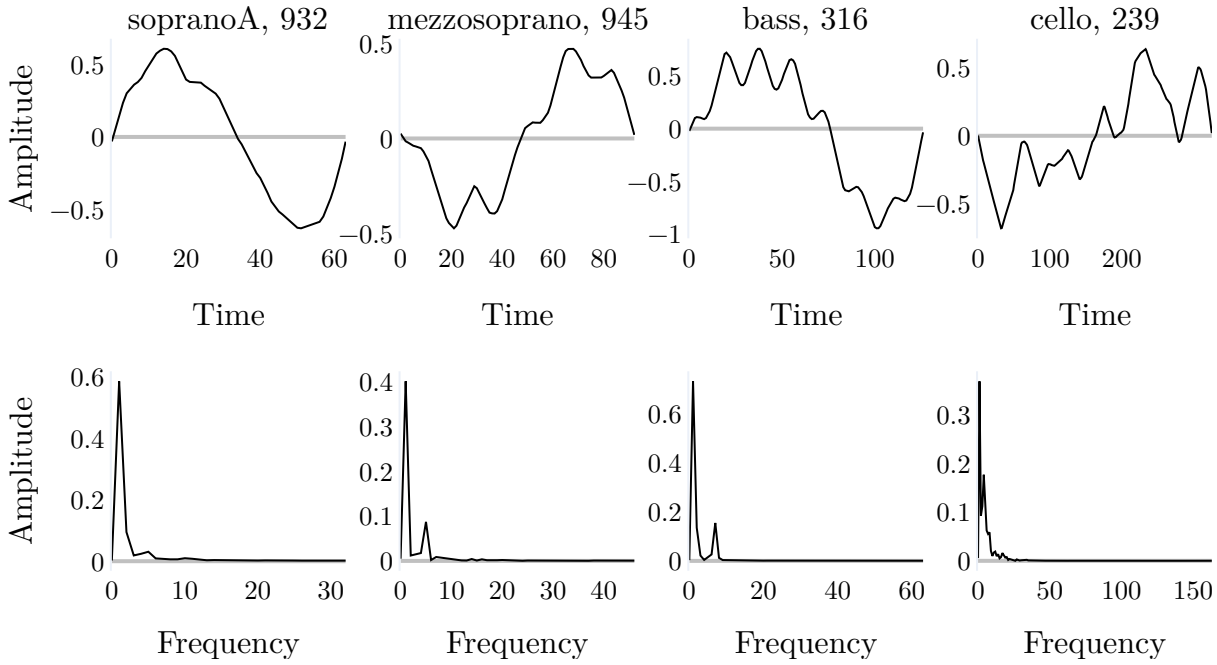

Figure 27: Selected pseudo cycles from various singing voices and a cello, represented in the time and frequency domains. The number of the pseudo cycle is presented after the name of the instrument, after a comma.

Interesting insights can be derived from Figure 27: it can be seen that the most meaningful values, in the frequency domain representation, are concentrated at the beginning of the spectra, making it easier to discard less relevant data in order to improve the efficiency of the algorithm.

This representation also facilitates the elimination of the DC offset, represented by the amplitude at the index 0 of the power spectra, eliminating fluctuations between adjacent pseudo cycles. It must be noted that, to ensure that no artifacts arise when the original signal is reconstructed, the segmentation algorithm must be precise in determining the boundaries between pseudo cycles.

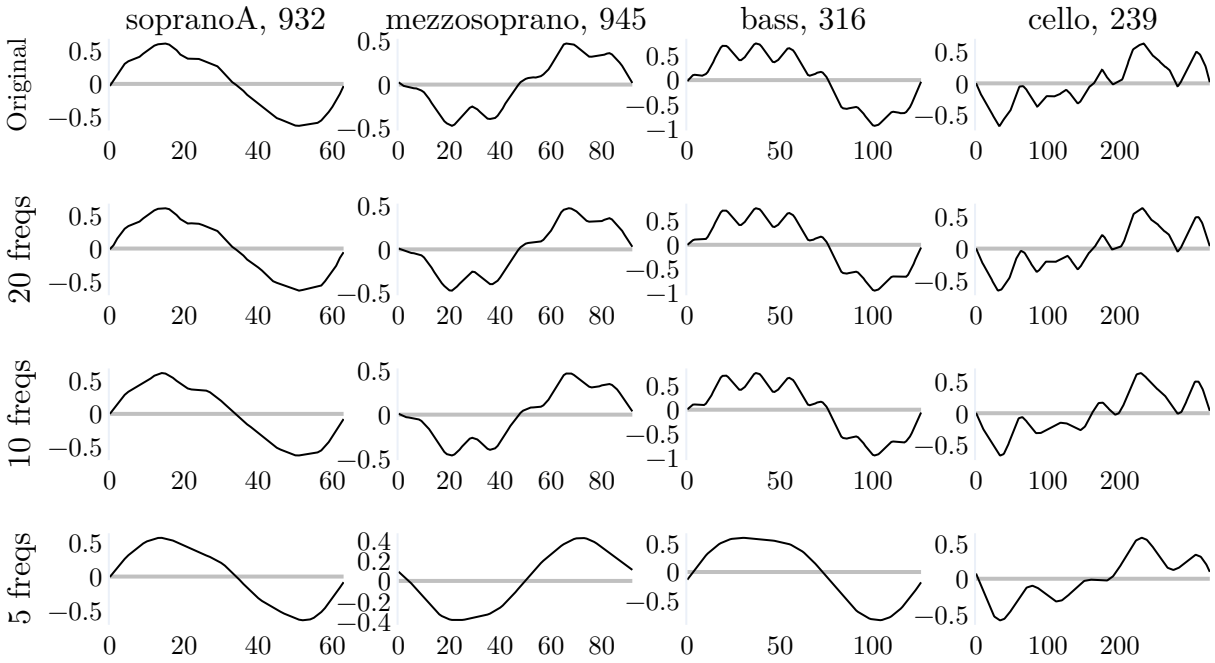

Figure 28: Selected pseudo cycles from various singing voices and a cello, and their reconstructed versions from partial frequency domain representations.

Figure 28 shows the original pseudo cycles and their reconstructions using different numbers of frequencies: it can be seen that they can be accurately reconstructed using a fraction of the total frequency domain information. This discussion concludes the high-level characterization of the model to be implemented in the remainder of this chapter, since it defines the inputs and the outputs they must be mapped to.

Given the complex relationship between the inputs and outputs, the core of the plugin relies on a neural network that receives a vector encoding the inputs as discussed in the previous section, and outputs the frequencies' amplitudes that characterize the pertinent pseudo cycle of the instrument to be emulated.

Having established the more important characteristics of the networks that serve as the core of the proposed plugin, it is now possible to design how the outputs of those networks are to be efficiently converted into sound signals.

This is no trivial task, as the demands of real-time sound synthesis impose a number of technical constraints in the implementation. Conceptually, the most distinguishing characteristic differentiating sound synthesis software from general programs is the existence of two processing threads running in the former.

A low-level thread is responsible for the sound generation process, and must be made as efficient as possible. All operations that aren't strictly necessary to the generation of the requested samples must not take place in this thread. Blocking operations, most

notably reading from and writing to disk, must be delegated to the high-level thread, for example.

The high-level tread is responsible, beyond other things, for pooling user input, managing the GUI, and the majority of general-purpose tasks common to general programs. In the case of sound synthesis plugins, the information received from the host, concerning relevant aspects for the synthesis such as the sampling rate in which the host is currently operating and the number of sound channels that need to be filled, is collected opportunely by this thread and made available to the inner thread in an unobtrusive way.

Determining exactly how much processing can be performed in the low-level thread — the thread responsible for the generation of the sound samples — before the introduction of unacceptable lag and other artifacts is a hard task, and often follows a trial-and-error approach. Hence, during the design phase of the plugin, it is a good practice to introduce a quality setting that can be tweaked to account for this potential bottleneck. In the case of the present work, this takes the form of the number of frequencies used to reconstruct the sound.

To obtain the amplitudes of those frequencies, however, it is necessary to perform inference, using the trained neural networks, at each successive period — obtained from the predominant frequency — of the sound that is to be synthesized. Noting that the period of a waveform is inversely proportional to its frequency, it is easy to see that this poses a challenge to the implementation, since the frequency of inferences necessary varies greatly according to the predominant frequency of the sound the plugin needs to emulate.

For a standard grand piano with 88 keys, for example, we have a highest predominant frequency of approximately 4186 Hz, implying as many cycles and, hence, neural network inferences per second when key 88 is pressed, while in the lowest frequency, triggered by key 1, only 27 inferences per second are necessary. To address this problem, the inferences are implemented asynchronously, in the following manner: once a particular note is requested for the first time by the low-level thread, the plugin requests, passing the relevant information, the corresponding frequencies' amplitudes from the network, and blocks the thread for a couple of milliseconds until those amplitudes are returned by the network and stored in a buffer.

At each successive new sample request, the inner thread uses the information about the amplitudes already stored in that buffer, and checks if the network has completed the last request. In case it has, a new request is made, and the network will asynchronously update the buffer as soon as a new set of amplitudes is available, and update its internal

state to signal that it is free to perform a new inference. The number of inferences, then, is automatically limited by the processing power of the underlying hardware, without the need for the design and implementation of sophisticated control logic.

A constant concern when emulating pitched instruments is enforcing the correct pitch. To guarantee that a reference to the base frequency is always organically present, all calculations necessary to transform the amplitudes to samples in the time domain are performed considering the current period, as established by the frequency associated with the currently pressed note.

The process is started by a request from the low-level thread for the generation of a number of samples, responding to a key down event that can be caused by a player pressing a midi controller key in real-time, or by a message from a pre-recorded MIDI file.

Two situations can happen when a key is pressed: if no other note is being played at the time of the key down event, or if there is another note already playing but the plugin is not in legato mode, the behavior is the same, and the plugin triggers an inference with the neural network, as noted before, sending the relevant data, and waits for the resulting amplitudes.

In the event of a note being already played, in legato mode, the key down event causes the dominant frequency to be shifted smoothly from the original note to the new note, and the new amplitudes to be requested from the network according to the new predominant frequency intended. From a psycho acoustical point of view, a slide effect is perceived by the listener.

It was hinted, during the high-level description of the plugin model given at Section 4.1, that not all original information from the original sound samples was needed for their reproduction. This is the case since the power spectra shown in Figure 27 do not include phase information.

One could reconstruct the samples from this information by arbitrarily filling this gap: assuming, for example, every sinusoid to have a zero phase. It is shown in Tarjano et al. (2019), however, that a similar result can be obtained from using random phases for each of the sinusoids constituting the final complex wave. To prevent inharmonicity, however, the same randomly generated phases must be used for the whole duration of a single note.

This last approach has two marked advantages: it enables the plugin to efficiently avoid the sensation of repetition, a common problem encountered in sample-based instruments known as the machine gun effect and, perhaps more importantly, it enables the

generation of multi-channel sounds from a model trained with mono samples. For that reason, for each channel, a random vector of phases is created, to be used during the whole period in which the note is sounding.

There are two additional controllers that enable the user to change the pitch of the note currently being played. The bend wheel changes the bend in real-time, proportionally to the amount by which it is moved. The vibrato wheel introduces a sinusoidal change in both the frequency and the amplitude of the current sound. The position of the vibrato wheel does not affect, however, the amplitude of the sinusoid that describes the vibrato effect.

Both of those effects are implemented as changes in the predominant frequency of the current note, having profound influence on the model, since they use the amplitudes that the network was trained to provide for those frequencies, thus emulating the natural dynamics of the underlying instrument.

Lastly, the attack control modulates the initial envelope of the sound, while the decay control limits the duration of the sound.

4.3 THE NEURAL NETWORKS

The design of neural network-based models is sometimes considered an art. Although much of the foundations arise from mathematical theories, in practice, one finds that the corresponding implementations are often tweaked ad hoc, empirically, in order to be more efficient.

This is the case, for example, of backpropagation, the optimization algorithm employed at the backward pass of the training process, where the parameters of the network are updated based on the gradients of the error. Although gradient descent, for example, is well established mathematically, it is seldom used naively, in practice, in the context of neural networks: extensions, such as the addition of momentum, are introduced to improve performance.

Another factor that contributes to this characteristic is the presence of many variables that independently influence the results. Generally, due to the time and effort required, it becomes infeasible to methodically investigate the interrelationship of all those factors, and one has to trace an inquiry path based partially on prior experience.

Regarding the design of the networks, one is presented with many choices of architectures and activation functions. In the training phase, a batch size, a loss function, and a learning rate must be chosen. Optionally, dropout and other regularization techniques

can also be used. Ideally, one would test all combinations of those parameters. In practice, however, this approach is seldom feasible because of time and resource constraints, and small subsets of those parameters are tested, based on prior findings in the literature and previous experience.

Since the aim of the plugin is not merely encoding a complete sequence, to be decoded unchanged at request, recurrent architectures, such as RNNs or the more recent Transformer (Vaswani et al., 2017) architecture, are not indicated. Those architectures use their past outputs as inputs for their next inference, being prone to degenerate into chaotic behavior once their outputs are further processed before being reutilized as inputs. To avoid that, it is common to generate a whole sequence before processing it.

At each instant during sound synthesis, however, characteristics of the inputs to the network, such as the velocity and other dynamics, are liable to change, and feedforward networks can handle those changes promptly, while avoiding instability, besides being more efficient.

In the case of the plugin proposed here, the raw data is a set of samples ideally representing all the possible articulations of a particular instrument, and also covering its whole frequency range in the case of discrete range instruments such as pianos and guitars. In the case of instruments with a continuous range of possible frequencies — such as the singing voice, and unfretted string instruments, such as violins and cellos — one would ideally have access to samples representing the range at arbitrary intervals. The available free sample libraries, however, are severely more limited, and the training phase has to be designed to be robust in relation to those limitations.

As commented before, the plugin has to deal with input parameters representing velocity, pitch, vibrato, decay, and potentially the two additional inputs; not all those parameters need to be fed to the neural network, however. To generate samples, the network needs the values of the first general articulation (gender, in the case of voice, for example), the second general articulation (vowel, in the case of voice, for example), the base frequency, the pseudo cycle number, and the velocity.

To generate this information, the samples from the instrument library are segmented into their pseudo cycles. Each pair input-target that must be learned by the network consists of the pseudo cycle description introduced below and the frequency domain representation, respectively.

The input information can be encoded in the vector $\mathbf{I} = \{g_0, g_1, f, p, v\} \in \mathbb{R}^5_{\geq 0}$, where $0.0 \leq g_0, g_1, f, p, v \leq 1.0$. The entries g_0 and g_1 are entries reserved for instrument

specific articulations; in the case of the networks emulating the singing voice we have that $g_0 = 1.0$ for male voices and $g_0 = 0.0$ for female voices. Likewise, g_1 encodes one of the five vowels most common in the English language: $g_1 = 0.0$ represents "a", $g_1 = 0.25$ for "e", $g_1 = 0.5$ for "i", $g_1 = 0.75$ for "o" and $g_1 = 1.0$ for "u".

The entry f encodes the base frequency corresponding to the key pressed by the performer. Recalling that the MIDI format provides 128 keys, encoded as an integer from 0 to 127, this information can be encoded as key /127, where key $\in \mathbb{N}$, $0 \leq$ key ≤ 127 represents the key pressed by the performer, linked to the expected frequency by Equation 1.

Despite the fact that it is common for the sample libraries to provide the nominal frequency of each sample in the form of the key or the note they correspond to, the frequency of each pseudo cycle is derived from the predominant frequency of the original sample, as obtained by the FFT. As a measure to render the model more robust to eventual processing errors, samples that exhibit a measured predominant frequency diverging from their labels by more than one note are automatically discarded.

The following entry in vector $\mathbf{I}$ encodes the index of the pseudo cycle to be learned. This involves segmenting all samples of the library, and determining which sample has the larger number $P_{\max} \in \mathbb{N}$ of pseudo cycles. Each index of the pseudo cycles is then divided by $P_{\max}$ to assure that $0.0 \leq p \leq 1.0$.

The last entry v in the input vector encodes the velocity, or intensity, with which the note was played. Besides having a direct effect on the loudness of the sound emitted, which will be predominantly addressed by the plugin, instead of the neural network, the velocity customarily also affects the characteristics of the sound. This information is also extracted from the whole sample, in the form of the maximum absolute amplitude of its individual samples and, depending on the library used to read the samples, comes normalized between 0.0 and 1.0.

The targets of the network follow from the discussion in the preceding section, illustrated in Figure 28, and consist of an arbitrary number t of entries of the power spectra of each pseudo cycle, starting from the index 1. This information can be encoded in the vector $\mathbf{T} = \{a_1, a_2, \cdots, a_t\} \in \mathbb{R}^t$, where $0.0 \leq a_1, a_2, \cdots, a_t \leq 1.0$ and $a_1 + a_2 + \cdots + a_t = 1.0$. The number t of frequencies used in the reconstruction will be determined empirically, as the maximum value that does not interfere with the real time performance of the plugin.

Since both the inputs and outputs of the network consist of positive values between 0.0 and 1.0, it was chosen for the activation function, to be used after each layer, a

modified version of the hyperbolic tangent, as shown in Figure 29. Especially in the output layer, this function assures that the outputs won't exceed 1.0, preventing clicks and other artifacts in the generated sounds. This function was chosen as an alternative to the more common Sigmoid function, since its inflection point is at $x = 0.5$, instead of $x = 0.0$, as is the case of the Sigmoid function.

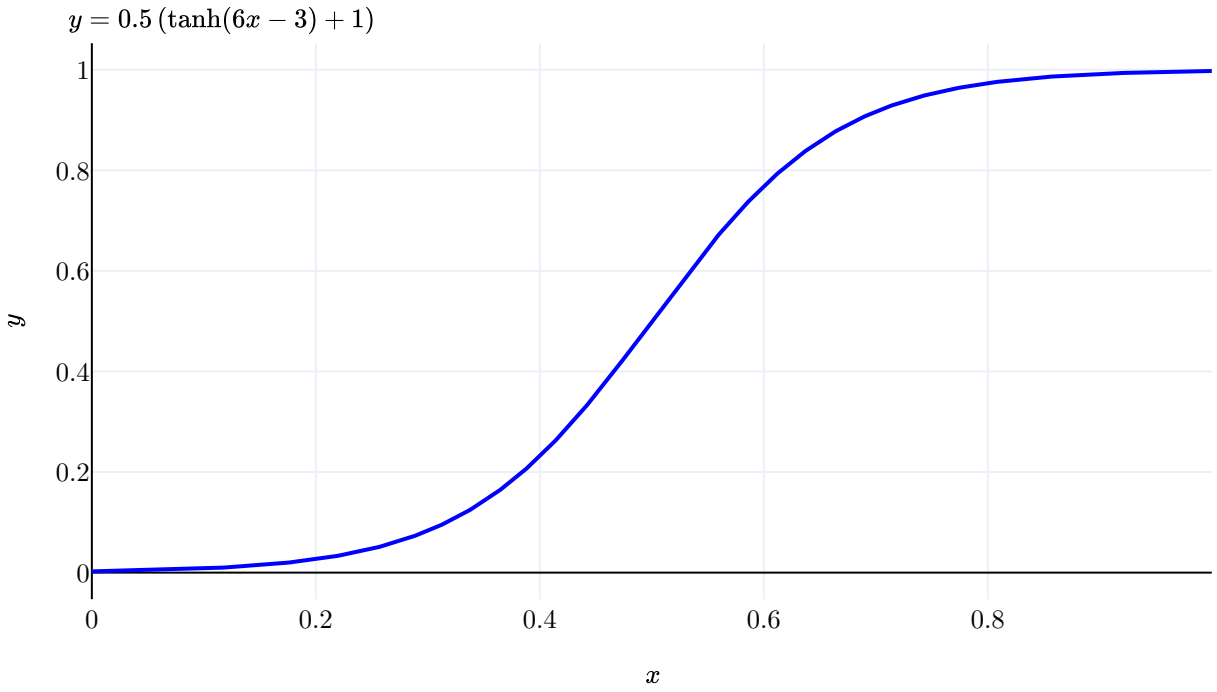

Figure 29: Activation function used in all layers of the network.

The operation of extracting inputs of the network from the original samples can be time-consuming; for this reason, this information is stored compactly in binary format, as a form of buffer, to speed up further training. That strategy is also employed in the generation of the targets. Once the input information for the sample library representing a specific instrument is created, it can be reused for every other network involving that instrument. The outputs, however, are dependent on the number of frequencies to be considered and are reutilized whenever possible.

Despite the fact that the number of samples available in the libraries used in this work is inferior to the ideal, the volume of data is still very high: the network must be able to learn from a massive number of input-target pairs, while relying on as few parameters as possible in order to be able to perform inferences in real-time.

There is also the problem that the psychoacoustic quality of the results cannot be assessed in the frequency domain and the outputs of the networks have to be transformed to the time domain to be compared with the targets. For this reason, the networks are

generally assessed using two metrics: the traditional loss metric compares the outputs of the network with the respective targets, in the frequency domain, and is the metric used during training for the computation of the gradients; all outputs of a given network are used in this computation.

Given the sheer amount of data, and the additional computational cost of domain conversions, however, it is not feasible to compare each output of the network with its time-domain counterpart. To circumvent this problem, this work makes use of a sampling strategy: at the end of each epoch, two sets of 200 samples are extracted from the outputs of the network, each sample representing a pseudo cycle, converted to the time domain, and their average errors are calculated.

The first is a random set of samples, that changes at each epoch, and helps to identify eventual outliers, while the second is a fixed set of samples consisting of 200 outputs equally spaced. Two metrics are calculated using those two sets of samples: the random error and the deterministic error, respectively. Those metrics provide a better idea of the training process.

While designing the general architecture of a neural network, it is often helpful to go through a rough testing phase, where more parameters can be investigated, albeit more superficially. The few more promising results from this preliminary test step can then be further investigated, via a series of more comprehensive tests. This approach is especially relevant when defining the best optimizer to be used in the training of the networks, since there is a considerable variety of options to choose from.

The approach employed in this work is as follows: in each test step run, the amount of time in which a network is permitted to run is constrained, and the networks able to reach the smaller errors in this time window are chosen, either to be evaluated in the next step, or to be ultimately used in the algorithm. Periodically, the state of the training process is saved, enabling recovery from eventual errors.

4.3.1 Optimizer

Using the described approach, we can start testing for the best optimizer. Pytorch, the framework used in the design and training of the networks in this work, offers an ample choice of pre-implemented optimizers, among which 11 are potentially suited for the characteristics of this model.

This first step compares those most prominent optimizers: Adagrad (Duchi et al., 2011), Adadelta (Zeiler, 2012), Adam (Kingma et al., 2014), Adamax (Kingma et al.,

2014), AdamW (Loshchilov et al., 2017), ASGD (Polyak et al., 1992), NAdam (Dozat, 2016), RAdam (Liu et al., 2019), RMSprop (Graves, 2013), Rprop (Riedmiller et al., 1993) and Stochastic Gradient Descent (SGD) (Sutskever et al., 2013).

Even for a preliminary step, with limited time for each network to run, the number of optimizers is relatively elevated. For this reason, we chose to test the optimizers with a limited dataset that is, nonetheless, representative of the kind of task to be accomplished.

Figures 30 and 31, where the average deterministic and random errors are shown, give an overview of the performance of those optimizers in the training of a test piano library. The sample library is composed of 88 samples, representing the 88 keys of a standard grand piano with only one velocity for each key. The network has a feedforward architecture, composed of 3 layers with 50 neurons each, and the custom tanh activation discussed before. The output is composed of the 10 first amplitudes of the frequency domain representation.

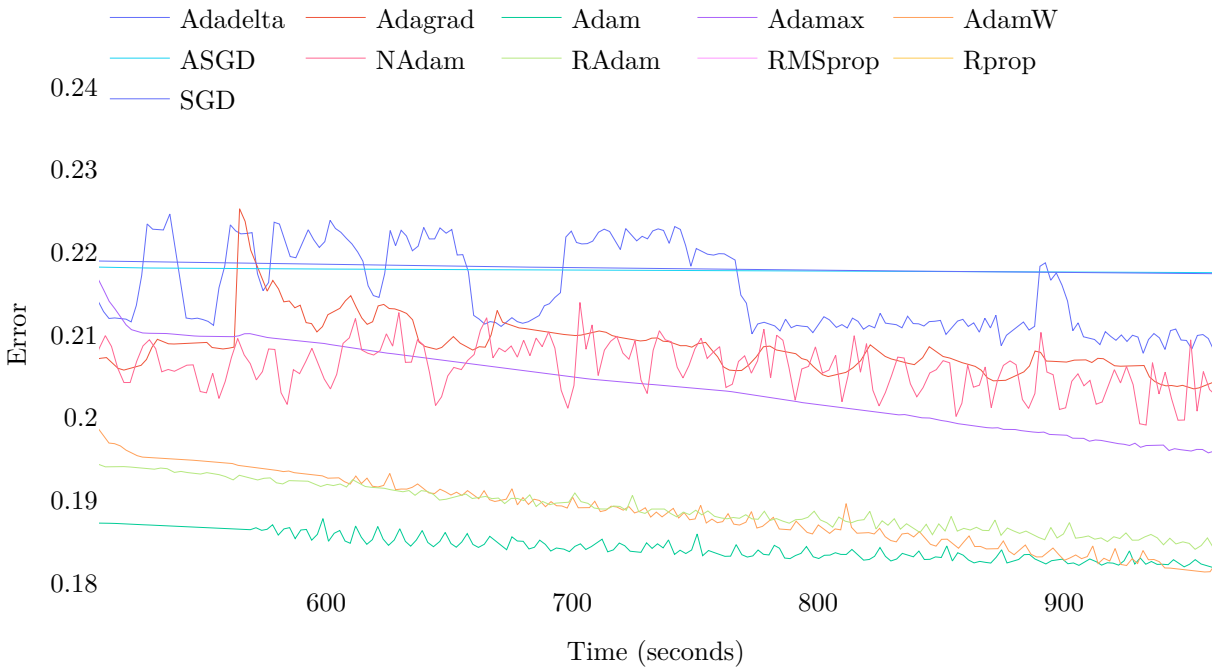

Figure 30: Comparison of the performance of the average deterministic error of the optimizers. The errors from the RMSprop and Rprop optimizers can't be seen in the figure, as they are above the shown region.

