Analyse von Waveshaping mittels Tetration

Bachelorarbeit aus Musikalische Akustik 1, SE

Wolfgang Beucher

Betreuung: Dr. Robert Höldrich

Graz, 10. Januar 2017

kunst uni graz

institut für elektronische musik und akustik

Zusammenfassung

In dieser Bachelorarbeit werden die Effekte von Waveshaping mittels bekannter Waveshapingformen und einer auf Tetration beruhender Funktion analysiert. Die nötige Mathematik der Tetration wird erläutert und es werden spektrale Änderungen des Signals beim Waveshaping aufgezeigt. Abschließend wird über Nützlichkeit der verwendeten Formel zum Waveshaping diskutiert.

Abstract

This bachelor thesis will analyze the effects of waveshaping through common waveshaping functions and through a mathematical function based on tetration. The necessary mathematics of tetration will be explained, the spectral change of the shaped signal will be shown and the benefits of the used formula for waveshaping will be discussed.

Inhaltsverzeichnis

1 Einleitung

Tetration ist eine mathematische Funktion, die sehr schnell wächst. Schon die natürliche Tetration von 3,61 hat als Ergebnis eine Zahl mit 220 Stellen. Dadurch kann sie gut zur Darstellung großer Zahlen verwendet werden. Ihre Nutzung ist in der Mathematik nicht sehr verbreitet, obwohl sie eine logische Fortsetzung bekannter mathematischer Formeln ist.

Waveshaping ist ein Verfahren zur Klangerzeugung und Klangmodellierung, bei dem ein Audiosignal mit Hilfe einer nichtlinearen Übertragungskennlinie, in dieser Arbeit mit dem Fokus auf Kennlinien mittels Tetration, verzerrt wird. Dabei kann man einerseits bewährte Methoden für vorhersehbare spektrale Änderungen nutzen, andererseits kann man eine beliebige Kennlinienfunktion angeben oder zeichnen und sich vom erzeugten Klang überraschen lassen.

In dieser Bachelorarbeit wird der Mittelweg zwischen diesen zwei Methoden gegangen. Dabei geht es um die Erforschung neuer und unbekannter Klangformen. Es werden erst einige bekannte Waveshapingformen beleuchtet, danach wird die Mathematik der Tetration erklärt. Der Kernpunkt der Arbeit ist, die in diesem Feld noch nicht genutzte Tetration zum Waveshaping zu verwenden und die erzeugten Klänge zu analysieren. Dabei wird die Beziehung eines Koeffizienten der Tetration und der Amplitude des Eingangssignals auf das Ausgangssignal und zueinander beschrieben. Es wird versucht herauszufinden, ob sich der Aufwand für die erzeugte Klangänderung lohnt und ob Waveshaping mittels Tetration ins Repertoire von Sounddesignern und Musikern aufgenommen werden sollte.

Im Anhang sind kommentierte MATLAB Skripte zu finden, sodass LeserInnen selbst einen Eindruck über die in dieser Bachelorarbeit angesprochenen Waveshaping Formen erhalten können.

2 Waveshaping

2.1 Einführung in das Waveshaping

Waveshaping ist eine ökonomische und flexible Soundsynthesemethode auf Basis nichtlinearer Verzerrung/Verarbeitung, mit der obertonreiche Klänge generiert werden können [1].

Das Grundprinzip lässt sich am leichtesten durch einen Sinus erklären, der eine nichtlineare Kennlinie durchläuft. Durch die Verzerrung entstehen Obertöne, welche Modulationsprodukte des Eingangssignals sind [1]. Bei dieser Anreicherung des Signals steigt der Obertonanteil bei vielen geläufigen Kennlinien mit der Amplitude des Eingangssignals. Dies ist wünschenswert, da dies auch in Physik meist der Fall ist. Als Beispiel denke man an eine Trompete, bei der durch stärkeres Blasen ein komplexerer Obertonanteil entsteht [2].

Die Kennlinie wird mathematisch als ein System mit einer Funktion beschrieben, durch die das Eingangssignal läuft. Die Funktion $f(x) = x$ hat beispielsweise eine lineare Kennlinie, Eingangs- und Ausgangssignal sind identisch [2]. Als nichtlineare Kennlinien werden in der vorliegenden Arbeit Hard Clipping, Sinus, Tangens Hyperbolicus, Chebychev Polynome 1. Ordnung und schließlich Tetration betrachtet. In diesem Kapitel dient außer bei Beispielen mit einem Gitarrenton jeweils ein Sinuston mit 100Hz und Amplitude 1 als Eingangssignal.

2.2 Hard Clipping

Grafisches Verständnis von Waveshaping kann man gut durch Hard Clipping erlangen. In Abbildung 1 sieht man einen Sinus als Eingangssignal, die NLK[1] mit der Funktion

$$f(x) = \begin{cases} 0{,}5 & (x > 0{,}5) \\ x & sonst \\ -0{,}5 & (x < -0{,}5) \end{cases} \quad (1)$$

[1] Nichtlineare Kennlinie

und das Ausgangssignal. Außerdem sieht man, wie im linearen Bereich ($|x| < 0{,}5$) der Sinus unverändert durchgelassen wird, bei Werten darüber bleibt das Ausgangssignal auf dem maximalen/minimalen Wert $\pm 0{,}5$. Das Leistungsdichtespektrum[2] des Ausgangssignals ist in Abbildung 2 zu sehen. Man bemerkt hier bei genauerer Betrachtung, dass eine ungerade Waveshaping Funktion $f(x) = -f(-x)$ nur ungerade Teiltöne hervorbringt (Grundton 100Hz; Teiltöne 300Hz, 500Hz usw., siehe auch Kapitel 2.3 und 2.4), während bei geraden Funktionen $f(x) = f(-x)$ nur gerade Teiltöne entstehen (siehe Kapitel 2.5) [1].

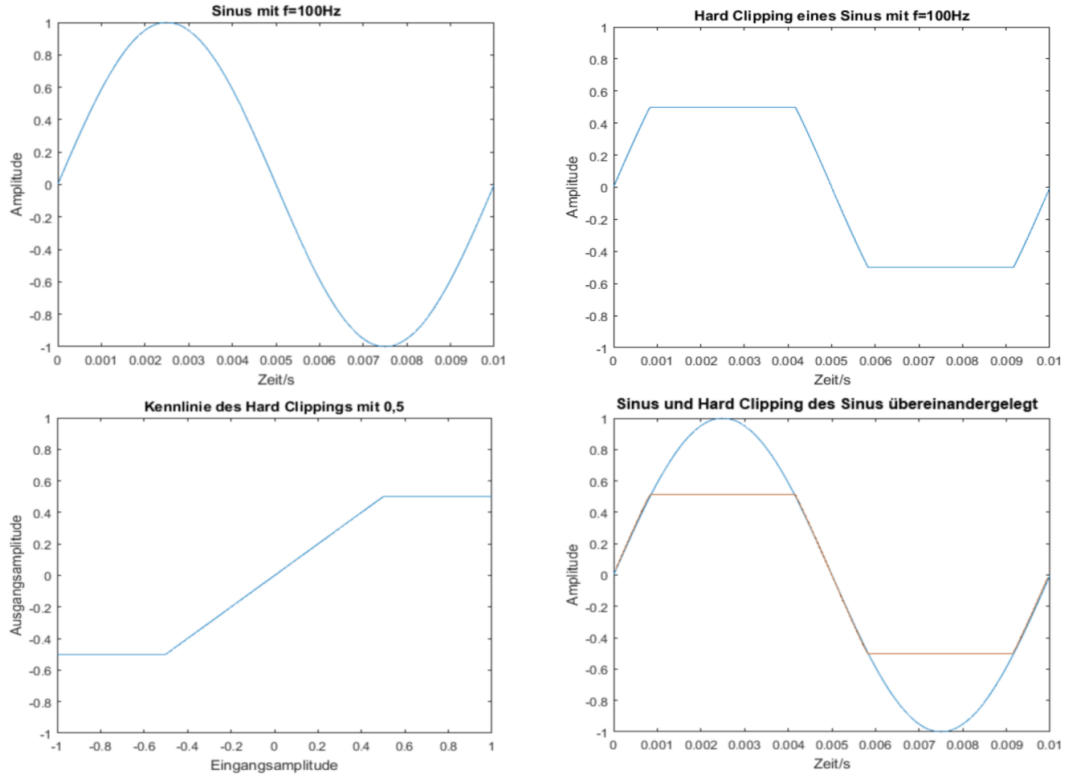

Abbildung 1: Eingangssignal (oben links), Ausgangssignal (oben rechts), Kennlinie (unten links) und Vergleichsbild (unten rechts) bei Waveshaping mit Hard Clipping bei 0,5

[2] Da die meisten Signale in der Musik stochastisch und nicht deterministisch sind, werden in dieser Arbeit einheitlich Leistungsdichtespektren verwendet, welche auf der Mittelung vieler FFT-Periodogramme beruhen. Durch sie kann, bei richtiger Einstellung der FFT-Parameter, ein direkter Bezug zu den Betragsquadraten der komplexen Fourierkoeffizienten und damit zu den Amplituden der Harmonischen hergestellt werden (siehe die verwendete MATLAB Funktion im Anhang 6.1). Bei den dargestellten Leistungsdichtespektren wird in dieser Arbeit grundsätzlich die stärkste Frequenz auf 0dB normiert.

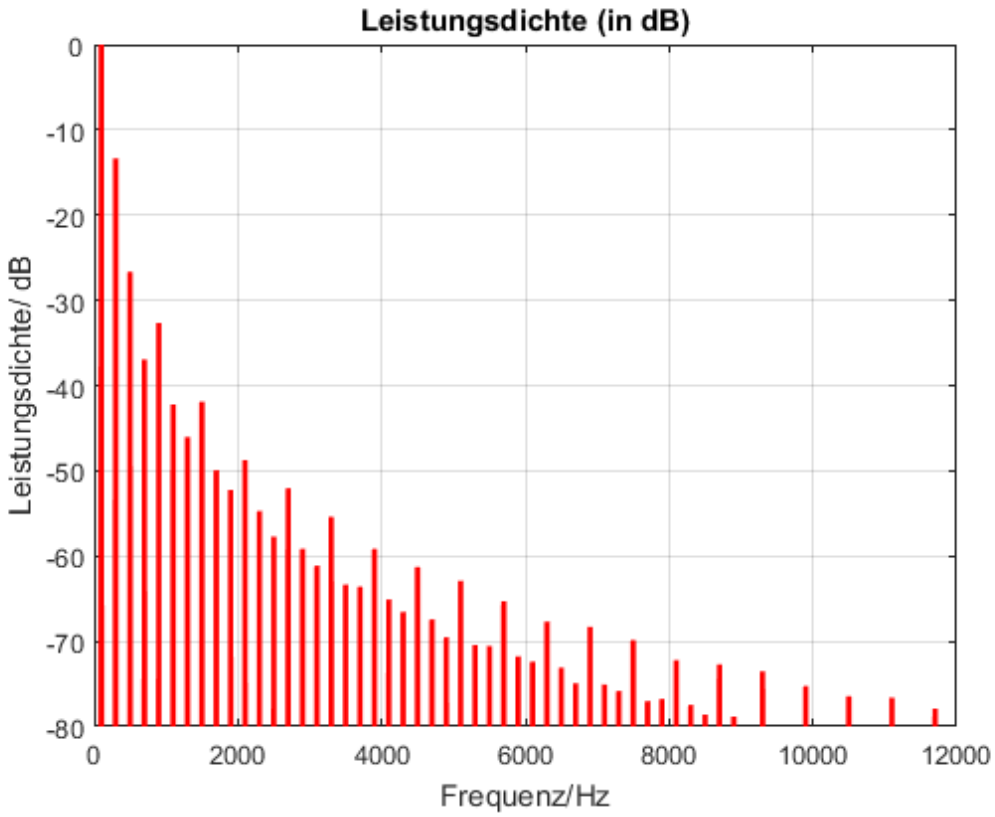

Abbildung 2: Erzeugtes Leistungsdichtespektrum eines Sinus, durch Hard Clipping bei 0,5 geformt

Elektronisch kann Hard Clipping durch Dioden Clipper verwirklicht werden.

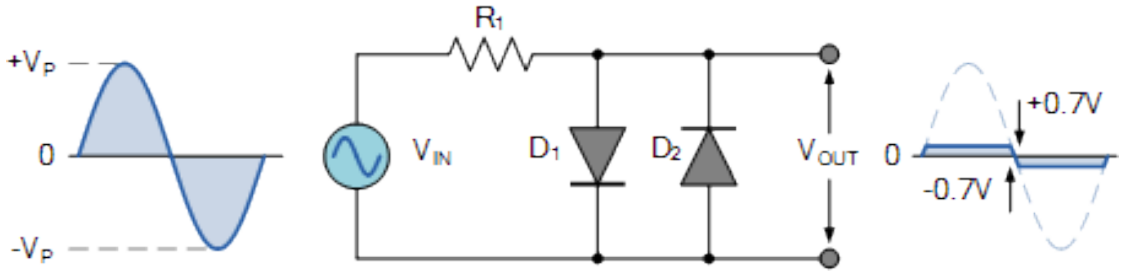

Abbildung 3: Eingangssignal, Schaltung eines Dioden Clippers, Ausgangssignal [3]

2.3 Waveshaping mit der Sinusfunktion

Bei Waveshaping mit der Sinusfunktion und dem in Kapitel 2.4 behandelten Tangens Hyperbolicus kann von Soft Clipping gesprochen werden. Als NLK wird nun eine Sinusfunktion $f(x) = \sin(\frac{\pi}{2} x)$ im Bereich x von -1 bis 1 benutzt. Die Funktion wurde durch den Faktor $\frac{\pi}{2}$ so gestaucht, dass für Eingangssignale mit der Amplitude ±1

auch das Ausgangssignal ±1 ist. Bei einem Sinus als Eingangssignal bekommt man für das Ausgangssignal $f(x) = \sin(\frac{\pi}{2}\sin(x))$. Wie in Abbildung 4 ersichtlich, sind kleine Amplituden fast unverändert, während große Amplituden durch die Kennlinie breiter werden. Wie man in Abbildung 5 sieht, ist der dritte Teilton relativ stark, der fünfte schon sehr gering.

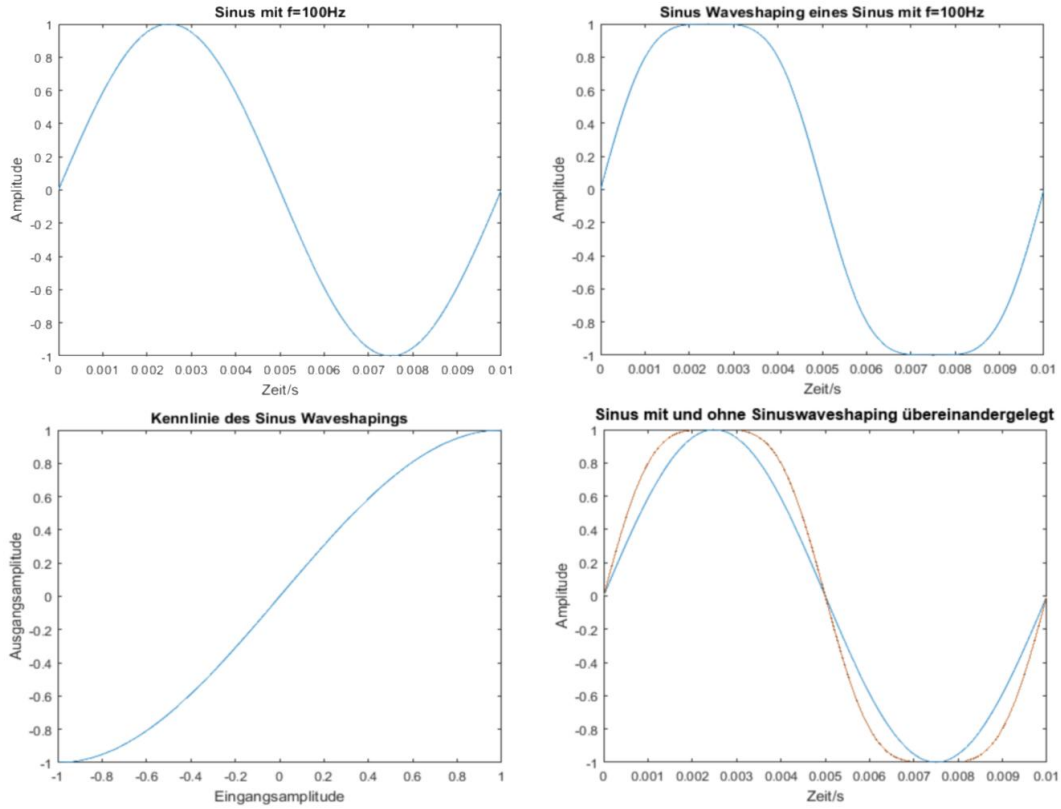

Abbildung 4: Eingangssignal (oben links), Ausgangssignal (oben rechts), Kennlinie (unten links) und Vergleichsbild (unten rechts) bei Waveshaping mit der Sinusfunktion

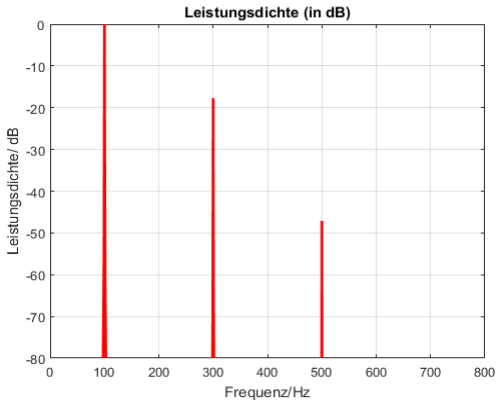

Abbildung 5: Erzeugtes Leistungsdichtespektrum eines Sinus durch eine Sinusfunktion geformt

Im Folgenden wird die mathematische Herleitung des Spektrums durch die Taylorreihe des äußeren Sinus und Umschreiben von $\sin(x)^n$ gezeigt:

$$\sin\left(\frac{\pi}{2}\sin(x)\right) = \sum_{n=0}^{\infty}(-1)^k \frac{(\frac{\pi}{2}\sin(x))^{1+2n}}{(1+2n)!} \tag{2}$$

$$= \sin(x) - \frac{(\frac{\pi}{2}\sin(x))^3}{6} + \frac{(\frac{\pi}{2}\sin(x))^5}{120} - \frac{(\frac{\pi}{2}\sin(x))^7}{5040} + \cdots$$

$$\approx \sin(x) - \left(\frac{\pi}{2}\right)^3 \frac{[3\sin(x)+\sin(3x)]}{24} + \left(\frac{\pi}{2}\right)^5 \frac{[10\sin(x)-5\sin(3x)+\sin(5x)]}{1920}$$

$$\approx 1{,}136\sin(x) - 0{,}157\sin(3x) + 0{,}005\sin(5x)$$

In dB beim Leistungsdichtespektrum[3] sind dies, wenn man auf 0dB beim Grundton normiert, für den 3.Teilton -17,19dB und für den 5.Teilton -47,13dB. Man sieht, dass diese Näherung ungefähr mit dem gemessenen Leistungsdichtespektrum übereinstimmt.

