# Transkription monophoner Gesangszeitreihen

**Dissertation**

zur Erlangung des Grades
eines Doktors der Naturwissenschaften
der Universität Dortmund

Dem Fachbereich Statistik
der Universität Dortmund

vorgelegt von
**Uwe Ligges**

Dortmund, April 2006

**Gutachter**:

Prof. Dr. Claus Weihs

Prof. Dr. Katja Ickstadt

**Tag der mündlichen Prüfung**:

28. Juni 2006

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Zum Musizieren, also zum Spielen eines Instrumentes oder zum Singen, gehört meist auch das Lesen von Noten und die Umsetzung der Notenschrift in Töne. Mit einem gewissen Maß an Übung können Menschen das Musizieren nach Noten erlernen. Auch Computer können ohne besonders großen Aufwand digitalisierte Noten in korrespondierende Töne umsetzen.

Die Umkehrung dieses Prozesses ist die *Transkription*, bei der Musik in Notenschrift überführt werden soll. Wenn Menschen von ihnen gehörte Musik aufschreiben sollen, brauchen sie dazu Talent und sehr viel Übung. Selbst einfache monophone Musikstücke müssen sie dazu mehrfach anhören. Wünschenswert ist es daher, die Transkription automatisch von einem Computer durchführen zu lassen.

Transkription wird besonders dann eingesetzt, wenn ein bisher nicht notiertes Volkslied aufgeschrieben werden soll, oder ein Laie oder Amateurmusiker Noten für Musik haben möchte, die er eben gehört hat aber nur summen und nicht namentlich angeben kann. Weitere Einsatzgebiete der Transkription können die Fehlerverbesserung beim Musikunterricht und beim Üben zu Hause sein sowie das automatische Erkennen eines Liedes und die Zuordnung zu einer CD, wenn ein Kunde im Kaufhaus den Titel vergessen hat, wohl aber noch die Melodie summen kann.

Aus diesen Gründen ist es besonders interessant, Transkription von Gesangsdaten

automatisch durchführen zu können. Da die menschliche Stimme aber ein besonders komplexes „Instrument“ ist, bei dem neben der Lautstärke auch der Klang sehr stark variiert werden kann, ist die Automatisierung hier besonders schwierig. Insbesondere muss dabei zusätzlich eine gewisse Robustheit gegen Fehler in dem Gesang selbst gegeben sein. Andererseits möchte man auch mit möglichst wenig Hintergrundinformation über die Stimme der Sängerin oder des Sängers auskommen, wobei das Anpassen von Parametern der Verfahren durch statistische Lern- oder Schätzverfahren an eine bestimmte Stimme von den Betrachtungen jedoch nicht ausgeschlossen werden soll.

Bisherige Verfahren und Algorithmen zur Transkription wurden meist entweder für MIDI-Daten oder für das Klavier und Zupf- oder Schlaginstrumente entwickelt. Die Transkription von MIDI-Daten ist recht einfach, denn Informationen zu Tonbeginn, Tonende und Tonhöhe liegen dort bereits in digitaler Form vor. Es muss also keine Schätzung dieser Informationen aus der Schwingung erfolgen. Die Transkription von Musikinstrumenten, die einen Anschlag haben (Klavier, Gitarre usw.), erfolgt zwar aus der Schwingung, wird aber dadurch erleichtert, dass allein durch die plötzlich ansteigende Amplitude des Signals der Beginn eines neuen Tons angezeigt wird.

Versuche mit Demo-Versionen einiger Transkriptionsprogramme haben gezeigt, dass sie für die in dieser Arbeit verwendeten Gesangsbeispiele völlig ungeeignet waren. Insbesondere das Vibrato von professionellen Sängerinnen und Sängern, die Möglichkeit Töne leise zu beginnen und zu entwickeln, das Singen von stimmlosen Konsonanten und die starke Formbarkeit des Klangs machen den vorhandenen Algorithmen zu schaffen. In dieser Arbeit werden Algorithmen entwickelt, die den bisherigen Verfahren durch Modellbildung aber auch durch Parameteroptimierung einer Heuristik bei der Transkription von Gesangsdaten überlegen sind.

Die Herangehensweise an das Problem der Transkription von Gesang ist wie folgt strukturiert. Zunächst werden in Kapitel 2 einige musikalische und statistische Grundlagen erläutert, die für das Verständnis des weiteren Vorgehens wichtig sind. Auch die verwendeten Daten werden dort beschrieben. Die einzelnen Schritte der

Transkription werden in Kapitel 3 zusammen mit bereits bekannten Verfahren beschrieben. Da sich gezeigt hat, dass besonders die genaue Schätzung der Grundfrequenz eines Tones eine hohe Schwierigkeit für Gesangsdaten darstellt, werden hierzu verschiedene Ansätze vorgestellt.

Ein auf einer Heuristik zur Grundfrequenzschätzung basierendes Verfahren wird in Kapitel 4 beschrieben. Dieser Algorithmus bietet in besonders kurzer Rechenzeit eine genaue Grundfrequenzschätzung, auf deren Basis bei simulierten Daten alle Notenwerte korrekt klassifiziert werden konnten. Die Anpassung eines Modells im Zeitbereich wurde zugunsten einer Optimierung eines ähnlichen Modells im Frequenzbereich verworfen (Kapitel 5.2). Einer extrem genauen Grundfrequenzschätzung stehen bei dieser Optimierung einige Ausreißer und eine lange Rechenzeit gegenüber. In Kapitel 5.3 wird die Schätzung eines Modells mit Hilfe eines MCMC Verfahrens beschrieben. Zur Durchführung der Schätzung wurde das R Paket *BRugs* mitentwickelt. Die verschiedenen Verfahren zur Frequenzschätzung werden in Kapitel 6 verglichen.

In Kapitel 7 wird das im Rahmen dieser Arbeit entwickelte R Paket *tuneR* vorgestellt, das eine Software Umgebung für die statistische Analyse von Musikdaten bildet. Eine Zusammenfassung der Ergebnisse und Ausblicke auf mögliche weitere Untersuchungen und die Transkription von polyphonem Klang gibt Kapitel 8.

# Kapitel 2

# Grundlagen

## 2.1 Musikalische Grundlagen

In diesem Abschnitt wird auf einige Aspekte aus dem Bereich der Musik eingegangen, die für die weiterführenden Überlegungen von grundlegender Bedeutung sind. Da es in verschiedenen Gegenden der Erde sehr unterschiedliche Hörgewohnheiten gibt, beschränkt sich die Betrachtung auf die zur Zeit im westeuropäischen Raum als üblich geltenden Hörgewohnheiten. Unterschiede gibt es beispielsweise bei Stimmung, Anzahl der Töne einer Tonleiter und dergleichen.

### 2.1.1 Musikzeitreihen und deren Repräsentation

Eine einfache und sehr gebräuchliche Art, Musik digital zu speichern, ist das binäre *Wave*–Format (Microsoft Corporation, 1991), das im Folgenden stets als Grundlage für die weiterführenden Analysen verwendet wird.

Bei Wave–Dateien wird die Amplitude der Schwingung mit konstant gleicher Abtastrate (auch Samplingrate genannt, z.B. 11025, 22050 oder 44100 Hertz) aufgezeichnet und einer Skala zugeordnet. Gebräuchlich für eine solche Skala ist das 8–Bit

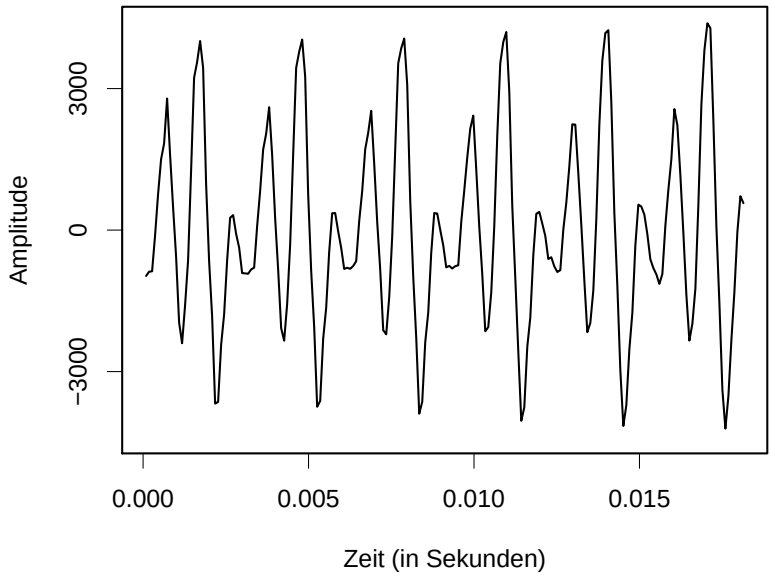


Abbildung 2.1: Ausschnitt einer Musikzeitreihe

bzw. das 16–Bit Format. Bei dem 8–Bit Format handelt es sich bei den möglichen Werten um ganze Zahlen zwischen 0 und 255 ($2^8$ Möglichkeiten, der Nulldurchlauf liegt bei 128), während in 16–Bit Wave–Dateien ganze Zahlen zwischen -32767 und 32766 ($2^{16}$ Möglichkeiten) angenommen werden können. Die Qualität der Aufnahme auf einer CD entspricht einer Wave–Datei mit 16–Bit und 44100 Hertz.

Aus einer Wave–Datei kann man die Musikzeitreihe nach kleinen technischen Veränderungen, etwa der Umwandlung von der binären Form der Wave–Datei in eine ASCII–Datei, ablesen. Auch das Paket *tuneR* (siehe Kapitel 7) bietet die Möglichkeit des Imports von Wave–Dateien.

In Abbildung 2.1 ist ein Ausschnitt einer Gesangszeitreihe abgebildet. Hier handelt es sich um eine mono 16–Bit Wave–Datei mit einer Samplingrate von 11025 Hertz. Offensichtlich ist nicht nur eine Sinusschwingung vorhanden, sondern eine Überlagerung mehrerer Schwingungen.

