

Heinrich-Heine-Universität Düsseldorf

Institut für Sprache und Information

Pure Data als Werkzeug phonetischer Analyse

Stimmqualität

Wintersemester 2011/12

Dr. Mats Exter

Heinrich-Heine-Universität Düsseldorf

von

Victor Persien

10. April 2012

Inhaltsverzeichnis

0	Einleitung	1
1	Das Zusammenspiel von Artikulation und Akustik	3
1.1	Modalstimme	3
1.2	Hauchstimme (Breathy Voice)	5
1.3	Knarrstimme (Creaky Voice)	6
1.4	Zusammenfassung	7
2	Messung akustischer Parameter	8
2.1	Die Autokorrelationsmethode	8
2.2	Harmonics-to-Noise-Verhältnis	10
2.3	Jitter	10
2.4	Shimmer	11
2.5	Cepstral Peak Prominence	11
3	Überblick über Pure Data	13
3.1	Kurzeinführung	14
3.2	Kritik	17
4	Pd als Werkzeug phonetischer Analyse	18
4.1	Vorüberlegungen	18
4.2	Zwei Beispielpatches	19
4.2.1	Patch #1 – HNR und Jitter	21
4.2.2	Patch #2 – Cepstral Peak Prominence	24
4.3	Evaluation	27
5	Fazit und Ausblick	28
	Literatur	28

PHONETISCHE ANALYSE MIT PURE DATA

0 Einleitung

Die Phonetik hat das Glück, zu denjenigen linguistischen Disziplinen zu gehören, welche konkrete quantitative Messungen vornehmen können. Unter den „klassischen“ sechs Teildisziplinen – Phonetik, Phonologie, Semantik, Pragmatik, Morphologie und Syntax – ist dies sogar ein Alleinstellungsmerkmal. Doch Messungen, welche die Funktionsweise der Sprechorgane beim Artikulationsvorgang offenlegen sollen, sind häufig nicht unproblematisch, besteht doch stets die Gefahr, durch ein Eindringen mit Messinstrumenten in den eigentlich weitestgehend verschlossenen Mund-Nasen-Rachenraum gerade diejenigen Eigenschaften der Artikulationsorgane, welche es zu untersuchen gilt, zu verfälschen. Ferner stellen solche Maßnahmen immer auch einen Eingriff in die Intimsphäre der Informanten dar. Daher bietet es sich an, auf nicht-penetrative Alternativen zurückzugreifen. Hier gibt es zum einen bildgebende Verfahren, wie etwa Ultraschall oder MRT. Diese sind jedoch häufig kostspielig, aufwendig oder bieten, im Falle von Ultraschall, eine schlechte räumliche Auflösung. Andererseits gibt es die Möglichkeit Audioaufnahmen von Sprachsignalen akustisch zu untersuchen und auf diesem Weg Rückschlüsse auf artikulatorische Merkmale des Gesprochenen zu ziehen. Aufgrund ihrer Unkompliziertheit und Kostengünstigkeit ist diese Methode wohl die am häufigsten verwendete Form phonetischer Analyse.

Die vorliegende Arbeit soll dazu dienen, die Liste an Werkzeugen zur akustisch-phonetischen Analyse um einen möglichen Kandidaten zu erweitern; nämlich um die grafische Multimedia-Programmiersprache *Pure Data* (Pd). Dazu soll in Kapitel 1 zunächst beispielhaft auf einige für die Sprachproduktion bedeutsame Funktionsweisen des Larynx und ihr akustisches Korrelat eingegangen werden. Anknüpfend daran werden in Kapitel 2 die physikalisch-technischen Grundlagen akustischer Analyse kurz erläutert. Kapitel 3 soll dazu dienen, ein grundlegendes Verständnis der Arbeitsweise von Pd zu schaffen, woraufhin in 4 zwei Implementierungen zur Echtzeitmessung der phonetisch interessanten Parameter *Jitter* und *Harmonics-to-Noise Ratio* (HNR) bzw. *Cepstral Peak Prominence* vorgestellt werden.

Das Fundament der Arbeit bilden ganz klar die Zusammenhänge zwischen Physiologie und Akustik und nicht so sehr die Grundlagen digitaler Signalverarbeitung. Ein wenigstens oberflächliches Verständnis von Konzepten wie *Sampling* oder *Diskrete Fouriertransformation* ist Teilvoraussetzung um diesen Text verstehen zu können. Begriffe, die im Zusammenhang mit dem genannten „Fundament“ stehen, werden an geeigneter Stelle erklärt. Dieses ermöglicht es, die Implementationen in Kapitel 4 zu verstehen. Am Schluss wird sich zeigen, dass es sich bei Pd um ein potentiell mächtiges Hilfsmittel für Phonetiker handelt, damit es sich etablieren kann, aber noch einige Hürden in Form von standardisierten Bibliotheken und Fehlerbehebungen genommen werden müssen.

1 Das Zusammenspiel von Artikulation und Akustik

Nutzen des aktuellen Kapitels ist es, einen Raum artikulatorischer Phänomene bereitzustellen, um diesen später über ihren akustischen Output mittels Pd zu ergründen. Hierbei soll nur eine kleine Teilmenge der für die Sprachproduktion verantwortlichen physiologischen Funktionen herangezogen werden, nämlich einige (supra-)glottale Funktionen, genauer: *Modalstimme*, *Hauchstimme* (*Breathy Voice*) und *Knarrstimme* (*Creaky Voice*).

Diese drei Stimmqualitäten bilden aufgrund der durch ihre Stimmbeteiligung verursachte Periodizität vergleichsweise einfache Fallbeispiele; und damit eine angemessene Grundlage für eine erstmalige Analyse mit Pd. Im Folgenden soll dazu zunächst die Modalstimme hinsichtlich ihrer physiologischen und akustischen Eigenschaften untersucht und als Default etabliert werden. So können Hauch- und Knarrstimme relativ zu ihr betrachtet werden. Jeder Abschnitt beinhaltet die Betrachtung der glottalen und supraglottalen Aktivitäten mit anschließender Erläuterung, wie sie sich im resultierenden akustischen Signal widerspiegeln. Eine detailliertere Beschreibung der dafür herangezogenen akustischen Parameter findet sich im nächste Kapitel. Abschließend folgt eine zusammenfassende Gegenüberstellung der Ergebnisse, auf die in den folgenden Kapiteln erneut zurückgegriffen wird.

1.1 Modalstimme

Die Modalstimme ist typologisch gesehen die am weitesten verbreitete Phona-tionsart mit Stimmbeteiligung, so gibt es weltweit keine Sprache ohne Vokale in modaler Aussprache (Gordon & Ladefoged 2001). Gründe dafür mögen ökonomischer oder perzeptiver Natur sein: Solche Laute sind vergleichsweise anstren-gungslos zu produzieren und weisen gleichzeitig eine hohe Schallfülle auf.

Die Produktion von Vokalen in Modalstimme stellt wohl den kanonischen Fall dar, wenn es um die Beschreibung der Funktionsweise der Glottis bei der Sprachproduktion geht. So stellt sie vielfach in der Literatur die neutrale Stimm-qualität dar, von der andere Stimmqualitäten nur abweichen. Nach Laver (1980: 14) äußert sich diese Neutralität dadurch, dass der Larynx weder angehoben

noch abgesenkt, die Zungenwurzel weder vor- noch zurückgezogen ist und der Gaumen-Rachen-Bogen („faucal pillars“) nicht den Vokaltrakt verengt. Die Stimmlippen sind in dieser Konfiguration adduziert, lassen jedoch genügend Freiraum, damit der aus der Lunge austretende Luftstrom durch die Glottis entweichen kann. Längs- und adduktive Spannung, sowie mediale Kompression sind als moderat beschrieben, jedoch genügend stark, um einen Schwingungsvorgang der Stimmlippen verantworten zu können (Laver 1980, Esling & Harris 2005).

Der Schwingungsvorgang selbst folgt dem *Bernoullieffekt*: Sobald der subglottale Druck größer ist als die mediale Kompression der Stimmlippen, werden diese auseinandergesprengt. Dadurch entsteht Unterdruck an den Rändern der Glottis, wodurch die Stimmlippen wieder zusammengezogen werden. Durch die stetig nachströmende Luft aus der Lunge und die gehaltene Spannung wiederholt sich der Zyklus, bis mindestens einer der beiden Faktoren entfällt (Laver 1980, Hirose 1997, Esling & Harris 2005).