2.4 Waveshaping mit dem Tangens Hyperbolicus und Distortion

Eine sinusähnliche NLK bietet der Tangens Hyperbolicus. In Abbildung 6 ist unter anderem die Kennlinie für Eingangswerte von -1 bis 1 dargestellt. Dadurch, dass der Wertebereich des Tangens Hyperbolicus nur von -1 bis 1 reicht und die Funktion streng monoton steigend ist, können auch sehr große Eingangsamplituden verarbeitet werden (im Gegensatz zur in 2.3 verwendeten Form des Waveshapings mit der Sinusfunktion).

[3] Die Leistungen sind Amplitude²/2, die Werte in dB auf den Maximalwert normiert sind 10*log(Leistung/maximale Leistung)

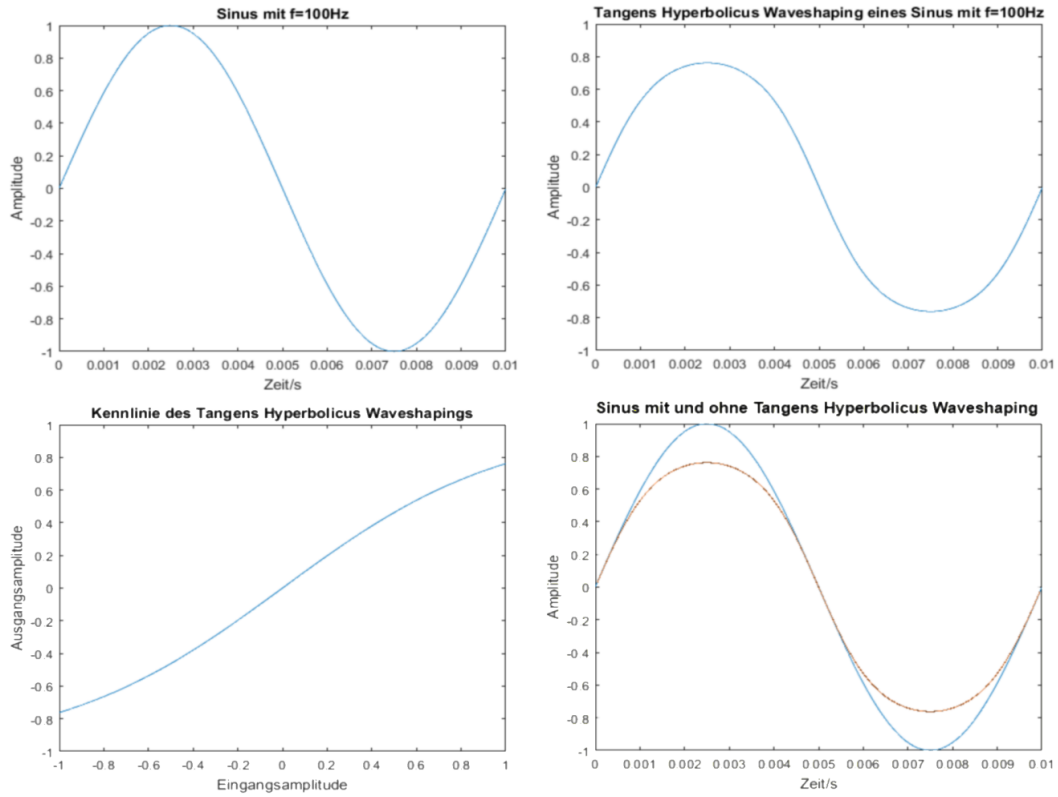

Abbildung 6: Eingangssignal (oben links), Ausgangssignal (oben rechts), Kennlinie (unten links) und Vergleichsbild (unten rechts) bei Waveshaping mit dem Tangens Hyperbolicus

Die Kurvenform des Tangens Hyperbolicus ist der eines Röhrenverstärkers relativ ähnlich. Für Audioanwendungen wurden diese im Gegensatz zu großen Teilen der restlichen Elektronik noch nicht von Transistoren verdrängt, da das sanftere Clipping bei den meisten Menschen durch Hörgewohnheiten als angenehmer wahrgenommen wird [4].

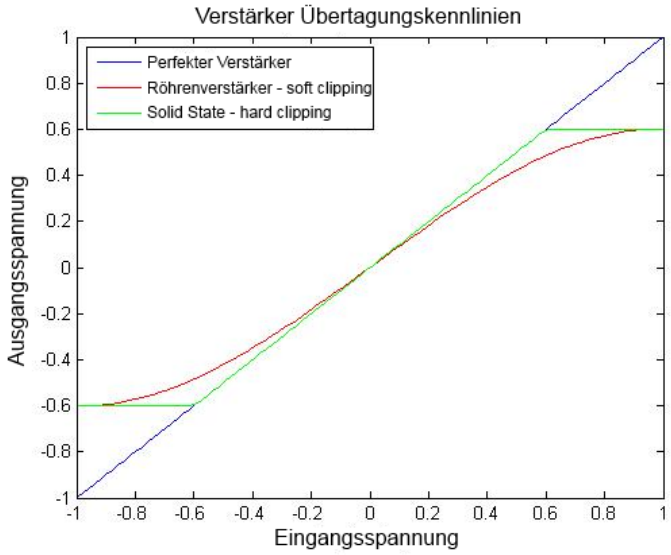

Abbildung 7: Vergleich eines idealen Verstärkers, eines Röhrenverstärkers und eines Transistorverstärkers [5]

Durch starkes Aufdrehen des Eingangs der Röhrenverstärker entsteht die Verzerrung, die z.B. den Sound einiger Gitarrenlegenden prägte [6]. Zur Veranschaulichung dieses Effekts werden in Tabelle 1 nicht nur die Teiltonstärken für ein Eingangssignal mit Amplitude 1, sondern auch für zwei Eingangssignale mit größerer Amplitude dargestellt. Es sind nur ungerade Teiltöne aufgelistet, da der Tangens Hyperbolicus eine ungerade Funktion ist und dadurch nur ungerade Teiltöne entstehen.

g/TT	1.	3.	5.	7.	9.	11.	13.	15.	17.	19.	21.	23.
1,00	0,00	-22,93	-44,17	-65,50								
2,00	0,00	-14,88	-27,38	-39,82	-52,44	-65,24	-78,34					
4,00	0,00	-10,74	-17,91	-24,65	-31,45	-38,42	-45,60	-52,58	-58,61	-64,88	-71,34	-78,11

Tabelle 1: Stärke der Teiltöne (TT) des Ausgangssignals in dB für eine Maximalamplitude g des Eingangssignals bei Waveshaping mit dem Tangens Hyperbolicus

Auch für moderne Gitarreneffekte wie Distortion und Overdrive werden dem Tangens Hyperbolicus ähnliche NLK genutzt, wobei durch große Vorverstärkung eine große Verzerrung verwirklicht wird und außerdem mehrere Hoch- und Tiefpässe den Klang mitgestalten.

Abbildung 8: Blockschaltbild eines Distortion Pedals[4] [7]

Dieses Blockschaltbild und die Werte des Boss DS-1 Distortion Pedals, auf denen der für die folgenden Verzerrungen verantwortliche MATLAB Distortion Code aufbaut, stammen aus einer Veröffentlichung der CCRMA der Stanford Universität. Im Gegensatz zu dieser Analyse inklusive klangverfeinernder Filter zum Abstrahieren eines Modells ist die Methode in dieser Bachelorarbeit eher puristisch und mathematisch. Es geht mehr darum, eine mathematische Funktion zur Klangerzeugung zu nutzen und zu überprüfen, ob die Ergebnisse wünschenswert sind.

In Abbildung 9 sind als Beispiel ein Ausschnitt eines Gitarrentons A[5] (Grundton 110Hz) ohne, mit schwacher und starker[6] Distortion dargestellt. Durch die Vorverstärkung des Eingangssignals und das Waveshaping mit dem Tangens Hyperbolicus erreicht das Ausgangssignal öfter den Bereich der maximalen Auslenkung[7] und verweilt dort länger.

[4] bipolarer Transistor Emitterfolger Buffer (Zwischenspeichern des Signals von den Tonabnehmern kommend am Eingang bzw. Ansteuern des Kabels zum nachfolgenden Verbraucher am Ausgang), Verstärkung und Filterung, Nichtlinearität durch Sättigung, Filter

[5] Deutsche Notation, in englischer Notation A2

[6] Die Distortionkonstante D des MATLAB Skriptes (siehe Anhang 4.2) wurde auf 0,25 bei schwacher Verzerrung und auf 1 bei starker Verzerrung gesetzt.

[7] Das Maximum von 1 wird nicht erreicht, da aber bei tanh(5) das Ergebnis schon größer als 0,9999 ist und die Vorverstärkerstufe um bis zu 36dB verstärkt kann der Unterschied für die Zwecke dieser Bachelorarbeit vernachlässigt werden.

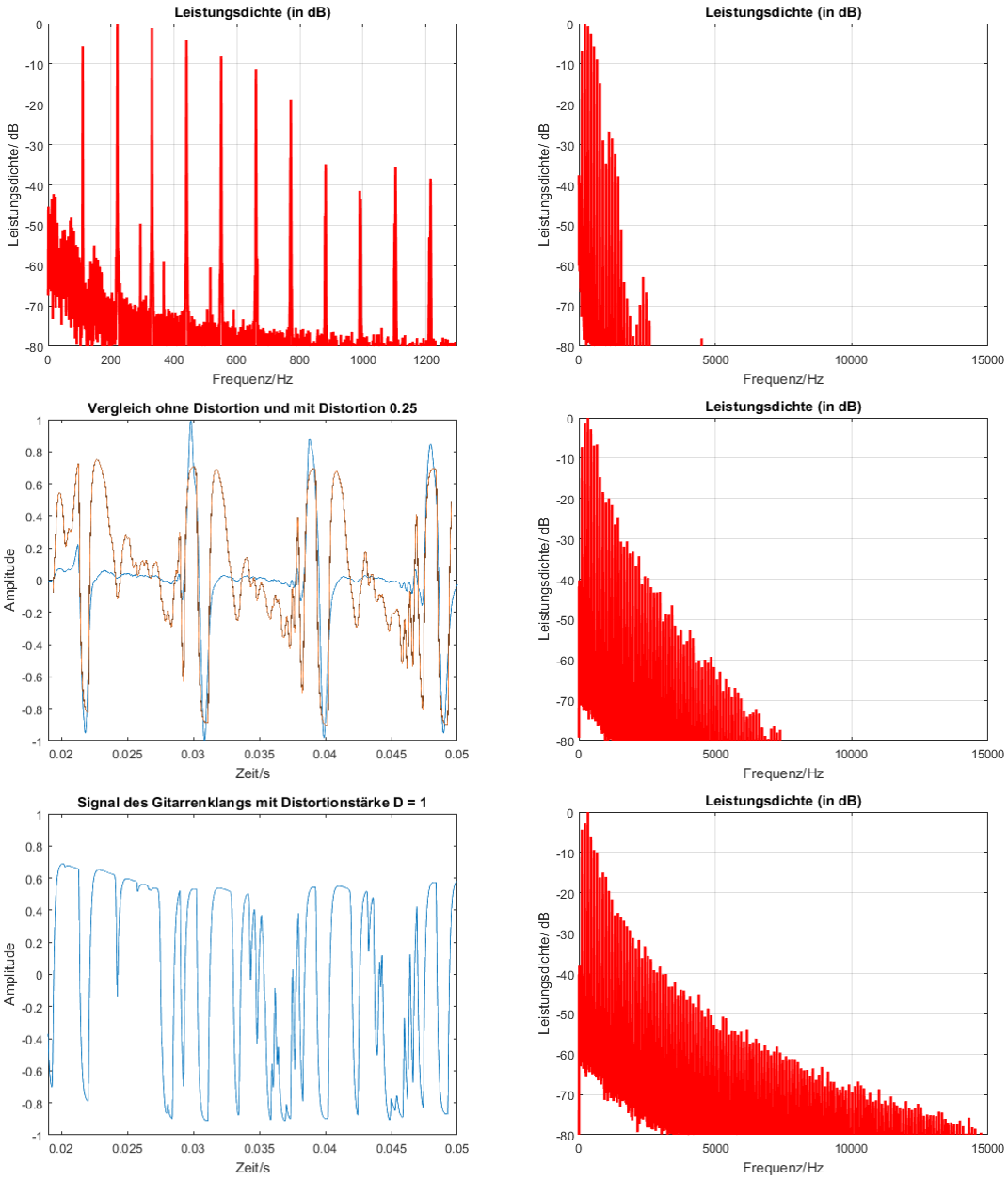

Abbildung 9: Leistungsdichtespektren eines Gitarrentons ohne (oben links mit Zoom, oben rechts ohne), mit schwacher (mittig rechts) und starker Verzerrung (unten rechts). Signale des Gitarrentons ohne Verzerrung (blau, mittig links), mit schwacher Verzerrung (braun, mittig links) und starker Verzerrung (unten links)

2.5 Waveshaping mit Chebychev Polynomen 1. Ordnung

Eine Möglichkeit zur guten Kontrolle über die Obertöne sind Chebychev Polynome erster Ordnung.

Mit

$$T_0(x) = 1 \tag{3}$$

$$T_1(x) = x$$

lässt sich jedes weitere Polynom

$$T_k(x) = 2x * T_{k-1}(x) - T_{k-2}(x) \tag{4}$$

bilden, also:

$$\begin{aligned} T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \end{aligned} \tag{5}$$

Ein Vorteil der Polynome ist, dass das Ausgangssignal bandbegrenzt ist. Dafür ist der Definitionsbereich ohne zusätzliche Transformation auf [-1; 1] beschränkt. Das k-te Polynom erzeugt bei einem sinusförmigen Eingang den k-ten Teilton [1]:

$$T_k(\cos(x)) = \cos(kx) \tag{6}$$

Durch die Subtraktion in den Polynomen werden zueinander phasenverkehrte Schwingungen erstellt, welche sich wieder aufheben, wodurch das Ausgangssignal eines Polynoms nur den gewünschten Teilton mit Amplitude 1 hat [2].

Als erstes Beispiel wurde hier $T_4(x)$ verwendet. In Abbildung 10 erkennt man, dass das Ausgangssignal ein Sinus mit vierfach höherer Frequenz und Phasenversatz ist. Der Phasenversatz entsteht bei Verwendung eines Sinus statt Cosinus als Eingangssignal.

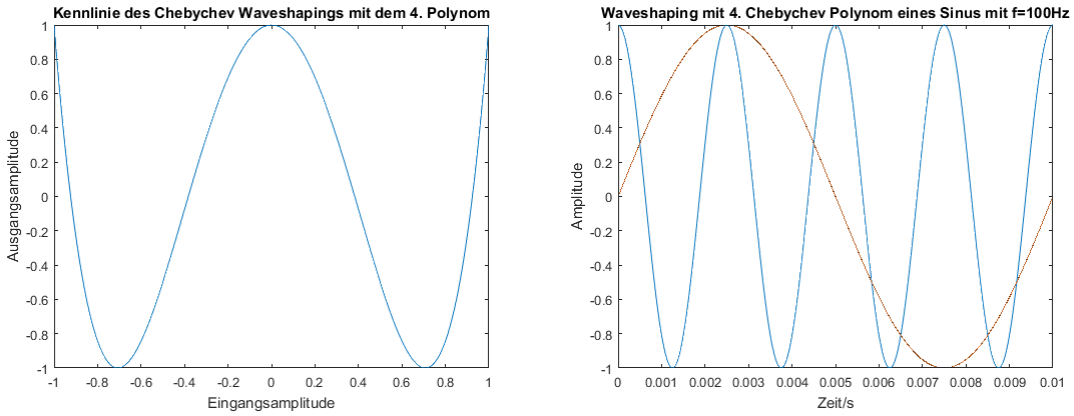

Abbildung 10: Kennlinie (links), Eingangssignal (rechts, braun) und Ausgangssignal (rechts, blau) beim Waveshaping mit dem 4.Chebychev Polynom 1. Ordnung

Die Polynome lassen sich in verschiedener Stärke miteinander kombinieren. Zur Demonstration wurde in Abbildung 11 das 6. Polynom mit der Stärke 0,5 und das 8. Polynom mit der Stärke -0,5 verwendet. Im Spektrum sind dann nur der 6. und 8. Teilton größer als -80dB. Durch Variation der Vorzeichen bei den Polynomen können verschiedene Kennlinien erzeugt werden, die denselben Effekt besitzen.