Falls die absolute Amplitude bei den folgenden Analysen keine Rolle spielt, werden die Zeitreihen immer auf das Intervall $[-1, 1]$ skaliert, um einen einfacheren Umgang

mit den Daten und Unabhängigkeit von dem verwendeten Aufzeichnungsformat zu erhalten.

Ein völlig anderes Format der Musikrepräsentation ist *MIDI* (MIDI Manufacturers Association, 2001). In MIDI-Dateien werden nicht Schallereignisse (wie bei Wave-Dateien) sondern Musikereignisse dargestellt. In einer MIDI-Datei enthaltene Informationen lassen sich wie folgt veranschaulichen:
„Instrument 1 beginnt Ton $a'$ mit Lautstärke 70% zum Zeitpunkt $t = 1.2123$ Sekunden, Stimmung mit 440 Hz“ gefolgt von
„Instrument 1 beendet Ton $a'$“. Die Klangeigenschaften von „Instrument 1“ können dabei beliebig festgelegt werden. Falls eine MIDI-Datei vorliegt, können bei der Transkriptionsaufgabe daher die sonst wesentlichen Schritte der Grundfrequenzschätzung und Segmentierung (siehe Kapitel 3) wegfallen.

### 2.1.2 Grundton, Obertöne und Klang

In der Musik enthält ein Ton im Allgemeinen nicht nur die Schwingung mit der Grundfrequenz (z.B. ein $a'$ mit 440 Hertz), sondern auch sogenannte *Obertöne*. Die Schwingungen von Grundton und Obertönen überlagern sich dann. Die Obertöne sind dadurch charakterisiert, dass ihre Frequenz ein ganzzahliges Vielfaches der Frequenz des Grundtons ist. Zu $a'$ mit 440 Hertz gehörige Obertöne sind:
$a''$ mit 880, $e'''$ mit 1320, $a'''$ mit 1760, $cis''''$ mit 2200 Hertz usw.

Die Amplitude, mit der Grundton und Obertöne schwingen, hängt von vielen Faktoren ab. Bei einem Sänger sind das beispielsweise der Bau von Kehlkopf und Rachenraum, die Übung oder auch die Stimmlage, in der der Ton gesungen wird (Seidner und Wendler, 1997).

Tabelle 2.1: Tonhöhenumfang (Frequenzen in Hertz)

| | tiefster Ton | | höchster Ton | |
|---|---|---|---|---|
| Bass | $D$ | (73) | $f'$ | (349) |
| Bariton | $F$ | (87) | $a'$ | (440) |
| Tenor | $H$ | (123) | $d''$ | (587) |
| Alt | $c$ | (131) | $g''$ | (784) |
| Mezzosopran | $g$ | (196) | $c'''$ | (1046) |
| Sopran | $h$ | (247) | $f'''$ | (1396) |

Bei der Analyse von Gesangszeitreihen sollte zunächst bekannt sein, in welcher Stimmlage welche (Grund-)Töne gesungen werden können (Seidner und Wendler, 1997). Der tiefste Ton eines Basses ist in der Regel ein $D$ mit etwa 73 Hertz, während der höchste Ton der Soprane ein $f'''$ mit etwa 1396 Herz sein kann (vgl. Tabelle 2.1).

Insbesondere die Verteilung der Anteile verschiedener Obertöne am Ton machen den Gesamtklang des Tons aus. Der Mensch kann den Klang seiner Stimme sehr stark beeinflussen. Ausgebildete Stimmen haben beispielsweise Spektren, die sehr viel reicher an Obertönen sind als die Stimmen von Laien. Gerade großer Oberton-Reichtum macht die Schätzung der Grundfrequenz schwierig. Eine interessante Arbeit zu den Obertonspektren der Vokale der menschlichen Stimme haben Klein et al. (1970) veröffentlicht. Eigene Arbeiten zu diesem Thema sind Weihs und Ligges (2003) und Weihs et al. (2005b). Detaillierteren Erklärungen zu Obertonspektren von Instrumenten findet man bei Berg und Stork (1982), Blackham (1988) und Reuter (2002). Einen anderen Ansatz zur Klangidentifikation verfolgt Röver (2003) mit Hilfe der Hough-Transformation.

### 2.1.3 Stimmung und Abstand von Halbtönen

Die Stimmung für ein $a'$ muss nicht unbedingt immer auf 440 Hertz festgesetzt sein, denn unabsichtlich aber auch gewollt kann die Stimmung davon abweichen.

Freunde alter Musik stimmen meist wesentlich tiefer (z.B. $a'$ mit 430 Hertz), während moderne Orchester oft höher (mit 442 bis 444 Hertz) stimmen.

Auch der Abstand der Halbtöne für die Stimmung ist zunächst nicht eindeutig festgelegt. Bei den weiteren Überlegungen wird davon ausgegangen, dass nach der heute gebräuchlichen *gleichtemperierten Stimmung*, und nicht etwa nach *reiner Stimmung*, gespielt bzw. gesungen wird. Der Vorteil der temperierten Stimmung für die Musik liegt darin, dass Modulationen durch die Tonarten möglich sind.

Ein Nebeneffekt der temperierten Stimmung ist, dass sich der Zusammenhang von Frequenz und Halbtönen mit Hilfe einer stetigen Funktion beschreiben lässt (vgl. auch Abbildung 2.2):

$$\lambda_Z = 2^{\frac{\Delta_H}{12}} \cdot \lambda_A, \tag{2.1}$$

wobei $\Delta_H$ den Abstand der Frequenz $\lambda_Z$ des Zieltons von der Frequenz $\lambda_A$ des Ausgangston in Anzahl an Halbtonschritten bezeichnet. Auch die direkt daraus folgende Berechnung des Abstandes zweier bekannter Frequenzen

$$\Delta_H = 12 \cdot log_2 \left( \frac{\lambda_Z}{\lambda_A} \right) \tag{2.2}$$

ist oft interessant.

Die Oktave eines Grundtons schwingt stets mit der doppelten Frequenz dieses Tons. Durch die 12 Halbtonschritte ist die Oktave bereits in 12 Teile zerlegt. Um zu einem Grundton die Frequenz des nächst höheren Halbtons zu berechnen, multipliziert man die Frequenz des Grundtons daher mit $\sqrt[12]{2}$.

Soll die ideale Frequenz eines $c'$ bei einem mit $a' = 440$ Hertz gestimmten Instrument berechnet werden, so ist der Abstand der Halbtöne des $c'$ vom $a'$ zunächst $\Delta_H = -7$. Nach (2.1) ist dann also $\lambda_{c'} = 2^{\frac{-7}{12}} \cdot 440 \text{ Hz} = 293.66 \text{ Hz}$.

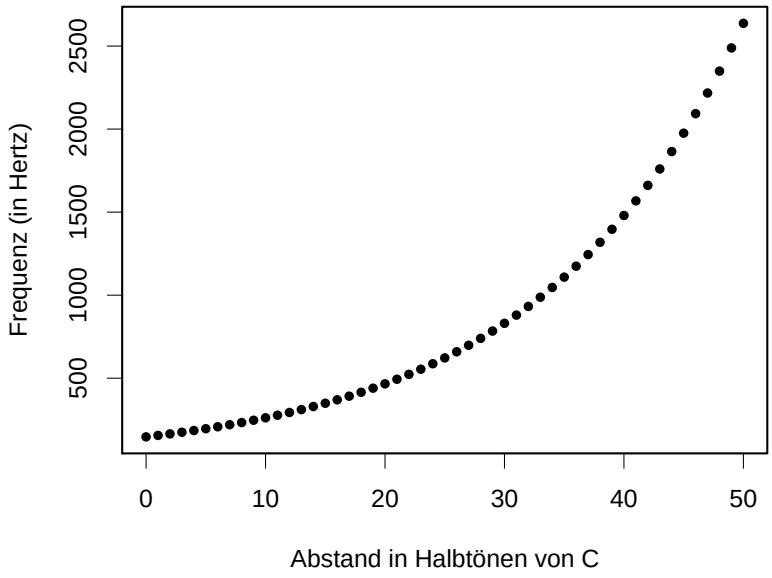


Abbildung 2.2: Halbtöne und deren Frequenzen

### 2.1.4 Vibrato und andere Verzierungen

Häufig werden in der Musik Töne „verziert“, so dass diese Töne als schöner oder interessanter empfunden werden. Es gibt dabei vom Komponisten vorgegebene Verzierungen wie *Tremolo* (kurzes, sehr schnelles Aufeinanderfolgen des gleichen Tons) oder *Triller*. Bei der einfachsten Art des Trillers wird abwechselnd der notierte Ton und der nächst höhere Ton (entsprechend der Tonart eventuell nur ein Halbton höher) gespielt bzw. gesungen.

Die Geschwindigkeit des Trillers ist häufig gleich der Geschwindigkeit des Vibratos, d.h. 5 bis 7 Töne pro Sekunde, kann aber auch zwischen 2 und 12.4 Tönen pro Sekunde liegen (Seidner und Wendler, 1997).

Andererseits gibt es auch vom Interpreten gewollt oder ungewollt gebrauchte Verzierungen wie das *Vibrato*. Hier wird beim Halten eines Tones nicht nur die Lautstärke (genauer: der Schalldruck) um 2 bis 3 Dezibel variiert, sondern auch die Tonhöhe schwankt etwa um einen Halbton ($\pm\frac{1}{4}$ Ton) um den notierten Ton (Seidner und Wendler, 1997). Nach Meyer (1995) kann die Frequenz bei einem besonders forcier-

ten Vibrato sogar um bis zu vier Halbtöne ($\pm\frac{1}{1}$ Ton) schwanken. Es treten dabei häufig 5 bis 7 Schwingungen der Tonhöhe nach oben und unten pro Sekunde auf.