Das von Laver (1980) als *moderat* bezeichnete Spannungs- und Druckverhältnis der Stimmlippen macht es möglich, dass diese bei konstantem pulmonalegressivem Druck weitestgehend periodisch schwingen können. Das von ihnen erzeugte periodische Signal weist also keine signifikanten Schwankungen in Amplitude (*Shimmer*) und Dauer (*Jitter*) aufeinanderfolgender Perioden auf. Die Werte für einen gesunden Menschen liegen hier bei unter 3,8% für Shimmer bzw. 1% für Jitter (Boersma & Weenink 2012).

Da es im supraglottalen Bereich zu keiner nennenswerten Bildung von Verengungen kommt, verlässt das Glottissignal weitgehend rauschfrei den Rachenraum, was sich in einem hohen *Harmonics-to-Noise-Verhältnis* (HNR) äußert.

Die Wellenform des Glottissignals erscheint leicht nach rechts geneigt, was darauf zurückzuführen ist, dass durch den hohen Luftdruck die Ausbreitung des Schalls hinter der räumlichen Ausbreitung zurückbleibt (Stevens 1997). Dies wird durch einen geringen *Open Quotient* (OQ) messbar, der das Verhältnis zwischen dem gesamten Schwingungszyklus und der Zeit beschreibt, in der die Glottis geöffnet ist. Ferner korreliert der *Spectral Tilt*, das Verhältnis der Grundfrequenz zu ihren Harmonischen, mit der Glottisöffnung. Ein positiver Tilt, wie er bei der Modalstimme vorliegt, bedeutet einerseits einen geringen OQ und

andererseits, damit zusammenhängend, eine abrupte Schließung der Glottis. Dennoch ist der Spectral Tilt der Modalstimme noch geringer als derjenige der Knarrstimme (Gordon & Ladefoged 2001: 397ff.).

1.2 Hauchstimme (Breathy Voice)

Hauchstimme oder englisch *Breathy Voice*¹ ist eine von deutlichem Hauchen begleitete Form der Stimmbildung. Auf den unbedarften Hörer wirkt sie wie ein Seufzen. Von einem reinen Hauchen (*Breath*) unterscheidet sie sich durch die zusätzlich vorhandene Stimmlippenschwingung.

Die Stimmlippen sind adduziert, jedoch weniger stark als bei modaler Phonation. Ebenso gering sind Längsspannung und mediale Kompression ausgeprägt. Dadurch kommen die beiden Stimmlippen nie oder kaum in Berührung miteinander (Laver 1980). Die Folge ist ein stetig der Lunge entweichender Luftstrom, welcher nicht zur Stimmlippenschwingung beiträgt.

Die entweichende Luft äußert sich in einer Rauschüberlagerung des an sich periodischen Signals der Schwingung. Aufgrund der relativ kleinen Lücke, durch welche die Luft strömt, sind hier vor allem die oberen Frequenzen ab circa 2 kHz betroffen (Childers & Lee 1991, Hillenbrand et al. 1994, Gordon & Ladefoged 2001). Das Rauschen ist visuell im Spektrogramm sehr gut dadurch zu erkennen, dass die oberen Formanten weniger stark ausgeprägt erscheinen bzw. tatsächlich sind. Das hochfrequente Rauschen resultiert in einem niedrigen HNR.

Das freie periodische Flattern der Stimmlippen ohne gegenseitige Berührung führt auch mit sich, dass sie nicht aneinander gesogen werden, was die Verschlüsse der Glottis weniger abrupt macht. Messbar wird dies durch einen höheren OQ im Vergleich zur Modalstimme (Gordon & Ladefoged 2001), ausgelöst durch eine f_0 mit besonders hoher Amplitude im Gegensatz zu den anderen Harmonischen (Hillenbrand et al. 1994). Insgesamt wird der Spectral Tilt bei Gordon & Ladefoged (2001: 397f.) als „most steeply negative“, also steil abfallend beschrieben.

Cross-linguistisch gesehen ist Hauchstimme überdurchschnittlich häufig mit

¹Ich werde im Folgenden die beiden Begriffe in austauschbarer Weise gebrauchen. Das gleiche gilt für das Paar Knarrstimme – Creaky Voice.

einer tiefen Grundfrequenz assoziiert (Gordon & Ladefoged 2001). Da es ohne Weiteres möglich ist, auch Laute mit hoher f_0 in Breathy Voice zu produzieren, kann dieses Detail lediglich als probabilistischer akustischer Cue bei der Einordnung dieser Stimmqualität dienen.

1.3 Knarrstimme (Creaky Voice)

Knarrstimme oder englisch Creaky Voice lässt sich auditiv als eine knarrende, knatternde Stimme beschreiben. Durch die häufig mit ihr verbundene sehr tiefe Stimmlage sind die einzelnen „Anschläge“ sogar leicht auseinanderzuhalten.

Die Artikulation der Knarrstimme geht mit einer starken Verengung der glottalen und supraglottalen Strukturen einher. Hiervon ist nicht nur, wie bei den beiden obigen Stimmqualitäten, die Glottis betroffen, sondern der gesamte Bereich unmittelbar oberhalb dieser. Die Verengung der Glottis wird durch eine Bewegung der Stellknorpel in Richtung des Thyroiden verursacht. Dadurch verkürzen sich die Stimmlippen und verlieren an Spannung (Esling & Harris 2005: 369f.). Weitere Konstriktion entsteht durch Sphinkterbildung mittels der aryepiglottischen Falten, sodass die Luft nur noch durch einen sehr kleinen Spalt entweichen kann (Esling & Harris 2005).

Die hörbaren Pulse der Creaky Voice entstehen durch ein abruptes Öffnen und schließen der Glottis umgeben von langer Stille. Tatsächlich befindet sich die Glottis 90% (Esling & Harris 2005: 370) der Dauer einer Periode in einem geschlossenen Zustand, resultierend in einem sehr geringen OQ (Blomgren et al. 1998). Damit zusammen hängt auch ein Spectral Tilt, der noch weniger ausgeprägt ist als jener der Modalstimme (Gordon & Ladefoged 2001).

Die relative Länge aufeinander folgender Perioden variiert beträchtlich, so messen Blomgren et al. (1998) einen durchschnittlichen Jitter von 14,9% für Männer bzw. 8.8% für Frauen. Als Grund dafür nennen Monsen & Engebretson (1977) die entspannten Stimmlippen in Verbindung mit niedrigem subglottalen Druck. Hollien et al. (1966) verweisen auf den Umstand, dass durch die Sphinkterbildung die aryepiglottischen Falten mit den Stimmlippen in Berührung geraten. Dies erhöht den Widerstand der in Schwingung zu versetzenden Einheit und kann daher weiter als für Unterschiede in Periodenlänge (und -amplitude)

verantwortlich herangezogen werden.

In der Literatur wird Knarrstimme oftmals mit einer im Vergleich zur Modalstimme sehr niedrigen Grundfrequenz assoziiert (vgl. bspw. Laver 1980, Blomgren et al. 1998, Esling & Harris 2005). Gordon & Ladefoged (2001) stellen jedoch heraus, dass diese Beobachtung nicht universell ist, sondern in den Sprachen der Welt durchaus auch mit hohem Ton assoziiert sein kann. Eine Verbindung von f_0 und Stimmqualität sei daher konsistenter für Hauchstimme (S. 400).


1.4 Zusammenfassung

Wir haben gesehen, wie die Stimmqualitäten Modalstimme, Hauchstimme und Knarrstimme produziert werden und welche akustischen Korrelate damit im Zusammenhang stehen, die sie letzten Endes für einen Hörer unterscheidbar machen. In Tabelle 1 findet sich eine Gegenüberstellung der wichtigsten akustischen Parameter, die mit den hier behandelten Stimmqualitäten assoziiert sind und ihre jeweilige tendenzielle Ausprägung. Nicht auf alle wurde in diesem Kapitel detailliert eingegangen. Dies soll daher im nachfolgenden Kapitel 2 in ausreichender Form geschehen.

	Jitter	Shimmer	HNR	f_0	OQ	Tilt	CPP
Modal	niedrig	niedrig	hoch	mittel	mittel	↗	hoch
Breathy	mittel	hoch	mittel	niedrig	hoch	↘	niedrig
Creaky	hoch	hoch	niedrig	niedrig	niedrig	↗	?

Tabelle 1: Assoziation der Stimmqualitäten Modalstimme, Breathy Voice und Creaky Voice mit unterschiedlichen akustischen Parametern.