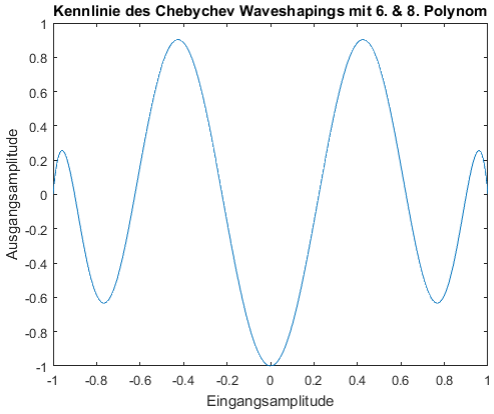

Abbildung 11: eine mögliche Kennlinie beim Waveshaping mit dem 6. und 8. Chebychev Polynom 1. Ordnung

Durch die Polynome kann mit relativ wenig Arbeit ein Klang konstruiert bzw. rekonstruiert werden. In Abbildung 12 sieht man noch einmal den Gitarrenton, der in Kapitel 2.4 schon verwendet wurde. Der Klang wurde bis zum 11. Oberton für diese Arbeit nachgebaut. Die zugehörige Transferfunktion, durch die ein Sinus mit 110Hz läuft, ist in Abbildung 13 zu sehen (das Ausgangssignal wurde skaliert, damit im wav Format nicht übersteuert wird). Die zum Nachbau verwendeten Faktoren der Polynome sind von T_1 bis T_{11}: 0,5; -1; -0,8; 0,6; -0,4; 0,3; -0,1; 0,017; -0,009; -0,015; -0,011

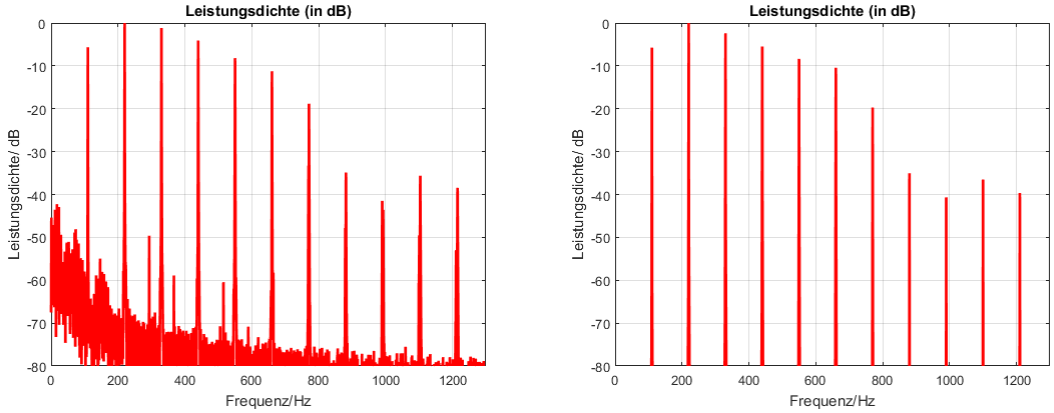

Abbildung 12: Leisungsdichtespektrum des Gitarrentons (links) und des mit Chebychev Polynomen nachgebauten Gitarrentons (rechts)

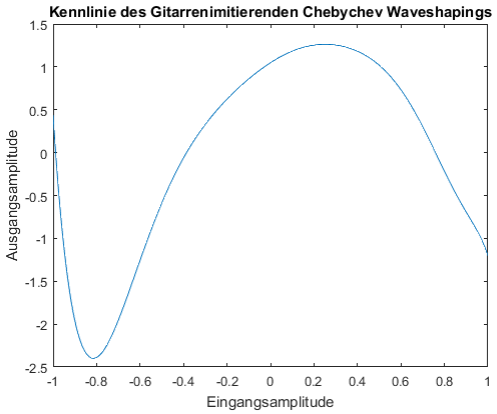

Abbildung 13: Kennlinie der gitarrenimitierenden Chebychev Polynome

Natürlich fehlt hier noch der für Hörereignisse extrem wichtige Einschwingvorgang, jedoch ist der Mittelteil klanglich sehr ähnlich und mit entsprechender Hüllkurve für den Sinus wirkt auch der Abklingvorgang akzeptabel.

3 Tetration[8]

3.1 Einführung in die Tetration

Der Begriff Tetration kommt von Tetra-Iteration, also die 4. Form einer mehrfachen Wiederholung [8][9]. Die mehrfache Wiederholung von Addition (1. Form) ist Multiplikation (2. Form). Bei wiederholter Multiplikation handelt es sich um die Exponentialfunktion (3. Form). Auf Basis dieser drei allgemein bekannten Berechnungsweisen kommt man zur Tetration [10]:

$$ {}^{n}x = x^{x^{x^{\cdots}}} \tag{7} $$

$$ Bsp.: {}^{2}x = x^{x} $$

Hier ist wichtig, dass die Exponentialfunktion nicht assoziativ ist, Zwischenergebnisse müssen von innen nach außen berechnet werden [9]:

$$ {}^{4}2 = 2^{2^{2^{2}}} = 2^{\left(2^{(2^{2})}\right)} = 2^{(2^{4})} = 2^{16} = 655336 \tag{8} $$

$$ {}^{4}2 \neq ((2^{2})^{2})^{2} = (4^{2})^{2} = 16^{2} = 256 $$

Weitere im Umlauf befindliche Namen sind z.B. Hyper-4, Superexponentialfunktion, Hyper Power und Power Tower, wobei die letzten beiden durch den namentlich Bezug zur Potenzfunktion (auf Englisch: power function) eher zu vermeiden sind [9].

Für die Iteration existiert die allgemeine Definition

$$ O_{n+1}(a, b+1) = O_{n}(a, O_{n+1}(a, b)), \tag{9} $$

wobei O die Funktion der Iteration, n die Stufe der Wiederholung, a die wiederholte Zahl und b die Anzahl der Wiederholungen ist [10]. Zum Beispiel:

$$ 4^{3} = O_{3}(4,3) = O_{2}\left(4, O_{3}(4,2)\right) = O_{2}(4,4^{2}) = 4 * 4^{2} = 4^{3} \tag{10} $$

$$ {}^{4}2 = O_{4}(2,4) = O_{3}\left(2, O_{4}(2,3)\right) = O_{3}(2, {}^{3}2) = O_{3}(2, O_{3}(2, {}^{2}2)) = 2^{(2^{(2^{2})})} $$

[8] Da die Mathematik der Tetration für diese Bachelorarbeit weniger der Untersuchungsmittelpunkt, sondern mehr Mittel zum Zweck einer neuen Art des Waveshapings ist, wird in der vorliegenden Arbeit ein grober Überblick über für diesen Zweck wichtige Aspekte gegeben.

3.2 Beschreibung der verwendeten Waveshaping Formel

In dieser Arbeit wird der Spezialfall der natürlichen Tetration betrachtet, bei der die Eulerzahl als Basis benutzt wird [8]. Diese ist wie folgt definiert:

$$tet(x) = {}^{x}e = e^{e^{e^{\cdots}}} \tag{11}$$
$$e^{tet(x)} = tet(x+1)$$
$$tet(0) = 1$$

Außerdem wird ihre inversen Funktion ate(x) verwendet (ArcTetration), welche durch folgende Identitäten beschrieben wird:

$$ate\big(tet(x)\big) = x \tag{12}$$
$$ate(e^{x}) = ate(x) + 1$$

Durch diese beiden Funktionen gelangt man zu der in dieser Arbeit verwendeten Waveshapingformel:

$$tet\big(c + ate(x)\big) = {}^{c}e^{x} \tag{13}$$
$$Bsp\text{: } {}^{2}e^{0{,}1} = e^{e^{0{,}1}} \approx 3{,}02$$

Die Korrektheit dieser Formel kann durch ate auf beiden Seiten (Formel (13) zu (14)) und durch die Identitäten aus Formel (12) bewiesen werden:

$$ate(tet(c + ate(x))) = ate({}^{c}e^{x}) \tag{14}$$
$$c + ate(x) = ate(x) + c$$

Mit Hilfe der Formel (13) verbindet man eine normale Exponentialfunktion mit der Iterationskonstante c. Die Waveshaping Formel wird so genutzt, dass x das Eingangssignal (zur Analyse meistens $x = \mathrm{g} * \sin(\omega t)$) und c der einstellbare Koeffizient zur Wahl der verschiedenen Kurvenformen ist (Tetrationskonstante, Anzahl der Iteration). Für $\mathrm{c} = 1$ erhält man die natürliche Exponentialfunktion $f(x) = e^{x}$, $\mathrm{c} = 0$ liefert eine lineare Kennlinie $f(x) = x$ und $\mathrm{c} = -1$ erzeugt den natürlichen Logarithmus $f(x) = \ln(x)$. Die Tetrationskonstante c und das Eingangssignal x müssen keine natürliche Zahlen sein, es können negative, gebrochene, irrationale und sogar komplexe Zahlen genutzt werden [8], wobei in der vorliegenden Arbeit nur mit

reellen Zahlen gearbeitet wurde. Abbildung 14 zeigt die Kennlinie der Tetrationsformel für verschiedene Tetrationskonstanten.

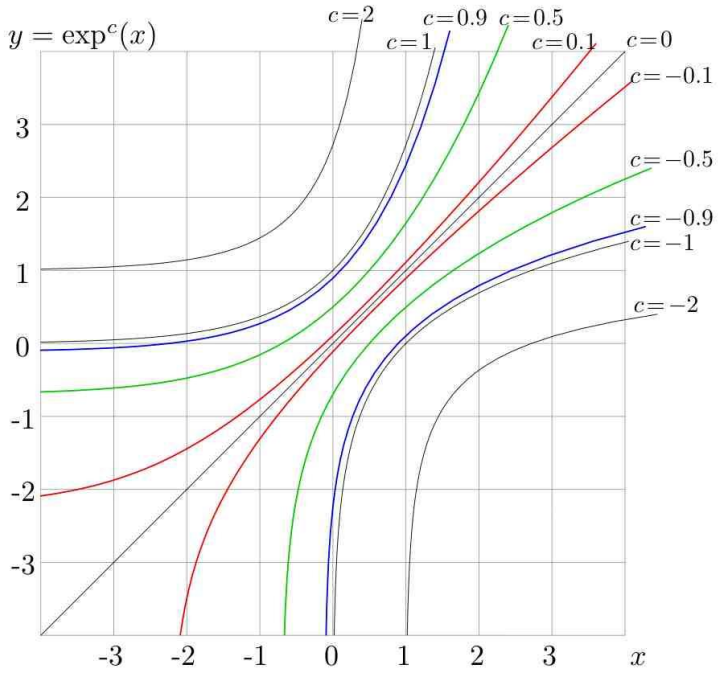

Abbildung 14: Kennlinien für ${}^{c}e^{x}$ bei verschiedenen Werten von c [11]

Die MATLAB Skripte zur Berechnung der Tetration und ihrer inversen Funktion wurden von Thomas Mayr für seine Masterarbeit „Modulation Power Spectrum Transformations“ geschrieben. Diese approximieren Werte der natürlichen Tetration und der inversen natürlichen Tetration zwischen ganzen Zahlen, unter anderem mit der Maclaurin Reihe [8]. Große Teile der Theorie beruhen auf der Arbeit von Dmitrii Kouznetsov.

4 Waveshaping mit Tetration

4.1 Waveshaping mit der natürlichen Exponentialfunktion

Da die in dieser Arbeit benutzte natürliche Tetration auf einer Iteration der natürlichen Exponentialfunktion beruht und es auch einen Zusammenhang der Spektren gibt, der in Kapitel 4.3 beschrieben wird, folgt in diesem Kapitel eine mathematische Herleitung des Obertonspektrums beim Waveshaping mit der Exponentialfunktion.

Die natürliche Exponentialfunktion in Potenzreihendarstellung lautet:

$$\mathrm{e}^{\mathrm{z}} = \sum_{n=0}^{\infty} \frac{z^n}{n!} \tag{15}$$

Mit einem Eingangssignal $\mathrm{z} = \mathrm{g}\cos(x)$ kommt man auf Formel (16).

$$\sum_{n=0}^{\infty} \frac{[g\cos(x)]^n}{n!} \tag{16}$$

Zum Ablesen der Harmonischen schreibt man den Cosinus mit Formel (17) um:

$$\cos^n(x) = \frac{1}{2^n}\sum_{m=0}^{n}\binom{n}{m}\cos((n-2m)x) \tag{17}$$

$$\sum_{n=0}^{\infty}\sum_{m=0}^{n}\left(\frac{g}{2}\right)^n \frac{1}{n!}\binom{n}{m}\cos((n-2m)x) \tag{18}$$

Daraufhin kann man den Vergleich zur Fourierreihe ziehen, welche wie folgt lautet:

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k\cos(kx) + b_k\sin(kx)) \tag{19}$$

Durch den Vergleich erhält man $\mathrm{b_k} = 0$. Für die restlichen Koeffizienten müssen weitere Überlegungen gemacht werden. Bei dem Koeffizientenvergleich für $\mathrm{a_k}$ müssen alle Cosinus Schwingungen gesammelt werden, bei denen $\mathrm{n} - 2\mathrm{m} = \mathrm{k}$ ist, damit der Cosinus aus Formel (18) und (19) für die gegebene k-te Harmonische übereinstimmt. Außerdem müssen $\mathrm{n} - 2\mathrm{m} = -\mathrm{k}$ gesammelt werden (da $n - 2m$ auch negativ werden kann), was auf einen Faktor 2 hinausläuft, da $\cos(-\mathrm{kx}) = \cos(\mathrm{kx})$ und $\binom{n}{\frac{n-k}{2}} = \binom{n}{\frac{n+k}{2}}$. Da der Gleichanteil nicht hörbar ist, kann dieser in der Fourierreihe ($\mathrm{a_0}$) und in Formel (18) ($\mathrm{n} - 2\mathrm{m} = 0$) vernachlässigt werden.

Für den Grundton a_1 sind beispielsweise alle Anteil $n - 2m = \pm 1$ relevant, also $m = \frac{n \pm 1}{2}$. Da m eine natürliche Zahl ist, muss n eine ungerade Zahl sein. Dadurch kann man Formel (20) aufstellen.

$$a_1 = \sum_{n=1,n\,ungerade}^{\infty} \left(\frac{g}{2}\right)^n \frac{2}{n!} \binom{n}{\frac{n-1}{2}} \tag{20}$$

Die untere Grenze der Summe erreicht man dabei, indem $m = 0$ durch $n - 2m = k$ zu $n = k$ (in Formel (20) $n = 1$) wird. Die obere Grenze wird zu ∞, da es unendlich viele Kombinationen von $n - 2m = k$ gibt.

Für die erste Harmonische a_2 sind alle Anteil $n - 2m = \pm 2$ relevant, also $m = \frac{n \pm 2}{2}$. Da m eine natürliche Zahl, ist muss n hier eine gerade Zahl sein.

$$a_2 = \sum_{n=2,n\,gerade}^{\infty} \left(\frac{g}{2}\right)^n \frac{2}{n!} \binom{n}{\frac{n-2}{2}} \tag{21}$$

Um eine einheitliche Formel für ungerade und gerade Teiltöne zu bekommen wird der Laufindex in der Formel mit $n = 2n - k$ substituiert (n zählt von k immer +2), wodurch Formel (22) erreicht wird.

$$a_k = \sum_{n=k}^{\infty} \left(\frac{g}{2}\right)^{2n-k} \frac{2}{(2n-k)!} \binom{2n-k}{n-k} \tag{22}$$

Diese Formel kann noch durch Auflösen des Binomialkoeffizienten und Kürzen von (2n-k)! zu Formel (23) vereinfacht werden.

$$a_k = \sum_{n=k}^{\infty} \left(\frac{g}{2}\right)^{2n-k} \frac{2}{(n-k)!n!} \tag{23}$$

Die Werte des Leistungsdichtespektrums werden bei Normierung des maximalen Wertes auf 0dB wie folgt berechnet[9]:

$$20\log\left(\frac{a_k}{a_{k,\max}}\right) \tag{24}$$

Es wurde versucht die Formel (23) weiter zu vereinfachen (unter anderem durch die Stirling Approximation), was jedoch zu keinem einfacheren Ergebnis führte. Das MATLAB Skript zur Berechnung dieser Werte ist im Anhang (5.1) zu finden.

[9] Vereinfachte Form von $10\log((a_k^2/2)/(a_{k,max}^2/2))$, $a_k^2/2$ sind die Leistungen

Da sich der Phasenversatz von $sin(2\pi ft + \varphi)$ gegenüber $cos(2\pi ft)$ im Leistungsdichtespektrum nicht auswirkt, kommt man bei Berechnungen mit dem Sinus zu den gleichen Leistungen der Teiltöne wie mit der gezeigten Formel (23) des Cosinus.

In Tabelle 2 sieht man, wie das normierte Leistungsdichtespektrum der Exponentialfunktion für wachsende Eingangssignalamplituden von 0,5 bis 64 immer breiter wird und die Stärke der Teiltöne wächst. Hier ist auch interessant zu sehen, dass bei großen Amplituden des Eingangssignals der Grundton nicht immer der stärkste Teilton ist (siehe Tabelle 2 erster bis dritter Teilton bei g = 32 bzw. g = 64).

In Abbildung 15 ist Tabelle 2 grafisch aufbereitet, man sieht den Verlauf der Teiltonstärke abhängig von der Teiltonnummer für verschiedene g.

