DIGITAL MODELLING OF GUITAR AUDIO EFFECTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ENGİN ZEKİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2015

Approval of the thesis:

DIGITAL MODELLING OF GUITAR AUDIO EFFECTS

submitted by **ENGİN ZEKİ** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. M. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Gönül Turhan Sayan _____
Head of Department, **Electrical and Electronics Engineering**

Prof. Dr. Tolga Çiloğlu _____
Supervisor, **Electrical and Electronics Eng. Dept., METU**

Examining Committee Members:

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Eng. Department, METU _____

Prof. Dr. Tolga Çiloğlu
Electrical and Electronics Eng. Department, METU _____

Prof. Dr. Uğur Halıcı
Electrical and Electronics Eng. Department, METU _____

Prof. Dr. Umut Orguner
Electrical and Electronics Eng. Department, METU _____

Prof. Dr. Salim Kayhan
Electrical and Electronics Eng. Dept., Hacettepe University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ENGİN ZEKİ

Signature :

ABSTRACT

DIGITAL MODELLING OF GUITAR AUDIO EFFECTS

Zeki, Engin
M.S., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Tolga Çiloğlu

January 2015, 126 pages

Digital audio effects are used by many electric guitar players. These effects help players to find their desired tones and sounds. Digital audio workstations (DAW) and Virtual studio technologies (VST) advance the development of guitar audio effects softwares. The aim of this thesis is to implement main linear guitar effects (delay, reverb, wah-wah, flanger) real-time and enhance the main nonlinear guitar effects (distortion, overdrive) using system identification methods. After the verification of the effect models with MATLAB, real-time implementations of these effects will be done using C/C++/C# software languages. For the modelling of main nonlinear guitar effects, distortion and overdrive, this thesis investigates current methods of static modelling and dynamic nonlinear state space solutions. After discussion of previous models, this thesis introduces a new method of distortion modelling with system identification called Enhanced Modelling of Guitar Distortion. Enhanced Modelling of Guitar Distortion algorithm will use neural network system identification method ANFIS (Adaptive-Network-Based Fuzzy Inference System). ANFIS is used as a system identification tool in Enhanced Modelling of Guitar Distortion algorithm. This algorithm takes the guitar output signal and pre-amplifies the input with 12AX7 vacuum tube amplifier simulation model to obtain clean channel. ANFIS system identification block is trained using desired distortion effect input output pair. This training and learning results into a ANFIS (Adaptive-Network-Based Fuzzy Inference System) structure that can be used for processing future inputs. Using clean channel

output as an input to the ANFIS structure, lead channel output is obtained. Real-time implementations of distortion and overdrive will be done using C/C++/C# software languages.

The results of the Enhanced Modelling of Guitar Distortion algorithm will show if it is applicable to model distortion and overdrive effects using system identification approach. Listening tests will be done to provide evidence that the simulation models are successful. Strict real-time constraints are going to be satisfied and the simulation of a guitar amplifier will be presented.

Keywords: Guitar, Audio, Audio Signal Processing, Nonlinear Signal Processing, Delay, Comb Filter, Wah-Wah, Flanger, Distortion, Overdrive, ANFIS, System Identification, Valve Tube Modeling, Guitar Effects

ÖZ

GİTAR EFEKTLERİNİN DİJİTAL MODELLENMESİ

Zeki, Engin
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Tolga Çiloğlu

Ocak 2015 , 126 sayfa

Çoğu elektrik gitarist dijital ses efektlerini kullanmaktadır. Bu efektler elektrik gitaristlerin istedikleri tonları ve sesleri elde etmelerini sağlamaktadır. Dijital Ses Çalışma Ortamları (DAW) ve Sanal Stüdyo Teknolojileri (VST) ses efektleri yazılımlarının geliştirilmesi ve yaygınlaşmasını sağlamışlardır. Bu tezin amacı genel doğrusal gitar efektlerinden olan delay, reverb, wah-wah ve flanger efektlerinin gerçek zamanlı olarak modellenmesi ve doğrusal olmayan gitar efektlerinden distortion ve overdrive efektlerinin sistem tanımlama algoritmalarını kullanarak geliştirilmesidir. MATLAB kullanılarak efektlerin doğrulanmasının ardından, gerçek zamanlı olarak C/C++/C# yazılım dilleri kullanılarak uygulaması yapılacaktır. Genel doğrusal olmayan efektlerden olan distortion ve overdrive efektlerinin modellenmesi için şu anda uygulanan metodlardan statik modelleme ve dinamik denklem çözümleri incelenecektir. Sistem tanımlama metodları kullanan Enhanced Modelling of Guitar Distortion algoritması tasarlanacaktır. Bu algoritma sinir ağları yapısı kullanan sistem tanımlama ANFIS (Adaptive-Network-Based Fuzzy Inference System) modelini kullanacaktır. Distortion ve overdrive gitar efektlerinin de C/C++/C# yazılım dilleri kullanılarak uygulaması yapılacaktır. ANFIS yapısı bir sistem tanımlama aracı olarak Enhanced Modelling of Guitar Distortion modeli içerisinde yer almaktadır. Bu algoritma gitar sinyalini alarak ön amfi olan 12AX7 vakum tüp amfi simülasyon modelinden geçirerek temiz kanalı elde etmektedir. İstenen distortion efekt giriş çıkış sinyali ile ANFIS sistem tanımlama bloğu eğitilir. Bu eğitilme ve öğrenme sonucunda bir ANFIS yapısı elde

edilmektedir. Bu ANFIS yapısı ile gelecek giriş sinyalleri de işlenebilmektedir. Temiz kanal çıkışının ANFIS yapısına giriş olarak kullanılmasıyla kirli kanal çıkışı da elde edilebilmektedir.

Enhanced Modelling of Guitar Distortion algoritmasının sonuçları sistem tanımlama metodlarının distortion ve overdrive gitar efektlerinin modellenmesinde kullanılabileceğini gösterecektir. Dinleme testleri sonucunda simülasyon modellerinin başarılı olduğuna kanaat getirilecektir. Gerçek zaman sınırlandırmaları öngörülerek, gitar ses güçlendiricisi simülasyonu sunulacaktır.

Anahtar Kelimeler: Gitar, Ses, Ses Sinyal İşleme, Doğrusal Olmayan Sinyal İşleme, Erteleme, Filtreleme, Wah-Wah Efekti, Flanger, Distortion, Overdrive, ANFIS, Sistem Tanımlama, Triode Modelleme, Gitar Efektleri

To my beloved parents

Cevdet Zeki, Huriye Zeki

ACKNOWLEDGMENTS

I personally thank Prof. Dr. Tolga Çiloğlu for letting me choose my own topic on audio signal processing, and all his efforts to support me in the preparation of this thesis. Through the search of a thesis topic, Bülent Bıyıkoğlu, the creator of SynthMaster and founder of KV331, lead me to the path of choosing digital audio effects, offering me to read some papers about Distortion Modelling and Synthesis, I also personally thank him for being a guide for me. Middle East Technical University, with all their precious lecturers and researches has been a home for me both for research and work experience purposes, I value these times of my life the most.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ANFIS	Adaptive-Network-Based Fuzzy Inference System
ASIO	Audio Stream Input Output by Steinberg
CPU	Central Processing Unit
DAFx	Digital Audio Effects
DAW	Digital Audio Workstation
DSP	Digital Signal Processing, Digital Signal Processor
FFT	Fast Fourier Transform
FIS	Fuzzy Inference System
FIR	Finite Impulse Response
FX	Effect
IIR	Inifinite Impulse Response
LCCDE	Linear Constant Coefficient Difference Equation
LTI	Linear Time-Invariant
MB	Mega Byte
RAM	Random Access Memory
RMSE	Root Mean Square Error
SDK	Software Development Kit
VST	Virtual Studio Technology
WDF	Wave Digital Filter

CHAPTER 1

INTRODUCTION

Electric guitar is home for many analog and digital effects, with its ability to convert vibration of metal strings into electrical signal. The guitar effects world is divided into two: analog effects and digital effects. Analog effects provide high fidelity, quality and organic sounds, while digital effects provide mobility, quantization and package of effects in a single box or software. The most common effects that an electric guitar player use are: Overdrive, Distortion, Delay, Reverb, Wah-Wah and Flanger. These are the main effects where other customized effects are derived of.

This thesis focuses on the real-time implementations of delay, reverb, wah-wah, flanger from the linear guitar audio effects and enhancement of overdrive and distortion from nonlinear guitar audio effects. Emerging with the enhancements in the ways of personal computing and digital signal processing, there is a need to emulate, simulate and synthesize digital effects successfully compared to their analog counterparts. Both from electrical engineering society and audio engineering world, researchers contribute to the evolution of various digital guitar effects. The Center for Computer Research in Music and Music Acoustics research group at Stanford University has put a tremendous work on the subject with J.O. Smith's books and papers on digital audio effects. Udo Zölzer and his team at Helmut Schmidt University of Hamburg contributed to digital audio effects with their work on State Variable Solutions and Enhanced Triode Models of Vacuum Tube Amplifiers.

High quality electric guitar audio equipments are expensive and people tend to look out for cheaper and effective solutions to obtain high quality effects in a package. As a result, we have seen an emergence of software digital effects modellers and

technical research papers and books about the topic. The book called Digital Audio Effects, DAFx by Udo Zölzer, contains research papers about digital audio effects [21]. These papers are the starting reference for this thesis with neat implementations and explanations of audio effects. Deriving from those chapters of work, I tried to enhance distortion and overdrive and implement delay, reverb, wah-wah and flanger real-time with my own experience as a 10 year electric guitar player. There are lots of brands, manufacturers and designers in the audio effects industry having diverse products with multiple effects implementations generally divided into two categories: analog and digital. Digital audio effects are also divided into subcategories: Linear Audio Effects and Nonlinear Audio Effects.

Linear audio effects include the digital audio effects where the output can be expressed by the linear combination of the input and output signals' present and past values (i.e. as a LCCDE). Delay, Reverb, Wah-Wah and Flanger are linear audio effects and can be expressed by LCCDEs. Real-Time implementations of these effects can be challenging in some audio processing applications, such as DAW (Digital Audio Workstations) and Guitar Audio Processing Softwares, on Windows, OS X and Linux environments. For this reason, these effects are implemented with strict real-time constraints that does not effect the electric guitar player's performance, and throughout the chapters this emphasis is given to the reader.

Nonlinear audio effects are mostly related with overdrive, fuzz and distortion, where the relation between input and output signals can not be expressed in linear terms. In that case, nonlinear signal processing algorithms can be used for modelling nonlinear guitar effects. Different methods are implemented by researchers. Basic implementations started with static nonlinear models between input and output. It is found that the static implementations are not satisfactory for the players. Dynamic models that solve the nonlinear state space solution of the physical effects circuits are introduced by many researchers. This method gives good results given that the circuit is known prior to the modelling. Wave digital filter models have also been tried to blockwise model the distortion algorithm. Wave digital filters for distortion modelling is not preferred due to the reason that block models are hard to standardize between different implementations. The actual physical circuit of analog distortion and overdrive effects are generally not well known and can be modelled as a black box system.

For this reason, nonlinear system identification is an alternative way of describing the relation between the input and output of nonlinear guitar audio effects. Additionally, equalizer filters are used to get much warm tonal output from these nonlinear effects. In this thesis, we are taking a system identification approach to the distortion and overdrive modelling by introducing a neural network system identification method to model input and output behavior of highly nonlinear audio circuits. Starting with basic building blocks, such as diode rectifier, we develop a training, learning and checking mechanism for the modelling of distortion and overdrive effects found in most guitar amplifier blocks. In this context, this thesis is an advancement of the current guitar distortion modelling methods of nonlinear state space solutions, giving the designer the opportunity to model without knowing the internal circuitry of the distortion/overdrive effect in investigation. The neural network solution that has been implemented is called ANFIS (Adaptive-Network-Based Fuzzy Inference System) where fuzzy rules and membership functions are used to optimize the overall system estimation.

The second chapter of the thesis, Linear Guitar Audio Effects, presents the theoretical background of main linear guitar audio effects (delay, reverb, wah-wah and flanger). The real-time implementations are provided in the appendix section. The emergence and development of these effects are investigated in this chapter. In order to model these effects, actual effects pedal's input and output values are recorded. These input and output values are then used to formulate the impulse response and filter characteristics. Delay guitar effect is an ideal delay with gain and delay parameters. Reverb is implemented using the colorless artificial reverberation algorithm introduced by Schroeder [16]. Wah-wah is modelled after the Jim Dunlop' s Original GCB95 Wah-Wah guitar effects pedal. State variable filter structure is used to model the changing behavior of this effect. Flanger is implemented by introducing sinusoidally changing delay to the input signal. All of these effects are used commonly by electric guitar players. Equalizer filters are also implemented for each of these effects to obtain similar output of guitar amplifier cone speaker's.

The third chapter, Basics of Guitar Distortion: Diode Limiter and Pre-Amplifier introduces the basic concepts and methodologies to model guitar distortion and overdrive. These guitar effects are highly nonlinear, and both of them contain diode limiter and

pre-amplifier structures. Diode limiter is a rectifier circuit with low-pass characteristics. Pre-amplifier in guitar amplifiers is the first stage of the input signal amplification. The guitar output signal is amplified by a vacuum tube amplifier 12AX7 in the pre-amplifier section. The simulation models for the diode limiter and vacuum tube amplifier is presented with circuit examples and parameter settings. 12AX7 vacuum tube amplifier has different versions due to differences between manufacturers. RSD-1, RSD-2 and EHX-1 are the most common ones used in guitar amplifiers. In the end of the chapter, the different parameter settings are given for these different tube model versions.

The fourth chapter discusses a different method of guitar distortion modelling using wave digital filters. This method is implemented in various papers in the field as an alternative method of simulation of vacuum tube amplifiers. However, wave digital blockwise models are not preferred due to nature of the strict real-time constraints of the nonlinear simulation. Since each blockwise model introduces significant delay in the path, the overall delay can be higher than expected. For this reason, just the basics of the method is mentioned and implementations of wave digital filters approach for distortion modelling is not included.

The fifth chapter introduces a system identification approach to model highly nonlinear guitar audio effects: distortion and overdrive. Firstly, static nonlinearity approach to distortion modelling is discussed. Dynamic models are selected to get better results in simulation. Among dynamic models, we have selected neural network system identification method, ANFIS (Adaptive-Network-Based Fuzzy Inference System) to model distortion/overdrive. ANFIS method is introduced to the reader in this chapter. The application of ANFIS algorithm is shown with specific implementations of Line 6 Spider IV amplifier and Marshall JCM900 pre-amplifier. With the help of ANFIS algorithm, we have developed a method called Enhanced Modelling of Guitar Distortion. This enhanced method uses ANFIS as a system identification tool to model the distortion effect in the lead channel of amplifier. Enhanced Modelling of Guitar Distortion is a 6 step algorithm summarized as follows:

1. The guitar output signal, $x[n]$, is used as an input to the 12AX7 pre-amplifier model.

2. Then the equalizer high-pass and band-pass filters are used to change tonal characteristics of pre-amplifier output.

3. EL84 vacuum tube power amplifier model amplifies the equalized signal. The output of the vacuum tube power amplifier is called clean channel output, $y_c[n]$.

4. ANFIS System Identification block model takes two inputs, $x_a[n]$ and $y_a[n]$. $x_a[n]$ is the training input signal to the distortion effects pedal that is modelled. $y_a[n]$ is the output of the distortion effects pedal for the training input $x_a[n]$. ANFIS System Identification code block trains these input output sequence to form an ANFIS structure called $outfis$.

5. This ANFIS structure can be used to evaluate future outputs for random inputs coming from the $y_c[n]$ clean channel. ANFIS Evaluator block evaluates the lead channel output, $y_l[n]$, using the ANFIS structure, $outfis$ and $y_c[n]$ as inputs.

6. The main channels' simulation of a guitar amplifier, clean channel ($y_c[n]$) and lead channel ($y_l[n]$) outputs are obtained.

Enhanced Modelling of Guitar Distortion algorithm introduces a training, learning and modelling approach to generic distortion algorithms. Numerous different distortion pedals can be modelled adjusting the number of membership functions and epochs. The algorithm provides flexible modelling scheme for the effects modeller. Previous distortion modelling algorithms focus on directly solving the nonlinear state space equation of a specific circuit. This final chapter concludes with the RMSE results obtained from simulating idealized circuits. RMSE values change with the ANFIS system design parameters: number of membership functions and epoch. These parameters are selected in correspondence with the order of nonlinearity and dynamics of the distortion effect that is investigated. The listening tests are done to verify the actual output of the Enhanced Modelling of Guitar Distortion Algorithm. It is also verified that the algorithm models sound nearly same as the real amplifiers.

Each guitar effect is designed in $MATLAB$, implemented in Microsoft Visual Studio 2010 using C++ and C# and then ported to Texas Instruments TMS320VC5505 eZdsp. $MATLAB$ is used for the initial design of the guitar effects algorithms. Microsoft Visual Studio 2010 is used for real time application development in the

Windows platform. ASIO (ASIO4ALL) - Universal ASIO Driver for WDM Audio is used to satisfy real-time development constraints. ASIO drivers can be set with 64 samples of ASIO Buffer Size, which is the best real-time constraint to satisfy in ASIO. Latency Compensation parameters are set to 0 samples for In and Out. Buffer Offset is also set to 0 ms. Realtek High Definition sound card is used for the testing. Realtek HD Audio output and Realtek HD Audio Line input are used for effect output and input.

Boss Inc., Mesa Boogie Inc., Marshall Inc., Line 6 Inc., Digitech Inc., Jim Dunlop Inc. are all trademarks having their own rights on their products, all of the mentions and references of implementations in this thesis are for academic and research purposes only.

CHAPTER 2

LINEAR GUITAR AUDIO EFFECTS

2.1 Basic Building Block: Delay

2.1.1 FIR Delay

In signal processing applications, delay is the most used building block which expectedly has effects both in time and frequency domains. Generally, audio engineers in the field consider the time domain changes related to the effects on the side neglecting the changes that eventually happened on the frequency domain [21]. A handful of effects can be related to time domain processing of the audio signal having the building block delay. The delay guitar audio effect can be implemented with just shifting the discrete-time signal in time domain by an integer amount d, as mathematically stated both in time and frequency domain below (Figure 2.1).

$$y[n] = x[n - d] \tag{2.1}$$

$$Y(e^{jw}) = X(e^{jw}).e^{-jwd} \tag{2.2}$$

$$H(e^{jw}) = e^{-jwd} \tag{2.3}$$

$$H(z) = z^{-d} \tag{2.4}$$

As can be seen from Discrete-Time Fourier Transform and Z-transform, delay shifts the signal in time causing also a phase shift in frequency domain. Expectedly, the phase changes linearly with frequency, where the delay amount, d, decides on the slope. As we can see from Figure 2.2, the magnitude of the filter stays at 0 dB for all

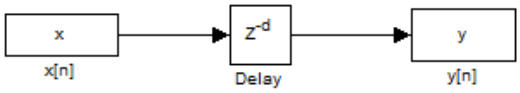

Figure 2.1: The Basic Building Block: Delay.

frequencies meaning that this filter does not affect magnitude while linearly changing phase. In digital guitar audio effect delay, the delayed signal is summed up to the original signal, as described in the next section: FIR Comb Filter.

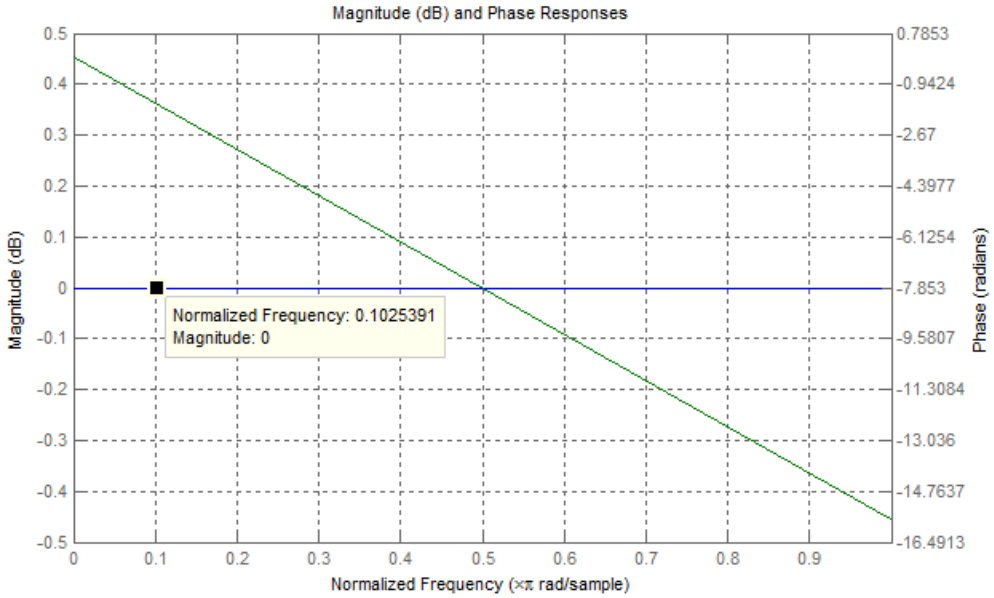

Figure 2.2: Magnitude and Phase Response of Delay (d = 5).

2.1.2 FIR Comb Filter

In digital guitar audio effect delay, we generally use the input signal buffer as our reference and filter that signal with gain and delay parameters than sum it up back to the original signal. In this way, we can hear our real-time input and delayed version of our signal in delay guitar audio effect. In signal processing terms, this can be stated mathematically as below, where d is the integer amount of delay and g is the delay gain. This structure is called FIR Comb Filter.

$$y[n] = x[n] + g.x[n-d] \tag{2.5}$$

$$Y(e^{jw}) = X(e^{jw}) + g.X(e^{jw}).e^{-jwd} \tag{2.6}$$

$$H(e^{jw}) = 1 + g.e^{-jwd} \tag{2.7}$$

$$H(e^{jw}) = 1 + g.cos(wd) - jg.sin(wd) \tag{2.8}$$

$$|H(e^{jw})| = \sqrt{|1 + g.cos(wd)|^2 + |g.sin(wd)|^2} \tag{2.9}$$

$$H(z) = 1 + g.z^{-d} \tag{2.10}$$

Following from Equation 2.9, we can state that magnitude response resembles a comb having amplitudes between |1+g| and |1-g|, assuming g selected as a positive constant. This statement has been made obvious by looking to Figure 2.4, where magnitudes are given in dBs, constant delay selected as d = 5 and gain g selected as g = 2. The peaks of the magnitude response occur at even multiples of 0.2π. This is due to the fact that $cos(w*d) = cos(0.4\pi*5) = cos(2\pi) = 1$ giving us the maxima of Equation 2.9. Generalizing this example, we can conclude that even multiples of π/d are the magnitude response maxima, while odd multiples of π/d are the magnitude response minima. Phase response is not linear in this case as in pure FIR delay. The effect of the delay parameter d can be basically seen by observing the magnitude response. Changing delay parameter d will affect the comb filter like compressing up and down an accordion. Increasing delay parameter d means that you will have more teeth of the comb, while decreasing d means less teeth.

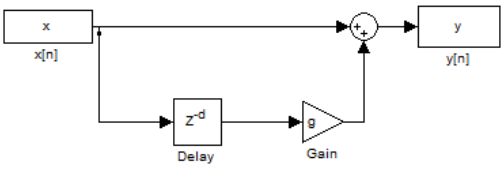

Figure 2.3: FIR Comb Filter.

FIR Comb Filter has a comb like frequency response. The filtering of the input signal with the FIR Comb Filter results into changes in the tonal characteristics. This change can be undesirable and must be treated carefully. Listening tests will help the design process of delay in that cases. The change in the frequency response can also be

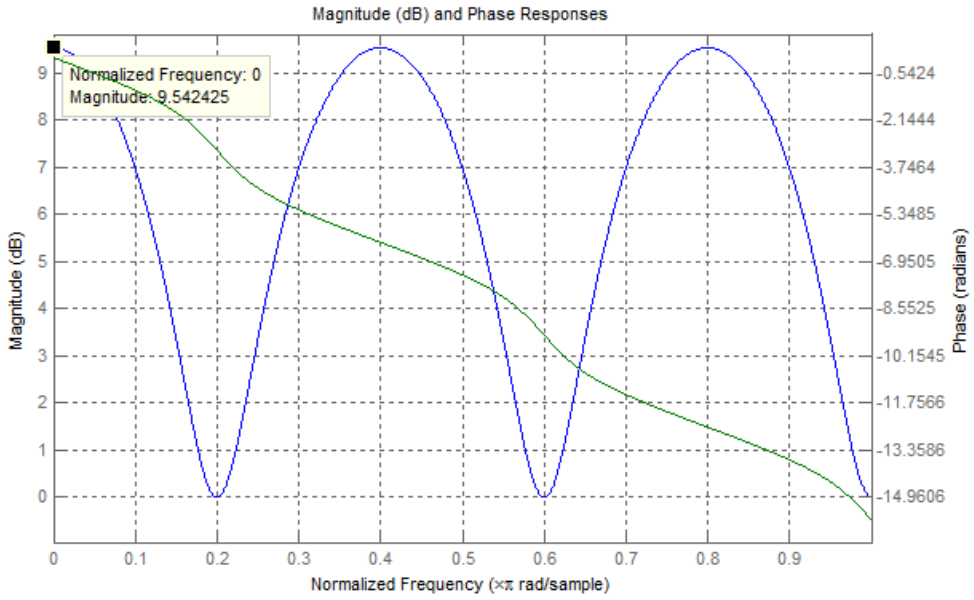

Figure 2.4: Magnitude and Phase Response of FIR Comb Filter (d =5, g=2).

used in a desirable guitar audio effect called Audio Flanger, where the delay amount changes with time.

2.2 Audio Flanger

Audio Flanger is basically a variable delay digital filter, where the amount of delay, either fractional or integer, changes with time index. The amount and pattern of change in delay defines what kind of an effect you obtain. The delay of an audio flanger generally ranges sinusoidally between 0 - 15 ms. Assuming $44100Hz(samples/sec)$ sampling rate, we take each sample on $2.2676e-05sec$ intervals. This results into a delay range of approximately 0 - 660 samples. This will in turn creates small variations over the input note that has been given by the player, sinusoidally changing with time. Different delay functions can lead to different effects and variations, such as jet airplane sounds, fuzzy and funky sounds created by customized flanger patterns. The

variable delay can be shown mathematically as below.

$$A = Floor(t_{flangemax}.T_s/2) \tag{2.11}$$

$$M[n] = Floor(A + Asin(2\pi F_{flange} n T_s)) \tag{2.12}$$

$$z[n] = x[n - M[n]] \tag{2.13}$$

$$y[n] = x[n] + g.z[n] \tag{2.14}$$

$$y[n] = x[n] + g.x[n - M[n]] \tag{2.15}$$

If we choose $T_s = 1/F_s = 2.2676e - 05sec$ where $F_s = 44100Hz$ and $t_{flangemax} = 15ms$, A turns out to be $A = 330$, which is expected, since we need a sinusoidal ranging from 0 - 660 in as our variable delay. The origins of the flange effect come from the direct tape mixing of 2 simultaneous recordings. Tiny differences on the mixing tape's speed result into the phasing effect which is mathematically same as applying a variable delay to the original mix. Flanging effect ranges from jet sound effects to chorus-like small variation effects in which you change the way and speed of your variable delay. There are considerations about the integer and fractional delay and the method to interpolate the samples. In the appendix section, details about practicality and the implementation is presented. The signal flow diagram is plotted as in Figure 2.5.

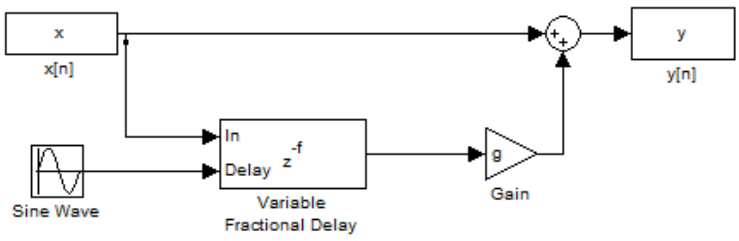

Figure 2.5: Audio Flanger.

Calling the variable delayed signal $z[n]$, we formulate the way to visualize what is happening both in time and frequency domains. Since we have an impulse response also changing with time, we expect a time dependent frequency response. Let us call the system following from $x[n]$ to $z[n]$, the response at time n to an impulse applied at $n - \tau$, as $h_1[\tau, n]$. Mathematically showing, the equations of the variable delayed signal $z[n]$ can be written as in Equation 2.16. These set of equations are

designed to introduce the linear-time variant (LTV) system of variable delay. As it is shown in Equation 2.17, the system $h_1[\tau, n]$ is a linear-time variant (LTV) system. The frequency response of a LTV system can be represented with respect to slowly changing time dimension. The impulse response $h_1[\tau, n]$ is a collection of impulse responses with respect to discrete-time parameter n. For a fixed n, we can say that the impulse response is a function of discrete-time index n, called $M[n]$. The frequency response $H_1(e^{jw}, n)$ turns out to be a phase shift by M[n] for a fixed discrete-time index n. The total response of the system, $H(e^{jw}, n)$, can be shown with respect to sample time index n in Equation 2.19.

$$z[n] = x[n - M[n]] \tag{2.16}$$

$$h_1[\tau, n] = \delta[\tau - M[n]] \tag{2.17}$$

$$H_1(e^{jw}, n) = e^{-jwM[n]} \tag{2.18}$$

$$H(e^{jw}, n) = 1 + g.e^{-jwM[n]} \tag{2.19}$$

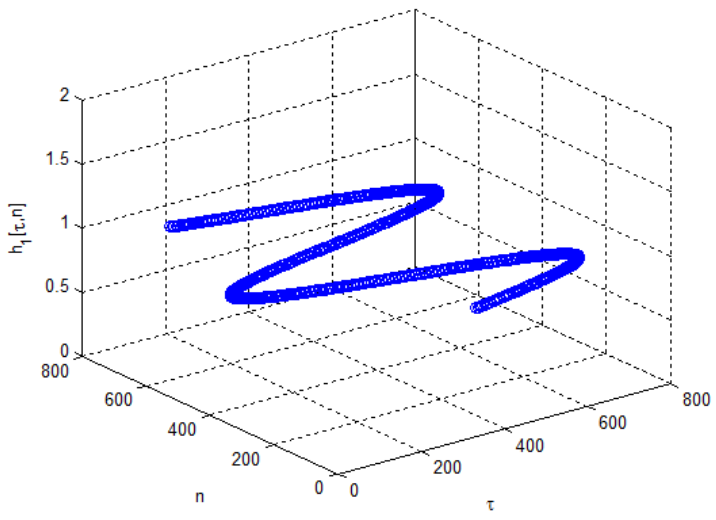

Figure 2.6: Variable Delay Impulse Response Scatter 3D Plot, $h_1[\tau, n]$.

Analyzing Figure 2.6 and Figure 2.7, we observe the changes of variable delay on both time and frequency domain. We can call n as sample index, while τ is a dummy variable. The impulse response plot in Figure 2.6 shows that for each sample time

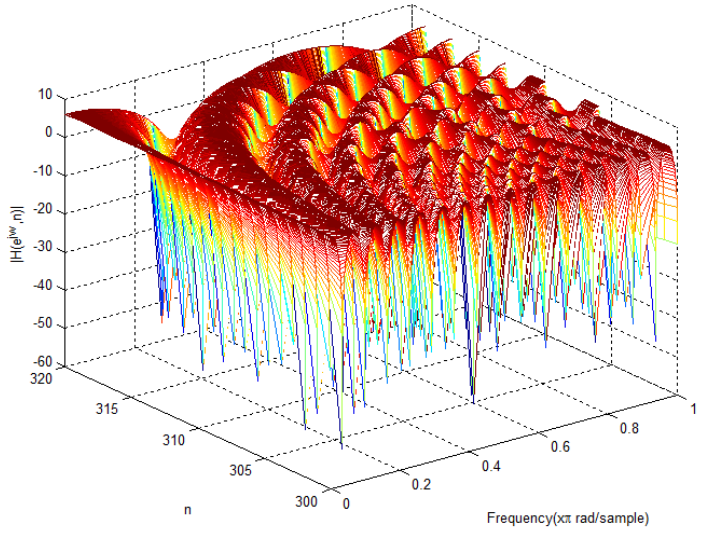

Figure 2.7: Variable Delay Frequency Response Mesh 3D Plot, $H(e^{jw}, n)$.

index n, we have a different delay. This delay has a sinusoidal pattern in the case of the original flanger effect. The frequency response plot in Figure 2.7 shows that for each sample time index n, we have a comb filter response. In comb filter frequency response, the number of peaks changes sinusoidally with time giving us a shape of an accordion. These impulse and frequency responses are just an extension of comb filter for each sample time index n.

2.3 Reverberation

In acoustics modelling and processing of a closed environment, such as a room or a concert hall, reverberation occurs when sound scatters and arrives back from surroundings. The surrounding materials' characteristics are effective to phase, amplitude and frequency characteristics of reverberation.

The information about static audio source and listening positions are important when designing a reverberation algorithm. The reverberation systems are divided into three parts: the direct sound, early reflections and late reverberation. As defined in the literature, the direct sound is the audio wave that reaches listener first [16]. Early reflections are the fastest arriving reflections from the sound source. Late reflections

are the distinct reflections coming back from the surroundings.

2.3.1 Reverberation Environment Modelling

Reverberation environments include the simulation of a room, concert hall, cave, arena, hangar, etc. Different reverberation algorithms are designed to model diverse reverberation environments. The basic of these algorithms is room acoustics modelling, where the reflections and scattering of the room is considered. The human ear does not exactly distinguish between the actual shape of the room with just only the convolved output. This is the reason why there exists no shape parameter, but only the amplitude and phase attenuations considered while designing the reverberation system.

Among the first implementations of reverberation, Manfred Schroeder's work on artificial reverberation is the most important [16]. In his work at 1960s, he designed the complex patterns of echoes using delay-based all-pass filters and recursive comb filters [16]. Mathematically, the basic recursive reverberation using delay lines is shown in Equation 2.20.

$$y[n] = -g.x[n] + x[n-m] + g.y[n-m] \tag{2.20}$$

The best way to understand what this recursive input output relation does is to look at the impulse response of the recursive reverberation system. We can also see that there is a feedback loop with a gain less than unity summed up to each output sample. This feedback loop is creating the acoustics of a reverberation environment, such as room, concert hall. The impulse response of reverberation is shown mathematically in Equation 2.21. The gain parameter, g, decides on the rate of exponential decay of the reverberation impulse response.

$$h[n] = -g.\delta[n] + (1-g^2).[\delta[n-m] + g.\delta[n-2.m] + ...] \tag{2.21}$$

The reverberation impulse response function creates a reverberation effect where the

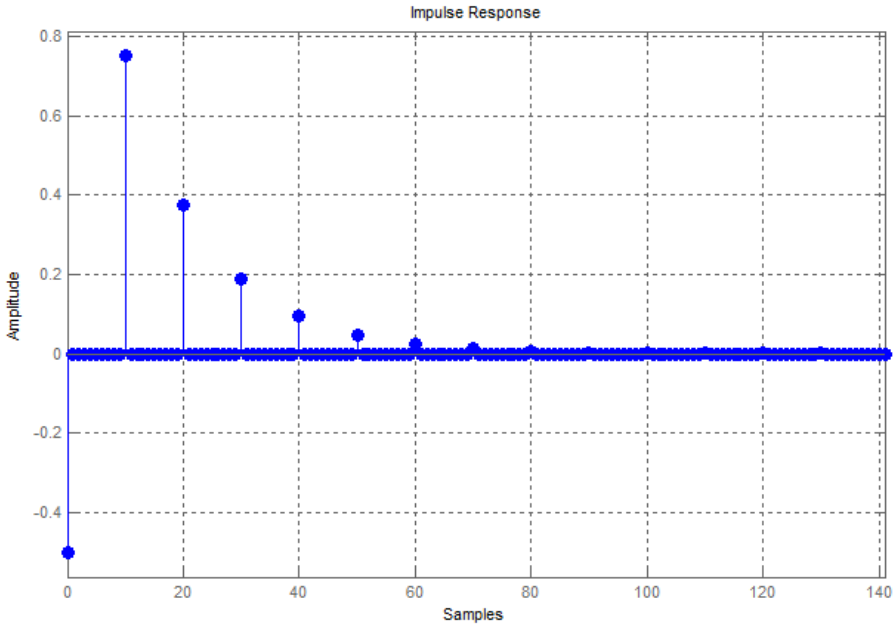

Figure 2.8: Impulse Response of the Recursive Reverb Equation.

listener thinks that there is scattering through the walls or other objects in the environment. The section will continue with the generalization of this reverberation environment, introducing the digital all-pass filter structure and its analysis leading to the colorless reverberation.