Given the data density, the figures are best visualized through their interactive versions available on the website prepared for this work (Tarjano, 2022). It can be seen that the Adam, AdamW, and RAdam optimizers offer the best performance overall, with the Adam and AdamW optimizers delivering virtually indistinguishable results. Those 3 optimizers are variations of the Adagrad algorithm that employ an exponential moving

average of the gradients as a means of regularization (Reddi et al., 2019).

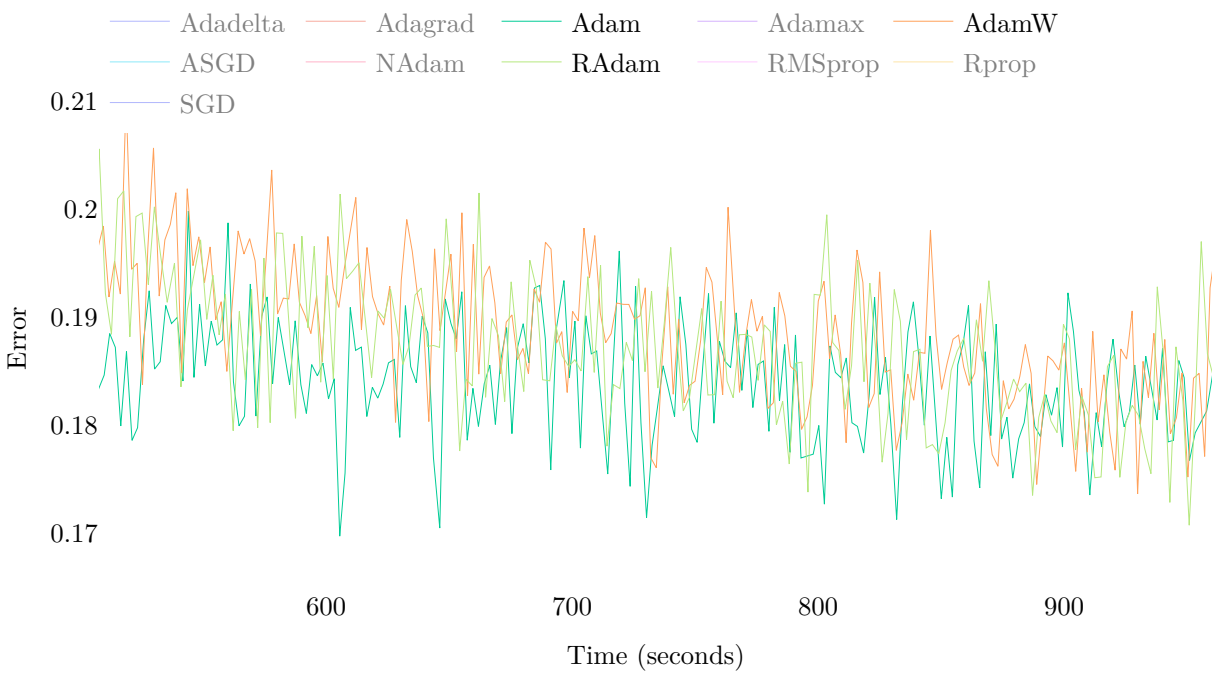

Figure 31: Comparison of the performance of the average random error of the Adam, AdamW and RAdam optimizers. The behavior of the other optimizers can be seen in the interactive version of the figure.

Based on those preliminary experiments, we can single out the Adam, AdamW and RAdam optimizers in order to further investigate their performance under more diverse circumstances, such as using different configurations of network parameters, and with the use of a realistic sample library.

The sample library used to compare the Adam, AdamW and RAdam optimizers is composed of 461 sounds recorded from a Steinway model D grand piano, representing various velocities for each key. After being segmented into pseudo cycles, the library is transformed into 562576 pairs of inputs and targets, occupying a total of 21.4 MB of disk space for the inputs and 754.6 MB for the targets. This information is condensed in Table 2, alongside information about the other sample libraries used in the rest of this work.

Figures 32, 33 and 34 show the comparison of the quality of those three optimizers in different architectures, using the three error metrics discussed before. The first metric shown in Figure 32, is the loss, the average error between all the targets and the outputs of the network in the frequency domain, as used during training; this metric can be considered the main measure of error, since it synthesizes the error of the whole network. As commented earlier, however, it doesn't directly convey the time-domain error of the network.

For this reason, Figures 33 and 34 present the two time-domain metrics, the deterministic and random error, respectively, described at the beginning of this chapter.

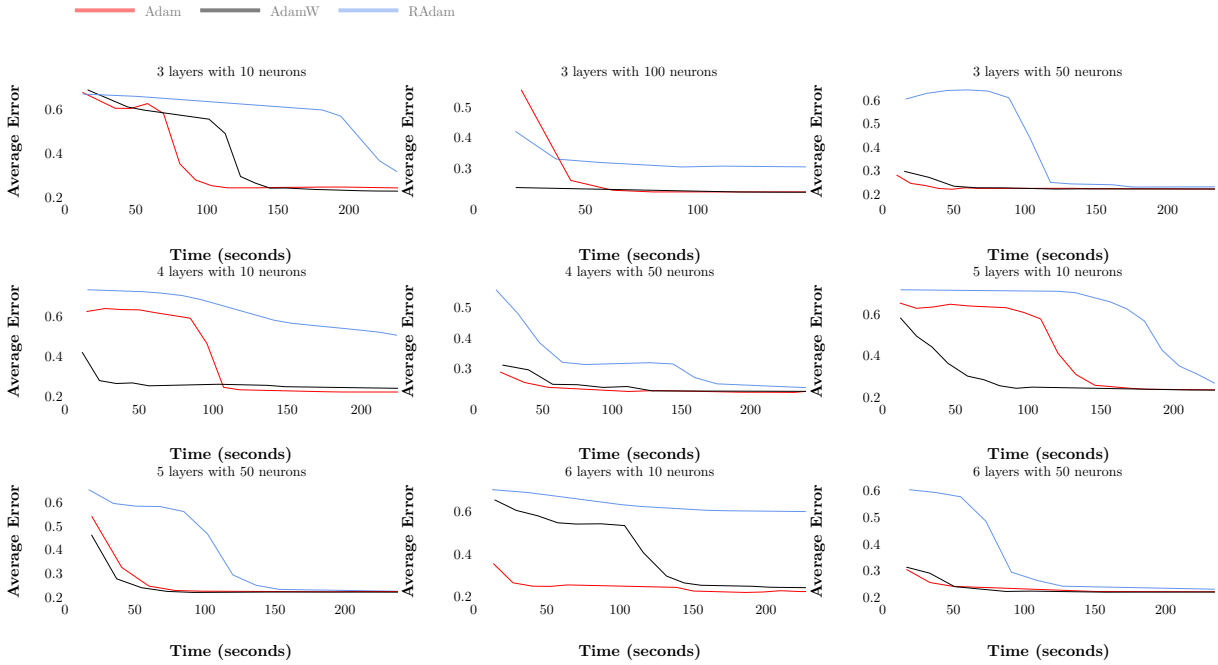

Figure 32: Comparison of the performance of optimizers, using the average deterministic error metric, for the Steinway model D sample library. Lower values are better.

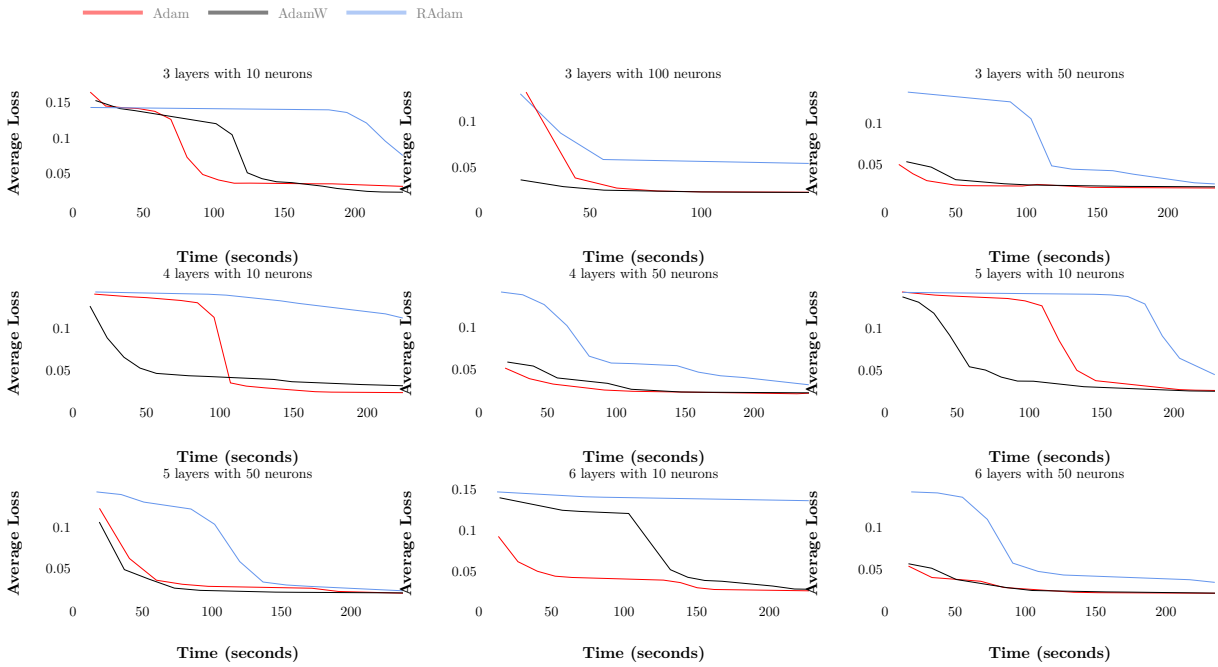

Figure 33: Comparison of the performance of optimizers, using the loss metric, for the Steinway model D sample library. Lower values are better.

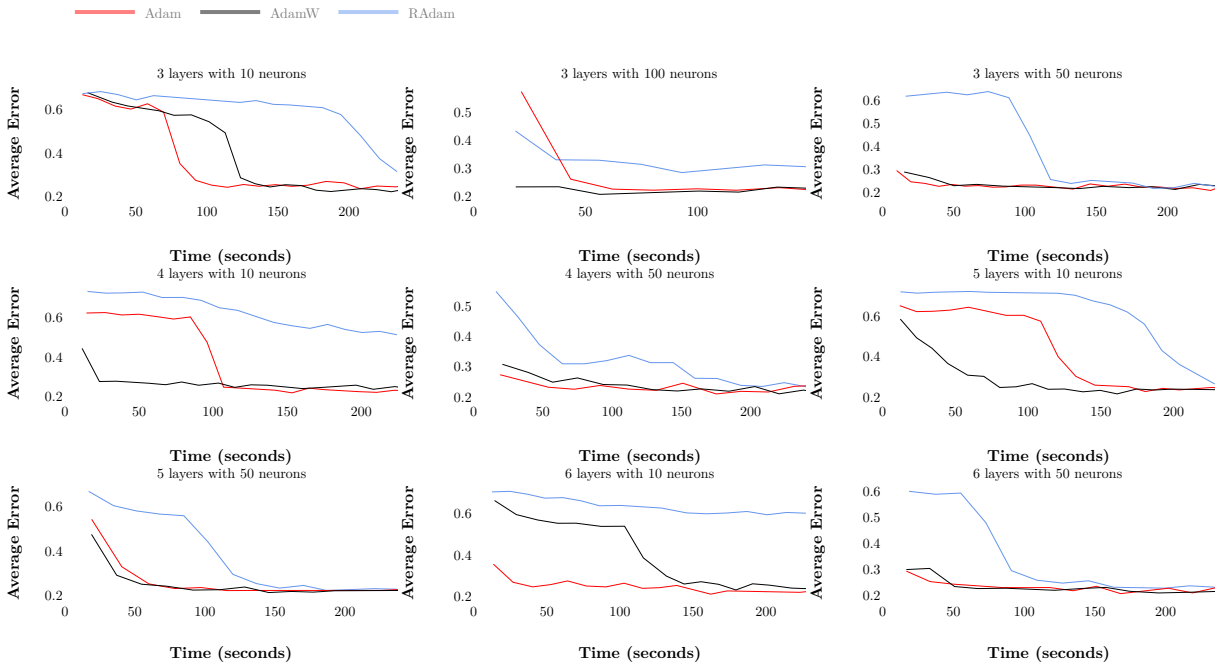

Figure 34: Comparison of the performance of optimizers, using the average random error metric, for the Steinway model D sample library. Lower values are better.

Table 1 presents Pearson's correlation coefficient between the data shown in Figures 32, 33 and 34. Observing the figures, it is possible to see that the behavior of the three metrics is similar; the correlations on the table confirm this intuition.

Regarding the correlation between the loss (**L** on the table) of all the outputs of the network in the frequency domain and the average error of 200 equally spaced outputs (**D**) transformed to the time domain, presented in the column labeled **LxD r**, one can see that the smallest coefficient is 0.91, while the average of all correlation coefficients is 0.97.

Similarly, for the column labeled **LxR r**, representing the correlation between the loss and the average error of 200 random sampled outputs (**R**), the average correlation is 0.92, while for the column comparing the deterministic and random errors (**LxD r**), the average correlation coefficient is 0.94.

Table 1: Correlation between the three metrics used throughout this work, for different neural network architectures. The labels at the top row are as follows: **l** - layers; **n** - neurons; **opt** - optimizer. The remaining labels use the following abbreviation: **L** - average loss, in the frequency domain; **D** - average error of 200 equally spaced outputs, transformed to the time domain; **R** - average error of 200 randomly sampled outputs, transformed to the time domain. The **r** and **p** indicators stand for the Pearson's correlation coefficient and the two-tailed p-value, respectively.

l	n	opt	LxD r	LxD p	LxR r	LxR p	DxR r	DxR p
3	10	Adam	1.00	1.1e-34	0.99	4.4e-32	1.00	3.2e-39
3	10	AdamW	1.00	2.9e-37	1.00	9.1e-35	1.00	1.4e-44
3	10	RAdam	0.97	9.2e-12	0.97	1.7e-12	1.00	4.3e-21
3	100	RAdam	0.97	5.9e-05	0.94	5.5e-04	0.98	1.5e-05
3	100	Adam	1.00	8.4e-23	1.00	1.5e-18	1.00	2.8e-19
3	100	AdamW	0.91	3.3e-09	0.31	0.16	0.28	0.21
3	50	Adam	0.99	1.2e-84	0.83	1.3e-28	0.83	7.1e-29
3	50	AdamW	0.98	7.3e-17	0.91	5.1e-10	0.92	3.2e-10
3	50	RAdam	0.99	1.4e-12	0.99	2.9e-13	1.00	5.8e-20
4	10	Adam	1.00	1.4e-34	1.00	8.3e-37	1.00	5.3e-43
4	10	AdamW	0.94	2.1e-17	0.92	1.4e-14	0.97	7.9e-23
4	10	RAdam	0.95	2.9e-09	0.94	1.2e-08	1.00	8.7e-18
4	50	RAdam	0.92	1.2e-06	0.90	5.3e-06	1.00	1.9e-14
4	50	Adam	0.98	2.2e-17	0.78	1.0e-05	0.81	3.0e-06
4	50	AdamW	0.98	3.0e-14	0.93	1.6e-09	0.94	2.4e-10
5	10	Adam	1.00	2.3e-41	1.00	4.8e-39	1.00	1.8e-45
5	10	AdamW	0.98	2.0e-25	0.98	1.1e-23	0.99	2.0e-33
5	10	RAdam	0.98	2.4e-14	0.98	1.1e-13	1.00	2.1e-27
5	50	RAdam	1.00	2.0e-13	0.99	4.1e-12	1.00	4.0e-16
5	50	Adam	0.98	3.5e-18	0.98	3.9e-17	1.00	1.4e-24
5	50	AdamW	0.99	4.1e-20	0.99	2.2e-20	0.99	1.2e-22
6	10	Adam	0.97	3.7e-19	0.90	1.7e-12	0.95	3.8e-16
6	10	AdamW	0.99	2.6e-27	0.99	7.5e-25	1.00	2.3e-32
6	10	RAdam	0.97	5.7e-11	0.96	1.4e-09	0.99	2.6e-13
6	50	RAdam	1.00	1.4e-17	1.00	8.5e-15	1.00	3.2e-15
6	50	Adam	0.95	7.1e-13	0.86	9.2e-08	0.88	1.4e-08
6	50	AdamW	0.95	2.2e-11	0.93	1.2e-09	0.96	1.7e-11

Since the error metrics are a more direct measure of the final quality of the model, the rest of this work will focus on those. It will be seen in Section 4.3.2 that this approach also facilitates the analysis of the performance of different loss functions.

As further tests will also make use of other sample libraries, representing not only other instruments but also singing voices in different registers, it is appropriate to introduce the sample libraries used in the rest of this work, and briefly comment on their characteristics. Their most important indicators can be seen in Table 2. The table

presents 7 libraries, of which 3 model the human voice and 4 model musical instruments.

The choice of instruments and voices, constrained by the availability of free sample libraries with sufficient quality and variety of articulations, was guided by the desire of testing the proposed framework in as wide a range of instruments as possible.

The SteinwayD library was based on samples recorded from a Steinway & Sons model D grand piano, originally made available by the bitKlavier (Trueman et al., 2021) project under the GPLv3 license, an open-source initiative to create a flexible digital instrument using the JUCE (Raw Material Software Limited, 2022) framework. Starting from A0, each minor third interval was recorded at 16 velocity layers in stereo format. More information about the recording setup used can be found at `https://bitklavier.com/the-bitklavier-grand`.

The Guitar library used samples from the Philharmonia (Rouvali, 2021) website, a symphony orchestra founded in 1945. Their website offers free-to-use samples recorded from standard orchestral instruments, as well as the most common popular instruments, captured by members of the orchestra. The samples used were originally in mono MPEG-2 Audio Layer III (MP3) format.

The SteinwayB library was based on samples made available by the University of Iowa Musical Instrument Samples (MIS) (Fritts, 1997), recorded in 2001 from a Steinway & Sons model B grand piano, and offered in stereo Audio Interchange File Format (AIFF) format. More details of the recording setup can be found at `https://theremin.music.uiowa.edu/MISpiano.html`

The Violin library was also based on samples offered by the MIS initiative, presented in mono AIFF format, with more recording details available at `https://theremin.music.uiowa.edu/MISviolin.html`.

Lastly, the three voice base libraries — Soprano, Vox and Metavox — were based on samples from the Vocalset (Wilkins et al., 2018) project, released under the Creative Commons Attribution 4.0 license, in the WAV mono format. The Soprano library roughly models the soprano register and, thus, has only 4 male voice samples. The Vox library has 182 female samples and the Metavox has 67 female voice samples.

Table 2: Information about the sample libraries used throughout this work. Storage sizes refer to the arrays stored on disk using the NumPy binary format (.npy).

Name	Instrument	Samples	Ipts-Tgts	I (MB)	O (MB)
SteinwayD	Piano	461	562576	21.4	754.6
Guitar	Acoustic Guitar	41	9873	0.384	12.6
SteinwayB	Piano	124	111796	4.3	83.5
Violin	Violin	96	84372	3.2	45.7
Soprano	Voice	119	177658	6.8	119.2
Vox	Voice	227	302971	11.5	217.1
Metavox	Voice	108	125313	4.8	97.4

Figures 35, 36 and 37, for example, show the comparison of the evolution of the average deterministic and random errors for a variety of architectures training on some of the sample libraries presented in Table 2.

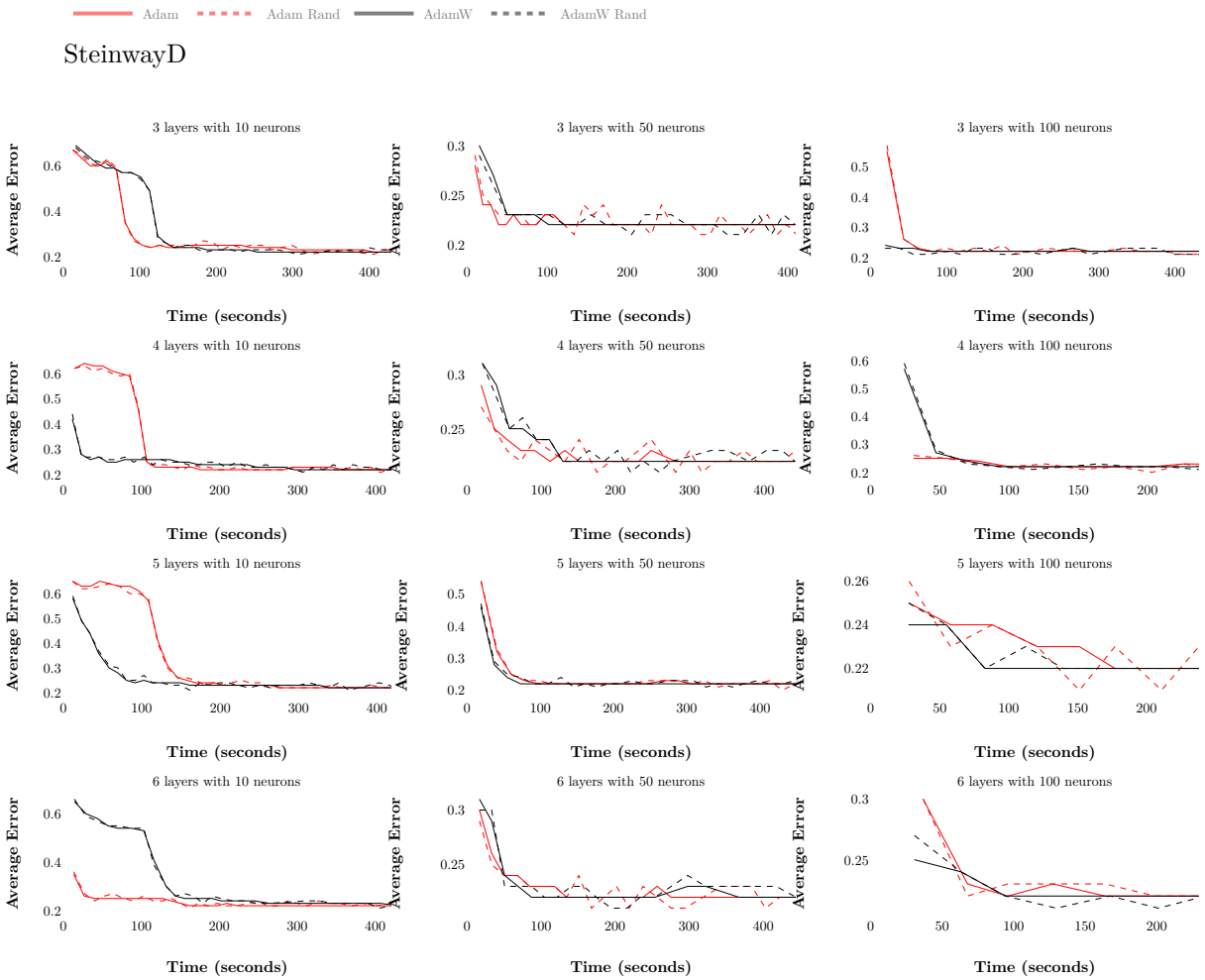

Figure 35: Comparison of the performance of optimizers, using the average deterministic and random error metrics, for the Steinway model D sample library.

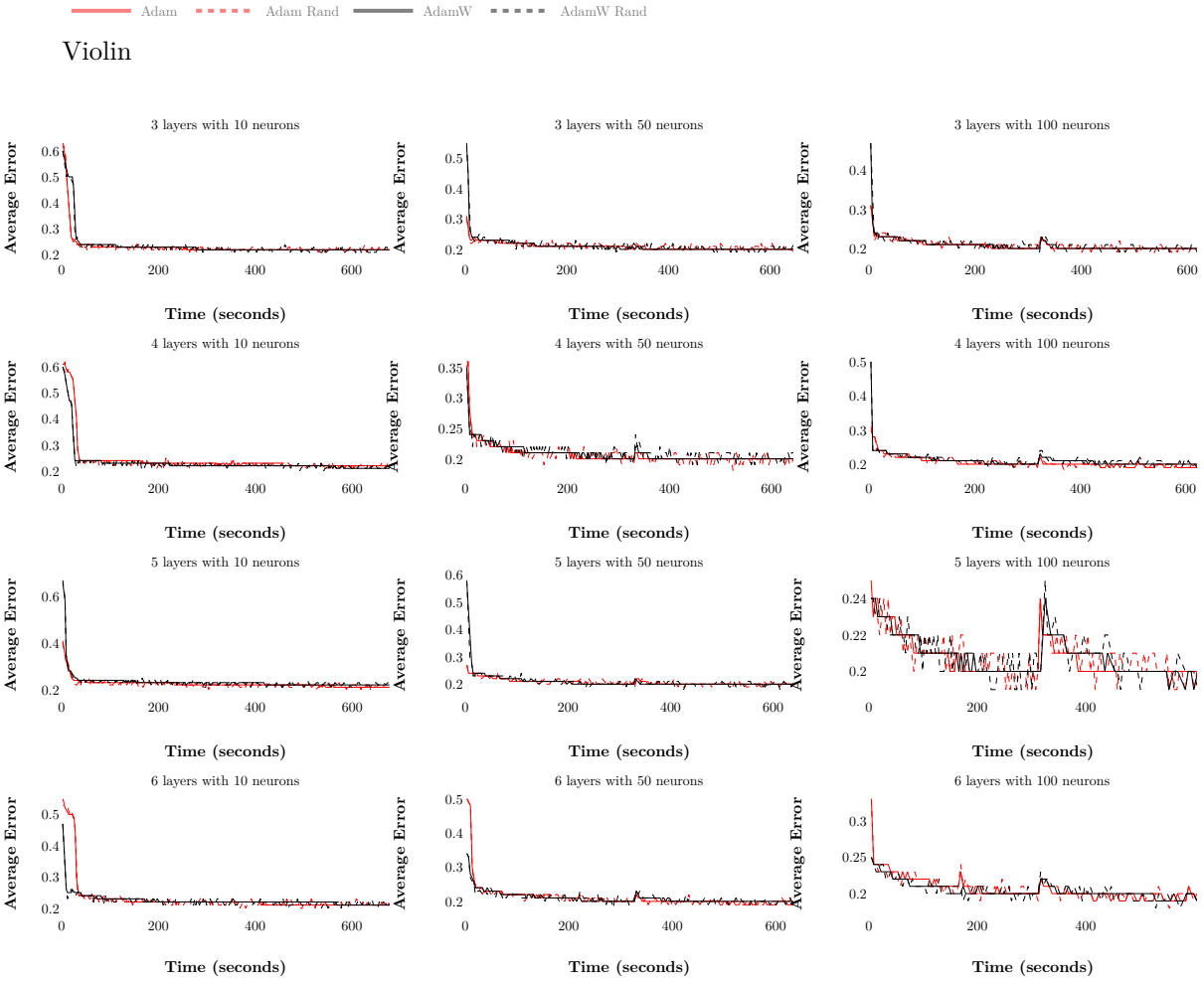

Figure 36: Comparison of the performance of optimizers, using the average deterministic and random error metrics, for the Violin sample library.