TT/g	0,5	1	2	4	8	16	32	64
1.	0	0	0	0	0	0	-1,45	-0,4
2.	-17,54	-11,78	-6,66	-3,03	-1,12	-0,23	0	0
3.	-39,09	-27,43	-16,78	-8,63	-3,88	-1,54	-0,6	-0,26
4.	-64,06	-46,48	-30,11	-16,95	-9,47	-4,36	-2,63	-1,61
5.		-65,51	-43,32	-24,86	-12,53	-5,81	-3,25	-1,18
6.			-59,79	-35,85	-19,08	-9,52	-6,28	-2,63
7.			-76,56	-46,95	-25,46	-12,72	-6,54	-3,18
8.				-59,19	-32,76	-16,51	-8,07	-3,95
9.				-74,09	-42,57	-22,19	-11,53	-6,6
10.					-50,01	-25,99	-12,89	-6,33
11.					-60,28	-31,99	-16,26	-8,28
12.					-70,49	-37,85	-19,29	-9,77
13.						-43,8	-22,22	-11,04
14.						-51,92	-27,16	-14,19
15.						-57,78	-29,72	-14,86
16.						-65,7	-34,18	-17,31
17.						-73,92	-38,73	-19,73
18.							-42,73	-21,5
19.							-48,71	-25,68
20.							-52,75	-26,73
21.							-58,24	-29,68
22.							-69,12	-33,01
23.							-76,97	-35,26
24.								-40,36
25.								-41,88
26.								-45,3
27.								-49,55
28.								-52,26
29.								-56,84
30.								-60,24
31.								-64,08
32.								-69,15
33.								-72,19
34.								-77,05

Tabelle 2: relevante Teiltöne (TT > -80dBFS[10]) des Ausgangssignals bei Waveshaping mit der natürlichen Exponentialfunktion für Eingangssignalamplituden g von 0,5 bis 64

[10] Decibel Fullscale, logarithmische Amplitude bezogen auf die maximale Amplitude

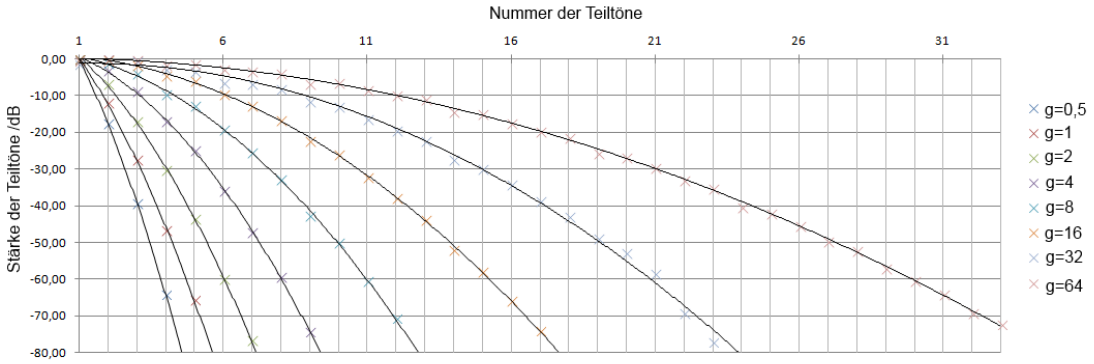

Abbildung 15: Stärke der Teiltöne als Funktion der Teiltöne für Eingangsamplituden von g=0,5 bis 64, Kurven als Trendlinien[11] der gemessenen Werte

4.2 Waveshaping bei einer positiven Tetrationskonstante

In Tabelle 3 sieht man die Teiltonstärken bei einem Sinus mit Amplitude 1 als Eingangssignal bei Waveshaping mit der Tetrationskonstante c von 0,5 bis 3. Die Verzerrung wird mit wachsender Tetrationskonstante immer größer.

[11] Formeln der Trendlinien von g = 0,5 links bis g = 64 rechts in Abbildung 15,
y(x) = dBFS(Teiltonnummer):

$y_{0,5} = -1{,}8575\,x^2 - 12{,}085\,x + 13{,}973$
$y_1 = -1{,}2786\,x^2 - 8{,}9006\,x + 10{,}526$
$y_2 = -0{,}9769\,x^2 - 5{,}1305\,x + 6{,}7429$
$y_4 = -0{,}7258\,x^2 - 2{,}0819\,x + 3{,}4424$
$y_8 = -0{,}4268\,x^2 - 0{,}9669\,x + 2{,}098$
$y_{16} = -0{,}2401\,x^2 - 0{,}3587\,x + 1{,}2754$
$y_{32} = -0{,}1544\,x^2 + 0{,}4123\,x - 1{,}5416$
$y_{64} = -0{,}0649\,x^2 - 0{,}0728\,x + 0{,}2931$

TT/c	0,50	1,00	1,50	2,00	2,50	2,75	3,00
1.	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2.	-16,27	-11,78	-8,20	-5,14	-1,79	-0,61	-1,17
3.	-42,60	-27,43	-19,73	-12,96	-5,83	-2,38	-1,45
4.	-78,03	-46,48	-32,07	-21,24	-10,14	-5,62	-1,71
5.		-65,51	-45,12	-29,95	-14,75	-7,39	-2,72
6.			-60,10	-40,35	-20,96	-14,47	-4,47
7.			-73,99	-49,38	-25,74	-18,05	-6,96
8.				-61,52	-33,57	-23,28	-7,61
9.				-70,69	-38,28	-26,40	-9,08
10.					-45,36	-31,47	-11,24
11.					-52,06	-36,14	-14,08
12.					-58,89	-40,63	-16,65
13.					-67,00	-47,05	-18,40
14.					-73,53	-50,98	-20,82
15.						-56,75	-23,89
16.						-62,44	-27,62
17.						-67,56	-30,14
18.						-74,40	-32,70
19.							-35,89
20.							-39,70
21.							-43,88
22.							-46,50
23.							-49,75
24.							-53,59
25.							-58,00
26.							-61,89
27.							-65,19
28.							-69,17
29.							-73,58
30.							-78,17

Tabelle 3: relevante Teiltöne des Ausgangssignals bei Waveshaping mit Tetration für Tetrationskonstanten von 0,5 bis 3 und jeweils Eingangsamplitude g = 1

In Abbildung 16 sind noch zwei Beispiele zu sehen, bei denen sowohl die Tetrationskonstante als auch die Amplitude des Eingangssignals größer als eins ist. Die Frequenz des Eingangssignals beträgt 200Hz, es sind bei beiden Signalen mehr als 80 Obertöne größer als -80dBFS. Hier sieht man, wie schnell die Verzerrung stärker werden kann, wenn Tetrationskonstante und Eingangsamplitude beim Waveshaping mit Tetration eines Sinus gleichzeitig erhöht werden.

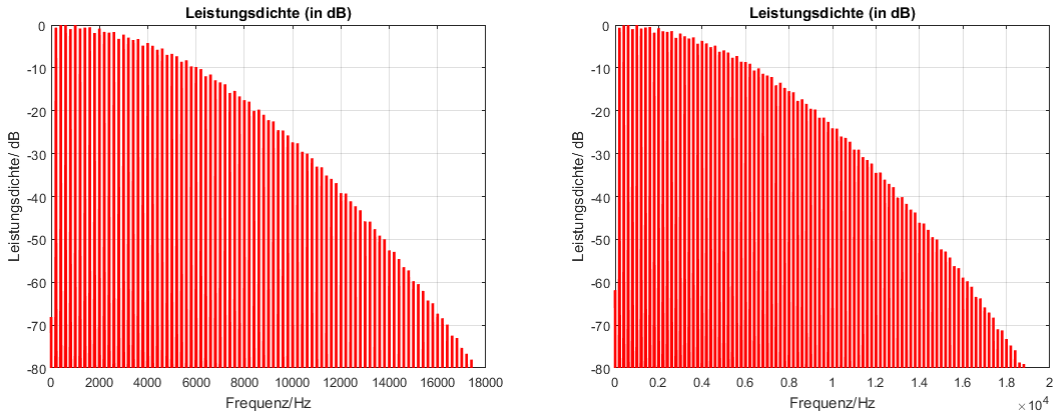

Abbildung 16: Leistungsdichtespektrum für Tetrationskonstante 2 und Eingangsamplitude 4,5 (links) und für Tetrationskonstante 2,5 und Eingangsamplitude 2,4 (rechts) bei Waveshaping mit Tetration eines Sinus von 200Hz

4.3 Vergleich von Waveshaping mit der Exponentialfunktion und Tetration bei positiver Tetrationskonstante

Interessant ist hierbei, dass bei gleichem Eingangssignal für jede Tetrationskonstante c beim Waveshaping mit Tetration und Amplitude 1 eine Eingangsamplitude g existiert, mit der bei Waveshaping mit einfacher natürlicher Exponentialfunktion ein äußerst ähnliches[12] Leistungsdichtespektrum erreicht werden kann. In Abbildung 17 kann man dafür einen ungefähren Verlauf erkennen, die Werte dafür wurden experimentell ermittelt. Um die Stelle c = 1 ändert sich der Kurvenverlauf. Die Tetration mit Tetrationskonstante 1 stimmt mit der Exponentialfunktion überein. Man erkennt, dass die Exponentialfunktion eine stärkere Tetration als Tetration mit c kleiner 1 ist und eine schwächere Tetration als mit c größer 1. Spektren von Klängen mit variablem c und Amplitude 1 werden in dieser Arbeit c-Spektren genannt, bei variabler Amplitude g und fester Tetrationskonstante c = 1 (Exponentialfunktion) wird von g-Spektren gesprochen.

[12] Ähnlich in dem Sinne, dass der Unterschied zwischen den starken Teiltönen sehr gering sind, siehe Seite 27

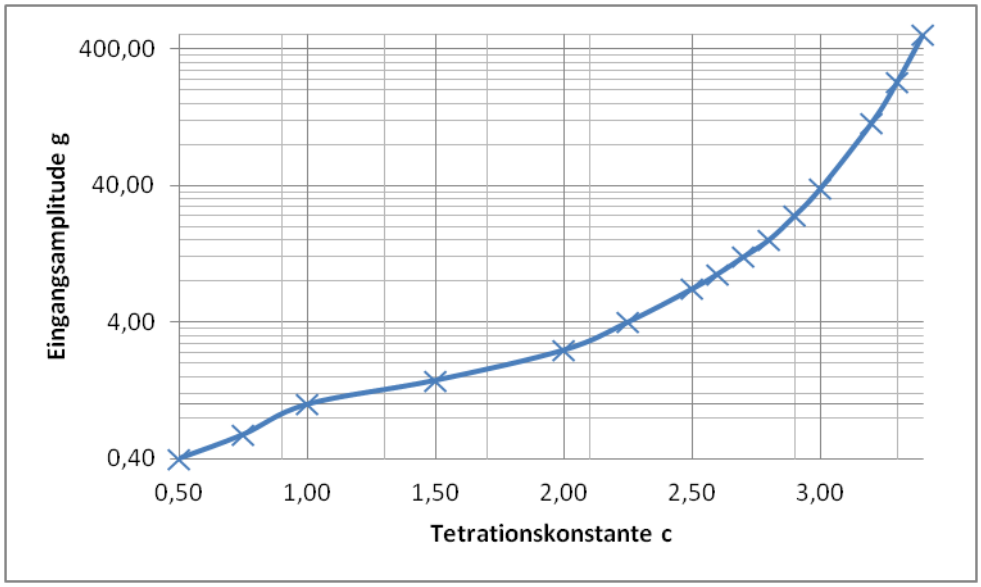

Abbildung 17: ungefährer Verlauf für ähnliche g- und c-Spektren, 15 Werte für c im Bereich 0,5 bis 3,4 erfasst und dann interpoliert.

Zum Vergleich sind das c-Leistungsdichtespektrum 2 und das g-Leistungsdichtespektrum 2,5 in Abbildung 18 dargestellt. Hier gibt es bei den beiden Grundtönen keine Differenz, da die stärkste Frequenz unabhängig vom anderen Signal jeweils auf 0dB normiert wurde.

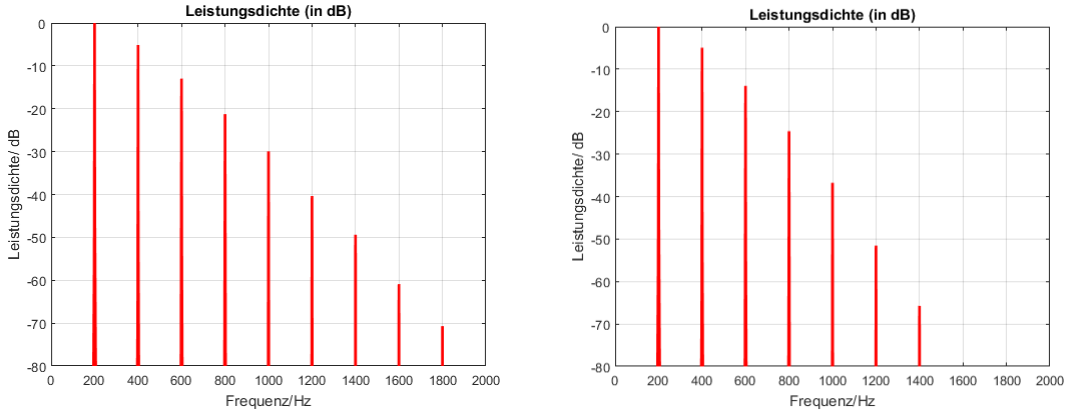

Abbildung 18: c-Leistungsdichtespektrum 2 (links) und g-Leistungsdichtespektrum 2,5 (rechts) bei f = 200Hz

Zur weiteren Darstellung sieht man in Abbildung 19 das LDS[13] [c = 2,5; g = 1] minus das LDS [c = 1; g = 7] und das LDS [c = 3; g = 1] minus das LDS [c = 1; g = 38] dargestellt (f = 200Hz bei allen Klängen). Dabei wurden zur besseren Veranschaulichung Frequenzen kleiner als -80dB vor der Subtraktion auf den Wert

[13] Leistungsdichtespektrum

-80dB gesetzt. Im Gegensatz zu vorher wurden bei Abbildung 19 die Klänge auf die stärkste Frequenz beider Signale normiert. Diese weichen voneinander ab, da die eingelesenen Audiodateien auf ihre Maximalamplitude in der Amplitude-Zeit Darstellung normiert wurden.

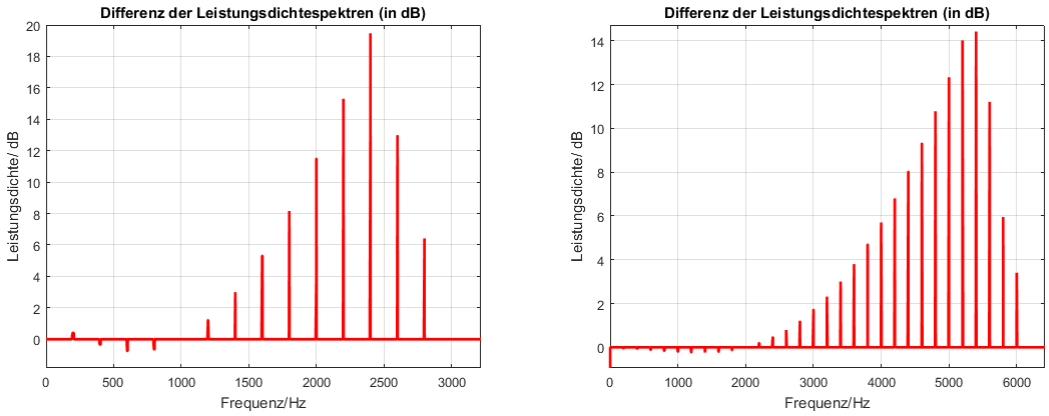

Abbildung 19: Differenz des c-Leistungsdichtespektrums 2,5 und des g-Leistungsdichtespektrums 7 (links) und Differenz des c-Leistungsdichtespektrums 3 und des g-Leistungsdichtespektrums 38 (rechts)

Bei den Vergleichen sieht man, dass im Grundtonbereich die c-Leistungsdichtespektren (bei c größer als 1) und im mittleren Obertonbereich die g-Leistungsdichtespektren stärker sind. Dies ist jedoch durch den geringen Unterschied vernachlässigbar (bei normaler Lautstärke sind Pegelunterschiede kleiner als 1dB kaum wahrnehmbar [12]). Relevant sind aber die bei c-Leistungsdichtespektren deutlich stärkeren hohen Obertöne, die den Klang hörbar aufhellen. Es gibt auch c-g Paare, bei denen die hohen Obertöne einheitlicher sind, jedoch sind dort die Unterschiede bei den ersten Obertönen deutlicher hörbar, da diese allgemein viel lauter sind.

4.4 Waveshaping bei einer negativen Tetrationskonstante

Bei negativen Tetrationskonstanten ist es wichtig, dass die Amplitude des Eingangssignals nicht zu groß ist, damit die Singularität bei den logarithmusähnlichen Kennlinien nicht erreicht wird (bei Überschreiten dieser würde das Signal einen Imaginärteil bekommen). Die Werte in Abbildung 20 wurden experimentell mit Hilfe der MATLAB Skripte ermittelt.

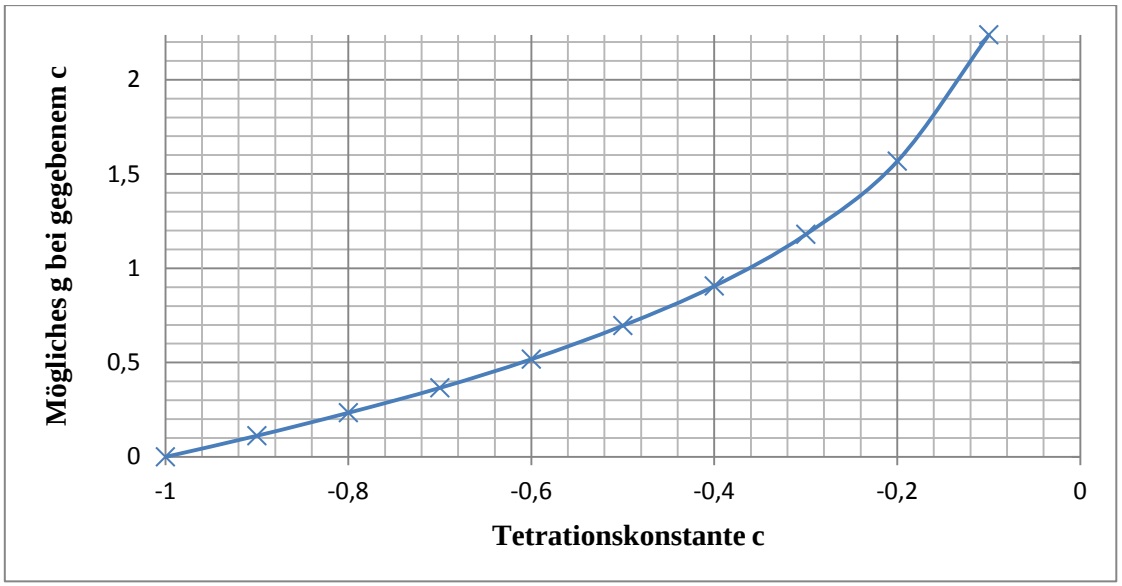

Abbildung 20: Maximal mögliche Eingangsamplitude für Tetrationskonstanten c, 10 Werte für c im Bereich -1 bis 1 erfasst und dann interpoliert.