2.3.2 Digital All-Pass Filter

A digital all-pass filter, similar to its name, does not alter the magnitude of the frequency response, but changes the phase of the input signal $x[n]$. A common form of digital all-pass filter is shown mathematically in Equation 2.22.

$$A_p^g(z) = -\frac{g + z^{-1}}{1 + g.z^{-1}} \tag{2.22}$$

$$A_p^g(e^{jw}) = -\frac{g + e^{-jw}}{1 + g.e^{-jw}} \tag{2.23}$$

$$|A_p^g(e^{jw})| = \frac{|g + e^{-jw}|}{|1 + g.e^{-jw}|} \tag{2.24}$$

$$|A_p^g(e^{jw})| = \frac{|g + cos(w) - j.sin(w)|}{|1 + g.cos(w) - j.g.sin(w)|} \tag{2.25}$$

$$|A_p^g(e^{jw})| = \frac{\sqrt{(g + cos(w))^2 + sin^2(w)}}{\sqrt{(1 + g.cos(w))^2 + g^2.sin^2(w)}} \tag{2.26}$$

$$|A_p^g(e^{jw})| = 1 \tag{2.27}$$

A digital all-pass filter is exactly the bilinear transform of an analog all-pass filter. Digital all-pass filter acts as a phaser when used cascaded with different parameters on the signal flow path. Digital all-pass filter has a flat magnitude response, but does not have linear phase response. This feature makes this filter applicable to such effects where we do not want to change magnitude characteristics. Using digital all-pass filter we can simultaneously change phase characteristics and obtain time-varying effects. For time varying guitar audio effects, digital all-pass filter is used as a building block. In mathematical terms, the phase response of the all-pass filter is shown in Equation 2.31.

$$A_p^g(z) = -\frac{g + z^{-1}}{1 + g.z^{-1}} \tag{2.28}$$

$$Arg(A_p^g(e^{jw})) = Arg(-g - e^{-jw}) - Arg(1 + g.e^{-jw}) \tag{2.29}$$

$$Arg(A_p^g(e^{jw})) = Arg(-g - cos(w) + j.sin(w)) - Arg(1 + g.cos(w) - j.g.sin(w)) \tag{2.30}$$

$$Arg(A_p^g(e^{jw})) = arctan(\frac{sin(w)}{-g - cos(w)}) - arctan(\frac{-g.sin(w)}{1 + g.cos(w)}) \tag{2.31}$$

Looking at the Figure 2.9, we can see a flat frequency response that characterizes the all-pass filter. The phase response is the direct representation of the Equation 2.31, which decreases with frequency.

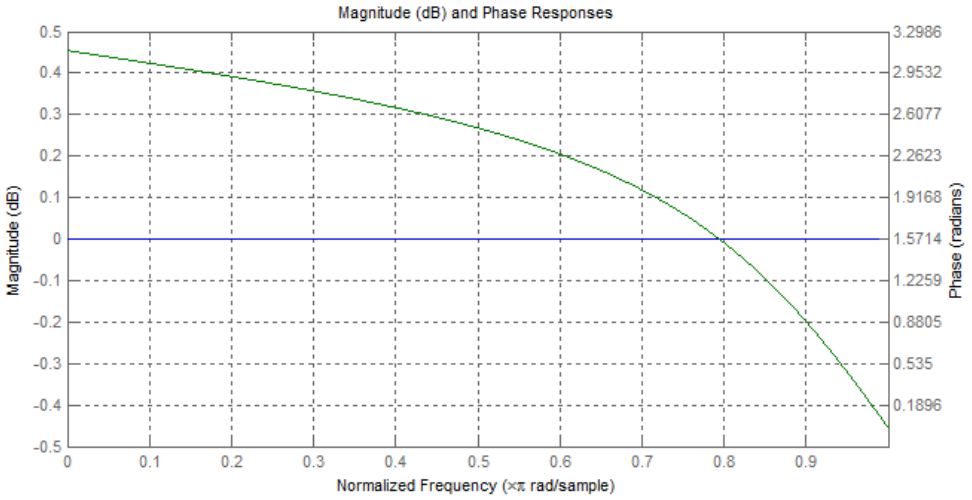

Figure 2.9: Digital All-Pass Filter Frequency and Phase Response Plot.

Another all-pass filter formulation, as discussed in the end of Reverberation Environment Modelling, is generally used for a building block for such reverberation systems, so we will continue with analyzing that recursive input output equation and using it in more complex algorithms. The recursive formula is mentioned again in Equation 2.32 and z transform and magnitude response equations are given in the following equations.

$$y[n] = -g.x[n] + x[n-m] + g.y[n-m] \tag{2.32}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{-g + z^{-m}}{1 - g.z^{-m}} \tag{2.33}$$

$$H(e^{jw}) = \frac{Y(e^{jw})}{X(e^{jw})} = \frac{-g + e^{-jwm}}{1 - g.e^{-jwm}} \tag{2.34}$$

$$|H(e^{jw})| = \frac{|-g + e^{-jwm}|}{|1 - g.e^{-jwm}|} \tag{2.35}$$

$$|H(e^{jw})| = \frac{|-g + cos(wm) - j.sin(wm)|}{|1 - g.cos(wm) + j.g.sin(wm)|} \tag{2.36}$$

$$|H(e^{jw})| = \frac{\sqrt{(-g + cos(wm))^2 + sin^2(wm)}}{\sqrt{(1 - g.cos(wm))^2 + g^2.sin^2(wm)}} \tag{2.37}$$

$$|H(e^{jw})| = 1 \tag{2.38}$$

The all-pass filter is the building block for general reverberation effects and used

extensively in the literature of audio effects. The MATLAB code for this kind of basic building is given as a function allpass.m in the following listing.

```
function y = allpass(x,M,g)
%Author: Engin Zeki, METU - Signal Processing, 2013
%ALLPASS filter building block
%x = input signal
%M = delay (number of samples)
%g = all-pass filter gain
%y[n] = -g.x[n] + x[n-M] +g.y[n-M]
M = int32(M);
N = length(x);
y = zeros(1,N);

    %Same with the IIR comb filter
    for n=1:N
        if (n-M) > 0
            y(n) = -g*x(n) + x(n-M) + g*y(n-M);
        else
            y(n) = -g*x(n);
        end
    end
end
```

Figure 2.10: Digital All-Pass Filter MATLAB Function Code Block.

The problem of reverberation is also discussed by Julius O. Smith III later in his work called Physical Audio Signal Processing [18]. In this literature, it is mentioned that the direct signal propagating from a sound source can be simulated using a single delay line in series with an attenuation scaling or low-pass filter. In addition to that since each wave is just a relation between a sound source and the listener, these waves can be modelled as a feed-forward comb filter. Feed-forward comb filter includes a tapped delay line to introduce the reflections. The model consists of changing amplitude and phase of the source signal and then passing through a designed filter to model the effects of air absorption and reflection material [2]. The drawback of this method is that we can only model source to listener (point to point) audio acoustics. Such a transfer function system for two sources and a listener with two ears (sinks / outputs) can be modelled in $MATLAB$ as in Figure 2.11.

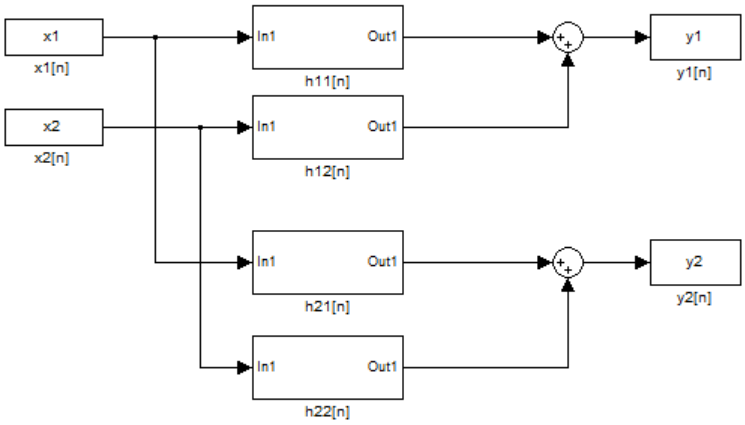

Figure 2.11: Exact Reverb via Transfer-Function Modelling.

As can be seen in the figure, $x_1[n]$ and $x2[n]$ are the input audio sources, $h_{11}[n]$ and $h_{12}[n]$ are the transfer functions reaching to first ear output $y_1[n]$ and $h_{21}[n]$ and $h_{22}[n]$ are the transfer functions reaching to second ear output $y_2[n]$ respectively. The outputs $y_1[n]$ and $y_2[n]$ can be seen as our two ears left and right respectively in this case. The transfer function can be shown as a matrix relation in z-domain as following.

$$\begin{bmatrix} Y_1(z) \\ Y_2(z) \end{bmatrix} = \begin{bmatrix} H_{11}(z) & H_{12}(z) \\ H_{21}(z) & H_{22}(z) \end{bmatrix} \begin{bmatrix} X_1(z) \\ X_2(z) \end{bmatrix} \tag{2.39}$$

It is stated that in order to compute the output coming to each ear $y_1[n]$ and $y_2[n]$, we have to take convolutions in the number of audio sources times number of audio sinks (ears), which is 4 convolutions (2 audio sources times 2 audio sinks (ears)) [18]. Mathematically this convolution can be shown in Equation 2.40.

$$y_i[n] = \sum_{k=1}^{2} x_k[n] * h_{ik}[n] = \sum_{k=1}^{2} \sum_{m=0}^{M_{ij}} x_k[n].h_{ik}[n-m], i = 1, 2. \tag{2.40}$$

In the case of electric guitar reverberation, we just have one audio source, the electric guitar input and two distinct systems processing the same input giving left and right ear outputs $y_1[n]$ and $y_2[n]$ respectively. This kind of a system can be shown as in the following $MATLAB$ Figure 2.12.

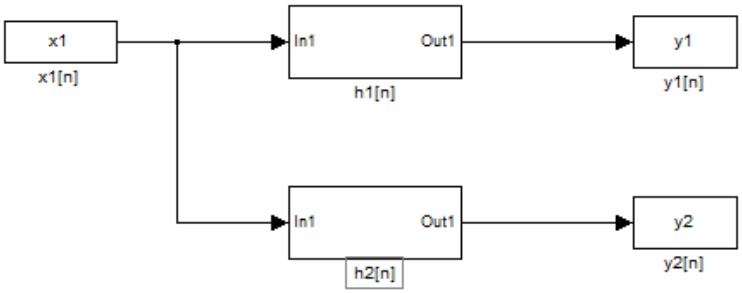

Figure 2.12: Exact Guitar Reverb via Transfer-Function Modelling.

The exact equation for electric guitar reverberation model is given in Equation 2.41. The guitar input is stated as $x[n]$, two systems for left and right as $h_1[n]$ and $h_2[n]$ respectively and two outputs for left and right ear, $y_1[n]$ and $y_2[n]$ respectively.

$$y_i[n] = \sum_{i=1}^{2} x[n] * h_i[n] = \sum_{m=0}^{M_i} x[n].h_i[n-m], i = 1, 2. \tag{2.41}$$

Modelling impulse response between each source and listener can be computationally complex. For these reasons, the methods to model reverberation virtually become much more convenient and later called as artificial reverberation. Modelling the acoustic reverberation environment also introduces delay restricting real-time implementation because of the reason that the number of multiplications and additions got much higher with each inch of width, height and length of the room. The solution to make calculations much easier and model the reverberation patterns, artificial reverberation algorithms are introduced to overcome these problems.

2.3.3 Artificial Reverberation

Artificial Reverberation is the perceptual modelling of reverberation, in which methods are devised to recreate reverberation studying the behavior of signal following from sources to sinks. These methods are efficient in the way that simple models mimicking the actual reverberation paths are used in the algorithms.

First of all, there are two terms to begin with: echo density and mode density. Echo density is the number of reflected and returned sound waves from audio source to sink

per unit time. Echo density is square proportional with time, t, that starts from the initial audio wave leaving the sink. So as time passes the echo density increases square proportional in a way that we can model echos as random processes and sample this processes to obtain our late reverberation model. In this section, it is necessary to mention Manfred R. Schroeder with his work Colorless Artificial Reverberation and Natural Sounding Artificial Reverberation which are explained in the next section.

2.3.3.1 Colorless Artificial Reverberation

In 1961, Manfred R. Schroeder and B. F. Logan presented a paper called "Colorless" Artificial Reverberation on Journal of Audio Engineering Society [16]. This paper introduced the term digital reverberation. Modern reverberation techniques rely on digital reverberation implementations. This paper is regarded as a milestone in digital audio effects for that reason.

The need for the design of artificial reverberation comes from the costly and inefficient implementations of reverberation on analog domain. The research to simulate multiple reflections in a room lead to the design of building blocks to simulate these reflections. "Colored" reverberation is introduced where reverberation is implemented with changes in the frequency response. The work of Schroeder and Logan aimed for the solution of this problem suggesting an all-pass filter with same reverberation effect.

Electronic reverberators have definite advantages over modelling real reverberation chambers directly. Price of modelling and having a reverberation chamber is not something everyone can afford. On the other hand an electronic reverberator is easy to implement on any microprocessor architecture with embedded audio codec. However, an ideal way to represent reverberation should be artificial in the sense that it can be easily represented as a LTI system overall.

Simulating the frequency response of large rooms is a chanllenging problem in reverberation algorithm designs. In his paper, Schroeder states that a room can be characterized by its normal modes of vibration which can be described mathematically as in Equation 2.42.

$$Number\ of\ Modes = \frac{4\pi V}{c^3} f^2 \tag{2.42}$$

From Equation 2.42, we can see that number of modes are independent from room shape and square proportional with frequency. In this equation V is the volume of the room, c is the velocity of sound and f is the frequency. There exists a term of critical frequency f_c, which is the moment when many modes overlap due to the high density of echoes. The critical frequency can be shown mathematically in Equation 2.43.

$$f_c = 2000\sqrt{\frac{T}{V}} \tag{2.43}$$

In the Equation 2.43, T is the reverberation time in seconds and V is the volume of the room in m^3. When the frequency gets near and above this frequency range, it is hard to hear individual scatterings and echoes due to the increased number of modes resulting into irregular amplitude frequency response overall. For a human ear, it is hard to distinguish these fluctuations in the frequency response so we perceive them as an overall change and a collection of each impulse response into a totalized impulse response. The method to test reverberation chambers is to apply sine waves having different frequencies leading to different impulse responses and hearing the output signals reflected back from the walls of the room. Another much effective method is to use psycho-acoustically enhanced test signals, specialized noises, to find much smoother impulse responses. Up until colorless reverberation, the only disadvantage of artificial electronic reverberators is its hard applicability to model the smooth frequency response of a room. The colored reverberation algorithm is shown in Equation 2.44. Magnitude response and z transform of Equation 2.44 is given in the following equations.

$$y[n] = x[n-d] + g.y[n-d] \tag{2.44}$$

$$H(e^{jw}) = \frac{e^{-jwd}}{1 - g.e^{-jwd}} \tag{2.45}$$

$$H(z) = \frac{z^{-d}}{1 - g.z^{-d}} \tag{2.46}$$

$$H(z) = \frac{z^{-10}}{1 - 0.5.z^{-10}} \tag{2.47}$$

$$H(e^{jw}) = \sum_{k=1}^{\infty} g^{k-1}.e^{-jwkd} \tag{2.48}$$

$$|H(e^{jw})|^2 = \frac{1}{1 + g^2 - 2.g.cos(w.d)} \tag{2.49}$$

$$|H(e^{jw})|^2 = \frac{1}{1 + g^2 - 2.g.cos(10w)} \tag{2.50}$$

$$h[n] = \sum_{k=1}^{\infty} g^{k-1}.\delta[n - k.d] \tag{2.51}$$

As can be seen from the impulse response and its Z-Transform, we can say that the frequency response is colored, in the sense that it effects the output radically like a comb filter. The colored reverberation algorithm is not applicable to use when frequency characteristics of the input is not intended to change. In these applications, we have used the colorless artificial reverberation algorithm in order to conserve the frequency characteristics of the input signal.

2.3.3.2 Early Reflections

Early reflections are the reflections coming back from audio source to audio sink (ear) in nearly 100ms. This short time of 100ms enables us to use tapped delay lines (TDL) to model early reflections. Early reflections are important perceptually that they can create the 3D spatial positioning of the audio sources to our ears, so if possible it is better to model spatial features in early reflections.

Modelling early reflections with tapped delay lines (TDL) is the most common approach to get the desired reverberation. Air absorption and reflection pattern model can be done by filtering the input signal first and then passing it from through TDL. Early reflections and late reverberation are concurrent. At the point that early reflec-

tions start to disappear, late reverberation starts to appear. Spotting early reflections and the feedback of early reflections to late reverberation are important in reverberation design. In this way, more natural and informed late reverberation can be achieved.

2.3.3.3 Late Reverberation

Late reverberation requires a reverberation impulse response having two important features: a smooth (not too smooth) decay and a smooth (not too regular) frequency response [16]. Natural reverberation decays exponentially, this gives us a smoothly decaying impulse response overall. In addition to this a decaying noise process can be sampled and introduced to decaying late reverberation to make it sound more natural. To make the frequency response smooth and free of irregularities, one should take the mode density especially high. This choice leads modes to spread out uniformly resulting into a smooth frequency response overall.

Moorer experimented with the method of introducing exponentially decaying white noise as a late reverberation impulse response[16]. White noise corresponds to both of the conditions that is required by the late reverberation: smooth decay and smooth frequency response. When compared with the ideal late reverberation, we can say that it is more colored than the real late reverberation obtained by the introduction of the exponentially decaying white noise impulse response. What comes to mind with late reverberation is that if we can use cascaded all-pass filters, as Schroeder suggested, we can obtain desired late reverberation algorithms without changing the frequency response of the input [16]. Even for this purpose, dedicated hardware optimized for all-pass sections are built in today's audio applications. All-pass filter design allowed audio engineers to separate the design procedure to two: First to design the duration and density of the reverberation and secondly to add desired coloration to the input by filtering the input with the desired response. In order to virtualize and separate the environment, Schroeder defined the term $colorless\ reverberation$, where he defined an ideal, virtual reverberation without changing the frequency response [16].

The most common way to use all-pass filters for late reverberation modelling is to cascade them to obtain impulse expanders, where you can form your unique reverberation effects with the desired gain and delay parameters. In order to simulate that in

MATLAB, we have written our own all-pass filter function called $allpass(x, M, g)$, where x is the input signal, M is the delay in terms of samples and g is the gain parameter. Using this function, cascaded all-pass filter implementation is straightforward to implement with this line of code: $y = allpass(allpass(x, M1, g1), M2, g2);$ where $M1 = 35000;$, $M2 = 60000;$, $g1 = 0.5;$ and $g2 = 1;$. The plot ot the cascaded all-pass filter output y can be seen in the Figure 2.14. In order to illustrate the all-pass functionality, we have also included the power spectral density estimate by Welch method to compare the input and output frequency responses. As can be seen from Figure 2.15 and 2.16, the only difference is in the amplitude since the gains we have given to the parameters are different than 1. Still the shape of the frequency response stays the same as expected.

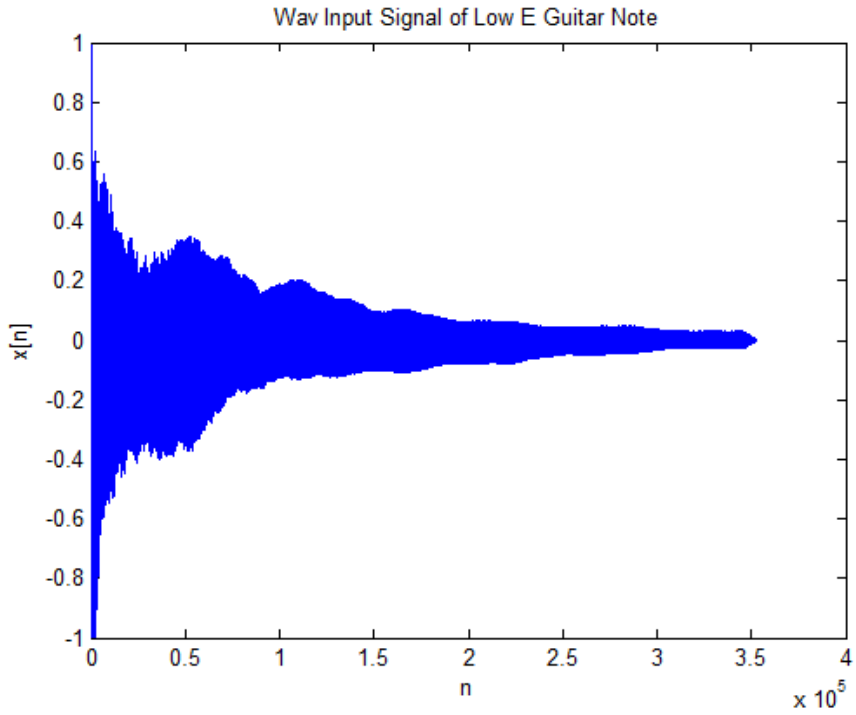

Figure 2.13: The time plot of the Original Input Signal x.

2.3.4 Reverberation Implementation

Overlap-Add method is used to model reverberation real-time. Frames are used to buffer a part of the input signal. Within the time limit of each frame, a new frame is

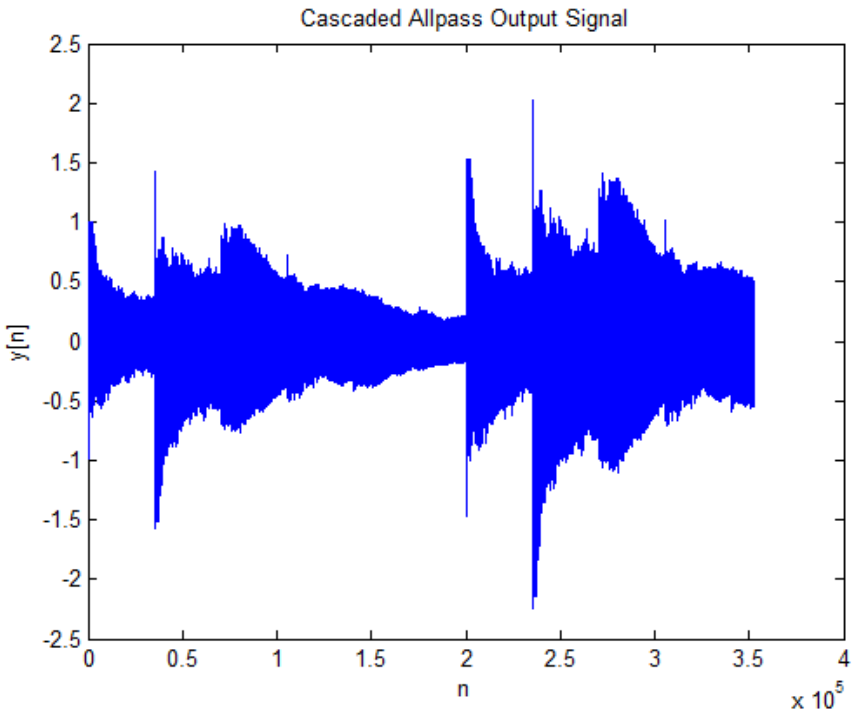

Figure 2.14: The time plot of the Output Signal y.

processed. The circular buffer implementation is designed using ConCurrentQueue structure in C#, which represents the circular buffer structure.

2.4 Wah-Wah Effect

Wah-wah effect, similar to its name, produces a variation of the input signal with the basic human voice "Wah". The wah-wah effect actually is a time varying band-pass filter where the center frequency, f_c, of the band-pass filter changes with time. This time-varying filtering with changing center frequency creates this unique effect called wah-wah. Wah-wah is generally used in rock lead guitar parts and offered to the guitar player as a foot controllable pedal. This allows the player to decide on the center frequency like a gas pedal in a car.

The structure of the time-varying band-pass filter of the wah-wah effect is the state variable filter. In wah-wah implementations, we preferred to use this structure because of the reason that it has many implementations in addition to wah-wah and has

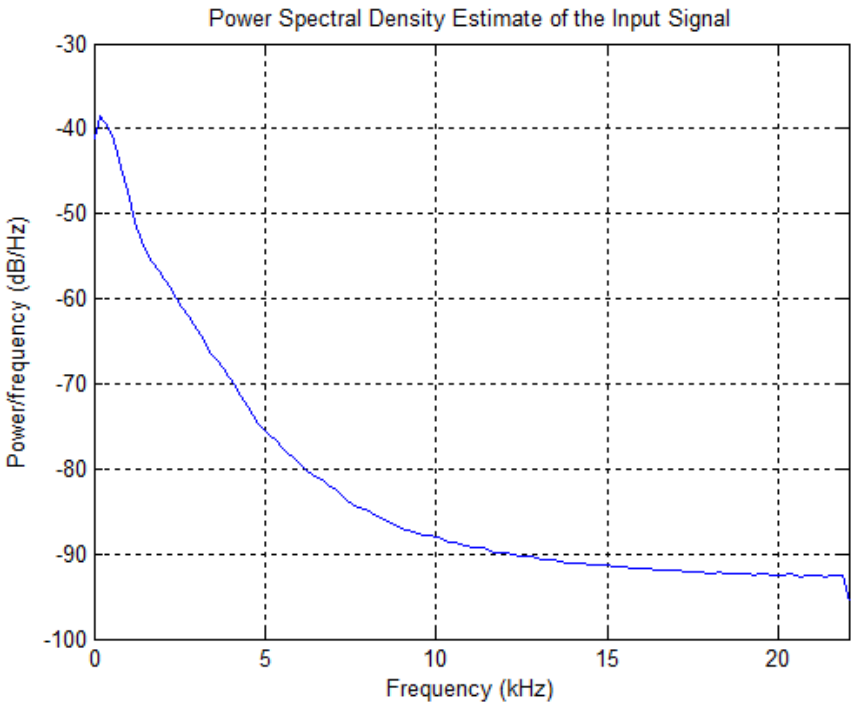

Figure 2.15: Power Spectral Density Estimate by Welch Method of the Original Input Signal x.

parameterized settings to obtain the desired center frequencies. The main advantage of the state variable filter is the ability to provide low-pass, band-pass and high-pass filter simultaneously. The state variable filter structure used in wah-wah effect is plotted in Figure 2.17.

Looking at Figure 2.17, we can see the input $x[n]$, low-pass output $y_l[n]$, band-pass output $y_b[n]$ and high-pass output $y_h[n]$. The outputs are created using the state variable filter where T represents a unit delay block. The equations for the outputs are given starting from Equation 2.52.

$$y_l[n] = F_1 y_b[n] + y_l[n-1] \tag{2.52}$$

$$y_b[n] = F_1 y_h[n] + y_b[n-1] \tag{2.53}$$

$$y_h[n] = x[n] - y_l[n-1] - Q_1 y_b[n-1] \tag{2.54}$$

where F_1 and Q_1 are related to the cut-off frequency f_c and damping parameter, d.

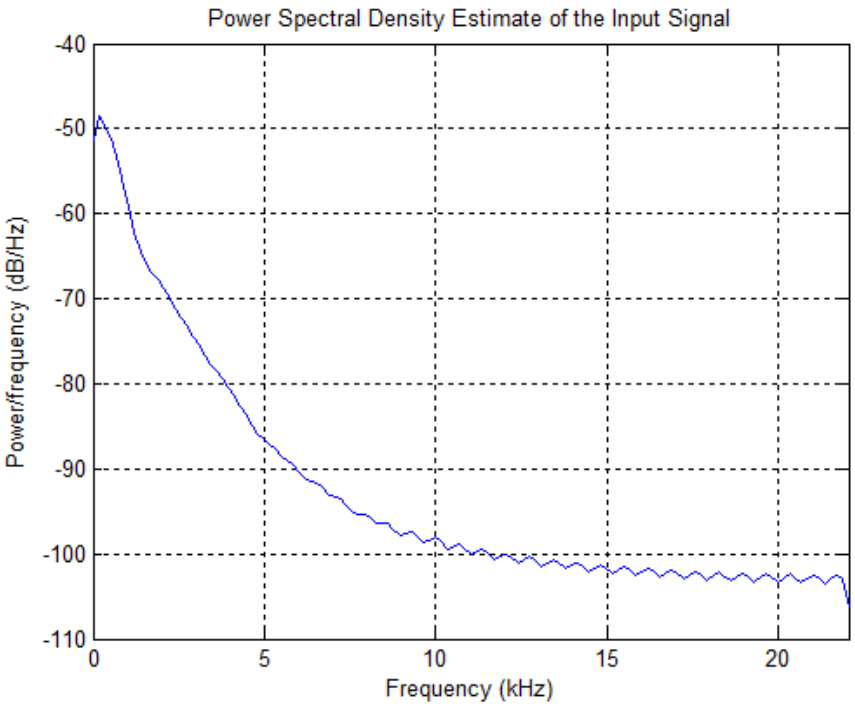

Figure 2.16: Power Spectral Density Estimate by Welch Method of the Output Signal y.

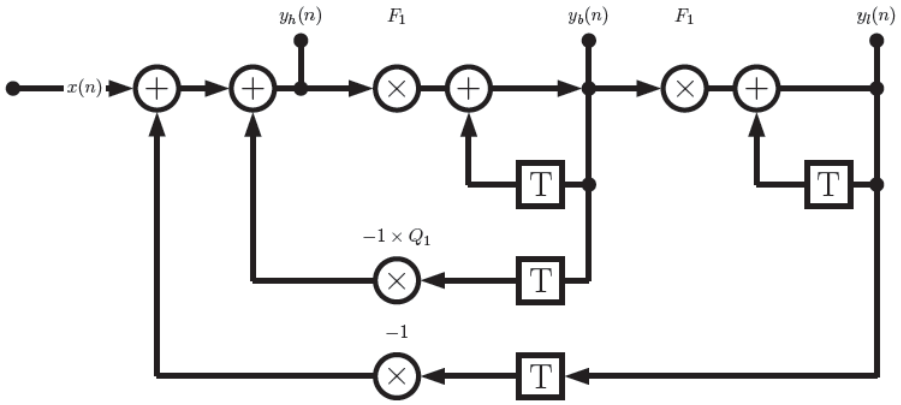

Figure 2.17: State Variable Filter Used In Wah-Wah Effect.

$$F_1 = 2sin(\pi f_c / f_s) \tag{2.55}$$

$$Q_1 = 2d \tag{2.56}$$

The highest frequency note produced by a 24th fret electric guitar (excluding har-

monics), is the 24th fret E note, which is approximately $1318Hz$. Generally, wah-wah effect is used with distortion which introduces and amplifies the harmonics beyond this range so we choose a band-pass cut off frequency, f_c ranging from 500 to 3000 Hz to include higher bandwidth. The algorithm we have implemented is called an Auto-Wah, where the band-pass cutoff frequency f_c ranges automatically in a triangular wave pattern during the real-time playing of the guitar. The parameters are $minf$, $delta$, $maxf$, desired minimum frequency, change in center frequency per sample and desired maximum frequency respectively. F_w is the triangular wave frequency parameter and effects the rate of change of the center frequency, f_c. Figure 2.18 shows the specific triangular band-pass frequency function for the parameters set to: $minf = 500$, $maxf = 3000$, $F_w = 2000$ and $delta = F_w/F_s = 2000/44100 = 0.0454$. F_w is chosen to be large to show the change over time easily, in normal wah-wah effects F_w is chosen to be much smaller.

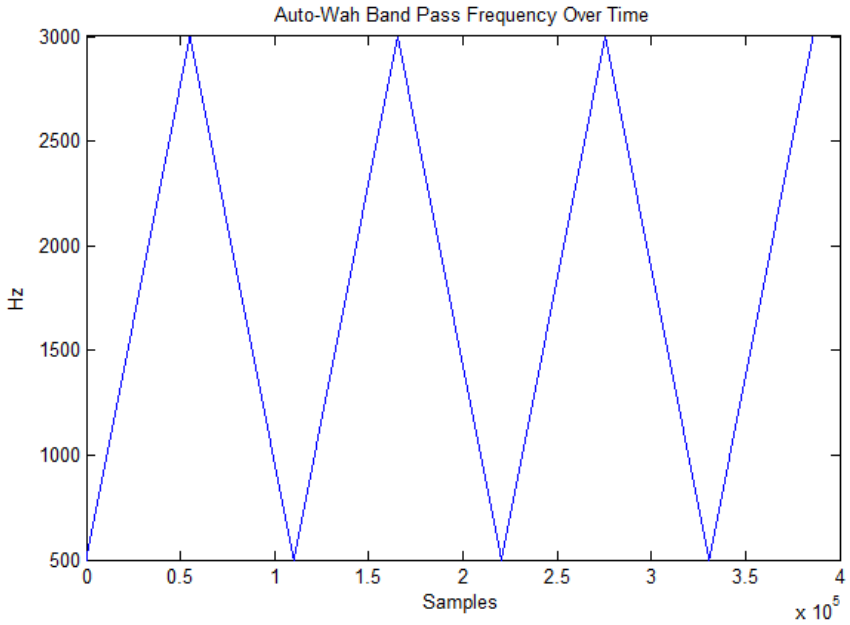

Figure 2.18: Auto-Wah Band Pass Frequency Over Time.

One of the most famous pedals in wah-wah guitar audio effects is the Jim Dunlop Original GCB95 model. This wah pedal originated from the 1966 Thomas organ model and basically is the oldest wah pedal ever designed. The foundations under the wah pedal is the time-varying band pass filtering, so the shape and range of the filter plays an important role in the output sound. Starting with MATLAB, we have im-

plemented this effect on various platforms and also in real-time using C# on windows applications and C/C++ on Texas Instruments eZDSP implementations. Let us first start with the basics of the effect using generic MATLAB function calls and plots. In MATLAB, we first start with the trimming of the triangular wave of band-pass frequency change to the length of the input signal. The other parameter sets are the damping factor, d which is initialized with $d = 0.05$ and F_1, the calculated f_c dependent parameter is recalculated in the loop for each changing value of f_c. F_c and Q_1 is calculated beforehand using the generic formulas given above. The next step is to put the parameters F_c and Q_1 into the for loop with the state variable filter equation to calculate each sample output with the corresponding F_1 value. The wah-wah output that we are interested is the $y_b[n]$ output, which is the band-pass output of the state variable filter. Finally, we normalize the output dividing each sample output value to the maximum of the outputs, to get an output ranging from -1 to 1. The input wav file and the output wavfile are shown as in Figure 2.19.

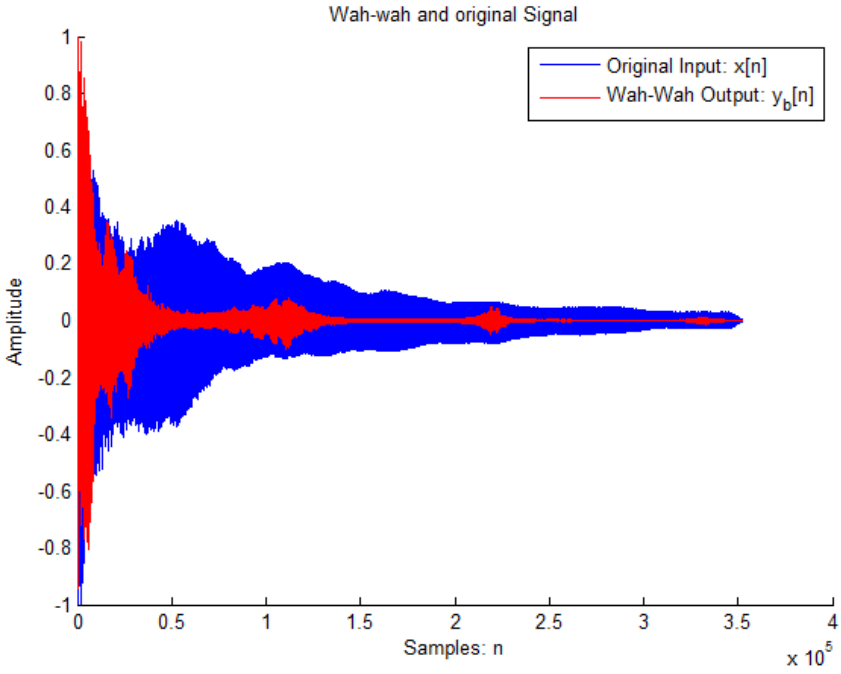

Figure 2.19: Comparison of the Original Input, $x[n]$, Auto-Wah Effect Output, $y[n]$.