Due to the similarities in the general performance of both optimizers, the network will make use of the Adam optimizer, for its generality (Reddi et al., 2019) and also its relevance in the deep neural networks community (Zou et al., 2019), which cause its implementations to be generally more mature and tested.

4.3.2 Loss function

The loss function, that is, the function used to compare the outputs of the network and their respective targets in the frequency domain, generating the gradients that are used to update the network parameters during the training process, can have an impact on the features learned by the network, for example by shifting the priority from the reduction of the biggest errors to a more uniform treatment of the errors.

To understand those effects, it is convenient to experiment with three different loss functions. Starting from the traditional MSE, one can deemphasize the contribution of

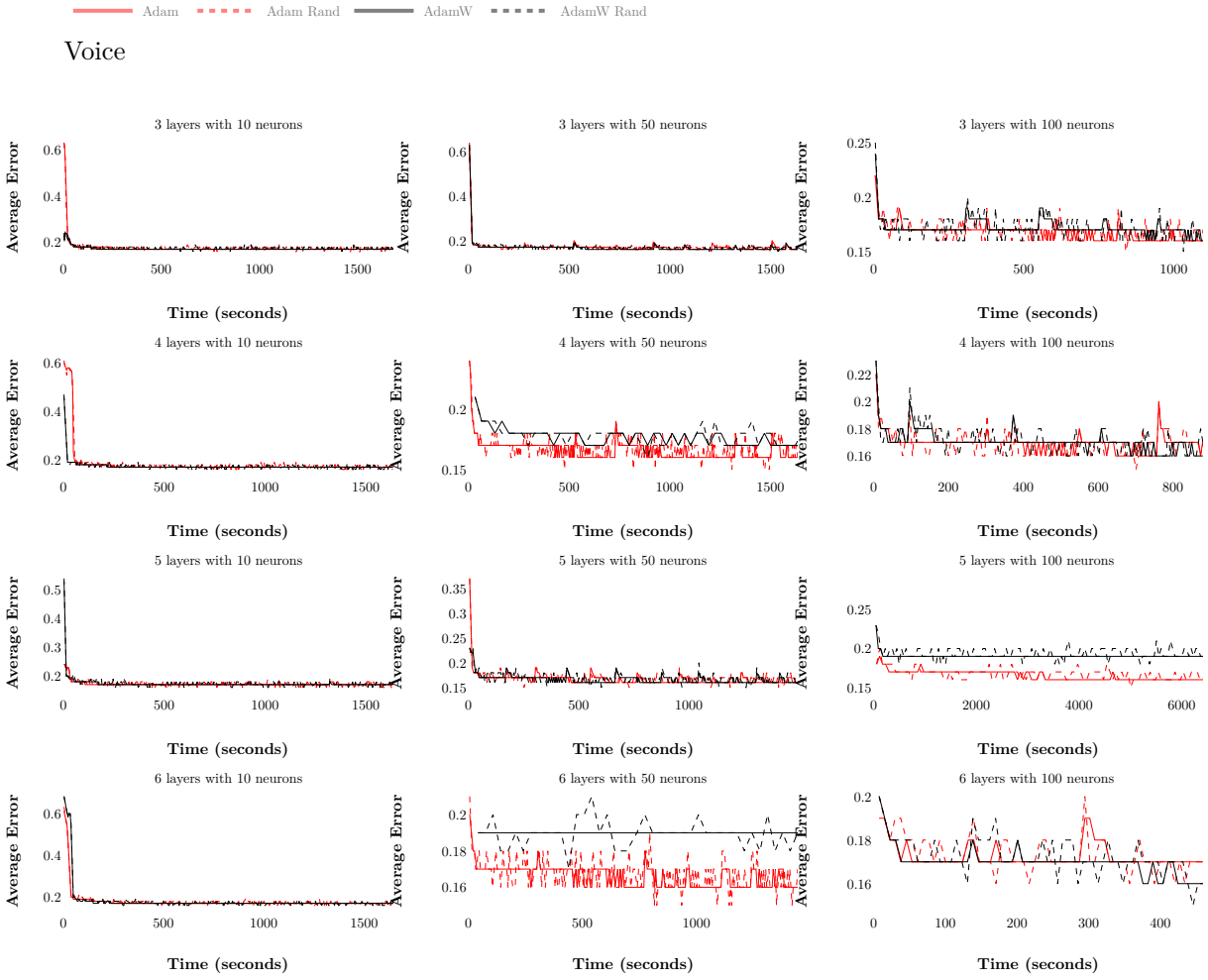

Figure 37: Comparison of the performance of optimizers, using the average deterministic and random error metrics, for the Voice sample library.

errors with relatively high magnitudes or, on the other hand, emphasize it.

To raise the penalty attributed to the biggest errors, one can use the average of the individual errors to the fourth power, instead of squaring the error, as is done by the MSE loss. Conversely, by defining the loss as the average of the absolute values of the individual errors, each error is treated equally, irrespective of its magnitude.

We thus define two additional loss functions, besides the MSE: A4E $= ((a_1 - t_1)^4 + (a_2 - t_2)^4 +$ and AAE $= (|a_1 - t_1| + |a_2 - t_2| + \cdots + |a_t - t_t|)/t$ where $\mathbf{T} = \{a_1, a_2, \cdots, a_t\}$ is the vector representing a specific target of the network, as defined before, and $\mathbf{O} = \{t_1, t_2, \cdots, t_t\}$ a vector representing its corresponding output.

Figure 38 compares the three loss functions for various architectures training on the Voice dataset. As a means to directly compare the behavior of the different loss functions, in the figure, the deterministic and random errors associated with each loss function are shown, and not the losses themselves.

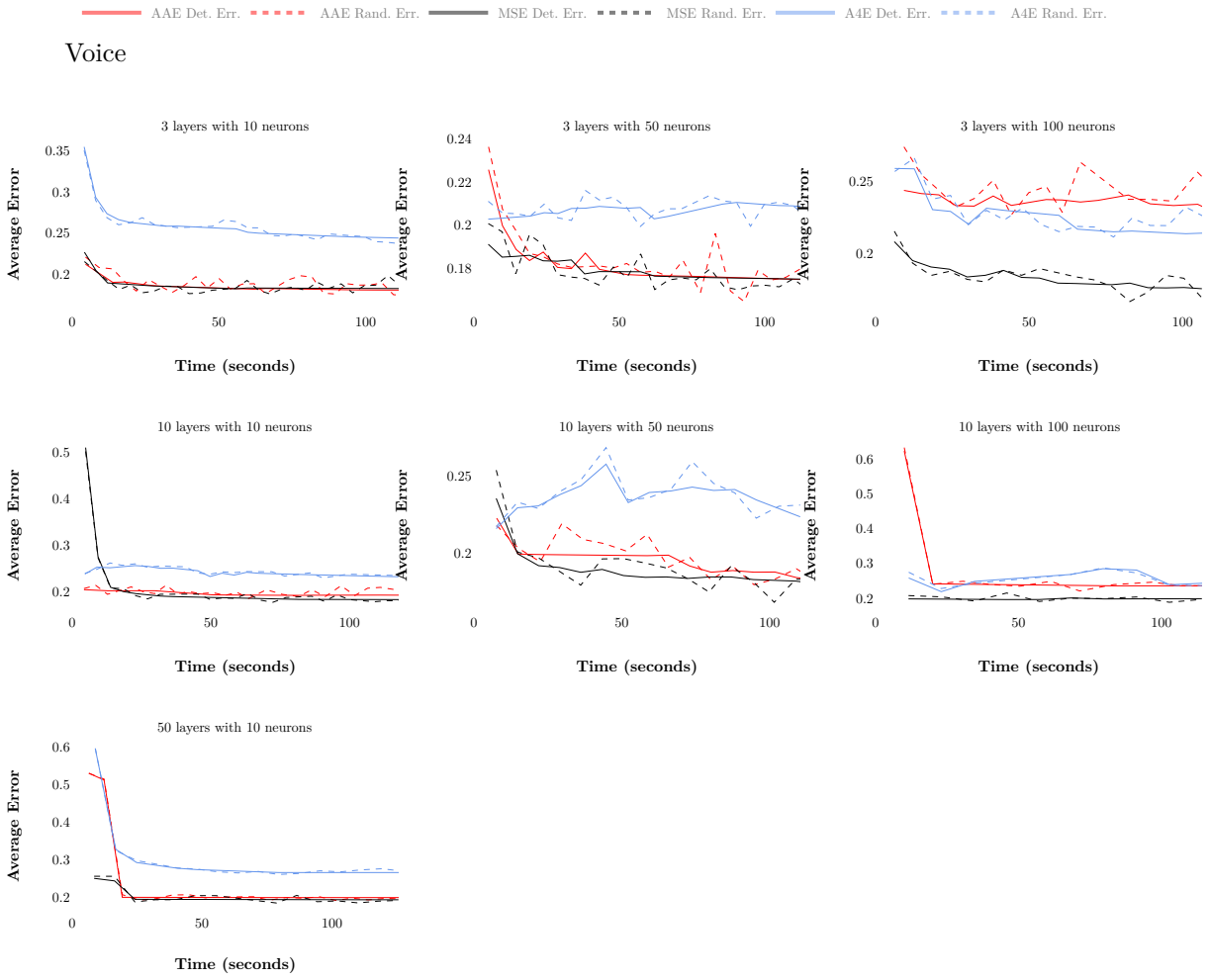

Figure 38: Comparison of the performance of three loss functions, using the average deterministic and random error metrics, for the Voice sample library. Lower is better.

While the errors associated with the A4E loss function exhibit a somewhat stable behavior, this is in general the function with the poorest performance. An equivalence is visible between the MSE and AAE loss functions, with the former consistently presenting better performance. Since the MSE loss is the standard loss function in most models, often implemented via native, more efficient implementations in most platforms, this result is not entirely unexpected, and its use will be preferred for the rest of the present work.

4.3.3 Number of parameters

Having decided on the feedforward architecture for the network, the activation function, the optimizer and the loss function to be used during training, it is now necessary to empirically investigate the parameters that lead to acceptable psychoacoustic results without compromising the efficiency of the plugin, and without causing the network to overfit the data.

For the training of the final networks, it is expected that the necessary number of layers and the number of neurons in each layer will change from instrument to instrument, reflecting the changing number of articulations that are available and need to be learned, the available number of samples to use in the training process and the intrinsic sound characteristics of the particular instrument.

As a consequence, it is necessary to investigate multiple architectures for each sample library, in order to empirically identify the one that yields the best results. For that reason, the training process was designed to enable its execution both in normal machines and in cloud computing environments. While this introduces difficulties in the comparison between networks trained in different hardware, it helps to speed up the exploration of parameters.

It is important to estimate a window, as narrow as possible, inside which the best settings will lie for every sample library. Figures 39, 40 and 41 show the minimum deterministic errors obtained, for each sample library, with a combination of different numbers of layers and neurons. Those images are best appreciated via their interactive counterparts available in the online version of the present work.

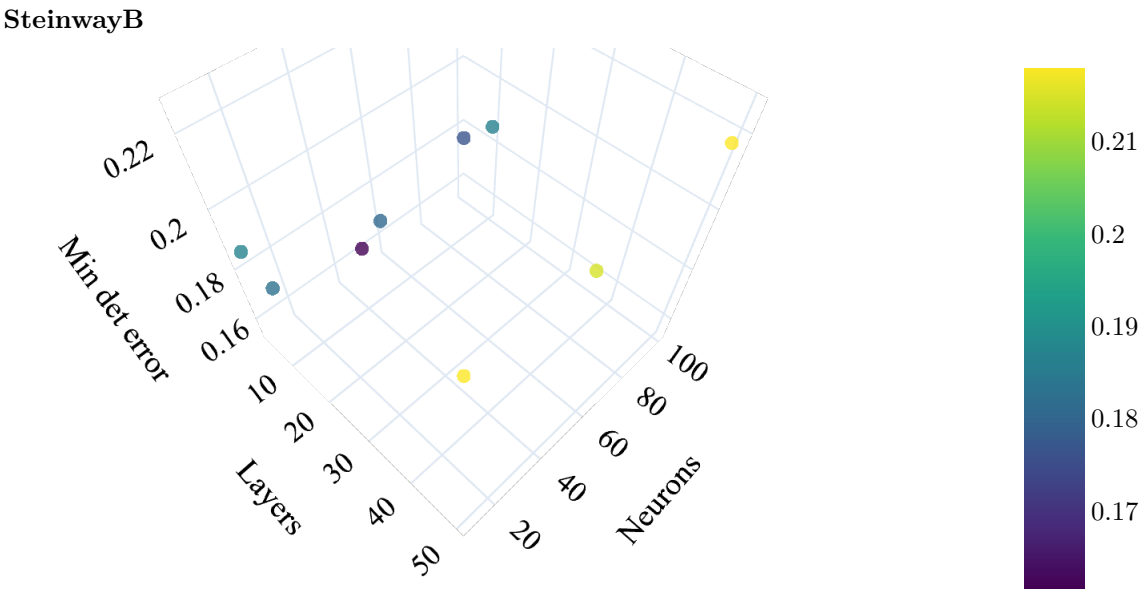

Figure 39: Minimum deterministic error obtained by training a network, with different numbers of layers and neurons, over the SteinwayB library.

For the SteinwayB library, the minimum deterministic error was obtained with an architecture of 3 layers, with 50 neurons each. For the Voice library, 4 layers with 100 neurons each enabled the lowest deterministic error, while the Violin library reached its minimum with an architecture of 5 layers with 200 neurons each. From the figures, it can

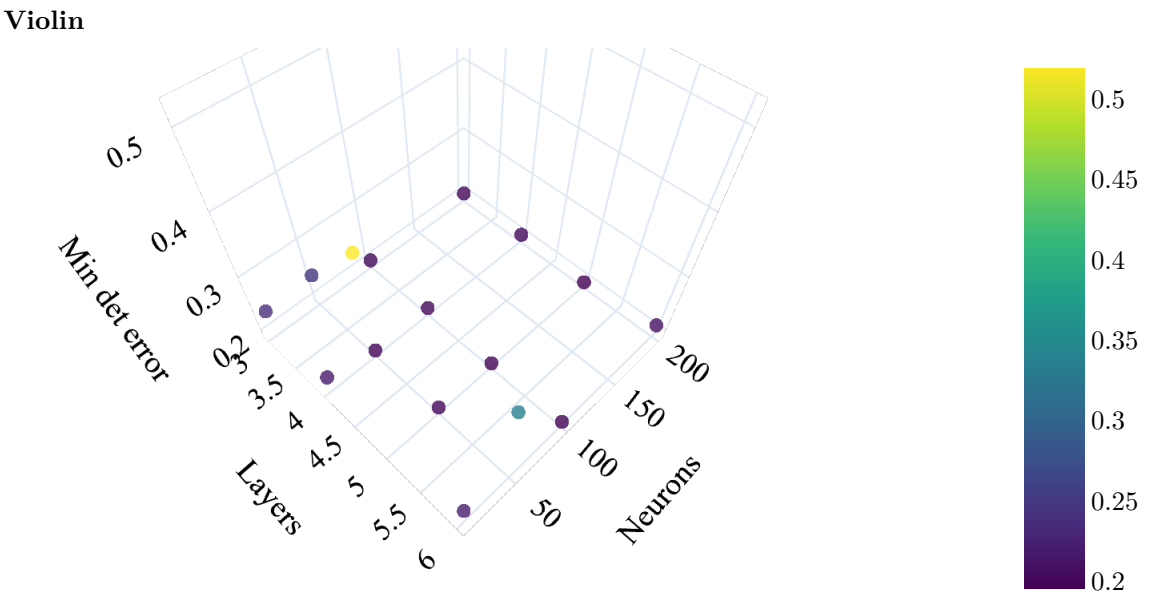

Figure 40: Minimum deterministic error obtained by training a network, with different numbers of layers and neurons, over the Violin library.

be seen that those are comfortable limits in which to train those instruments, with slight variations of those parameters not causing, in general, a great change in the minimum error obtainable.

This information will inform the training step of the instruments, which is also constrained by the necessity of operating in real-time without introducing a noticeable lag to the final implementation.

4.4 THE TRAINED INSTRUMENTS

Having established the most prominent aspects of the whole plugin architecture, the specificities of the core networks encoding the information that enables the emulation of the actual instruments, and the implementation details in the preceding sections, this section describes the training of the final networks. Each sample library presented in Table 2 will give rise to a different instrument, in the form of a trained network.

The last aspect that demands further investigation is the optimal number of frequencies' amplitudes, which was briefly discussed in Section 4.1. Observing the performance of the implementation operating at different numbers of amplitudes on average hardware, this value can be empirically set at 200 amplitudes; above this value, performance degradation starts to be noticeable.

As Figure 42 makes clear, however, considering more amplitudes does not necessarily translate to better results. It can be the case where the network spends resources

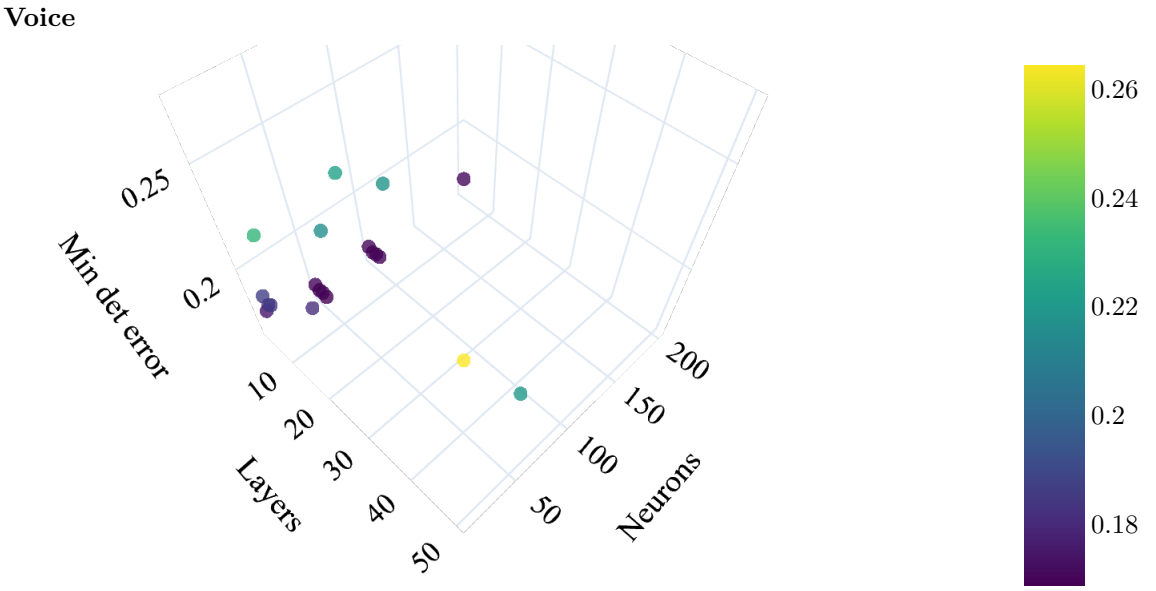

Figure 41: Minimum deterministic error obtained by training a network, with different numbers of layers and neurons, over the Voice library.

trying to learn vanishingly small amplitudes that do not contribute significatively to the overall time-domain quality of the outputs. This can be seen, in Figure 42, in the form of the increased average deterministic error when the number of amplitudes outputted by the network and used in the reconstructing of the sound rises from 50 to 100.

In general, a value around 50 amplitudes seems to yield the best reconstructed sound; all networks in Table 3 reached the best average deterministic error using this number of amplitudes.

Table 3: Name and minimum deterministic error of the final networks used as instrument engines.

Instrument	**Training time (s)**	**Min avg det error**	**Layers**	**Neurons**
Metavox	33491.926679	0.048290	4	50
SteinwayB	13551.295610	0.063610	4	50
Voice	91251.573417	0.068629	4	50
Vox	38397.425609	0.082569	4	70
Violin	5087.474661	0.087303	4	50
SteinwayD	75576.204085	0.115561	4	50
Guitar	3052.396993	0.190615	4	50

It can also be seen from Table 3 that the best architectures are similar: all exhibit 4 layers, and only for the Vox sample library the number of neurons was different from 50 neurons in each layer. Besides not introducing a considerable lag in the plugin implementation, it will be seen that the size on disk of the trained networks, at less than

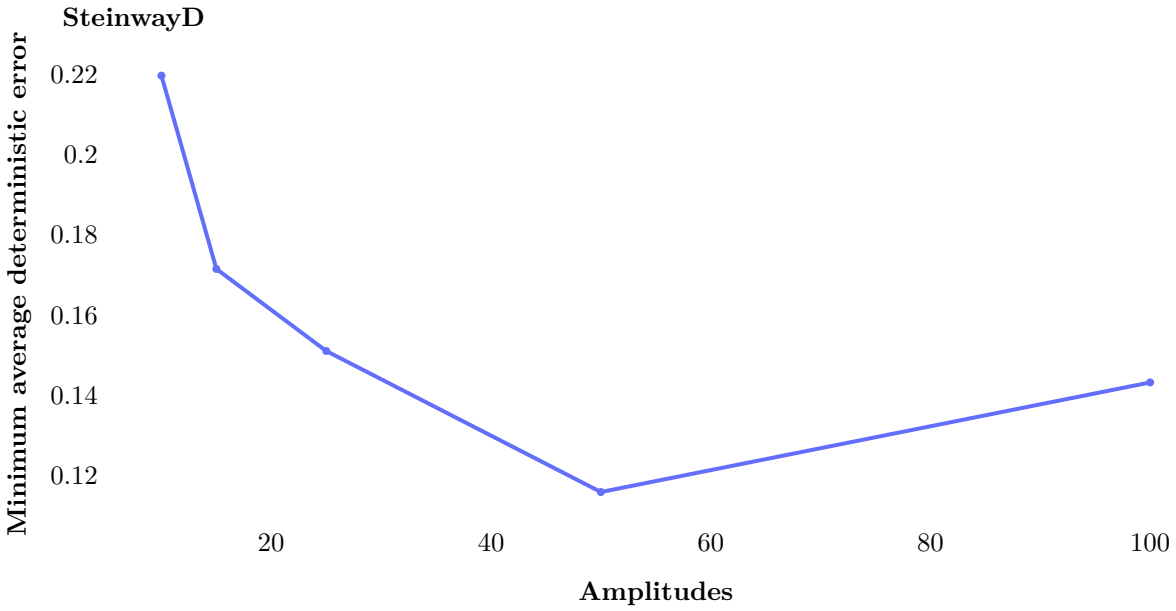

Figure 42: Minimum deterministic error for various numbers of amplitude outputs, for the SteinwayD sample library.

50 KB each, is orders of magnitude smaller than the equivalent state-of-the-art digital instruments available.

5 RESULTS

The first two sections of this chapter analyze the quality of the algorithms introduced in sections 3.3 and 3.4, since they form the foundation of the plugin presented in Section 4.2. While the analysis of the envelope extraction algorithm follows a more theoretical approach, the segmentation algorithm is analyzed from a pragmatic perspective, considering an almost direct application to lossy sound compression, and how the results compare to established codecs.

The section that closes this chapter (Section 5.3) analyses the implementation of the audio plugin, in relation to its quality. To that end, its first subsection explores the errors of the networks and their meaning in the context of the perceived quality of the generated sound, while the second subsection compares the plugin with commercially available digital instruments.

5.1 QUALITY OF THE ENVELOPE

The lack of a precise definition of what an envelope is makes the assessment of the quality of an envelope a difficult task (X. Hu et al., 2012), for which there are no agreed-upon objective metrics. Recalling Figure 10, however, and the accompanying discussion, if one divides, elementwise, the original signal by the extracted envelope, the carrier wave obtained is expected to be bounded between -1 and 1.

Figures 43, 44 and 45 present real-world discrete waves $\mathbf{w}$, their extracted envelope $\mathbf{e}$, and the inferred carrier $\mathbf{c}$.

From Figure 43 it can be seen that the algorithm performs well under the carrier boundedness criteria. This is especially remarkable for the case of spoken voice, a notorious complex wave with fast changes in pitch and high noise content.