Da bei Triller und Vibrato sowohl die Geschwindigkeit als auch Stärke der Schwankung der Tonhöhe gleich sein kann, besteht der einzige sich aus der Definition ergebende messbare Unterschied darin, dass die Schwankung der Tonhöhe beim Vibrato um den notierten Ton und beim Triller zwischen diesem und dem nächst höheren Halbton liegt.

Zur Vibratoanalyse gibt es eine Arbeit von Rossignol et al. (1999a), die mit als Basis für das in Kapitel 5 beschriebene Modell dient. Ein anderes Verfahren zur Vibratoerkennung in monophonem Klang stellen Pang und Yoon (2005) vor.

### 2.1.5 Pausen, Stille und Rauschen

Selten sind Musikaufnahmen so perfekt, dass bei Stille tatsächlich nichts zu hören ist. Vielmehr werden neben Atemgeräuschen von Musikern und Lüftungsgeräuschen im Tonstudio auch Geräusche durch die Aufnahmeelektronik verursacht. In Zeiten von musikalisch gewollten Pausen wird daher stets ein Rauschen im Signal vorliegen. Dabei handelt es sich meist um sogenanntes „rosa Rauschen“ (Radeka, 1969), dessen Name von der Farbe des Lichts abgeleitet ist. Niedrige Frequenzen, also tiefe Töne, haben dabei nämlich eine größere Amplitude als hohe Frequenzen. Wikipedia[1] beschreibt es sehr anschaulich:

> „Aufgrund der Tatsache, dass die spektrale Leistungsdichte sich reziprok zur Frequenz $f$, d.h. proportional zu $1/f$, verhält, bezeichnet man dieses [rosa] Rauschen auch als $1/f$-Rauschen. [...]
>
> Dabei enthält die Oktave zwischen 20 und 40 Hz die gleiche Rauschleistung wie die Oktave zwischen 10000 und 20000 Hz. Bei jeder doppelten Frequenz ist die Leistung halbiert. [...] Rosa Rauschen klingt fast so, als ob gleichmäßige Lautstärke bei allen Frequenzen vorhanden wäre.“

[1] http://www.wikipedia.de, Stand: 15.04.2006

Die (spektrale) Leistung(sdichte) ist dabei definiert als die Fouriertransformation der zeitlichen Autokorrelationsfunktion, die dem Periodogramm entspricht.

Polotti und Evangelista (2000) modellieren gerade Phänomene wie rosa Rauschen mit Hilfe von Wavelet Techniken.

## 2.2 Statistische Grundlagen

### 2.2.1 Stückweise lokale Stationarität

Für die meisten Methoden der Zeitreihenanalyse werden stationäre Prozesse vorausgesetzt. Wegen des konstanten Mittelwertes von 0 (vgl. Kapitel 2.1.1) handelt es sich bei Gesangszeitreihen um sogenannte *mittelwertstationäre Prozesse* (Schlittgen und Streitberg, 1997), denn es ist der Erwartungswert $\mu_t = 0$ für jeden Index (Zeitpunkt) $t$ aus der Indexmenge $T$. Es liegen wegen der häufigen Änderungen von sich überlagernden Frequenzen, wie auch Änderungen der Amplitude, im Allgemeinen jedoch keine *kovarianzstationären Prozesse* bei Gesangszeitreihen vor. Für einen kovarianzstationären Prozess muss nach Schlittgen und Streitberg (1997) gelten, dass die Kovarianzfunktion $\gamma(s,t)$ des Prozesses nur von der Entfernung $s-t$ abhängt:

$$\gamma(s,t) = \gamma(s-t) \quad \forall\, s,t \in T.$$

Prozesse, die sowohl mittelwertstationär als auch kovarianzstationär sind, heißen *schwach stationär* und werden im Folgenden vereinfacht als *stationär* bezeichnet.

Leider liegen Musikzeitreihen jedoch keine stationären Prozesse zugrunde. Auch mit den von Dahlhaus (1997) definierten lokal stationären Prozessen können Musikzeitreihen nicht modelliert werden, da es beispielsweise bei Notenwechseln zu extrem abrupten Änderungen der Amplitude und damit der Kovarianz kommen kann. Der Begriff der Stationarität muss für die Modellierung also weiter aufgeweicht werden.

Adak (1998) entwickelte einen Algorithmus zur Segmentierung von Zeitreihen, wie

er für die Transkription zur Unterscheidung von Noten sinnvoll zu sein scheint. *Stückweise lokal stationäre Prozesse* definiert Adak (1998) daher in Anlehnung an Dahlhaus (1997) wie folgt:

> *Definition 1*: Eine Folge von stochastischen Prozessen $X_{t,N}(t = 1, \ldots, N)$ mit Erwartungswert 0 heißt *lokal stationär*, falls man sie darstellen kann in der Form
>
> $$X_{t,N} = \int_{-\frac{1}{2}}^{\frac{1}{2}} A^0_{t,N}(\lambda) e^{i2\pi\lambda t} dZ(\lambda),$$
>
> wobei $Z(\lambda)$ ein „orthogonal increment“ Prozess (Brockwell und Davis, 1991) mit Erwartungswert 0 ist. Weiter müssen Konstanten $K \geq 0$, $c > 0$ und $0.5 < \alpha < 1$ sowie eine stetige Transferfunktion
>
> $$A : [0,1] \times \left[-\frac{1}{2}, \frac{1}{2}\right] \to \mathbb{C} \quad \text{mit} \quad A(u,\lambda) = \overline{A(u,-\lambda)}$$
>
> zur Zeit $0 \leq u \leq 1$ existieren, so dass für alle $N$ gilt:
>
> $$\max_{t:(t/N)\in\varepsilon_N(u)} \sup_{\lambda} \left|A^0_{t,N}(\lambda) - A(u,\lambda)\right| \leq KN^{-\alpha},$$
>
> wobei $\varepsilon_N(u) = [u - cN^{-\alpha}, u + cN^{-\alpha}]$ ein um $u$ zentriertes Intervall ist.

Die durch die Beschränkung des Supremums geforderte Glattheit der Funktion $A$ um die Stelle $u$ garantiert, dass sich der Prozess $X_{t,N}$ an der Stelle $u$ stationär verhält. Gilt dies für alle $u \in [0,1]$, so zeigt der Prozess also an allen einzelnen Stellen $u$ lokal die Eigenschaft eines stationären Prozesses.

Weiter definiert Adak (1998):

> *Definition 2*: Für eine Folge von lokal stationären Prozessen zur Zeit $u$ (siehe Definition 1) ist das zeitabhängige Spektrum gegeben durch
>
> $$f(u,\lambda) = |A(u,\lambda)|^2.$$

*Definition 3*: Eine Folge von stochastischen Prozessen mit Erwartungswert 0 heißt *stückweise lokal stationär*, falls sie gemäß Definition 1 zu allen Zeitpunkten $u \in [0,1]$ lokal stationär ist, abgesehen von endlich vielen Strukturbrüchen.

Weiter kann gezeigt werden, dass die Definition eines lokal stationären Prozesses nach Dahlhaus (1997) die Bedingungen von Definition 1 zu allen Zeitpunkten $u, u \in [0,1]$ erfüllt, falls $A(u,\lambda)$ in $u$ und $\lambda$ differenzierbar ist und die Ableitungen beschränkt sind.

Daher sind die lokal stationären Prozesses nach Dahlhaus (1997) mit den angegebenen zusätzlichen Bedingungen ein Spezialfall der von Adak (1998) definierten stückweise lokal stationären Prozesse.

Die Idee der stückweise lokal stationären Prozesse ist von offensichtlich bedeutender Wichtigkeit in der Musik. Ein von der menschlichen Stimme gesungener Ton ist wegen der Schwankungen in Klang, Tonhöhe und Lautstärke sicherlich keine Realisierung eines stationären Prozesses, kann wohl aber als Realisierung eines lokal stationären Prozesses angenommen werden. Weiterhin kann dann eine Folge von endlich vielen Tönen als Realisierung eines stückweise lokal stationären Prozesses angesehen werden, der bei $n$ Tönen voraussichtlich mindestens $n-1$ Strukturbrüche aufzeigt. Die Anzahl der Strukturbrüche wird zum Beispiel beim nacheinander Singen von Konsonanten und Vokalen bei demselben Ton erhöht.

In Ligges (2000) konnte gezeigt werden, dass der von Adak (1998) vorgestellte Algorithmus für die Musikanalyse und die Online-Analyse von Zeitreihen einige Defizite aufweist.

Wenn angenommen wird, dass ein einzelner Ton in der Zeitreihe einen lokal stationären Anteil eines Prozesses bildet, so können Periodogramme (vgl. Kapitel 2.2.2) von kleinen Teilen dieser Töne geschätzt werden, weil kürzere Stücke der einzelnen Töne damit auch Realisierungen von (lokal) stationären Prozessen sind. Periodogramme kleiner Teile desselben Tons sollten sich nicht wesentlich voneinander

unterscheiden.

Nimmt man weiter an, dass an Stellen, an denen ein Bruch der lokalen Stationarität vorliegt, ein neuer Ton beginnt, so sollten sich Änderungen der Töne durch Änderungen des Periodogramms bemerkbar machen. Einen statistischen Test auf Nichtstationarität haben unter anderem Priestley und Subba Rao (1969) auf der Grundlage des linearen Modells entwickelt. Weitere Tests auf Strukturbrüche in Zeitreihen wurden von Picard (1985) vorgestellt. Leider arbeitet der größte Teil der bekannten Tests zur Erkennung von Strukturbrüchen im Zeitbereich. Im Frequenzbereich ist vorhandene Literatur dünner und häufig an ökonometrische Probleme angepasst.

Eine Alternative zu diesen Segmentierungen bietet die SLEX (Smooth Localized Complex Exponential) Transformation, die sich ähnlich wie Wavelets flexibel an Zeit- und Frequenzauflösung des Problems anpasst. Ombao et al. (2001) verwenden diese Methode zur Segmentierung bivariater nicht-stationärer Zeitreihen in fast stationäre Segmente.