CHAPTER 3

BASICS OF GUITAR DISTORTION: DIODE LIMITER AND PRE-AMPLIFIER

3.1 The Basic Building Block: Diode Limiter

Distortion and overdrive are two main highly nonlinear guitar audio effects. The signature guitar tones of famous guitar players are compared discussing their distortion and overdrive guitar effects. Through the emergence of blues, rock and metal, the implementations of distortion and overdrive guitar effects pedals have changed drastically. The main tone of a guitar player can be separately investigated into two separate signal channels: Clean Channel and Lead/Overdrive/Distortion Channel (different names can appear between manufacturers). This thesis presents a method to easily simulate these two channels, obtaining desired audio outputs.

Famous blues and rock players have unique amplifier, equalizer, rack and stomp box effects to characterize their sounds. The unique sounds of amplifiers, drives, equalizers and effects pedals give the guitar player the opportunity to form their unique sound and feel through music. Guitar audio effect distortion first used by over-driving valve tube amplifiers in a way that the amplifiers are used at their nonlinear points. In distortion and overdrive effects circuits a structure called diode limiter is commonly used as a building block. Diode limiter provides the two way clipping in its simplest way. Starting from the modelling of the diode, we will advance to how we can consider the effects of capacitors on the dynamic processing of the distortion. First we should begin with the famous Shockley diode equation as shown in Equation 3.1 [20]:

$$I_d = I_s.(e^{\frac{V_d}{n.V_T}} - 1) \tag{3.1}$$

$I_d =$ Diode Current (A)

$I_s =$ Diode Reverse Bias Saturation Current (A)

$V_d =$ Diode Voltage (V)

$V_T =$ Diode Thermal Voltage (V)

$n =$ Emission Coefficient / Quality Factor

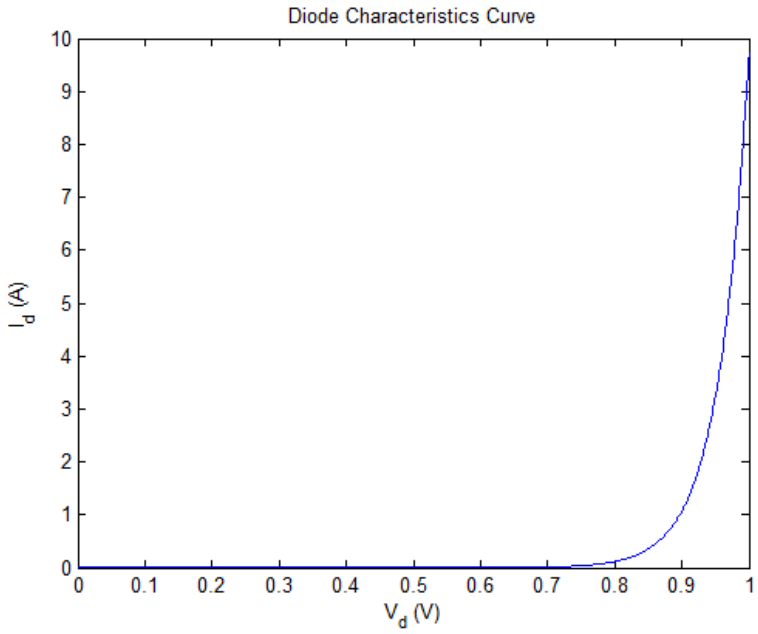

Figure 3.1: The I-V Characteristic of Shockley Diode Model.

The parameters used for the I-V characteristics for the diode model simulation are as follows: $I_s = 2.52e-9A$, $V_T = 45.3e-3V$ and $n = 1$ for ideal conditions [12].

3.2 The Importance of Pre-Amplifier

Whether you use a solid state (FET) amplifier or tube amplifier, the first stage of the signal amplification follows the route of a structure called pre-amplifier. The importance of the pre-amplifier is the amplification of the input signal to be later processed at the power amplifier and tone sections. A vacuum tube amplifier called

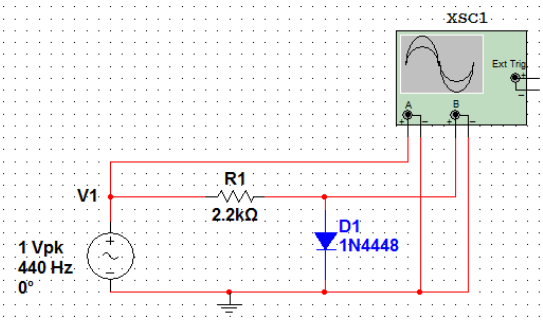

Figure 3.2: National Instruments Multisim 11.0 Schematic of a Simple Diode Circuit.

12AX7 is used extensively in pre-amplifier designs. Pre-amplifier circuit including 12AX7 can be modified with different resistance and capacitance values to obtain different guitar tones. Testing many of the offered designs from numerous guitar amplifier manufacturers, the following configuration from AX-84 Cooperative Tube Guitar Amp Project gave me the best results overall, both in listening and analyzing phases [17].

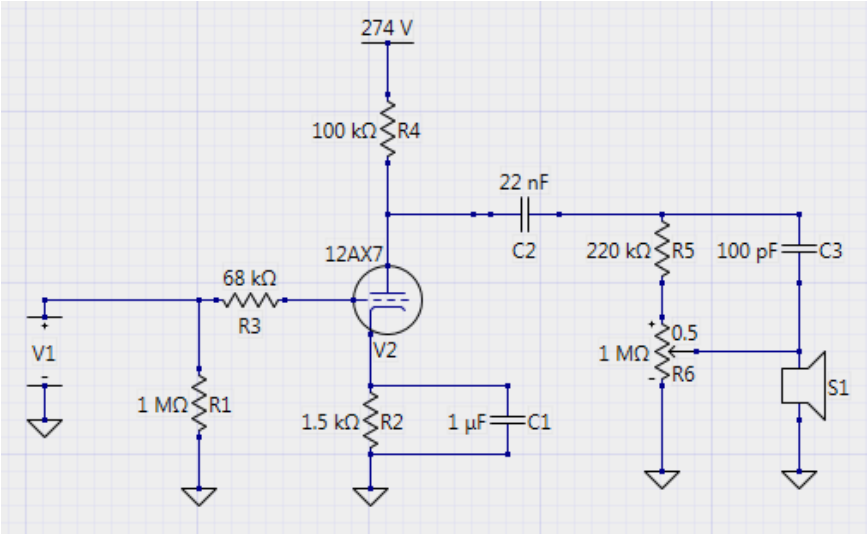

Figure 3.3: Generic 12AX7 Vacuum Tube Pre-Amplifier Schematic.

The design includes firstly the input voltage divider feeding the gate of the 12AX7 Vacuum Tube Amplifier, amplified by the 12AX7 amplifier and then DC block capacitance applies tone control over the output signal. There are couple of models when analyzing and modelling vacuum tube amplifiers: Koren's Tube Model [13], Child-Langmuir Model and recently introduced physically-informed real-time capa-

ble Vacuum Tube Model of Dempwolf-Zölzer Model [6]. We will briefly look into each model and decide on our simulation method.

In DAFx-11 Conference, Dempwolf and Zölzer presented a physically-motivated 12AX7 triode model, which is also real-time capable due to its low computational complexity [6]. The model starts with stating the cathode current and that the exponent of the generic equation can be different from 1.5 as stated in Koren's Model and other main vacuum tube models. Triode characteristics are dependent on a number of parameters: V_a, anode voltage, V_g, grid voltage and I_k, I_a and I_g, cathode, anode and grid currents respectively. Since grid and anode currents flow both to the cathode, the equation $I_g + I_a = I_k$ follows naturally. Another pre-requisite for easier analysis is to model the three terminal triode as a two-terminal one introducing the effective voltage which is basically the amalgamation of grid and anode voltages, V_g and V_a by the amplification factor, μ. The effective voltage is stated mathematically as in Equation 3.2.

$$V_{eff} = (V_g + \frac{1}{mu}.V_a) \tag{3.2}$$

Following the effective voltage equation, the triode model can be simplified as in the diode model using effective voltage as in Equation 3.3.

$$I_k = G.x(V_{eff})^{\frac{3}{2}} \tag{3.3}$$

The cathode current equation is applicable under normal operating conditions ($V_g = -2V$ and $V_a = 300V$) [6].The ideal tube models ignore the grid current($I_g = 0$) since it just has an effect on the capacitive distortion models. However, in some cases realistic approaches tend to take I_g into consideration. Practically, deviations from the ideal models occur at 20 percent, such that even the tubes manufactured by the same company may have differences in characteristics. The cathode current is modelled using Equation 3.4. The parameter γ represents these changing manufacturing differences. Dempwolf and Zölzer also suggest using a parameterized cathode current as in Equation 3.4.

$$I_k = G.(V_{eff})^{\gamma},\ V_{eff} > 0,\ V_{eff} < 0\ is\ not\ considered \tag{3.4}$$

The grid and cathode also constitutes a relation between anode and cathode in a two-terminal tube diode, so we can consider the same physical background formulization for the grid current. Upon studies, it is found that the anode voltage, V_a has little or no effect on the grid current, so we can model grid current as a nonlinear function of grid voltage, V_g,the grid perveance, G_g and exponential parameter ξ.

$$I_g = f(V_g) = G_g.{V_g}^{\xi} \tag{3.5}$$

Dempwolf and Zölzer suggested using smooth functions for the triode model, eliminating discontinuities coming from piecewise functions [6]. They offered to use logarithmic functions as the $h(x)$ function below rather using piecewise functions.

$$h(x) = log(1 + e^{C.x}).\frac{1}{C} \tag{3.6}$$

Finalizing the equations requires to replace the smooth versions the previous cathode and grid currents with the new logarithmic smoothing function as in equations 3.7 and 3.8.

$$I_k = G.(log(1 + exp(C.(\frac{1}{\mu}.V_a + V_g))).\frac{1}{C})^{\gamma} \tag{3.7}$$

$$I_g = G_g.(log(1 + exp(C_g.V_g)).\frac{1}{C_g})^{\xi} + I_{g0} \tag{3.8}$$

Subtracting I_g from I_k gives the desired anode current I_a as in Equation 3.9.

$$I_a = I_k - I_g \tag{3.9}$$

As a result, we have three equations at hand and two sets of parameters: (G, C, μ, γ) for cathode current and (G_g, C_g, ξ, I_{g0}) for grid current. Dempwolf-Zölzer [6], used a $MATLAB$ Curve Fitting Tool to find the suitable parameters for their proposed model. The results are promising such that they have nearly exact match with the actual tube measurements. The table for the parameters used to model 3 different tubes are given in Figure 3.4. [6].

	RSD-1	RSD-2	EHX-1
G	2.242E-3	2.173E-3	1.371E-3
μ	103.2	100.2	86.9
γ	1.26	1.28	1.349
C	3.40	3.19	4.56
G_g	6.177E-4	5.911E-4	3.263E-4
ξ	1.314	1.358	1.156
C_g	9.901	11.76	11.99
I_{g0}	8.025E-8	4.527E-8	3.917 E-8

Figure 3.4: The Model Parameters Found by Curve Fitting Toolbox of MATLAB.

CHAPTER 4

WAVE DIGITAL FILTER MODELLING APPROACH OF GUITAR DISTORTION

4.1 Wave Digital Filter Modeling of Guitar Distortion

Wave Digital Filters are developed by Alfred Fettweis in 1969 to model classical filters using port equivalents of actual circuitry present in the designed schematics [7]. These filters are preferably in lattice or ladder configurations. In order to understand the design a wave digital filter model, a sufficient knowledge of network theory is required. This section will continue by first introducing a two-port network analysis, then the applicability of wave digital filters to the real-time implementations of distortion and overdrive guitar audio effects will be discussed.

4.1.1 Two-Port Networks

A general two-port network is defined by its input, output voltages and currents as given by parameters: V_1, I_1 for input V_2, I_2 for output as shown in Figure 4.1.

The essential rule for a two-port network is to have the current going into one terminal of a port same as the current leaving the other terminal of the same port [1]. With this rule satisfied, a two-port network can be described by parameter matrix. Z parameters, y parameters and h parameters are given respectively as in Equations 4.1, 4.2 and 4.3.

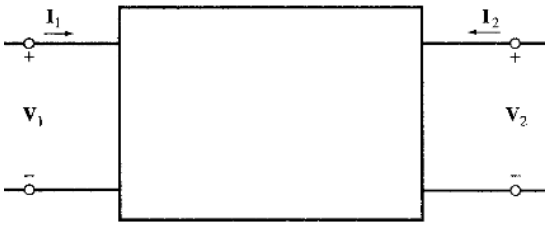

Figure 4.1: Simplest Two-Port Network Configuration.

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \tag{4.1}$$

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \tag{4.2}$$

$$\begin{bmatrix} V_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} I_1 \\ V_2 \end{bmatrix} \tag{4.3}$$

Using two-port network solutions, wave digital filters are used to model each and every component of a previously known circuit. It is an alternative way to directly solving the state space solutions of a nonlinear circuit.

4.1.2 Application of Wave Digital Filter Theory to Nonlinear Audio Circuits

Wave Digital Filter Theory focuses on how we can represent digital filters in wave digital domain. We first establish our reference domain then convert each one and two port circuit block into wave digital domain. In order to do that we have to give reference to Alfred Fettweis' Wave Digital Filters: Theory and Practice [1985] Invited Paper [7]. In this paper Fettweis defines the wave digital filter as a representation of a class of digital filters that are closely related to classical filter networks preferably lossless filters inserted between resistive terminations. For each and every wave digital filter, there corresponds a reference filter from which it is derived.

In 2006, Matti Karjalainen and Jyri Pakarinen published a paper called "Wave Digital Simulation of a Vacuum-Tube" Amplifier as an alternative method for the nonlinear state-space or SPICE circuit simulations of vacuum tube amplifiers [11]. The wave digital modelling of nonlinear audio circuits allow the designer to model each and individual block and then simulate them together to form much complex circuits [4]. Though the wave digital filters allow block modelling of each component, it is not a standardized process to model each and every block and make the connection algorithms [15]. In order for a wave digital filter to be formed, the inside circuitry must be known prior to modelling the input output filter, so that wave digital filters can not be used for our purposes of nonlinear system identification. Since wave digital filtering is an alternative way of simulating distortion and overdrive circuits, this chapter is included to introduce alternative methods to the reader. The implementations of distortion modelling with wave digital filters are not included because of these reasons.

CHAPTER 5

SYSTEM IDENTIFICATION APPROACH TO DISTORTION MODELLING

5.1 System Identification Approach to Distortion Modelling

System identification is used to model dynamic systems from collected data [8]. A system can be characterized by its input output pair. External factors such as noise and unobservable data effect the system unintentionally [3]. Distortion and overdrive nonlinear effects are similar highly nonlinear dynamic systems. For the modelling of these effects, black box system identification approach can be selected. Black box modelling of a system includes forming a system structure only with input and output pair analysis [14]. No additional information about the inside circuitry is known in black box modelling.

The most common approach to identify a system is to first model that system linearly. For that purposes, we can use LCCDEs known also as the discrete time difference equation shown in the following equation:

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = b_1 u(t-1) + ... + b_m u(t-m) \qquad (5.1)$$

One of the best ways to understand this equation is to write it in the form of output predictions with previous information as follows:

$$y(t) = -a_1 y(t-1) - ... - a_n y(t-n) + b_1 u(t-1) + ... + b_m u(t-m) \quad (5.2)$$

In order to shorten the notation into the matrix form we wanted to denote the parameters and the shifted responses as follows:

$$\theta = [a_1 ... a_n \; b_1 ... b_m]^T \quad (5.3)$$

$$\varphi = [-y(t-1) - ... - y(t-n) \; u(t-1) + ... + u(t-m)]^T \quad (5.4)$$

Resulting into writing the difference equation in the matrix form as follows:

$$y(t) = \varphi^T(t)\theta \quad (5.5)$$

Where such an estimation depends on the parameter set θ, the prediction equation should be written as follows:

$$\hat{y}(t|\theta) = \varphi^T(t)\theta \quad (5.6)$$

Linear estimation models for distortion and overdrive are not used since these effects are highly nonlinear. Static nonlinear modelling approaches are used. These models try to simulate hard and soft clipping static functions [21]. The introduction to distortion modelling is done by implementing hard clipping to an audio input[21]. This is the static distortion model found in most applications. Distortion guitar audio effect is an intentional harmonics exaggeration to obtain more sustain, boldness and compression, eventually leading to blues and rock sound. Mathematically shown in equations 5.7 and 5.8, is the hard clipping and an extension to it, compressed clipping. Compressed clipping additionally has amplitude degradation if the input signal is above the specific threshold, a. This will in turn results into harmonics having much more power in the output.

$$y_{hc}[x[n]] = \begin{cases} x[n], & \text{if } -th \leq x[n] \leq th \\ -th, & \text{if } x[n] < -th \\ th, & \text{if } x[n] > th \end{cases} \tag{5.7}$$

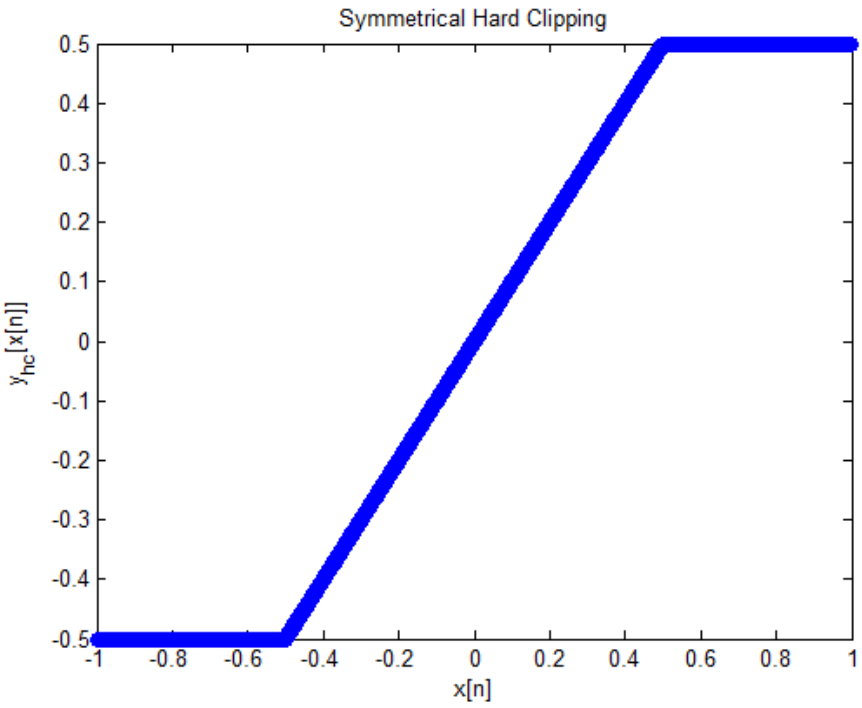

Figure 5.1: Symmetrical Hard Clipping.

Upon observing the output signals of various distortion amplifiers and elementary input/output models, we came up with introducing a hard compression to the hard clipping function. This application exaggerates the odd harmonics created by the clipping function, resulting into compression of the input values behind a threshold level, a. The mathematical models are shown in the output equation 5.8 of $y_{cc}[x[n]]$ with the parameters a and b. In addition to output function, $MATLAB$ plot and power spectral density of the compressed clipping output signal are shown in Figures 5.3 and 5.4.

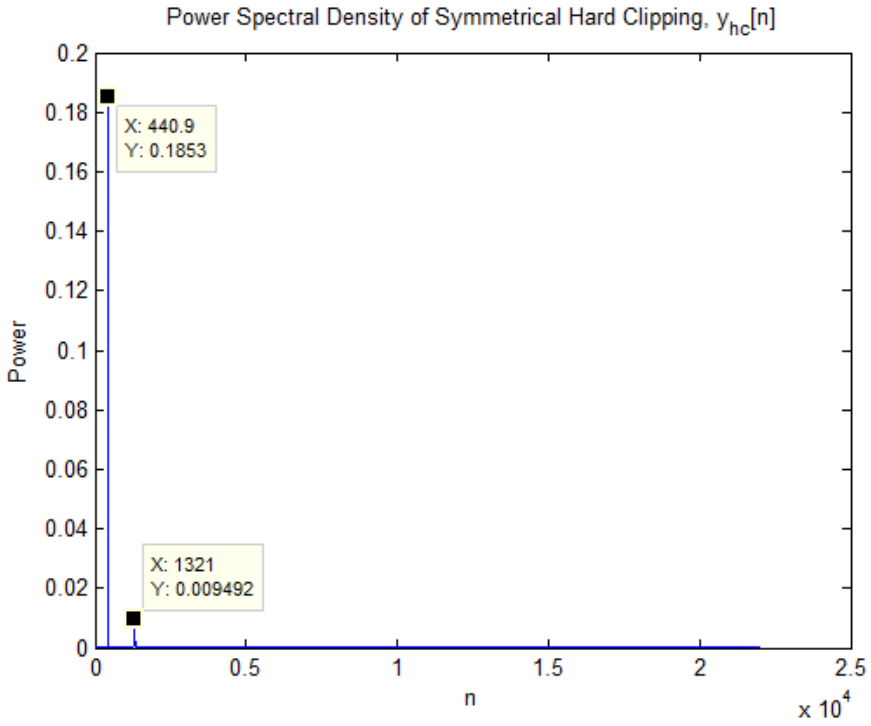

Figure 5.2: Power Spectral Density of the Symmetrical Hard Clipping Output to 440 Hz Sinusoidal Tone.

$$y_{cc}[x[n]] = \begin{cases} x[n]/a, & \text{if } -a \leq x[n] \leq a \\ \frac{1-b}{a-1}.x[n] + \frac{a.b-1}{a-1}, & \text{if } x[n] > a \\ \frac{1-b}{a-1}.x[n] - \frac{a.b-1}{a-1}, & \text{if } x[n] < -a \end{cases} \tag{5.8}$$

The results of this comparison can be seen by looking at the power spectral densities of each output function, Figures 5.2 and 5.4. Compressed Clipping method enhances the odd harmonics higher than the static hard clipping. This method is used generally for the elementary distortion models with compression. While listening to the results, in case of compressed clipping we hear the enhanced odd harmonic inside the signal. This gives the output signal additional harshness that we want in a good distortion model.

The next implementation we are going to present is the smooth static nonlinearity. This method is mentioned in DAfx [21], to basically simulate and model distortion and fuzz in general. The difference is that the input and output relational mapping

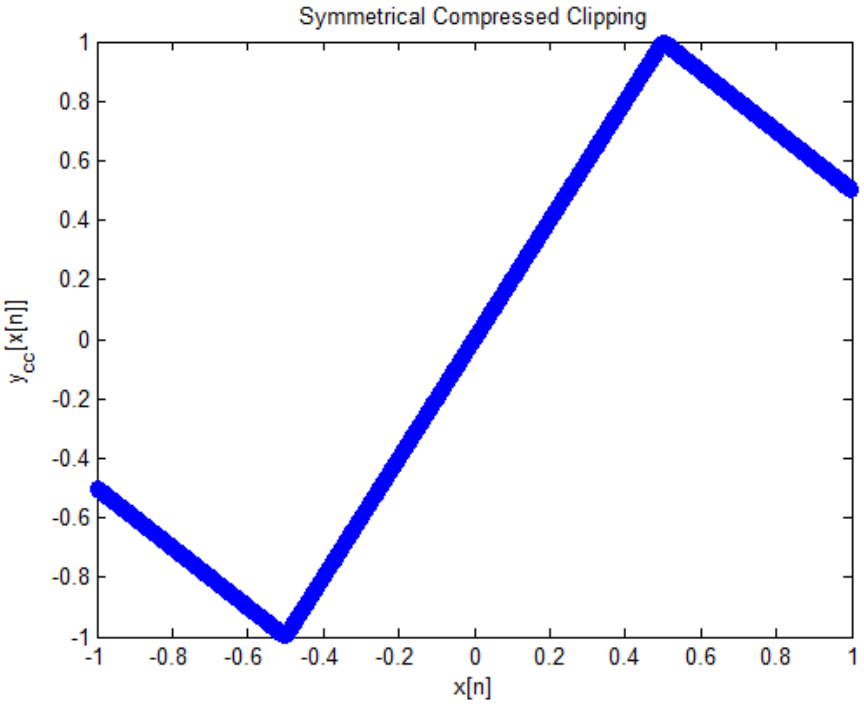

Figure 5.3: Symmetrical Compressed Clipping.

function is a smooth function compared to hard clipping and compressed clipping. The mathematical model is shown in Equation 5.9.

$$f(x) = sgn(x)(1 - e^{-|x|}) \tag{5.9}$$

What this smooth static nonlinearity does is that given input values x, it generates output values y for each sample input. The input output relationship of the nonlinear static distortion function is shown in Figure 5.5.

The introduction of smoothness makes the distortion output much more organic to the ear than the previous models. In the previous models, we can sense some digitization and inorganic sounds created by the harsh clipping in the output in the previous implementations.

Continuing with a more generalized expression for nonlinear black-box models for nonlinear system estimation, we can state the generalized nonlinear system in Equation 5.10.

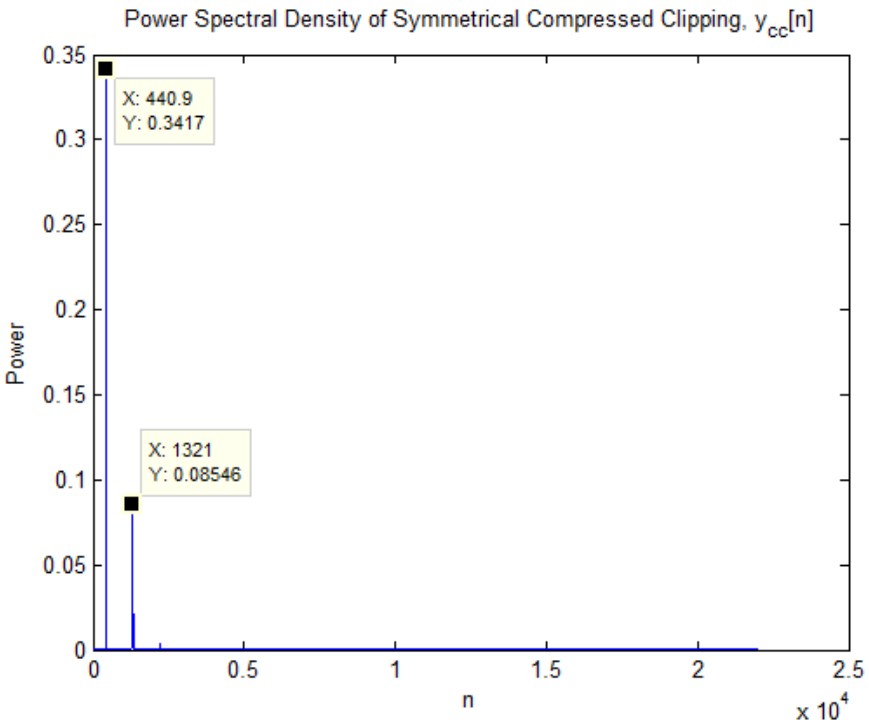

Figure 5.4: Power Spectral Density of the Symmetrical Compressed Clipping Output to 440 Hz Sinusoidal Tone.

$$\hat{y}(t|\theta) = g(Z^{t-1}, \theta) \tag{5.10}$$

In this general model, explicit time dependence is omitted and Z^{t-1} is the regression parameter where past input and output values are stored and θ is the nonlinear mapping parameter from the regressor space to the output space.

The regressor model can be expanded and generalized using the following equations:

$$g(Z^{t-1}, \theta) = g(\varphi(t), \theta) \tag{5.11}$$

Generalized equations give us an insight about the estimation methods. In order to specialize our estimation method, we will continue with the neural network estimation method called ANFIS (Adaptive-Network-Based Fuzzy Inference System) in the following chapter. This method uses fuzzy rules to train, learn and model nonlinear functions and is applicable to model distortion and overdrive.

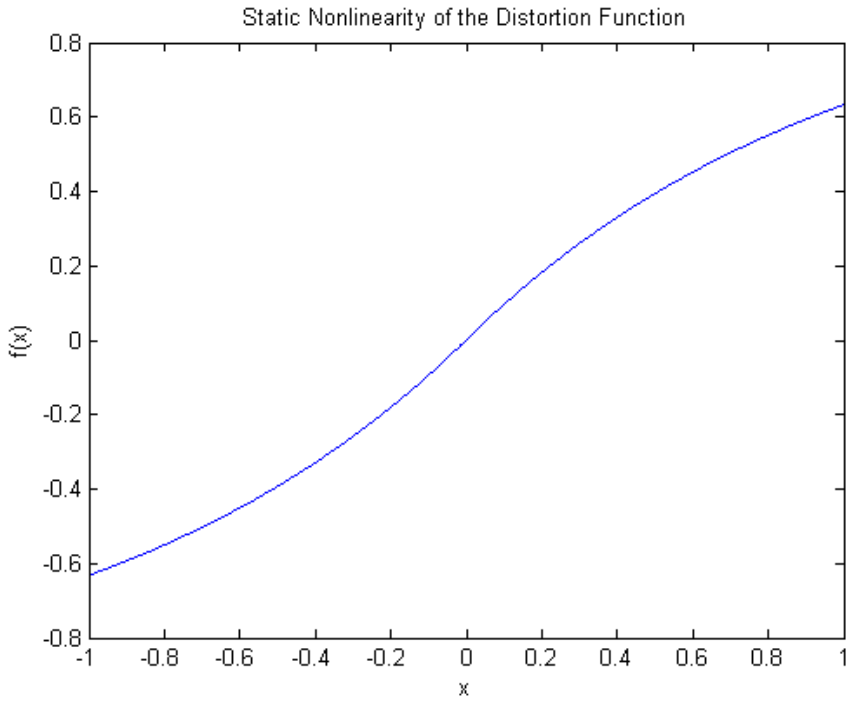

Figure 5.5: Static Nonlinearity of the Distortion Function.

CHAPTER 6

ANFIS: ADAPTIVE-NETWORK-BASED FUZZY INFERENCE SYSTEM

6.1 ANFIS: Adaptive-Network-Based Fuzzy Inference System

Neural networks are used as a system identification method with training, learning and modelling steps. ANFIS is one of the applicable neural network system identification methods to model highly nonlinear dynamic systems such as distortion and overdrive. Introduced in 1993 by Jyh-Shing Roger Jang [10], ANFIS nonlinear modelling approach introduced an adaptive node based neural network system identification. This method involves training a set of input and output values and learning the relation between the pair. Considering highly nonlinear audio effects, distortion and overdrive, it is applicable to use a black box neural network system identification method. At this point ANFIS provides a cost-efficient and reliable way of modelling the black box between input and output pairs of distortion and overdrive guitar audio effects.

ANFIS method is introduced to standardize the existing methods which transfers human knowledge and decision making into a rule based mathematical method and training data. Another point is the need for effective methods for tuning the membership functions (MF) to minimize output error, hence maximizing performance index. Firstly this chapter introduces what this method is, and then the chapter continues with the adaptation of this algorithm to our case of modelling overdrive and distortion.

In ANFIS structure, the fuzzy logic is implemented using if-then structure. This method is basically a formulated mapping of the inputs to outputs with the introduction of parameters. Fuzzy if-then rules are used extensively in system identification, where a dynamic system can be represented by decision making rules. The ANFIS structure used in modelling distortion or overdrive effect can be shown using layer representations. In this specific ANFIS structure, we have inputs ($x[n]$ and $x[n-1]$) and obtained a single output ($y[n]$). For the training, inputs $x_k[n]$ and $x_k[n-1]$ are the current guitar output signal array and single delayed guitar output signal array of length k. The training output array of length k, $y_k[n]$ is obtained either by the ideal nonlinear state space circuit space solution of the distortion/overdrive guitar effect circuit or real overdrive/distortion guitar effect pedal output for the input $x_k[n]$.

The architecture and learning rule of ANFIS relies on the definition of an adaptive network. A multilayer feed-forward adaptive network has nodes where each node has its own function processing its incoming input signal into output. Each node is connected to one another via links where the weights are mentioned in their specific functions. Square nodes are adaptive nodes which have parameters. Circle nodes are fixed nodes which do not have parameters. Parameters of an adaptive network is a set where each parameter of every node is included. The procedure of updating the parameters is called training and learning. To illustrate such a network, the ANFIS structure with two inputs, single output and N rules is shown in Figure 6.1.

First layer consists of adaptive nodes, which are represented as square nodes. The node functions are the fuzzy membership functions, which generally are generalized or gaussian bell functions. In this structure, we used generalized bell-shaped function for fuzzy membership functions for both A and B fuzzification members. The node outputs of the first layer are represented as O_i^1. In our ANFIS implementations, number of rules is selected as 10 ($N = 10$).

$$O_i^1 = \mu_{A_i}(x[n]) \qquad for\ i = 1, 2, ..., 10. \tag{6.1}$$

$$O_i^1 = \mu_{B_i}(x[n-1]) \qquad for\ i = 11, 12, ..., 20. \tag{6.2}$$

This special function $\mu(x)$ is called the membership function of A_i or B_i, and it specifies the degree to which the given input $x[n]$ or $x[n-1]$ satisfies the quantifier A_i or B_i. In this ANFIS application, $\mu_{A_i}(x)$ is chosen to be the generalized bell-

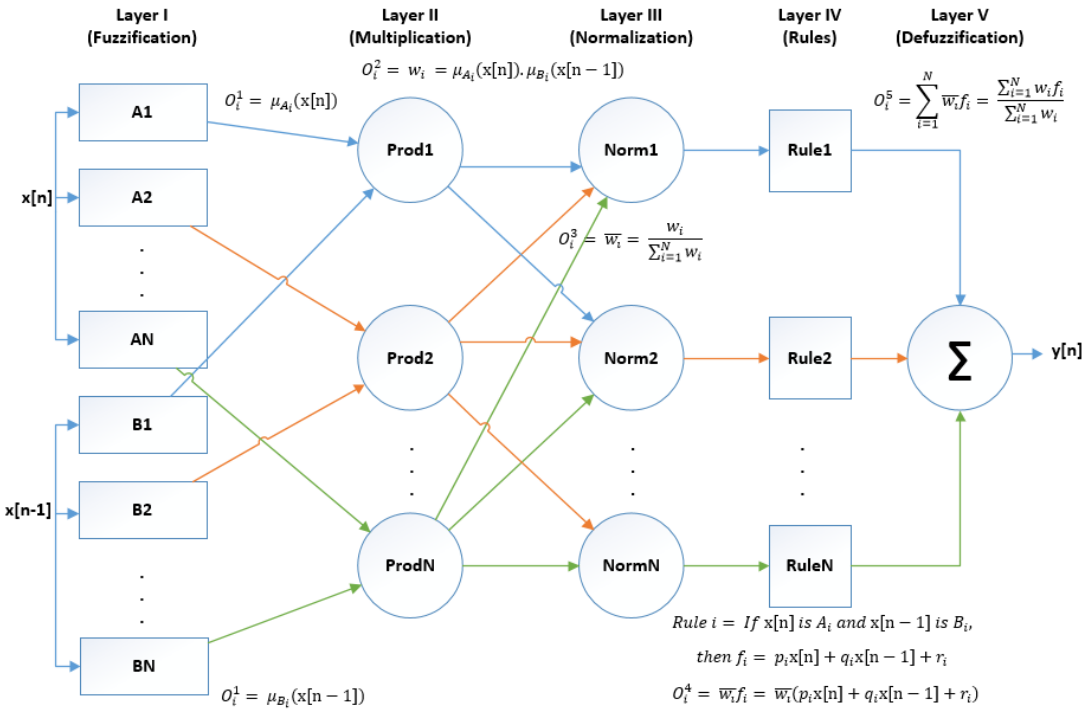

Figure 6.1: ANFIS structure with 2 inputs $x[n]$ and $x[n-1]$, and single output $y[n]$ [10].

shaped distribution with maximum 1 and minimum 0 as given in the following two bell shaped formulas.

$$\mu_{A_i}(x[n]) = \frac{1}{1 + \left[\left(\frac{x[n]-c_i}{a_i}\right)^2\right]^{b_i}}. \tag{6.3}$$

$$\mu_{B_i}(x[n-1]) = \frac{1}{1 + \left[\left(\frac{x[n-1]-d_i}{e_i}\right)^2\right]^{f_i}}. \tag{6.4}$$

In this two bell-shaped formulas $[a_i\ b_i\ c_i]$ and $[d_i\ e_i\ f_i]$ are the parameter sets, also called $premise\ parameters$. Changing these parameters results into different bell-shaped member functions, also different system models. As MATLAB provides, there are different types of membership functions named by their distribution shapes such as: trimf, trapmf, gbellmf, gaussmf, gauss2mf, pimf, dsigmf, psigmf. This layer is called fuzzification layer since the inputs are mapped with corresponding fuzzy

membership functions.