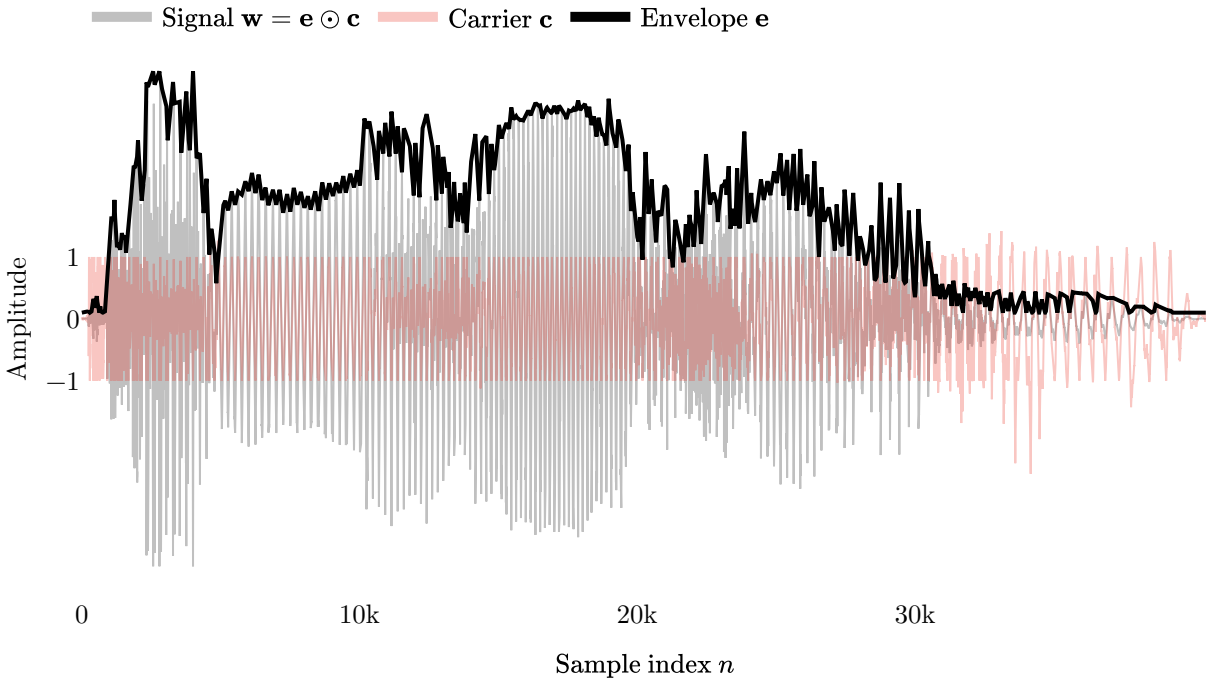

Figure 43: Signal, envelope, and carrier of a male voice uttering the word “amazing”.

Those characteristics are in stark contrast with those of the singing voice in Figure 44, where a recording of an alto singer sustaining a steady note is shown. There, the wave is almost periodic, with a stable waveform and envelope, reflecting the clear and pitched qualities of the underlying sound. The superior frontier of the carrier wave in this case is approximately horizontal.

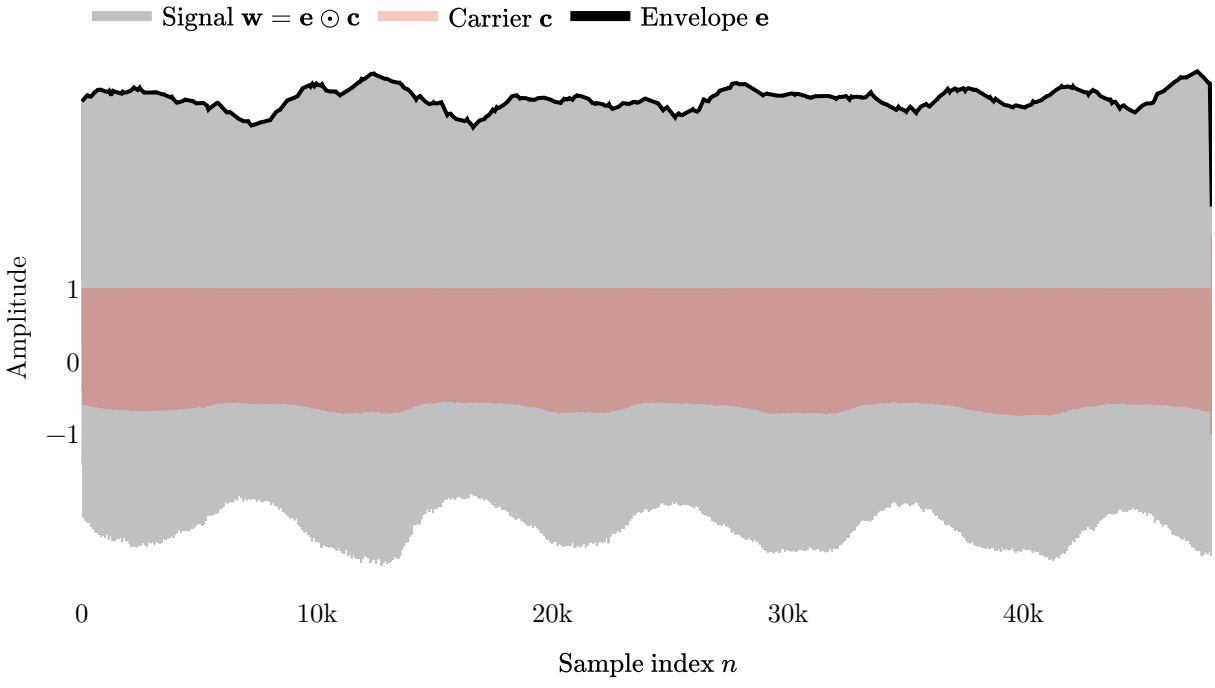

Figure 44: Signal, envelope, and carrier for an alto singer sustaining a steady note.

Figure 45 shows the representation of a bend performed on an electric guitar. The fast frequency variations in the signal did not affect the envelope extracted, nor the boundedness of the carrier wave.

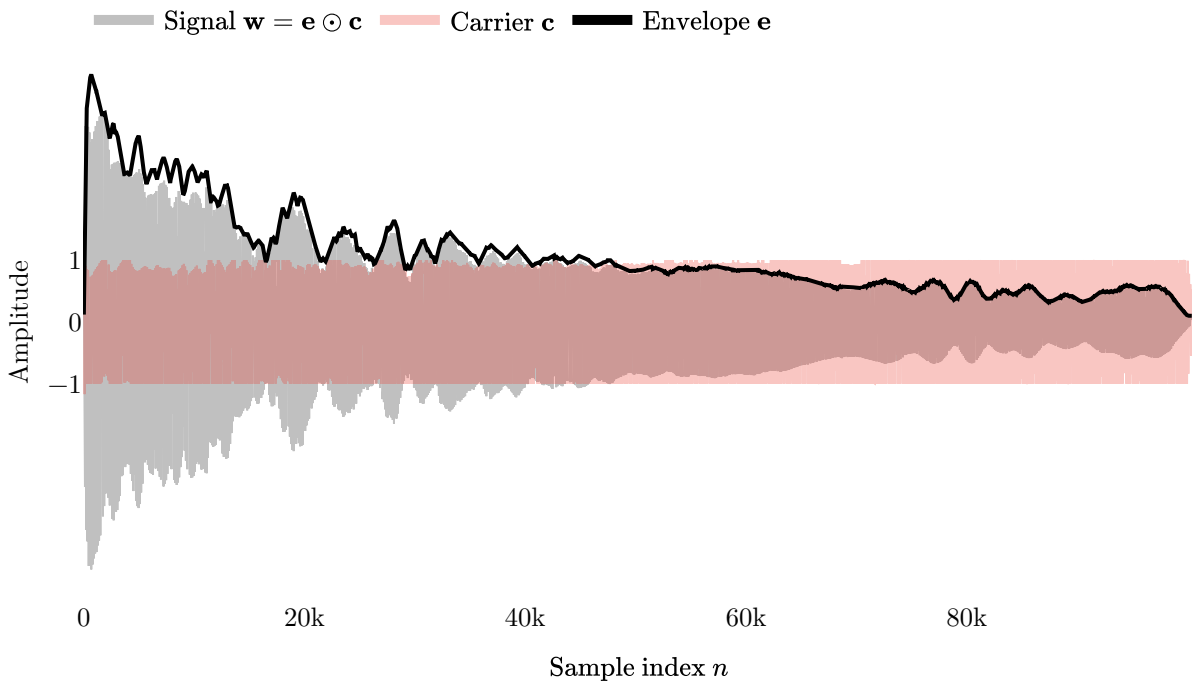

Figure 45: Signal, envelope, and carrier for a bend performed on an electric guitar.

5.1.1 Reference envelope

In Bruce et al. (1992) and Jia et al. (2019), the envelope is defined as the boundary of the region filled by a family of curves. Figure 46 illustrates this concept for a family of sinusoids, modulated by a polynomial, that share the same frequency and amplitude, and whose phases vary from 0 to 2π.

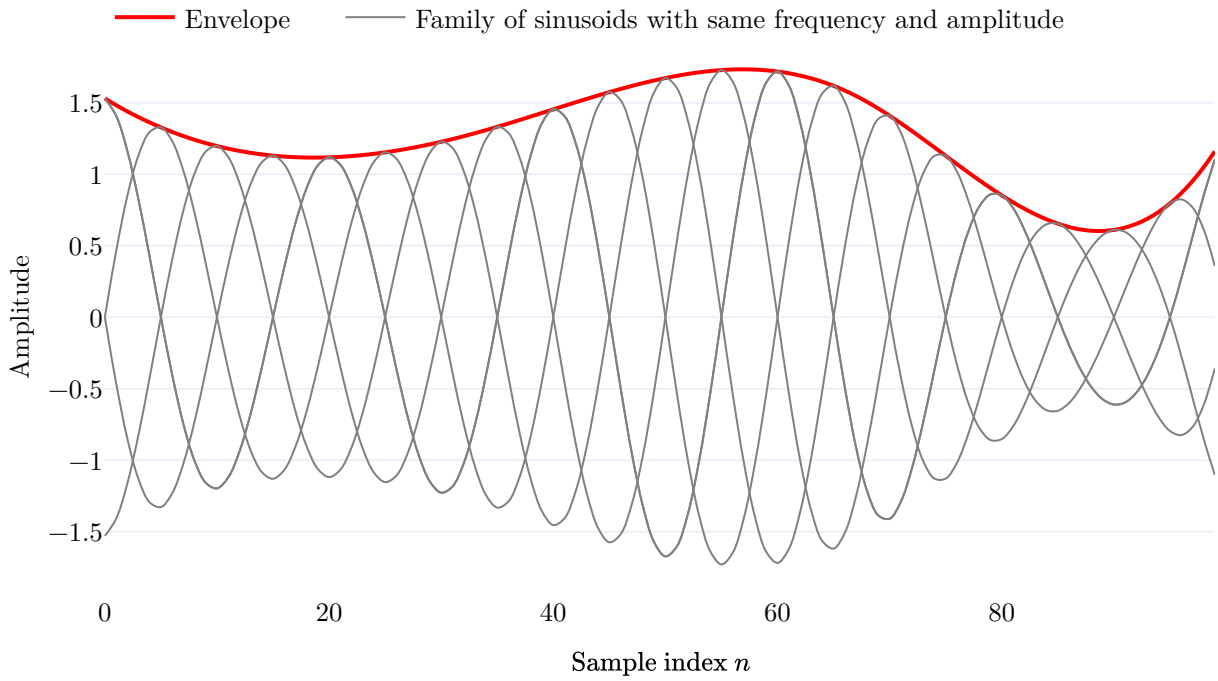

Figure 46: Envelope as the boundary of the region filled by a family of curves, for a family of sinusoids with the same frequency and amplitude, modulated by a polynomial.

In practical applications, one is generally interested in obtaining the envelope of a single known signal, seldom having access to a family of curves. By shifting this signal around its initial position, an approximation to a family of curves can be obtained, however, and can be used to construct a reference envelope.

Given a discrete signal $\mathbf{w}$, the reference envelope $\mathbf{r} = (r_0, r_1, \cdots, r_{N-1})$ is formalized in Equation 26, where k_{pred} is the predominant frequency of $\mathbf{w}$, as obtained by the index of the maximum absolute value of the DFT of the original signal, and T_{max} its predominant period. Moreover, the value of the original signal $\mathbf{w}$ outside the interval $0 \leq n < N$ is assumed to be zero.

$$\begin{aligned} k_{\text{pred}} &= \text{argmax}\left(\mathcal{F}(\mathbf{w})\right) \\ T_{\text{max}} &= N/k_{\text{pred}} \\ r_n &= \max\left(w_{n-\frac{T_{\text{max}}}{2}}, w_{n-\frac{T_{\text{max}}}{2}+1}, w_{n-\frac{T_{\text{max}}}{2}+2}, \cdots, w_{n+\frac{T_{\text{max}}}{2}}\right) \end{aligned} \tag{26}$$

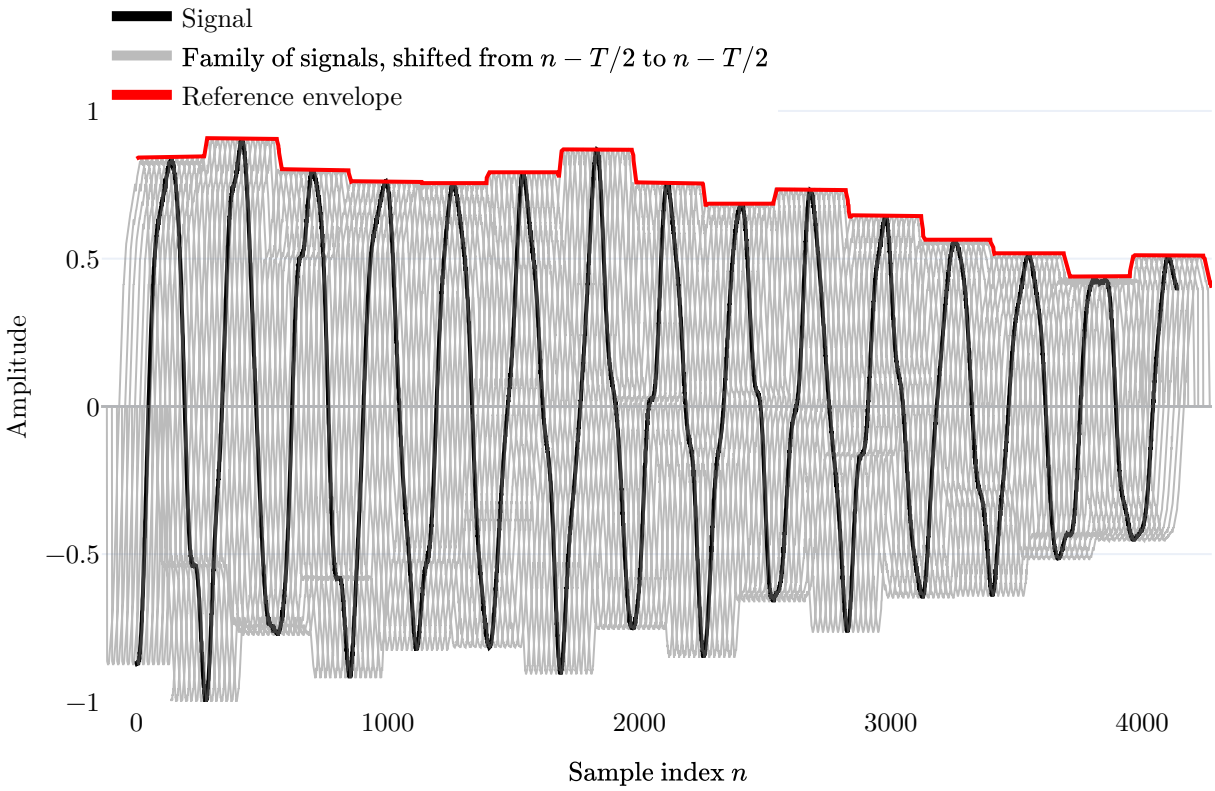

Figure 47: Part of the reference envelope obtained by shifting the original signal horizontally, for the sound of a tom drum.

The envelope obtained is not smooth, being composed of a series of horizontal segments, as can be seen in Figure 47. Those segments, however, agree with the general contour of the wave and overestimate and underestimate the envelope in equal measures.

Although lacking in applicability, both because of its discontinuous shape and the high computational cost involved in its estimation, an envelope constructed in this way arises from a mathematically sound background and can serve as a reference to infer the quality of more practical methods. This envelope is used in the comparisons presented in Table 4, for example.

5.1.2 Comparison with traditional algorithms

Direct comparison with many recent algorithms is made difficult by the unavailability of digital implementations of such works, some of which were designed to process analog signals (e.g. Assef et al., 2018).

Nevertheless, insight can be gained from comparing the method here proposed with some of the most common envelope extraction algorithms. Specifically, we compare the present method with the following approaches:

1. Smoothing - This is a relatively simple procedure that, applied to the absolute values of a discrete signal, can provide an approximation of the variation of its amplitude in time. This work uses the Savitzky-Golay algorithm (Savitzky et al., 1964) with a window width of 3001 samples and a cubic polynomial fitting.

2. Filtering - The absolute value of the wave is filtered using a digital implementation of the Butterworth (Butterworth et al., 1930) low pass filter. This work uses an order 2 filter with a cut-off frequency of 10 Hz.

3. Hilbert Transform - The Hilbert transform is applied to the original signal, filtering the absolute value of the complex result with a Butterworth filter with a cut-off frequency of 1/10 of the fundamental frequency of the original signal.

Figure 48 presents the comparison in the case of a simple sinusoid. The algorithm introduced in this work and the Hilbert transform approach comply with the third condition proposed by Loughlin et al. (1996), which states that the envelope of a periodic wave should be a straight line.

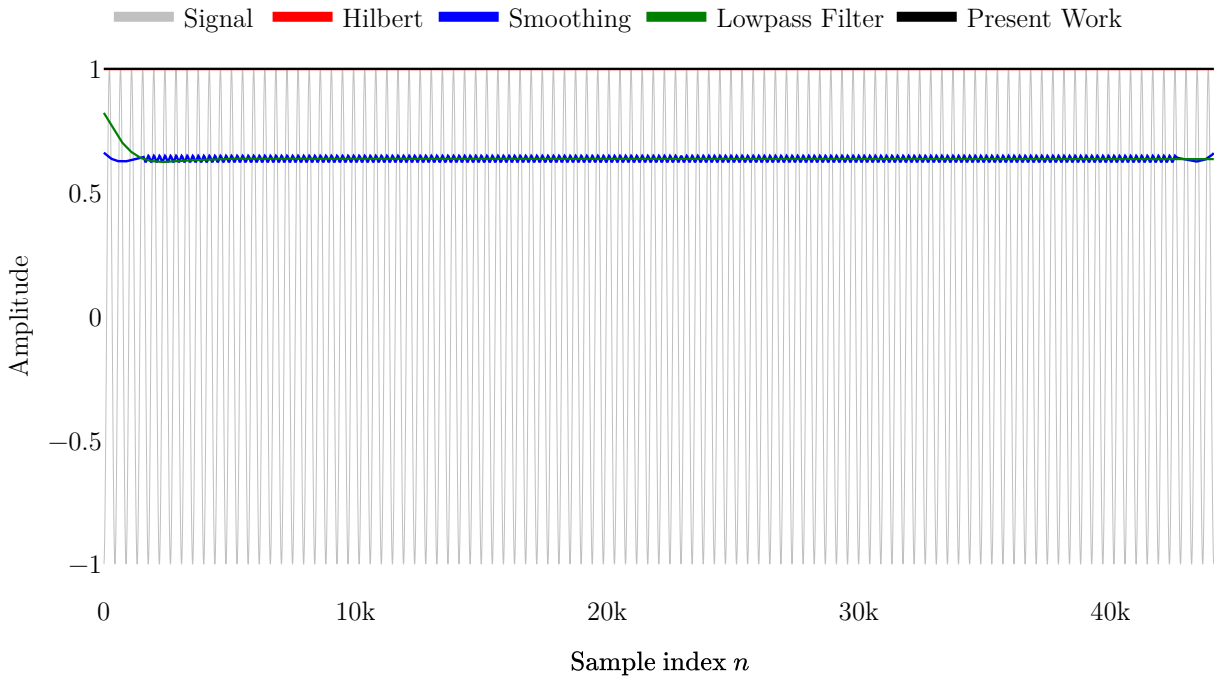

Figure 48: Comparison of envelope detection algorithms for a simple sinusoid. The envelope for the Hilbert transform and for the present work are coincident in the image.

For a recording of the key 33 of a grand piano, where parts of the sound, especially at the beginning, are highly percussive and noisy, all traditional approaches undershoot the original signal, as can be seen in Figure 49.

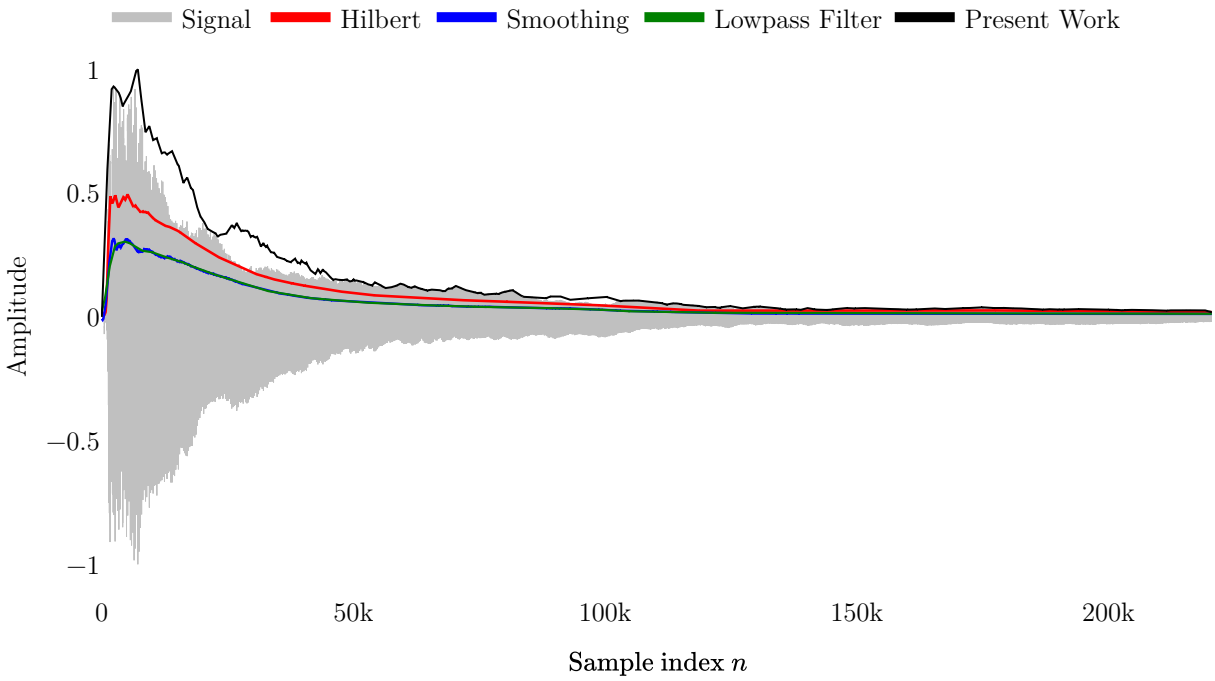

Figure 49: Comparison of envelope detection algorithms for the sound of the key 33 of a grand piano

In Figure 50 the representation of the signal of a soprano singer sustaining a steady note is shown. All envelopes exhibit a similar movement along the samples, but the ones generated by traditional algorithms are consistently undershooting the original wave.

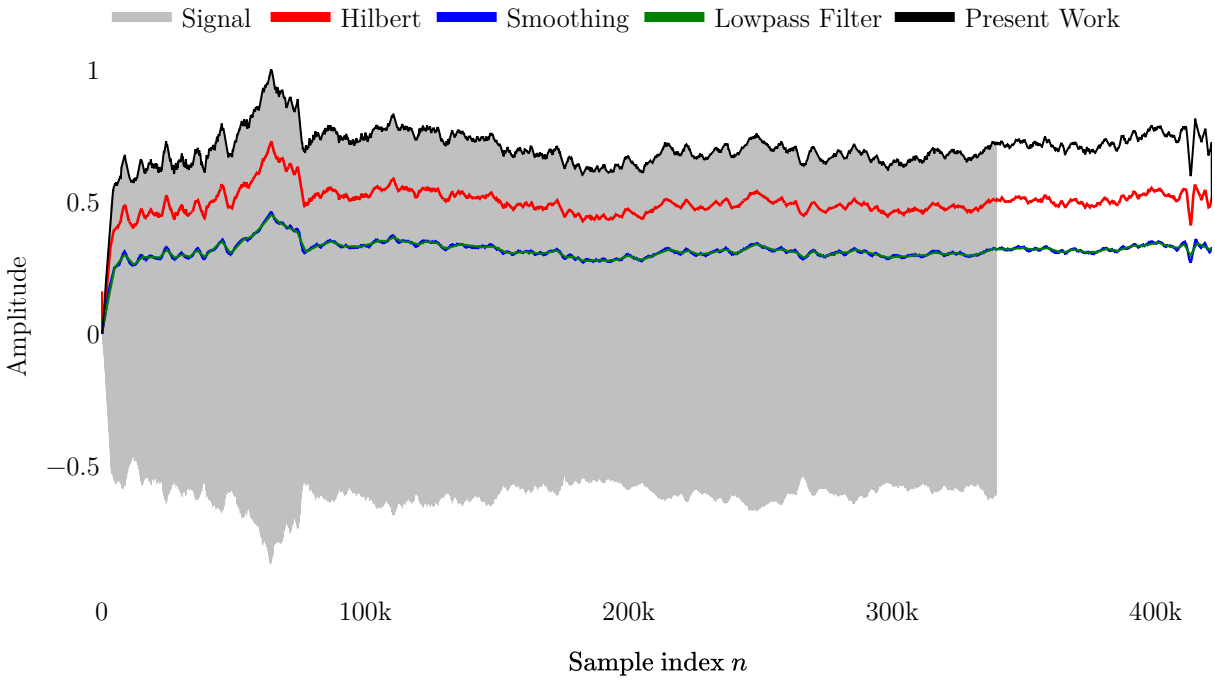

Figure 50: Comparison of envelope detection algorithms for the vocalization of a soprano singer

Besides the comparatively inferior results, the traditional methods demanded a careful choice of parameters. In the case of the Hilbert transform, post-processing in the form of filtering was needed, and an appropriate cut-off frequency had to be chosen. That is in contrast with our method, which organically tunes itself via the automatic choice of the circle's radius representing the average discrete curvature of the signal.

Table 4 presents the average of the absolute errors of the techniques, when compared with the reference envelope introduced at the end of Section 5.1.

Table 4: AAEs of each algorithm in relation to the reference envelope. All signals were previously normalized.

Signal	**Present Work**	**Smoothing**	**Low pass**	**Hilbert**
alto	0.0033	0.5800	0.5836	0.4021
bend	0.0038	0.1268	0.1278	0.0435
brass	0.0109	0.3154	0.3154	0.1866
nonperiodic	0.0000	0.3623	0.3680	0.0002
piano	0.0195	0.0678	0.0680	0.0388
sinusoid	0.0000	0.3636	0.3614	0.0001
soprano	0.0027	0.3911	0.3911	0.2091
spoken_voice	0.0496	0.2793	0.2794	0.1612
tom	0.0073	0.0506	0.0532	0.0228
Average	0.0108	0.2819	0.2831	0.1182

Table 4 shows that the proposed method consistently outperforms the traditional approaches for all waves tested, yielding an error of less than 2%, on average, of the error of the traditional methods.