2 Messung akustischer Parameter

Dieses Kapitel soll dazu dienen, akustische Parameter, die zur phonetischen Analyse herangezogen werden können, zu definieren und den schon im vorangegangenen Kapitel beschriebenen Ausdrücken eine berechenbare Gestalt zu geben. Davon abgesehen ist das Verständnis dieser Parameter Voraussetzung dafür, die Implementationen in Kapitel 4 nachvollziehen zu können. Diejenigen Parameter, die auch dort eine Rolle spielen werden, sind mit einem  kenntlich gemacht.

2.1 Bestimmung der Periodenlänge mittels Autokorrelation

Autokorrelation (AC) ist ein spezieller Fall der *Kreuzkorrelation* (CC). Bei Letzterer ist das Ziel, zwei Funktionen, oder im diskreten Fall Listen, auf Ähnlichkeit zu untersuchen. Die CC-Funktion erreicht dort ihr Maximum, wo die zweite Funktion der ersten am ähnlichsten ist. AC ist nun die CC einer Funktion mit sich selbst und wird dazu verwendet, Perioden in einem gegebenen Signal zu finden. Mit ihrer Hilfe kann die Grundfrequenz eines periodischen Signals ermittelt werden, sowie das Harmonic-to-Noise-Verhältnis.

Die gängige Autokorrelationsfunktion (vgl. bspw. Boersma 1993) vergleicht eine Funktion $x(t)$ punktweise mit Abstand τ mit sich selbst. Sie lautet daher

$$r_x(\tau) := \int x(t)x(t + \tau)dt. \quad (1)$$

Es handelt sich also um eine Funktion in der Abstandsdomäne. Diese erfüllt die nötigen Eigenschaften, die eine gute Autokorrelationsfunktion erfüllen soll: Sie hat ein globales Maximum an der Stelle 0 und lokale Maxima an den Stellen $\tau \geq 0$ an denen $x(t)$ maximal ähnlich zu sich selbst ist. Das größte Maximum $r_x(\tau)$ mit gleichzeitig kleinstem $\tau \neq 0$ und weiteren lokalen Maxima an den Stellen $n\tau$ ($n \in \mathbb{N}$) bezeichnet den Abstand zur ersten Periode von $x(t)$. Sind alle diese lokalen Maxima zugleich auch globale Maxima, so bezeichnet man die Funktion als *periodisch*, sind sie nur lokale Maxima, kann man ihren *periodischen Anteil* relativ zu $r_x(0)$ mittels der *normalisierten Autokorrelation* als Wert zwischen

0 und 1 darstellen:

$$r'_x(\tau) := \frac{r_x(\tau)}{r_x(0)}. \quad (2)$$

Da ein in digitaler Form vorliegendes akustisches Signal jedoch nicht kontinuierlich ist, sondern in diskreten Samplen vorliegt, muss es eine an den diskreten Fall angepasste Variante der AC-Funktion geben. Hier bedient man sich der geometrischen Auffassung des Integrals als Funktion, die einen Flächeninhalt zurückgibt und definiert die diskrete Autokorrelation (DAC) als

$$r_x(\tau) := \sum_{i=0}^{T-\tau} x(i)x(i+\tau), \quad (3)$$

wobei T für die Länge des Fensters steht, über das integriert wird. Die normalisierte Variante lautet entsprechend

$$r'_x(\tau) := \frac{\sum_{i=0}^{T-\tau} x(i)x(i+\tau)}{\sum_{i=0}^{T-\tau} x(i)^2}. \quad (4)$$

Ein naiver Algorithmus, der DAC implementiert, hätte eine Laufzeit von N^2 bei N Eingaben. Daher bedient man sich hier in der Regel des Wiener-Chintschin-Theorems, welches darauf basiert, Operationen der Zeitdomäne in der Frequenzdomäne durchführen zu können. Sei $FT(x(t))$ die Fourier-Transformierte von $x(t)$ und $IFT(x(t))$ die Inverse Fourier-Transformierte, dann gilt

$$r'(x) = IFT(|FT(x(t))|^2). \quad (5)$$

Algorithmen, die auf diesem Theorem basieren erreichen nur eine Laufzeitkomplexität von $N \log N$ und sind daher schneller als der Weg über das Integral (vgl. bspw. Rabiner & Schafer 1978: 163f.).

Das später von mir verwendete Modul `helmholtz~` von Vetter (2012) benutzt eine andere normalisierte DAC-Funktion als jene in 4, nämlich die *Special Normalisation of the Autocorrelation Function* (SNAC). Diese geht auf McLeod (2008)

zurück und ist folgendermaßen definiert:

$$r'_x(\tau) := \frac{2 \sum_{i=0}^{T-\tau} x(t)x(t+\tau)}{\sum_{i=0}^{T-\tau} (x(t)^2 + x(t+\tau)^2)}. \quad (6)$$

Vorteile dieser Funktion sind, dass sie zur Ermittlung der Periode nur ein 1, 2-mal so großes Fenster wie diese benötigt; im Gegensatz zu DAC, bei der das Fenster mindestens zweimal so groß sein muss. Weiter soll SNAC akkurater und schneller sein als DAC, wie McLeods Benchmarks ergeben haben.

2.2 Harmonics-to-Noise-Verhältnis

Der Begriff des *Harmonics-to-Noise-Verhältnisses* (HNR) hängt eng mit den Eigenschaften der normalisierten Autokorrelation zusammen. Ziel ist es, ein Maß für das Verhältnis von periodischem Anteil und Rauschen in einem Signal zu finden. Im letzten Abschnitt wurde bereits erwähnt, dass der Funktionswert von $r'_x(\tau) \in [0, 1]$ an der (Quasi-)Periode τ_{max} als Maß für die Periodizität des Signals gereichen kann. Setzt man ihn daher ins Verhältnis zum Abstand zu $r'_x(0) = 1$, lässt sich das HNR folgendermaßen mit einem Rückgabewert in dB definieren:

$$HNR = 10 \log_{10} \frac{r'_x(\tau_{max})}{1 - r'_x(\tau_{max})}. \quad (7)$$

2.3 Jitter

Mit *Jitter* wird die durchschnittliche Varianz in der Periodenlänge eines Signals bezeichnet. Sie ist schlicht der mittlere Abstand aller Periodenpaare eines Signals dividiert durch die mittlere Periodenlänge. Sei x_t ein (pseudoperiodisches) Signal und seien T_1, \dots, T_n die Längen seiner Perioden.

$$Jit(x_t) = \frac{\frac{1}{N-1} \sum_{i=2}^N T_i + T_{i-1}}{\frac{1}{N} \sum_{i=1}^N T_i} \quad (8)$$

Vgl. hierzu auch Boersma & Weenink (2012).

2.4 Shimmer

Mit *Shimmer* wird die durchschnittliche Varianz in der maximalen Amplitude der Perioden eines Signals bezeichnet. Die Definition erfolgt analog zu der von Jitter, mit A_1, \dots, A_n als maximale Amplituden der Perioden.

$$\text{Shim}(x_t) = \frac{\frac{1}{N-1} \sum_{i=2}^N A_i + A_{i-1}}{\frac{1}{N} \sum_{i=1}^N A_i} \quad (9)$$

2.5 Cepstral Peak Prominence

Das *Cepstrum* als Mittel in der Signalverarbeitung geht zurück auf Bogert et al. (1963). Es basiert auf der Feststellung, dass es sich beim Frequenzspektrum eines gegebenen Signals auch um eine Wellenform handelt, von der sich wiederum ein Spektrum ableiten lässt. Für ein Signal $x(t)$ lässt sich daher das Cepstrum definieren (vgl. bspw. Rabiner & Schafer 1978: 363f.) als

$$\text{IFT}(\log[|FT(x(t))|^2]). \quad (10)$$

Das Cepstrum ist eine Funktion der *Quefrenz*², einer Größe in der Zeitdomäne. Die von ihr Abhängige Variable ist die Magnitude in dB. Dennoch stellt es nicht wieder das Ursprungssignal dar, wie man meinen könnte. Vielmehr lässt sich an ihm die Verteilung der Energie über die Zeit ablesen. So hat bspw. die Periode der Grundfrequenz eines periodischen Signals mit vielen Harmonischen eine besonders hohe Auslenkung.