Für negative Tetrationskonstanten hat eine Amplitudenänderung des Eingangssignals um den gleichen Wert meistens einen stärkeren Effekt als bei positiven c. Je näher die Eingangsamplitude an der Stelle der Singularität liegt, desto breiter ist das Leistungsdichtespektrum, wobei diese Beziehung mit steigender Amplitude immer stärker steigt. Als Beispiel wurden in Abbildung 21 für die Tetrationskonstanten -0,1 und -0,8 die Leistungsdichtespektren dargestellt mit einem Sinus von 200Hz als Eingangssignal, dessen Amplitude sehr nah an der Singularität ist (2,238 bzw. 0,233) und verglichen mit den Leistungsdichtespektren für 99% und 98% dieser Eingangsamplitude (2,216 und 2,193 bzw. 0,231 und 0,228).

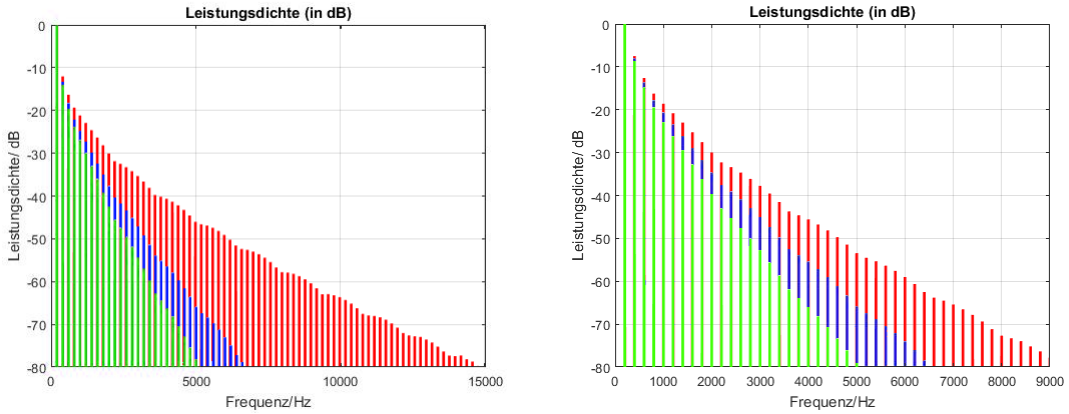

Abbildung 21: Leistungsdichtespektrum für c = -0,1 mit Maximaleingangsamplitude 2,238 (links) und für c = -0,8 mit Maximaleingangsamplitude 0,233 (rechts) für eine Eingangsamplitude von 100 Prozent (rot), 99% (blau) und 98% (grün) der Maximaleingangsamplitude

Man sieht, dass das erste Prozent Änderung einen stärkeren Einfluss hat. Bei gleicher Amplitude des Eingangssignals wäre die Verzerrung durch die Tetrationskonstante mit dem größeren Betrag viel stärker. Da jedoch die benutzte Eingangsamplitude bei $c = -0{,}1$ durch Rundungen etwas näher an der Singularität liegt, ist die Verzerrung bei der kleineren Tetrationskonstante größer.

Es folgen in Abbildung 22 noch die Leistungsdichtespektren bei der Tetrationskonstante $c = 2$, einer Frequenz von 200Hz und einer Eingangsamplitude von 4,5 bzw. 4,455. Trotz der starken Verzerrung ist der Einfluss der Amplitudenänderung um einen Prozent sehr gering. Dies verdeutlicht im Vergleich mit Abbildung 21 die Wichtigkeit der Singularität bei negativen Tetrationskonstanten.

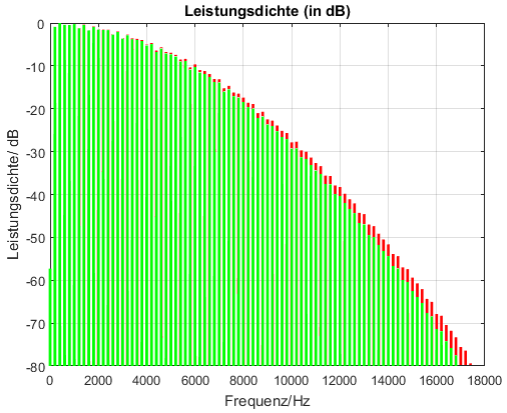

Abbildung 22: Leistungsdichtespektrum bei $c = 2$ und Eingangssignal mit $f = 200$ und $g = 4{,}5$ (rot) bzw. $g = 4{,}455$ (grün)

Bei negativen Tetrationskonstanten können also mit geringer Amplitude teiltonreiche Leistungsdichtespektren entstehen. Wegen der Instabilität in der Nähe der Singularität sind negative Tetrationskonstanten jedoch mit Vorsicht zu nutzen. Zur Untersuchung der Auswirkung verschiedener negativer Tetrationskonstanten, die durch unterschiedliche Eingangsamplituden gleich viele Teiltöne aufweisen, wurden in Abbildung 23 zwei der Kombinationen dargestellt, bei denen der 25. Teilton jeweils ungefähr -77,5dBFS hat. Man kann erkennen, dass bei Tetrationskonstanten mit größerem Betrag der Grundton schwächer ist und die Teiltöne von stärker nach schwächer abnehmen. Ein vermeintlicher Zusammenhang zwischen Stärke der

Teiltondifferenz zu der Differenz der Tetrationskonstanten hat sich bei anderen Kombinationen[14] als nicht richtig herausgestellt.

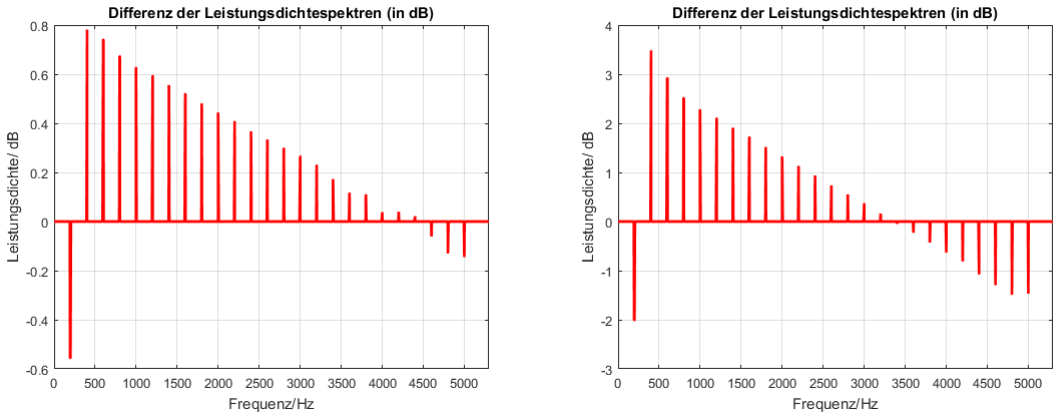

Abbildung 23: LDS $[c = -0{,}5;\ g = 0{,}677]$ minus LDS $[c = -0{,}3;\ g = 1{,}152]$ (links) und LDS $[c = -0{,}8;\ g = 0{,}228]$ minus LDS $[c = -0{,}1;\ g = 2{,}19]$ (rechts) für f=200Hz

Abschließend sind in Tabelle 4 noch die Teiltonstärken der Leistungsdichtespektren für $c = -0{,}5$ mit einer Amplitude von 55% bis 95% der fast Maximalamplitude 0,695 dargestellt, um die ungefähre Entwicklung der Leistungsdichtespektren bei negativer Tetrationskonstante auch für geringere Eingangsamplituden aufzuzeigen.

TT/g	0,382	0,417	0,452	0,486	0,521	0,556	0,591	0,626	0,660
1.	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
2.	-17,06	-16,12	-15,20	-14,30	-13,39	-12,46	-11,48	-10,39	-9,06
3.	-31,57	-29,69	-27,88	-26,10	-24,32	-22,49	-20,57	-18,44	-15,86
4.	-44,49	-41,69	-38,99	-36,34	-33,69	-30,98	-28,12	-24,98	-21,15
5.	-55,60	-52,89	-49,29	-45,77	-42,25	-38,65	-34,87	-30,70	-25,64
6.	-69,51	-65,84	-60,39	-55,98	-51,60	-47,12	-42,41	-37,21	-30,91
7.		-74,79	-69,40	-64,15	-58,90	-53,53	-47,88	-41,66	-34,12
8.				-74,25	-68,73	-62,48	-55,29	-48,04	-39,26
9.					-75,01	-67,88	-60,36	-52,10	-42,08
10.						-75,28	-66,85	-57,54	-46,29
11.							-72,64	-62,29	-49,81
12.							-78,21	-66,96	-53,19
13.								-72,44	-57,48
14.								-76,29	-60,08
15.									-64,72
16.									-67,00
17.									-70,92
18.									-73,94
19.									-77,10

Tabelle 4: relevante Teiltöne des Ausgangssignals bei Waveshaping mit Tetration für Eingangssignalamplituden g von 0,328 bis 0,660 und einer Tetrationskonstante von -0,5

[14] Z.B. ist die stärkste Differenz bei Tetrationskonstanten -0,3 und -0,1 größer als 2,2dB.

4.5 Unterschiede bei positiver und negativer Tetrationskonstante

Leistungsdichtespektren mit Tetrationskonstanten größer als null sehen eher konvex aus[15] (viele und starke Obertöne), während bei negativem c ein konkaves Erscheinungsbild entsteht (viele Obertöne aber weniger stark, vor allem im mittleren relevanten Teiltonbereich). Dieser Eindruck verstärkt sich jeweils mit steigender Eingangsamplitude und größerem Betrag von c. In Abbildung 24 werden Spektren verglichen, die jeweils 14 Teiltöne im relevanten Frequenzbereich besitzen. Dabei sieht man, dass der Unterschied vor allem bei den mittleren Obertönen und der Stärke des Grundtons liegt.

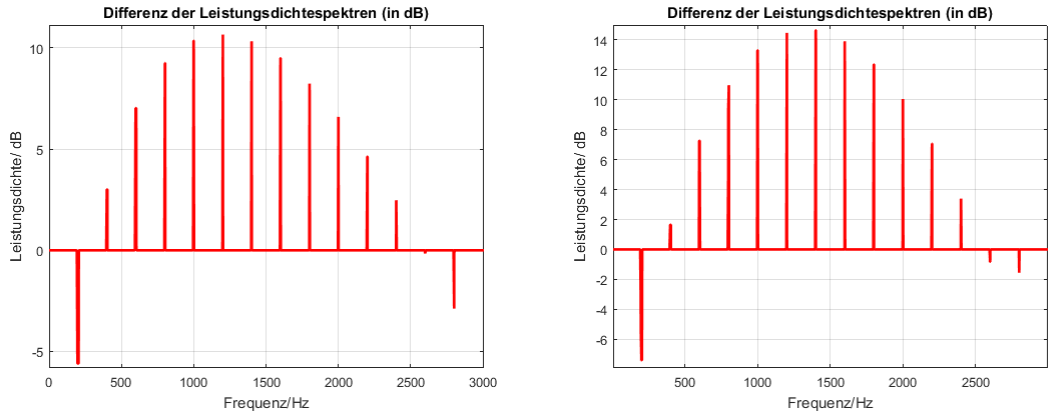

Abbildung 24: Spektrum $[c = 2{,}5;\ g = 1]$ minus Spektrum $[c = -0{,}5;\ g = 0{,}6255]$ (links) und Spektrum $[c = 1;\ g = 10]$ minus Spektrum $[c = -0{,}9;\ g = 0{,}1]$ (rechts) bei $f = 200Hz$

4.6 Betrachtung des Zeitverhaltens des Ausgangssignals

Bislang wurden in diesem Kapitel nur Sinusschwingungen mit stationärer Amplitude betrachtet. In Abbildung 25 sieht man das Ausgangssignal, wenn das Eingangssignal linear über hundert Millisekunden von null zu eins eingefadet wird, ohne dass am Ende das Ausgangssignal auf den Wertebereich ±1 skaliert wird.

[15] Bezogen auf eine Fläche unterhalb der Trendlinie der Teiltonstärken.

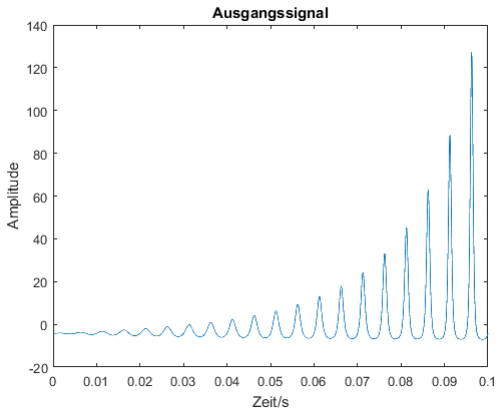

Abbildung 25: Ausgangssignal bei $[c = 2{,}5;\ g = Fade(0-1);\ f = 200]$

Man kann erkennen, dass die negativen Maximalwerte vom Betrag sehr gering sind während die positiven Werte mit steigender Amplitude immer extremer werden. Der Effekt ist, wie zu erwarten war, wie eine stärkere Version der Exponentialfunktion. Man kann den starken Amplitudenunterschied erkennen, die größte Amplitude beträgt (bei Abzug des Gleichanteils) 131,4 bzw. 42.4dB. Dies könnte bei Eingangssignalen mit großem Dynamikbereich ein Problem werden. In Abbildung 26 wurde die Aufnahme eines E-Gitarrenriffs mit derselben Tetration versehen.

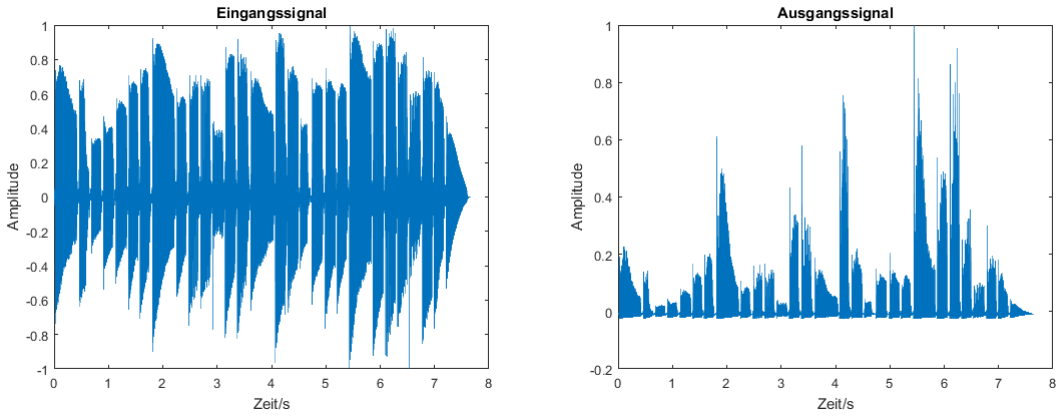

Abbildung 26: E-Gitarrenriff als Eingangssignal (links) und das Ausgangssignal bei Waveshaping mit Tetrationskonstante 2,5 und Skalierung des Ausgangssignals auf ±1 (rechts)

Selbst bei der relativ geringen Dynamik dieses Klangbeispiels ist das Ausgangssignal stellenweise sehr leise und poppt an den etwas lauteren Stellen stark auf.

Durch ein z.B. auf 0,5 skaliertes Eingangssignal kann dieses Problem im Zeitbereich zwar behoben werden, jedoch wird dadurch auch die erwünschte spektrale Änderung durch das Waveshaping verringert (siehe u.a. 4.1).

4.7 Vibrato zur klanglichen Verbesserung

Für weitere Versuche zur Ermittlung präferierter Tetrationskonstanten und Eingangsamplituden wird in diesem Kapitel eine Methode aufgezeigt, mit der die Audiosignale einen subjektiv wohltuenderen Klang bekommen könnten. Mit einem Vibrato kann ein synthetischer Klang mehr Komplexität und mit Hilfe von Zufallseigenschaften auch Natürlichkeit erlangen. Ein Vibrato ist eine Frequenzmodulation mit einer Modulationsfrequenz von ungefähr 6Hz [13].

In das Eingangssignal x (in diesem Beispiel ein Sinus) wird das Vibratosignal v zur Frequenz addiert, wobei v wiederum selbst ein Sinussignal ist.

$$\mathrm{x} = \mathrm{g} \sin\left(2\pi(f + v)t\right) \tag{25}$$

$$v = g_v \sin\left(2\pi f_v t\right)$$

Dabei wird zusätzlich zur einstellbaren Amplitude und Frequenz des Vibratos ein Zufallsanteil g_z bzw. f_z hinzugefügt. Dieser wird erzeugt, in dem normalverteilte Pseudozufallszahlen mit geringerer Abtastfrequenz generiert werden. Diese werden tiefpassgefiltert, auf die für das Eingangssignal genutzte Abtastfrequenz durch Interpolation der Zwischenwerte gestreckt und noch einmal tiefpassgefiltert. Dies ist nötig, um Störgeräusche durch zu starke Änderung des Vibratosignals vorzubeugen. Die Ergebnisse der Tiefpassfilterung werden mit einer Variablen zum Einstellen der Streuung des Zufallsgehalt a_z multipliziert und als Exponent der natürlichen Exponentialfunktion benutzt, um gleichverteilte Werte um 1 zu bekommen (siehe Anhang 3.5). Danach werden sie jeweils mit der einstellbaren Vibratoamplitude bzw. der Vibratofrequenz multipliziert:

$$v = g_v \, g_z \sin\left(2\pi f_v f_z t\right) \tag{26}$$

Bei einem Vibrato bieten sich auch Fades der Vibratostärke g_v an, wodurch auch diese zeitabhängig wird. Die Momentanfrequenz bei der Frequenzmodulation berechnet sich wie folgt:

$$f + \frac{dv}{dt} = f + \frac{d}{dt}\left[2\pi g_v(t) g_z(t) \sin(2\pi f_v f_z(t) t)\right] \tag{27}$$

Unter der Annahme, dass die Zufallssignale $f_z(t)$ und $g_z(t)$ sich durch die zweifache Tiefpassfilterung und Interpolation nur sehr langsam ändern, können ihre Ableitungen vernachlässigt werden:

$$\approx f + \frac{dg_v(t)}{dt} g_z(t) \sin(2\pi f_z(t) f_v) + 2\pi g_v(t) g_z(t) f_z(t) f_v \cos(2\pi f_z(t) f_v t) \quad (28)$$

Abbildung 27 zeigt das Spektrogramm bei Waveshaping mit Tetration mit und ohne Vibrato.