The processing times of the algorithms, for each signal, are shown in Table 5. The Python source for the tests, as well as the samples used, are available at the repository dedicated to the envelope algorithm (Tarjano, 2020), in the file `Comparison of Envelope Algorithms.py`.

The methods were performed as described at the beginning of this section, using the implementations available at the Scipy library, under the Signal processing module. The proposed algorithm is, on average, the second-fastest.

Table 5: Processing time of the algorithms, in seconds.

Signal	Present Work	Smoothing	Low pass	Hilbert
alto	0.010	0.096	0.007	0.006
bend	0.009	0.146	0.005	0.067
brass	0.015	0.253	0.005	0.035
nonperiodic	0.002	0.064	0.003	0.007
piano	0.022	0.316	0.009	0.030
sinusoid	0.003	0.069	0.003	0.006
soprano	0.111	0.574	0.013	0.293
spoken_voice	0.003	0.062	0.003	0.017
tom	0.002	0.070	0.001	0.004
Average	0.020	0.183	0.005	0.052

5.2 QUALITY OF THE SEGMENTATION

In this section, the quality of the segmentation algorithm presented in Section 3.4 is assessed from various perspectives. Due to the lack of established metrics, a lossy compression application is designed in Section 5.2.1, and its performance analyzed. In Section 3.4.2 a theoretical analysis of the algorithm was performed, in order to establish its theoretical complexity.

The goal of this section is to assess, as objectively as possible, the quality of the segmentation algorithm discussed in Section 3.4, since the absence of similar algorithms and consequently established metrics makes the quality assessment more difficult. Although the algorithm is implemented in the form of a usable executable file and can be employed in the compression of harmonic files with modest durations, it is not meant for general usage in its current form.

Particularly, the lack of provisions for dealing with stochastic residuals and files of arbitrary length must be addressed in order to transform the algorithm into a general-purpose codec. Currently, the compression algorithm uses a single basis waveform, obtained from the average of the pseudo cycles segmented with the segmentation algorithm, as the blueprint to resynthesize the original signal, that being the main cause of its shortcomings.

It is worth noting, however, that in the context of the general framework for emulation of instruments, the main object of the present work, each pseudo cycle in which the original samples are divided is learned by a neural network, that has the potential to learn it as precisely as desired. For this reason, the limitations of the compression codec do not propagate to the general framework, and are not addressed in this work.

5.2.1 Application to Lossy Audio Compression

Sound signals can be broadly classified as harmonic or percussive (Driedger et al., 2014), with harmonicity being related to the periodicity of the signal (Mitrović et al., 2010). In most real-life applications, signals of interest can be seen as a mix of both components, and harmonic/percussive source separation techniques can be used when one component is best considered in isolation (Masuyama et al., 2019).

In the musical domain, where harmonicity plays a fundamental role, highly harmonic signals are more abundant; the sound libraries that serve as the foundation for sample-based musical instruments are an example of an application where such signals are abundant.

Domain-specific compression algorithms and codecs designed to exploit particular characteristics of families of related signals exist in many areas. W. Chen et al. (2016), for example, present a scheme for compressing CNNs, while Cánovas et al. (2014), besides revising existing methods, propose two lossy approaches for the compression of quality score data of genomic sequencing. The use of lossy compression for wireless networks of sensors is studied in Zordan et al. (2014).

More recently, the work of Calhoun et al. (2019) explores the benefits of specific lossy compression algorithms in the context of checkpoints of computational simulations, stored between sessions. New technologies, such as 3D immersive audio, have propelled specific compression algorithms, like the one presented in C. Hu et al. (2021).

Another example is the ACER codec, introduced in Cunningham et al. (2014), which offers an unconventional approach to compression, in that it identifies redundant pieces of a signal that can be indexed in a dictionary. The results seem to be satisfactory, from a perceptual standpoint, when compared with MP3 and Advanced Audio Coding (AAC) codecs (Cunningham et al., 2019). Regarding sample-based digital musical instruments, however, no such specialized algorithms, despite the general high harmonicity often exhibited by those signals, exist.

The adoption of special compressed formats would enable sample-based digital instruments to be implemented in a wider range of hardware, where the storage and processing requirements now make the cost prohibitive, such as electronic drum kits, digital pianos, lightweight hardware such as Raspberry Pi and Arduino and even mobile phones and tablets.

Starting from the discussion in Section 3.4.1, a compressed encoding scheme can

be derived by storing the vectors $\mathbf{t}, \mathbf{w}, \mathbf{a}$. They can be later used to reconstruct the wave, as in Equation 25.

The binary format used to store those vectors uses C++ built-in types, and consists of a header of 4 positive integers, where the number of individual samples N, the signal's original sampling rate, the envelope, and the waveform size are stored. The header is followed by the periods, as an array of unsigned shorts. The waveform amplitudes and lengths, both encoded as arrays of unsigned chars, are also stored.

The `Main.cpp` file at the root of the accompanying repository (Tarjano, 2021) contains the `write_bin` function, responsible for saving the representation to disk in binary format. It will be seen in Section 5.2 that this representation is reasonable for a large number of real-world signals, especially in the case of sample-based digital musical instruments.

5.2.2 Comparison With Traditional Lossy Codecs

In order to investigate how the codec introduced in Section 3.3.2 performs in practice, we implemented, in the C++ language, a compression algorithm based on it; the source code and a compiled executable for machines running Windows OS are available at the GitHub repository prepared for the algorithm (Tarjano, 2021).

We compare the performance of this implementation, dubbed HC, with three of the most common lossy codecs, in the compression of 16 sound samples chosen to represent audio as would be encountered in the libraries of sample-based instruments, while providing enough diversity to test several aspects of the algorithm. They present various degrees of harmonicity and envelope modulations, as well as duration and frequencies, and are divided into 8 voice samples and 8 instrument samples.

Some extreme sounds, less likely to be present in those libraries, were added to test the resilience of the algorithm.

The mezzo-soprano sample, for example, contains a trill at the beginning that evolves to a pronounced vibrato towards the end of the signal. It was chosen to provide feedback on how the algorithm deals with a rapid alternation between pitches.

The trumpet sample, with more than 70 seconds of duration, is also unlikely to be found in sample-based libraries. Being a somewhat stable sound, it would be generated by combining a short initial articulation with a repeating sample. The crash cymbal signal, being fundamentally inharmonic, was added to test the algorithm's performance in a worst-case scenario.

Details about the original samples are shown in Table 6. The MPEG-7 standard defines 17 low-level descriptors for audio files, that summarize how different characteristics of the sound evolve over time (Sikora et al., 2005), one of them being harmonicity. The third column of the table shows a similar measure of harmonicity that addresses, however, the zero-lag peak problem identified by Sikora et al. (2005) in the original MPEG-7 standard implementation, which causes a large value of correlation to be encountered, for all signals, at lags near zero. The full Python implementation of the algorithm can be found at the repository dedicated for this algorithm (Tarjano, 2021), and is directly accessible from the URL `github.com/tesserato/compression/blob/main/harmonicity.py`.

The samples from professional singers were obtained by processing sounds from the Vocalset library(Wilkins et al., 2018). The male singer sample was based on a sample from the Human Voice Dataset (Vocobox, 2015).

The instrument samples were based on samples freely available at the resources page of the Phillarmonia symphony orchestra website (Rouvali, 2021), except the violoncello samples, which were processed from the base samples available at the MIS website (Fritts, 1997).

All original signals are uncompressed mono audio in the WAV format, sampled at a standard sampling rate of 44100 Hz and encoded with a bit rate of 705 kilobits per second (kbit/s), normalized, and with eventual long silences manually removed.

The first format we compare the HC codec to is the popular MP3 format, the third layer of the MPEG-1/2 specification, proposed in 1992 (Britanak et al., 2018) by the Moving Picture Experts Group.

The AAC format was chosen as the second format. In addition to being the formal MP3 successor, developed by a joint effort between industry and academia (Britanak et al., 2018), it is used in services like Apple iTunes and YouTube (Seichter et al., 2016).

Outside the MPEG committee, the open-source Opus format, standardized by the Internet Engineering Task Force (IETF) in 2012, was chosen as the third comparison format. Being considered Vorbis' successor (Gunawan et al., 2018), the codec supports both speech and music, has very low delay (Valin et al., 2013), and is used in real-time audio applications such as WhatsApp (Srivastava et al., 2019).

The main goal of introducing the compression codec is to illustrate a practical application of the features extracted with the segmentation algorithm. Demonstrating that the original signal can be approximately reconstructed from those features establishes the potential of the algorithm in classification tasks such as the one in Muhammad et

al. (2014), where MPEG-7 low-level features are used to separate normal voice from pathological voice.

For this reason, preference was given to compare the HC codec with well-known formats with readily available implementations. It is worth noting, however, that the MPEG-4 Audio Standard presents speech coding tools such as Harmonic Vector Excitation Coding (HVXC) and Harmonic and Individual Lines plus Noise (HILN), aimed at parametrically representing voice and general signals, respectively (Purnhagen et al., 2000), in a manner related to the one presented in this work.

The HC encoding and decoding were done with the implementation available at the segmentation's repository (Tarjano, 2021), while the MP3 encoding and decoding were done with the LAME library (LAME Developers, 2017). All other format conversions were made with the FFmpeg library. The PowerShell scripts used in the process are available in the related repository (Tarjano, 2021); the original waves and the compressed files can also be heard at the repository or, more conveniently, at `harmoniccompression.firebaseapp.com`.

Table 6: Details of the original signals used to test the algorithm. The path is in relation to the root of the repository prepared for the algorithm (Tarjano, 2021).

Path	Name	Harm.	Duration (ms)
01_sopranoA.wav	soprano A	0.987	2597.78
02_sopranoB.wav	soprano B	0.990	2299.34
03_mezzosoprano.wav	mezzo-soprano	0.886	4255.10
04_baritone.wav	baritone	0.896	1838.98
05_countertenorA.wav	countertenor A	0.766	2470.27
06_countertenorB.wav	countertenor B	0.998	2414.65
07_bass.wav	bass singer	0.999	2623.81
08_male.wav	male singer	0.980	1214.06
09_cello.wav	violoncello	0.993	3463.88
10_saxophone.wav	saxophone	0.875	19824.60
11_bassoon.wav	bassoon	0.727	1408.48
12_doublebass.wav	double bass	0.995	1388.46
13_frenchhorn.wav	French horn	0.995	8712.90
14_piano.wav	piano	0.997	4729.37
15_trumpet.wav	trumpet	0.998	70607.98
16_crash.wav	crash cymbal	0.146	1629.64

5.2.2.1 Compression

The compression rate of the representation introduced in Section 3.4 cannot be controlled: the predominant period T is defined by intrinsic characteristics of the original

discrete signal and is, thus, fixed.

Therefore, the encoding configurations for the traditional codecs were chosen in order to generate files with sizes compatible with the ones generated by the proposed compression algorithm.

For the MP3 format, a nominal bit rate of 8 kbit/s and a sampling rate of 8000 Hz were chosen. For the other two formats, no sampling rate reduction was necessary, and the nominal bit rates were 4 kbit/s for the AAC format and 6 kbit/s for the Opus format. Those settings resulted in measured bit rates of 9.680 kbit/s for the HC format, 8.659 kbit/s for the AAC format, 8 kbit/s for the MP3 format, and 7.518 kbit/s for the Opus format.

The MP3 codec is operating in its most extreme compression condition, while further compression could be achieved by reducing the sampling rate of the other two traditional codecs.

It is important to note that those settings are considerably lower than the customary settings used for encoding music signals, and were adopted to generate files with similar sizes in disk. For the MP3 format, for example, a bit rate of at least 128 kbit/s is generally used in normal conditions.

A table with information about individual bit rates for all the samples is available at the repository dedicated to the algorithm (Tarjano, 2021), under the path `github.com/tesserato/compression/blob/main/004_results_HC/bitrates.csv`.

Figure 51 presents, on the left plot, the sum of the compressed sizes of all 16 sound files, for each codec. The right plot shows the average compression rate, calculated as the ratio between the original size and the compressed size, for each codec, as well as the 95% confidence interval around each average; this plot makes particularly clear the equivalence of compression rates between the HC, MP3 and AAC codecs, with OPUS presenting a slightly higher compression rate.

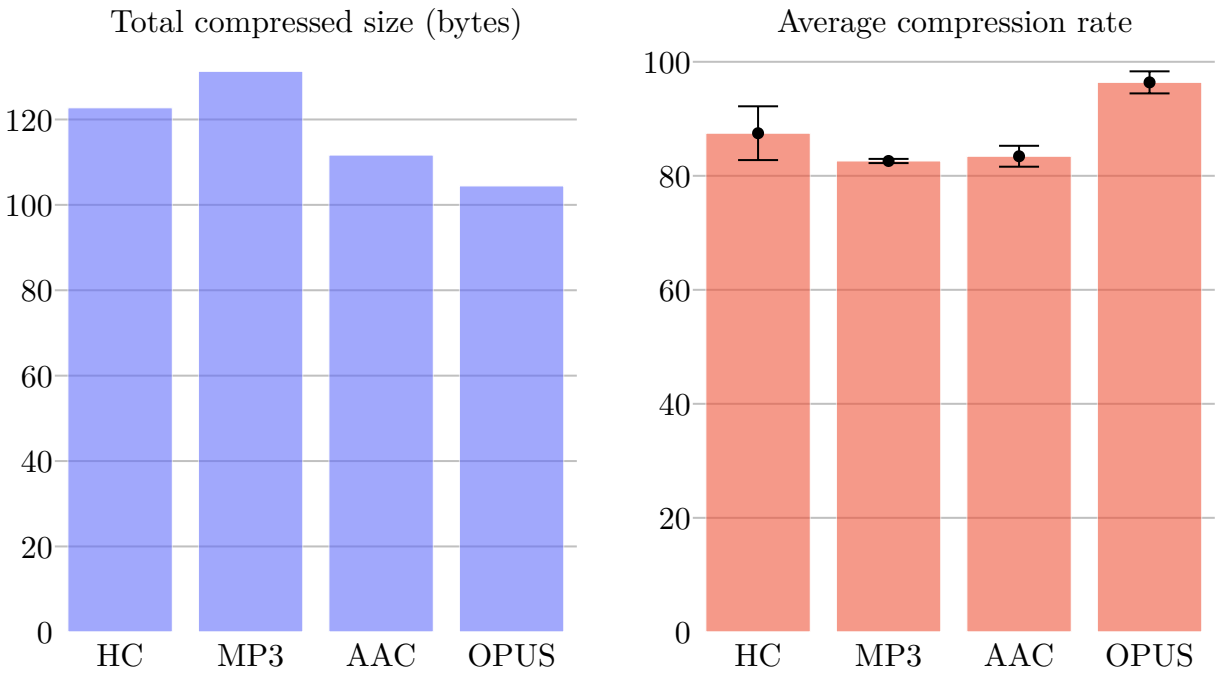

Figure 51: The left plot shows the total size of the 16 compressed samples (lower values are better). The right plot shows the average compression rate and the 95% confidence interval around the average (higher values are better).

5.2.2.2 Timing

The implementation was designed with a focus on maintaining the decoding step as fast as possible, to maximize the real-time performance of the algorithm. In the context of sample-based digital musical instruments, the encoding occurs only once, and generally takes place in more robust hardware than the one used for decoding; it is more important to streamline as much as possible the decoding effort, since, besides enabling the use of more affordable hardware, this task is normally performed a far greater number of times.

For the timing tests, the 16 original samples were encoded and decoded 1000 times with each of the 4 codecs, using the procedure described in Section 5.2. The hardware used was an AMD Ryzen 7 laptop, running Windows 10 operating system.

The average encoding and decoding times, in milliseconds, are presented in Figure 52, alongside a 95% confidence interval around the average. For digital musical instruments, it is important to maintain latency below 10 milliseconds, since that is the limit above which users start to experience performance degradation (Wessel et al., 2002). It can be seen that the proposed algorithm presents the lowest decoding times, at the cost of the highest encoding times; about 3 times the MP3 codec, to approximately 1.5 times the time of the OPUS codec.

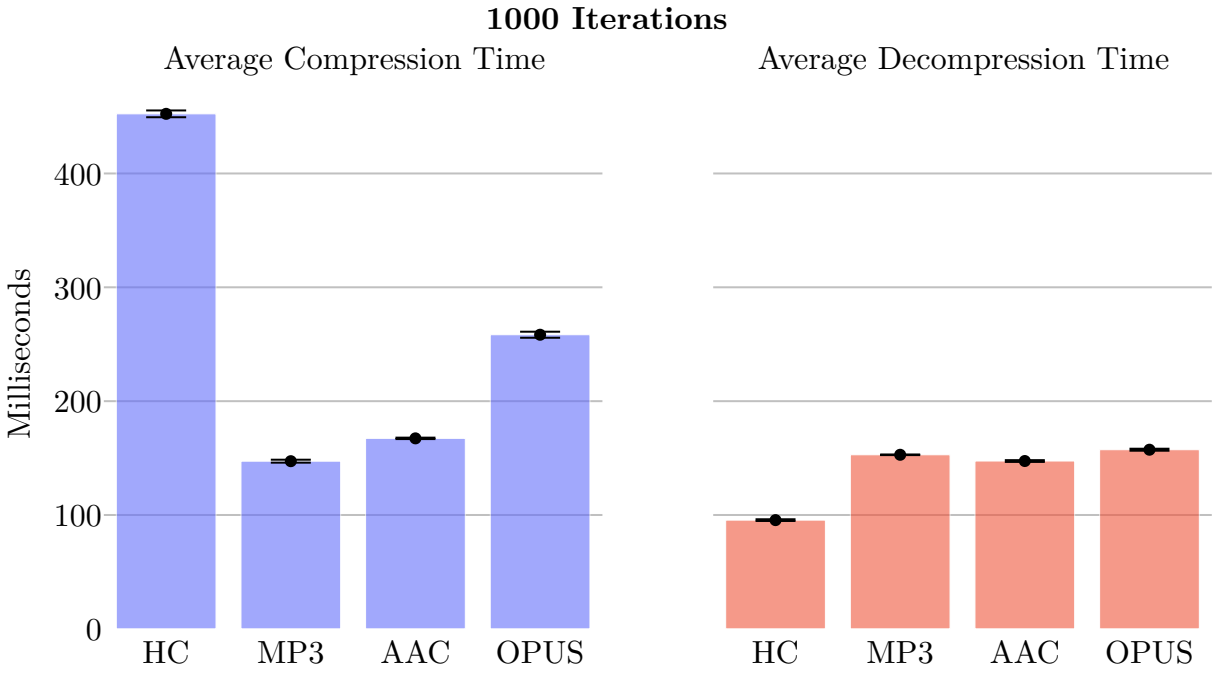

Figure 52: Average encoding (left) and decoding (right) times for the 4 codecs, within a 95% confidence interval. Lower values are better.

5.2.2.3 Quality

For each compression algorithm, the MSE between the compressed and original signal is presented in the left plot of Figure 53, along with a 95% confidence interval around the mean. This metric serves as a proxy for the overall quality of the segmentation algorithm, since all the features extracted — amplitudes, length of pseudo cycles, and the basis waveform — are used in the reconstruction of the original signal and contribute to the MSE.

The right plot in Figure 53 shows the AAE and the 95% confidence interval between the power spectra of the original and the compressed representation, and is important to mitigate eventual shifts in the time-domain position of the compressed digital signal introduced by the compression algorithms.

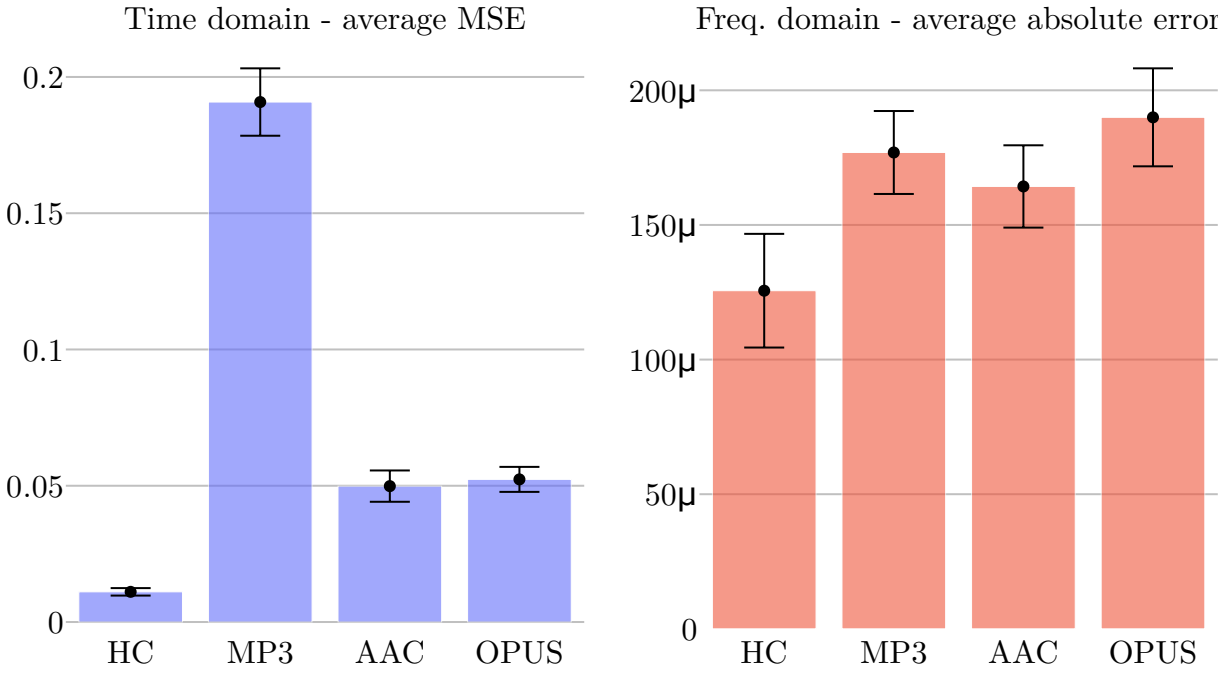

Figure 53: Average Mean Squared Error in the time domain (left) and Average Absolute Error in the frequency (right) for the 4 codecs, within a 95% confidence interval. Lower values are better.

Besides objective metrics shown in Figure 53, directly linked to the quality of the segmentation algorithm, a more extensive investigation of the HC codec can help identify potential applications of the proposed method, especially in signal compression.

Estimating the general quality of a compression algorithm is a complex task, that would ideally rely on subjective listening tests. The International Telecommunication Union's Radiocommunication (ITU-R) offers guidelines for such tests in the form of the Multiple Stimuli with Hidden Reference and Anchor (MUSHRA) methodology (ITU-R, 2014).

Even those tests, however, are not immune to systematic errors, that can contribute to as much as 40% of the attributed score (Zielinski et al., 2008), leading them to be less effective as an absolute measure of general quality (Zielinski, 2016).

As an alternative, libraries were developed to simulate the results of those tests directly from the comparison of the degraded signals against their source. Since the aim of the tests conducted in this work is to investigate the applicability of the segmentation algorithm in a practical setting, the use of simulated listening tests is reasonable.

The Perceptual Evaluation of Speech Quality (PESQ) (Rix et al., 2002; Beerends et al., 2002) and its successor Perceptual Objective Listening Quality Assessment (POLQA) (Beerends et al., 2013) are two of such methodologies, standardized by the ITU-R with

the objective of serving as a consistent alternative to listening tests.

Initially designed inside Google as an alternative to POLQA for Voice over Internet Protocol (VoIP) quality assessment (Hines et al., 2015b), the Virtual Speech Quality Objective Listener (ViSQOL) library rapidly received a general audio mode (Hines et al., 2015a). Since results obtained with this library are comparable with results from the Perceptual Evaluation of Audio Quality (PEAQ) and POLQA libraries (Sloan et al., 2017), and an open-source implementation, now in its third version, is available via a GitHub repository (Chinen et al., 2020), the ViSQOL library is used in this work to further evaluate the compressed signals.

This library comes pre-trained with data from real perceptual tests, and emits a MOS-LQO score ranging from 1 (the worst grade) to 5 (the best grade) when comparing a degraded signal with the original signal from which it was derived.

We divide the tests into two groups, according to the library's modes: voice, for the first 8 signals shown in Table 6, and audio for the last 8. Figure 54 presents the average MOS-LQO score obtained by each group, using Google's ViSQOL library, as well as their 95% confidence interval.

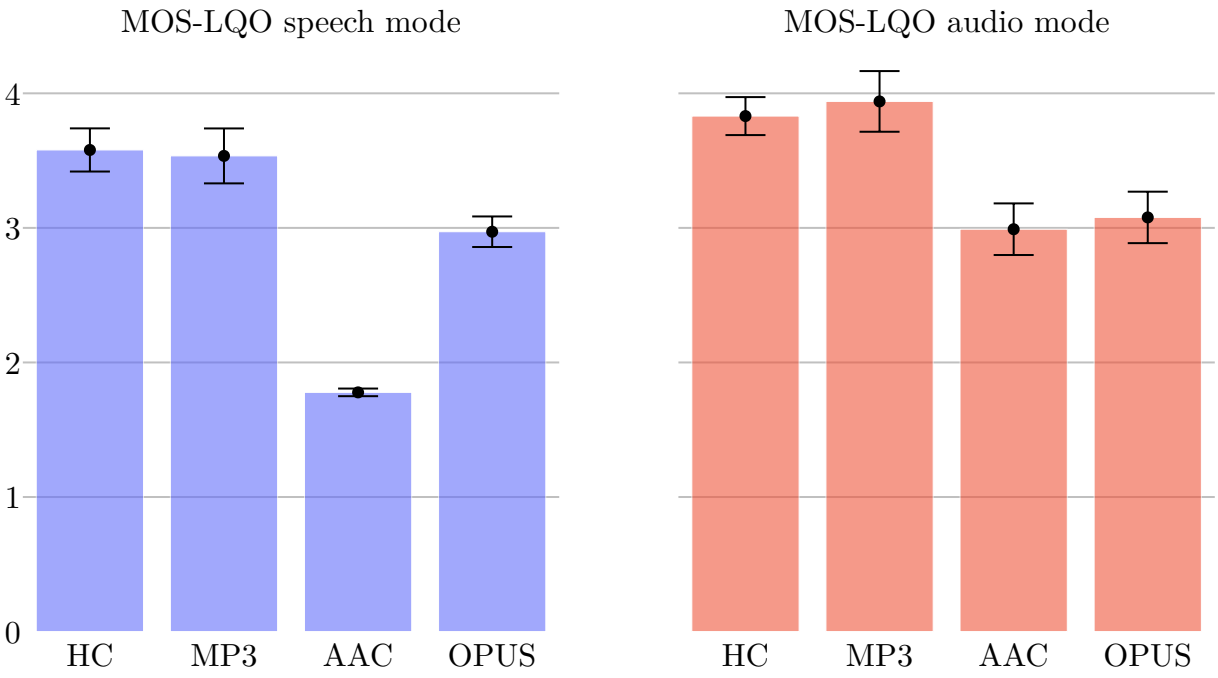

Figure 54: Average MOS-LQO scores for the speech (left) and audio (right) modes, for the 4 codecs, within a 95% confidence interval. Higher values are better.