Second layer is the multiplication layer, where the fixed circle nodes make simple multiplications of the incoming inputs. The outputs of this layer are called the fire strengths of the rules. The firing strengths of each rule are given in Equation 6.5.

$$O_i^2 = w_i = \mu_{A_i}(x[n]).\mu_{B_i}(x[n-1]). \qquad for\ i = 1, 2, ..., 10. \tag{6.5}$$

In the third layer, the firing strengths of each previous node outputs are normalized. This layer is called the normalization layer. These nodes are also fixed circle nodes as the second layer. The normalization function for the third layer is given in Equation 6.6.

$$O_i^3 = \bar{w_i} = \frac{w_i}{\sum_{i=1}^{10} w_i}. \qquad for\ i = 1, 2, ..., 10. \tag{6.6}$$

Fourth layer consists of adaptive square nodes, which multiplies the normalized firing strengths with the parameterized rules. This layer is called rules. These rules are coming from the Sugeno fuzzy rule model. Sugeno fuzzy rule parameters are called $consequent\ parameters$, and denoted with p_i, q_i and r_i. Sugeno fuzzy rules have if-then structure as shown in Equations 6.7 to 6.13. Each normalized firing strength, $\bar{w_i}$, is then multiplied with the corresponding rule parameterized polynomial to obtain the fourth layer outputs, O_i^4. The formulization is given in Equation 6.14.

$$Rule\ 1 : If\ x[n]\ is\ A_1\ and\ x[n-1]\ is\ B_1, \tag{6.7}$$

$$then\ f_1\ =\ p_1x[n] + q_1x[n-1] + r_1. \tag{6.8}$$

$$Rule\ 2 : If\ x[n]\ is\ A_2\ and\ x[n-1]\ is\ B_2, \tag{6.9}$$

$$then\ f_2\ =\ p_2x[n] + q_2x[n-1] + r_2. \tag{6.10}$$

$$... \tag{6.11}$$

$$Rule\ 10 : If\ x[n]\ is\ A_{10}\ and\ x[n-1]\ is\ B_{10}, \tag{6.12}$$

$$then\ f_{10}\ =\ p_{10}x[n] + q_{10}x[n-1] + r_{10}. \tag{6.13}$$

$$O_i^4 = \bar{w}_i f_i = \bar{w}_i(p_i x[n] + q_i x[n-1] + r_i). \tag{6.14}$$

The fifth and last layer is the defuzzification layer, which sums up the incoming node outputs. This summation output is also called the crisp output, $y[n]$ in a single output ANFIS structure. The output of the ANFIS structure, $y[n]$ is given in Equation 6.15.

$$y[n] = O_i^5 = \sum_{i=1}^{10} \bar{w}_i f_i = \frac{\sum_{i=1}^{10} w_i f_i}{\sum_{i=1}^{10} w_i}. \tag{6.15}$$

In the final output node equation, we have obtained an output depending on both premise parameters (a_i, b_i, c_i, d_i, e_i, f_i) and consequent parameters (p_i, q_i, r_i). These parameters are coming from two adaptive layers of the ANFIS structure, Layer I (Fuzzification) and Layer IV (Rules) respectively. Using different learning algorithms, ANFIS method tries to optimize both premise and consequent parameters using the training input and output data.

ANFIS method can be summarized with the steps below:

1. Compare the input variables with the membership functions on the premise part to obtain the membership values (or compatibility measures) of each linguistic label. (This step is often called Fuzzification.)

2. Combine (through a specific T-norm operator, usually multiplication or min.) the membership values on the premise part to get firing strength (weight) of each rule.

3. Generate the qualified consequent (either fuzzy or crisp) of each rule depending on the firing strength.

4. Aggregate the qualified consequents to produce a crisp output. (This step is called defuzzification.)

6.1.1 Hybrid Learning Algorithm

The learning method that we use in ANFIS implementation is the hybrid learning algorithm. The output of the ANFIS structure, $y[n]$ can be written as a linear combination of consequent parameters (p_i, q_i, r_i), assuming that the premise parameters (a_i, b_i, c_i, d_i, e_i, f_i) are fixed.

$$y[n] = \bar{w_1} f_1 + \bar{w_2} f_2 + ... + \bar{w_{10}} f_{10}. \tag{6.16}$$

$$= \bar{w_1}(p_1 x[n] + q_1 x[n-1] + r_1) + \bar{w_2}(p_2 x[n] + q_2 x[n-1] + r_2) + ... \tag{6.17}$$

$$+ \bar{w_{10}}(p_{10} x[n] + q_{10} x[n-1] + r_{10}). \tag{6.18}$$

$$= \sum_{i=1}^{10} (\bar{w_i} x[n]) p_i + (\bar{w_i} x[n-1]) q_i + (\bar{w_i}) r_i \tag{6.19}$$

Hybrid Learning Algorithm consists of both forward and backward pass through the adaptive node parameter updates. In the forward pass of the hybrid learning algorithm, the consequent parameters in Equation 6.19 are estimated by the least squares method. In the backward pass, the error rates propagate backward and the premise parameters are updated by the gradient descent [10]. The identified consequent parameters are optimal under the condition that the premise parameters are fixed. It is shown that the hybrid approach is much faster than the gradient descent only method. There are four proposed methods to update the premise and consequent parameters [10]:

1. Gradient Descent Only: All parameters are updated by gradient descent.
2. Gradient Descent and One Pass of LSE: The LSE is applied only once at the very beginning to get the initial values of the consequent parameters and then the gradient descent takes over to update all parameters.
3. Gradient Descent and LSE: This is the proposed hybrid learning rule, which applies LSE in the forward pass and applies gradient descent in the backward pass [10].
4. Sequential (Approximate) LSE Only: The ANFIS is linearized w.r.t. the premise parameters and the extended Kalman filter algorithm is employed to update all parameters. This is proposed in the neural network literature before ANFIS architecture.

Mathematical formulation of the node function requires the knowledge of its previous outputs and the parameters used in the functions. For each layer of the adaptive network there exist a number of nodes which can have different functions. We can denote this function as O_i^k, which denotes the output of the ith node at the kth layer. Assuming that the adaptive network has L layers, we can define the function as follows.

$$O_i^k : O_i^k(O_1^{k-1}, ...O_{\#(k-1)}^{k-1}, a, b, c, ...) \tag{6.20}$$

The node output function depends on the node output functions of the previous node outputs and its own parameters denoted by a, b and c and more. These outputs are compared with the training data T to find the error measure, which in this case is the cumulative squared error as depicted in the following formula.

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2 \tag{6.21}$$

Summing up each error value for each training data entry p, where p ranges between 1 and P, we can obtain the total error measure, denoted by E.

$$E = \sum_{p=1}^{P} E_p \tag{6.22}$$

The method for finding the optimal parameters (learning procedure) follows the gradient descent in E over the parameter space, calculating the error rate for pth training data. The formulation for the error rate for the output node at ith node of the Lth layer can be shown as below.

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L) \tag{6.23}$$

Each internal ith node of kth layer has its own error rate obtained by the chain rule of derivation, where k is between 1 and $L-1$.

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k} \tag{6.24}$$

This gives us the gist of the ability to calculate the error rate of an internal node as a linear combination of the error rates in the next layer. For all k k between 1 and L and i between 1 and $\#k$, we can find the specific error rate by using the previous two formulas.

Introducing an arbitrary parameter α of the given adaptive network and output related nodes O^* whose outputs depend on α, we can define the error output according to parameter α as follows. In the following function S is the set of nodes whose outputs depend on α.

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \tag{6.25}$$

For the gradient descent algorithm the update formula for the parameter α can be shown as below.

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{6.26}$$

where η is the learning rate, k is the step size and can be manipulated as:

$$\eta = \frac{k}{\sqrt{\sum_\alpha (\frac{\partial E}{\partial \alpha})^2}} \tag{6.27}$$

The speed of convergence, as stated in the original paper, depends on step size parameter k. The speed also depends on the length of each gradient transition in the

parameter space. In the next sub-chapter, I will talk about the two methods for learning algorithms: Batch (Off-Line) Learning and Pattern (On-Line) Learning which are both under Hybrid Learning Rules.

6.1.1.1 Batch (Off-Line) Learning - Hybrid Learning Technique

This method incorporates both the gradient descent algorithm and the least squares estimate (LSE) to identify parameters, which makes it more robust than just using the gradient method. To explain this method of solution, we have to first look at the main input and output relations. Consider an adaptive network having set of input variables I and one output and output function F having parameters from the set S as follows [10]:

$$output = F(I, S) \tag{6.28}$$

Using linearity, assuming that there happens to be a function H, which makes $H \circ F$ linear with some of the elements of S, this makes it possible to use least squares method identifying parameters. In mathematical representation, if we can show that the parameter set S can be decomposed into two sets as follows.

$$S = S_1 \oplus S_2 \tag{6.29}$$

The parameter sets can be summarized as follows:

$$S = set\ of\ total\ parameters \tag{6.30}$$

$$S_1 = set\ of\ premise\ parameters \tag{6.31}$$

$$S_2 = set\ of\ consequent\ parameters \tag{6.32}$$

where assuming that $H \circ F$ is linear in the elements of S_2, if we apply the Equation 6.28 with this property, we obtain the Equation 6.33.

$$H(output) = H \circ F(I, S) \tag{6.33}$$

Assuming that the training data P is given, plugging this into Equation 6.33, the general solution for the output problem can be stated as in Equation 6.34.

$$AX = B \tag{6.34}$$

In this general equation, X is the vector representing unknown parameters in S_2, having length M. The dimensions of A, X and B are $P{\times}M$, $M{\times}1$ and $P{\times}1$, respectively. The problem is just an input, output linear algebra solution, where the number of training data P is, in most cases, are larger than the number of parameters M. As a result, this problem is an overdetermined problem where there is no exact solution to the equation at hand. Instead of generating the exact solution, the possible pseudo solution having the minimized squared error solution can be presented which is called the $LeastSquaresEstimate(LSE)$. The best possible solution in the least squares sense can be stated as follows.

$$X^* = (A^T A)^{-1} A^T B \tag{6.35}$$

In this formula, we call the part $(A^T A)^{-1} A^T$ as the pseudo-inverse of A. The solution for this equation is computationally expensive because of the inverse present in the formula. In order to solve this problem, iterative approach is selected to compute the LSE of X. These formulas are used in Adaptive Signal Processing extensively as follows.

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \tag{6.36}$$

$$S_{i+1} = S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}} \tag{6.37}$$

where S_i is called the covariance matrix and i ranges from 0 to $P-1$. As expected this iterative algorithm needs to have some initial conditions as: $X_0 = 0$ and $S_0 = \gamma I$, where γ is a positive large number and I is the $M \times M$ identity matrix. It is time to combine the gradient and the least squares approach to obtain the hybrid learning technique. This hybrid learning technique includes $epochs$ which includes both forward and backward passes. The forward pass includes introducing the input data to the system and signals go forward to calculate each node output until the matrices A and B in general equation are obtained. Hence, the parameters in S_2 are identified by the sequential least squares formulas in the previous equations. The forward pass continues until all the node error measures calculated for the first time. The backward pass does the error rate updating parameters in S_1 by propagating the output end toward the input using gradient method.

6.1.2 Pattern (On-Line) Learning

Pattern learning induces that the parameters of the system are updated after each data presentation to the system. If a system has a changing characteristics with time, this pattern learning technique is much preferable to use. For this kind of online learning the update equations are changed from the previous learning algorithm. This algorithm incorporates the decaying of the old parameters with the decaying or forgetting factor λ, stated as follows:

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \tag{6.38}$$

$$S_{i+1} = \frac{1}{\lambda} [S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}] \tag{6.39}$$

where λ is between 0 and 1. If the λ is small, this means that the old data effects decays faster. Faster convergence in some cases can introduce instability and should be avoided.

6.1.3 Implementations of ANFIS using MATLAB

MATLAB has an extensive toolbox on fuzzy systems called Fuzzy Logic Toolbox where you can model your adaptive-network-based fuzzy inference system. Mandani type fuzzy systems are modelled by mapping input characteristics to input member functions, input membership functions to rules, rules to a set of output characteristics. Output characteristics are modelled by mapping output member functions, and the output membership function mapped to a single-valued output or a decision associated with the output. The function used for Adaptive-Network-Based Fuzzy Inference System (ANFIS) is called $anfis$. The membership functions for the ANFIS structure can be selected using the $genfis1$ command.

The basic principles under the ANFIS method is neural networks. These networks incorporates neural adaptive learning patterns to learn information about an input-output (I/O) relationship. ANFIS derives its name from "Adaptive-Network-Based Fuzzy Inference System" where membership function parameters are tuned using a back-propagation algorithm alone or in combination with a least squares type of method. In this way, the fuzzy system update and train their parameters to minimize total error. The learning and training process helps the parameters to find their optimum inputs. Parameter computation is handled by the traction of the gradient vector. ANFIS uses generally the same procedure with other system identification algorithms. What ANFIS model uses to find the parameters are the input and output for forming the training data and the ANFIS system.

Input and output membership functions are modelled using this training data, later defuzzification of the output membership functions lead to the crisp output which is the desired output. At this point, the training data is really important to represent the features of the system, because it is the data that shapes the ANFIS system to model the nonlinear dynamics. Training data should not be noisy or corrupted hence this leads to corrupt system models which are not desired. What is needed for this testing is the "model validation using testing and checking data sets". In this step, the ANFIS system formed from training data is put under an unknown input and ANFIS output is compared to the desired output to see if the system works correctly. This is the crucial step where you use the algorithm to process different inputs and get your

desired result. In our case, the input is the guitar audio signal which comes from the electrical guitar to the ANFIS system. The designed system should be put under test with random playing and testing sound perceptionally to the desired output. The gist of collecting the training data is to find the most random and lengthy sequence that you are allowed to have the necessary time to process it. This leads to a greatly designed member functions and good results in terms of performance and efficiency.

MATLAB allows to test various scenarios before delving into much detail in real-time coding. For this reason, this part will first start with a basic example to model a distortion effect that is investigated. I currently use Line 6 Spider IV 15 Watt Combo Amplifier at home, it is both versatile and effective in its digital signal processing implementation especially in distortion. In my first implementation, I switched to the distortion effect, then feed the amplifier with a 440Hz (A440 - 4th Octave A Note) sinusoidal having amplitude unity, and simultaneously recorded the output of the amplifier. This is done to see at least what kind of clipping has the amplifier done to just a single sinusoidal input. With the help of MATLAB, the input output graphs of the functions are shown in Figure 6.2.

Expected as usual, the distortion channel of my amplifier clipped the sinusoidal around +0.5 on the positive side and -0.5 on the negative side. This clipping seems to be uniform, but the parts where the signal is clipped differs from one to one because of the dynamics processing of the amplifier, response of the loudspeaker and additionally the noise added through the processing. Overall, this gives us a gist of how the distortion channel behaves when feed with distinct inputs. The next step is to build the ANFIS structure over the training data at hand. The training data for this specific example is just the input $x[n]$ in one column and the last column is used for the real output obtained from the output of the amplifier denoted by $l[n]$. This is represented in MATLAB as $trnData = [x\ l]$;. Introductory to the ANFIS method I wanted to choose 3 input member functions and 3 output member functions for this method, for tracking to be easy and I will increase member functions to show the effect of choosing the necessary number of member functions and iterations. The results of this implementation are given in the Results chapter.

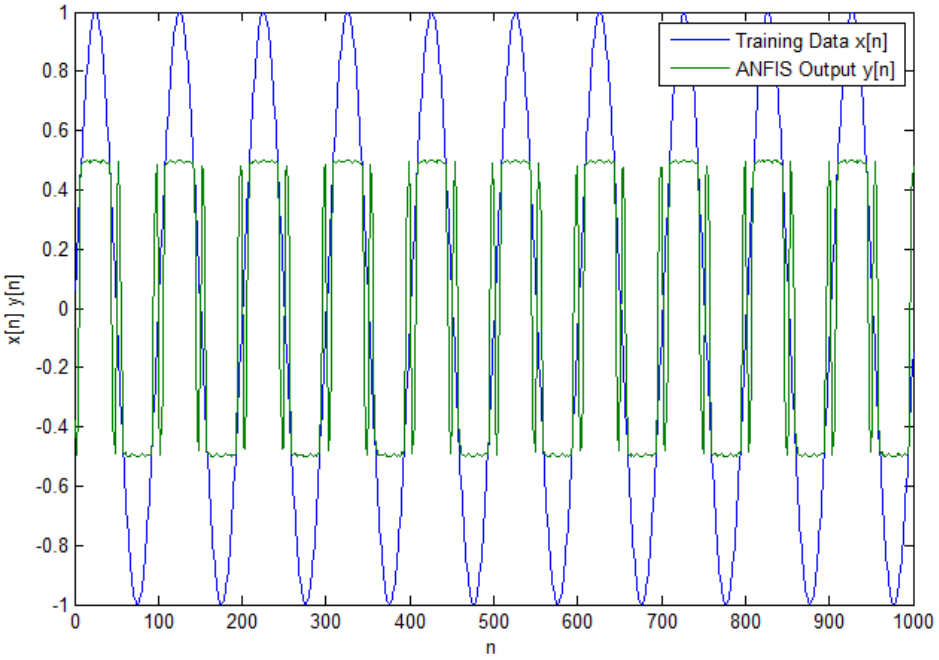

Figure 6.2: The output of the Line 6 Spider IV Distortion Channel for Sinusoidal Input at 440Hz.

CHAPTER 7

ENHANCED MODELLING OF GUITAR DISTORTION

7.1 Enhanced Modelling of Guitar Distortion

In Chapter 6, we have introduced a method for modelling guitar audio effect distortion/overdrive using ANFIS system identification algorithm. This chapter continues with introducing simulation of clean channel and lead channel of a guitar amplifier using Enhanced Modelling of Guitar Distortion algorithm. Through the emergence of blues, rock and metal, there happens to be tons of changes of how the electric guitar input is amplified, shaped and distorted to get the best possible tone from the supplied equipment. The main tone of a guitar player can be separately investigated into two separate signal channels: Clean Channel and Lead/Overdrive/Distortion Channel, changing names through different manufacturers and amplifiers appear. This thesis presents a method to easily simulate these two channels, obtaining desired audio outputs.

Each of electric guitar players have unique amplifier, equalizer, rack effects and stomp box effects to characterize their sounds. In this chapter, we introduce an applicable method to analyze and model such a signal processing algorithm to obtain desired clean and lead tones at the output. The signal flow path is given in Figure 7.1, where $x[n]$ shows the input from the electric guitar, $y_c[n]$ represents the clean channel output and $y_l[n]$ represents the lead/distortion channel output. The inputs for the ANFIS System Identification (Distortion) are the training input and output pairs recorded simultaneously from the desired distortion effects pedal, $x_a[n]$ and $y_a[n]$ respectively. $x_a[n]$ and $y_a[n]$ are training input and output arrays of desired length. Before the real-

time loop starts, the block called the ANFIS System Identification (Distortion) needs to be trained with the sampled distortion/overdrive effects pedal input and output, as mentioned, $x_a[n]$ and $y_a[n]$ respectively. This training results into an ANFIS Identification System called outfis which consists of the membership function type, number, rules of the neural network, firing strength of each path and all other necessary information needed to define the neural network. We then use this outfis system and the clean tubed output signal $y_c[n]$ as inputs into the ANFIS Evaluator (Distortion) system. This is basically the neural network evaluator, which applies the neural network rules designed inside the outfis to shape the input $y_c[n]$, in order to obtain the distorted/overdrived lead output called $y_l[n]$. This in turn allows the user to shape their own distortion/overdrive curve with their desired overdrive/distortion effect/pedal. The signal chain figure of the whole system is given in Figure 7.1.

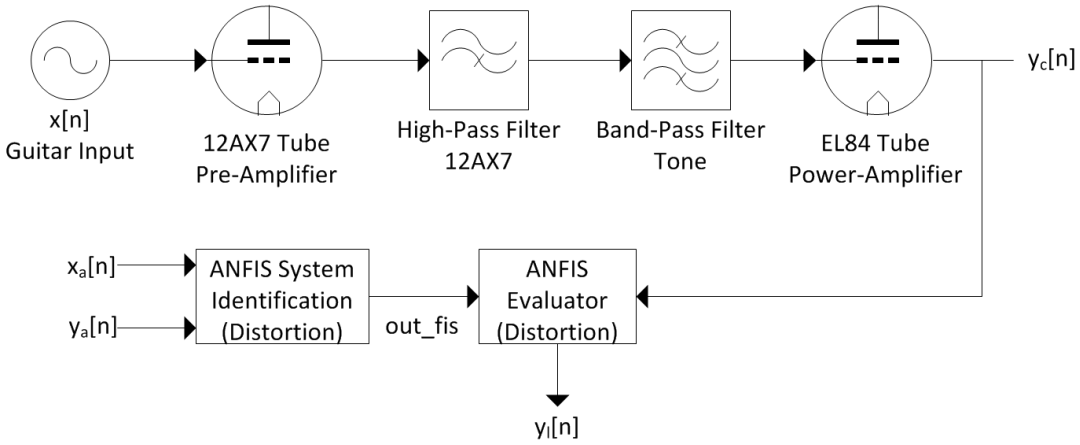

Figure 7.1: The Signal Chain of the Clean and Lead Guitar Output.

7.1.1 12AX7 Tube Pre-Amplifier

Guitar tube amplifiers have the necessery first stage pre-amplifier section, which amplifies the signal and gives its warm and touchy feel of the tube amplification. It is very common to see nearly the same pre-amplifier schematics with changing resistance and capacitor values inside different amplifiers. In order to simulate a tube amplifier, some important simulation methods have been introduced by different researchers. Due to its dynamic nonlinearity and high fidelity of the amplifier simula-

tion, this topic is still debated among researchers.

One of the most important attempts to standardize tube amplifier simulation have been introduced by W. Marshall Leach, Jr. in his paper called SPICE Models for Vacuum-Tube Amplifiers in 1995 [12]. In this paper, he proposes a method to simulate and approximate the behavior of one of the most commonly used pre-amplifier vacuum tubes: 12AX7. The approximation of plate current can be shown as below, directly the way he had shown:

$$i_P = K.(\mu.V_{GK} + V_{PK})^{\frac{3}{2}} \qquad for\ \mu.V_{GK} + V_{PK} \geq 0 \tag{7.1}$$

$$= 0. \qquad for\ \mu.V_{GK} + V_{PK} < 0 \tag{7.2}$$

Leach [95], then states the small-signal plate current for 12AX7 Tube Amplifier Triode as:

$$i_p = \frac{\partial I_P}{\partial V_{GK}}.v_{gk} + \frac{\partial I_P}{\partial V_{PK}}.v_{pk} \tag{7.3}$$

$$= g_m.v_{gk} + \frac{v_{pk}}{r_p} \tag{7.4}$$

where the transconductance g_m and the plate resistance r_p is represented by the following equations and the Figure 7.2 represents the equivalent circuits for the tube model.

$$g_m = \frac{\partial I_P}{\partial V_{GK}} = \frac{3\mu I_p}{2(\mu V_{GK} + V_{PK})} \tag{7.5}$$

$$r_p = \frac{\partial I_P}{\partial V_{GK}}^{-1} \tag{7.6}$$

Pspice subcircuit models are created by the SPICE netlists of the designed triode tube amplifier model to incorporate the model into the standardized SPICE circuit design and analysis software. What we have incorporated from his simplified model is the frequency response of the amplifier. Generally amplifiers are designed to use their linear region with the desired frequencies to be amplified having a flat gain in

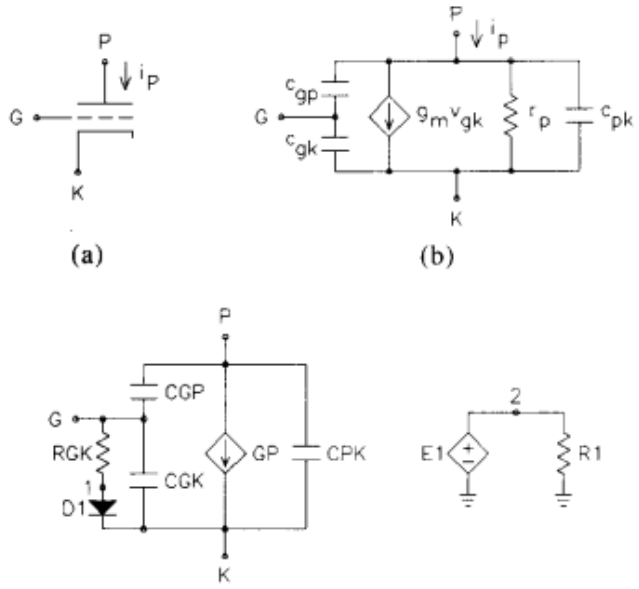

Figure 7.2: The Equivalent Model for the Triode Tube Amplifier.

the frequency response plots. In audio processing, we have the whole hearing range between 20Hz to 20kHz to be processed with the frequency response of the amplifier, which in turn is very important for tonal characteristics of the output. Taking a look at the sample frequency response of a general tube amplifier circuit design, we have also modelled our system to get tube warm and characteristic output.

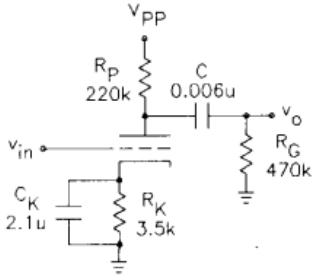

Figure 7.3: The Sample Pre-Amplifer Circuit Using 12AX7 Tube Amplifier, Leach[95].

In current ideal modelling schemes, it is always assumed that we have a complete knowledge over the circuit of the distortion or the amplifier part that we want to model. For an end-user, or the music enthusiast, what is more important is how it sounds overall in the output. For these purposes, we will continue with combining real-time models coming from previous chapters with the ANFIS modelling results of

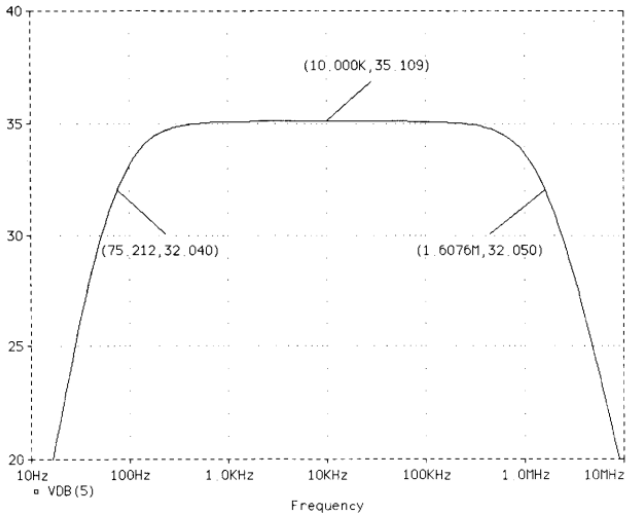

Figure 7.4: The Frequency Response of 12AX7 Tube Pre-Amplifier Circuit.

the distortion input output training pairs at hand. For this purpose, we will introduce the state space model generally used for solving linear systems. This nonlinear state space solution is the ideal solution that we compare with our ANFIS estimation that models the system as a black box.

7.1.2 Obtaining Training Data For ANFIS System Identification

Nonlinear state space solutions are used to model nonlinear guitar audio effects assuming that the inside circuitry is known. [18] [19]. This solution is the ideal case where results come close to the actual circuit solution. Given the system parameter values and the connections, resistors, capacitors, inductors, the state-space equations can be as below [5]:

$$\dot{x}(t) = A.x(t) + B.u(t) \tag{7.7}$$

$$y(t) = D.x(t) + E.u(t) \tag{7.8}$$

where $x(t)$ is the state variable, $\dot{x}(t)$ is the derivative of the state variable, $u(t)$ is the input and $y(t)$ is the output functions with respect to time, t. A is the state, B the input, D the output and E the feed-through matrix of the general linear state-space

equation [9]. The next two steps is the discretization of this continuous time linear state-space equation and introducing a nonlinear element into the functions to solve nonlinear state-space equations as in Equations 7.9 to 7.11.

$$\dot{x} = a.x + b.u + C.i(v) \tag{7.9}$$

$$y = d.x + e.u + f.i(v) \tag{7.10}$$

$$v = G.x + h.u + K.i(v) \tag{7.11}$$

This equation is an extension to the previous linear state-space equation, by introducing nonlinear function of $i(v)$ that depends nonlinearly to parameter v [5]. The discretization and canonicalization of the nonlinear state-space equation results into the following discrete time representations of the solution in Equations 7.12 to 7.14.

$$x_c[n] = \bar{A}.x_c[n-1] + \bar{b}.u[n] + \bar{c}.i[v[n]] \tag{7.12}$$

$$y[n] = \bar{d}.x_c[n-1] + \bar{e}.u[n] + \bar{f}.i[v[n]] \tag{7.13}$$

$$v[n] = \bar{G}.x_c[n-1] + \bar{h}.u[n] + \bar{K}.i[v[n]] \tag{7.14}$$

The canonicalized matrix equations are represented in Equations 7.15 to 7.19.

$$\bar{A} = 2(\frac{2}{T}I - A)^{-1}C \tag{7.15}$$

$$\bar{f} = f + d(\frac{2}{T}I - A)^{-1}C \tag{7.16}$$

$$\bar{G} = \frac{2}{T}G(\frac{2}{T}I - A)^{-1} \tag{7.17}$$

$$\bar{h} = h + G(\frac{2}{T}I - A)^{-1}b \tag{7.18}$$

$$\bar{K} = K + G(\frac{2}{T}I - A)^{-1}C \tag{7.19}$$

Dempwolf, Holters and Zölzer have clearly stated the algorithm to simulate a solution with a software like $MATLAB$ [5], as summarized with the following steps of the analysis:

1. State-space analysis of the circuit: Find a matrix formulation in the continuous time domain. The voltages across the capacitors are defining state. Contributions from non-linear elements are expressed by $i[v[n]]$.

2. Perform the discretization (e.g. trapezoidal rule) to achieve the discrete-time representation.

3. Filtering process:

- Solve the non-linear equation given in equation of $v[n]$ to obtain $i[v[n]]$ from the current input $u[n]$ and the previous state $x_c[n-1]$. Here we exploit the fact that equation of $v[n]$ depends on $x_c[n-1]$, not $x_c[n]$.

- Once i[v[n]] is determined, the new state and the output are computed with the above equations of $x_c[n]$ and $y[n]$.
- Update the matrix entries in case of parameter changes.

The following algorithm is then applied to model a pre-amplifier of the famous Marshall JCM900 vacuum tube guitar amplifier [5]. This is exactly the A-channel of the Hi-Gain Dual Reverb head. This section is perfect for our purposes because of its dual rectifier/limiters nonlinearity present inside the circuit, seen as in Figure 7.5 [5].

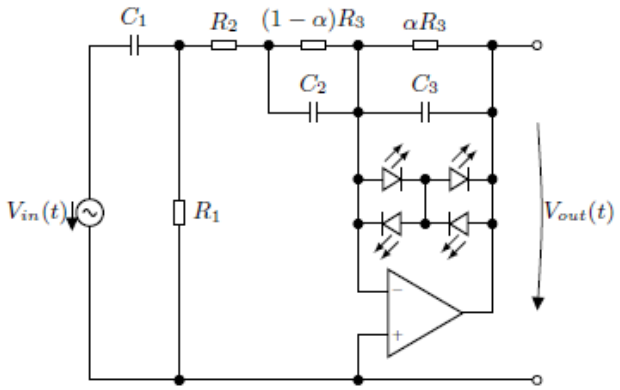

Figure 7.5: Dual Rectifier Pre-Amplifier Stage of Marshall JCM900.

This circuit design is referenced from the DAFx-10 paper of Dempwolf, Holters and Zölzer called "Discretization of Parametric Analog Circuits for Real-Time Simula-

tions" and JCM900 and all of its parts are trademarks of the Marshall Inc. [5]. The parameter values can be listed as follows: $R_1 = 22k\Omega$, $R_2 = 12k\Omega$, $R_3 = 220k\Omega$, $C_1 = 47nF$, $C_2 = 1nF$, $C_3 = 47pF$ and the α is the potentiometer gain parameter. Nonlinear devices are the diodes which has red color with 3mm LEDs. The open-loop gain of the pre-amplifier is selected as 25dB, which is higher than the actual gain it supplies in the whole circuit. In the simulations, the operational amplifier is assumed to be ideal, having no voltage drop between the inputs.