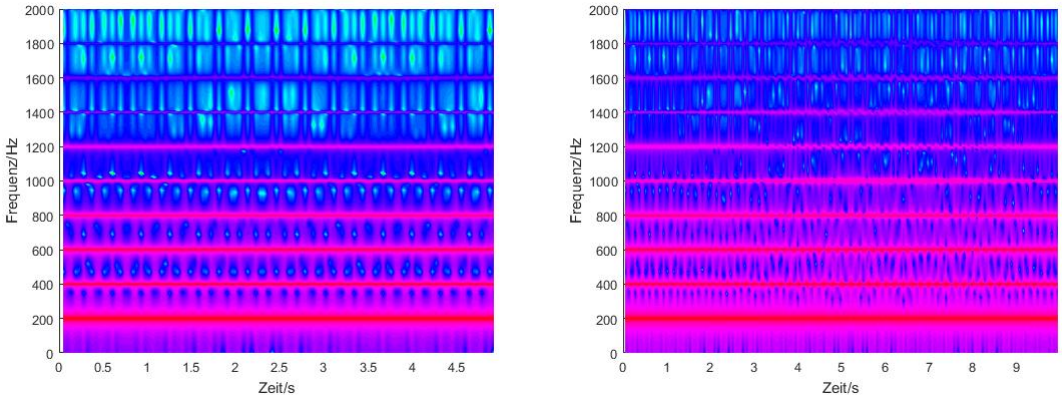

Abbildung 27: Ausschnitt des Spektrogramms bei LDS $[c = 2{,}5;\ g = 1;\ f = 200]$ ohne Vibrato (links) und mit Vibrato beim Eingangssignal (rechts). Der Koeffizient der Zufallsstärke a_z vor der Exponentialfunktion beträgt 0,015, g_v ist eine Dreiecksfunktion mit Maximalwert 0,01 und einer Periodendauer von 10s.

Um ein Ausgangssignal mit gut hörbarem Vibrato ohne Störgeräusche zu erzeugen, haben sich $g_v = 0{,}01$ und $a_z = 0{,}015$ als geeignete Werte erwiesen.

5 Schlussfolgerungen

Es wurden verschiedene Waveshapingformen erläutert, um ein Verständnis für diese Art der Klangerzeugung zu vermitteln, bevor die nötige Mathematik der verwendeten Tetrationsformel erklärt wurde.

Die in dieser Bachelorarbeit aufgezeigte Methode der Tetration zum Waveshaping bietet drei Ansätze, durch welche die erzeugten Spektren gesteuert werden können. Die Beträge der Tetrationskonstante und der Eingangsamplitude stellen ein, wie viele Teiltöne das Spektrum des Ausgangssignals besitzen soll. Durch das Verhältnis der beiden können bei positiven Tetrationskonstanten die hohen Obertöne angepasst werden. Bei negativen Tetrationskonstanten kann durch Anpassen dieses Verhältnisses auch das Ausgangssignal verändert werden, jedoch ändert sich dadurch das komplette Spektrum. Die Wahl zwischen positiver und negativer Tetrationskonstante ist grundsätzlich die zwischen stärkeren bzw. schwächeren mittleren Obertönen.

Die Probleme des Waveshapings mittels Tetration wurden im Bezug auf die Empfindlichkeit der Eingangsamplitude bei negativen Tetrationskonstanten und auf die Dynamik des Ausgangssignals erläutert. Mit der Verwendung eines Vibratos wurde ein Optimierungsvorschlag gemacht, mit dem eine subjektive Verbesserung des Klangs möglich ist.

Eine einleitende Frage, ob diese Art des Waveshapings Einzug ins Repertoire von Sounddesignern und Musikern erlangen sollte, konnte nicht ganz geklärt werden. Die subjektive Meinung des Autors ist, dass der Klang bei Waveshaping aufgenommener Signale nicht wünschenswert ist und bei synthetisch erzeugten Signalen andere Waveshapingformen existieren, die einfacher zu implementieren (Tangens Hyperbolicus) oder bessere Kontrolle bieten (Chebychev Polynome 1. Ordnung). Um diese Frage objektiv zu klären, müssten jedoch statistisch relevante Hörversuche durchgeführt werden. Zur weiteren Forschung an diesem Thema und für Künstler, die nach neuen Klangformen suchen, wurde durch die Erkenntnisse und die MATLAB Skripte dieser Bachelorarbeit ein vereinfachter Einstieg in die Materie ermöglicht.

6 Literatur

[1]C. Roads, J. Strawn, *Foundations of Computer Music*, Seite 83-90, 1985, ISBN: 0-262-181142

[2]A. Farnell, *Designing Sound*, Seite 282 & 287, 2010, ISBN 978-0-262-01441-0

[3]http://www.electronics-tutorials.ws/diode/diode-clipping-circuits.html (zuletzt aufgerufen am 30.11.2016)

[4]D. T. Yeh, J. Pakarinen, *A review of digital techniques for modeling vacuum-tube guitar amplifiers*, Computer Music Journal Summer 2009, vol. 33, no. 2, 2009

[5]S. Peley, *Frequency Domain Guitar Effects Processor*, https://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2007to2008/sjp46/Part2/website/index.html (zuletzt aufgerufen am 07.01.2017, für diese Arbeit übersetzt)

[6]P. Taylor, *Guitar preamp tone explained by Effectrode founder Phil Taylor*, in: Tone Quest Report, vol. 15, no. 5, 2014

[7]D. T. Yeh, J. S. Abel and J. O. Smith, *SIMPLIFIED, PHYSICALLY-INFORMED MODELS OF DISTORTION AND OVERDRIVE GUITAR EFFECTS PEDALS*. - Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07), Bordeaux, France, 2007

[8]D. Y. Kuznetsov, *Tetrational as special function*, Vladikavkaz. Mat. Zh., **12**:2 (2010), 31-45

[9]https://en.wikipedia.org/wiki/Tetration (zuletzt aufgerufen am 30.11.2016)

[10]R. L. Goldstein, *Fundamental concepts of mathematics*, Seite 41-42, 1962, ISBN 0-08-021665-8

[11]http://en.citizendium.org/images/3/39/Expc.jpg (zuletzt aufgerufen am 07.01.2017)

[12]E. Zwicker, H. Fastl, *Psychoacoustics - Facts and Models*. Springer Verlag, Berlin 1990, Seite 175ff

[13]C. Roads, *The Computer Music Tutorial (Technology)*, Mit Press Ltd, 1996, Seite 239

Anhang mit verwendeten MATLAB Skripten

Die in dieser Arbeit verwendeten MATLAB Skripte sind im Folgenden wiedergegeben. Die Abschnittsnamen (z.B. bei 1.1 „tet.m“) sind die Namen der MATLAB Skipte, der Text kann direkt in ein leeres Skript kopiert werden und hat für die Standardeinstellung die richtige Formatierung, weswegen die Formatierung in diesem Dokument teilweise unüblich scheint.

In Abschnitt 1 sind die zur natürlichen Tetration notwendigen Skripte und in Abschnitt 2 die zur inversen natürlichen Tetration. Diese stammen von Thomas Mayr und wurden vom Autor nur durch Kommentierungen verändert.

In Abschnitt 3 befinden sich Skripte zur Klangerzeugung durch Waveshaping mit Tetration. Dabei kann entweder mit „Tetrasound.m“ ein einzelner Klang erstellt werden oder es können mit 3.2 & 3.3 mit einem Aufruf mehrere Klänge erzeugt werden, bei denen die Tetrationskonstante oder die Eingangsamplitude bei jedem neuen Klang auf immer gleiche Art verändert wird. Außerdem können dort automatisch die Leistungsdichtespektren als Bilddatei gespeichert werden. Zur Erweiterung der Klangerzeugung sind auch Skripte vorhanden, durch die ein Fade oder Vibrato zum Eingangssignal hinzugefügt werden kann. Diese müssen im gleichen Ordner vorhanden sein, auch wenn sie nicht angewendet werden.

In Abschnitt 4 befindet sich ein Skript für Waveshaping mit Hard Clipping, Tangens Hyperbolicus, Sinusfunktion oder Chebychev Polynomen 1. Ordnung sowie ein Skript, durch welches ein Distortion Pedal imitiert wird.

Das Skript in Abschnitt 5 berechnet die Teiltonstärken bei Waveshaping mit der Exponentialfunktion und speichert diese in einem Vektor.

In Abschnitt 6 sind Skripte zur Erstellung von Abbildungen der Differenz von zwei Leistungsdichtespektrenen, eines Leistungsdichtespektrums oder eines Spektrogramms bzw. Wasserfalldiagramms.

1 Funktionen für die natürliche Tetration

1.1 tet.m

```
function [s] = tet(z)
%Aufruf [s] = tet(3);
%Berechnet die natürliche Tetration von z, also „z-mal“ e^e^e^...
%Dabei kann z eine komplexe Zahl sein, für die Berechnungen beim
%Waveshaping werden jedoch nur reelle Zahlen genutzt.
%Eingabeparameter: z   "Anzahl" der Iterationen
%Ausgabeparameter: s   Ergebnis der natürlichen Tetration von z
 y = imag(z);
%Bei reellen Zahlen kann gleich zu maclo_1(z) gesprungen werden.
if (y > 4.5)
   %disp('tet > 4.5');
   s = fima_1(z);
elseif(y > 1.5)
   %disp('tet > 1.5');
   s = tai3_1(z);
elseif(y > -1.5)
   %disp('tet > -1.5');
   s = maclo_1(z);
elseif(y > -4.5)
   %disp('tet > -4.5');
   s = conj(tai3_1(conj(z)));
else
   %disp('tet else');
   s = conj(fima_1(conj(z)));
end
```

1.2 fima_1.m

```
function [z1] = fima_1(z)
%FIMA, nur für komplexe Eingangssignale
x = real(z);
y = imag(z);
```

```
if(y < 0.2379 * x)
   z1 = exp(fima_1(z-1.));
   return;
else
   z1 = fima(z);
   return;
end
end
```

1.3 fima.m

```
function [c] = fima(z)
%fima, nur für komplexe Eingangssignale
 Zo = complex(.31813150520476413, 1.3372357014306895);  % b = e
 Zc = complex(.31813150520476413,-1.3372357014306895);  % b = e
 R = complex(1.0779614375280, -.9465409639480);
 a2=.5/(Zo-1.);
 a3=(a2+1./6.)/(Zo*Zo-1.);
 a4=(a2/2.+a2*a2/2.+a3+1./24.)/(Zo*Zo*Zo-1.);
 a5=(.5*a2*a2+a2/6.+a2*a3+a3/2.+a4+1./120.)/(Zo*Zo*Zo*Zo-1.);
 Li=2*pi*1i;
 b0=complex(0.1223, -0.02370);
 e=exp(Zo*z+R);
 c = Zo + e*( 1. + e*(a2+e*(a3+e*(a4+e*a5))) + b0*exp(Li*z));
 end
```

1.4 tai3_1.m

```
function [z1] = tai3_1(z)
%FIMA, nur für komplexe Eingangssignale
x = real(z);
if(x > 0.5)
   z1 = exp(tai3_1(z-1.));
elseif(x < -0.5)
   z1 = log(tai3_1(z+1.));
```

```
else
   z1 = tai3(z);
end
end
```

1.5 tai3.m

```
function [s] = tai3(z)
%tai3, nur für komplexe Eingangssignale
 K = 50;
DER3 = [complex(0.37090658903228507226, 1.33682167078891400713)
complex( 0.03660096537598455518, 0.13922215389950498565)
complex(-0.16888431840641535131, 0.09718533619629270148)
complex(-0.12681315048680869007,-0.11831628767028627702)
complex( 0.04235809310323926380,-0.10520930088320722129)
complex( 0.05848306393563178218,-0.00810224524496080435)
complex( 0.02340031665294847393, 0.01807777011820375229)
complex( 0.00344260984701375092, 0.01815103755635914459)
complex(-0.00803695814441672193, 0.00917428467034995393)
complex(-0.00704695528168774229,-0.00093958506727472686)
complex(-0.00184617963095305509,-0.00322342583181676459)
complex( 0.00054064885443097391,-0.00189672061015605498)
complex( 0.00102243648088806748,-0.00055968657179243165)
complex( 0.00064714396398048754, 0.00025980661935827123)
complex( 0.00010444455593372213, 0.00037199472598828116)
complex(-0.00011178535404343476, 0.00016786687552190863)
complex(-0.00010630158710808594, 0.00002072200033125881)
complex(-0.00005078098819110608,-0.00003575913005741248)
complex(-0.00000314742998690270,-0.00003523185937587781)
complex( 0.00001347661344130504,-0.00001333034137448205)
complex( 0.00000980239082395275, 0.00000047607184151673)
complex( 0.00000355493475454698, 0.00000389816212201278)
complex(-0.00000021552652645735, 0.00000296273413237997)
complex(-0.00000131673903627820, 0.00000097381354534333)
```

```
complex(-0.00000083401960806066,-0.00000018663858711081)
complex(-0.00000022869610981361,-0.00000037497716770031)
complex( 0.00000005372584613379,-0.00000023060136585176)
complex( 0.00000011406656653786,-0.00000006569510293486)
complex( 0.00000006663595460757, 0.00000002326630571343)
complex( 0.00000001396786846375, 0.00000003315118300198)
complex(-0.00000000684890556421, 0.00000001713041981611)
complex(-0.00000000916619598268, 0.00000000403886083652)
complex(-0.00000000502933384276,-0.00000000222121299478)
complex(-0.00000000084484352792,-0.00000000273668661113)
complex( 0.00000000070086729861,-0.00000000124687683156)
complex( 0.00000000070558101710,-0.00000000021962577544)
complex( 0.00000000035900951951, 0.00000000018774741308)
complex( 0.00000000005248658571, 0.00000000021201177126)
complex(-0.00000000006264758835, 0.00000000009059171879)
complex(-0.00000000005333473585, 0.00000000001006078866)
complex(-0.00000000002432138144,-0.00000000001506937008)
complex(-0.00000000000331880379,-0.00000000001544700067)
complex( 0.00000000000501652570,-0.00000000000658967459)
complex( 0.00000000000401214135,-0.00000000000036708383)
complex( 0.00000000000158629111, 0.00000000000119885992)
complex( 0.00000000000019668766, 0.00000000000106532662)
complex(-0.00000000000036355730, 0.00000000000047229527)
complex(-0.00000000000029920206, 0.00000000000001251827)
complex(-0.00000000000010305550,-0.00000000000009571381)
complex(-0.00000000000000910369,-0.00000000000007087680)
complex( 0.00000000000002418310,-0.00000000000003240337)
];
z = z - complex(0,3);
z = z / 2.;
t = 1.;
s = 0;
for k = 1:K
   s = s + DER3(k) * t;
```

```
    t = t * z;
end
end
```

1.6 maclo_1.m

```
function [z1] = maclo_1(z)
%Aufruf: [z1] = maclo_1(3);
%Es wird die Tetration solange durchgeführt, also
%solange erneuter Aufruf mit z plus bzw. minus 1, wobei das Ergebnis durch
%e^c bzw. log(c) erweitert wird, bis Re(z) im Bereich ]-0.5;0.5[ ist.
%Dann ist das Ergebnis nahe genug an 0, um die Maclaurin Reihe
%durchzuführen (in maclo(z)).
x = real(z);
c = 0;
 if(x > 0.5)
    %disp('maclo > 0.5');
    c = z - 1;
    c = maclo_1(c);
    c = exp(c);
    z1 = c;
    return;
elseif(x < -0.5)
    %disp('maclo < -0.5');
    z1 = log(maclo_1(z+1));
    return;
else
    %disp('maclo else');
    z1 = maclo(z);
    return;
end
```

1.7 maclo.m

```
function [s] = maclo(z)
%Aufruf: [s] = maclo(3);
%Führt die Maclaurin Reihe durch
%Formel: f(z) = log(z+2)+sum(n=0-100)[d*(x/2)^2]
%Vektor d besitzt die Koeffizienten der Reihe von n = 0 bis 109
 d = [0.30685281944005469058
 1.18353470251664338875
 1.58593285160678321155
 1.36629265207672068172
 1.36264601823980036066
 1.21734246689515424045
 1.10981816083559525765
 0.96674692974769849130
 0.84089872598668435888
 0.71353210966804747617
 0.60168548504001373445
 0.49928574281440518678
 0.41140086629121763728
 0.33506195665178500898
 0.27104779243942234146
 0.21728554054610033086
 0.17311050207880035456
 0.13690016038526570119
 0.10765949732729711286
 0.08413804539743192923
 0.06542450487497340761
 0.05060001212013485322
 0.03895655493977817629
 0.02985084640296329153
 0.02277908979501017117
 0.01730960309240666892
 0.01310389615589767874
```