Das Maß der *Cepstral Peak Prominence* (CPP) stammt von dem Versuch Hillenbrand et als. (1994) einen geeigneten akustischen Indikator für Breathy Voice zu finden. Der *Cepstral Peak* ist zumeist das Maximum des Cepstrums außerhalb seiner niedrigsten Quefrenzen und sein Abstand zur Nullquefrenz entspricht der Periode der Grundfrequenz des Signals. Da der Funktionswert an dieser Stelle variieren kann, ist die CPP der Abstand dieses Wertes zu seinem durch Lineare Regression vorhergesagten Wert. Die Autoren geben den Quefrenzbereich

²Dieses anagraphische Motiv setzt sich terminologisch für Eigenschaften und Operationen im Zusammenhang mit dem Cepstrum fort: Saphe statt Phase, Lifter statt Filter, Gamnitude statt Magnitude, u.a.

unter 10 ms als sehr verrauscht an und ziehen diesen daher nicht zur Berechnung der Linearen Regression heran. Erwartungsgemäß haben Laute in Hauchstimme eine niedrigere CPP, da sie durch den Rauschanteil weniger Energie in den Harmonischen der Grundfrequenz besitzen.

3 Überblick über Pure Data

Pure Data (Pd) ist eine grafische Programmierumgebung mit gleichnamiger Programmiersprache, welche Mitte der 1990er Jahre von Miller S. Puckette zu entwickeln begonnen wurde (vgl. Puckette 1996). Die Entwicklung war anfangs von dem Gedanken geleitet, die Software Max (seinerzeit Max/FTS) verbessern zu wollen. Heute steht Pd als freie (BSD-Lizenz) Alternative neben seiner kommerziellen Schwester Max (© Cycling '74).

Pd war ursprünglich dafür gedacht, elektronische Musik zu entwickeln und wird auch heute hauptsächlich dafür verwendet. Dennoch reicht das potentielle Anwendungsgebiet heute weiter. Dies ist nicht zuletzt auch auf die Möglichkeit zurückzuführen, eigene Erweiterungen (*Externals*) in der Programmiersprache C zu schreiben. So gibt es mittlerweile ein breites Angebot an frei verfügbaren Externals, u.a. auch für das Plotten von Funktionen und Visualisierung (GEM) oder physikalische Modellierung (pmpd). Dadurch wird Pd von einem Werkzeug für musikalische Zwecke zu einer Programmierumgebung für viele weitere Aspekte der Signalverarbeitung.

Auch wenn Pd vor allem in der Produktion Anwendung findet, so gibt es in dieser immer auch einen Nutzen für analysierende Funktionen, wie dem Verfolgen von Signalen oder FFT. Und in der Analyse wird Gebrauch von manipulativen Mitteln gemacht, etwa dem Filtern. Daher bietet es sich an, auch die Anwendung in der phonetischen Analyse von Sprachsignalen ins Auge zu fassen.

Dieses Kapitel soll daher dazu dienen, die grundlegende Funktionsweise von Pure Data verständlich zu machen. Diese wird hier nur so weit beleuchtete, wie es das Lesen der „Quelltexte“ im nächsten Kapitel erforderlich macht. Für weiterführende Informationen bietet sich beispielsweise das auch online verfügbare Buch von Kreidler (2009) an. Abschließend soll noch auf einige Schwächen des Programms eingegangen werden.

Anmerkung: Zwar ist Puckette hauptverantwortlich für die Programmierung der Kernkomponenten von Pd, meine Anstrengungen basieren jedoch auf der Distribution *Pd-extended* (puredata.info), welche über eine leicht modifizier-

te grafische Oberfläche und von Haus aus m.E. unverzichtbare Externals verfügt.

3.1 Kurzeinführung

Dateien oder vielmehr Programme in Pure Data bezeichnet man als *Patches*. Diese bestehen aus verschiedenen Elementen³, die einander Daten austauschen, und, da es sich um eine grafische Programmiersprache handelt, aus Koordinaten, die angeben, wie die Elemente auf der Arbeitsfläche miteinander in Beziehung stehen. Das grundlegendste dieser Elemente stellt das *Objekt* dar. Auf Abbildung 1 sind die zwei Arten von Objekten zu sehen, die es gibt, nämlich „normale“ und *Tilde*-Objekte. Letztere zeichnen sich dadurch aus, (Audio-)Daten gemäß der aktuell verwendeten Samplerate zu übertragen, sodass sichergestellt ist, dass alle Tilde-Objekte im selben Takt Daten austauschen. Ordinäre Objekte hingegen verarbeiten Daten sofort, d.h. so schnell es der Rechner ermöglicht. Man spricht in diesem Zusammenhang davon, dass Objekte in *Audio*- bzw. *Nachrichtenrate* senden oder empfangen.

Damit überhaupt Daten ausgetauscht werden können, müssen die Beziehungen der einzelnen Elemente zueinander spezifiziert werden. Dies geschieht über Leitungen, die zwischen den Elementen (nicht nur Objekten) gezogen werden. Zu diesem Zweck verfügen Elemente über eine spezifische aber prinzipiell beliebige Anzahl an Ein- und/oder Ausgängen, wobei sich Eingänge stets oben, Ausgänge unten befinden. Es gibt speziell an Objekten zwei Sorten von Eingängen, namentlich *heiße* und *kalte*. Es gibt höchstens einen heißen Eingang, der sich immer ganz links befindet, alle anderen sind kalt. Der Unterschied hat zum Zweck, dass die kalten Eingänge Werte solange speichern bis auch der heiße Eingang einen erhalten hat. Erst dann werden die Operationen des jeweiligen Objekts ausgeführt.

Beim Ziehen von Verbindungen bestimmt die Reihenfolge in der die Verbindungen an die Eingänge eines Objekts angelegt wurden auch die zeitliche Ab-

³Es gibt keinen mir bekannten etablierten Ausdruck für die Gesamtheit aller in Pd vorhanden funktionalen Einheiten – Objekte, Nachrichten-, Nummer-, Symbolboxen, Kommentarfelder und Arrays. Daher werde ich diese i.F. als *Elemente* bezeichnen.

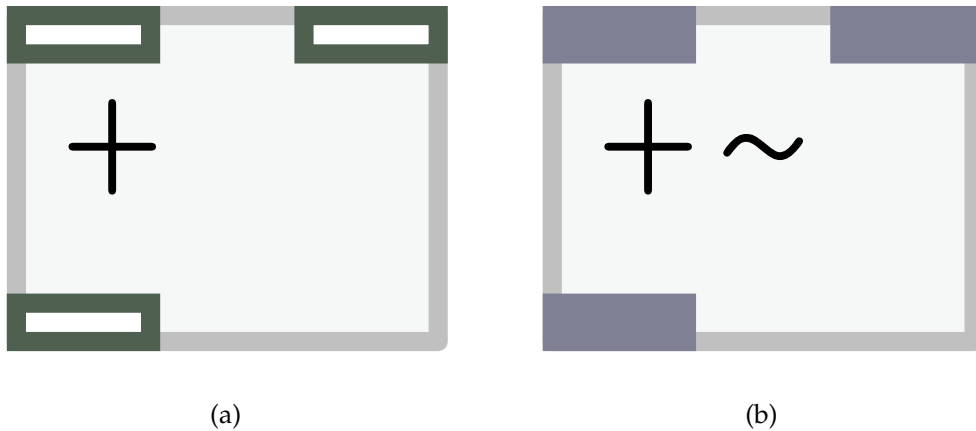


Abbildung 1: Zwei Arten von Objekten in Pure Data. (a) sendet und empfängt in Nachrichtenrate und addiert zwei Zahlen, (b) sendet und empfängt in Audiorate und addiert zwei Signale.

folge der Datenübertragung an das Objekt, wobei zuerst gezogene Leitungen auch zuerst übertragen. Leider ist die Steckreihenfolge in einem vorliegenden Patch nirgends außer im Quelltext ablesbar. Es ist aber in der Regel davon auszugehen, dass diese von rechts nach links erfolgt ist. Andernfalls gibt es Objekte, wie etwa *trigger* (s.u.), die besondere Reihenfolgen zum Einen ermöglichen und zum Anderen erkenntlich machen.