### 2.2.2 Periodogramm und schnelle Fourier Transformation

Die Verwendung von Periodogrammen ist ein zentraler Ansatz zur Erkennung von Frequenzen der Töne in Musikzeitreihen. Das *Periodogramm* $P(\lambda)$ der Zeitreihe $(x_t)_{t=1,\dots,N}$ ist nach Schlittgen und Streitberg (1997) die Fouriertransformierte der empirischen Kovarianzfunktion $(c_\tau)$ an den *Fourierfrequenzen* $\lambda_k = k/N$ mit $1 \leq k \leq \lfloor \frac{N}{2} \rfloor$, $k \in \mathbb{N}$:

$$\begin{aligned} P(\lambda_k) &= \frac{1}{N}\left(\sum_{t=1}^{N}(x_t - \bar{x})\cos 2\pi\lambda_k t\right)^2 + \frac{1}{N}\left(\sum_{t=1}^{N}(x_t - \bar{x})\sin 2\pi\lambda_k t\right)^2 \\ &= c_0 + 2\sum_{\tau=1}^{N-1} c_\tau \cos 2\pi\lambda_k\tau \ . \end{aligned} \tag{2.3}$$

In der Signalverarbeitung wird statt dessen häufiger die folgende Definition des Periodograms verwendet (Brockwell und Davis, 1991):

$$P(\lambda_k) = \frac{1}{N}\left|\sum_{t=1}^{N}(x_t e^{-it\lambda_k})\right|^2 \tag{2.4}$$

Analog zum Periodogramm ist die Spektraldichte als Fouriertransformierte der theoretischen Kovarianzfunktion $(\gamma_\tau)$ definiert. Man beachte, dass das Periodogramm trotzdem kein konsistenter Schätzer für die Spektraldichte $f(\lambda)$ ist, sondern nur Flächen unter der Spektraldichtefunktion durch Flächen unter dem Periodogramm konsistent geschätzt werden können (Schlittgen und Streitberg, 1997).

In Formel (2.3) wird deutlich, dass zur Bestimmung des Periodogramms an allen Fourierfrequenzen $\lambda_k$ eine Laufzeit von $O(N) = N^2$ benötigt wird. Angesichts des Umfangs von Gesangszeitreihen ($N = 661500$ bei einer einminütigen Gesangsdarbietung mit einer Samplingrate von 11025 Hertz) ist eine schnellere Laufzeit wünschenswert.

Mit Hilfe der *schnellen Fourier Transformation* (Fast–Fourier–Transformation, kurz: FFT), die von Cooley und Tukey (1965) eingeführt wurde, kann das Laufzeitverhalten zur Bestimmung des Periodogramms an allen Fourierfrequenzen auf $O(N) = N \log_2 N$ reduziert werden. Um diese Laufzeit zu erreichen, muss die Voraussetzung erfüllt sein, dass $N$ eine Potenz von zwei ist, also $N \in 2^{\mathbb{N}}$. Bis auf einen Faktor ändert sich das Periodogramm nicht, wenn man die Zeitreihe $(x_t)_{t=1,\ldots,N}$ mit Nullen so weit „auffüllt“, dass $N$ eine Potenz von zwei ist (Brockwell und Davis, 1991). Die Voraussetzung schränkt die Benutzung der FFT also nicht ein. Um die höchstmögliche Effizienz der FFT zu erreichen, wird diese im Folgenden stets auf Zeitreihen der Länge $N \in 2^{\mathbb{N}}$ angewendet.

Brockwell und Davis (1991) beschreiben den Algorithmus der FFT aus statistischer Sicht im Zusammenhang mit der Zeitreihentheorie und erläutern auch, wie die oben beschriebene Verbesserung der Laufzeit gegenüber herkömmlichen Verfahren durch geschicktes Faktorisieren erreicht wird. Auch Bloomfield (2000) und Brillinger (1975) geben einen weiten Überblick über die Frequenz- und Fourier-Analyse von

Zeitreihen. Spezieller auf die Signalanalyse und Signalverarbeitung abgestimmt ist Van Trees (2001).

## 2.3 Daten

Als Beispiele für die folgenden Untersuchungen wurden im Tonstudio der Universität Dortmund Aufnahmen in CD Qualität von 17 Sängerinnen und Sängern gemacht.

Alle Sängerinnen und Sänger sangen das Weihnachtslied „Tochter Zion“ von G.F. Händel, von dem eine tiefe Fassung in D-Dur für Alt und Bass (jeweils um eine Oktave versetzt) sowie eine hohe Fassung in F-Dur für Sopran und Tenor vorlag. Für jede Fassung wurde jeweils eine standardisierte Klavierbegleitung über Kopfhörer eingespielt, so dass Tonhöhe und Geschwindigkeit des Stücks durch das Klavier vorgegeben waren, die korrekten Noten also zu jedem Zeitpunkt bekannt waren. Klavierbegleitung und Gesang liegen nach der Aufnahme auf getrennten Spuren (linker und rechter Kanal der Stereoaufnahme) vor.

Unter diesen Sängerinnen und Sängern sind eine professionelle Sopranistin (im Folgenden mit S5 bezeichnet), ein Kammersänger mit Stimmlage Bass sowie einige Amateure (mit Gesangunterricht oder Chorerfahrung) und einige Laien. Damit entstanden Aufnahmen von 4 Bässen, 3 Tenören, 6 Alt-Stimmen und 4 Sopranen, die einen akzeptablen Datensatz darstellen, der die Allgemeinheit eines damit entwickelten Transkriptionsalgorithmus nicht zu sehr einschränkt.

Die so aufgezeichneten Waves wurden für die weiteren Analysen mit dem Paket *tuneR* (vgl. Kapitel 7) in R importiert, bei 16 bit Qualität belassen, jedoch auf der Zeitachse wegen der enormen Datenmenge von 44100 Hertz auf 11025 Hertz Samplingrate reduziert, so dass weiter Frequenzen bis 5512.5 Hertz ohne Aliasbildung im Periodogramm darstellbar sind. Höhere Frequenzen sind bei Gesang nicht zu erwarten (vgl. Seidner und Wendler (1997) und Abschnitt 2.1.2). Da die einzelnen Aufnahmen im Tonstudio auch einzeln ausgesteuert wurden, der absolute Schalldruck also nicht messbar ist, können die Periodogramme ohne Informationsverlust

auf Summe 1 normiert werden.

Aus zwei Gründen wird die Datenbasis weiter reduziert. Zu Beginn des Stücks machen viele der Sängerinnen und Sänger individuelle Fehler, denen ein Transkriptionsalgorithmus folgt. Daher kann besonders dort nicht automatisiert die Fehlerrate des Algorithmus festgestellt werden. Da „Tochter Zion“ eine `ABA` Struktur hat, wird daher ausschließlich der zweite `A` Teil für die weiteren Analysen verwendet, auf den sich die Sängerinnen und Sänger im ersten Teil schon eingesungen haben. Der zweite Grund zur Reduktion der Datenbasis ist das nach wie vor hohe Datenaufkommen. Insbesondere musste wegen der unterschiedlichen Aufnahmestartpunkte leider doch für viele Sängerinnen und Sänger manuell der wahre Anfangszeitpunkt der einzelnen Töne festgelegt werden. Der zweite `A` Teil der Interpretation von Sopranistin S5 ist als Beispieldatensatz im Paket *tuneR* enthalten.

# Kapitel 3

# Transkription

Die Transkription aus der Musikzeitreihe in Noten hat, wie in der Einleitung beschrieben, verschiedene Einsatzgebiete. Hier soll das Verfahren der Transkription aus Wave-Daten beschrieben werden, nicht nur, weil die Daten für Gesang meist immer im Wave-Format vorliegen, sondern auch weil die Transkription aus Wave-Daten wesentlich mehr Probleme aufweist als die Transkription von MIDI-Daten, die bereits Informationen über Tonhöhe und Tonbeginn enthalten.

Die Transkription aus Wave-Daten lässt sich kurz zu folgenden Schritten zusammenfassen:

1. Separierung der interessierenden Stimme (z.B. Gesang) von anderen Klängen (z.B. Begleitung),
2. Grundfrequenzbestimmung auf Fenstern für die ganze Musikzeitreihe,
3. Segmentierung der interessierenden Stimme in Abschnitte, die einzelnen Noten, Stille oder anderem Geräusch entsprechen, und Notenklassifikation für jeden Abschnitt,
4. Quantisierung (Einteilung der Segmente in relative Zeitintervalle zur Festlegung der Notenlängen) und Metrumerkennung (3/4, 4/4 Takt usw.),

5. Tonartbestimmung und

6. Umsetzung in Notenschrift.

In dieser Arbeit werden insbesondere die Schritte 2, 3 und 6 näher untersucht, weil gerade diese Schritte von besonderer Wichtigkeit für die Transkription sind. In Kapitel 4 wird beschrieben, wie diese Schritte in einem Transkriptionsalgorithmus umgesetzt werden können. Dennoch werden in den folgenden Abschnitten alle 6 Teilschritte detailliert beschrieben, um die Transkription in einen globalen Zusammenhang zu stellen. Dazu gehört auch die Nennung der im jeweiligen Zusammenhang besonders relevanten Literatur. In Abschnitt 3.7 werden bereits vorhandene Softwareprodukte zur Transkription kurz vorgestellt.

Für die Beschreibung eines kompletten aber leider nicht erhältlichen Transkriptionsprogramms wird auf Pressing und Lawrence (1993) verwiesen.

In anderen Projekten wird versucht, online bereits bekannte Noten zu verfolgen, die gerade gespielt werden. Cano et al. (1999) benutzen dazu Hidden Markov Models. Raphael (2001) hat ein Expertensystem, basierend auf Bayes Belief Networks, entwickelt, das eine musizierende (oder singende) Person automatisch mit bekannten Noten begleiten soll, also die Noten mitverfolgen muss. Solche Methoden müssen online mitrechnen können, also schnell sein, können dafür jedoch bei Grundfrequenzschätzung und Quantisierung auf die bereits bekannten Noten vertrauen.