Listening to the compressed results, as presented in `harmoniccompression.firebaseapp.com`, can provide insight beyond the metrics presented.

For the soprano A and B samples, the HC codec presents results very close to the original signal. MP3 results are also close to the original, albeit less clear than the HC compressed signal.

For the mezzo-soprano signal, the HC codec introduces a synthetic feel to the intense vibrato articulation. This makes the MP3 compression signal more realistic, despite the muffled quality introduced.

The baritone and countertenor A signals, compressed with the HC codec, are indistinguishable from the original. MP3 results for those two signals are very close to the original, but less clear in comparison.

Both the HC and the MP3 compressed signals for the countertenor B and the bass Singer signals are indistinguishable from the original samples. The male singer signals compressed with those two codecs are also very close to the original.

For all the 8 voice signals tested, the AAC and Opus codecs present very low perceptual quality in such an extreme regimen, with the files compressed with the AAC codec guarding only a resemblance with the original. Opus encoded files are generally ridden with clicks and artifacts.

This general pattern, observed above in the case of voice samples, persists in the context of instrument signals. MP3 encoding seems perceptually better than the HC encoding only when the sound is characterized by a percussive part, as is the case of the very beginning of the piano signal. Highly inharmonic signals, however, seem to be detrimental to all algorithms, as can be seen in the poor results of the compression of the crash signal with all codecs.

By listening to the compressed results, one can empirically infer a harmonicity threshold above approximately 0.85 for the HC codec: signals above that value suffer a minimal quality loss during encoding. The result of compressing the crash cymbal signal suggests that, on the other hand, encoding samples with harmonicity below 0.5 renders unintelligible results. Overall, the results suggest the potential of the algorithm in sample-based synthesis and domain-specific lossy compression.

5.3 QUALITY OF THE PLUGIN

As seen in Section 5.2 it is already difficult to evaluate the quality of static sound samples. To objectively assess the quality of digital musical instruments, with all their possible dynamics and intricacies, is even harder. Ideally, one would conduct tests with a great number of users, collecting their opinion about different aspects of the instruments,

and possibly comparing those with other similar instruments.

Since this approach would extrapolate the resources and the time available for the development of the present work, the preceding sections tried to evaluate, as objectively as possible, the most important theoretical components of the current plugin in isolation.

In the first subsection of the current section, however, we analyze the quality of the whole plugin, by examining the behavior of the errors of each network that serves as the engine of the instruments presented.

As a means to complement those observations, the second subsection comments on the characteristics of the plugin using its piano engine, in comparison with other piano implementations found in the market.

The reader can also draw its own conclusions, either by downloading and testing the implementations available at the website dedicated to this work (Tarjano, 2022), or by listening to the available pre-recorded performances available there.

5.3.1 Analysis of the errors of the networks

In Section 4.3, where the training process of the networks is presented, it was explained that, due to computational constraints, the average MSE of two sets of 200 network outputs, one deterministic and one random, were used to keep track of the network performance. Table 3 in Section 4.4 presents the minimum deterministic error for the networks at the core of the trained instruments.

Similarly, in this section, Table 7 presents the average of all MSE, beside other statistical information, for each trained neural network behind the instruments. As can be seen in the column that shows the time, in seconds, of the whole process, this measure is extremely time-intensive to derive in average hardware, rendering it impractical to execute at each epoch during the training process, and motivating the sampling approach described in Section 4.4. The total processing time needed to calculate all the MSE errors for the networks that are shown in the table exceeds 7 hours.

The measures presented in Table 7 are derived from a vector created with all the MSE errors of each network. Each entry in those vectors represents the MSE between one of the network's outputs and its respective target, in the time domain.

For each trained instrument, Table 7 presents the average of this vector, that is, the average of all MSE errors, where a single MSE is obtained from the comparison of an original pseudo cycle and its reconstructed version.

In the following columns, the maximum and minimum MSEs are also presented.

Their variance is shown in the following column, and the time required to calculate all MSEs is shown in the next column. The last column of the table shows the total number of pseudo cycles in which the original library was segmented, corresponding to the total number of MSEs calculations performed.

Table 7: Name and statistics of the totality of errors of the final networks used as instrument engines. After the name of each instrument, the subsequent columns show the average, maximum value, minimum value, and variance of the MSEs. The following column shows the calculation time, in seconds, and the last column shows the number of pseudo cycles.

Name	**Ave**	**Max**	**Min**	**Var**	**Time (s)**	**N**
Metavox	0.0504	0.6714	0.0101	0.0020	2618	125313
SteinwayB	0.0663	0.9167	0.0122	0.0029	2297	111796
Voice	0.0723	0.9655	0.0140	0.0033	2885	177658
Vox	0.0882	0.9659	0.0105	0.0041	5571	302971
Violin	0.0924	0.6566	0.0149	0.0025	1529	84372
SteinwayD	0.1182	0.9799	0.0131	0.0079	10865	562576
Guitar	0.1863	0.8774	0.0140	0.0261	189	9873

From Table 7 some insights can be derived. Considering that the time-domain representation of both the outputs and the targets of the networks are normalized between -1.0 and 1.0 before the MSE calculation is performed, it is easy to see that the maximum value possible for the MSE is 4.0. The maximum errors reported in Table 7, however, never exceed 1.0, and are less than 0.7 in the Vox and Violin cases. The table also shows that the average errors are in general close to 0.1.

The error distribution can be better analyzed in Figure 55, where the frequency of the MSE errors is shown, as well as their 50th percentiles (that are also their medians). From the figure, it is readily noticeable that the average error measure in Table 7 can be misleading, as the majority of errors, for all instruments, are less than 0.1, making the values of the 50th percentiles always lower than the averages.

From Figure 55 it can also be seen that large errors, with values above approximately 0.3 are, in general, very rare.

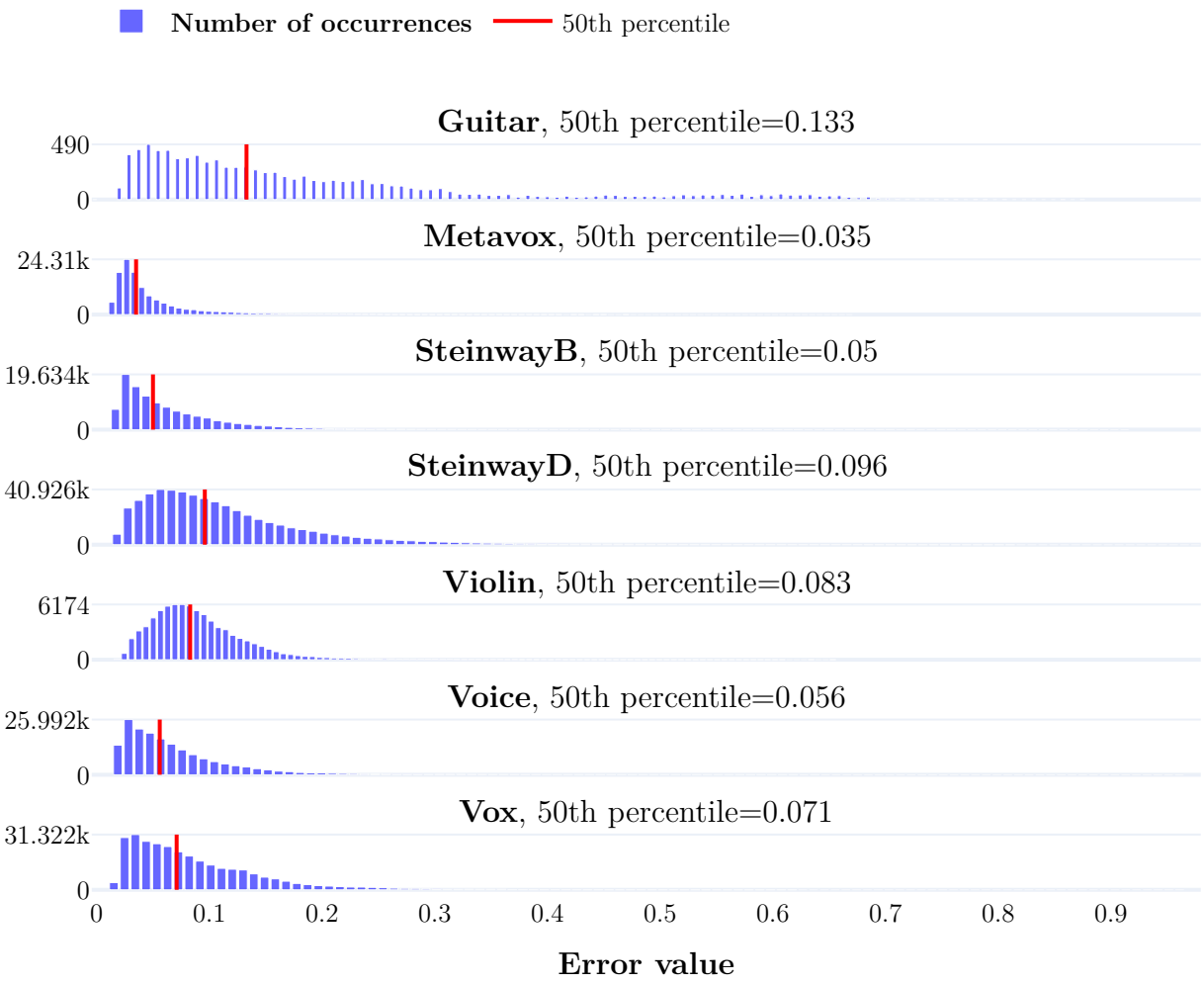

Figure 55: Histogram of the MSE errors of the trained networks, as well as their respective 50th percentiles.

5.3.2 Comparison with available piano plugins

Given their importance in western music, on one hand, and their size and price, on the other, pianos are the preferred candidates for emulation via digital plugins. For that reason, an abundance of plugins, free and commercial, exist. Table 8 compares the piano implementation presented in this work with the most prominent implementations available. With the exception of Pianoteq 7, which uses physical synthesis to generate the piano sounds, and of the Omnes Sonos plugin introduced in this work, all other instruments in Table 8 rely on sample libraries and thus have their size on disk measured in GBs.

The piano digital instrument presented in this work is considerably smaller than the alternatives. Even when compared with Pianoteq 7, the second smaller piano, its library size is approximately 1136 times smaller. Since they are not sample-based, both

Pianoteq 7 and the Omnes Sonos actually support a continuous velocity range, with the 127 layers limitation arising from the MIDI specification. It can be seen that, among the sample-based instruments, the number of velocity layers is limited to a maximum of 100 in the case of the Vienna Imperial implementation, being as low as 12 in the Addictive Keys Studio Grand.

Table 8: Comparison of the most prominent digital instruments that emulate a grand piano. Adapted from Sluis (2022).

Name	**Vel Layers**	**Size**	**Mic Perspectives**
Keyscape (Spectrasonics, 2022)	32	80 GB	1
Garritan CFX (Garritan, 2022)	20	123 GB	3
Vienna Imperial (Vienna Symphonic Library, 2022)	100	46.8 GB	3
Ravenscroft 275 (VI Labs, 2022)	19	5.32 GB	4
Pianoteq 7 (Modartt, 2022)	127	50 MB	5
Ivory II (Synthogy, 2022)	18	77 GB	1
Studio Grand (XLN Audio, 2022)	12	1.5 GB	6
Omnes Sonos (Tarjano, 2022)	127	44 KB	1

6 DISCUSSION

The research presented in this work was developed at the interface of two areas, AI and DSP, that have a very pragmatic component. Generally speaking, progress in DSP is informed by the desire of accomplishing objectives in related fields, such as the creation of new instruments and sounds and the desire to better convey information, for instance.

Similarly, the field of AI and, more specifically, neural networks, was originally motivated by the desire to understand the biological mechanisms of thought and consciousness and, ultimately, implement them in artificial constructs.

To realize this bold endeavor, practitioners borrowed tools from diverse disciplines, the most prominent of which are computer science and mathematics, from the very beginning. This tendency became ingrained, it seems, in the innate approach of researchers and practitioners in the field, and brought about incontestable advancements, marking AI as interdisciplinary since its conception.

Recent progress in the field of neural networks is highlighted by an abundance of impressive results, that have consolidated the approach of using advancements in related disciplines to the betterment of neural networks theory in general, be it in the form of CNNs, better hyperparameter search and parallel implementations, between numerous other examples.

Paradoxically, however, neural networks are often applied to a problem with minimal concern about the representation used, outside the strictly necessary adaptations, such as normalization. It is not uncommon to see works where the hasty application of neural networks culminates in algorithms that could be streamlined if informed by the common practices of the area of interest.

We feel that this tool still offers great potential in its current state of development, a potential that will only be fully realized once it is understood that a more holistic approach is needed, one that considers the problem of interest as a whole.

Furthermore, the resurgence of the interest in AI was largely motivated by hardware and general infrastructure improvements. While it is undeniable that a great theoretical development was also obtained, it is not impossible that the interest will dwindle once not propelled by the continuous advancement of the underlying hardware.

In the near future, therefore, the regimen of paradigm-shifting breakthroughs will likely be replaced by one of incremental progress, where the current inefficiencies will be the primary focus of research, and the foundation for the solutions will necessarily

lie outside the core theoretical framework of AI, in areas of intersection with traditional disciplines.

Real-time sound synthesis is perhaps the field where this effect is currently more present, since the challenges that will permeate this effort are more prominent: thousands of meaningful data points need to be generated on-demand, often in constrained hardware conditions. Algorithms that encapsulate those calculations in higher-level entities, such as pseudo cycles, are one possible approach to mitigate this problem.

In other words, real-time sound synthesis is a complex task, that also draws inspiration from, and demands a reasonable understanding of, a number of ancillary disciplines. Traditionally, it begins with the design of a mathematical model based on a deep understanding of the physics underlying the instrument of interest. Like with neural networks, one of the reasons for its complexity arises from the high dimensionality of the data involved.

The area of AI, therefore, could benefit from tighter integration with more traditional areas, such as real-time sound synthesis. In practice, the general approach presented here in the form of the Omnes Sonos plugin can be used to substitute the sample-based virtual instruments currently representing the state of the art, with the advantage of demanding considerably lower storage requirements.

This result has a deeper, ideological implication: its concreteness can perhaps appeal to the more pragmatic nature of practitioners in both areas; by presenting an algorithm where this multidisciplinary environment is considered from the very beginning, this work hopes to promote the integration and cross-fertilization between all the fields involved.

In other words, real-time sound synthesis is a complex task, that also draws inspiration from, and demands a reasonable understanding of, a number of ancillary disciplines. Traditionally, it begins with the design of a mathematical model based on a deep understanding of the physics underlying the instrument of interest. Like with neural networks, one of the reasons for its complexity arises from the high dimensionality of the data involved.

The area of AI, therefore, could benefit from tighter integration with more traditional areas, such as real-time sound synthesis. In practice, the general approach presented here in the form of the Omnes Sonos plugin can be used to substitute the sample-based virtual instruments currently representing the state of the art, with the advantage of demanding considerably lower storage requirements.

This result has a deeper, ideological implication: its concreteness can perhaps appeal to the more pragmatic nature of practitioners in both areas; by presenting an algorithm where this multidisciplinary environment is considered from the very beginning, this work hopes to promote the integration and cross-fertilization between all the fields involved.

6.1 CONCLUSION

The concrete contribution of this work in the form of the Omnes Sonos plugin, and the multiple libraries made available, representing a gamut of instruments and the singing voice, demonstrate the feasibility of using the proposed framework as the basis for a flexible musical instrument, able to realistically emulate existing real-world instruments and the singing voice, in real time.

This digital instrument provides evidence that virtual instruments generating realistic sounds in real-time do not need to necessarily rely on sample-based approaches, as is the current consensus in the sound production community, and can perhaps motivate a new wave of research in alternative models for sound synthesis.

The novel signal representation — used to encode the samples used to train the neural networks that serve as the instrument's engine — embodies the theoretical advancements that enabled the success of the digital instrument's implementation.

As noted, while developing this new, neural-networks-friendly, signal representation, two deep gaps in classical DSP literature were identified: those two chasms were the lack of an appropriate envelope detection approach for broadband digital signals and the lack of a general signal segmentation methodology.

To solve those problems, this work adopted a multidisciplinary approach, unraveling related theoretical and practical gaps in the field of DSP and other sound-related areas, with inspiration from fields such as computational geometry and harmonic analysis.

As a result, multiple auxiliary algorithms and theories were developed — such as a new envelope detection algorithm, a pseudo cycles focused signal segmentation algorithm, a novel discrete curvature estimation method and a lossy compression codec — with the ultimate aim of introducing a neural-networks-friendly representation of discrete signals, flexible enough to be the basis for a general framework for the real-time emulation of pitched musical instruments and the singing voice.

While attacking the envelope detection problem, as presented in Section 3.3, this work, following the paper in which the approach was first published (Tarjano et al.,

2022b), presents mathematical proofs of important theoretical guarantees of the proposed method.

Albeit not directly indispensable to the representation that is the ultimate object of the present work, this detour was made with the hope of contributing, in the future, to a formal general mathematical definition of an envelope, one that accounts for broadband signals.

Besides the introduction of a new envelope extraction method per se, its formulation in terms of (differential) geometric concepts is, perhaps, as great a contribution as the algorithm itself, in the sense that it encourages experimentations with theories foreign to the traditional DSP viewpoint.

The work also contributes two methods for assessing the quality of an extracted envelope, one intuitive, based on a graph of the original signal with its envelope removed, that enables a quick visual assessment of how much it deviates from a signal with unitary amplitude, and a formal method that provides a numerical measure of the envelope quality.

Those tools can be used to assess the quality of future envelope detection algorithms, guiding upcoming research and helping in the classification of algorithms according partly to the quality of their outputs.

The lack of a general segmentation algorithm was the second subproblem addressed in this work. The solution proposed segments a quasi-periodic signal into its pseudo cycles: meaningful building blocks that correspond to the patterns that repeat themselves in the signal. Philosophically, this segmentation allows a new interpretation of digital signals, as a simpler waveform evolving through time. Besides its importance in the implementation of the Omnes Sound plugin, this work illustrated how this segmentation algorithm can be used in lossy compression.

Although much was done during the elaboration of the present work, time constraints hindered some possible further developments. It is the author's opinion that sound results could be improved, perhaps significantly so, if the samples used to train the different instruments could have received more attention, and could have been manually treated individually. It is needless to mention that recording such samples, in accordance with the necessities of the model, would have been an even more beneficial scenario.

The author also regrets the lack of time to further improve the implementation of the segmentation algorithm, with provisions for a variable average period, for example, and possibly a set of ad hoc checks. The lack of a wider range of freely available samples, with the diversity and quality necessary, also prevented the training of a greater number

of musical instruments.

As for some of the limitations of the final model, it is important to note that the representation exploits redundancy in quasi-periodic signals, and is thus tailored to musical sounds that exhibit a defined dominant frequency. Although those requirements are very liberal, the approach is not designed for the emulation of percussive instruments, such as drums and cymbals.

Digital musical acoustics borrows heavily from the digital signal processing terminology, the latter having its roots in the analog world. Many algorithms are conceived in terms of filters, circuits, and other similar analog constructs, a fact that biases the conceptual, theoretical framework of the area towards specific strategies.

By developing the present work with an open-minded approach, this dissertation suggests paths of communication with various ancillary disciplines, pointing to multiple routes for future developments, and tries to renovate the area, potentially inspiring further research on unorthodox approaches for solving DSP problems.

REFERENCES

AHO, Marko. “Almost like the real thing”: how does the digital simulation of musical instruments influence musicianship? **Music Performance Research**, Royal Northern College of Music, v. 3, p. 22–35, 2009.

AIORDACHIOAIE, Dorel; POPESCU, Theodor Dan. VIBROCHANGE—a development system for condition monitoring based on advanced techniques of signal processing. **The International Journal of Advanced Manufacturing Technology**, Springer Science and Business Media LLC, v. 105, n. 1-4, p. 919–936, Aug. 2019. DOI: 10.1007/s00170-019-04255-3.

ALESSIO, Silvia Maria. **Digital Signal Processing and Spectral Analysis for Scientists**. Switzerland: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-25468-5.

AMIDROR, Isaac. **Mastering The Discrete Fourier Transform In One Two Or Several Dimensions: Pitfalls And Artifacts**. London: Springer London Ltd, 2013. ISBN 9781447151661.

ANDREÃO, Rodrigo Varejão; DORIZZI, Bernadette; BOUDY, Jérôme. ECG signal analysis through hidden Markov models. **IEEE Transactions on Biomedical Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 53, n. 8, p. 1541–1549, Aug. 2006. DOI: 10.1109/tbme.2006.877103.

ASSEF, Amauri Amorin; FERREIRA, Breno Mendes; MAIA, Joaquim Miguel; COSTA, Eduardo Tavares. Modeling and FPGA-based implementation of an efficient and simple envelope detector using a Hilbert Transform FIR filter for ultrasound imaging applications. **Research on Biomedical Engineering**, FapUNIFESP (SciELO), v. 34, n. 1, p. 87–92, Jan. 2018. DOI: 10.1590/2446-4740.02417.

BANK, Balazs; CHABASSIER, Juliette. Model-Based Digital Pianos: From Physics to Sound Synthesis. **IEEE Signal Processing Magazine**, Institute of Electrical and Electronics Engineers (IEEE), v. 36, n. 1, p. 103–114, Jan. 2019. DOI: 10.1109/msp.2018.2872349.

BEERENDS, John G.; HEKSTRA, Andries P.; RIX, Antony W.; HOLLIER, Michael P. Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for End-to-End Speech Quality Assessment Part II: Psychoacoustic Model. **Journal of the Audio Engineering Society**, v. 50, n. 10, p. 765–778, 2002. Available from: <http://www.aes.org/e-lib/browse.cfm?elib=11062>.

BEERENDS, John G.; OBERMANN, Matthias; ULLMANN, Raphael; POMY, Joachim; KEYHL, Michael. Perceptual Objective Listening Quality Assessment (POLQA), The Third Generation ITU-T Standard for End-to-End Speech Quality Measurement Part II–Perceptual Model. **Journal of the Audio Engineering Society**, v. 61, n. 6, p. 18, 2013.

BERG, Mark de; CHEONG, Otfried; KREVELD, Marc van; OVERMARS, Mark. **Computational Geometry: Algorithms and Applications**. Berlin: Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-77974-2.

BIRD, Sarah; DZHULGAKOV, Dmytro. **ONNX V1 released**. 6 Dec. 2017. Available from: <https://research.facebook.com/blog/2017/12/onnx-v1-released/>. Visited on: 10 Apr. 2022.

BIRKHOFF, Garrett. Integration of functions with values in a Banach space. **Transactions of the American Mathematical Society**, American Mathematical Society (AMS), v. 38, n. 2, p. 357–378, 1935. DOI: 10.1090/s0002-9947-1935-1501815-3.

BONADA, J.; SERRA, X.; AMATRIAIN, X.; LOSCOS, A. Spectral Processing. In: DAFX: Digital Audio Effects. New Jersey, United States: John Wiley & Sons, Inc., 2011. chap. 10, p. 393–445. ISBN 9781119991298. DOI: https://doi.org/10.1002/9781119991298.ch10.

BOVERMANN, Till; CAMPO, Alberto de; EGERMANN, Hauke; HARDJOWIROGO, Sarah-Indriyati; WEINZIERL, Stefan (Eds.). **Musical Instruments in the 21st Century**. Singapore: Springer Singapore, 2017. DOI: 10.1007/978-981-10-2951-6.

BOYD, Eric. **Microsoft and Facebook create open ecosystem for AI model interoperability**. 7 Sept. 2017. Available from: <https://azure.microsoft.com/en-us/blog/microsoft-and-facebook-create-open-ecosystem-for-ai-model-interoperability/>. Visited on: 10 Apr. 2022.

BRACEWELL, Ronald Newbold. **The Fourier transform and its applications**. New York, United States: McGraw Hill, 2000. p. 616. ISBN 0073039381.

BRITANAK, Vladimir; RAO, K. R. Audio Coding Standards, (Proprietary) Audio Compression Algorithms, and Broadcasting/Speech/Data Communication Codecs: Overview of Adopted Filter Banks. In: COSINE-/SINE-MODULATED Filter Banks. Cham: Springer International Publishing, 2018. chap. 2, p. 13–37. ISBN 978-3-319-61078-8 978-3-319-61080-1. DOI: 10.1007/978-3-319-61080-1_2. Visited on: 3 Mar. 2021.

BRUCE, J. W.; GIBLIN, P. J. **Curves and Singularities**. Cambridge: Cambridge University Press, Aug. 1992. 340 pp. ISBN 0521429994. DOI: https://doi.org/10.1017/CBO9781139172615.

BRUNEAU, Michel. **Fundamentals of acoustics**. London Newport Beach, CA: ISTE Ltd, 2006. ISBN 9780470612439.

BUTTERWORTH, Stephen et al. On the theory of filter amplifiers. **Wireless Engineer**, v. 7, n. 6, p. 536–541, 1930.

CAETANO, Marcelo; RODET, Xavier. Improved estimation of the amplitude envelope of time-domain signals using true envelope cepstral smoothing. In: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Prague, Czech Republic: IEEE, May 2011. DOI: 10.1109/icassp.2011.5947290.