Wie schon erwähnt, gibt es neben Objekten noch weitere Elemente in Pd, deren Funktionen teils selbsterklärend sind: *Nummernboxen* geben Zahlen aus, *Symbolboxen* Symbole und *Kommentarfelder* ermöglichen es Kommentare in einen Patch zu schreiben. Einzig Nachrichtenboxen und Arrays verdienen hier eine gesonderte Betrachtung. Diese geschieht im Folgenden zusammen mit der Erläuterung einiger wichtiger Objekte. Alle weiteren Funktionalitäten ergeben sich aus dem Kontext der Implementationen in Kapitel 4.

bang Das bang-Objekt, grafisch ein Kreis in einem Quadrat, stellt einen einfachen Taster, der einen binären Impuls aussendet, dar. Viele Objekte erzeugen überhaupt erst eine Ausgabe, wenn sie einen solchen Impuls erhalten haben, d.h. *getriggert* worden sind. Es ist auch möglich, Elementen beim Aufrufen ei-

nes Patches einen bang zu übermitteln. Dafür existiert das Objekt `loadbang`.

Das Objekt `metro` erzeugt bangs in einem regelmäßigen Abstand in Millisekunden, der ihm als Argument übergeben wurde.

Nachrichten Nachrichten werden durch Nachrichtenboxen (rechteckig mit einer konkaven Seite) übermittelt, die entweder angeklickt oder getriggert werden. Neben dem Setzen von numerischen Werten werden diese vor allem dafür verwendet, an Objekte solche Parameter zu übergeben, die nicht als Argument spezifiziert werden können oder für die kein Eingang vorgesehen ist. In Abschnitt 4 ist auf Abb. 2 ein Beispiel davon zu sehen. Weiter können an Elemente, die einen Namen besitzen, wie z.B. Arrays, auch ohne feste Verbindung Nachrichten übermittelt werden.

float und Arrays Einzelne numerische Werte lassen sich im `float`-Objekt, meist als `f` abgekürzt, zwischenspeichern. Dieses gibt einen Wert aus, wenn dieser am kalten Eingang anliegt und der heiße Eingang per bang getriggert wurde oder direkt, wenn ein Wert an den heißen Eingang gesendet wird.

Für die Speicherung mehrerer Werte eignen sich, wie in anderen Programmiersprachen auch, Arrays. In Pd sind diese allerdings zusätzlich stets grafisch repräsentiert und eignen sich so als Ausgabe für Wellenformen, Spektren oder Funktionsgraphen. Arrays können auf viele Arten ausgelesen und manipuliert werden. `tabread` nimmt als Argument einen Index und gibt den entsprechenden Wert aus, `tabwrite` schreibt einen Wert an einen Index, während `tabwrite~` ein ganzes Fenster eines Signals an ein Array sendet. Darüber hinaus gibt es noch weitere Objekte, die nicht Teil der Standardbibliothek sind. Hier sei besonders die Bibliothek `iem_tab` erwähnt, deren Externals viele nützliche Funktionen zum Umgang mit Arrays bereitstellen, deren Implementation in Pd ansonsten sehr performanzlastig wäre.

trigger Das Objekt `trigger`, kurz `t`, gibt einen Input beliebig vervielfacht in einer vorbestimmten Reihenfolge aus. So gibt `t f f` zweimal dieselbe Gleitkommazahl von rechts nach links aus. Neben Gleitkommazahlen können auch Lis-

ten, Symbole oder bangs von trigger verteilt werden. Zudem kann jeder Input einen bang triggern.

expr Pd bietet für die gebräuchlichsten arithmetischen Operationen bereits eigene Objekte. Komplexe Rechenoperationen können so aber schnell unübersichtlich und zudem langsam werden. Das Objekt `expr` nimmt als Argument einen arithmetischen Ausdruck (in C-Syntax) und beliebig viele numerische Inputs, die mit `$f1`, `$f2`, ... angesprochen werden können.

3.2 Kritik

An Pd sind viele Vor- und Nachteile freier Software erkennbar. Einerseits sind Software und Quellcode für Jedermann frei zugänglich und mit den nötigen Fachkenntnissen auch individuell anpassbar. Andererseits sind solche Projekte, zumindest dann, wenn kein größerer Konzern im Hintergrund mitarbeitet, sehr stark von der Aktivität der Community abhängig, die sich darum bildet. Pd hat zwar eine rege Community, die Kernentwicklung ist jedoch hauptsächlich von Puckette abhängig, wodurch Bugs, die im Zusammenhang mit den Kernkomponenten stehen, nicht allzu schnell behoben werden können. So ist beispielsweise der Betrieb auf 64-Bit-Systemen teilweise mit ärgerlichen Abstürzen verbunden.

Pd ist prinzipiell einfach zu bedienen, allerdings enthält die Standardbibliothek zum Großteil nur sehr elementare Operationen, was das Arbeiten auf einem höheren Abstraktionsniveau schwierig oder langsam macht. Pd-extended versucht dem durch die Integration etlicher zusätzlicher Externalen aus verschiedenen Quellen entgegenzuwirken. Hier wäre ein standardisierter Satz an Modulen, der auch in die Kernversion integriert und regelmäßig gewartet wird, wünschenswert, um die Kompatibilität zwischen Versionen und Systemen zu erhöhen.

4 Pd als Werkzeug phonetischer Analyse

Bisher wurde nur Vorarbeit geleistet, um Kenntnisse um unterschiedliche Stimmqualitäten, Parameter akustisch-phonetischer Analyse und der Programmierung in Pure Data auf eine grundlegende Basis zu bringen. Nun können diese zu einem größeren Ganzen vermengt werden. Mit diesem Kapitel verfolge ich daher vor allem zwei Ziele, die aber nah bei einander liegen: Zum Einen soll Pd als phonetisches Werkzeug ausprobiert und evaluiert werden. Dies geschieht beispielhaft an zwei von mir „geschriebenen“ Patches. Und zum Anderen sollen eben diese dem interessierten Leser als Ausgangspunkt für eigene Unternehmungen in dieser Richtung dienen.

4.1 Vorüberlegungen

Obwohl die Parameter, die untersucht werden wollen im Grunde genommen dieselben sind, unterscheiden sich statische Analyseprogramme, wie etwa Praat, und Echtzeitanwendungen in einigen Aspekten sehr. Dies betrifft vor allem die zugrundeliegenden Algorithmen und die Bereiche, in denen die Programme Anwendung, und deren spezielle Anforderungen finden.

Statische Software wird besonders dort verwendet, wo (Sprach-)daten bereits als Ganzes vorliegen und einzelne akustische Teilaspekte dieser mit hohem Detailgrad untersucht werden sollen. Die Daten sollen von möglichst vielen Seiten betrachtet werden können und die Ausgabe eine hohe Präzision aufweisen. Rechenzeit spielt hier keine primäre Rolle und gerät höchstens mit der Geduld des Anwenders in Konflikt. Die offensichtlichsten Anwendungsbereiche sind hier wohl Forschung und Lehre.

Für Echtzeitanwendungen gelten andere Voraussetzungen. Diese finden dort Einsatz, wo eine unmittelbare Bewertung von Sprachdaten wichtig ist. Hier denke man zum Beispiel an elektronische Spracherkennung und Aussprachetrainer, Diagnostik, automatisiertes Feedback in Logopädie und Rehabilitation von Sprechorganspatienten, die Arbeit im Feld oder Unterrichtszwecke. Die Software sollte intuitiv nutzbar sein und eine schnelle Ausgabe erzeugen. Hier erhofft man sich keine neuen Entdeckungen, sondern möchte, dass aus den Daten all

das herausgelesen wird, was in der aktuellen Situation von Relevanz ist. Der Preis dafür ist, dass ein Teil der Entscheidungsgewalt über den Umgang und die Bewertung der Daten an die Software abgegeben wird. Weiter werden die Ergebnisse nicht eine so hohe Präzision erreichen wie die statischer Anwendungen, insbesondere dann nicht, wenn die Rechenoperationen eigentlich viel Zeit in Anspruch nehmen würden.

Echtzeitalgorithmen müssen deshalb einige besondere Kriterien erfüllen. Während bei bereits vorliegenden Aufnahmen Daten wie Anfang, Ende und der genaue Wellenverlauf bereits bekannt sind, rechnen Echtzeitanwendungen in aller Regel mit dem Ungewissen, denn der Datenstrom ist stetig und weder ist aus ihm ersichtlich, wann er enden, noch wie der zukünftige Verlauf aussehen wird. Weiter können Daten über das aktuelle Signal, die bereits weiter zurückliegen schon aus dem Speicher entfernt worden sein. Strategien könnten hier der Aufbau eines Puffers oder die zeitliche Terminierung von Aufnahmen sein. Mit hoher Wahrscheinlichkeit aber wird die Verarbeitung der Daten nicht an die Präzision derjenigen durch gute statische Software heranreichen. Dadurch, dass weniger Datenpunkte in die Berechnungen mit einfließen, ist gleichzeitig die Gefahr der Störung durch einzelne Ausreißer zusätzlich erhöht. Auch hier müssen wohlüberlegte Heuristiken erhalten.