## 3.1 Separierung der interessierenden Stimme

Als erster Schritt für die Transkription muss aus dem Gesamtklang der einem zu transkribierenden Instrument entsprechende Teilklang extrahiert werden. Wenn beispielsweise Gesang in Noten umgesetzt werden soll, muss zunächst der Gesang von einer eventuell vorhandenen Klavierbegleitung getrennt werden.

Als Standardverfahren für „Sound Source Separation“ hat sich die von Hyvärinen

et al. (2001) vorgeschlagene *ICA* (Independent Component Analysis) zur Separierung von polyphonem Klang etabliert. Einige Schwächen des Verfahrens zeigt von Ameln (2001) an Beispielen auf.

Klapuri (2001) benutzt das „Spectral Smoothness“ Prinzip zur Separierung und zur polyphonen Grundfrequenzschätzung. Auch Viste und Evangelista (2001) schlagen ein Verfahren zur „Sound Source Separation“ vor, sehen die Anwendung aber vorwiegend für die Audiokodierung in komprimierten Formaten wie MPEG und für den Einsatz in Hörgeräten. Eine Erweiterung dazu wird in Viste und Evangelista (2002) vorgestellt.

## 3.2 Grundfrequenzbestimmung

Die Bestimmung der Grundfrequenz ist der nächste Schritt der Transkription. Aus der Grundfrequenz müssen in späteren Schritten Noten abgeleitet werden. Die Grundfrequenzschätzung aus dem Originalsignal macht allerdings besonders beim Gesang einige Schwierigkeiten. Es werden nämlich nicht nur Vokale gesungen, sondern auch Konsonanten, die sich im Periodogramm meist eher als Rauschen äußern. Gerade stimmlose Konsonanten haben keine zu einer Note korrespondierende Grundfrequenz.

Ein weiteres Problem ist die Verteilung von Grundton und Obertönen (siehe Abschnitt 2.1.2). Gerade großer Oberton-Reichtum, der den Klang der Stimme stark beeinflusst und besonders bei ausgebildeten Stimmen vorkommt, macht die Schätzung der Grundfrequenz schwierig. Abbildung 3.1 zeigt das Periodogramm eines Tons eines Kammersängers (Bass). Insbesondere fällt der kleine, sich vom Rauschen kaum unterscheidende Anteil des Grundtons (GT) im Vergleich zu den vielen Obertönen (OT1 - OT20) auf.

Ein anderes extremes Beispiel wird in Abschnitt 4.1, Abbildung 4.4 gezeigt. Dort ist im Periodogramm fast ausschließlich der erste Oberton zu erkennen. Es kann also weder die Frequenz des Grundtons direkt erkannt werden, noch kann auf die

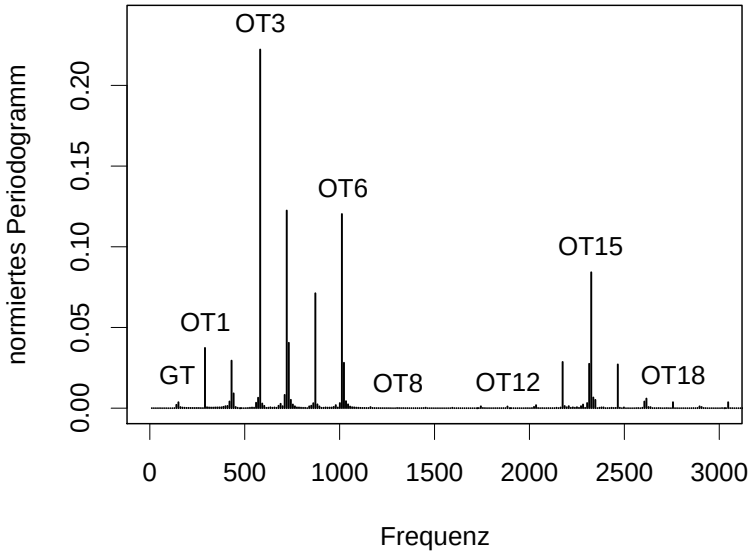


Abbildung 3.1: Periodogramm des Tons d (145Hz) von Bass B7

Grundfrequenz durch die Anordnung von Obertönen zurückgeschlossen werden. Es sind mindestens zwei Obertöne erforderlich, um aus deren Lage auf den Grundton zurückzuschließen. Dazu kann man die Eigenschaft verwenden, dass die Frequenzen der Obertöne ganzzahlige Vielfache der Grundfrequenz sind.

In manchen Situationen kommt es vor, dass die Frequenz von Obertönen leicht verschoben ist, d.h. minimal vom Vielfachen der Grundfrequenz abweicht. Das ist besonders bei Polyphonie ein Problem, da ein Oberton dann keinem Grundton mehr zugeordnet werden kann, wenn dieses Phänomen nicht mitmodelliert wurde. Mit entsprechender Modellierung beschäftigt sich eine neue Arbeit von Godsill und Davy (2005).

Zur Grundfrequenzschätzung, auch oft $f_0$-Schätzung genannt, für monophonen und polyphonen Klang werden in der Literatur verschiedene Ansätze vorgestellt. Dixon (1996) beschreibt ein heuristisches Verfahren zur Notenidentifikation und Klapuri (2001) ein Verfahren zur polyphonen Grundfrequenzschätzung. Smaragdis und Brown (2003) setzen auf die schnelle Fouriertransformation (FFT) eine Non-Negative Matrix Factorization (Brown und Puckette, 1992, 1993) für die polyphone

Transkription auf. Mit Grundfrequenzschätzung von monophonem und polyphonem Klang mit Hilfe von Bayes Verfahren beschäftigen sich Walmsley et al. (1999), Davy und Godsill (2002) und Godsill und Davy (2003). In einer theoretischeren Arbeit stellen Wolfe et al. (2004) bayesianische Variablenselektion zur Spektralschätzung vor.

Polotti und Evangelista (2000) modellieren Phänomene wie das rosa Rauschen mit Hilfe von Wavelet Techniken. In einer Folgearbeit (Polotti und Evangelista, 2001) gehen Sie insbesondere auf die Modellierung von Gesang mit all den „Störgeräuschen“ und Konsonanten ein. Auch einen allgemeineren Artikel zur Wavelet Analyse bei Musikzeitreihen schrieb Evangelista (2001).

Das MAMI Projekt (Musical Audio-Mining, siehe Lesaffre et al. (2003)) hat Software zur Grundfrequenzschätzung entwickelt.

Ein eigener Versuch, Hidden Markov Models zur Schätzung der Grundfrequenz mit anschließender Notenklassifikation zu verwenden (Weihs et al., 2005a), war weniger erfolgversprechend als die Konzentration auf die in den folgenden Kapiteln (4 bis 5) vorgestellten Verfahren, die unter anderem auf den Arbeiten von Rossignol et al. (1999a) sowie Davy und Godsill (2002) aufbauen.

## 3.3 Segmentierung und Notenklassifikation

Auch wenn es zunächst plausibel erscheint, auf einer bereits in einzelne Töne segmentierten Zeitreihe Grundfrequenzbestimmung und Notenklassifikation durchzuführen, hat es sich gezeigt, dass dieser Ansatz in der Praxis bei Gesangsdaten wenig erfolgversprechend ist.

Einen Vorschlag für einen Algorithmus zur Segmentierung von Sprachsignalen macht Adak (1998). Die zeitliche Segmentierung einer Musikzeitreihe untersuchen aber auch Rossignol et al. (1999b). Eine Alternative zu diesen Segmentierungen bietet die SLEX (Smooth Localized Complex Exponential) Transformation, die von Om-

bao et al. (2001) zur Segmentierung bivariater nicht-stationärer Zeitreihen in fast stationäre Segmente verwendet wird. In Abschnitt 2.2.1 wird beschrieben, dass eine Musikzeitreihe als stückweise lokal stationärer Prozess, d.h. als lokal stationärer Prozess mit endlich vielen Bruchpunkten, modelliert werden kann und wie eine Segmentierung in lokal stationäre Anteile erfolgen könnte.

Bei den meisten Segmentierungsverfahren für Musikzeitreihen wird die Amplitude berücksichtigt. Bei Schlag- und Zupfinstrumenten (Schlagzeug, Trommel, Gitarre, Klavier, usw.) ist nämlich bei jedem neuen Ton ein plötzlicher starker Anstieg der Amplitude zu erkennen. Ein neues Segment beginnt bei diesen Verfahren also dort, wo die Amplitude einen plötzlichen signifikanten Anstieg erfährt.

Das Vorgehen in dieser Arbeit unterscheidet sich von den oben genannten Methoden etwas in der Reihenfolge der Vorgehensweise, denn bei Gesang und auch anderen Instrumenten, z.B. Streichinstrumenten, ist bei einem Tonwechsel nicht immer ein Amplitudenanstieg gegeben, so dass ein anderes Segmentierungsverfahren gefunden werden muss. In Kapitel 4 wird beschrieben, dass direkt nach der Grundfrequenzbestimmung (siehe auch Abschnitt 3.2) die Notenklassifikation für jedes einzelne Fenster durchgeführt wird. Damit erhält man eine Zeitreihe aus klassifizierten Noten zu äquidistanten Punkten gemäß der Breite der Verschiebung bei der Fensterung. Nachdem eine solche Zeitreihe geglättet ist, wird an den Stellen, an denen sich der Notenwert ändert, die Segmentierung vorgenommen.

## 3.4 Quantisierung und Metrumerkennung

Nach der Segmentierung wird bei der Quantisierung die relative Länge von Noten geschätzt. Müllensiefen und Frieler (2004) definieren quantisierte Melodien als „[...] melodies where the durations are integer multiples of a smallest time unit $T$“. Es wird also nach der Länge einer kürzesten Note gesucht, deren Länge größter gemeinsamer Teiler aller in dem Musikstück enthaltenen Notenlängen ist. Kommen in einem Musikstück Achtelnoten (Länge $\frac{1}{8}$, punktierte Achtelnoten ($\frac{1}{8} + \frac{1}{16}$), Viertel-

noten und halbe Noten vor, so ist die gesuchte Notenlänge $\frac{1}{16}$.