CALHOUN, Jon; CAPPELLO, Franck; OLSON, Luke N; SNIR, Marc; GROPP, William D. Exploring the feasibility of lossy compression for PDE simulations. **The International Journal of High Performance Computing Applications**, v. 33, n. 2, p. 397–410, Mar. 2019. ISSN 1094-3420, 1741-2846. DOI: 10.1177/1094342018762036. Available from:

<http://journals.sagepub.com/doi/10.1177/1094342018762036>. Visited on: 3 Mar. 2021.

CÁNOVAS, Rodrigo; MOFFAT, Alistair; TURPIN, Andrew. Lossy compression of quality scores in genomic data. **Bioinformatics**, v. 30, n. 15, p. 2130–2136, Aug. 2014. ISSN 1460-2059, 1367-4803. DOI: 10.1093/bioinformatics/btu183. Available from: <https://academic.oup.com/bioinformatics/article-lookup/doi/10.1093/bioinformatics/btu183>. Visited on: 3 Mar. 2021.

CARBONELL, J. G.; SIEKMANN, J. (Eds.). **Music and Artificial Intelligence**. Berlin: Springer Berlin Heidelberg, 28 Aug. 2002. 220 pp. ISBN 354044145X. Available from: <https://www.ebook.de/de/product/7083376/music_and_artificial_intelligence.html>.

CARROLL, Daniel; HANKINS, Eleanor; KOSE, Emek; STERLING, Ivan. A Survey of the Differential Geometry of Discrete Curves. **The Mathematical Intelligencer**, Springer Science and Business Media LLC, v. 36, n. 4, p. 28–35, Oct. 2014. DOI: 10.1007/s00283-014-9472-2.

CECCHERINI-SILBERSTEIN, Tullio; SCARABOTTI, Fabio; TOLLI, Filippo. **Discrete Harmonic Analysis**. Cambridge: Cambridge University Press, June 2018. DOI: 10.1017/9781316856383.

CHEN, Yi-Chen; HUANG, Sung-Feng; LEE, Hung-yi; WANG, Yu-Hsuan; SHEN, Chia-Hao. Audio Word2vec: Sequence-to-Sequence Autoencoding for Unsupervised Learning of Audio Segmentation and Representation. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 27, n. 9, p. 1481–1493, Sept. 2019. DOI: 10.1109/taslp.2019.2922832.

CHEN, Wenlin; WILSON, James; TYREE, Stephen; WEINBERGER, Kilian Q.; CHEN, Yixin. Compressing Convolutional Neural Networks in the Frequency Domain. In: KDD '16: THE 22ND ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. San Francisco California USA: ACM, Aug. 2016. p. 1475–1484. ISBN

978-1-4503-4232-2. DOI: 10.1145/2939672.2939839. Available from: <https://dl.acm.org/doi/10.1145/2939672.2939839>. Visited on: 3 Mar. 2021.

CHINEN, Michael; LIM, Felicia S. C.; SKOGLUND, Jan; GUREEV, Nikita; O'GORMAN, Feargus; HINES, Andrew. ViSQOL v3: An Open Source Production Ready Objective Speech and Audio Metric. **arXiv:2004.09584 [cs, eess]**, Apr. 2020. arXiv: 2004.09584. Available from: <http://arxiv.org/abs/2004.09584>. Visited on: 17 Mar. 2021.

CLELLAND, Jeanne N.; VASSILIOU, Peter J. Strings attached: New light on an old problem. arXiv, 2013. DOI: 10.48550/ARXIV.1302.6672.

COEURJOLLY, David; MIGUET, Serge; TOUGNE, Laure. Discrete Curvature Based on Osculating Circle Estimation. In: LECTURE Notes in Computer Science. Berlin: Springer Berlin Heidelberg, 2001. p. 303–312. DOI: 10.1007/3-540-45129-3_27.

COLLINS, Nick; D'ESCRIVAN, Julio (Eds.). **The Cambridge Companion to Electronic Music**. Cambridge: Cambridge University Press, Dec. 2007. DOI: 10.1017/ccol9780521868617.

COLONNA, Juan Gabriel; CRISTO, Marco; SALVATIERRA, Mario; NAKAMURA, Eduardo Freire. An incremental technique for real-time bioacoustic signal segmentation. **Expert Systems with Applications**, Elsevier BV, v. 42, n. 21, p. 7367–7374, Nov. 2015. DOI: 10.1016/j.eswa.2015.05.030.

CONDON, E. U. Clubs and Allied Activities. **The American Mathematical Monthly**, Informa UK Limited, v. 49, n. 5, p. 330–335, May 1942. DOI: 10.1080/00029890.1942.11991234.

CONKLIN, Harold A. Design and tone in the mechanoacoustic piano. Part I. Piano hammers and tonal effects. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 99, n. 6, p. 3286–3296, June 1996. DOI: 10.1121/1.414947.

CONKLIN, Harold A. Design and tone in the mechanoacoustic piano. Part II. Piano structure. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 100, n. 2, p. 695–708, Aug. 1996. DOI: 10.1121/1.416233.

_____. Design and tone in the mechanoacoustic piano. Part III. Piano strings and scale design. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 100, n. 3, p. 1286–1298, Sept. 1996. DOI: 10.1121/1.416017.

COOLEY, James W. The re-discovery of the fast Fourier transform algorithm. **Mikrochimica Acta**, Springer Science and Business Media LLC, v. 93, n. 1-6, p. 33–45, Jan. 1987. DOI: 10.1007/bf01201681.

COOLEY, James W.; TUKEY, John W. An algorithm for the machine calculation of complex Fourier series. **Mathematics of Computation**, American Mathematical Society (AMS), v. 19, n. 90, p. 297–301, 1965. DOI: 10.1090/s0025-5718-1965-0178586-1.

COULSON, Charles Alfred. **Waves: A mathematical approach to the common types of wave motion**: A Mathematical Approach to the Common Types of Wave Motion (Longman Mathematical Texts). London: Longman Group United Kingdom, 1977. p. 240. ISBN 9780582449541.

CUNNINGHAM, Stuart; GROUT, Vic. Data reduction of audio by exploiting musical repetition. **Multimedia Tools and Applications**, v. 72, n. 3, p. 2299–2320, Oct. 2014. ISSN 1380-7501, 1573-7721. DOI: 10.1007/s11042-013-1504-y. Available from: <http://link.springer.com/10.1007/s11042-013-1504-y>. Visited on: 3 Mar. 2021.

CUNNINGHAM, Stuart; MCGREGOR, Iain. Subjective Evaluation of Music Compressed with the ACER Codec Compared to AAC, MP3, and Uncompressed PCM. **International Journal of Digital Multimedia Broadcasting**, v. 2019, p. 1–16, July 2019. ISSN 1687-7578, 1687-7586. DOI: 10.1155/2019/8265301. Available from: <https://www.hindawi.com/journals/ijdmb/2019/8265301/>. Visited on: 3 Mar. 2021.

D'ALEMBERT, Jean le Rond. **Eléments de musique théorique et pratique suivant les principes de M. Rameau**. Paris: David, 1752. Available from: <https://books.google.com.br/books?id=w1yCDmFqjqYC>.

D'ANGELO, John P. **Hermitian Analysis**. New York: Springer New York, 2013. DOI: 10.1007/978-1-4614-8526-1.

DAU, Torsten. **Speech perception and auditory disorders**. Ballerup: Danavox Jubilee Foundation, 2012. ISBN 9788799001330.

DEMIDOV, S. S. The study of partial differential equations of the first order in the 18th and 19th centuries. **Archive for History of Exact Sciences**, Springer Science and Business Media LLC, v. 26, n. 4, p. 325–350, 1982. DOI: 10.1007/bf00418753.

DESSEIN, Arnaud; CONT, Arshia. An Information-Geometric Approach to Real-Time Audio Segmentation. **IEEE Signal Processing Letters**, Institute of Electrical and Electronics Engineers (IEEE), v. 20, n. 4, p. 331–334, Apr. 2013. DOI: 10.1109/lsp.2013.2247039.

DIDEROT, Denis; ROND D'ALEMBERT, Jean le (Eds.). **Encyclopédie, ou dictionnaire raisonné des sciences, des arts et des métiers, etc.** University of Chicago: ARTFL Encyclopédie Project. Chicago: University of Chicago, 2021. Available from: <https://encyclopedie.uchicago.edu/>.

DINES, L. L. On Convexity. **The American Mathematical Monthly**, Informa UK Limited, v. 45, n. 4, p. 199–209, Apr. 1938. DOI: 10.1080/00029890.1938.11990794.

DIVISION, National Communications System (U.S.). Technology & Standards; SERVICE, United States. General Services Administration. Information Technology. **Telecommunications: Glossary of Telecommunication Terms**. United States: General Services Administration, Information Technology Service, 1996. (Federal standard). Available from: <https://books.google.com.br/books?id=RIrPtQEACAAJ>.

DOLAN, Emily I. **The Orchestral Revolution**. Cambridge: Cambridge University Press, 2013. DOI: 10.1017/cbo9781139235976.

DOMINGUEZ, Alejandro. Highlights in the History of the Fourier Transform. **IEEE Pulse**, Institute of Electrical and Electronics Engineers (IEEE), v. 7, n. 1, p. 53–61, Jan. 2016. DOI: 10.1109/mpul.2015.2498500.

DONAHUE, Chris; MCAULEY, Julian; PUCKETTE, Miller. Adversarial Audio Synthesis. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS. **International Conference on Learning Representations**. New Orleans: ICLR, 2019. Available from: <https://openreview.net/forum?id=ByMVTsR5KQ>.

DONGARRA, J.; SULLIVAN, F. Guest Editors Introduction to the top 10 algorithms. **Computing in Science & Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 2, n. 1, p. 22–23, Jan. 2000. DOI: 10.1109/mcise.2000.814652.

DONOVAN, Tristan. **Replay: the history of video games**. East Sussex, England: Yellow Ant, 2010. ISBN 9780956507228.

DOZAT, Timothy. Incorporating Nesterov Momentum into Adam. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS. **International Conference on Learning Representations**. San Juan, Puerto Rico: ICLR, 2016. Available from: <https://cs229.stanford.edu/proj2015/054_report.pdf>.

DRIEDGER, Jonathan; MULLER, Meinard; EWERT, Sebastian. Improving Time-Scale Modification of Music Signals Using Harmonic-Percussive Separation. **IEEE Signal Processing Letters**, Institute of Electrical and Electronics Engineers (IEEE), v. 21, n. 1, p. 105–109, Jan. 2014. DOI: 10.1109/LSP.2013.2294023.

DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. **J. Mach. Learn. Res.**, JMLR.org, v. 12, null, p. 2121–2159, July 2011. ISSN 1532-4435. Available from: <https://dl.acm.org/doi/10.5555/1953048.2021068>.

EDELSBRUNNER, H.; KIRKPATRICK, D.; SEIDEL, R. On the shape of a set of points in the plane. **IEEE Transactions on Information Theory**, Institute of

Electrical and Electronics Engineers (IEEE), v. 29, n. 4, p. 551–559, July 1983. DOI: 10.1109/tit.1983.1056714.

EDELSBRUNNER, Herbert; MÜCKE, Ernst P. Three-dimensional alpha shapes. **ACM Transactions on Graphics (TOG)**, Association for Computing Machinery (ACM), v. 13, n. 1, p. 43–72, Jan. 1994. DOI: 10.1145/174462.156635.

ENGEL, Jesse; AGRAWAL, Kumar Krishna; CHEN, Shuo; GULRAJANI, Ishaan; DONAHUE, Chris; ROBERTS, Adam. GANSynth: Adversarial Neural Audio Synthesis. In. Available from: <https://openreview.net/pdf?id=H1xQVn09FX>.

ENGEL, Jesse; RESNICK, Cinjon; ROBERTS, Adam; DIELEMAN, Sander; ECK, Douglas; SIMONYAN, Karen; NOROUZI, Mohammad. Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders, 5 Apr. 2017. arXiv: 1704.01279 [cs.LG].

ERSOY, Okan K; HU, Neng-Chung. Fast algorithms for the real discrete Fourier transform. **Department of Electrical and Computer Engineering Technical Reports. Paper 591. Purdue University**, 1988. Available from: <https://docs.lib.purdue.edu/ecetr/591/>.

FAIR. **PyTorch**. 2016. Available from: <https://pytorch.org/>. Visited on: 11 Apr. 2022.

FLEISCHMANN, Oliver; WIETZKE, Lennart; SOMMER, Gerald. A Novel Curvature Estimator for Digital Curves and Images. In: LECTURE Notes in Computer Science. Berlin: Springer Berlin Heidelberg, 2010. p. 442–451. DOI: 10.1007/978-3-642-15986-2_45.

FLETCHER, Neville. **The physics of musical instruments**. New York: Springer, 1998. ISBN 9780387216034.

FOURIER, Jean-Baptiste Joseph. **Théorie analytique de la chaleur**. Paris: Chez Firmin Didot, père et fils, 1822. Available from: <https://books.google.com.br/books?id=TDQJAAAAIAAJ>.

FRANCK, Andreas; VALIMAKI, Vesa. An ideal integrator for higher-order integrated wavetable synthesis. In: 2013 IEEE International Conference on Acoustics, Speech and

Signal Processing. Vancouver, BC, Canada: IEEE, May 2013. DOI: 10.1109/icassp.2013.6637605.

FRENCH, Richard Mark. **Technology of the Guitar**. New York: Springer US, 14 May 2012. 352 pp. ISBN 1461419204. Available from: <https://www.ebook.de/de/product/16501084/richard_mark_french_technology_of_the_guitar.html>.

FRITTS, Lawrence. **The University of Iowa Musical Instrument Samples (MIS)**. 1997. Available from: <http://theremin.music.uiowa.edu/MIS.html>. Visited on: 8 Mar. 2021.

FUJIMURA, O. An approximation to voice aperiodicity. **IEEE Transactions on Audio and Electroacoustics**, Institute of Electrical and Electronics Engineers (IEEE), v. 16, n. 1, p. 68–72, Mar. 1968. DOI: 10.1109/tau.1968.1161951.

GABOR, D. Theory of communication. Part 1: The analysis of information. **Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering**, Institution of Engineering and Technology (IET), v. 93, n. 26, p. 429–441, Nov. 1946. DOI: 10.1049/ji-3-2.1946.0074.

GARRETT, Steven L. **Understanding Acoustics**. Switzerland: Springer Cham, 24 Feb. 2017. 896 pp. ISBN 9783319499789. Available from: <https://www.ebook.de/de/product/33441603/steven_l_garrett_understanding_acoustics.html>.

GARRITAN. **Yamaha CFX Concert Grand Piano**. 16 May 2022. Available from: <https://www.garritan.com/products/cfx-concert-grand-virtual-piano/>. Visited on: 16 May 2022.

GRAHAM-CUMMING, John. Let's build Babbage's Analytical Engine. **New Scientist**, Elsevier BV, v. 208, n. 2791, p. 26–27, Dec. 2010. DOI: 10.1016/s0262-4079(10)63100-4.

GRAVES, Alex. Generating Sequences With Recurrent Neural Networks. arXiv, 2013. DOI: 10.48550/ARXIV.1308.0850.

GRIFFIN, Daniel W.; LIM, Jae S. Multiband excitation vocoder. **IEEE Transactions on Acoustics, Speech, and Signal Processing**, Institute of Electrical and

Electronics Engineers (IEEE), v. 36, n. 8, p. 1223–1235, Aug. 1988. DOI: 10.1109/29.1651.

GUNAWAN, Teddy Surya; KARTIWI, Mira. Performance Evaluation of Multichannel Audio Compression. **Indonesian Journal of Electrical Engineering and Computer Science**, v. 10, n. 1, p. 146, Apr. 2018. ISSN 2502-4760, 2502-4752. DOI: 10.11591/ijeecs.v10.i1.pp146-153. Available from: <http://ijeecs.iaescore.com/index.php/IJEECS/article/view/10871>. Visited on: 3 Mar. 2021.

HAHN, Stefan L. The History of Applications of Analytic Signals in Electrical and Radio Engineering. In: EUROCON 2007 - The International Conference on "Computer as a Tool". Warsaw, Poland: IEEE, 2007. DOI: 10.1109/eurcon.2007.4400463.

HAWLEY, Scott H. Synthesis of Musical Instrument Sounds: Physics-Based Modeling or Machine Learning? **Acoustics Today**, Acoustical Society of America (ASA), v. 16, n. 1, p. 20, 2020. DOI: 10.1121/at.2020.16.1.20.

HE, Lei; DELLWO, Volker. A Praat-Based Algorithm to Extract the Amplitude Envelope and Temporal Fine Structure Using the Hilbert Transform. In: INTERSPEECH 2016. San Francisco: ISCA, Sept. 2016. DOI: 10.21437/interspeech.2016-1447.

HEIDEMAN, Michael T.; JOHNSON, Don H.; BURRUS, C. Sidney. Gauss and the history of the fast Fourier transform. **Archive for History of Exact Sciences**, Springer Science and Business Media LLC, v. 34, n. 3, p. 265–277, 1985. DOI: 10.1007/bf00348431.

HILL, Matthew. George Breed and His Electrified Guitar of 1890. **The Galpin Society Journal**, The Galpin Society, v. 61, p. 193–203, 2008.

HILLER, Lejaren A.; ISAACSON, Leonard M. **Experimental music; composition with an electronic computer**. New York: McGraw-Hill, 1959. ISBN 9780313221583. Available from: <https://archive.org/details/experimentalmusi00hill/page/n5/mode/2up>.

HINES, Andrew; GILLEN, Eoin; KELLY, Damien; SKOGLUND, Jan; KOKARAM, Anil; HARTE, Naomi. ViSQOLAudio: An objective audio quality metric for low bitrate codecs. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 137, n. 6, el449–el455, June 2015. DOI: 10.1121/1.4921674.

HINES, Andrew; SKOGLUND, Jan; KOKARAM, Anil; HARTE, Naomi. ViSQOL: an objective speech quality model. **EURASIP Journal on Audio, Speech, and Music Processing**, 2015 (13), p. 1–18, 2015.

HLAWATSCH, Franz; AUGER, Franois (Eds.). **Time-Frequency Analysis: Concepts and Methods**. New Jersey, United States: ISTE, Jan. 2008. ISBN 9780470611203. DOI: 10.1002/9780470611203.

HU, Chenhao; WANG, Xiaochen; HU, Ruimin; WU, Yulin. Audio object coding based on N-step residual compensating. **Multimedia Tools and Applications**, Feb. 2021. ISSN 1380-7501, 1573-7721. DOI: 10.1007/s11042-020-10339-0. Available from: <http://link.springer.com/10.1007/s11042-020-10339-0>. Visited on: 3 Mar. 2021.

HU, Xiyuan; PENG, Silong; HWANG, Wen-Liang. EMD Revisited: A New Understanding of the Envelope and Resolving the Mode-Mixing Problem in AM-FM Signals. **IEEE Transactions on Signal Processing**, Institute of Electrical and Electronics Engineers (IEEE), v. 60, n. 3, p. 1075–1086, Mar. 2012. DOI: 10.1109/tsp.2011.2179650.

HUBERT, Paulo; PADOVESE, Linilson; STERN, Julio. A Sequential Algorithm for Signal Segmentation. **Entropy**, MDPI AG, v. 20, n. 1, p. 55, Jan. 2018. DOI: 10.3390/e20010055.

HUZAIFAH, Muhammad; WYSE, Lonce. Deep Generative Models for Musical Audio Synthesis. In: HANDBOOK of Artificial Intelligence for Music. Switzerland: Springer International Publishing, 2021. p. 639–678. DOI: 10.1007/978-3-030-72116-9_22.

ITU-R. Method for the subjective assessment of intermediate quality level of audio systems. **International Telecommunication Union Radiocommunication**

Assembly, 2014. Available from:
<https://www.itu.int/rec/R-REC-BS.1534-3-201510-I/en>.

JIA, Linshan; ZHANG, Qing; ZHENG, Xiang; YAO, Pulin; HE, Xiaogao; WEI, Xiaohan. The empirical optimal envelope and its application to local mean decomposition. **Digital Signal Processing**, Elsevier BV, v. 87, p. 166–177, Apr. 2019. DOI: 10.1016/j.dsp.2019.01.024.

JINGZHOU, Sun; YONGBIN, Wang; XIAOSEN, Chen. Audio Segmentation and Classification Approach Based on Adaptive CNN in Broadcast Domain. In: 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS). Beijing, China: IEEE, June 2019. DOI: 10.1109/icis46139.2019.8940257.

KAMMLER, David W. **A First Course in Fourier Analysis**. Cambridge: Cambridge University Press, 2008. p. 864. ISBN 9780521709798.

KELLY, Martin. **Computer : a history of the information machine**. Boulder, Colorado: Westview Press, a member of the Perseus Books Group, 2014. ISBN 9780813345901.

KENMOCHI, Hideki; OHSHITA, Hayato. VOCALOID-commercial singing synthesizer based on sample concatenation. In: PROCEEDINGS of the 8th Annual Conference of the International Speech Communication Association. Japan: Interspeech, 2007. p. 4009–4010.

KIEFER, Chris. Sample-level sound synthesis with recurrent neural networks and conceptors. **PeerJ Computer Science**, PeerJ, v. 5, e205, July 2019. DOI: 10.7717/peerj-cs.205.

KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. arXiv, 2014. DOI: 10.48550/ARXIV.1412.6980.

KOBAYASHI, Hajime; TAGUCHI, Takashi. Virtual idol Hatsune Miku: Case study of new production/consumption phenomena generated by network effects in Japan's online environment. **Markets, Globalization & Development Review**, v. 3, n. 4, 2019.

KRYMOVA, Ekaterina; NAGATHIL, Anil; BELOMESTNY, Denis; MARTIN, Rainer. Segmentation of music signals based on explained variance ratio for applications in spectral complexity reduction. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). New Orleans: IEEE, Mar. 2017. DOI: 10.1109/icassp.2017.7952147.

LAME DEVELOPERS. **The LAME Project**. 2017. Available from: <https://lame.sourceforge.io/index.php>.

LEVINE, Scott N.; SMITH III, Julius Orion. A Sines+Transients+Noise Audio Representation for Data Compression and Time Pitch Scale Modifications. **Audio Engineering Society Convention**, p. 21, 1998.

LIU, Liyuan; JIANG, Haoming; HE, Pengcheng; CHEN, Weizhu; LIU, Xiaodong; GAO, Jianfeng; HAN, Jiawei. On the Variance of the Adaptive Learning Rate and Beyond. arXiv, 2019. DOI: 10.48550/ARXIV.1908.03265.

LOKKI, Tapio; PÄTYNEN, Jukka; TERVO, Sakari; SILTANEN, Samuel; SAVIOJA, Lauri. Engaging concert hall acoustics is made up of temporal envelope preserving reflections. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 129, n. 6, el223–el228, June 2011. DOI: 10.1121/1.3579145.

LOSHCHILOV, Ilya; HUTTER, Frank. Decoupled Weight Decay Regularization. arXiv, 2017. DOI: 10.48550/ARXIV.1711.05101.

LOSTANLEN, Vincent; ANDÉN, Joakim; LAGRANGE, Mathieu. Fourier at the heart of computer music: From harmonic sounds to texture. **Comptes Rendus Physique**, Elsevier BV, v. 20, n. 5, p. 461–473, July 2019. DOI: 10.1016/j.crhy.2019.07.005.

LOUGHLIN, Patrick J.; TACER, Berkant. On the amplitude- and frequency-modulation decomposition of signals. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 100, n. 3, p. 1594–1601, Sept. 1996. DOI: 10.1121/1.416061.

LOY, Gareth. Musicians Make a Standard: The MIDI Phenomenon. **Computer Music Journal**, JSTOR, v. 9, n. 4, p. 8, 1985. DOI: 10.2307/3679619.

LYONS, R. dsp tips & tricks - the sliding DFT. **IEEE Signal Processing Magazine**, Institute of Electrical and Electronics Engineers (IEEE), v. 20, n. 2, p. 74–80, Mar. 2003. DOI: 10.1109/msp.2003.1184347.

LYONS, Richard. Digital Envelope Detection: The Good, the Bad, and the Ugly [Tips and Tricks]. **IEEE Signal Processing Magazine**, Institute of Electrical and Electronics Engineers (IEEE), v. 34, n. 4, p. 183–187, July 2017. DOI: 10.1109/msp.2017.2690438.

MAESTRE, Esteban; RAMIREZ, Rafael; KERSTEN, Stefan; SERRA, Xavier. Expressive Concatenative Synthesis by Reusing Samples from Real Performance Recordings. **Computer Music Journal**, MIT Press - Journals, v. 33, n. 4, p. 23–42, Dec. 2009. DOI: 10.1162/comj.2009.33.4.23.

MAHER, Robert C. Wavetable Synthesis Strategies for Mobile Devices. **Journal of the Audio Engineering Society**, v. 53, n. 3, p. 205–212, Mar. 2005.

MANDEL, J.; BRUN, L. (Eds.). **Mechanical Waves in Solids**. Vienna: Springer Vienna, 1975. DOI: 10.1007/978-3-7091-2728-5.

MASUYAMA, Yoshiki; YATABE, Kohei; OIKAWA, Yasuhiro. Phase-aware Harmonic/percussive Source Separation via Convex Optimization. In: ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Brighton, UK: IEEE, May 2019. DOI: 10.1109/icassp.2019.8683821.

MATHEWS, M. V. The Digital Computer as a Musical Instrument. **Science**, American Association for the Advancement of Science (AAAS), v. 142, n. 3592, p. 553–557, Nov. 1963. DOI: 10.1126/science.142.3592.553.

MATHEWS, Max V. An Acoustic Compiler for Music and Psychological Stimuli. **Bell System Technical Journal**, Institute of Electrical and Electronics Engineers (IEEE), v. 40, n. 3, p. 677–694, May 1961. DOI: 10.1002/j.1538-7305.1961.tb03237.x.

MCAULAY, Robert J; QUATIERI, Thomas F. Sinusoidal transform coding. In: PROCEEDINGS of the Mobile Satellite Conference (Jet Propulsion Lab). Pasadena, California: NASA, 1988.