Die Kunst ist nun, aus diesen suboptimalen Bedingungen zufriedenstellende Ergebnisse zu erzeugen.

4.2 Zwei Beispielpatches

Im Folgenden sollen nun zwei Patches vorgestellt werden, die elementare akustisch-phonetische Parameter in Echtzeit messen. Patch #1 (Abschnitt 4.2.1) misst das Harmonics-to-Noise-Verhältnis und den Jitter eines eingehenden Signals. Der zweite (4.2.2) misst die Cepstral Peak Prominence. Bei der Vorstellung werde ich auch auf Aspekte des Designs zu sprechen kommen, die mit den Überlegungen aus dem letzten Abschnitt zusammenhängen.

setting patch-wide constants

loadbang

0.1

s \$0-bias

bias

2048

s \$0-framesize

size of analysis frame

0.1

s \$0-fidelity

clarity threshold

60

s \$0-sensibility

threshold in dB

r \$0-sensibility

sens \$1

r \$0-bias

bias \$1

r \$0-fidelity

fidelity \$1

adc~

audio input

+

hip~ 160

hi-pass noise

helmholtz~ 2048 1

clarity

0.973788

pd hnratio

15.6996

HNR in dB

pitch >

96.8831

pd jitter

0.336955

sustain voice for at least 3 seconds

delay~ 2048 64

tabwrite~ \$0-snac

\$0-snac

20

4.2.1 Patch #1 – HNR und Jitter

Der erste Patch implementiert Algorithmen zur Messung des Harmonics-to-Noise-Verhältnisses und Jitter. In der vorliegenden Form wurde er hauptsächlich durch das External `helmholtz~` von Vetter (2012) ermöglicht, es wäre jedoch ohne weiteres möglich gewesen mit den Mitteln, die Pd(-extended) bietet, auch die DAC-Funktion selbst zu implementieren. Externals erhöhen jedoch die Performanz, da ihre Funktionen nicht erst zur Laufzeit interpretiert werden müssen, und schaffen nicht zuletzt auch Abstraktion für den Entwickler.

Abbildung 2 zeigt die Hauptroutine des Programms. Oben werden zunächst Konstanten initialisiert, die im Programm Verwendung finden. Drei von ihnen werden direkt an `helmholtz~` übergeben. Der Parameter `sens` bestimmt den Schwellwert in dB ab dem das Objekt auf eingehende Audio-Signale reagiert, `fidelity` setzt einen weiteren Schwellwert in Bezug auf den *Clarity*-Wert (s.u.) und `bias` bestimmt wie stark die Autokorrelation als Funktion des Abstands vom Nullpunkt abfällt. So werden Maxima mit kürzerem Abstand bevorzugt und es lässt sich vermeiden, dass spätere Maxima fälschlicherweise als Periode erkannt werden.

Das Programm an sich nimmt ein einkommendes Signal vom Mikrofoneingang des Rechners, addiert linken und rechten Input (konvertiert also Stereo zu Mono) und filtert Rauschen unterhalb von 160 Hz heraus. Das so vorbereitete Signal wird alsdann an `helmholtz~` übergeben, welches auf eine Fenstergröße von 2048 Samples eingestellt ist, um auch niedrige Frequenzen erkennen zu können. Bei einer Samplerate von 44.100 KHz, erkennt das Objekt Frequenzen ab $\frac{44.100}{2048} \cdot 1,2 \approx 25,8$ Hz. Der linke Ausgang enthält die SNAC-Funktion als Signal, welches auf das Array unten geplottet wird. Das dafür Zuständige Objekt `tabwrite~` wird von der Ausgabe der Grundfrequenz aus dem mittleren Ausgang getriggert. So sind Plot und Anzeige der Frequenz auf jeden Fall im Takt. Die Grundfrequenz wird außerdem zur Berechnung des Jitter-Wertes im Unterpatch `pd_jitter` herangezogen.

Der rechte Ausgang von `helmholtz~` gibt die Clarity an. Dieser Wert bezeichnet nichts anderes als $r'_x(\tau_{max})$ aus Abschnitt 2.1, also die normalisierte Periodi-

zität. Damit lässt sich umgehend das HNR errechnen, was im Subpatch `hnratio` (Abb. 3) geschieht. Dort wird zunächst der eingehende Clarity-Wert von 1 subtrahiert und das Ergebnis anschließend durch `moses` mit 0 verglichen. Ist er kleiner oder gleich 0, wird 1000 als Ausgabe getätigt, stellvertretend für einen unendlich hohen Wert, gegen den die HNR-Funktion konvergiert, wenn ihr Argument gegen 0 geht. Andernfalls wird das HNR auf die reguläre Weise errechnet.

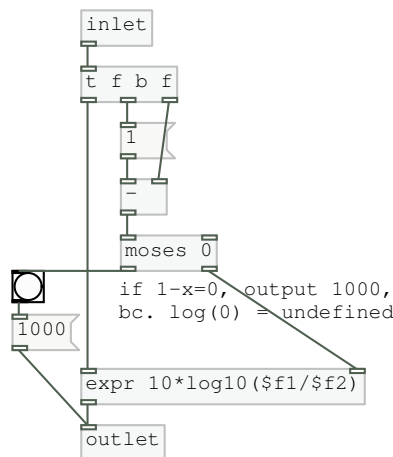


Abbildung 3: Subpatch zur Errechnung des HNR aus der normalisierten Periodizität (Clarity) eines Signals.

Auf Abbildung 4 ist der Unterpatch `pd jitter` zu sehen. Der dort verwendete Algorithmus ähnelt dem regulären zur Errechnung des Jitters, ist jedoch nicht völlig identisch, da er an Echtzeitverhältnisse angepasst werden musste. Da das Ende eines eingehenden Datenstroms nicht vorhersagbar ist, kann nicht über eine feste Anzahl an Perioden gemittelt werden. Der paarweise Vergleich zweier Perioden hat sich als zu instabil und anfällig gegenüber schon minimalen Schwankungen erwiesen. Daher gibt es einen Puffer der Größe 20, der mit Periodenlängewerten gefüllt wird und aus dem anschließend die Werte wieder zur Berechnung gezogen werden. Ein weiterer Unterschied ist der, dass nicht alle Perioden in die Berechnung mit einfließen, sondern nur die jeweils erste, die sich im Fenster der verwendeten DAC-Funktion abzeichnet. Dieser Punkt lässt sich sicher noch beseitigen, für Demonstrationszwecke ist diese Abkürzung jedoch völlig ausreichend. Diese beiden Eigenschaften der Echtzeit-Jitter-

Implementation führen jedoch dazu, dass die Stimme zur Errechnung des Jitters erst für eine Zeit von mindestens $2048 \cdot 20$ Samplen gehalten werden muss, bis eine verlässliche Ausgabe erfolgen kann.

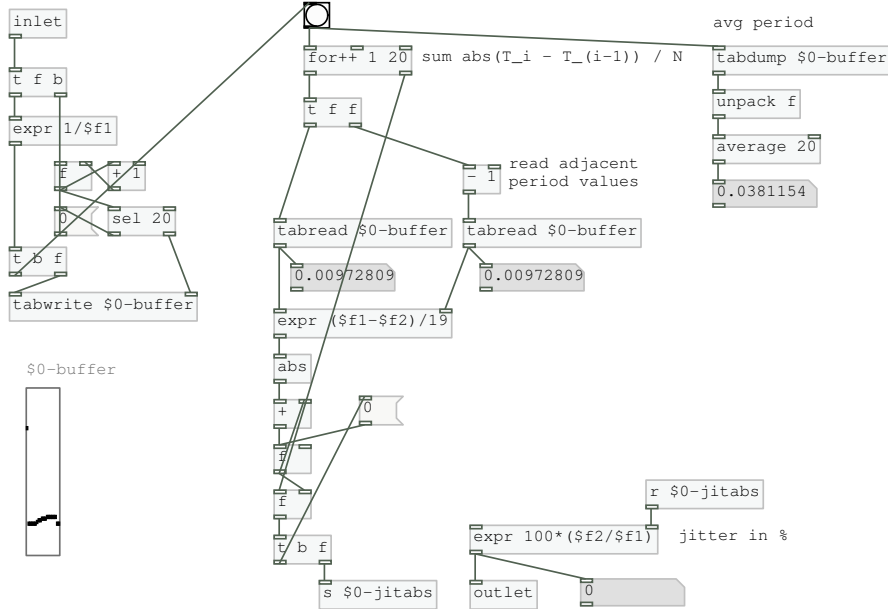


Abbildung 4: Subpatch zur Errechnung des Jitters eines Signals. Dazu wird zunächst ein Puffer (u.l.) gefüllt und gemittelt. Dadurch wird die Berechnung langsamer, aber genauer.