Nonlinear part of the circuit, rectifier diodes, are assumed to be ideal and identical with each other. Given the voltage applied to the diode, V_F, the current following through the diode can be stated using the Shockley equation [5]:

$$i_{D1} = I_s.(e^{\frac{V_F}{n.V_t}} - 1) \tag{7.20}$$

The Shockley equation parameters are stated as following: reverse saturation current $I_s = 6.5x10^{-20}A$, thermal voltage $V_t = 25.3mV$ and the emission coefficient $n = 1.68$ [5]. The combination of two sided diodes can be summarized into a single equation as in Equation 7.22.

$$i = i_D = I_s.(e^{\frac{V_{C3}}{2.n.V_t}} - 1) - I_s.(e^{\frac{-V_{C3}}{2.n.V_t}} - 1) \tag{7.21}$$

$$= 2.I_s.sinh(\frac{V_{C3}}{2.n.V_t}) \tag{7.22}$$

The state space solution can be done by stating the state-space matrix according to the input voltage u, and the voltages across capacitors.

$$A = \begin{bmatrix} \frac{R_1+R_2}{C_1.R_1.R_2} & -\frac{1}{R_2.C_1} & 0 \\ -\frac{1}{R_2.C_2} & 0 & -\frac{R_2+(1-\alpha).R_3}{(1-\alpha).R_2.R_3.C_2} \quad 0 \\ -\frac{1}{R_2.C_3} & -\frac{1}{R_2.C_3} & -\frac{1}{\alpha.R_3.C_3} \end{bmatrix} \tag{7.23}$$

$$b = \begin{bmatrix} \frac{R_1+R_2}{C_1.R_1.R_2} \\ -\frac{1}{R_2.C_2} \\ -\frac{1}{R_2.C_3} \end{bmatrix} \tag{7.24}$$

$$C = \begin{bmatrix} 0 \\ 0 \\ -\frac{1}{C_3} \end{bmatrix} \tag{7.25}$$

$$x = \begin{bmatrix} v_{C_1} \\ v_{C_2} \\ v_{C_3} \end{bmatrix} \tag{7.26}$$

$$y = \begin{bmatrix} -v_{C_3} \end{bmatrix} \tag{7.27}$$

$$d = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix} \tag{7.28}$$

$$e = \begin{bmatrix} 0 \end{bmatrix} \tag{7.29}$$

$$f = \begin{bmatrix} 0 \end{bmatrix} \tag{7.30}$$

$$G = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{7.31}$$

$$h = \begin{bmatrix} 0 \end{bmatrix} \tag{7.32}$$

$$K = \begin{bmatrix} 0 \end{bmatrix} \tag{7.33}$$

Pre-calculation of canonized version of the actual solution is required because it takes long time for calculation of inverse matrix afterwards, or in the simulation. The resulting steps can be summarized as follows:

$$A = -\begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix}^{-1} \times \left(\begin{bmatrix} \frac{1}{R_1} & 0 & 0 \\ 0 & \frac{1}{(1-\alpha).R_3} & 0 \\ 0 & 0 & \frac{1}{\alpha.R_3} \end{bmatrix} + \frac{1}{R_2} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \right) \tag{7.34}$$

and the matrix inversion used in the equations can be written as the following:

$$(\frac{2}{T}I - A)^{-1} = \left(\begin{bmatrix} {\beta_1}^{-1} & 0 & 0 \\ 0 & {\beta_2}^{-1} & 0 \\ 0 & 0 & {\beta_3}^{-1} \end{bmatrix} + \frac{1}{R_2} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \right)^{-1} \tag{7.35}$$

$$\cdot \begin{bmatrix} C_1 & 0 & 0 \\ 0 & C_2 & 0 \\ 0 & 0 & C_3 \end{bmatrix}$$

where the parameters β is defined as follows:

$$\beta_1 = \frac{R_1.T}{2.C_1.R_1 + T} \tag{7.36}$$

$$\beta_2 = \frac{(1-\alpha)R_3.T}{2.(1-\alpha).C_2.R_3 + T} \tag{7.37}$$

$$\beta_3 = \frac{\alpha.R_3.T}{2.\alpha.C_3.R_3 + T} \tag{7.38}$$

Applying Sherman-Morrison formula to the previous equations, we obtain the simplified solution to the matrices required to solve nonlinear state-space equations [5].

$$\bar{A} = \begin{bmatrix} \frac{2.C_1.R_1 - T}{2.C_1.R_1 + T} & 0 & 0 \\ 0 & \frac{2.(1-\alpha).C_2.R_3 - T}{2.(1-\alpha).C_2.R_3 + T} & 0 \\ 0 & 0 & \frac{2.\alpha.C_3.R_3 - T}{2.\alpha.C_3.R_3 + T} \end{bmatrix} \tag{7.39}$$

$$-\frac{4}{R_1.T.(R_2 + \beta_1 + \beta_2)} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} \cdot \begin{bmatrix} \beta_1.C_1 & \beta_2.C_2 & 0 \end{bmatrix}$$

$$\bar{b} = -\frac{2}{R_1.(R_2 + \beta_1 + \beta_2)} \begin{bmatrix} \beta_1.(R_1 + R_2 + \beta_2) \\ \beta_2.(R_1 - \beta_1) \\ \beta_3.(R_1 + \beta_1) \end{bmatrix} \tag{7.40}$$

$$\bar{C} = -2 \begin{bmatrix} 0 \\ 0 \\ \beta_3 \end{bmatrix} \tag{7.41}$$

$$\bar{d} = \frac{2}{T} \cdot \begin{bmatrix} \beta_1.C_1.\frac{\beta_3}{R_2+\beta_1+\beta_2} & \beta_2.C_2.\frac{\beta_3}{R_2+\beta_1+\beta_2} & -\beta_3.C_3 \end{bmatrix} \tag{7.42}$$

$$\bar{e} = -\frac{\beta_3.(R_1 - \beta_1)}{(R_2 + \beta_1 + \beta_2).R_1} \tag{7.43}$$

$$\bar{f} = \beta_3 \tag{7.44}$$

$$\bar{G} = -\bar{d} \tag{7.45}$$

$$\bar{h} = -\bar{e} \tag{7.46}$$

$$\bar{K} = -\beta_3 \tag{7.47}$$

This nonlinear state space solution is solved using $MATLAB$ as can be found on the appendix section, to obtain an input and output reference for the training of the ANFIS structure. We will use the output from the solved nonlinear state space equation as the ideal output of the JCM900 pre-amp as we have not the real amplifier. This ideal solution is taken as a reference to our training input and output pair to the ANFIS System Identification (Distortion) part as signals $x_a[n]$ and $y_a[n]$ respectively.

7.1.3 Enhanced Modelling of Guitar Distortion Algorithm

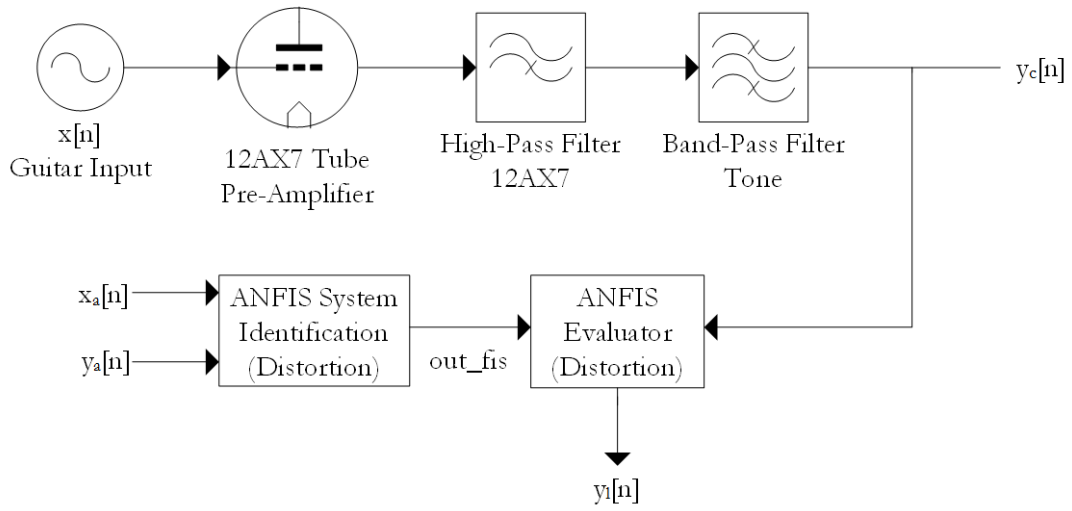

Figure 7.6: Enhanced Modelling of Guitar Distortion Model Basic Signal Flow Diagram.

Enhanced Modelling of Guitar Distortion algorithm uses the ideal training input ($x_a[n]$) and output ($y_a[n]$) pair obtained from the nonlinear state space solution offered by Dempwölf and Zölzer. Using this training pair, the black box system is estimated using ANFIS System Identification block. Then the ANFIS system is used to process future outputs to obtain lead channel. The type and number of membership functions, epoch and input selection affect the performance of the ANFIS estimation. Different parameters are selected to get the best result both computationaly and performance

wise. Another important point is the selection of the length of the training input output pair. For our purposes, we have recorded multiple 140000 sample length electric guitar output which includes different guitar playing styles and speeds. We trained the system with the corresponding input and output pair that produces the nearest estimate to the ideal model. The results are presented in the Results chapter. Enhanced Modelling of Guitar Distortion is a 6 step algorithm summarized as follows:

1. The guitar output signal, $x[n]$, is used as an input to the 12AX7 pre-amplifier model.

2. Then the equalizer high-pass and band-pass filters are used to change tonal characteristics of pre-amplifier output.

3. EL84 vacuum tube power amplifier model amplifies the equalized signal. The output of the vacuum tube power amplifier is called clean channel output, $y_c[n]$.

4. ANFIS System Identification block model takes two inputs, $x_a[n]$ and $y_a[n]$. $x_a[n]$ is the training input signal to the distortion effects pedal that is modelled. $y_a[n]$ is the output of the distortion effects pedal for the training input $x_a[n]$. ANFIS System Identification code block trains these input output sequence to form an ANFIS structure called $outfis$.

5. This ANFIS structure can be used to evaluate future outputs for random inputs coming from the $y_c[n]$ clean channel. ANFIS Evaluator block evaluates the lead channel output, $y_l[n]$, using the ANFIS structure, $outfis$ and $y_c[n]$ as inputs.

6. The main channels' simulation of a guitar amplifier, clean channel ($y_c[n]$) and lead channel ($y_l[n]$) outputs are obtained.

CHAPTER 8

RESULTS

8.1 Results

There are different testing methods for different digital guitar effects. Most of them includes the comparison of the ideal guitar audio effect time domain output to the simulated one. The guitar audio effects that change the frequency characteristics of the input signal are also compared using the frequency domain comparisons. Linear guitar audio effects, such as delay, reverb, wah-wah, flanger are tested regarding these testing methods. For nonlinear guitar audio effects such as distortion and overdrive, additional tests are presented. Sinusoidal input test and log-sine sweep spectrogram are two methods for testing distortion and overdrive simulated methods that are presented in the results section.

A sample input is recorded to test the delay, reverberation, wah-wah and flanger effects. This recording is the low E note of the electric guitar. The sample input is called samplein. This input is fed to the designed guitar effect algorithms in C, C++ and C# and the output is recorded. Then the output is called sampleout. For each of the linear guitar audio effects, we compare samplein and sampleout to see the behaviour of each effect. The ANFIS System Identification Algorithm will be tested with different membership and epoch functions to find the best possible RMSE values as presented as a testing method [10]. For the testing of the Enhanced Modelling of Guitar Distortion Algorithm, we selected the method presented in Dempwolf and Zolzer' s paper [6].

Each guitar effect is designed in $MATLAB$, implemented in Microsoft Visual Stu-

dio 2010 using C++ and C# and then ported to Texas Instruments TMS320VC5505 eZdsp. $MATLAB$ is used for the initial design of the guitar effects algorithms. Microsoft Visual Studio 2010 is used for real time application development in the Windows platform. ASIO (ASIO4ALL) - Universal ASIO Driver for WDM Audio is used to satisfy real-time development constraints. ASIO drivers can be set with 64 samples of ASIO Buffer Size, which is the best real-time constraint to satisfy in ASIO. Latency Compensation parameters are set to 0 samples for In and Out. Buffer Offset is also set to 0 ms. Realtek High Definition sound card is used for the testing. Realtek HD Audio output and Realtek HD Audio Line input are used for effect input and output.

8.1.1 Delay Results

The sample input samplein is fed to the input of the designed delay system. The comparison of samplein and sampleout is presented in Figure 8.1. In the implementation of delay, a complementary band-pass filtering can be applied to obtain desired guitar tone in the output. The characteristics of this filter is shown in the Figure 8.4. The delay is implemented using the FIR comb filter structure in the Linear Guitar Audio Effects - Basic Building Block: Delay section. The FIR comb filter has been called with delay parameter of 0.5 sec and 1 gain. Additionally, group delay (in samples) plots for single delay element and FIR Comb Filter are shown respectively in Figures 8.2 and 8.3.

8.1.2 Reverberation Results

The sample input samplein is fed to the input of the designed reverberation system. The comparison of samplein and sampleout is presented in the below graphs. In the implementation of reverberation, a complementary band-pass filtering can be applied to obtain desired guitar tone in the output. The characteristics of this filter is shown in Figure 8.4. The reverberation is implemented using the digital all-pass filter structure in the Reverberation - Digital All-Pass Filter section. The digital all-pass filter has been called with parameters of 0.5 sec of delay and $b0 = 0.2$, $b1 = 0.1$ and $a1 =$

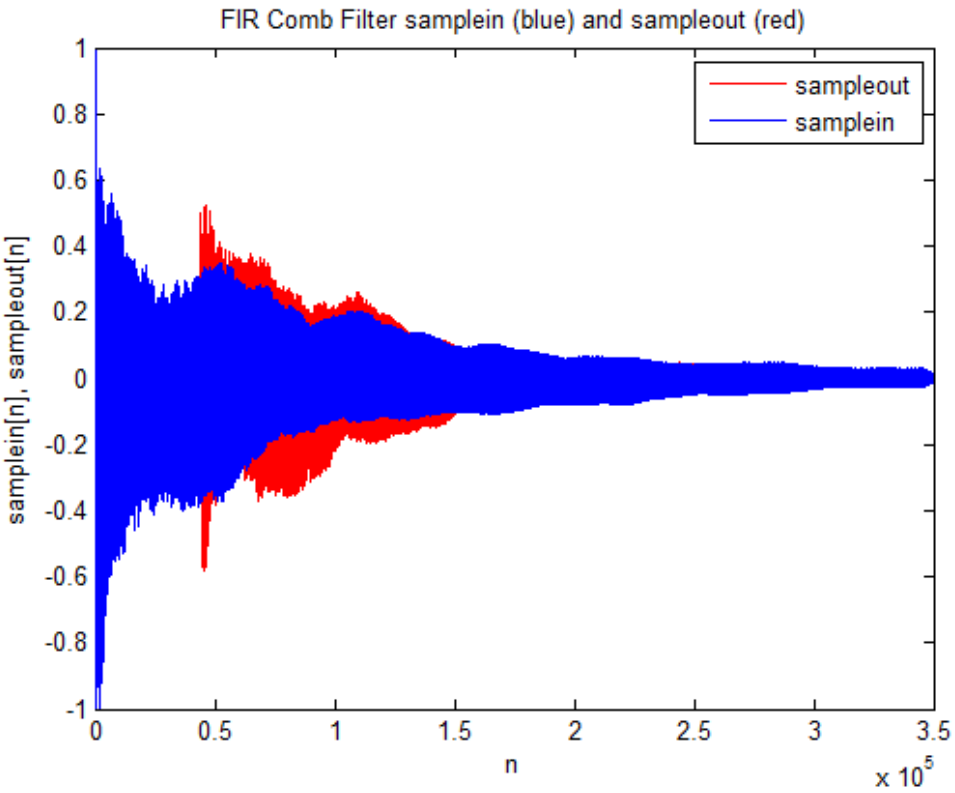

Figure 8.1: FIR Comb Filter Test Input samplein and Output sampleout.

-0.5. The group delay (in samples) plot with the same reverberation parameters is shown in Figure 8.7.

8.1.3 Wah-Wah Results

Wah-Wah implementation is done by referring to the Jim Dunlop Original Wah-Wah Pedal, GCB95. For the test of this effect, the state variable filter structure is used to implement bandpass filter with changing frequencies. One of the most famous pedals in wah-wah guitar audio effects is the Jim Dunlop Original GCB95 model. This wah pedal originated from the 1966 Thomas organ model and basically is the oldest wah pedal ever designed. The foundations under the wah pedal is the time-varying band pass filtering, so the shape and range of the filter plays an important role in the output sound. Starting with MATLAB, we have implemented this effect on various platforms and also in real-time using C# on windows applications and

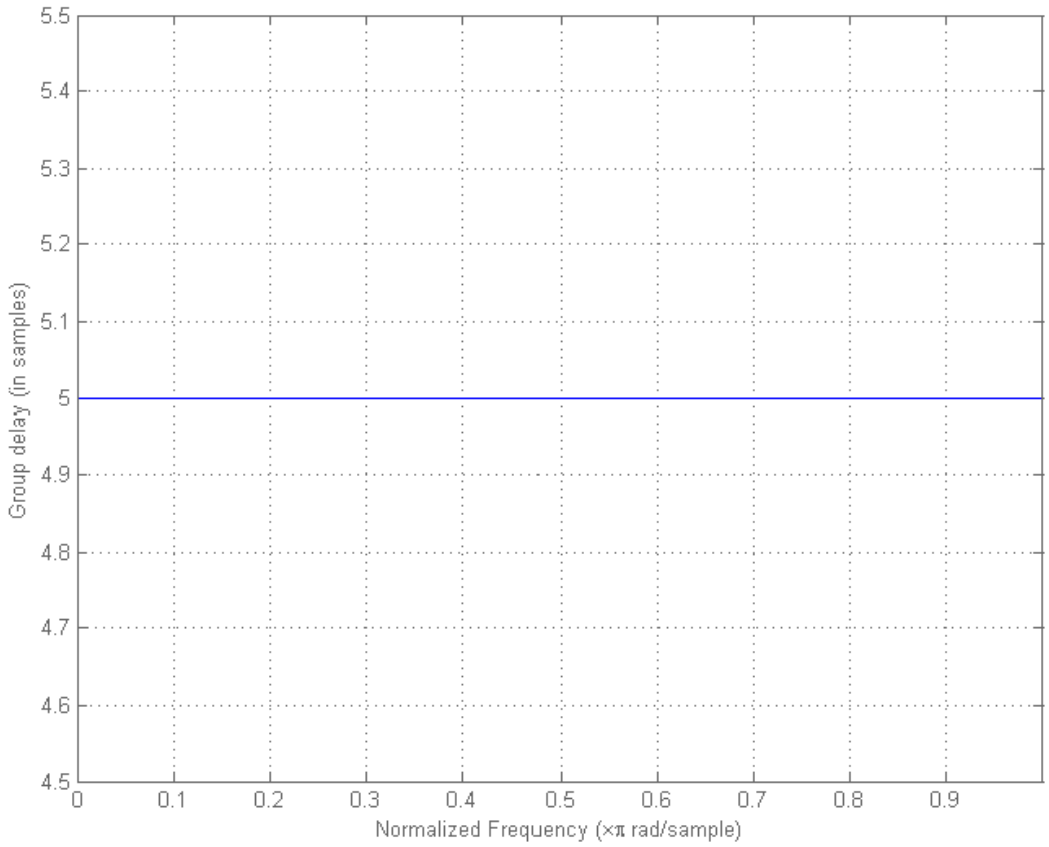

Figure 8.2: Group delay (in samples) of single delay element with 0.5 gain and 5 delay parameters.

C/C++ on Texas Instruments eZDSP implementations. Let us first start with the basics of the effect using generic MATLAB function calls and plots. In MATLAB, we first start with the trimming of the triangular wave of band-pass frequency change to the length of the input signal. The other parameter sets are the damping factor, d which is initialized with $d = 0.05$ and F_1, the calculated f_c dependent parameter is recalculated in the loop for each changing value of f_c. F_c and Q_1 is calculated extensively using the generic formulas given in Equations 2.55 and 2.56. The next step is to put the parameters F_c and Q_1 into the for loop with the state variable filter equation to calculate each sample output with the corresponding F_1 value. The wah-wah output that we are interested is the $y_b[n]$ output, which is the band-pass output of the state variable filter. Finally, we normalize the output dividing each sample output value to the maximum of the outputs, to get an output ranging from -1 to 1.

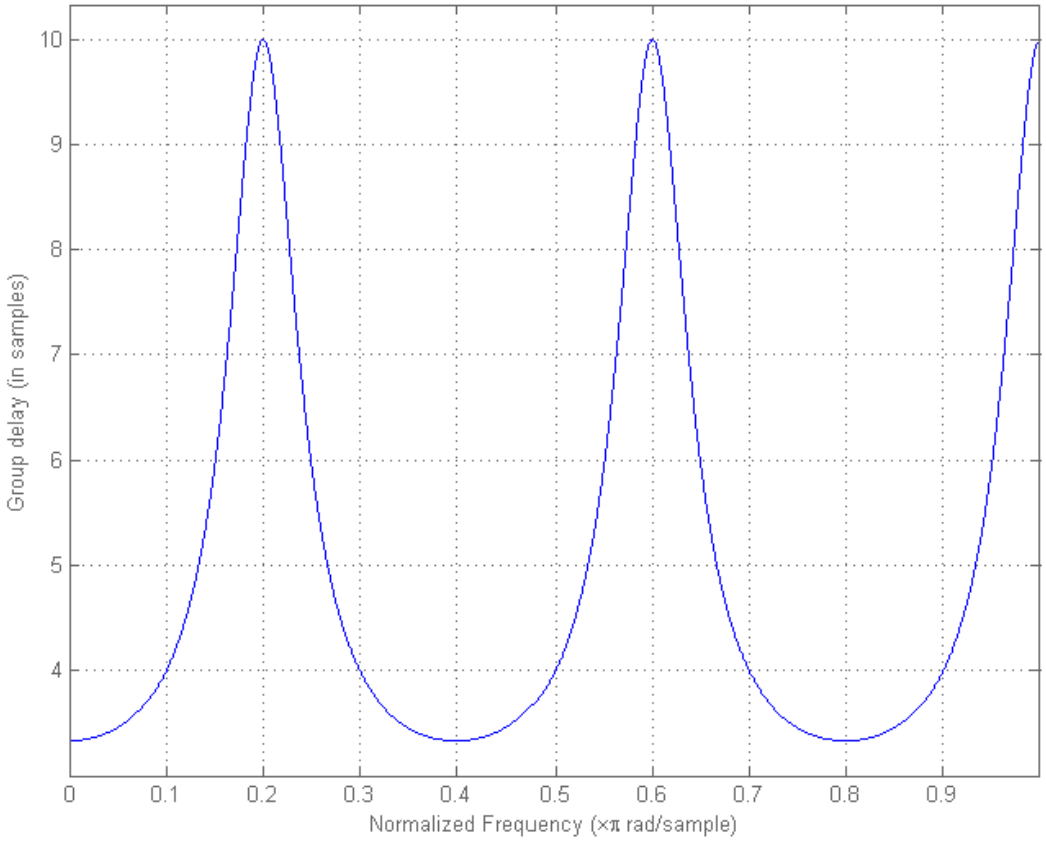

Figure 8.3: Group delay (in samples) of FIR Comb Filter 0.5 gain and 5 delay parameters.

8.1.4 ANFIS Modelling Results

Using $MATLAB$ functions, we build the $ANFIS$ model for a more generic example where a randomly played guitar input signal is fed through the $ANFIS$ system. It is aimed to estimate the black - box model of the nonlinear system using training data and to check the model with the checking data obtained from the recordings. In order to do such a comparison, one should simultaneously record the input and output of the real black-box model of the system. The black-box that is desired to be modelled in the following scenario is the amplifier called Line 6 Spider IV running in the distortion effect mode calibrated to the desired output effect.

Simultaneous input and output can be accomplished using several methods, one of which includes using Data Acquisition Toolbox from $MATLAB$. Toolbox allows

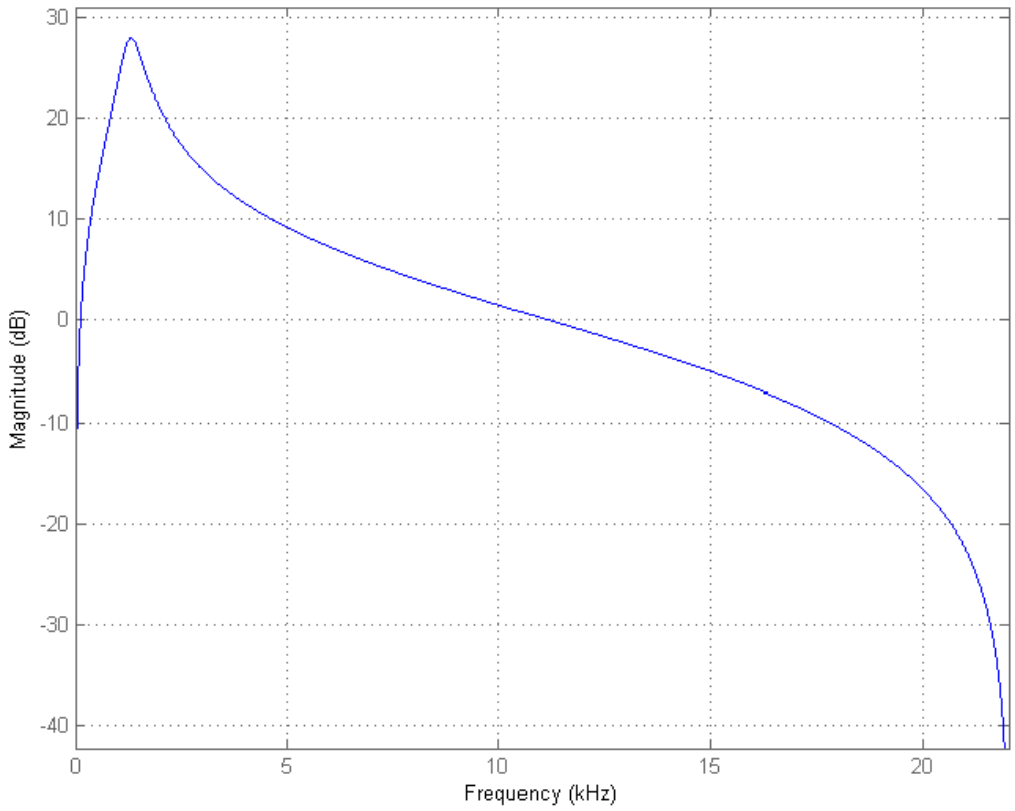

Figure 8.4: Complementary Band Pass Filter Design to Obtain Different Guitar Tones.

to trigger input and output approximately at the same time, allowing user to record input and output simultaneously. In the Figure 8.10 code block, ai is used to refer to the analog input channel, whereas ao refers to the analog output channel. First, we are going to record a sample of input guitar signal randomly played including guitar licks and rhythms using the analog input channel ai. The data is written to an input variable called ll which represents the input to the system of amplifier. This part of code records 10 seconds of samples having sampling rate of 44100 Hz.

After obtaining a proper input signal with this part of code, then we want to trim the start and end of the signal such that we have a good training input without the presence of artifacts. Applying trimming, ll becomes a signal of length 200000 samples. Scaling the signal such that the max and min values gets near to 1 and -1 gives us a nice input signal ll to use as a training input. The final version of the input signal ll can be seen as in Figure 8.11.

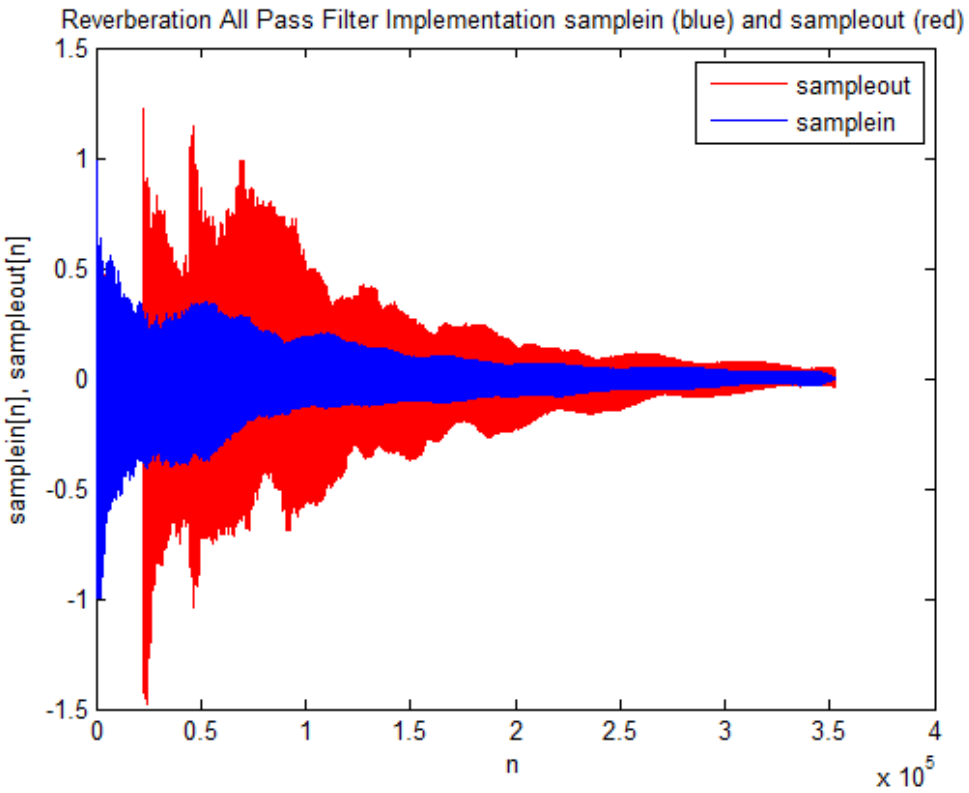

Figure 8.5: Reverberation Test Input samplein and Output sampleout.

Having the input at hand, we need to feed the amplifier with this trimmed and scaled clean input ll and simultaneously record the output dd, which is the distorted signal from the amplifier output. In order to do this we use the $trigger$ function of $MATLAB$, as shown in the following code block to record the output on the variable dd.

After $N = 100$ number of trials we calculated the correlation between the undistorted output and then the distorted outputs and find on the average our system of recording has on the average 1240 samples of delay on the input through output path which resembles a delay of $1240 samples/44100 samples/sec = 0.0281 sec$, which is a fair amount of delay on the signal path. It is important to use this data when comparing the input and output sample by sample. What is more intriguing is that selecting the distortion channel adds up $80 samples$ of delay which sums up to total of $1320 samples$. As a result, I have trimmed both the input ll and output dd just to

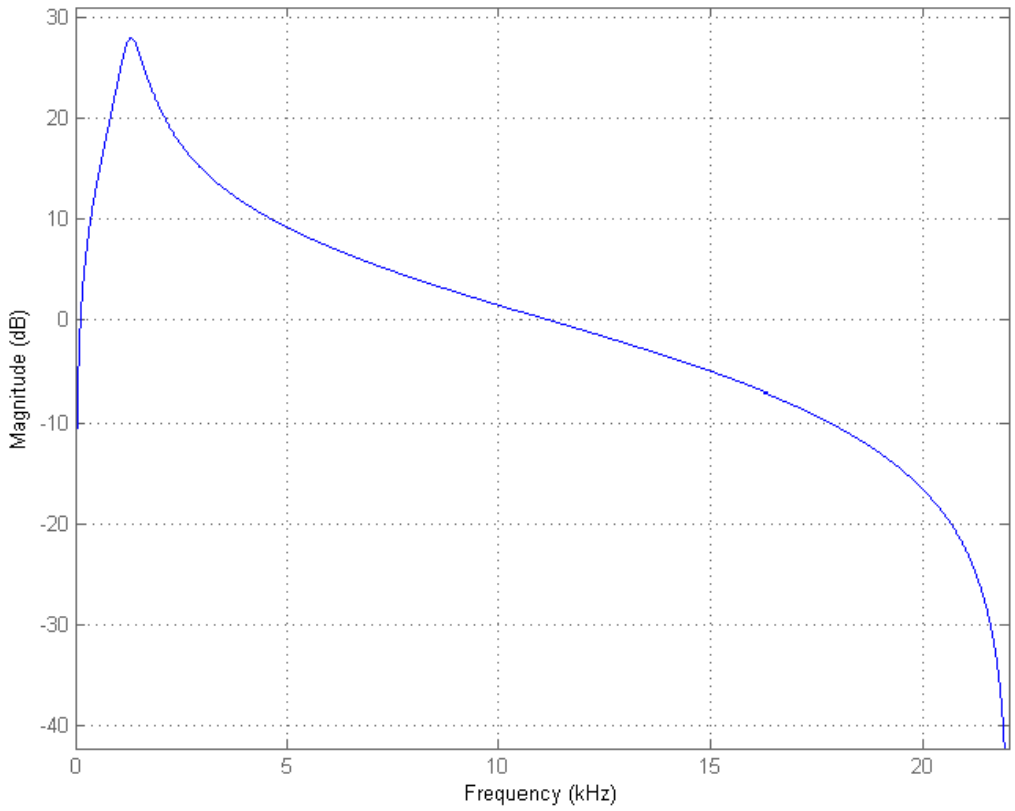

Figure 8.6: Complementary Band Pass Filter Design to Obtain Different Guitar Tones.

match the two signals at the right moment, shifted the output signal by 1320 samples to obtain the exact input output relationship. The resulting trimmed, adjusted and distorted output signal dd is shown in Figure 8.13.

As can be seen from the output signal dd, the distortion channel output has a special clipping on the input signal. It is not only clipping, also the dynamics of this nonlinear processing that we are trying to model with the ANFIS nonlinear black-box approach to the distortion problem. The mathematical foundation of a black-box model stands on the decision what kind of input parameters the model is going to use. The basic equation for a generic discrete time input signal $x[n]$ output signal $y[n]$ can be shown as below.

$$y[n] = f(x[n-1], x[n-2], ..., x[n-d_x], y[n-1], y[n-2], ..., y[n-d_y]) \quad (8.1)$$

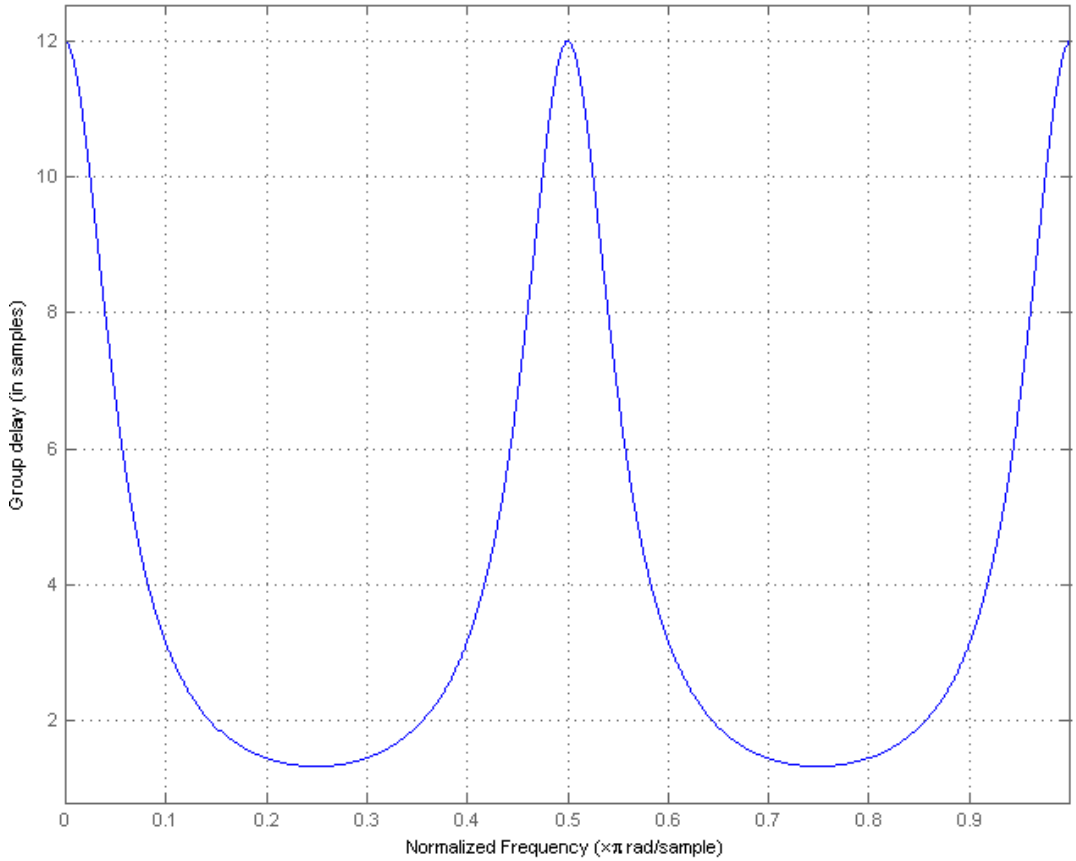

Figure 8.7: Group delay (in samples) of Reverberation.

The above equation is the very foundation of the system modelling environment where the input parameters are the input signals delayed up until d_x and also feedback from the previous outputs are fed to the input line to the evaluation function f, leading to the output sample $y[n]$. What is required to successfully model a system behavior is to estimate the function f such that the chosen inputs leads to the output with the least error possible when compared to the original output. In our method to estimate the dynamical model behind the distortion problem, we will use Adaptive-Network-Based Fuzzy Inference System (ANFIS) to formulate an estimation to the amplifier output. Such systems, in general, are called neural networks, where previous input and output are trained to form much detailed network of nodes to formulate weighted outputs according to special member functions. The theory behind the ANFIS structure is referred in the previous chapters, where a structure with membership functions are introduced extensively.