0.00988251130733762764

0.00742735935367278347

0.00556296426263720549

0.00415334478103463346

0.00309116153137843543

0.00229387529664008653

0.00169729976398295653

0.00125245885041635465

0.00092172809095368547

0.00067661152429638357

0.00049544127485341987

0.00036192128589181518

0.00026376927786672476

0.00019180840045267570

0.00013917553105723647

0.00010077412023867018

0.00007281884753121133

0.00005251474516228446

0.00003779882770351268

0.00002715594536867241

0.00001947408515177282

0.00001394059355016322

0.00000996213949015693

0.00000710713872292710

0.00000506199803708578

0.00000359960968975399

0.00000255569149787694

0.00000181175810338313

0.00000128245831538430

0.00000090647322737496

0.00000063980422418981

0.00000045095738191441

0.00000031741772125007

0.00000022312521183625

0.00000015663840476155
0.00000010982301013230
0.00000007690305934973
0.00000005378502675604
0.00000003757126131521
0.00000002621429405247
0.00000001826909956818
0.00000001271754463425
0.00000000884310192977
0.00000000614230041407
0.00000000426177146865
0.00000000295386817285
0.00000000204522503591
0.00000000141464900426
0.00000000097750884878
0.00000000067478454029
0.00000000046535930671
0.00000000032062550784
0.00000000022069891976
0.00000000015177557961
0.00000000010428189463
0.00000000007158597119
0.00000000004909806710
0.00000000003364531769
0.00000000002303635851
0.00000000001575933679
0.00000000001077213757
0.00000000000735717912
0.00000000000502077719
0.00000000000342362421
0.00000000000233271256
0.00000000000158818623
0.00000000000108046566
0.00000000000073450488

```
 0.00000000000049894945
 0.00000000000033868911
 0.00000000000022973789
 0.00000000000015572383
 0.00000000000010548054
 0.00000000000007139840
 0.00000000000004829557
 0.00000000000003264619
 0.00000000000002205299
 0.00000000000001488731
 0.00000000000001004347
 0.00000000000000677124
 0.00000000000000456225
 0.00000000000000307196
 0.00000000000000206720];
 K = 101;
 s = 0;
z2 = z / 2;
t = 1.;
for k = 1:K
    s = s + d(k) .* t;
    t = t .* z2;
end
s = s + log(z + 2.);
```

2 Funktionen für die inverse natürliche Tetration

2.1 ate.m

```
function [s] = ate(z1)
%Aufruf: s = ate(1);
%Berechnet die zur natürlichen Tetration inverse Funktion, z.B. ate(e^e)=2.
%Erstes if könnte zum Waveshaping mit reellen Zahlen gestrichen werden, da
%Imag(x)=0 nie kleiner als Imag(Zo)=1.337.
```

```
%Eingabeparameter: z1  Ergebnis von tet(s)
%Ausgabeparameter: s   Eingang der natürlichen Tetration, damit das
%                      Ergebnis z1 ist
Zo = complex(.31813150520476413, 1.3372357014306895);
Zc = complex(.31813150520476413,-1.3372357014306895);
x = real(z1);
y = imag(z1);
 if(abs(y) > abs(imag(Zo)))
    s = ate(log(z1))+1.;
    return;
elseif(x < real(Zo))
    s = ate(exp(z1))-1.;
    return;
elseif(x > 2)
    s = ate(log(z1))+1.;
    return;
end
z = z1 - 1;
LZ = log(z1);
if(abs(LZ - 1) < abs(z))
    s = ate(LZ) + 1.;
    return;
else
    s = slo(z1);
    return;
end
end
```

2.2 slo.m

```
function [s] = slo(z1)
%Näherung des Superlogarithmus, DE sind die Koeffizienten der Näherung
 DE = [
1.4192252155045112363
```

-0.05213258059503801667
0.00693219127232187586
-0.00015617045803377859
-0.00100912103192385785
0.00082172671942507903
-0.00035776641706493177
-0.00000931803078422933
0.00016678111348857047
-0.00014181446429806932
0.00003186488716454018
0.00006022937595596333
-0.00007769822429012910
0.00002881816919640196
0.00003346765914794806
-0.00005635940084759932
0.00002613708927959275
0.00002533341053138444
-0.00005010441151688034
0.00002593810263640952
0.00002404611936084357
-0.00005163238246428602
0.00002794638872473000
0.00002700739592764804
-0.00005939035114511979
0.00003210955504312860
0.00003438232223525011
-0.00007428278434380886
0.00003866302665809861
0.00004803799918127077
-0.00009914141292927652
0.00004800025436154043
0.00007191960183392704
-0.00013922522917537457
0.00006043126908005670

0.00011338211995240744
-0.00020351111316586852
0.00007562971718596908
0.00018585637209671475
-0.00030700846364341576
0.00009132512919756623
0.00031386108850653502
-0.00047464470561729965
0.00010030770871287629
0.00054232170050706079
-0.00074759603610534669
0.00008375204845585605
0.00095389423083896800
-0.00119336225449479362
-0.00000327383738614825
0.00170107934819055851
-0.00192109516273315209
-0.00026290310001950487
0.00306590657916818192
-0.00310372506294090238
-0.00091982425407694250
0.00556979490215834833
-0.00500546835451257978
-0.00245869651201214212
0.01017705593773498771
-0.00800820238034244403
-0.00590649361431362999
0.01866321477729812259
-0.01260156096677063874
-0.01341963601206602220
0.03429254345271898208
-0.01926894593144593687
-0.02946277663641767158
0.06300065960337521143

-0.02800532706641396460
-0.06325609033757989552
0.11556117355587923468
-0.03708367328869965895
-0.13352664223772733876
0.21104853030721187901
-0.03941069742436464907
-0.27853504829255953945
0.38331715278474703945
-0.01491397058539710788
-0.57446133459038406510
0.68905734940479856920
0.09065013779953061401
-1.17542205931169707611
1.22536662105515059551
0.40500835675024698945
-2.37977019901803332758
2.13411821810153146117
1.24184396615612624437
-4.78947531960227212977
3.64093109251482172084
3.27095016057312193425
-9.53051815711462246838
5.92750355113636295812
8.12068845726284394004
-18.89123486907114468636
9.07245090167984002960
18.99435214920948311601
-36.62201395750987842348
12.69160696640316032813
43.73569046442687380249
-71.43155879446639744401
14.83661067193675719977
95.94011857707508283966

```
-135.28537549407113260713
 4.55415019762845751927
212.46383399209483400227
-258.45286561264816782568
-34.35533596837944259050
440.37608695652170354151
-466.49328063241102881875
-184.78893280632408391284
969.10988142292478642048
-888.80079051383393107244
-485.21897233201576682404
1912.15652173913008482486
-1381.80553359683767666866
-1490.15335968379417863616
4216.82213438735107047250
-2565.99525691699591334327
-3155.47826086956501967506
7689.57470355731129529886
-3206.47588932806274897303
-8547.41501976284416741692
17971.70592885375299374573
-7203.41501976284507691162
-16388.65454545454485923983
28589.12885375493715400808
-2355.80711462450562976301
];
 L = complex(0.3181315052047641353, 1.337235701430689409);
Zo = complex(.31813150520476413, 1.3372357014306895);
Zc = complex(.31813150520476413, -1.3372357014306895);
 z = z1 - 1;
z = z / 2;
s = 0;
K = 128;
for k = K-1:-1:2
```

```
    s = s + DE(k);
    s = s .* z;
end
s = s + DE(1);
s = s + log(z1-Zo)/Zo + log(z1-Zc)/Zc;
end
```

3 Klangerzeugung durch Waveshaping mit Tetration

3.1 Tetrasound.m

```
%Erstellt und speichert eine Sounddatei, bei der ein Sinussignal oder
%ein Signal aus einer eingelesenen Audiodatei durch die Waveshaping-
%Transferfunktion tet(c+ate(x)) läuft.
%Möglich sind Fades für c und für die Stärke des Eingangssignals (linear
%oder gegeben durch die Transferkennlinie). Außerdem kann ein Vibrato
%hinzugefügt werden, das entweder linear oder mit der Transferkennlinie
%zu- und wieder abnimmt. Beim Vibrato wird noch für natürlicheren Klang
%ein Zufallsgehalt hinzugefügt.
 %Grundlegendes:
fs = 44100;    %Samplefrequenz in Hz
t = 0:1/fs:10; %Zeitvektor in Sekunden
c = 2;         %Einstellen Tetrationskonstante c; in der
               %Waveshapingschleife c zu c(i) umschreiben und c = t/k
               %für einen Fade der Tetrationskonstante
g = 1;         %Amplitude des Eingangssignals
nameoutput = 'Tetrac2g1.wav'; %Name der zu speichernden Datei
 %Eingangssignal:
s = 1;                  %Input Sinus 1, Audiodatei einlesen 2
f= 200;                 %Grundfrequenz in Hz bei Sinus
nameinput = 'Audiodatei.wav'; %Name einzulesender Datei
 %Fade hinzufügen:
dofade = 0; %kein Fade 0, linearer Fade 1, Fade mit Transferfunktion 2
fadex = Fadefunction(dofade,t,c);
```

```
 %Vibrato hinzufügen:
dovib = 0; %kein Vibrato 0, steigend mit Dreieck 1, steigend mit Transfer 2
vibf = 5;           %Vibratofrequenz
vibmaxstr = 0.02;  %Amplitude des Modulationssinus
vibrandval = 0.015; %Gewichtung des Zufallsanteils
vib = Vibratofunction(dovib,vibf,vibmaxstr,vibrandval,t,c);
 %Erstellt das Eingangssignal:
if s == 1
    x = g*sin(2.*pi.*(vib+f).*t);
elseif s == 2
    [x,Fs] = audioread(nameinput);
    fadex = ones(1,length(x));
    x = g*x/max(abs(x));
end
 %Waveshaping:
y = zeros(1,length(x));
for i=1:length(x)
    y(i) = tet(c+ate(fadex(i).*x(i)));
end
y = y-mean(y); %Gleichanteil löschen
 %Skalierung um Übersteuerung zu vermeiden(mit extra Headroom):
skalier = max(abs(y))*1.1;
y = y/skalier;
 %Ausschreiben des Audiofiles:
audiowrite(nameoutput,y,fs);
```

3.2 Mehrere_Sounds.m

```
%Dieses Skript vereinfacht die Steuerung von Tetrasoundfunction.m
%zum Erstellen mehrerer Tetrationsaudiodateien mit sich immer gleich
%verändernden Werten.
 %Grundlegendes:
fs = 44100;     %Samplefrequenz in Hz
t = 0:1/fs:5;  %Zeitvektor in Sekunden
```

```
f = 200;        %Grundfrequenz in Hz
c = 2.75;       %Einstellen der ersten Tetrationskonstante c
g = 1;          %erste Amplitude des Eingangssignals
 %Eingangssignal:
s = 1;    %1 für einen Sinus, sonst Einlesen einer Audiodatei
Dateinamein = 'Sound.wav'; %Name der Audiodatei
dofade = 0; %kein Fade 0, linearer Fade 1, Fade mit Transferfunktion 2
dovib = 0; %kein Vibrato 0, steigend mit Dreieck 1,steigend mit Transfer 2
vibf = 5;           %Vibratofrequenz
vibmaxstr = 0.02;    %Vibratostärke
vibrandval = 0.015;  %Gewichtung des Zufallsanteils
print = 0; %Speichern des Leistungsdichtepektrums: ja 1, nein sonst,
          %Namen und Fensterwerte in Tetrasoundfunction überprüfen!
deltaF = 2500; %Frequenzbereich der Abbildung
 %Formatbeispiel (Name der Leistungsdichtespektrenbilder):
 %c=2, g=13: Tetrac2g13.wav
for i = 1:4     %1:a, a = Anzahl erstellter Audiobeispiele
   Dateinameout = strcat('Tetrac',num2str(c),'g',num2str(g),'.wav');
Tetrasoundfunction(Dateinameout,c,g,fs,t,f,dofade,dovib,vibf,vibmaxstr,vibrandval,Dat
einamein,s,print,deltaF);
   c = c + 0.25; %Änderung von c bei jedem neuen Soundfile
   g = g + 0; %Änderung von g bei jedem neuen Soundfile
end
```

3.3 Tetrasoundfunction.m

```
function [] =
Tetrasoundfunction(Dateinameout,c,g,fs,t,f,dofade,dovib,vibf,vibmaxstr,vibrandval,Dat
einamein,s,print,deltaF)
%Aufruf:
Tetrasoundfunction(Tetrac1g2.wav,1,2,44100,t,200,0,1,5,0.02,0.015,‘audio.wav‘,1,0,20
00)
%Erstellt und speichert eine Audiodatei, bei der ein Sinussignal oder
%ein Signal aus einer eingelesenen Audiodatei durch die Waveshaping-
```

```
%Transferfunktion y = tet(c+ate(x)) läuft.
%Möglich sind Fades für c und für die Stärke des Eingangssignals(linear
%oder gegeben durch die Transferkennlinie). Außerdem kann ein Vibrato
%hinzugefügt werden, das entweder linear oder mit der Transferkennlinie
%zu- und wieder abnimmt. Beim Vibrato wird noch für natürlicheren Klang
%ein Zufallsgehalt hinzugefügt.
%Eingabeparameter: Dateinameout Name des zu erstellenden Audiofiles
%               c          Tetrationskonstante
%               g          Amplitude des Eingangssignals beim Shaping
%               fs         Samplefrequenz
%               t          Zeitvektor
%               f          Frequenz des Sinus bei Nutzung Sinus
%               dofade      Hinzufügen Fade 1 linear, 2 mit
%                          Tetrationskennlinie, sonst nicht
%               dovib       kein Vibrato 0, Vibrato in Deiecksform 1,
%                          Vibrato auf/ab mit Tetrationskennlinie 2
%               vibf        Vibratofrequenz
%               vibmaxstr   Maximale Auslenkung des Vibratosinus (ohne
%                          Einfluss der Zufallswerte)
%               vibrandval  Stärke des Zufallscharakters beim Vibrato
%               Dateinamein  Name der Sounddatei, die statt einem Sinus
%                          als Eingangssignal verwendet wird
%               s          1 um Sinus als Eingangssignal zu benutzen
%               print       1 um Leistungsdichtespektrum zu speichern
%               deltaF      Frequenzbereich des gespeicherten Spektrums
%Fade hinzufügen:
fadex = Fadefunction(dofade,t,c);
%Vibrato hinzufügen:
vib = Vibratofunction(dovib,vibf,vibmaxstr,vibrandval,t,c);
 %Eingangssignal erzeugen
if s == 1                       %Input Sinus
   x = g*sin(2.*pi.*(vib+f).*t);
else                            %Input Audiodatei
   [x,fs] = audioread(Dateinamein);
```

```
    fadex = ones(1,length(x));       %Für Audiodatei kein Fade im Programm
    x = g*x/max(abs(x));             %Aussteuern des Eingangssignals
end
 %Waveshaping:
y = zeros(1,length(x));
for i=1:length(x)
    y(i) = tet(c+ate(fadex(i).*x(i)));
end
 y = y-mean(y); %Gleichanteil löschen
 %Skalierung um Übersteuerung zu vermeiden(mit extra Headroom):
skalier = max(abs(y))*1.1;
y = y/skalier;
 %Ausschreiben des Audiofiles:
audiowrite(Dateinameout,y,fs);
 %Erstellen des Spektrums
if print == 1
    nfft = 65536;
    noverlap = 32768;
     % Leistungsdichtespektrum mit cpsd
    [Pxx,F] = cpsd(y,y,hanning(nfft),noverlap,nfft,fs,'twosided');
     % Darstellung des Spektrums (nur die Hälfte bis fs/2)
    N = floor(length(F)/2);
    Pxx = Pxx/max(abs(Pxx));
    Pxx = 10*log10(Pxx);
    %Ändere .wav zu .png beim Dateinamen
    Dateinameout = strcat('Tetrac',num2str(c),'g',num2str(g),'.png');
     %Erstelle Plot
    plot(F(1:N),Pxx(1:N),'r','LineWidth',2);
    xlabel('Frequenz/Hz')
    ylabel('Leistungsdichte/dB')
    title('Leistungsdichte (in dB)')
     %Zoom auf relevanten Bereich und speichern
    h=gcf;
    set(h.Children,'Xlim',[0 deltaF]);
```

```
    set(h.Children,'Ylim',[-80 0]);
    saveas(gcf,Dateinameout)
    grid
end
end
```

3.4 Fadefunction.m

```
function [fadex] = Fadefunction(dofade,t,c)
%Aufruf: [fadex] = Fadefunction(1,t,2);
%Erstellt einen Vektor, mit dem ein linear Fade oder ein mit der
%Transferfunktion der Tetration steigender Fade realisiert werden kann
%Eingabeparameter: dofade   Bestimmt Art des Fades: 1 linear, 2 mit
%                           Tetrationskennlinie, sonst kein Fade
%                  t        Zeitvektor
%                  c        Tetrationskonstante
%Ausgabeparameter: fadex    Fadevektor
 if dofade == 1
    fadex = t/t(end);          %linearer Fade
elseif dofade == 2
    fadex = -1:2/length(t):1;   %Transferfunktionsfade
    for i = 1:length(t)  %Erstelle Kennlinie für Werte von -1 bis 1
        fadex(i) = tet(c+ate(fadex(i)));
    end
    %Skalierung der Kennlinie von 0 bis 1
    fadex = fadex-min(fadex);
    fadex = fadex/max(abs(fadex));
else
    fadex = ones(1,length(t));   %kein fade,
end
end
```