Der Ablauf des Subpatches lässt sich folgendermaßen beschreiben: Eingehende Frequenzen werden in Perioden konvertiert ($T = \frac{1}{f}$) und im Puffer abgelegt. Der Zähler, der die Indizes hochzählt, setzt sich alle 20 Schritte wieder auf 0 zurück. Anschließend wird einerseits der absolute Jitter errechnet (Mitte) und andererseits die mittlere Periode (rechts).

Zur Errechnung des absoluten Jitter werden mittels des jeweiligen Index' der Vorschleife zwei benachbarte Werte aus dem Puffer geholt, ihre Differenz ermittelt und auf die schon vorhandenen Werte in einem float-Speicher aufsummiert. Da die Summierung die Nutzung des heißen Eingangs des f-Objekts benötigt, gibt dieses jedes Mal den aktuellen Wert aus. Dieser wird vom kalten Eingang eines weiteren floats aufgefangen. Das for++-Objekt sendet nach Beendigung eines Schleifendurchgangs einen bang und triggert damit die Ausgabe

des absoluten Jitters, welcher zunächst an den kalten Eingang des `expr`-Objekts unten rechts angelegt wird.

Die Errechnung der durchschnittlichen Periode ist wesentlich einfacher nachzuvollziehen: `tab_sum` summiert alle Werte des Puffers auf, wodurch durch die Anzahl der Werte dividiert wird. Das Verhältnis von absolutem Jitter zu durchschnittlicher Periode wird noch ver Hundertfacht, um eine prozentuale Ausgabe zu erreichen.

Der Patch erzeugt für Jitter und HNR in Bezug auf Modal-, Hauch- und Knarrstimme in aller Regel zufriedenstellende Daten, die denen aus der Literatur sehr nahe kommen. Es wäre jedoch interessant zu sehen, wie er sich in den Grenzbereichen von bspw. gesunder zu pathologischer Stimme im Vergleich zu bereits etablierter Software verhält. Da ich denke, dass besonders diese Informationen für die Anwendung interessant sind, wäre eine Optimierung der Präzision in solchen Schwellwertbereichen ein nächster Schritt.

4.2.2 Patch #2 – Cepstral Peak Prominence

Der zweite Patch auf Abbildung 5 errechnet die Cepstral Peak Prominence eines (periodischen) Inputs. Das Cepstrum wird erzeugt durch das `cepstrum~`-Objekt aus der Bibliothek *timbreID* von Brent (2009). Es gibt das Cepstrum als Liste aus, sodass diese leicht mittels `list split` zurechtgestutzt werden kann. Da Sprache auch sehr tiefe Frequenzbereiche umfasst, wird hier das gesamte darstellbare Cepstrum bis Sample 1025 verwendet; allerdings erst ab Sample 12, da der Bereich darunter von Hillenbrand et al. (1994) als besonders störanfällig angegeben wird.

Der Subpatch `pd regression` errechnet die Gerade der Linearen Regression des (geglätteten) Cepstrums. Aufgrund seiner hohen Komplexität habe ich davon abgesehen, diesen Unterpatch hier näher zu erläutern, würde es doch den Skopus dieser Arbeit übersteigen. Daher soll nun direkt der Subpatch `pd cpp` beleuchtet werden.

In diesem spielt wieder `helmholtz~` als Stimmgerät eine Rolle. Da sich die gesuchte Spitze des Cepstrums, der Cepstral Peak, an einer Stelle befindet, die der Grundfrequenz des Signals entspricht, kann andersherum die Grundfre-

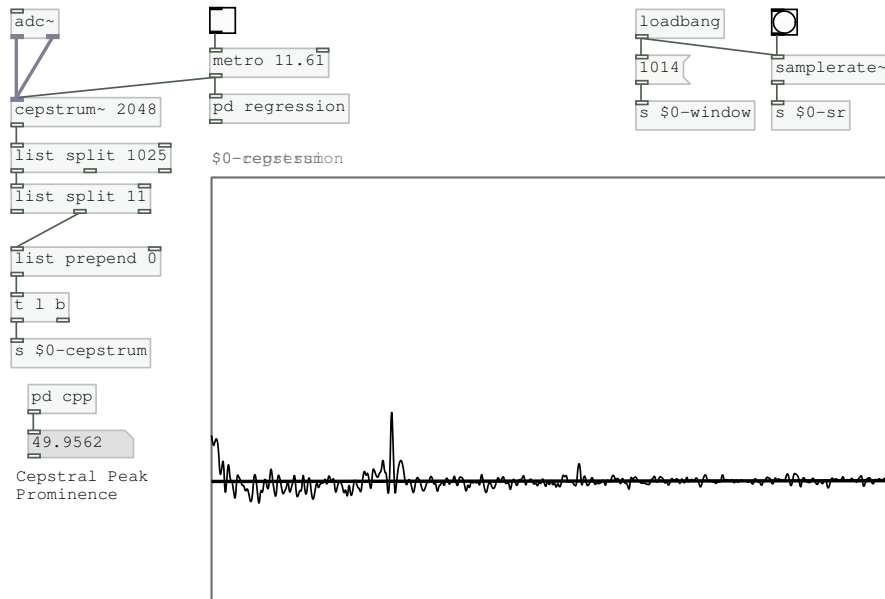


Abbildung 5: Hauptroutine eines Patches zur Errechnung der Cepstral Peak Prominence. Der Plot zeigt das Cepstrum eines Vokals und seine Regressionslinie.

quenz genutzt werden, nach eben dieser Stelle zu suchen. Dafür wird unter Berücksichtigung der Samplerate die Grundfrequenz in Quefrenz bzw. Sample umgerechnet – $\frac{44.100}{f_0}$ – und von dieser der Offset von 11 Samples und ein weiterer, abgeschätzter Toleranzwert abgezogen. Das Objekt `tab_max_index` findet das Maximum und dessen Index in einem Array. Zudem kann ihm ein Argument in Form von Startpunkt und Länge eines Bereichs übergeben werden, innerhalb dessen nach dem Maximum gesucht werden soll. Hier ist dies die aus der Grundfrequenz errechnete Position und eine Länge von 8 (Schätzung) Punkten bzw. Samples. Damit das Objekt etwas ausgibt, muss es gebangt werden. Dies geschieht über Triggerung durch die Ausgabe der Grundfrequenz des `helmholtz~`-Objekts. Dadurch ist gewährleistet, dass nur dann der CPP errechnet wird, wenn auch wirklich ein Signal gesendet wird. Die abschließende Multiplikation mit 1000 dient nur der besseren Lesbarkeit der Ausgabe und ist kein Wert in dB.

Anmerkung: Der Patch funktioniert zwar, jedoch noch nicht sehr zuverlässig.

PHONETISCHE ANALYSE MIT PURE DATA

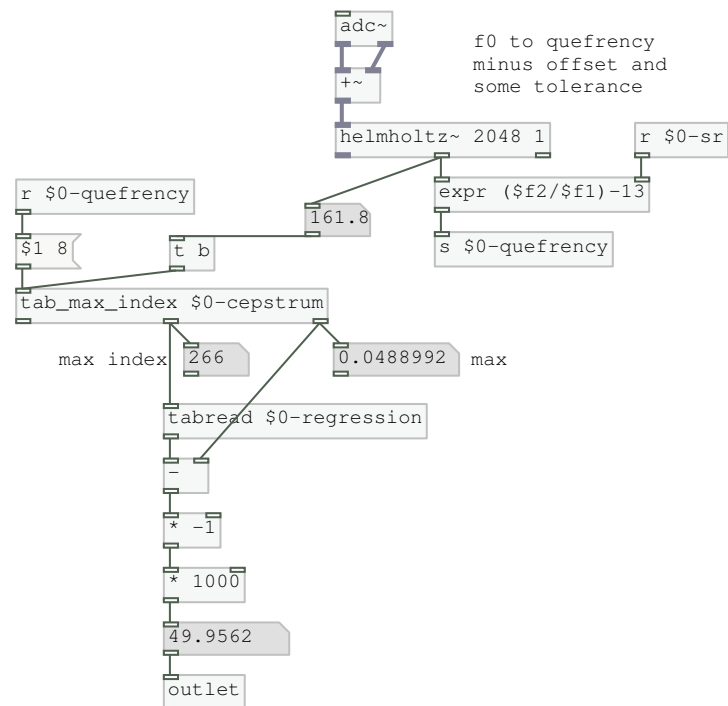


Abbildung 6: Dieser Subpatch errechnet die CPP. Zur Eingrenzung des Suchraumes wird die Grundfrequenz des Signals mit einkalkuliert (o.r.).