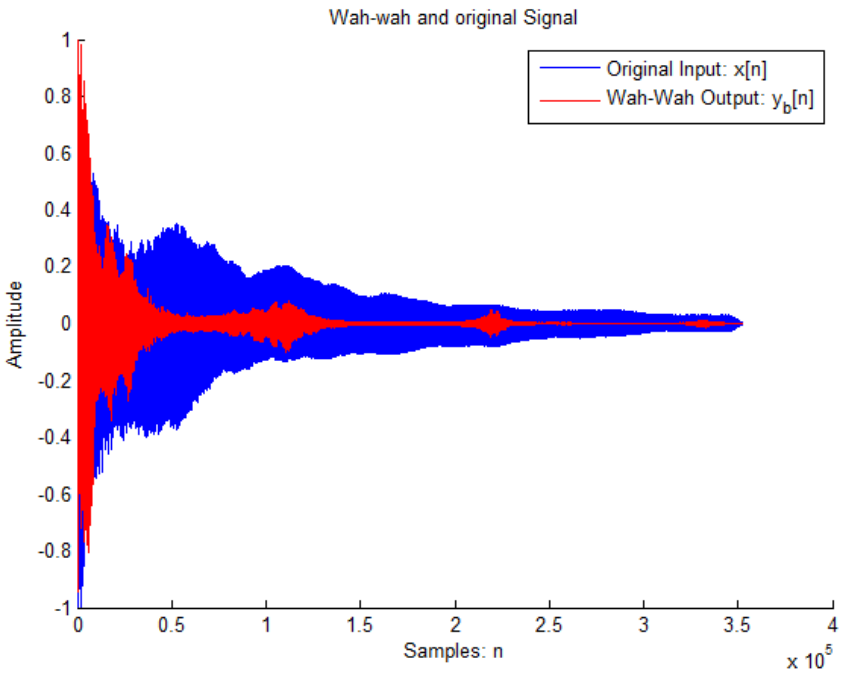

Figure 8.8: Auto-Wah Effect Output Shown With the Original Input.

The most important part of any system identification problem is to choose the number and details of the inputs chosen for the system. In such a problem of audio processing, we have an input $x[n]$ at hand where we can use the nearest $x[n-1]$ sample for estimation of the output sample at present time, $y[n]$. Apart from using $x[n-1]$, we can also use $x[n-1]$, $x[n-2]$, ..., $x[n-d_x]$, where in our following application $d_x = 6$. On the other hand, we can also use the feedback from the outputs on the input signal, the most recent output sample that we have at present time is $y[n-1]$. Continuing with others, we can also feed $y[n-1]$, $y[n-2]$, ..., $y[n-d_y]$ to the input signal path, where in our following application $d_y = 4$. As a result, we can summarize that in the input signal matrix for the following application, we are using $y[n-1]$, $y[n-2]$, $y[n-3]$, $y[n-4]$, $x[n-1]$, $x[n-2]$, $x[n-3]$, $x[n-4]$, $x[n-5]$, $x[n-6]$ respectively in this order. The choice of d_x and d_y are application and programmer specific that we are trying to minimize RMSE with these input parameters. Since we are shifting the signals at most 6 samples we trim from the beginning of the input by 6 samples in order to make the start of the input signal meaningful.

The next step is the training of previously observed and recorded audio input and

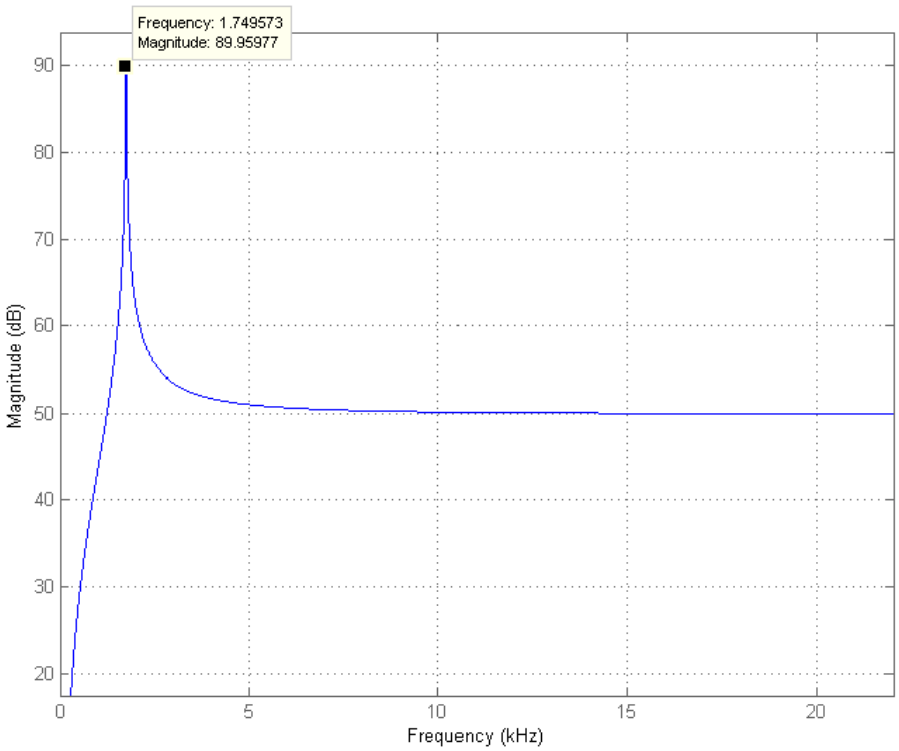

Figure 8.9: Wah-Wah Effect Band-Pass Filter.

```
ai = analoginput('winsound');
addchannel(ai,1);

ai.SampleRate = 44100;
ai.SamplesPerTrigger = 10*44100;
ai.TriggerType = 'Immediate';

start(ai);
[ll,t] = getdata(ai);
```

Figure 8.10: Method for recording from Guitar to ll variable.

output pairs for the ANFIS system. Our $x[n]$ input signal array is called $llclear$ and $y[n]$ output signal array is called $ddclear$ in our application. As a result, we train our input and output arrays, $llclear$ and $ddclear$ respectively to form our ANFIS nodal structure. There are two methods for training, when you want to train for ANFIS system: Sequential Training and Exhaustive Training. Compared to other training algorithms which inherits linear estimation such as ARX and Hammerstein-Wiener method, ANFIS relies directly on nodal nonlinear estimation of each input to output

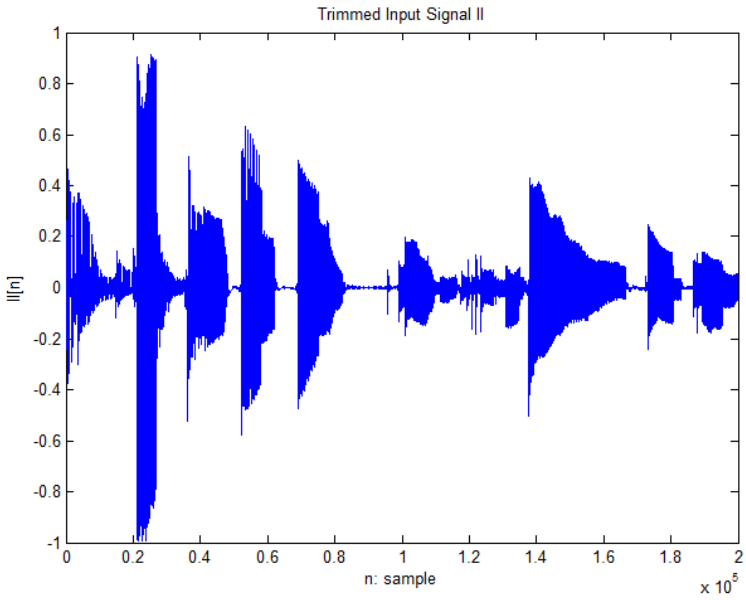

Figure 8.11: Trimmed and Scaled Input Signal ll.

pair. As mentioned earlier, even with training, we can decide on what kind of inputs we are giving to the ANFIS system.

Sequential training starts with trying one by one the each possible input array candidate, which are: $y[n-1]$, $y[n-2]$, $y[n-3]$, $y[n-4]$, $x[n-1]$, $x[n-2]$, $x[n-3]$, $x[n-4]$, $x[n-5]$, $x[n-6]$ respectively. For each of the candidate inputs, ANFIS training with sequential search $seqsrch$, calculates the RMSE error for training and checking data. In our input and output signal pair, we used first 75000 samples for training and the next 75000 samples for just checking the ANFIS output, making in total a 150000 length of signal. Below are the results for using just one input to the ANFIS system. Looking at the results, the best first input selection might be the feedback from the output: $y[n-1]$.

The second selection of input for sequential search follows the same rules that accepting that the $y[n-1]$ is the first input, it selects the best second input with respect to this decision and lists the RMSE according to all input candidates. The third input selection follows the same pathway and attaches the third input RMSE results for both training and checking data. As can be observed, in our application we select 3 inputs that gives us the best RMSE value from the 10 possible input candidates.

```
load('ll.mat');
M = 200000;
N = 100;
avgdelay = zeros(1,N);

ai = analoginput('winsound');
addchannel(ai,1);
ao = analogoutput('winsound');
addchannel(ao,1);

for k = 1:N

    ai.SampleRate = 44100;
    ai.SamplesPerTrigger = M;


    ao.SampleRate = 44100;
    set([ai ao],'TriggerType','Manual');

    putdata(ao,ll);
    start([ai ao]);
    trigger([ai ao])
    [dd,t] = getdata(ai);

%     plot(t,e);
%     zoom on

%     aitime = get(ai,'InitialTriggerTime');
%     aotime = get(ao,'InitialTriggerTime');
%     delta = abs(aotime - aitime);
%
%     delta_sample = 44100*delta;

    iocorr = xcorr(ll,dd);
    avgdelay(k) = find(iocorr==max(iocorr));
end

corr_avg = mean(avgdelay);
corr_sample = M - corr_avg;

%Test showed that Line 6 amplifier delay sample value is 1240

line6_lag = 1240/44100;
```

Figure 8.12: Simultaneous Input Output Code Block.

Another method for candidate input selection is to use exhaustive search method with the MATLAB function called $exhsrch$. This method differs from sequential search method with trying all combinations of 3 candidate inputs to find the triad that gives the best possible RMSE. This method is consuming in the sense of calculations, but effective when choosing the right input candidates for the system, hence every possible input combination is tried. The below listings show the exhaustive search trials

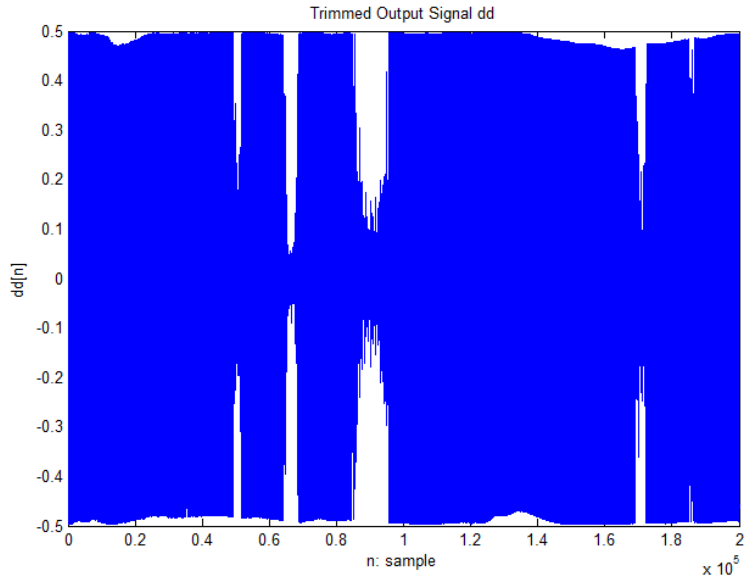

Figure 8.13: Trimmed and Scaled Output Signal dd.

```
Selecting input 1 ...
ANFIS model 1: y(k-1) --> trn=0.0758, chk=0.0861
ANFIS model 2: y(k-2) --> trn=0.1410, chk=0.1593
ANFIS model 3: y(k-3) --> trn=0.1928, chk=0.2165
ANFIS model 4: y(k-4) --> trn=0.2314, chk=0.2579
ANFIS model 5: u(k-1) --> trn=0.3627, chk=0.3099
ANFIS model 6: u(k-2) --> trn=0.3624, chk=0.3098
ANFIS model 7: u(k-3) --> trn=0.3621, chk=0.3099
ANFIS model 8: u(k-4) --> trn=0.3618, chk=0.3104
ANFIS model 9: u(k-5) --> trn=0.3614, chk=0.3111
ANFIS model 10: u(k-6) --> trn=0.3611, chk=0.3122
Currently selected inputs: y(k-1)
```

Figure 8.14: First ANFIS Input Selection Results.

for each input candidate triads. The best possible scenario for the minimization of the RMSE is to select inputs as $y[n-1]$, $y[n-2]$, $u[n-1]$ for this example. This means that our input signal consists of a matrix including feedback from the output and the previous input.

Looking at the RMSE values, the best choice for the input is the following model: ANFIS model = 6: y(k-1) y(k-2) u(k-6) –> trn=0.0427, chk=0.0474. This means that for our input selections we select two feedback signals coming from the output and input coming from the actual input signal delayed by 6, denoted by $u[n-6]$ in this

```
Train 36 ANFIS models, each with 3 inputs selected from 10 candidates.

ANFIS model = 1: y(k-1) y(k-2) u(k-1) --> trn=0.0428, chk=0.0476
ANFIS model = 2: y(k-1) y(k-2) u(k-2) --> trn=0.0428, chk=0.0476
ANFIS model = 3: y(k-1) y(k-2) u(k-3) --> trn=0.0428, chk=0.0476
ANFIS model = 4: y(k-1) y(k-2) u(k-4) --> trn=0.0428, chk=0.0475
ANFIS model = 5: y(k-1) y(k-2) u(k-5) --> trn=0.0428, chk=0.0475
ANFIS model = 6: y(k-1) y(k-2) u(k-6) --> trn=0.0427, chk=0.0474
ANFIS model = 7: y(k-1) y(k-3) u(k-1) --> trn=0.0523, chk=0.0581
ANFIS model = 8: y(k-1) y(k-3) u(k-2) --> trn=0.0523, chk=0.0581
ANFIS model = 9: y(k-1) y(k-3) u(k-3) --> trn=0.0523, chk=0.0581
ANFIS model = 10: y(k-1) y(k-3) u(k-4) --> trn=0.0522, chk=0.0581
ANFIS model = 11: y(k-1) y(k-3) u(k-5) --> trn=0.0522, chk=0.0581
ANFIS model = 12: y(k-1) y(k-3) u(k-6) --> trn=0.0522, chk=0.0581
ANFIS model = 13: y(k-1) y(k-4) u(k-1) --> trn=0.0601, chk=0.0675
ANFIS model = 14: y(k-1) y(k-4) u(k-2) --> trn=0.0601, chk=0.0676
ANFIS model = 15: y(k-1) y(k-4) u(k-3) --> trn=0.0600, chk=0.0676
ANFIS model = 16: y(k-1) y(k-4) u(k-4) --> trn=0.0600, chk=0.0676
ANFIS model = 17: y(k-1) y(k-4) u(k-5) --> trn=0.0600, chk=0.0676
ANFIS model = 18: y(k-1) y(k-4) u(k-6) --> trn=0.0600, chk=0.0676
ANFIS model = 19: y(k-2) y(k-3) u(k-1) --> trn=0.1012, chk=0.1120
ANFIS model = 20: y(k-2) y(k-3) u(k-2) --> trn=0.1011, chk=0.1121
ANFIS model = 21: y(k-2) y(k-3) u(k-3) --> trn=0.1011, chk=0.1121
ANFIS model = 22: y(k-2) y(k-3) u(k-4) --> trn=0.1010, chk=0.1121
ANFIS model = 23: y(k-2) y(k-3) u(k-5) --> trn=0.1010, chk=0.1121
ANFIS model = 24: y(k-2) y(k-3) u(k-6) --> trn=0.1010, chk=0.1120
ANFIS model = 25: y(k-2) y(k-4) u(k-1) --> trn=0.1148, chk=0.1275
ANFIS model = 26: y(k-2) y(k-4) u(k-2) --> trn=0.1148, chk=0.1276
ANFIS model = 27: y(k-2) y(k-4) u(k-3) --> trn=0.1147, chk=0.1277
ANFIS model = 28: y(k-2) y(k-4) u(k-4) --> trn=0.1147, chk=0.1278
ANFIS model = 29: y(k-2) y(k-4) u(k-5) --> trn=0.1146, chk=0.1278
ANFIS model = 30: y(k-2) y(k-4) u(k-6) --> trn=0.1146, chk=0.1278
ANFIS model = 31: y(k-3) y(k-4) u(k-1) --> trn=0.1611, chk=0.1774
ANFIS model = 32: y(k-3) y(k-4) u(k-2) --> trn=0.1610, chk=0.1776
ANFIS model = 33: y(k-3) y(k-4) u(k-3) --> trn=0.1609, chk=0.1777
ANFIS model = 34: y(k-3) y(k-4) u(k-4) --> trn=0.1609, chk=0.1779
ANFIS model = 35: y(k-3) y(k-4) u(k-5) --> trn=0.1608, chk=0.1780
ANFIS model = 36: y(k-3) y(k-4) u(k-6) --> trn=0.1608, chk=0.1780
```

Figure 8.15: Exhaustive Search ANFIS Input Selection Results.

context. As a result our input data matrix consists of $y[n-1]$, $y[n-2]$ and $x[n-6]$. These three input choices gives us the best results both in training and checking with the ANFIS algorithm. The input choices differs from application from application, the best results that are obtained from both checking and training data lead us to this choice. Future testing with other checking data is required to justify these choices.

The next step is the formulation of the ANFIS system for our input and output training pairs. In order to do this, we have to decide the number and kind of the member functions used in the ANFIS structure. The training data used for ANFIS structure generation includes $y[n-1]$, $y[n-2]$ and $x[n-6]$ respectively on the first three

columns, the last and the fourth column belongs to the training output column $y[n]$. We give this training data to the generate ANFIS input structure command input, $genfis1(trainingdata)$ and by default built a ANFIS structure of having 2 member functions of type 'gbellmf' which is Gaussian Bell Function. Then using this generated ANFIS input structure, we generate an output ANFIS structure to evaluate future inputs of any kind. In order to create this output ANFIS structure, we use the built in function [FIS,ERROR,STEPSIZE,CHKFIS,CHKERROR] = anfis(TRNDATA ,INITFIS ,TRNOPT ,DISPOPT ,CHKDATA) and as an output we get the output training output fis structure, training mean square error, step size, checking ouput fis structure and checking mean square error respectively. We have set decremental rate as 0.5 and incremental rate as 1.5 for this application and epoc is just 1 and can be increased for different applications. The ANFIS structure that we desired to find is the checking fis structure called $chkoutfismat$ in the structures. This structure contains everything about our ANFIS structure such as neural network rules, number and type of membership functions, input to output mappings. So let us take a look at the ingredients of $chkoutfismat$ and the relationship of our input selections with the output one by one in Figure 8.16.

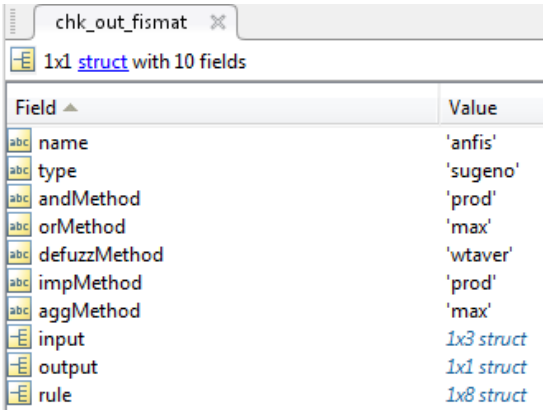

Figure 8.16: The attributes of the $chkoutfismat$ data structure.

As one can observe, the finalized system has 3 inputs $input1$, $input2$, $input3$ namely, representing $y[n-1]$, $y[n-2]$ and $x[n-6]$ respectively and the $output$ is just a single array $y[n]$.

What this FIS editor can provide us is the rules of the ANFIS system that is designed. For each of the membership function provided the inputs are weighted by the firing

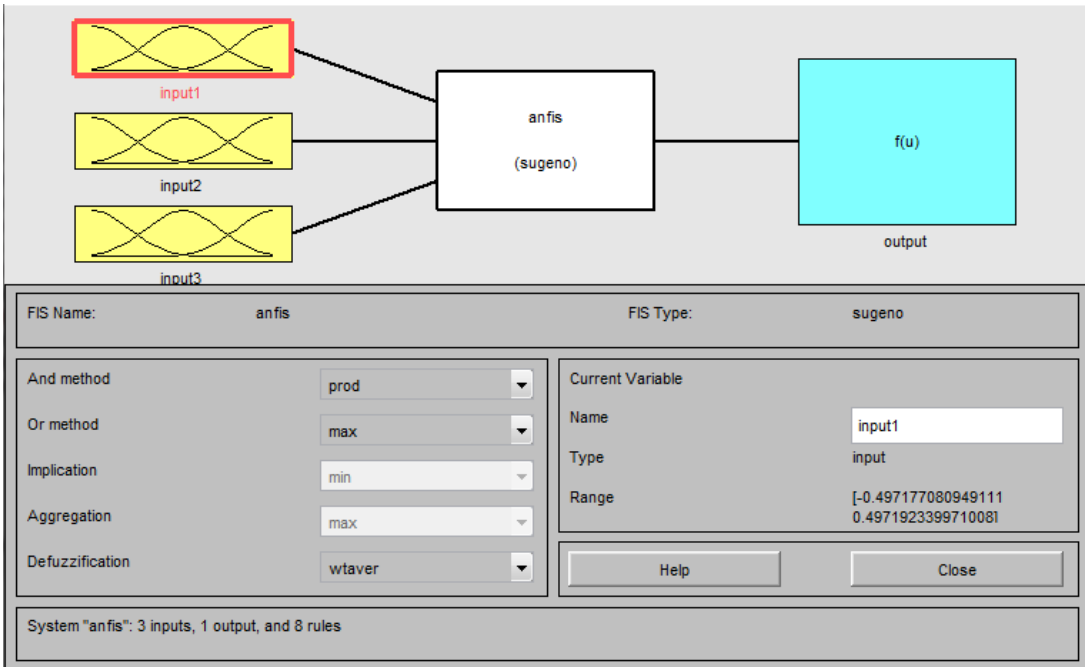

Figure 8.17: The ANFIS structure from the System Identification Toolbox of MATLAB.

strengths assigned to them, leading to a cumulative output. For every pair of inputs, $input1$, $input2$, $input3$, the output can be calculated by dragging and dropping the horizontal line assigned to each input. The rules are like the summary of the system, it resembles the kind of the relationship between the inputs and the outputs of the system.

ANFIS results into relationship between inputs and output of the system. These relationships can also be visualized as 3D surface plots between each chosen two input pairs and the output. The following surface plot graphs include the combinations the input pairs leading to the output value.

Having shown of all the combinations of 3D surface plots between inputs and output, we can finally go to the conclusions of ANFIS Method, where we will compare our method to the ARX Model Identification. Let us look how good the ANFIS prediction is visually. Figure 8.25 shows the comparison between the training, checking data, represented as solid blue line to the ANFIS predictions, represented as the green dots on the blue line original values. Figure 8.26 will give an insight detail zooming in the

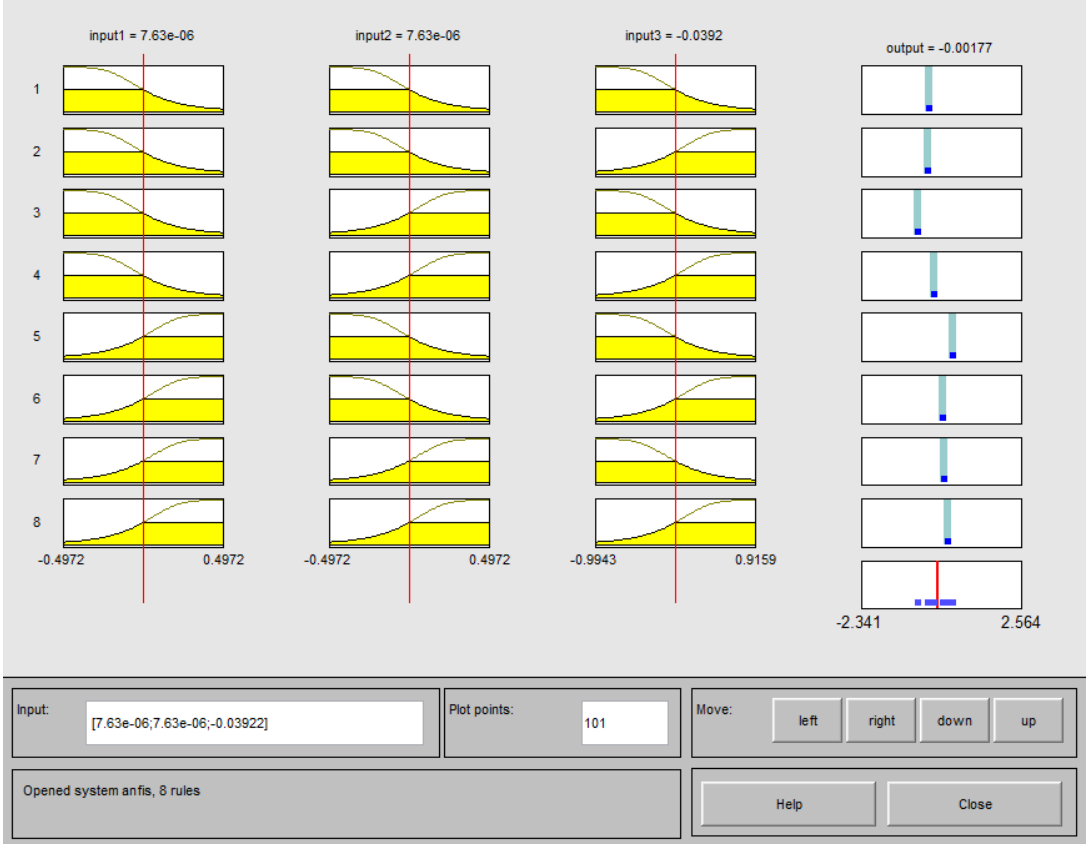

Figure 8.18: The rules of the ANFIS structure from the System Identification Toolbox of MATLAB.

neat details of the prediction on a sample interval.

The through look to the estimated output, green dots, one can see that the estimation is very successful, on the premise of avoiding the overshoots resulted from the high nonlinearity of the system. This in turn is expected due to the nature of the rectifiers, high nonlinear process on the input signal and this result can be seen in the zoomed in Figure 8.26. What is more overwhelming is the fact that, the estimated output signal, $y_{est}[n]$ sounds perceptually the same as the real output signal $y[n]$.

The RMSE values are our comparison values between training data input output pairs and checking data input output pairs. The lower and closest the training and checking RMSE values, the better the estimation performance. In our through experiments, we have observed quite close training and checking RMSE values, which is promising for a nonlinear estimation problem of this density. The RMSE calculation for both

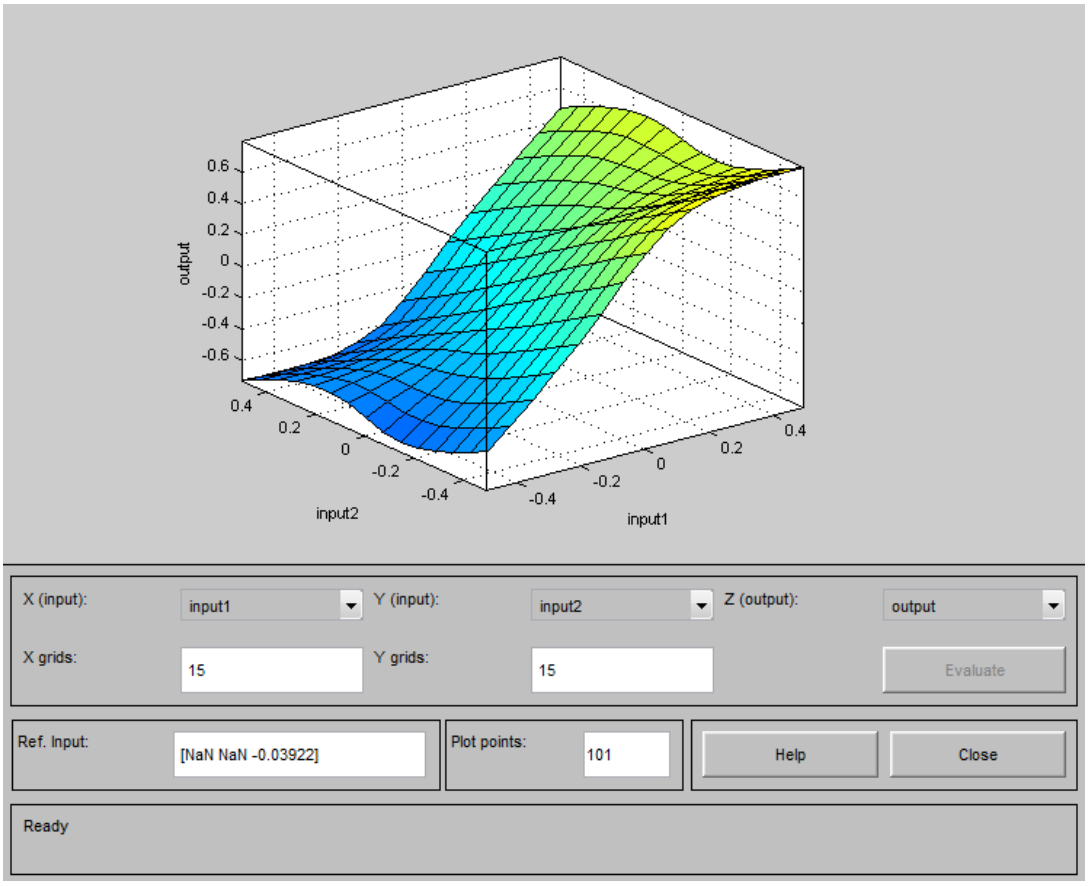

Figure 8.19: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

the training and checking outputs are done using the following formula. In addition to RMSE, we can normalize the RMSE value with the peak to peak value of the signal, where we divide the RMSE value with the difference between the maximum value of the output signal and the minimum value of the output signal. In our case, the maximum value of the output signal can be approximated as 0.5, and minimum value of the output signal can be approximated as -0.5. As a result, the range is 1 and RMSE and NRMSE values are the same for this implementation as shown in Equations 8.2 and 8.3.

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n} \left(y_{est}[k] - y[k]\right)^2}{N}} = 0.04 \tag{8.2}$$

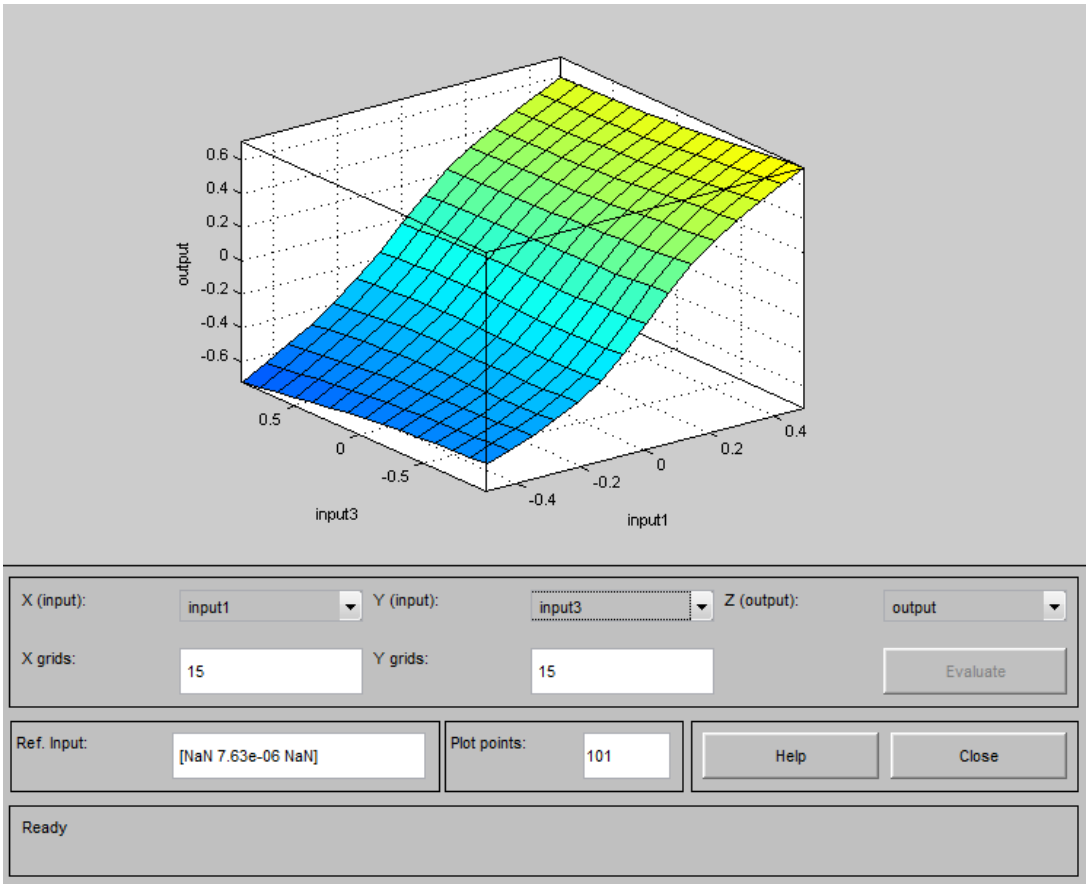

Figure 8.20: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

$$NRMSE = \frac{RMSE}{y_{max} - min} = \frac{\sqrt{\frac{\sum_{k=1}^{n} (y_{est}[k] - y[k])^2}{N}}}{1} = 0.04 \tag{8.3}$$

The RMSE value can be affected by many factors, such as the pre amplifier, power amplifier and speaker model of the amplifier at test. The system under test is a Line 6 Spider 4 Guitar Amplifier which has built in total amplifier and effects models working real-time. This is the general test for a nonlinear system identification model for such nonlinear guitar effects. So the RMSE is neat and enough in this application leading us to get even better results with only distortion and overdrive effects pedals under test without the amplifier. This ANFIS model even represents the dynamics of the amplifier such as its pre-amplifiers model, tone stage, power amplifier and distortion stage. For such complex signal pathways, this nonlinear system identification

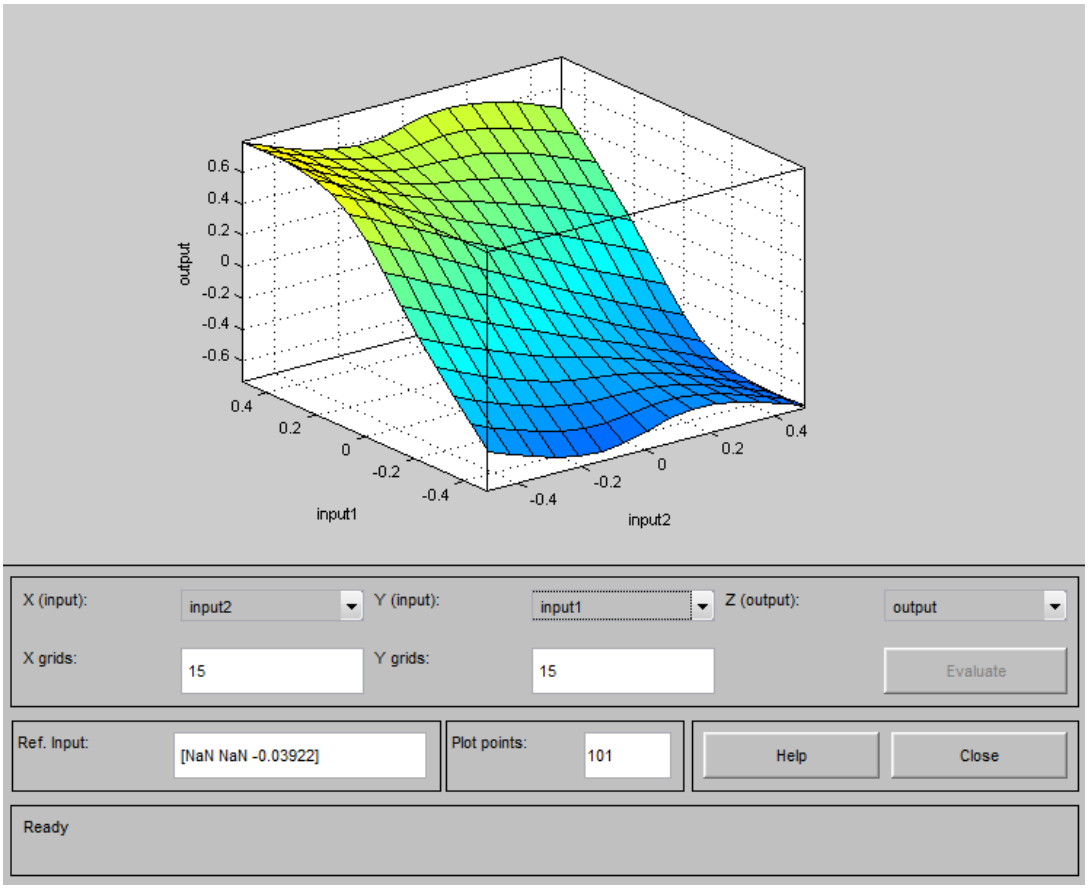

Figure 8.21: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

approach is better than other methods of modelling the nonlinearity. This structure can be complicated, but it is easy to implement and resembles the system as a black box which processes signals through its path. Even when we do not know the inner circuitry of the system at test, we can model its behavior under the tests that is present and find the future outputs as a consequence.