3.5 Vibratofunction.m

```
function [vib] = Vibratofunction(dovib,vibf,vibmaxstr,vibrandval,t,c)
%Aufruf: [vib] = Vibratofunction(2,5,0.02,0.015,t,1.5);
%Erstellt einen Modulationssinus, der bei vibf im Bereich 6Hz durch
%sin(2*pi*t*(f+vib)) als Vibrato benutzt werden kann.
%Eingabeparameter: dovib:      Einstellen Vibrato: kein Vibrato 0,
%                              steigend/fallend mit Dreieck 1,
%                              steigend/fallend mit Transferfunktion 2
%                  vibf:       Vibratofrequenz
%                  vibmaxstr:  Maximale (ohne Zufallsgehalt) Amplitude
%                              des Modulationssinus
%                  vibrandval: Verändern der Spannweite zufälliger Werte
%Ausgabeparameter: vib:        Modulationssinus
 if dovib == 1
    %Dreiecksfunktion
    %Steigende Flanke bis 1 mit halb so vielen Samples wie t:
    rising = 0:2/length(t):1;
    falling = fliplr(rising);  %Fallende Flanke
    falling(1) = []; %Lösche Spitzenwert (schon in rising vorhanden)
    dreieck = [rising falling]; %Zusammenfügen steigend/fallend
    vibstr = vibmaxstr.*dreieck; %Skallieren mit max. gewünschter Amplitude
elseif dovib == 2
    %Transferfunktion
    %Steigende Flanke mit halb (2*1/4) so vielen Samples wie t:
    rising = -1:4/length(t):1;
    risingtet = zeros(1,length(rising));
    for i = 1:length(rising) %Bilde Kennlinie für Werte von -1 bis 1
        risingtet(i) = tet(c+ate(rising(i)));
    end
    %Skallierung der Kennlinie auf 0 bis 1
    risingtet = risingtet-min(risingtet);
    risingtet = risingtet/max(abs(risingtet));
    fallingtet = fliplr(risingtet); %Fallende Kennlinie
```

```
    fallingtet(1) = []; %Lösche Spitzenwert (schon in risingtet vorhanden)
    tetdrei = [risingtet fallingtet]; %Zusammenfügen steigend/fallend
    vibstr = vibmaxstr.*tetdrei; %Skallieren mit max. gewünschter Amplitude
end
if dovib == 0
    vib = 0;
else
    ipolk = 50*t(end);   %Interpolationskonstante
%Filterkoeffizient für TP mit 3 Hz bei uninterpoliertem Signal
    [b1,a1] = butter(1,3/(44100/ipolk));
%Filterkoeffizient für TP mit 3 Hz bei interpoliertem Signal
    [b2,a2] = butter(1,3/44100);
%um 0 normalverteiltes Rauschen, uninterp. für Zufallsgehalt der Amplitude
    vibrand1 = randn(1,(length(t)-1)/ipolk);
    vibrand1 = filter(b1,a1,vibrand1);    %Tiefpassfilterung des Rauschen
    vibrand1 = interp(vibrand1,ipolk);    %Interpol. des Rauschens
    vibrand1(length(t)) = 1;              %Setzen des letzten Samples
    vibrand1 = filter(b2,a2,vibrand1);    %Tiefpassfilterung des Rauschen
%Mittelwert wird durch exp zu 1, "Zufallsbreite"  durch vibrandval
    vibrand1 = exp(vibrandval*vibrand1);
%Gleiches Verfahren wie bei vibrand1 für den Zufallsgehalt der Vibratofrequenz
    vibrand2 = randn(1,(length(t)-1)/ipolk);
    vibrand2 = filter(b1,a1,vibrand2);
    vibrand2 = interp(vibrand2,ipolk);
    vibrand2(length(t)) = 1;
    vibrand2 = filter(b2,a2,vibrand2);
    vibrand2 = exp(vibrandval*vibrand2);
%Zusammenfügen des Modualtionssinus
    vib = (vibstr.*vibrand1).*sin(2.*pi.*vibrand2.*vibf.*t);
end
end
```

4 Andere Waveshaper

4.1 Normale_Waveshaper.m

```
%Erzeugt eine Audiodatei, bei der Waveshaping mit Hard Clipping, Tangens
%Hyperbolicus, Sinus oder Chebychev Polynomen 1. Ordnung benutzt werden
%können. Mögliche Eingangssignale sind ein Sinus oder eine Audiodatei.
 %Eingabemöglichkeiten:
fs = 44100; % Samplefrequenz in Hz
t = 0:1/fs:5; % Zeitvektor in Sekunden
g = 1; %Amplitude des Eingangssignals, bei Sinus & Cheby höchstens 1
nameoutput = 'Shaped.wav'; %Name der zu speichernden Audiodatei
method = 1; %1 = Clipping, 2 = tanh, 3 = sin, 4 = Chebychev 1. Ordnung
input = 1; %1 für Sinus, sonst Audiodatei als Eingangssignal
%Eingangssignal:
f = 200; % Grundfrequenz in Hz bei einem Sinus als Eingangssignal
nameinput = 'Audio.wav'; %Name der einzulesenden Datei im gleichen Ordner;
c = 0.5; %Wert für die Grenze bei Hard Clipping
polystr = [0,0,0,1,0,0,0,0,0,0]; %Stärke der Chebychev Polynome angeben
%          1,2,3,4,5,6,7,8,9,10  (1. Element ist Faktor von T1, 2. T2 usw.)
if input == 1
   x = sin(2.*pi.*f.*t);
else
   [x,FS] = audioread(nameinput);
end
x = g*x/max(abs(x)); %Skaliere Audiodatei, eventuell Headroom einfügen
l = length(x);
if method == 1 %Clipping
   y = x;
   y(y>c) = c;
   y(y<-c) = -c;
end
if method == 2 %Tanh
   y = tanh(x);
end
```

```
if method == 3 %Sin
    y = sin(pi*x/2);
end
%Chebychev Polynome 1. Ordnung, T[n] = 2*x*T[n-1]-T[n-2]
if method == 4
    nminus1 = x;                %Tn-1
    nminus2 = ones(1,l);    %Tn-2
    n = ones(1,l);          %Tn, nötig um Tn-1 zwischenzuspeichern
    y = polystr(1)*x;           %Verändere y durch T1
    for i = 2:length(polystr)
        n = 2*x.*nminus1-nminus2;   %Tn
        y = y + polystr(i)*n;       %Verändere y dur aktuelles Tn
        nminus2 = nminus1;          %Tn-1 wird zu Tn-2
        nminus1 = n;                %Tn wird zu Tn-1
    end
end
y = y/max(abs(y)*1.1);  %Skaliere Ergebnis mit extra Headroom
audiowrite(nameoutput,y,fs); %Ausschreiben der Audiodatei
```

4.2 DistortionPedal.m

```
%Vereinfachte Matlab-Adaption eines Distortion Pedals durch Analyse eines
%Boss DS-1 Distortion Pedal. Quelle der Analyse:
%https://ccrma.stanford.edu/~dtyeh/papers/yeh07_dafx_distortion.pdf
%Quelle am 29.11.2016 um 14:10 noch aktuell
 %Grundlegendes
fs = 44100;     %Samplefrequenz in Hz
t = 0:1/fs:10;  %Zeitvektor in Sekunden
sinus = 1;       %1 für Sinus, sonst Einlesen Audiodatei
g = 1;          %Amplitude des Eingangssignals
D = 1;          %zwischen 0-1, steuert Größe der Verstärkung vor dem
                %Waveshaping und dadurch die Stärke des Distortion-Effekts
nameoutput = 'Distorted.wav'; %Name der zu speichernden Datei
Analyse = 0; %setze Analyse = 1 für die Ausgabe der Zwischenergebnisse
```

```
            %y1 bis y5 als Audiodatei Distortedy1 bis Distortedy5
 %Eingangssignal
f = 100;        %Frequenz des Sinus als Eingangssignal in Hz
nameinput = 'Sound.wav'; %Name der Audiodatei als Eingangssignal
 if sinus == 1
    x = sin(2.*pi.*f.*t);
else
    [x,FS]=audioread(nameinput); %Einlesen der Audiodatei
end
x = g*x/max(abs(x));  %Skaliere Audiodatei, eventuell Headroom einfügen
%1. Stufe: Hp mittlerer Ordnung bei 3 Hz durch emitter follower buffer
%am Eingang
Wn1 = 3*2/fs;                  %Normalized cutoff frequency
[b1,a1] = butter(3,Wn1,'high');  %Butterworth filter
y1 = filter(b1,a1,x);
%2. Stufe: single bipolar transistor transimpedance gain stage
%Verstärkung von bis zu 36dB durch Übertragungsfunktion mit doppelter
%Nullstelle bei 0 Hz, Polstellen bei 3Hz und 600 Hz
num1 = [1,0,0];
den1 = [1,3788.76,71061.1517];
[num1d,den1d] = bilinear(num1,den1,fs);
y2 = filter(num1d,den1d,y1);
 %3.Stufe: OP amp gain stage
R1 = 10000;
R2 = (1-D)*100000+4700;
C1 = 0.000001;
C2 = 0.000000000250;
num2=[1,1/(R2*C2)+2/(R1*C2)*2/(R2*C1),1/(R2*R1*C2*C2)];
den2=[1,2/(R1*C2)+2/(R2*C1),1/(R1*R2*C1*C2)];
[num2d,den2d] = bilinear(num2,den2,fs);
y3 = filter(num2d,den2d,y2);
 %4.Stufe Diode clipper, Waveshaping durch Nichtlinearität der Dioden.
y4 = tanh(y3); %oder y4=y3./(1+abs(y3).^2.5).^(1/2.5);
 %5.Stufe Tone Stage, Fade zwischen Hp von 320Hz und einem TP von 1,16kHz
```

```
Wn2 = 1*2/fs;                %Normalized cutoff frequency
[b2,a2] = butter(1,Wn2,'high'); %Butterworth filter
y5 = filter(b2,a2,y4);
Wn3 = 1160*2/fs;             %Normalized cutoff frequency
[b3,a3] = butter(1,Wn3,'low');  %Butterworth filter
y5 = filter(b3,a3,y5);
 %6. Stufe, Hp mittlerer Ordnung bei 3 Hz durch emitter follower buffer
%am Ausgang
Wn4 = 3*2/fs;                %Normalized cutoff frequency
[b4,a4] = butter(3,Wn4,'high');  %Butterworth filter
y6 = filter(b4,a4,y5);
 %fertiges Ausgangssignal
y6=y6/(max(abs(y6))*1.1);      %Skaliere das Ausgangssignal mit Headroom
audiowrite(nameoutput,y6,fs);
 %Ausgabe der Zwischenergebnisse
if Analyse == 1
    y1=y1/(max(abs(y1))*1.1);
    audiowrite('Distortedy1.wav',y1,fs);
    y2=y2/(max(abs(y2))*1.1);
    audiowrite('Distortedy2.wav',y2,fs);
    y3=y3/(max(abs(y3))*1.1);
    audiowrite('Distortedy3.wav',y3,fs);
    y4=y4/(max(abs(y4))*1.1);
    audiowrite('Distortedy4.wav',y4,fs);
    y5=y5/(max(abs(y5))*1.1);
    audiowrite('Distortedy5.wav',y5,fs);
end
```

5 Teiltöne bei Waveshaping mit der Exponentialfunktion

5.1 TeiltoeneExponentialfunktion.m

```
%Berechnet die Stärke aller interessanten Teiltöne eines durch die
%Exponentialfunktion geshapeten Sinus mit angegebener Amplitude.
```

```
 %Eingangsparameter:
g = 4;  %Eingangsamplitude
b = -80; %Grenze Teiltonstärke in dB
d = 1e-10; %Grenze Genauigkeit
%Ausgangsparameter:
a = zeros(1,1000); %Vektor mit Fourierkoeffizienten des Cosinus
c = zeros(1,1000); %Vektor mit Teiltonstärken für Leistungsdichtespektrum
k = 1; %Counter Harmonische
%Schleife für den Grundton (sonst Problem mit i in der while Schleife)
n = 1;      %Counter Summenbildung
adder = 1;  %Bei j hinzugekommener Anteil
while adder > d %Solange der neue Anteil noch groß genug ist
    adder = (g/2)^(2*n-k)*2/(factorial(n-k)*factorial(n));
    a(k) = a(k) + adder;
    n = n+1;
end
c(k) = 20*log10(a(k)/a(1));
 %Schleife für restliche Harmonische
while c(k) > b %Solange der Teilton noch interressant ist
    k = k+1;
    adder = 1;
    n = k;
  while adder > d
      adder = (g/2)^(2*n-k)*2/(factorial(n-k)*factorial(n));
      a(k) = a(k) + adder;
      n = n+1;
  end
  c(k) = 20*log10(a(k)/a(1));
end
%Falls auf die maximale Amplitude und nicht auf den Grundton normiert
%werden soll, kann man die Auskommentierung der nächsten Zeile löschen
%c = 10*log10(a.^2/max(abs(a))^2);
 %Kürze Vektorlänge auf Menge der tatsächlich enthaltenen Teiltöne
c = c(1:k); %Eigentlich i-1 für alle Werte über b, da der 1. Wert unter
```

```
%b aber schon berechnet ist wird er mit in die Ausgabe übernommen
```

6 Darstellung der Spektren

6.1 DisplaySpectrum.m

```
function [Pxx,F] = DisplaySpectrum(audiofile,nfft,noverlap,deltaF)
%Aufruf: [Pxx,F] = DisplaySpectrum('audio.wav',1024,512,4000);
%       ->für gute Genauigkeit 64*1024,32*512 verwenden
%Berechnet das Leistungsdichtespektrum eines Audio-Files
%Eingabeparameter: audiofile     Name des Audiofiles
%                  nfft         FFT-Länge für das
%                               Leistungsdichtespektrum
%                  noverlap     Zahl der Überlappungspunkte
%                  deltaF       dargestellter Frequenzbereich
%Ausgabeparameter: Pxx          Leistungsdichte
%                  F            Frequenzvektor
 %Laden des Audio-Files
[y,Fs] = audioread(audiofile);
signal = y(:,1); %erster Kanal, falls Stero-Signal (Sicherheitsanweisung)
%Leistungsdichtespektrum mit cpsd
[Pxx,F] = cpsd(signal,signal,hanning(nfft),noverlap,nfft,Fs,'twosided');
 %Darstellung des Spektrums (nur die Hälfte bis fs/2)
N = floor(length(F)/2);
Pxx = Pxx/max(abs(Pxx));
Pxx = 10*log10(Pxx);
 %Erstelle den Plot
plot(F(1:N),Pxx(1:N),'r','LineWidth',2);
xlabel('Frequenz/Hz')
ylabel('Leistungsdichte/ dB')
title('Leistungsdichte (in dB)')
%Zoom auf relevanten Bereich
h=gcf;
set(h.Children,'Xlim',[0 deltaF]); %Zoom Frequenz
```

```
set(h.Children,'Ylim',[-80 0]);    %Zoom Frequenzamplitude
grid
end
```

6.2 CompareSpectrum.m

```
function [] = CompareSpectrum(audiofile1,audiofile2,nfft,noverlap,deltaF)
%Aufruf: CompareSpectrum('audio1.wav','audio2.wav,1024,512,4000);
%       ->für gute Genauigkeit 64*1024,32*512 verwenden
%Berechnet die Differenz der Leistungsdichtespektren zweier Audio-Files
%Eingabeparameter: audiofile1/2  Name der Audiofiles
%                  nfft         FFT-Länge für das Leistungsdichtespektrum
%                  noverlap     Zahl der Überlappungspunkte
%                  deltaF       dargestellter Frequenzbereich
 %Laden der Audio-Files
[y1,Fs1] = audioread(audiofile1);
signal1 = y1(:,1); %erster Kanal, falls Stero-Signal (Sicherheitsanweisung)
[y2,Fs2] = audioread(audiofile2);
signal2 = y2(:,1); %erster Kanal, falls Stero-Signal (Sicherheitsanweisung)
 %Warnung falls Samplefrequenzen nicht übereinstimmen
if Fs1 ~= Fs2
   warning('Samplefrequenzen stimmen nicht überein!')
end
 %Leistungsdichtespektrum mit cpsd
[Pxx1,F1] = cpsd(signal1,signal1,hanning(nfft),noverlap,nfft,Fs1,'twosided');
[Pxx2,~] = cpsd(signal2,signal2,hanning(nfft),noverlap,nfft,Fs1,'twosided');
 %Normieren auf die stärkste Frequenz der beiden Signale
if max(abs(Pxx1)) > max(abs(Pxx2))
   Pmax = max(abs(Pxx1));
else
   Pmax = max(abs(Pxx2));
end
Pxx1dB = 10*log10(Pxx1/Pmax);
Pxx2dB = 10*log10(Pxx2/Pmax);
```

```
 %Zur besseren Darstellung zwischen relevanten Frequenzen
Pxx1dB(Pxx1dB<-80) = -80;
Pxx2dB(Pxx2dB<-80) = -80;
 %Darstellung des Spektrums (nur die Hälfte bis fs/2)
N = floor(length(F1)/2);
 %Erzeugen des Differenzspektrums
PxxdB = Pxx1dB - Pxx2dB;
 %Erstelle den Plot
plot(F1(1:N),PxxdB(1:N),'r','LineWidth',2);
xlabel('Frequenz/Hz')
ylabel('Leistungsdichte/dB')
title('Differenz der Leistungsdichtespektren (in dB)')
 %Zoom auf relevanten Bereich
h=gcf;
set(h.Children,'Xlim',[0 deltaF]); %Zoom Frequenz
grid
end
```

6.3 calcspectrogram.m

```
function [time,frq,tfspectrum] = calcspectrogram(audiofile,nfft,deltaF,sow)
%Aufruf:[time,frq,tfspectrum] = calcspectrogram('audio.wav',4*1024,5000,1);
%Die vorliegende Funktion liest ein Audiofile ein und berechnet eine
%Zeit-Frequenz-Analyse mit Hilfe der eingebauten Funktion spectrogram.
%Eingabeparameter: audiofile     Audiodatei in gängigem Format
%                  nfft          Zahl der fft-Punkte pro Block
%                  deltaF        Endfrequenz für die Darstellung
%                  sow           1 für Spektrogramm, sonst Wasserfall
%Ausgabeparameter: tfspectrum    Zeit-Frequenz-Spektrum
%                  time          Zeitachse
%                  freq          Frequenzachse
% Einlesen des Audiofiles und der Abtastrate
[x,Fs] = audioread(audiofile);
 % Prüfen, ob Stereo-Signal
```

```
[~,m] = size(x);
if m > 1
    warning('Stereo-Signal! Verarbeite nur ersten Kanal!');
    x = x(:,1);
end
% Spektrogramm berechnen
[tfspectrum,frq,time] = spectrogram(x,rectwin(nfft),nfft/2,nfft,Fs);
tfspectrum = abs(tfspectrum/max(max((tfspectrum))));
% Spektrogramm darstellen
surf(time,frq,10*log10(tfspectrum),'LineStyle','none','FaceColor','interp')
colormap(hsv)
xlabel('Zeit/s')
ylabel('Frequenz/Hz')
zlabel('Amplitude/dB')
axis([0,time(end),0,deltaF,-80,0])
view([-104,36]) %Wasserfalldarstellung
if sow == 1
    view(0,90)  %Ändern in Spektrogramm
end
end
end
```