MEHRI, Soroush; KUMAR, Kundan; GULRAJANI, Ishaan; KUMAR, Rithesh; JAIN, Shubham; SOTELO, Jose; COURVILLE, Aaron; BENGIO, Yoshua. SampleRNN: An Unconditional End-to-End Neural Audio Generation Model, 22 Dec. 2016. arXiv: 1612.07837 [cs.SD].

MILLS, Mara. Media and Prosthesis: The Vocoder, the Artificial Larynx, and the History of Signal Processing. **Qui Parle**, Duke University Press, v. 21, n. 1, p. 107–149, June 2012. DOI: 10.5250/quiparle.21.1.0107.

MIRANDA, Eduardo Reck (Ed.). **Handbook of Artificial Intelligence for Music**. Switzerland: Springer International Publishing, 2021. DOI: 10.1007/978-3-030-72116-9.

_____. **Readings in Music and Artificial Intelligence**. London: Routledge, 2000. DOI: 10.4324/9780203059746.

MIRANDA, Eduardo Reck; WILLIAMS, Duncan. Artificial Intelligence in «Organised Sound». **Organised Sound**, Cambridge University Press (CUP), v. 20, n. 1, p. 76–81, Mar. 2015. DOI: 10.1017/s1355771814000454.

MITROVIĆ, Dalibor; ZEPPELZAUER, Matthias; BREITENEDER, Christian. Features for Content-Based Audio Retrieval. In: ADVANCES in Computers. Amsterdam: Elsevier, 2010. p. 71–150. DOI: 10.1016/s0065-2458(10)78003-7.

MODARTT. **Pianoteq 7**. 16 May 2022. Available from: <https://www.modartt.com/pianoteq>. Visited on: 16 May 2022.

MONTAGU, Jeremy. **Origins and Development of Musical Instruments**. Lanham, Maryland, United States: Scarecrow Press, 29 Oct. 2007. 280 pp. Available from: <https://www.ebook.de/de/product/15249274/jeremy_montagu_origins_and_development_of_musical_instruments.html>.

MOOG, Robert A. Voltage controlled electronic music modules. **journal of the audio engineering society**, v. 13, n. 3, p. 200–206, July 1965.

MOORE, F. Richard. The Dysfunctions of MIDI. **Computer Music Journal**, JSTOR, v. 12, n. 1, p. 19, 1988. DOI: 10.2307/3679834.

MORSE, Philip M.; INGARD, K. Uno. **Theoretical Acoustics**. Princeton, New Jersey, United States: Princeton University Press, 21 Jan. 1987. 952 pp. ISBN 0691024014. Available from: <https://www.ebook.de/de/product/2478529/philip_m_morse_k_uno_ingard_theoretical_acoustics.html>.

MOUKADEM, Ali; DIETERLEN, Alain; HUEBER, Nicolas; BRANDT, Christian. A robust heart sounds segmentation module based on S-transform. **Biomedical Signal Processing and Control**, Elsevier BV, v. 8, n. 3, p. 273–281, May 2013. DOI: 10.1016/j.bspc.2012.11.008.

MOULINES, Eric; CHARPENTIER, Francis. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. **Speech communication**, Elsevier, v. 9, n. 5-6, p. 453–467, 1990.

MUHAMMAD, Ghulam; MELHEM, Moutasem. Pathological voice detection and binary classification using MPEG-7 audio features. **Biomedical Signal Processing and Control**, Elsevier BV, v. 11, p. 1–9, May 2014. DOI: 10.1016/j.bspc.2014.02.001.

NATSIOU, Anastasia; O'LEARY, Sean. Audio representations for deep learning in sound synthesis: A review. In: 2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA). Tangier, Morocco: IEEE, Nov. 2021. DOI: 10.1109/aiccsa53542.2021.9686838.

NEUMANN, J. von. First draft of a report on the EDVAC. **IEEE Annals of the History of Computing**, Institute of Electrical and Electronics Engineers (IEEE), v. 15, n. 4, p. 27–75, 1993. DOI: 10.1109/85.238389.

OLIVEIRA, Agamenon R. E. D'Alembert and the Wave Equation: Its Disputes and Controversies. **Advances in Historical Studies**, Scientific Research Publishing, Inc., v. 09, n. 04, p. 229–239, 2020. DOI: 10.4236/ahs.2020.94019.

OORD, Aaron van den; DIELEMAN, Sander; ZEN, Heiga; SIMONYAN, Karen; VINYALS, Oriol; GRAVES, Alex; KALCHBRENNER, Nal; SENIOR, Andrew; KAVUKCUOGLU, Koray. WaveNet: A Generative Model for Raw Audio, 12 Sept. 2016. arXiv: 1609.03499 [cs.SD].

OORD, Aaron van den; KALCHBRENNER, Nal; VINYALS, Oriol; ESPEHOLT, Lasse; GRAVES, Alex; KAVUKCUOGLU, Koray. Conditional Image Generation with PixelCNN Decoders, 16 June 2016. arXiv: 1606.05328 [cs.CV].

OPPENHEIM, Alan V.; SCHAFER, Ronald W. **Discrete-Time Signal Processing**: Pearson New International Edition. [S.l.]: Pearson Education, Limited, 2013. p. 1060. ISBN 9781292025728.

ORALLO, Carlos Martin; CARUGATI, Ignacio. Single Bin Sliding Discrete Fourier Transform. In: FOURIER Transforms - High-tech Application and Current Trends. London: InTech, Feb. 2017. DOI: 10.5772/66337.

POLYAK, B. T.; JUDITSKY, A. B. Acceleration of Stochastic Approximation by Averaging. **SIAM Journal on Control and Optimization**, Society for Industrial & Applied Mathematics (SIAM), v. 30, n. 4, p. 838–855, July 1992. DOI: 10.1137/0330046.

POPESCU, Theodor D. Signal segmentation using changing regression models with application in seismic engineering. **Digital Signal Processing**, Elsevier BV, v. 24, p. 14–26, Jan. 2014. DOI: 10.1016/j.dsp.2013.09.003.

PURNHAGEN, H.; MEINE, N. HILN-the MPEG-4 parametric audio coding tools. In: 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353). Geneva, Switzerland: Presses Polytech. Univ. Romandes, 2000. DOI: 10.1109/iscas.2000.856031.

QI, Beier; MAO, Yitao; LIU, Jiaxing; LIU, Bo; XU, Li. Relative contributions of acoustic temporal fine structure and envelope cues for lexical tone perception in noise. **The Journal of the Acoustical Society of America**, Acoustical Society of America (ASA), v. 141, n. 5, p. 3022–3029, May 2017. DOI: 10.1121/1.4982247.

RABINER, Lawrence. **Theory and application of digital signal processing**. Englewood Cliffs, N.J: Prentice-Hall, 1975. ISBN 9780139141010.

RAW MATERIAL SOFTWARE LIMITED. **JUCE**. en. Available from: <https://juce.com/>. Visited on: 11 Apr. 2022.

REDDI, Sashank J.; KALE, Satyen; KUMAR, Sanjiv. On the Convergence of Adam and Beyond. arXiv, 2019. DOI: 10.48550/ARXIV.1904.09237.

REDHEFFER, Raymond. A Machine for Playing the Game Nim. **The American Mathematical Monthly**, Informa UK Limited, v. 55, n. 6, p. 343–349, June 1948. DOI: 10.1080/00029890.1948.11999249.

RIEDMILLER, M.; BRAUN, H. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: IEEE International Conference on Neural Networks. San Francisco, CA, USA: IEEE, 1993. DOI: 10.1109/icnn.1993.298623.

RIX, Antony W.; HOLLIER, Michael P.; HEKSTRA, Andries P.; BEERENDS, John G. Perceptual Evaluation of Speech Quality (PESQ) The New ITU Standard for End-to-End Speech Quality Assessment Part I–Time-Delay Compensation. **Journal of the Audio Engineering Society**, v. 50, n. 10, p. 755–764, 2002. Available from: <http://www.aes.org/e-lib/browse.cfm?elib=11063>.

ROUVALI, Santtu-Matias (Ed.). **Philharmonia Sound Samples**. en-GB. Available from: <https://philharmonia.co.uk/resources/sound-samples/>. Visited on: 8 Mar. 2021.

RYBACH, David; GOLLAN, Christian; SCHLUTER, Ralf; NEY, Hermann. Audio segmentation for speech recognition using segment features. In: 2009 IEEE International Conference on Acoustics, Speech and Signal Processing. Taipei, Taiwan: IEEE, Apr. 2009. DOI: 10.1109/icassp.2009.4960554.

SAVITZKY, Abraham.; GOLAY, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. **Analytical Chemistry**, American Chemical Society (ACS), v. 36, n. 8, p. 1627–1639, July 1964. DOI: 10.1021/ac60214a047.

SCHWARZ, Diemo. Current research in concatenative sound synthesis. In: INTERNATIONAL Computer Music Conference (ICMC). Barcelona, Spain: ICMC, 2005. p. 1–1.

SEICHTER, Daniel; CUCCOVILLO, Luca; AICHROTH, Patrick. AAC encoding detection and bitrate estimation using a convolutional neural network. In: 2016 IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING (ICASSP). **2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. Shanghai: IEEE, Mar. 2016. p. 2069–2073. ISBN 978-1-4799-9988-0. DOI: 10.1109/ICASSP.2016.7472041. Available from: <http://ieeexplore.ieee.org/document/7472041/>. Visited on: 3 Mar. 2021.

SERRA, Xavier. **A System for Sound Analysis/Transformation/Synthesis Based on a Deterministic Plus Stochastic Decomposition**. 1989. PhD thesis – Stanford University, Stanford, CA. Available from: <https://ccrma.stanford.edu/files/papers/stanm58.pdf>.

SERRA, Xavier et al. Musical sound modeling with sinusoids plus noise. **Musical signal processing**, Lisse, the Netherlands, p. 91–122, 1997.

SERRA, Xavier; SMITH III, Julius Orion. Spectral Modeling Synthesis: A Sound Analysis/Synthesis System Based on a Deterministic Plus Stochastic Decomposition. **Computer Music Journal**, The MIT Press, v. 14, n. 4, p. 12–24, 1990. ISSN 01489267, 15315169. DOI: https://doi.org/10.2307/3680788. Available from: <http://www.jstor.org/stable/3680788>. Visited on: 27 June 2022.

SHAH, S. **Microsoft and Facebook's open AI ecosystem gains more support**: Intel, IBM, Huawei, and more pledge allegiance to the ONNX framework. 11 Oct. 2017. Available from: <https://www.engadget.com/2017-10-11-microsoft-facebooks-ai-onxx-partners.html>. Visited on: 10 Apr. 2022.

SHANNON, R. V.; ZENG, F.-G.; KAMATH, V.; WYGONSKI, J.; EKELID, M. Speech Recognition with Primarily Temporal Cues. **Science**, American Association for the Advancement of Science (AAAS), v. 270, n. 5234, p. 303–304, Oct. 1995. DOI: 10.1126/science.270.5234.303.

SIKORA, Thomas; MOREAU, Nicolas; KIM, Hyoung-Gook. **Mpeg-7 Audio and Beyond: Audio Content Indexing and Retrieval**. Hoboken, New Jersey: WILEY, Dec. 2005. 304 pp. ISBN 047009334X.

SLOAN, Colm; HARTE, Naomi; KELLY, Damien; KOKARAM, Anil C.; HINES, Andrew. Objective Assessment of Perceptual Audio Quality Using ViSQOLAudio. **IEEE Transactions on Broadcasting**, v. 63, n. 4, p. 693–705, Dec. 2017. ISSN 0018-9316, 1557-9611. DOI: 10.1109/TBC.2017.2704421. Available from: <http://ieeexplore.ieee.org/document/7940042/>. Visited on: 17 Mar. 2021.

SLUIS, Samantha van der. **Best Piano VST Plugins 2022**. 2022. Available from: <https://www.pianodreamers.com/best-piano-vst-plugins/>. Visited on: 16 May 2022.

SMITH, Dave; WOOD, Chet. The «USI», or Universal Synthesizer Interface. **journal of the audio engineering society**, Oct. 1981.

SMITH III, Julius Orion. Viewpoints on the History of Digital Synthesis. In: PROCEEDINGS of the International Computer Music Conference. Montreal: ICMC-91, 1991. Available from: <https://ccrma.stanford.edu/~jos/kna/>.

SPECTRASONICS. **Keyscape**. 16 May 2022. Available from: <https://www.spectrasonics.net/products/keyscape/>. Visited on: 16 May 2022.

SRIVASTAVA, Saurabh Ranjan; DUBE, Sachin; SHRIVASTAYA, Gulshan; SHARMA, Kavita. Smartphone Triggered Security Challenges - Issues, Case Studies and Prevention. In: LE, DacNhuong; KUMAR, Raghvendra; MISHRA, Brojo Kishore; KHARI, Manju; CHATTERJEE, Jyotir Moy (Eds.). **Cyber Security in Parallel and Distributed Computing**. Hoboken, NJ, USA: John Wiley & Sons, Inc., Mar. 2019. p. 187–206. ISBN 978-1-119-48833-0 978-1-119-48805-7. DOI: 10.1002/9781119488330.ch12. Available from: <http://doi.wiley.com/10.1002/9781119488330.ch12>. Visited on: 8 Mar. 2021.

STAFF, Prandoni Paolo. **Signal Processing for Communications**. [S.l.]: Taylor & Francis Group, 2008. p. 300. ISBN 9781420070460.

STEIN, Elias M.; SHAKARCHI, Rami. **Complex Analysis (Princeton Lectures in Analysis)**. Princeton: Princeton University Press, 2003. p. 392. ISBN 9780691113852.

STUBBS, David. **Mars by 1980 : the story of electronic music**. London: Faber and Faber Limited, 2018. ISBN 9780571323975.

SUTSKEVER, Ilya; MARTENS, James; DAHL, George; HINTON, Geoffrey. On the importance of initialization and momentum in deep learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 3. **Proceedings of the 30th International Conference on Machine Learning**. Ed. by Sanjoy Dasgupta and David McAllester. Atlanta, Georgia, USA: PMLR, June 2013. v. 28. (Proceedings of Machine Learning Research, 3). ICML, p. 1139–1147. Available from: <https://proceedings.mlr.press/v28/sutskever13.html>.

SYNTHOGY. **Ivory II Grand Pianos**. 16 May 2022. Available from: <https://synthogy.com/index.php/products/software-products/ivory-2-grand-pianos>. Visited on: 16 May 2022.

TAHIROĞLU, Koray; MAGNUSSON, Thor. Introduction to the special issue on socio-cultural role of technology in digital musical instruments. **Journal of New Music Research**, Informa UK Limited, v. 50, n. 2, p. 117–120, Mar. 2021. DOI: 10.1080/09298215.2021.1907421.

TARJANO, Carlos. **Compression GitHub repository**. Rio de Janeiro: Zenodo, 2021. DOI: 10.5281/zenodo.5048453. Available from: <https://github.com/tesserato/compression>.

_____. **Envelope Repository**. 2020. DOI: 10.5281/zenodo.4719717. Available from: <https://github.com/tesserato/envelope>. Visited on: 3 Dec. 2020.

_____. **Omnes Sonos**. 29 Mar. 2022. Available from: <https://omnessonos.web.app/>. Visited on: 10 Apr. 2022.

_____. **Redes neurais aplicadas à modelagem de instrumentos acústicos para síntese sonora em tempo real**. 2018. MA thesis – Universidade Federal

Fluminense. DOI: 10.22409/TPP.2018.m.11839251743. Available from: <https://app.uff.br/riuff/handle/1/7613>.

TARJANO, Carlos; PEREIRA, Valdecy. An Efficient Algorithm For Segmenting Quasi-Periodic Digital Signals Into Pseudo Cycles: Application in Lossy Audio Compression. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, Institute of Electrical and Electronics Engineers (IEEE), p. 1–1, 2022. DOI: 10.1109/taslp.2022.3171969.

_____. Envelope estimation using geometric properties of a discrete real signal. **Digital Signal Processing**, Elsevier BV, v. 120, p. 103229, Jan. 2022. DOI: 10.1016/j.dsp.2021.103229.

_____. Neuro-Spectral Audio Synthesis: Exploiting Characteristics of the Discrete Fourier Transform in the Real-Time Simulation of Musical Instruments Using Parallel Neural Networks. In: ARTIFICIAL Neural Networks and Machine Learning – ICANN 2019: Text and Time Series. Switzerland: Springer International Publishing, 2019. p. 362–375. DOI: 10.1007/978-3-030-30490-4_30.

THE LINUX FOUNDATION. **Open Neural Network Exchange,The open standard for machine learning interoperability**. 2019. Available from: <https://onnx.ai/>. Visited on: 10 Apr. 2022.

THE MIDI ASSOCIATION. **The History Of MIDI**. 9 Apr. 2022. Available from: <https://www.midi.org/midi-articles/the-history-of-midi>.

TRUEMAN, Dan; MULSHINE, Mike; WANG, Matt. **bitKlavier**. 2021. Available from: <https://bitklavier.com/#aboutBody>.

TURING, A. M. On Computable Numbers, with an Application to the Entscheidungsproblem. **Proceedings of the London Mathematical Society**, Wiley, s2-42, n. 1, p. 230–265, 1937. DOI: 10.1112/plms/s2-42.1.230.

TURNER, Richard E.; SAHANI, Maneesh. Demodulation as Probabilistic Inference. **IEEE Transactions on Audio, Speech, and Language Processing**, Institute of

Electrical and Electronics Engineers (IEEE), v. 19, n. 8, p. 2398–2411, Nov. 2011. DOI: 10.1109/tasl.2011.2135852.

VALBRET, H.; MOULINES, E.; TUBACH, J.P. Voice transformation using PSOLA technique. **Speech Communication**, Elsevier BV, v. 11, n. 2-3, p. 175–187, June 1992. DOI: 10.1016/0167-6393(92)90012-v.

VALIN, Jean-Marc; MAXWELL, Gregory; TERRIBERRY, Timothy B.; VOS, Koen. high-quality, low-delay music coding in the opus codec. **journal of the audio engineering society**, Oct. 2013. Available from: <https://www.aes.org/e-lib/browse.cfm?elib=16992>.

VARYTIMIDIS, Christos; RAPANTZIKOS, Konstantinos; AVRITHIS, Yannis; KOLLIAS, Stefanos. Alpha-shapes for local feature detection. **Pattern Recognition**, Elsevier BV, v. 50, p. 56–73, Feb. 2016. DOI: 10.1016/j.patcog.2015.08.016.

VÁŠA, L.; VANĚČEK, P.; PRANTL, M.; SKORKOVSKÁ, V.; MARTINEK, P.; KOLINGEROVÁ, I. Mesh Statistics for Robust Curvature Estimation. **Computer Graphics Forum**, Wiley, v. 35, n. 5, p. 271–280, Aug. 2016. DOI: 10.1111/cgf.12982.

VÁŠA, Libor; KÜHNERT, Tom; BRUNNETT, Guido. Multivariate analysis of curvature estimators. **Computer-Aided Design and Applications**, CAD, v. 14, n. 1, p. 58–69, June 2016. DOI: 10.1080/16864360.2016.1199756.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. arXiv, 2017. DOI: 10.48550/ARXIV.1706.03762.

VI LABS. **Ravenscroft 275**. 16 May 2022. Available from: <https://www.vilabsaudio.com/Ravenscroft-By-VI-Labs>. Visited on: 16 May 2022.

VIENNA SYMPHONIC LIBRARY. **Vienna Imperial virtual grand piano**. 16 May 2022. Available from: <https://www.vsl.co.at/en/Keyboards_Complete/Vienna_Imperial>. Visited on: 16 May 2022.

VOCOBOX. **Human Voice Dataset**. original-date: 2014-12-27T21:21:24Z. 2015. Available from: <https://github.com/vocobox/human-voice-dataset>. Visited on: 8 Mar. 2021.

WESSEL, David; WRIGHT, Matthew. Problems and Prospects for Intimate Musical Control of Computers. **Computer Music Journal**, v. 26, n. 3, p. 11–22, Sept. 2002. ISSN 0148-9267, 1531-5169. DOI: 10.1162/014892602320582945. Available from: <https://direct.mit.edu/comj/article/26/3/11-22/94758>. Visited on: 16 Mar. 2021.

WHEELER, Gerald F.; CRUMMETT, William P. The vibrating string controversy. **American Journal of Physics**, American Association of Physics Teachers (AAPT), v. 55, n. 1, p. 33–37, Jan. 1987. DOI: 10.1119/1.15311.

WHITHAM, G. B. **Linear and Nonlinear Waves**. Hoboken, New Jersey, United States: John Wiley & Sons, Inc., June 1999. DOI: 10.1002/9781118032954.

WILKINS, Julia; SEETHARAMAN, Prem; WAHL, Alison; PARDO, Bryan. Vocalset: A Singing Voice Dataset. In: 19TH International Society for Music Information Retrieval Conference. Paris,France: Zenodo, Mar. 2018. type: dataset. DOI: 10.5281/ZENODO.1203819. Available from: <https://zenodo.org/record/1203819>. Visited on: 3 Mar. 2021.

WOLF, Mark. **Encyclopedia of video games : the culture, technology, and art of gaming**. Santa Barbara, California: Greenwood, an imprint of ABC-CLIO, LLC, 2021. ISBN 9781440870200.

WU, Bin; YU, Bailang; HUANG, Chang; WU, Qiusheng; WU, Jianping. Automated extraction of ground surface along urban roads from mobile laser scanning point clouds. **Remote Sensing Letters**, Informa UK Limited, v. 7, n. 2, p. 170–179, Dec. 2015. DOI: 10.1080/2150704x.2015.1117156.

XLN AUDIO. **Studio Grand**. 16 May 2022. Available from: <https://www.xlnaudio.com/products/addictive_keys/instrument/studio_grand>. Visited on: 16 May 2022.

XU, Xin; CISEWSKI-KEHE, Jessi; DAVIS, Allen B.; FISCHER, Debra A.; BREWER, John M. Modeling the Echelle Spectra Continuum with Alpha Shapes and Local Regression Fitting. **The Astronomical Journal**, American Astronomical Society, v. 157, n. 6, p. 243, May 2019. DOI: 10.3847/1538-3881/ab1b47.

YAN, Zhipei; SCHILLER, Stephen; WILENSKY, Gregg; CARR, Nathan; SCHAEFER, Scott. k -curves. **ACM Transactions on Graphics**, Association for Computing Machinery (ACM), v. 36, n. 4, p. 1–7, July 2017. DOI: 10.1145/3072959.3073692.

YANG, Cong; GRZEGORZEK, Marcin; LUKASIK, Ewa. Representing the evolving temporal envelope of musical instruments sounds using Computer Vision methods. In: SIGNAL Processing: Algorithms, Architectures, Arrangements, and Applications (SPA). Poznań: IEEE, Sept. 2015. DOI: 10.1109/spa.2015.7365137.

YANG, Cong; TIEBE, Oliver; GRZEGORZEK, Marcin; LUKASIK, Ewa. Skeleton-based audio envelope shape analysis. In: 3RD IAPR Asian Conference on Pattern Recognition (ACPR). Nanjing, China: IEEE, Nov. 2015. DOI: 10.1109/acpr.2015.7486556.

YANG, Lijun; YANG, Zhihua; YANG, Lihua. The theoretical analysis for an iterative envelope algorithm. **Digital Signal Processing**, v. 38, p. 32–42, 2015. ISSN 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2014.12.006. Available from: <https://www.sciencedirect.com/science/article/pii/S1051200414003480>.

YANG, Lijun; YANG, Zhihua; ZHOU, Feng; YANG, Lihua. A novel envelope model based on convex constrained optimization. **Digital Signal Processing**, Elsevier BV, v. 29, p. 138–146, June 2014. DOI: 10.1016/j.dsp.2014.02.017.

YANG, Wei; YANG, Yuanhong; YANG, Mingwei. Fast digital envelope detector based on generalized harmonic wavelet transform for BOTDR performance improvement. **Measurement Science and Technology**, IOP Publishing, v. 25, n. 6, p. 065103, Apr. 2014. DOI: 10.1088/0957-0233/25/6/065103.

ZEILER, Matthew D. ADADELTA: An Adaptive Learning Rate Method, 22 Dec. 2012. arXiv: 1212.5701 [cs.LG]. Available from: <https://arxiv.org/abs/1212.5701>.

ZHU, Zhi; MIYAUCHI, Ryota; ARAKI, Yukiko; UNOKI, Masashi. Contributions of temporal cue on the perception of speaker individuality and vocal emotion for noise-vocoded speech. **Acoustical Science and Technology**, Acoustical Society of Japan, v. 39, n. 3, p. 234–242, May 2018. DOI: 10.1250/ast.39.234.

ZIELINSKI, Slawomir. On Some Biases Encountered in Modern Audio Quality Listening Tests (Part 2): Selected Graphical Examples and Discussion. **Journal of the Audio Engineering Society**, v. 64, n. 1, p. 55–74, Feb. 2016. ISSN 15494950. DOI: 10.17743/jaes.2015.0094. Available from: <http://www.aes.org/e-lib/browse.cfm?elib=18105>. Visited on: 30 Mar. 2021.

ZIELINSKI, Slawomir; RUMSEY, Francis; BECH, Søren. On Some Biases Encountered in Modern Audio Quality Listening Tests-A Review. **Journal of the Audio Engineering Society**, v. 56, n. 6, p. 427–451, 2008. Available from: <http://www.aes.org/e-lib/browse.cfm?elib=14393>.

ZILS, Aymeric; PACHET, François. Musical mosaicing. In: CITESEER. PROCEEDINGS of the COST G-6 Conference on Digital Audio Effects (DAFX-01). Limerick, Ireland: DAFX-01, 2001. v. 2, p. 135.

ZORDAN, Davide; MARTINEZ, Borja; VILAJOSANA, Ignasi; ROSSI, Michele. On the Performance of Lossy Compression Schemes for Energy Constrained Sensor Networking. **ACM Transactions on Sensor Networks**, v. 11, n. 1, p. 1–34, Nov. 2014. ISSN 1550-4859, 1550-4867. DOI: 10.1145/2629660. Available from: <https://dl.acm.org/doi/10.1145/2629660>. Visited on: 3 Mar. 2021.

ZOU, Fangyu; SHEN, Li; JIE, Zequn; ZHANG, Weizhong; LIU, Wei. A Sufficient Condition for Convergences of Adam and RMSProp. In: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Berlin: CVPR, June 2019.