8.1.5 Enhanced Modelling of Guitar Distortion Algorithm Results

The input to this system $x[n]$, is the input directly from the electrical guitar, values ranging from -1 to 1 with 16 bits digital quantization and 44.1 kHz of sampling frequency. Later $x[n]$ is fed to nonlinear pre-amplifier, in this case we used 12AX7 simulation model as our pre-amplifier. The pre-amplifier circuitry, in most cases, in-

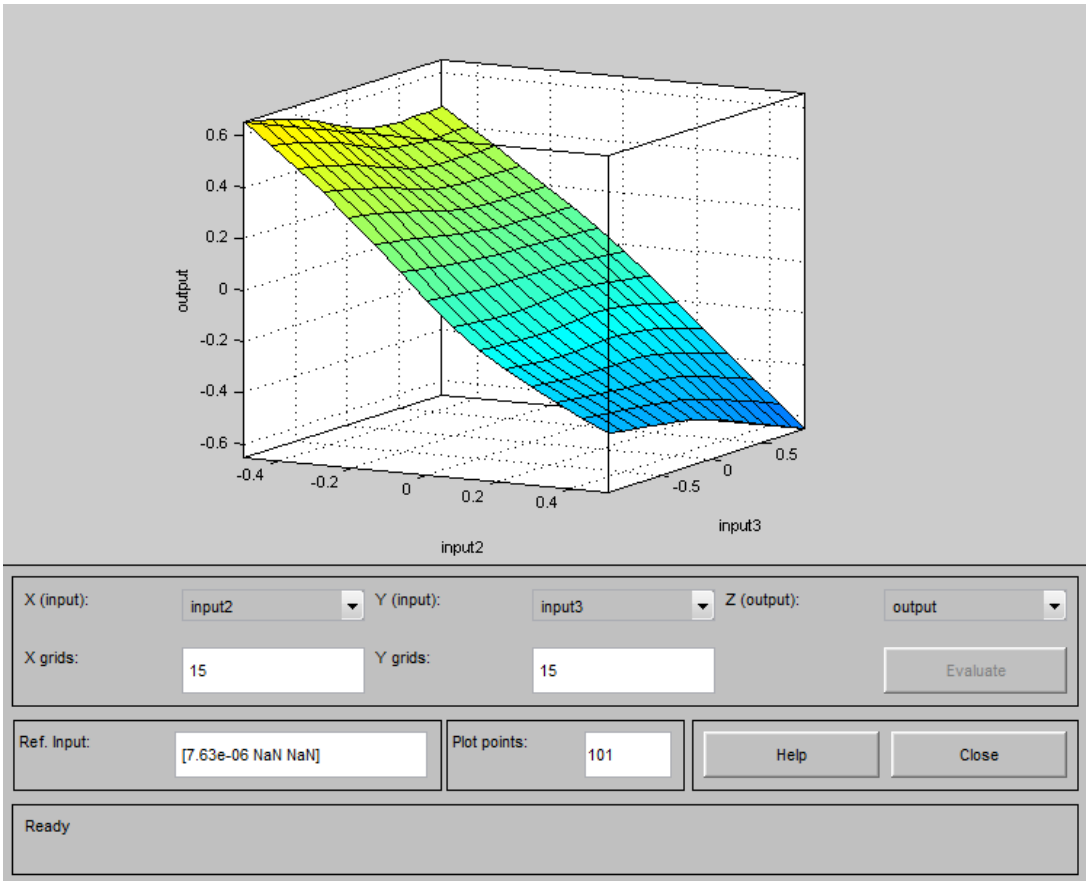

Figure 8.22: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

cludes potentiometers and tone/equalization circuitry, so we model those parts in the High-Pass Filter 12AX7 and Band-Pass Filter Tone parts. What this filters allow is parameterized settings for the user selected tones and characteristics to be modified later by the user. At the end of the first signal flow line, we obtain a clean channel (undistorted) sound with vacuum tube elements, which we use for the effects that uses clean channel (etc. Delay, Reverb, Flange, Wah-Wah).

The enhancement of distortion comes from the ANFIS System that flows the second signal flow path. The training input and output pairs are $x_a[n]$ and $y_a[n]$ respectively. This training sequence is obtained by recording a guitar training input $x_a[n]$ and then passing this input to the nonlinear state-space solution depicted in the Dempwolf and Zölzers paper [5] to obtain the training output $y_a[n]$. This training input output pair is pre-calculated before the ANFIS System Identification (Distortion) block is imple-

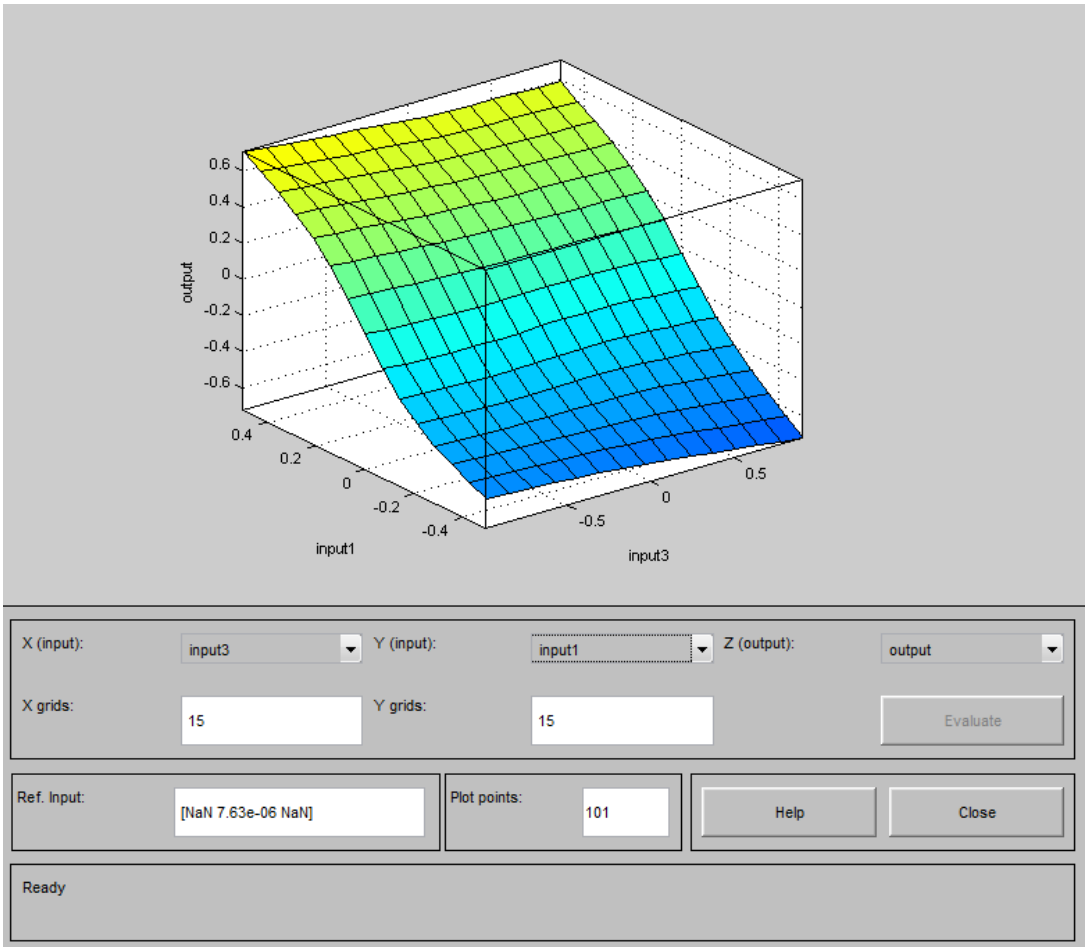

Figure 8.23: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

mented. This block takes the training pair and form a backward propagation Sugeno-Type neural network ANFIS model for the distortion or any nonlinear system that is processing input to output pair. As an output, system identification block has a special structured ANFIS model called $outfis$ that can be evaluated with external input to form new outputs with the rules coming from the FIS structure. As a result, the output, $y_l[n]$ becomes the amalgamation of the vacuum tube characteristics with the distortion block that is under concern. Calling $y_l[n]$ as the lead channel is appropriate, hence it is distorted with the system identification ANFIS approach.

The beginning of the ANFIS training starts with the formation of the input training array, what we call in the implementations as the $x_a[n]$ array. This array contains a randomly played song, which includes different guitar techniques, range of frequen-

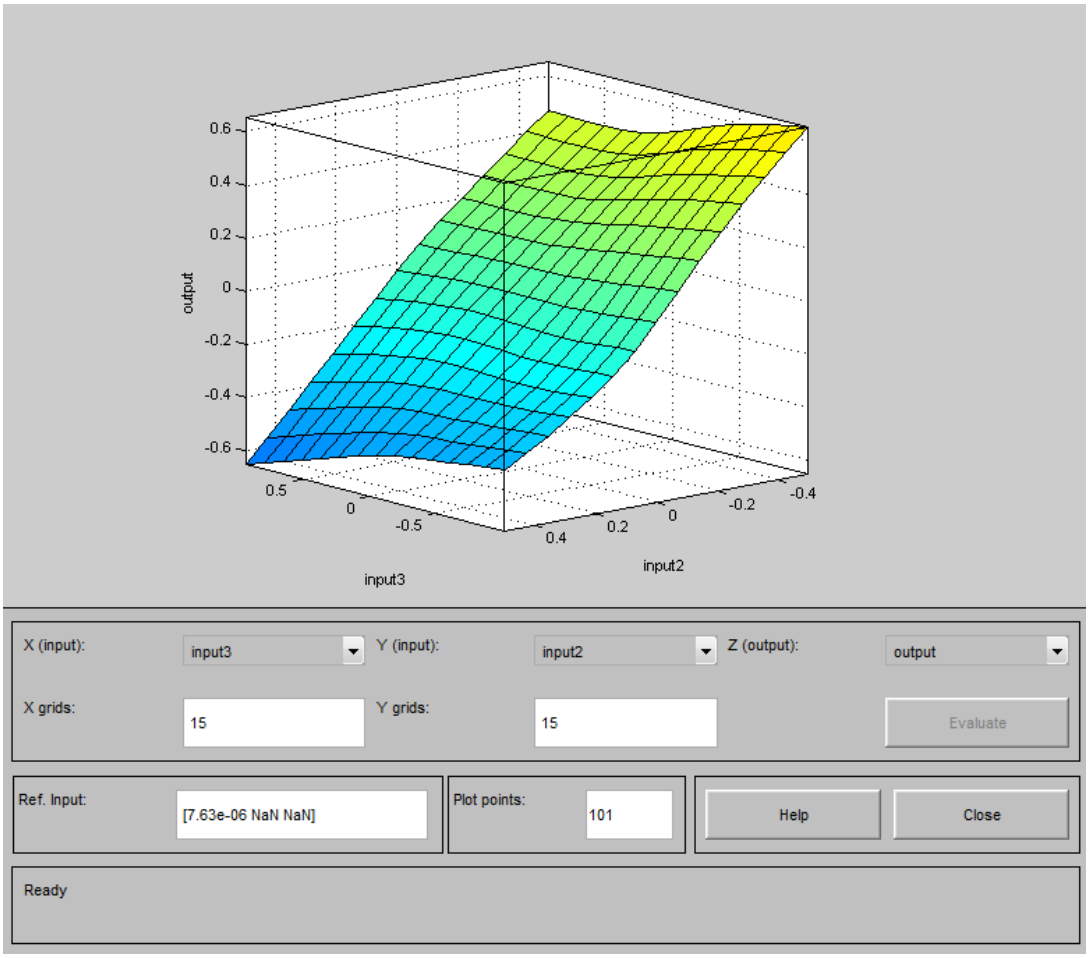

Figure 8.24: The 3D Surface Plot of the ANFIS structure from the System Identification Toolbox of MATLAB.

cies and amplitudes ranging from -1 to 1 to represent a random electric guitar input signal. Then the nonlinear state-space solution to the JCM900 rectifier part is applied as explained above and in the paper of Dempwolf and Zölzer' s [5] to obtain the output training pair called $y_a[n]$. Together the input and output pairs, $x_a[n]$ and $y_a[n]$ respectively are feed to the ANFIS System Identification block which forms ANFIS rules regarding the membership and epoch parameters to obtain the structured ANFIS system called $outfis$. The structure $outfis$ contains every bit of information regarding the ANFIS System designed by the training input and output pair, such that any new input to the ANFIS System can be evaluated with the $outfis$ structure to obtain the desired output.

The last part of the chain is the ANFIS Evaluator (Distortion), which is basically a

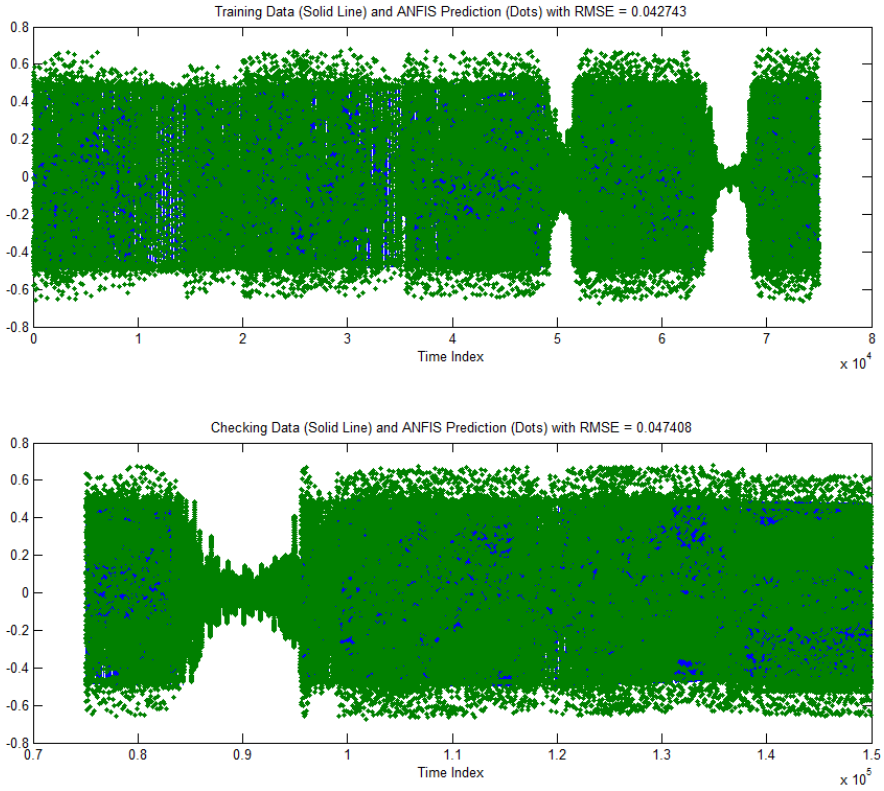

Figure 8.25: The Output Plot of the ANFIS structure.

neural network evaluator to form the desired output with using clean channel, $y_c[n]$ and the ANFIS System structure called $outfis$ to obtain the lead channel distorted output $y_l[n]$. As a result, the desired outputs $y_c[n]$ and $y_l[n]$, clean channel and lead-/distortion channel outputs are obtained using the Enhanced Modelling of Guitar Distortion algorithm.

To conclude this section, we will continue with detailed results analysis and music tests to see if we succeeded on enhancing the distortion in favor of putting some flavor into the effect. Let's start with comparing the log sine-sweep spectrograms and see the harmonics effects on the distorted signal in Figures 8.28 and 8.29. The input is a sinusoidal chirp signal which has an inreasing frequency characteristic from 1Hz to 3000Hz, electric guitar's general maximum frequency. The output is taken from the lead/distorted channel, $y_l[n]$.

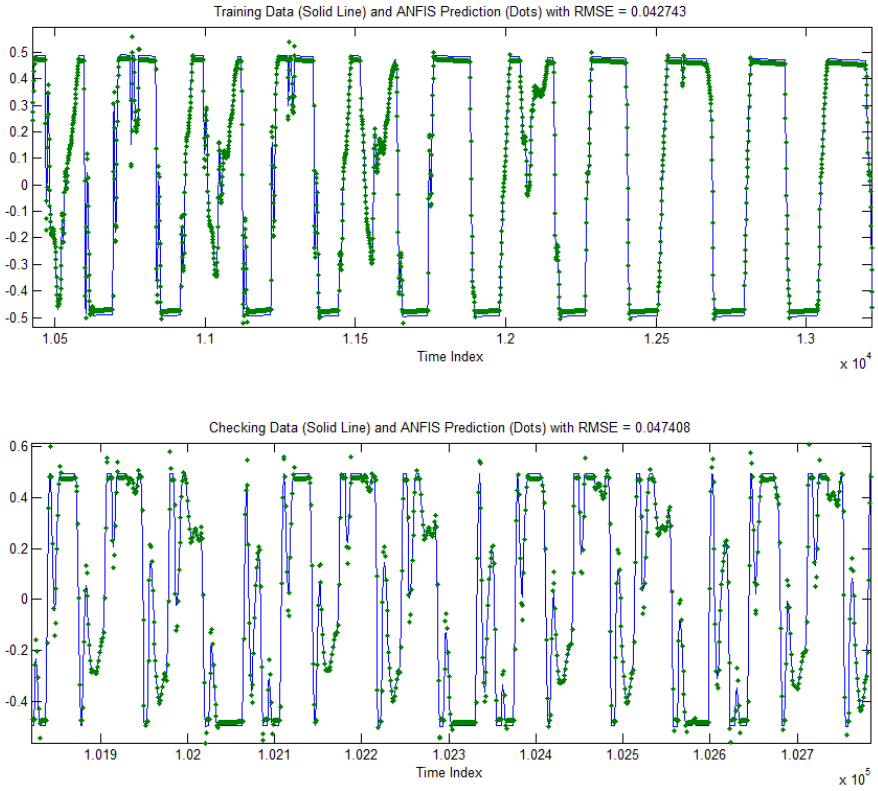

Figure 8.26: The Zoomed Output Plot of the ANFIS structure.

Clearly, the difference can be seen as a result of odd-harmonics starts to appear after the distortion effect has been applied to the input. This waterfall sine-sweep log spectrogram tells much about the distortion effect, as how the magnitude of the harmonics spread over the whole hearing range up to 22050Hz. Every single distortion unit have unique waterfall representations that represent the characteristics of the distortion provided. For our purposes, the music side evaluation of the resulting output should be done to finalize the results. When listened to the real-time outputs with playing the electric guitar as an input, one can tell the difference between the static wave-shaping and the ANFIS model of the distortion block. The static wave-shaping feels digital, while ANFIS output gives much organic, warm and bluesy sounding distortion effect while combining the gain to the input signal. The training input represents a portion of the JCM900 amplifier of the Marshall Inc., such that the resulting output represents the characteristics of a Marshall amplifier on sound and note articulation and feel.

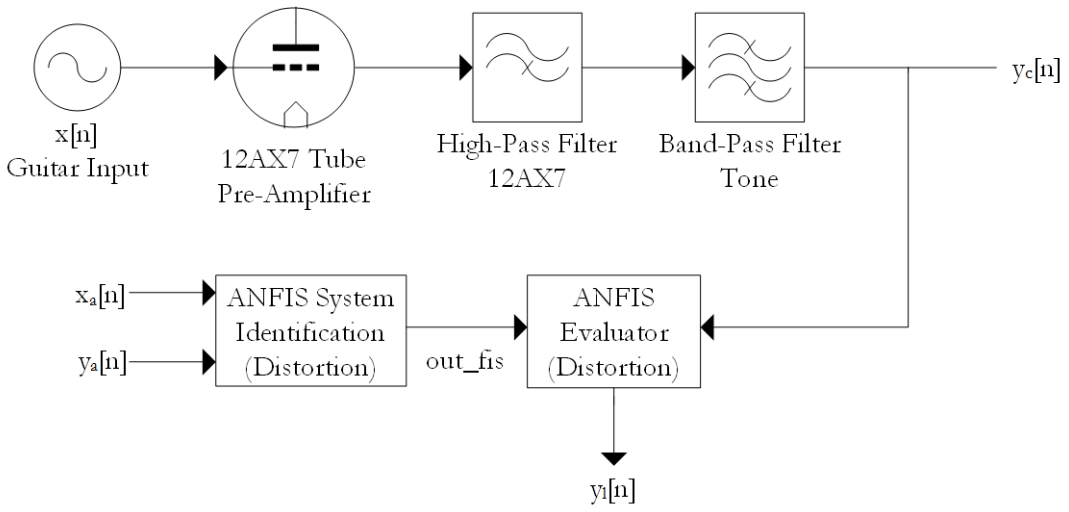

Figure 8.27: Enhanced Modelling of Guitar Distortion Model Basic Signal Flow Diagram.

Another result should be derived with comparing the nonlinear state-space solution to the ANFIS system identification and tell whether the blindfold black box system identification works good or not. One way to point out this is to compare the training and checking input pairs estimations with the ANFIS structure. In order to show this, we first feed the ideal system with an arbitrary sinusoidal signal, for our choice it is the A note of 440 Hz, called as the discrete-time signal, $a[n]$, shown below.

The arbitrary input signal first is passed from the idealized system, where we have directly solved the nonlinear state space equation and obtained the ideal solution represented as $y[n]$. The ANFIS solution would be expected to bring the best possible solution compared to the ideal output $y[n]$. The previously trained ANFIS system is trained with 30 membership functions having 50 epochs and used only the current input and the one previous input, represented as $a[n]$ and $a[n-1]$. The $evalfis$ command function is used with the resulting ANFIS structure $outfis$ and the other two parameters as the input to this function are $a[n]$ and $a[n-1]$. The ANFIS evaluated and estimated output called, $yanf[n]$ is very close to the ideal output $y[n]$, having a RMSE value of 0.023 with the length of 2000 samples. The underneath blue plot shows the ideal output $y[n]$ and the red plot shows the estimated output $yanf[n]$. There happened to be some minor overshoots, but in general the estimated output

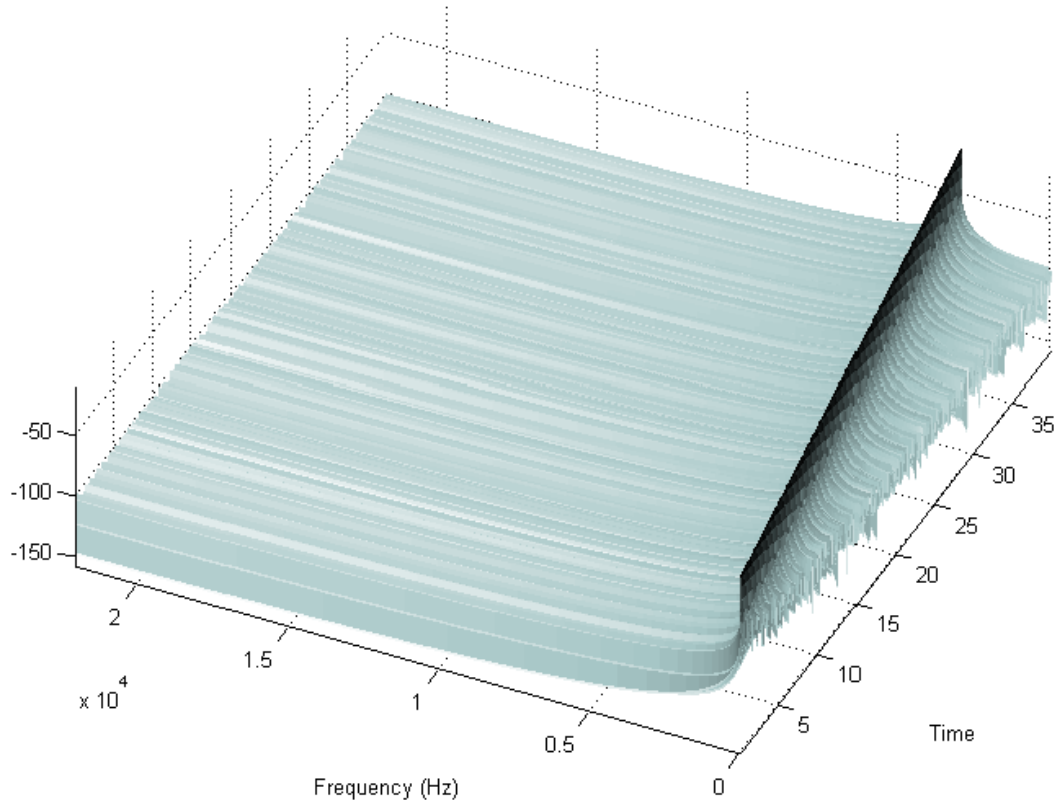

Figure 8.28: The log Sine-Sweep spectrogram of The Chirp Signal Ranging From 1 - 3000 Hz.

follows the ideal output.

To conclude this chapter, we present the relationship between the epochs and number of membership functions with the RMSE of the ANFIS System Identification method used inside the Enhanced Modelling of Guitar Distortion algorithm. In ANFIS structure epochs are the total number of forward and backward parameter updates for the premise and consequent parameters. Number of membership functions are defined in the first layer of the ANFIS structure and represent the fuzzification detail of the inputs. These are the two determining parameters of the simulation models of the ANFIS training structure. In most cases, higher epoch and number of membership functions mean lower RMSE. Low RMSE is always aimed to get similar results to the ideal nonlinear effect output. In Figures 8.32, 8.33 and 8.34, the relationship between RMSE, epochs and number of member functions is observed. The best possible choice to get the lowest RMSE value is to select epoch as 50 and number of

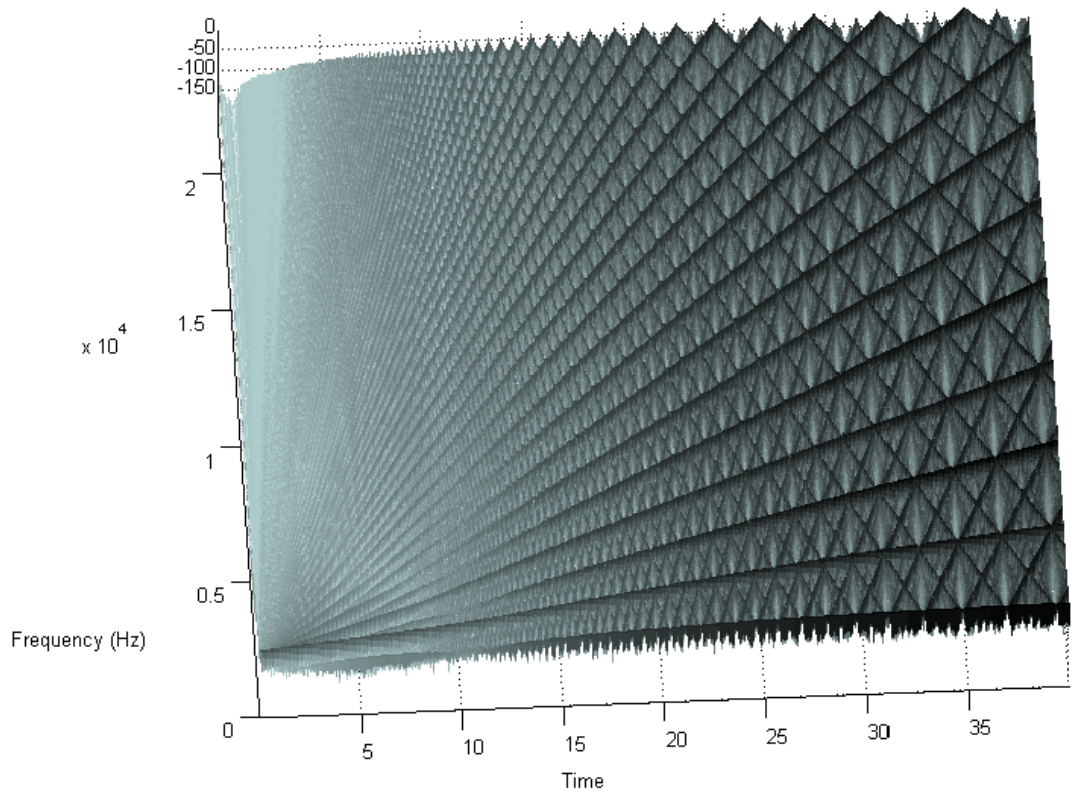

Figure 8.29: The log Sine-Sweep spectrogram of The Distorted Chirp Signal Ranging From 1 - 3000 Hz.

membership functions as 20. Higher epoch and number of membership functions can give better results, but also results into much higher computation complexity and parameters. There is a tradeoff between these values when chosing for the optimum epoch and number of membership functions.

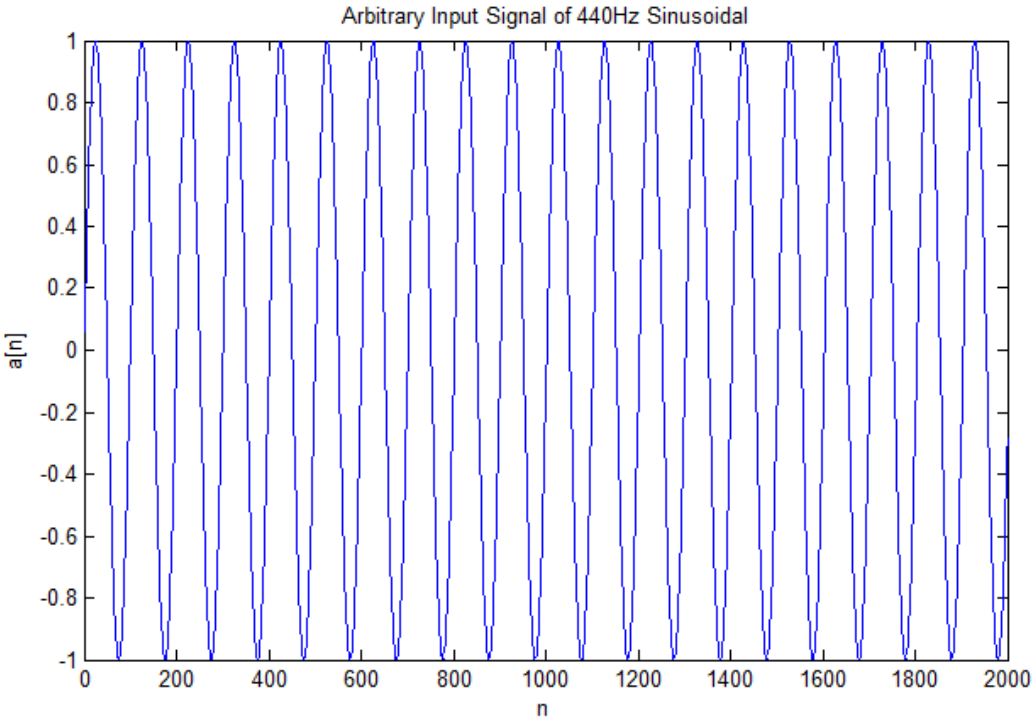

Figure 8.30: Arbitrary Input Signal of 440Hz Sinusoidal.

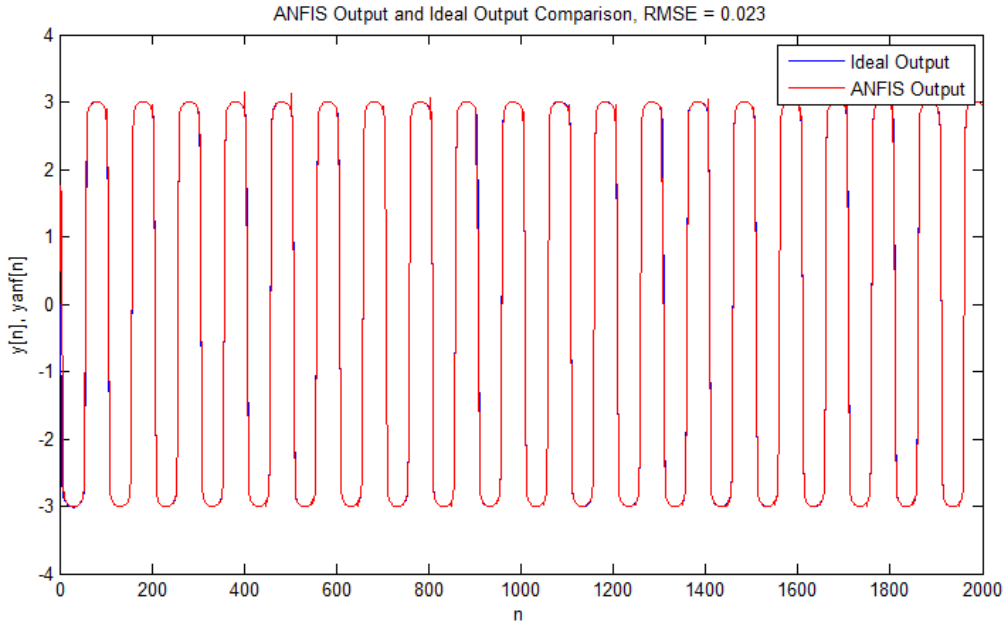

Figure 8.31: ANFIS Output and Ideal Output Comparison, RMSE = 0.023

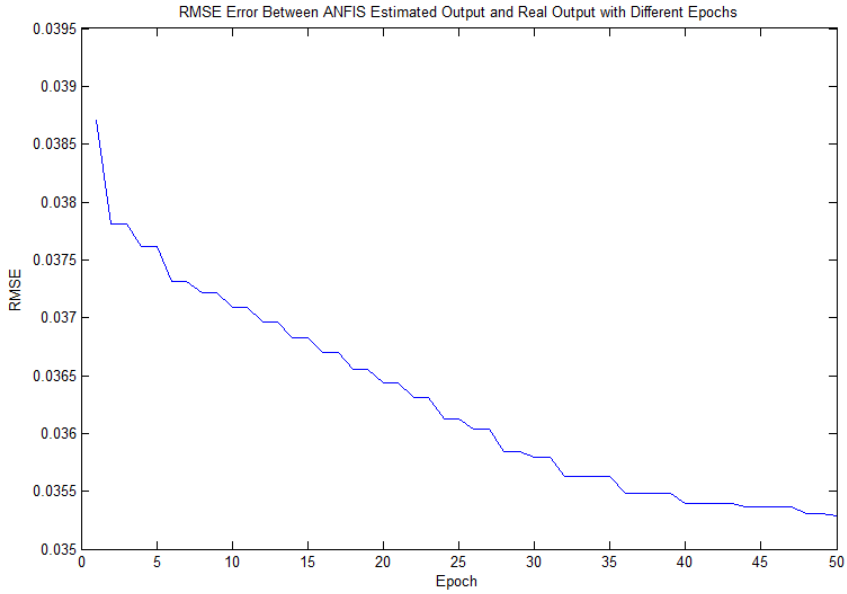

Figure 8.32: RMSE Error Between ANFIS Estimated Output and Real Output with Different Epochs.