Die relative Notenlänge kann aus der absoluten Länge der Segmente aus Abschnitt 3.3 geschätzt werden. Da bis zu diesem Schritt bereits mehrere Verarbeitungsschritte in der Transkription erledigt wurden, die zumeist selbst aus Schätzungen bestehen, muss mit Ungenauigkeiten gerechnet werden. Die größte Ungenauigkeit wird hier allerdings von dem die Musik produzierenden Menschen eingebracht, sowohl beabsichtigt als auch unbeabsichtigt. Unbeabsichtigt werden lange Töne oftmals zu kurz von Sängern gehalten und Anfangsnoten zu spät begonnen, etwa beim Atmen. Beran (2004) analysiert beispielsweise die (beabsichtigten) Tempovariationen bei Pianisten. Neben nicht exakt eingehaltenen Notenlängen muss also auch immer mit Tempoverschärfungen und -verlangsamungen innerhalb eines Stückes gerechnet werden.

Ein Quantisierungsverfahren muss also sehr robust mit Ungenauigkeiten umgehen können und idealerweise auch bei sich dynamisch änderndem Tempo unterschiedlich lange absolute Segmente der gleichen (relativen) Notenlänge zuordnen. Gelingt dies dem Quantisierungsverfahren nicht, ist letztendlich ein fast unlesbares Notenbild das Ergebnis, etwa mit $\frac{1}{128}$ Noten oder mit völlig verschobenen Taktanfängen und Notenlängen.

Die meisten Arbeiten, die sich mit Quantisierung beschäftigen, basieren auf Segmentierungsverfahren, die die Segmentierung durch Erkennung von starken Amplitudenanstiegen erreichen (siehe Abschnitt 3.3). Ein solches Verfahren zur Quantisierung stellen Cemgil et al. (2000) vor, das später erweitert wird (Cemgil et al., 2001), um dynamisch sich änderndes Tempo verfolgen zu können. Als Alternative dazu werden von Cemgil und Kappen (2003) Monte Carlo Methoden vorgestellt. Auch Davies und Plumbley (2004) versuchen die Quantisierung an sich dynamisch änderndes Tempo anzupassen.

Auf dynamische Quantisierung kann allerdings dann verzichtet werden, wenn die Musikzeitreihe mit Hilfe einer Tempovorgabe, z.B. unterstützt durch ein Metronom, aufgezeichnet wurde. Auch für die in dieser Arbeit verwendeten Beispieldatensätze

wurde das Tempo für die Sängerinnen und Sänger fest durch die eingespielte Klavierbegleitung vorgegeben. Daher wird hier ein einfaches statisches Quantisierungverfahren verwendet, das die segmentierten Teilstücke zunächst in Achtelnoten (als kürzeste Einheit) zusammenfasst und anschließend mehrere Achtelnoten derselben Notenhöhe zu längeren Noten gruppiert.

Nachdem durch die Quantisierung relative Notenlängen für alle Segmente gefunden wurden, muss nun noch die Metrumerkennung erfolgen, um Metrum und Taktstriche für das Notenbild zu ermitteln. Alle bekannten Verfahren zur Metrumerkennung basieren auf der Auswertung der Amplitudenstärken, um die Akzentuierung im fertig quantisierten Musikstück erkennen zu können. Eine typische Idee ist die Betrachtung der Abstände zwischen Akzenten, d.h. kurzen Zeitintervallen mit größerer Amplitude. Eine Schätzung ohne Vorwissen mit akzeptabler Genauigkeit ist hier jedoch nicht in Sicht, fällt es doch schon dem Menschen oft schwer $\frac{2}{4}$, $\frac{4}{4}$ und $\frac{4}{8}$ zu unterscheiden. Im Folgenden wird das Metrum als gegeben angenommen.

Für Transkriptionsverfahren ist es also sehr sinnvoll, als Vorgaben die Länge der kürzesten Note und das Metrum zu verlangen. Das gilt insbesondere bei der Transkription von Musikzeitreihen, die nicht durch Schlag- oder Zupfinstrumente erzeugt wurden.

## 3.5 Tonartbestimmung

Die Bestimmung der Tonart eines Musikstücks ist ein letzter Schritt vor der Umsetzung zur Notenschrift. Tonartbestimmung kann natürlich nur gelingen, wenn die Tonart auch klar definiert ist, also westliche und nicht allzu moderne Musik zugrunde liegt. Die Idee ist, alle im Stück enthaltenen Noten in einer Tabelle oder auch Kontingenztafel zu halten und über die Häufigkeiten die danach wahrscheinlichste Tonart zu bestimmen. Weiterführend könnte man Folgen von Tönen oder erklingende Akkorde untersuchen. Entsprechende Arbeiten zur Tonartschätzung haben beispielsweise Brown et al. (1994) vorgestellt.

## 3.6 Umsetzung in Notenschrift

In den vorangegangenen Abschnitten wurde beschrieben, wie nahezu alle Eigenschaften (abgesehen von Dynamik) bestimmt oder geschätzt werden können, die zum Umsetzen des Klangs in Notenschrift benötigt werden. Letztendlich folgt dann die Umsetzung in Notenschrift. Das ist ein Problem des Notensatzes, da die Eigenschaften der Note bereits determiniert worden sind. Als freies Notensatzsystem ist LilyPond (Nienhuys et al., 2005) besonders zu empfehlen. Mit Hilfe eines in dem R Paket *tuneR* (siehe Kapitel 7) enthaltenen Interface (Preusser et al., 2002) kann dies auch aus der von uns verwendeten statistischen Programmierumgebung geschehen. LilyPond basiert auf dem Textsatzsystem LaTeX (Lamport, 1994), das auf TeX (Knuth, 1984) aufbaut.

In der Implementierung des in Kapitel 4 beschriebenen Algorithmus werden Töne mit der Minimallänge der Quantisierervorgabe erst im Schritt der Umsetzung in Notenschrift zu längeren Noten zusammengefasst.

Neben der Ausgabe von Notenschrift kann LilyPond auch MIDI–Dateien erzeugen, so dass das resultierende Ergebnis nicht nur visuell, sondern auch per Gehör kontrolliert werden kann. Mit dem MIDI Format lässt sich das Ergebnis damit auch für andere Programmpakete lesbar machen, falls der Notensatz mit LilyPond als zu wenig benutzerfreundlich erscheint.

## 3.7 Softwareprodukte zur Transkription

Es gibt bereits einige Softwareprodukte, die laut Werbebeschreibung mindestens die Verfolgung der Grundfrequenz oder sogar eine vollständige Transkription leisten können. Unter anderem wurden die folgenden Produkte, zu denen es Test- oder Demoversionen gibt, auf den vorliegenden Gesangsdaten der professionellen Sopranistin S5 (siehe Abschnitt 2.3) getestet:

- AmazingMidi (http://www.pluto.dti.ne.jp/~araki/amazingmidi),
- Akoff Music Composer (http://www.akoff.com),
- Audio to score (logic) (http://www.emagic.de),
- Autotune (http://www.antarestech.com),
- DigitalEar (http://www.digital-ear.com),
- Melodyne (http://www.celemony.com),
- IntelliScore (http://www.intelliscore.net) und
- Widi (http://www.widisoft.com).

Alle Produkte lieferten zum Teil völlig unterschiedliche, aber durchweg sehr schlechte Ergebnisse. Fehlerraten wurden wegen des Aufwands nicht bestimmt, denn die Demoversionen erlaubten zum Teil keinen Daten- oder Grafikexport. Es war jedoch offensichtlich, dass keines der Produkte eine Fehlerrate von weniger als 50% liefern konnte. Einige Transkriptionen erinnerten dabei an Jazzstücke (fehlende Robustheit gegen Vibrato), wobei einige wenige Teile der Tonhöhe korrekt erkannt wurden, während bei anderen Produkten gar kein Zusammenhang zum eingegebenen Musikstück (falsche Erkennung von Grundfrequenz und Tonanfängen) zu erkennen war.

Insgesamt muss man annehmen, dass die Produkte für ganz andere Klänge, etwa Klavierklang, optimiert wurden. Melodyne war das beste Produkt für den verwendeten Datensatz, denn es zeigten sich die deutlichsten Zusammenhänge zu der Originalreihe. Die Abbildungen 3.2 und 3.3 zeigen Transkriptionsergebnisse der Produkte Widi und Melodyne auf den ersten 4 Takten der Interpretation von Sopranistin S5.