Insbesondere das Grundrauschen des Cepstrums stellt hier einen großen Störfaktor dar, der schwache Peaks zuweilen überdeckt. Hier sollte die Einbringung einer elaborierteren Glättungsfunktion, auch schon im Frequenzbereich, wie etwa von Hillenbrand & Houde (1996) beschrieben, ins Auge gefasst werden. Kritik gibt es auch am `cepstrum~`-Objekt: Die Ausgabe bewegt sich größtenteils in einem Bereich zwischen $-0,1$ und $0,2$ Einheiten, wobei nicht dokumentiert scheint, um was für Einheiten es sich handelt.

4.3 Evaluation

Mit Pure Data lassen sich auf vergleichsweise einfache Weise Anwendungen erstellen, die in Echtzeit akustische Daten verarbeiten. In den über 15 Jahren der Existenz dieses Systems gab es immer wieder Beiträge in Form von Erweiterungen aus der Community, die das Arbeiten mit der Umgebung zum Einen erleichtert und sie zum Anderen auch offen für andere Bereiche als die reine Audio-Verarbeitung gemacht haben.

Trotz des intuitiven ersten Eindrucks einer grafischen Programmierumgebung sollte man sich bewusst sein, dass der Umgang mit Pd ein tieferes Verständnis von Akustik und Signalverarbeitung erfordert. Viele höhere Analysemethoden wird man vor allem als Phonetiker vergeblich suchen. Doch: Hat man die nötigen Kenntnisse und sich mit der Funktionsweise des Programms vertraut gemacht, wird die Arbeit mit ihm zunehmend intuitiver und man wird häufig sogar mit sofortigem Feedback belohnt.

Die grafische Oberfläche ermöglicht es zudem im Prinzip Anwendungen für Dritte zu erschaffen, die diese dann für ihre Zwecke einsetzen. Dafür sollte aber ein Interface geschaffen werden, welches das unwillentliche Verändern der Programmstrukturen erschwert oder verhindert. Weiter hat Pd noch häufig mit Abstürzen zu kämpfen, was auch erst in den Griff bekommen werden muss.

Meine Patches, auch wenn sie nicht vollends ausgereift sind, haben, so denke ich, gezeigt, wie und dass es in Pure Data möglich ist, Echtzeitanwendungen zu erschaffen, die phonetisch Relevante Parameter untersuchen. Ich hoffe zudem, mit meinem Unterfangen Interesse an dieser Art akustischer und phonetischer Arbeit geschaffen zu haben.

5 Fazit und Ausblick

Diese Arbeit hat am Beispiel dreier Stimmqualitäten gezeigt, wie artikulatorische und akustische Phänomene im Zusammenhang stehen. Dabei lag der Schwerpunkt in Kapitel 1 stärker auf der Artikulation, wohingegen in Kapitel 2 die daraus resultierenden und messbaren akustischen Parameter im Vordergrund standen. Dies galt als Vorbereitung für ein anderes Ziel, nämlich die Multimedia-Programmierung Pure Data in den phonetischen Diskurs zu bringen. Ihre Funktionsweise wurde in Kapitel 3 kurz erläutert, um dann in Kapitel 4 anhand zweier in ihr programmierten Programme auf die ersten beiden Kapitel zurückzublicken. Hier hat sich nämlich gezeigt, dass Pd gut darauf ausgelegt ist, auch im Bereich phonetischer Datenverarbeitung in Echtzeit Verwendung zu finden.

Damit Pd sich aber weiter in diesem Bereich etablieren kann, müssen noch eine Reihe an Verbesserungen vorgenommen werden: Regelmäßigere Updates, eine umfangreichere Standardbibliothek, Erhöhung der Kompatibilität zwischen Systemen und Versionen, und einiges mehr.

Nichtsdestoweniger halte ich Pd für ein Werkzeug, was man im Auge behalten sollte und ich blicke gespannt und erwartungsfroh in die Zukunft.

Literatur

- Blomgren, M., Y. Chen, N. L. Ng, & H. R. Gilbert. 1998. Acoustic, aerodynamic, physiologic, and perceptual properties of modal and vocal fry registers. *Journal of the Acoustic Society of America* 103:2649–2658.
- Boersma, P. 1993. Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. In *IFA Proceedings*, 17. Amsterdam: University of Amsterdam.
- Boersma, P., & D. Weenink. 2012. *Praat manual*. University of Amsterdam, Amsterdam. URL <http://www.fon.hum.uva.nl/praat/manual/Intro.html>.
- Bogert, B. P., M. J. R. Healy, & J. W. Tukey. 1963. The quefrency alalysis of time series for echoes: Cepstrum, pseudo autocovariance, cross-cepstrum and saphe-cracking. In *Proceedings of the Symposium on Time Series Analysis*, Hrsg. M. Rosenblatt, 209–243. New York: Wiley.

- Brent, W. 2009. A timbre analysis and classification toolkit for pure data. URL <http://williambrent.conflations.com/papers/features.pdf>.
- Childers, D. G., & C. K. Lee. 1991. Vocal quality factors: Analysis, synthesis, and perception. *Journal of the Acoustic Society of America* 90:2394–2410.
- Esling, J. H., & J. G. Harris. 2005. States of the glottis: An articulatory phonetic model based on laryngoscopic observations. In *A figure of speech. a festschrift for John Laver*, Hrsg. William J. Hardcastle & J. Mackenzie Beck, 347–383. New Jersey: Lawrence Erlbaum Associates.
- Gordon, M., & P. Ladefoged. 2001. Phonation types: a cross-linguistic overview. *Journal of Phonetics* 29:383–406.
- Hillenbrand, J., R. A. Cleveland, & R. L. Erickson. 1994. Acoustic correlates of breathy vocal quality. *Journal of Speech and Hearing Research* 37:769–778.
- Hillenbrand, J., & R. A. Houde. 1996. Acoustic correlates of breathy vocal quality: Dysphonic voices and continuous speech. *Journal of Speech and Hearing Research* 39:311–321.
- Hirose, H. 1997. Investigating the physiology of laryngeal structures. In *The handbook of phonetic sciences*, Hrsg. William J. Hardcastle & John Laver, 116–136. London: Blackwell.
- Hollien, H., P. Moore, R. W. Wendahl, & J. Michel. 1966. On the nature of vocal fry. *Journal of Speech and Hearing Research* 9:245–247.
- Kreidler, J. 2009. *Loadbang: Programming electronic music in Pure Data*. Hofheim am Taunus: Wolke.
- Laver, J. 1980. *The phonetic description of voice quality*. Cambridge University Press.
- McLeod, P. G. 2008. *Fast, accurate pitch detection tools for music analysis*. Dissertation, University of Otago, Dunedin.
- Monsen, R. B., & A. M. Engebretson. 1977. Study of variations in the male and female glottal wave. *Journal of the Acoustic Society of America* 62:981–93.
- Puckette, M. S. 1996. Pure Data: another integrated computer music environment. In *Second Intercollege Computer Music Concerts*. Tachikawa, Japan: Kunitachi College of Music.
- Rabiner, L. R., & R. W. Schafer. 1978. *Digital processing of speech signals*. New Jersey: Prentice-Hall.
- Stevens, K. N. 1997. Articulatory-acoustic-auditory relationships. In *The handbook of phonetic sciences*, Hrsg. William J. Hardcastle & John Laver, 462–506. Oxford: Black.
- Vetter, K. 2012. Helmholtz pitch tracker for puredata. URL <http://www.katjaas.nl/helmholtz/helmholtz.html>.