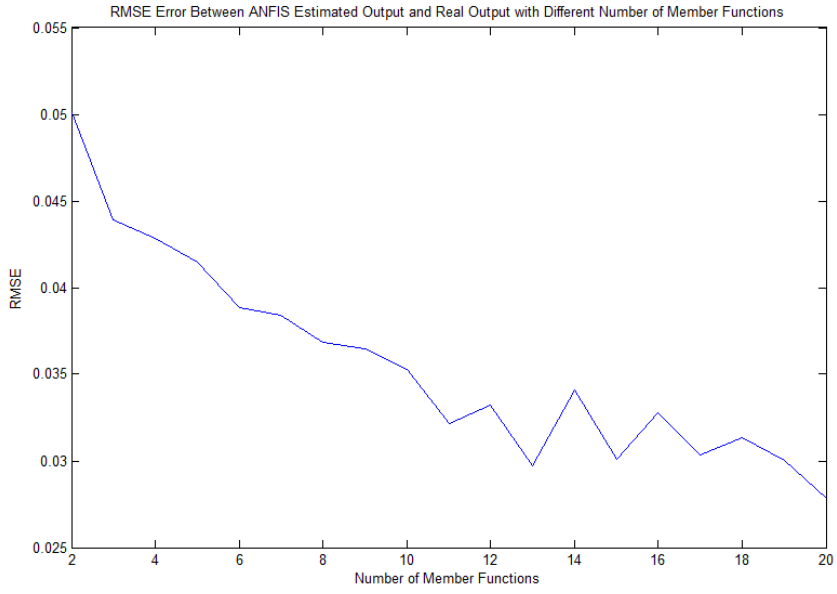

Figure 8.33: RMSE Error Between ANFIS Estimated Output and Real Output with Different Number of Member Functions.

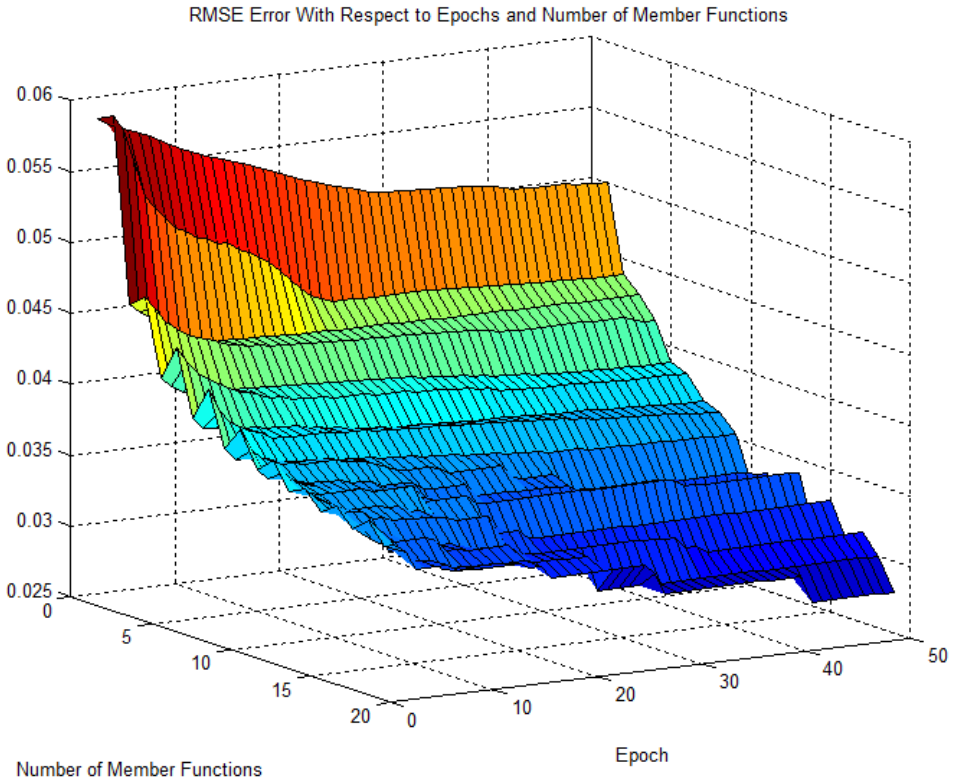

Figure 8.34: RMSE Error with Respect to Epochs and Number of Member Functions.

CHAPTER 9

CONCLUSION

9.1 Conclusion

In this thesis, we implemented delay, reverb, wah-wah, flanger and introduced a new method called Enhanced Modelling of Guitar Distortion. Each guitar effect is designed in $MATLAB$, implemented in Microsoft Visual Studio 2010 using C++ and C# and then ported to Texas Instruments TMS320VC5505 eZdsp. The chapters start with the introduction of the guitar effects and continue with the implementation details. Linear Guitar Audio Effects chapter includes delay, reverberation, flanger and wah-wah guitar effects theory and implementations. Starting from Basics of Guitar Distortion: Diode Limiter and Pre-Amplifier chapter, we introduced current methods for guitar distortion/overdrive modelling. We briefly mentioned Wave Digital Filter Modelling Approach of Guitar Distortion. System Identification Approach to Distortion Modelling chapter introduces a black-box modelling approach to distortion/overdrive. Using ANFIS: Adaptive-Network-Based Fuzzy Inference System as a system identification toolbox, we developed a new method called Enhanced Modelling of Guitar Distortion.

Previous methods mostly rely on solving directly the nonlinear state space solutions of internal circuitry of distortion/overdrive pedal. This thesis aimed to develop a black-box approach where no information is known about the internal circuit of distortion/overdrive. Black-box approach to distortion/overdrive modelling is preferable for digital guitar amplifier systems. Digital guitar amplifier systems include lots of different types of distortion/overdrive that can be easily modelled with the Enhanced

Modelling of Guitar Distortion algorithm. The equalizer band-pass filters are used to obtain desired tones from the linear and nonlinear guitar effects. We modelled Marshall JCM-900 pre-amplifier with Enhanced Modelling of Guitar Distortion algorithm. We obtained 64 samples of ASIO Buffer Size, which is the best real-time constraint to satisfy in ASIO implementations. The average RMSE value is 0.04, which is obtained with different membership function and epoch parameters. These algorithms are designed in $MATLAB$, implemented in Microsoft Visual Studio 2010 using C++ and C# and then ported to Texas Instruments TMS320VC5505 eZdsp. The results show that Enhanced Modelling of Distortion Algortihm provides good nonlinear modelling of guitar distortion/overdrive.

The developed guitar effects can be used professionally on Windows platform. DSP implementations can also be used as a standalone guitar effects processor. Total guitar amplifier models, based on modelling different nonlinear parts, can be developed using offered algorithms. A vacuum tube guitar amplifier consisting of numerous nonlinear parts such as the pre-amplifier and power-amplifier is introduced in Enhanced Modelling of Guitar Distortion chapter. It is recommend to implement different amplifier and distortion/overdrive nonlinear models using Enhanced Modelling of Guitar Distortion algorithm. This will allow further testing of the algorithm that is offered. MATLAB Fuzzy Logic Toolbox is used in the development stage of the nonlinear system identification models in this thesis. In the future iterations of the algorithm, a faster training and learning algorithm can be developed using C++. In this thesis, ANFIS method is used as a system identification algorithm. A different system identification tool can be used for obtaining different results. Among our tests, we found that ANFIS method suits best to our needs. Development of better neural network solutions can aid the Enhanced Modelling of Guitar Distortion.

The effects are tested also with listening tests. Even though technical results are enough to show that the algorithms are successful, listening tests are also important from musical perspective. Comparing the results of the designed guitar effect algorithms with current guitar effects processor softwares, we found that our effects provide virtual studio quality effects. The real time constraints are satisfied with equal or better real-time output using ASIO driver at 64 samples of buffer size. It is recommended for guitar effects developers to follow black-box approaches like Enhanced

Modelling of Guitar Distortion algorithm. This system identification approach allows the third party user to model numerous nonlinear overdrive/distortion effects just by recording input and output of the desired distortion/overdrive effect simultaneously. In the future, these algorithms can be ported to other platforms such as Mac Os X, Android and iOs. Mobile applications can be developed using Enhanced Modelling of Guitar Distortion algorithm. Even though iOs is much preferable for real-time audio input and output algorithms, Android' s Audio API is developing rapidly to allow low latency audio applications.

In conclusion, different approaches for implementing main linear guitar audio effects (delay, reverb, flanger and wah-wah) and main nonlinear guitar audio effects (distortion, overdrive) are presented and Enhanced Modelling of Guitar Distortion algorithm is introduced in this thesis. Theoretical background for modelling each effect is given in detail. Black box approach for modelling distortion and overdrive is introduced and implemented using ANFIS system identification tool. Enhanced Modelling of Guitar Distortion algorithm is formulated and implemented real-time. As a result, the distortion/overdrive effect has a warm and organic sound that is expected from a good distortion/overdrive effect. Even though nonlinear state space solutions provide direct results of the internal circuit of these effects, Enhanced Modelling of Guitar Distortion provides an alternative way of black box modelling to allow user to model their own distortion/overdrive effects.

REFERENCES

[1] J. O. Attia. *Electronics and Circuit Analysis using MATLAB*. CRC Press LLC, 1999.

[2] E. Battenberg and R. Avizienis. Implementing real-time partitioned convolution algorithms on conventional operating systems. *Proc. of the 14th Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23, 2011*, pages 1–7, 2011.

[3] S. A. Billings. *Nonlinear system identification : NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley and Sons, 2013.

[4] G. De Sanctis and A. Sarti. Virtual analog modelling in the wave-digital domain. *IEEE Transactions on Audio, Speech and Language Processing*, 18(4):715–727, 2010.

[5] K. Dempwolf, M. Holters, and U. Zölzer. Discretization of parametric analog circuits for real-time simulations. *Proc. of the 13th Conference on Digital Audio Effects (DAFx-10), Graz, Austria, September 6-10, 2010*, pages 1–8, 2010.

[6] K. Dempwolf and U. Zölzer. A physically-motivated triode model for circuit simulations. *Proc. of the 14th Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23, 2011*, pages 257–264, 2011.

[7] A. Fettweis. Wave digital filters: Theory and practice. *Proceedings of the IEEE*, 74(2):270–318, 1986.

[8] R. A. Gonzalo. *Nonlinear Signal Processing*. John Wiley and Sons, 2005.

[9] D. Houcque. Applications of matlab: Ordinary differential equations (ode). *Robert R. McCormick School of Engineering and Applied Science - Northwestern University*, pages 1–12.

[10] J.-S. R. Jang. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, May/June 1993.

[11] M. Karjalainen and J. Pakarinen. Wave digital simulation of a vacuum-tube amplifier. *ICASSP 2006*, (V):153–156, 2006.

[12] W. M. Leach J.R. Spice models for vacuum-tube amplifiers. *Journal of Audio Engineering Society*, 43(3):117–126, 1995.

[13] J. Macak and J. Schimmel. Real-time guitar tube amplifier simulation using an approximation of differential equations. *Proc. of the 13th Conference on Digital Audio Effects (DAFx-10), Graz, Austria, September 6-10, 2010*, pages 1–8, 2010.

[14] S. Möller, M. Gromowski, and U. Zölzer. A measurement technique for highly nonlinear transfer functions. *Proc. of the 5th Conference on Digital Audio Effects (DAFx-02), Hamburg, Germany, September 26-28, 2002*, pages 203–206, 2002.

[15] J. Pakarinen, M. Tikander, and M. Karjalainen. Wave digital modeling of the output chain of a vacuum-tube amplifier. *Proc. of the 12th Conference on Digital Audio Effects (DAFx-09), Como, Italy, September 1-4, 2009*, pages 1–5, 2009.

[16] M. R. Schroeder and B. F. Logan. "colorless" artificial reverberation. *IRE Transactions on Audio*, November-December(1):209–214, 1961.

[17] D. Sorlien and S. Keller. Introduction to tube amplifier theory: 10.02.15. *Featuring the AX84 P1-eXtreme Amplifier*, pages 1–38, 2009.

[18] D. T. Yeh, J. S. Abel, and J. O. Smith. Automated physical modeling of nonlinear audio circuits for real-time audio effects-part i: Theoretical development. *IEEE Transactions on Audio, Speech and Language Processing*, 18(4):728–737, 2010.

[19] D. T. Yeh, J. S. Abel, and J. O. Smith. Automated physical modeling of nonlinear audio circuits for real-time audio effects-part i: Theoretical development. *IEEE Transactions on Audio, Speech and Language Processing*, 20(4):1207–1216, 2012.

[20] D. T.-M. Yeh. *Digital Implementation of Musical Distortion Circuits by Analysis and Simulation*. PhD thesis, Stanford University, 2009.

[21] U. Zölzer. *DAFX: Digital Audio Effects, Second Edition*. John Wiley and Sons Ltd., 2011.

APPENDIX A

IMPLEMENTATION DETAILS

A.1 Implementation of FIR Delay and FIR Comb Filter

A.1.1 MATLAB Implementation

The FIR Delay in MATLAB is just a matrix shifting operation which is included also in FIR Comb Filter resembles the line of code like: $[zeros(1, dsample)\ x(1 : length(x) - dsample)]$;. Using this portion of the code summed up with the input signal x, one can obtain the code for the FIR Comb Filter, written as a function called $fircomb$ function for general use as below:

```
function [y] = fircomb(x,Fs,d,g)
%x = input
%Fs = sampling frequency(samples/sec)
%d = delay (secs)
%g = delay gain
%y[n] = x[n] + g*x[n-Fs*d]
y=x.';
d_sample = floor(Fs*d); %delay (samples)
if d>0 && d_sample<length(x)
    y = x + g.*[zeros(1,d_sample) x(1:length(x)-d_sample)];
end
end
```

Figure A.1: MATLAB Function of fircomb(x,Fs,d,g).

The parameters for this $fircomb(x, Fs, d, g)$ function are easily to figure out. x is the input signal read by the wavread function of MATLAB. F_s is the sampling frequency in Hz (samples/sec), which also comes from the [x Fs nBits] = wavread('sample.wav') command. d is the delay parameter in seconds, which is then converted to samples using F_s inside the function, and finally g is the gain parameter, generally between

0 and 1, but can also be larger than 1 in specific cases. The output of this function behaves as explained mathematically and figuratively in the previous chapters. The use and practicality of it is really easy.

A.1.2 C# Implementation

The solution in Visual Studio 2010 uses the ASIO SDK support by Steinberg to develop the basic real-time circular buffer to output the real-time processed audio. In order to use this SDK I have written a class called AudioFX under the AudioDSP namespace. AudioFX includes my own implementations for Distortion, Wah-Wah, Flanger, Delay, Reverb and etc. Among these functions $VFDelay$ does the FIR Comb Filter job and can be used as follows:

```
public static void VFDelay(Channel realIn, Channel realOut, double[] delay, float g)
{
    for (int k = 0; k < realIn.BufferSize; k++)
    {
        if (k < (int)delay[k])
        {
            realOut[k] = realIn[k];
        }
        else
        {
            realOut[k] = realIn[k] + g*realIn[k - (int)delay[k]];
        }
    }
}
```

Figure A.2: C# Function of VFDelay(realIn,realOut,delay,g).

The parameters in the C# implementation is nearly the same with the MATLAB. The main difference is the sample by sample processing of the VFDelay function while MATLAB does total array processing. Due to this reason the parameters $realIn$ and $realOut$ are input and output float samples. d in this case delay amount in terms of samples not seconds. g is again the gain parameter, generally between 0 and 1, in specific cases can be larger than 1.

A.2 Implementation of Audio Flanger

A.2.1 MATLAB Implementation

In order to implement Audio Flanger in a computer environment, one do have to implement a function having the variable delay functionality. This can be programmed dynamically for real-time purposes, but for MATLAB environment, we do not need dynamic programming. I have written a basic function to shift each input signal from input x, with the array of delay, having values for how much shift will each input sample get, called $delay$ parameter. The important point in this function is to zero the input signal where the delay value for that sample is larger than the index of it. This is due to fact that we do not have information about that much past values for that signal. The function called $vdelay2(x, delay)$ can be used for that variable delay purposes inside of the audio flanger implementation.

```
function [ y ] = vdelay2(x,delay)
%x:input
%delay:time-varying delay array
N=length(x);
y=zeros(1,N);
    for k=1:N
        if k <= delay(k)
            y(k)  = 0;
        else
            y(k) = x(k-delay(k));
        end
    end
end
```

Figure A.3: MATLAB Function of vdelay2(x,delay).

As described in previous chapters theoretically, first we have to define the desired shaped variable delay array with the pre-defined parameters than put this delay array into the $vdelay2$ array. The procedure to create the suitable variable delay is given as in the sample MATLAB code where regular own delay function is used with index k to get the desired delay value at that specific index and return it to the output variable of the function $vdelay2$. Next, we sum up the output of the $vdelay2$ function to the input $x[k]$ at that instance to obtain the output of the flanger.

A.2.2 C# Implementation

Real-time constraints requires ASIO 2.3 drivers to be used in the implementation process. Audio Flanger is written under the AudioDSP namespace and AudioFX class, having the usage method of **Flanger(float RealIn, int flange_s, int flange_max)** . This method takes the $RealIn$: the input sample at that time, $flange\_s$: the changing index of the delay, $flange\_max$: the maximum of the changing index of the delay. In order to use this function, first the user have to choose what kind of changing delay function to use. For our purposes, we have chosen the most general sinusoidal delay array structure for our $FlangeBuffer$ function as shown below.

```
//FLANGE SINUSOIDAL DELAY ARRAY
private static void FlangeBuffer(int flange_param)
{
    for (int q = 0; q < flange_param; q++)
    {
        flange_delay[q] = (int)System.Math.Floor(M_flange + M_flange * System.Math.Sin(2 * System.Math.PI * F_flange * q * (1 / Fs)));
    }
}
```

Figure A.4: FlangeBuffer Function to Form Sinusoidal Changing Delay Index.

Following the definition of the changing delay array, we pass each member of the changing delay array and the input sample from the **Flanger(float RealIn, int flanges, int flange_max)** function. The following code block uses circular buffer concurrent queue to make it happen the changing delay implementation which is the core of the flanger audio effect. $TryDequeue$ and $Enqueue$ are the functions to pull and push the First In First Out system of the circular buffer in order to maintain the changing delay mechanics. The user have to input the $flange\_max$ parameter as to set the maximum value of delay that the flanger system introduces to buffer. The code block of the AudioDSP function can be seen as the following:

The test results satisfy the needs for a flanger effect, where a sinusoidally changing delay parameter is implemented. Different kind of delay arrays can be defined mathematically for different needs, such as jet airplane effects or othe customized flanger used in funk, rock and hard rock. Real-time needs are tested with the help of ASIO support with interoperable code pieces. In order to implement a good flanger effect, one needs 5 to 20 MB RAM to supply the needs for the concurrent circular buffer queue and to operate a nice real-time constraint, the user should have decre-

```
//FLANGER IMPLEMENTATION
public static float Flanger(float RealIn, int flange_s, int flange_max)
{
    FlangeOut = RealIn;

    if (flange_cnt < 2*flange_max)
    {
        fq.Enqueue(RealIn);
        flange_cnt++;
    }
    else
    {
        fq.TryDequeue(out flange_data);
        flange_data = fq.ElementAt(flange_s);
        FlangeOut = RealIn + flange_data;
        fq.Enqueue(RealIn);
    }

    return FlangeOut;
}
```

Figure A.5: Flanger Function implement under the AudioDSP namespace in AudioFX Class.

ment BufferSize to 64 in the ASIO Control Panel.

A.3 Implementation of Reverberation

A.3.1 C# Implementation

Similar to Flanger effects implementation reverberation also uses the concurrent circular buffer queue implementation, the difference is that reverberation does not include changing delay parameter, but includes feedback component where we enqueue and dequeue from the circular buffer each time the code block is called. Let us look at the code block and give the brief explanation regarding the parameters and how it does the reverberation.

Implementing reveberation in real-time needs two concurrent circular buffer queues, one for storing past samples of the input $x[n]$ and the other for storing information about output $y[n]$. These two concurrent circular buffers are represented as the parameters, rg_x and rg_y respectively. The delayed input and output samples, $x[n-m]$ and $y[n-m]$ respectively, are dequeued from the circular buffers into the floating point parameters, $delay\_data\_rb$ and $delay\_data\_rb\_y$, respectively. The parameter

```
//REVERB IMPLEMENTATION
static ConcurrentQueue<float> rq_x = new ConcurrentQueue<float>();
static ConcurrentQueue<float> rq_y = new ConcurrentQueue<float>();
static float RealOut_rb = 0;
static int reverb_cnt = 0;
static int delay_s_rb = 0;
static float delay_data_rb = 0;
static float delay_data_rb_y = 0;
//Reverb Implementation with delay parameter: m and gain parameter: g
public static float Reverb(float RealIn, float m, float g)
{
    RealOut_rb = RealIn;
    delay_s_rb = (int)(2 * 44100 * m);

    if (reverb_cnt < delay_s_rb)
    {
        rq_x.Enqueue(RealIn);
        rq_y.Enqueue(RealOut_rb);
        reverb_cnt++;
    }
    else
    {
        rq_x.TryDequeue(out delay_data_rb);
        rq_y.TryDequeue(out delay_data_rb_y);
        RealOut_rb = -(float)g * RealIn + delay_data_rb + (float)g*delay_data_rb_y;
        rq_x.Enqueue(RealIn);
        rq_y.Enqueue(RealOut_rb);
    }

    return RealOut_rb;
}
```

Figure A.6: Reverberation Code Block Written in C#.

called $delay\_s\_rb$ is used to calculate the parameter m in samples, where the function takes the parameter m in seconds to make it much easier for the user. As we have two channels for the stereo, we multiply the seconds with two times the sampling frequency, 44.1 kHz. The first thing the code does is to fill the buffers until the counter, $reverb\_cnt$, reaches the sample number of the input delay. After the first filling the buffer is completed, it is time to dequeue from the both queues, the values of the delayed input and output samples and implement the reverberation equation, with the decaying gain parameter g. We close that case part with the enqueuing to the circular buffer the latest inputs coming from the ASIO drivers, and returning the calculated output value, $RealOut\_rb$.

A.4 Implementation of Wah-Wah Effect

A.4.1 C# Implementation

When heared to the wah-wah output file, one can hear the changing bandpass frequency filter and the speed of the change by ear. These can be adjusted changing the wave function of the band-pass center frequency, f_c and damping factor d as desired. But the real challenge is to implement this algorithm real-time using C# on Windows Applications and C/C++ on the implementation on eZDSP board by Texas Instruments. I will first continue with the C# implementation that results into Windows Application on C#. The following code block function does the wah-wah job with the parameters $realIn$, F_c, F_s, floating point value of the sample input, bandpass-center frequency and sampling frequency chosen generally as $44.1kHz$ respectively. What this function does is that it takes the input sample at that real-time instance and calculate the state variable equations and store the previous $y_b[n-1]$ and $y_l[n-1]$ in the parameters $ybpast$ and $ylpast$ respectively in the function. With each followinf input and calculated state variable equation, these stored values from the previous iteration is updated accordingly, since the system includes one previous state variables inside.

```
public static float Wahwah(float realIn, float Fc, int Fs)
{

    yh = realIn - ylpast - Q1 * ybpast;
    yb = F1 * yh + ybpast;
    yl = F1 * yb + ylpast;

    F1 = (float)(2*System.Math.Sin((System.Math.PI * Fc) / (float)Fs));

    ybpast = yb;
    ylpast = yl;

    return yb;
}
```

Figure A.7: Wah-Wah Effect Code Block In C#.

Using this $Wahwah$ function in the main block of the program is crucial and the library to use this function is under the AudioDSP namespace AudioFX class. The line to call the function to read the input[index] sample then simultaneously writes

back to input[index] using the function with the previously defined variables: AudioDSP.AudioFX.Wahwah(input[index], input[index], Fc, Fs, d);. This line should be enrolled in the circular buffer read and write sequence. The successful real-time execution of this code block is only possible using Steinberger ASIO SDK to directly reach to the sound card drivers and possible with the common onboard sound cards like the one that I use on my desktop computer, Realtek High Definition Audio and also Creative Sound Blaster Z which both have ASIO support built into the software that comes with them. Selecting the necessary sound card driver will let the ASIO to run the code and give real-time Wah-Wah output to the user. I tested the system built with my own electric guitar, where I jacked in my electric guitar, Dean ML79F to the Line In port and take the output from the Line Out to my 2.1 speaker system and enjoyed the result with the output got as I desired.

When we come to the last part, the eZDSP programming, it was a little bit exhausting since I come with the idea to define each filter by hand as enumerated parameters, since it became consuming to calculate filter parameters right before the filtering process made the system non-real time. I found the solution with pre-defining a set of filter parameters related to the different filters with different desired center frequencies, f_c as shown below with some example filter parameters. The parameter definitions for the IIR filter numerator and denominator parameters are as shown below. The parameters are B0, B1/2, B2, A0, A1/2, A2, where B0, B1/2 and B2 are the numerator coefficients, A0, A1/2 and A2 are the denominator coefficients.

```
const signed int shelving_001[6] = {2111,-1058,7,32767,-31753,30770};
const signed int shelve_101[6] = {2111,-1051,7,32767,-31518,30770};
```

Figure A.8: Wah-Wah Shelving Filter Parameters for Min and Max Band-Pass Frequency.

The MATLAB output of both parameter lines for the maximum f_c and minimum f_c are shown as below:

As can be seen from previous two plots, we apply shelving filters ranging desiredly from f_cmin to f_cmax. In our choice of triangular wave pattern f_c ranges up from f_cmin to fc_max, then fc_max to f_cmin periodically with time. What this normalized frequencies correspond to in our case is that if we multiply the normalized frequencies

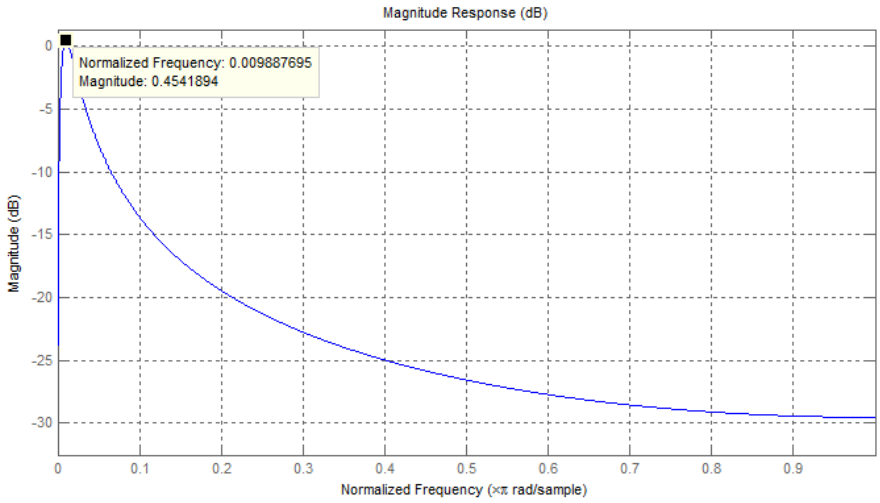

Figure A.9: Wah-Wah Shelving Filter for Min f_c.

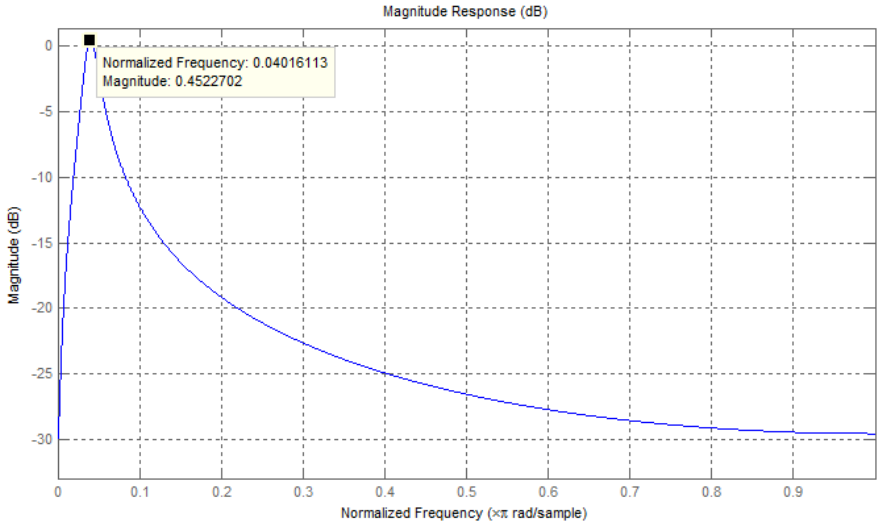

Figure A.10: Wah-Wah Shelving Filter for Max f_c.

by the half of the sampling frequency (44.1kHz) we obtain where in really the filtering operation proceeds. For $f_c min_{real} = 22050Hz * 0.009887695 = 218.0237Hz$ and $f_c max_{real} = 22050Hz * 0.04016113 = 885.5529Hz$. So we can summarize that for Auto-Wah guitar audio effect we have a periodic triangular wave ranging between 218Hz and 885Hz for the band-pass frequency f_c of the shelving filter.

The original wah-wah effect introduced was the Original Crybaby GCB95 by Jim Dunlop. In order to compensate and compare the filters and bandpass frequencies, I fed the input of the guitar effects pedal with white noise and measured the output to estimate the power spectral density and min max points of the real filters. Let's look at the min and max bandpass shelving filters of the real guitar pedal as follows:

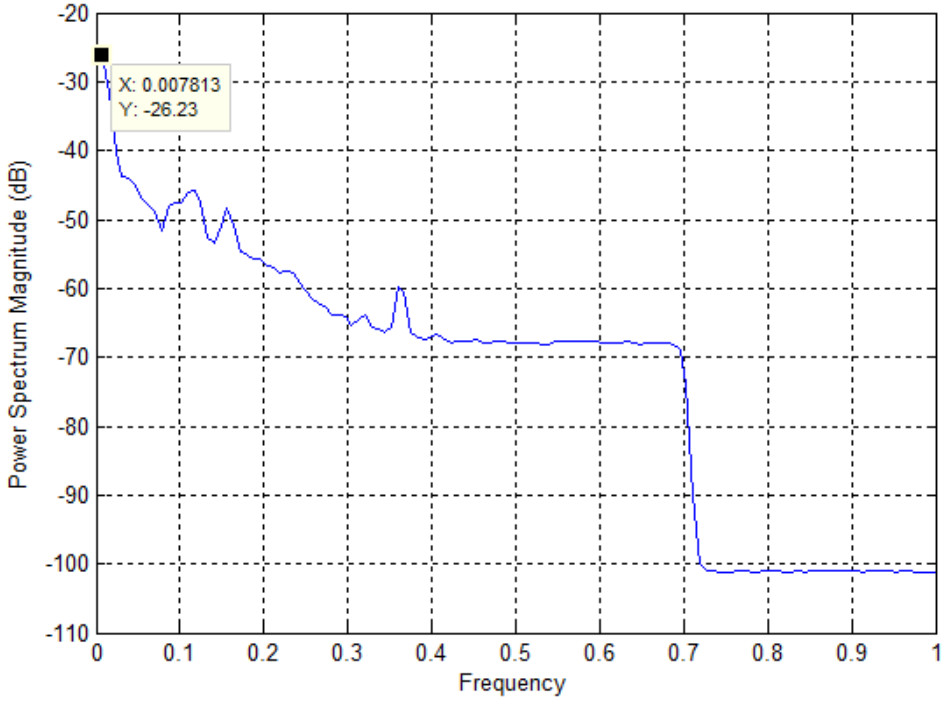

Figure A.11: Original Crybaby GCB95 Wah-Wah Shelving Filter for Min f_c.

As can be seen from the previous two graphs, the Original CryBaby from Jim Dunlop basically does the time variable shelving filtering on the input signal. The calculations with the sampling frequency F_s shows us the shelving filters that we have designed are the ideal approximations to the original guitar wah pedal GCB95. The sound tests show the same results with the pleasure of the guitar player with the implementation of the Auto-Wah pedal. When we convert the normalized frequencies to the continuous frequency values we will see how similar the min and max frequencies. $f_c min_{CryBaby} = 22050Hz * 0.007813 = 172.2767Hz$ and $f_c max_{CryBaby} = 22050Hz * 0.04688 = 1033.7040Hz$. These frequencies are really similar to our filters designed in MATLAB and implemented inside the Texas Instruments eZdsp

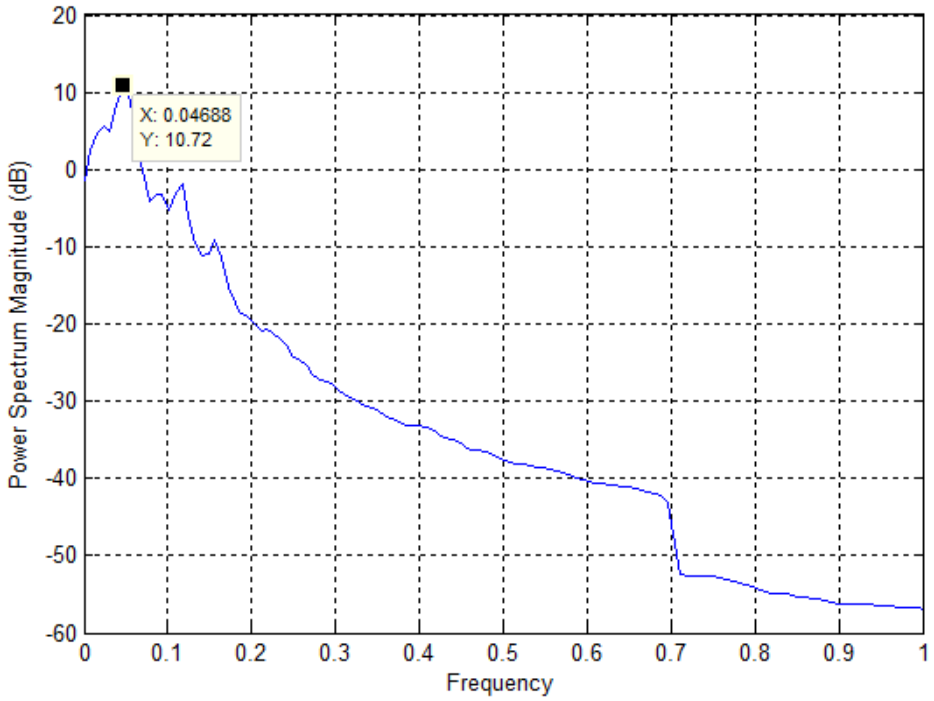

Figure A.12: Original Crybaby GCB95 Wah-Wah Shelving Filter for Max f_c.

coded using the Code Composer Studio.

Shelving Filter concept used in the Wah-Wah design process is the critical point of designing a good and desired Wah-Wah effect. In our case we have used the shelving filter concept referred in the Second Edition DAFX: Digital Audio Effects reference book. If we call the numerator parameters as the b vector and the denominator parameters as the a vector. The shelving filter equations calculates the denominator and the numerator of the IIR band-pass filter that is desired to be used. The equations are as follows, where additionally f_b is the bandwidth of the shelving filter:

$$V_0 = 10^{\frac{G}{20}} \tag{A.1}$$

$$H_0 = V_0 - 1 \tag{A.2}$$

$$d = -cos(\frac{2 * \pi * f_c}{F_s}) \tag{A.3}$$

$$ab = \frac{tan(\frac{\pi * f_b}{F_s}) - V_0}{tan(\frac{\pi * f_b}{F_s}) + V_0} \tag{A.4}$$

$$a = [2 + H_0 + H_0 * ab \; 2 * (d - d * ab) \; - H_0 - ab * (2 + H_0)] \tag{A.5}$$

$$b = [2 \; 2 * (d - d * ab) \; 2 * ab] \tag{A.6}$$

The shelving filter design procedure gives us the parameters for the denominator and numerator of the desired band-pass IIR filter with the specified f_b , f_c and G as bandwidth, center cut-off frequency and the gain. We then take this parameters, apply it to the input electric guitar signal and test the feel of it, then decide on the range of the f_c and time-variable change it to obtain the wah-wah effect basically. Using eZdsp by Texas Instruments, a generic code block is used to process the input signal sample by sample, if applicable with the real-time constraints. In the wah-wah code block, in order not to compute each time-changing shelving filter parameters again and again, I have pre-calculated the necessary filter parameters that are used frequently by the Wah-Wah system and write them into the memory and call them when needed to process the signal with the desired signal.