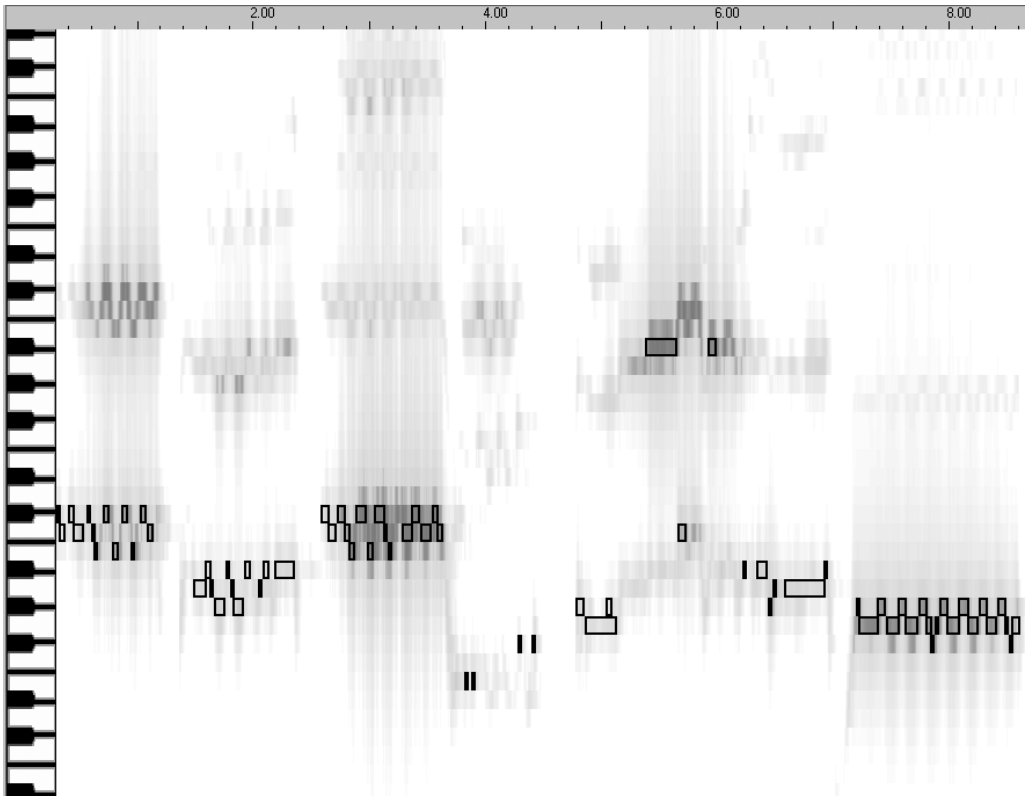


Abbildung 3.2: Transkription mit Widi, erkannte Noten sind schwarz umrandet

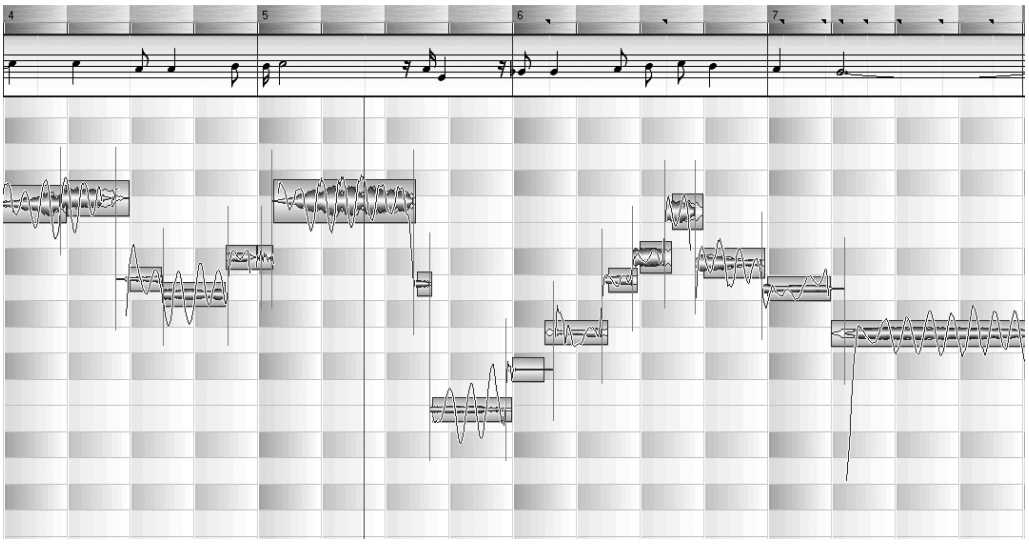


Abbildung 3.3: Transkription mit Melodyne

# Kapitel 4

# Transkription mit Hilfe einer Heuristik

Ein Algorithmus, der die in Kapitel 3 beschriebenen Einzelschritte der Transkription umsetzt, wird in Abschnitt 4.1 vorgestellt. Dabei liegt der Fokus auf der möglichst fehlerfreien Schätzung der Grundfrequenz, aus der die jeweilige Note bestimmt wird. Insbesondere findet in dem hier vorgestellten Algorithmus auch die Erkennung von neuen Tonanfängen über die Tonhöhe statt, denn die üblichen Verfahren, die über starke Änderungen der Amplitude Tonanfänge detektieren, sind für Gesang – im Gegensatz zu Schlag- oder Zupfinstrumenten – nicht geeignet (siehe Kapitel 3). Für die Schätzung der Grundfrequenz wird hier eine Weiterentwicklung der Heuristik verwendet, deren Vorläufer in Ligges et al. (2002) beschrieben wurde. Dieser Vorläufer war im Wesentliche nur auf eine geeignete Segmentierung ausgerichtet und verwendete eine schlechtere Interpolation für die Grundfrequenzschätzung (siehe Abschnitt 4.1.3).

In Abschnitt 4.2 wird gezeigt, wie der Algorithmus durch Training verbessert werden kann. Dabei werden die Parameter der Heuristik für die Klangcharakteristiken der jeweiligen Stimme angepasst.

## 4.1 Automatische Transkription mit Hilfe einer heuristischen Grundfrequenzschätzung

In diesem Abschnitt wird ein Algorithmus zur automatischen Transkription von Musikzeitreihen anhand von den in Abschnitt 2.3 beschriebenen Beispielen vorgestellt. Dabei werden insbesondere Umsetzungen für die in Kapitel 3 beschriebenen Schritte 2, 3 und 6 im Detail gezeigt.

Die Informationen der anderen in Kapitel 3 beschriebenen Schritte (1, 4 und 5) werden im Wesentlichen als gegeben angesehen. Die Separierung (Schritt 1, Abschnitt 3.1) des Gesamtklangs der Musikzeitreihe in Einzelklänge ist nicht notwendig, da Gesang und Klavierbegleitung in den hier verwendeten Daten bereits auf getrennten Kanälen aufgezeichnet wurden.

Die Quantisierung (Schritt 4, Abschnitt 3.4) erfolgt statisch mit der Vorabinformation, dass die minimale Notenlänge $\frac{1}{8}$ beträgt. Als Metrum wird ein $\frac{4}{4}$ Takt vorgegeben und auch die jeweilige Tonart (Schritt 5, Abschnitt 3.5) wird mit F-Dur (hohe Stimmen, Sopran und Tenor) oder D-Dur (tiefe Stimmen, Alt und Bass) als vorab bekannt vorausgesetzt.

### 4.1.1 Fensterung und diskrete Short Time Fast Fourier Transformation

In Abschnitt 2.2.1 wurde beschrieben, dass man Prozesse, die bei Musikzeitreihen zugrunde liegen, als höchstens stückweise lokal stationär ansehen kann. Weil es jedoch nur Sinn macht, Periodogramme (siehe Abschnitt 2.2.2) von Zeitreihen zu berechnen, denen ein stationärer Prozess zugrunde liegt, verwendet man in der Praxis in der Regel eine *Short Time Fast Fourier Transformation* zur Berechnung der Periodogramme.

Dabei wird die Zeitreihe auf einem Fenster der Breite $w$ mit $o$ überlappenden Be-

obachtungen betrachtet, wobei als Fensterfunktion die Indikatorfunktion verwendet wird, also keine Herabgewichtung an den Rändern erfolgt. Auf diesem Fenster wird jeweils eine (diskrete) schnelle Fourier Transformation berechnet. Für die vorliegenden Beispiele (siehe Abschnitt 2.3), nach Datenreduktion mit einer Samplingrate von 11025 Hertz, empfiehlt sich eine Fensterbreite $w$ von 512 Beobachtungen als Kompromiss zwischen zeitlicher Genauigkeit und Genauigkeit der Frequenzauflösung.

Wenn ein schnell gesungener oder gestoßener Ton nicht kürzer als $\frac{512}{11025} \approx \frac{1}{20}$ Sekunde (zeitliche Genauigkeit) ist, kann das Periodogramm an Fourier Frequenzen im Abstand von $\frac{11025}{512}$ Hertz (Genauigkeit der Frequenzauflösung), also an den Stellen $21.53, 43.07, \ldots, 5512.50$ Hertz, berechnet werden. Sollten die Tonfolgen noch schneller sein, können nur noch weniger Beobachtungen zur Berechnung der Periodogramme verwendet werden. Tabelle 2.1 in Abschnitt 2.1.2 zeigt, dass der tiefste Ton eines Basses das $D$ mit etwa 73 Hertz ist. Ein halber Ton höher ($Dis$) hat bei so niedrigen Frequenzen dann mit $73 \cdot 2^{1/12} \approx 77.34$ einen Abstand von nur 4.34 Hertz. Damit ist die Auflösung des so berechneten Periodogramms, insbesondere bei tiefen Frequenzen, zunächst nicht ausreichend. Eine Lösungsmöglichkeit für eine genaue Grundfrequenzbestimmung mit Hilfe dieser Periodogramme wird im nächsten Abschnitt vorgestellt.

Als Überlappungsbreite $o$ der Fensterung hat sich in Vorversuchen eine halbe Fensterbreite $o = \frac{w}{2} = \frac{512}{2} = 256$ als geeignet gezeigt, so dass man $\frac{11025}{256} \approx 43$ Periodogramme pro Sekunde erhält und damit zeitlich auch schnelle Tonfolgen genau genug darstellen kann. Die einzelnen durch Fensterung betrachteten Abschnitte $X_{w,o}(\vec{x}, i)$ der Zeitreihe $x_t, t \in \{1, \ldots, T\}$ lassen sich also darstellen als:

$$X_{w,o}(\vec{x}, i) = \{x_{i \cdot o+1}, \ldots, x_{i \cdot o+w}\} \quad \text{mit} \quad i \in \left\{0, 1, \ldots, \left\lfloor \frac{T-w}{o} \right\rfloor\right\}.$$

Die zugehörigen normierten Periodogramme $P_i(\lambda) := P(\lambda, X_{w,o}(\vec{x}, i))$ werden bei einer Samplingrate von $s$ Hertz an den Fourier Frequenzen $\lambda \in \left\{\frac{1s}{w}, \frac{2s}{w}, \ldots, \frac{(w/2)s}{w}\right\}$ berechnet. Die Periodogramme werden dabei so normiert, dass $\sum_\lambda P_i(\lambda) = 1$.

### 4.1.2 Grundfrequenzschätzung

Die Grundfrequenzschätzung wird als nächster Schritt auf der im vorigen Abschnitt 4.1.1 beschriebenen Fensterung durchgeführt. Es ist bereits deutlich gemacht worden, dass es nicht reicht, sich ausschließlich auf die Fourier Frequenz des stärksten Peaks im Periodogramm zu konzentrieren, da dieser zu einer Frequenz eines Obertons gehören könnte. Selbst bei der Frequenz des Grundtons reicht die Auflösung des Periodogramms nicht notwendigerweise aus, um zwischen zwei verschiedenen Halbtönen zu unterscheiden (siehe Abschnitt 4.1.1: 4.34 Hertz zwischen zwei Halbtönen, aber 21.53 Hertz zwischen zwei Fourier Frequenzen).

Für jeden Abschnitt wird daher aus dem Periodogramm $P_i(\lambda) = P(\lambda, X_{w,o}(\vec{x}, i))$ die Frequenz $f_{0,i}$ des Grundtons (siehe Abschnitt 2.1.2) des vorliegenden Klangs geschätzt. Dazu könnten auch die in Abschnitt 3.2 beschriebenen Verfahren genutzt werden, die sich in Versuchen mit den vorliegenden Gesangsdaten jedoch als weniger geeignet erwiesen haben. Statt dessen wird hier eine auf Ligges et al. (2002) basierende, später erweiterte und optimierte Heuristik vorgeschlagen.

Die Schätzung der Grundfrequenz $f_{0,i}$ erfolgt in jedem Abschnitt der Fensterung nach demselben Muster. Die Schätzung ist unabhängig von anderen, auch von benachbarten Abschnitten. Daher wird im Folgenden mit leicht vereinfachter Notation der Schätzer $f_{\text{heur}}$ zur Schätzung der Grundfrequenz $f_0$ eines beliebigen festen Abschnitts mit zugehörigem Periodogramm $P(\lambda)$ betrachtet.

Der Algorithmus zur Berechnung der Heuristik $f_{\text{heur}}$ arbeitet wie folgt:

1. Schließe die ersten $q$ Fourier Frequenzen des Periodogramms von allen folgenden Betrachtungen aus. Gerade in den tiefsten Frequenzen erhält man häufig Störungen durch Rauschen, die unerwünscht und uninteressant sind (siehe Abschnitt 2.1.5). Für die vorliegenden Musikzeitreihen hat sich ein Wert von $q = 10$ als geeignet erwiesen. Für tiefe Musikstücke im Bass muss dieser Wert allerdings gesenkt werden, damit die tiefen Frequenzen noch repräsentiert werden.

2. Suche *interessante Peaks* im Periodogramm. Das ist die Menge $\tilde{\Lambda}$ aller Frequenzen $\tilde{\lambda}$, deren Periodogrammwert größer ist als der $h$-te Teil ($0 \leq h \leq 1$) des höchsten Peaks des Periodogramms:

$$\tilde{\Lambda} = \left\{ \tilde{\lambda} : P(\tilde{\lambda}) > h \cdot \max_{\lambda}(P(\lambda)) \right\}. \tag{4.1}$$

   Als oftmals geeignete Einstellung hat sich $h = 0.015$ gezeigt.

3. Als nächstes betrachte den ersten interessanten Peak mit minimaler Frequenz. Ist dies die Frequenz des Grundtons, so wird kein weiterer noch interessanterer Peak bis zum doppelten seiner Frequenz (am ersten Oberton) vorhanden sein. Sollte dort doch ein größerer Peak vorliegen, so wird der erste Peak durch Rauschen hervorgerufen worden sein und statt dessen wird der größere Peak weiter verwendet. Formal notiert wird die Frequenz

$$\lambda^* = \text{argmax}_{\min(\tilde{\lambda}) \leq \lambda \leq m \cdot \min(\tilde{\lambda})} P(\lambda) \quad \text{mit} \quad \lambda, \tilde{\lambda} \in \tilde{\Lambda} \tag{4.2}$$

   gesucht, wobei der Faktor $m$ knapp unter der Verdoppelung (Frequenz des ersten Obertons) liegen sollte und sich mit $m = 1.8$ als günstig erwiesen hat.

4. Ermittele, ob nicht doch eine zu hohe Frequenz $\lambda^*$ gefunden wurde. Falls ein $\lambda^\circ$ aus der Menge $\tilde{\Lambda}$ existiert mit $l \cdot \lambda^* < \lambda^\circ < r \cdot \lambda^*$, so wird das für geeignete Werte von $l$ (1.35) und $r$ (1.65) gefundene $\lambda^\circ$ die Frequenz eines zweiten Obertons sein und $\lambda^*$ die Frequenz eines ersten Obertons. In dem Fall ersetze $\lambda^* := \lambda^*/2$

5. Nun definiere diejenige direkt benachbarte Fourier Frequenz $\lambda^{**}$ von $\lambda^*$ mit größtem Wert des Periodogramms:

$$\lambda^{**} := \text{argmax}_{\lambda \in \left\{\lambda^* - \frac{1}{w}, \lambda^* + \frac{1}{w}\right\}} P(\lambda). \tag{4.3}$$

   Als Schätzung für die Grundfrequenz wird dann die folgende Interpolation der beiden stärksten Grundfrequenzen, die im folgenden Abschnitt 4.1.3 motiviert wird, vorgeschlagen:

$$f_{\text{heur}} := \lambda^* + \frac{\lambda^{**} - \lambda^*}{2} \cdot \sqrt{\frac{P(\lambda^{**})}{P(\lambda^*)}} \quad . \tag{4.4}$$

Für die ersten 16 Abschnitte der Fensterung des zweiten `A` Teils von „Tochter Zion“, gesungen von Sopranistin S5, erhält man als Schätzungen für die zugehörigen Grundfrequenzen (in Hertz) mit den oben angegebenen Einstellungen:

```
  78.50  216.44   91.87   95.14  500.75  499.87 1046.44 1043.81
1050.90 1053.58 1054.09 1042.02  511.66  513.60  523.24  532.41
```

### 4.1.3 Interpolation für die Grundfrequenzschätzung

Die in Abschnitt 4.1.2 vorgeschlagene Interpolationsformel (4.4),

$$f_{\text{heur}} := \lambda^* + \frac{\lambda^{**} - \lambda^*}{2} \cdot \sqrt{\frac{P(\lambda^{**})}{P(\lambda^*)}},$$

der zwei Fourier Frequenzen $\lambda^*$ (die Frequenz mit dem größerem Periodogrammwert in der Nähe der Grundfrequenz) und $\lambda^{**}$ (die Frequenz mit dem kleineren Periodogrammwert) soll hier motiviert und empirisch auf ihre Genauigkeit hin untersucht werden.

Idee ist es, im Periodogramm die Werte der Fourier Frequenzen desjenigen Peaks zu interpolieren, der durch die Grundfrequenz eines Tons verursacht ist. Dazu wird ein quadratisches Modell

$$\tilde{P}(\lambda) = -a(\lambda - \lambda_{\max})^2 + \tilde{P}(\lambda_{\max}) \tag{4.5}$$

angenommen, das zwischen der Fourier Frequenz $\lambda^*$ des größten beteiligten Wertes $P(\lambda^*) = \tilde{P}(\lambda^*)$ des Periodogramms in der Nähe der Grundfrequenz und den beiden benachbarten Fourier Frequenzen interpoliert. Die benachbarte Fourier Frequenz mit zugehörigem nächst großen Periodogrammwert $P(\lambda^{**}) = \tilde{P}(\lambda^{**})$ ist $\lambda^{**}$ und die benachbarte Fourier Frequenz mit kleinerem Periodogrammwert $P(\lambda^{***}) = \tilde{P}(\lambda^{***})$ ist $\lambda^{***}$. Somit ist entweder $\lambda^{***} < \lambda^* < \lambda^{**}$ oder $\lambda^{**} < \lambda^* < \lambda^{***}$ und es ist $|\lambda^{**} - \lambda^*| = |\lambda^{***} - \lambda^*|$. Die Skizze Abbildung 4.1 veranschaulicht das Modell, aus dem $\lambda_{\max}$ als ein Punktschätzer für die zugrunde liegende wahre Frequenz hervorgeht.

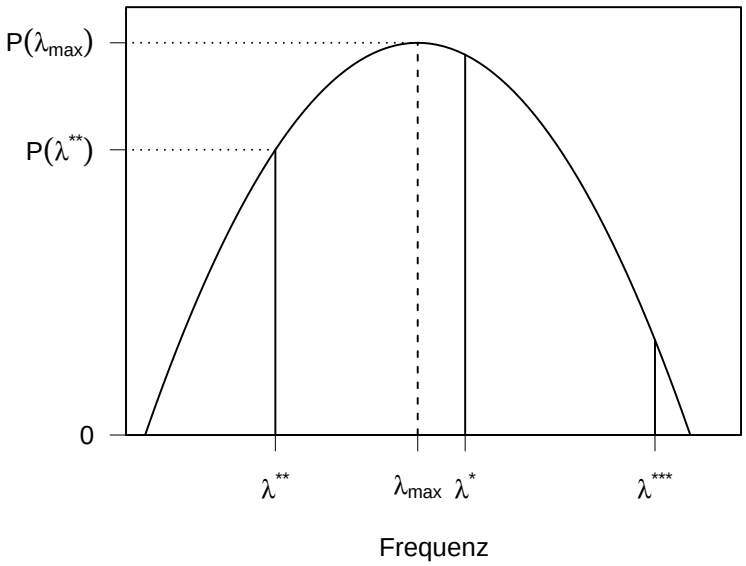


Abbildung 4.1: Skizze zur Interpolation im Periodogramm

Unter den Modellannahmen ergibt sich aus (4.5) das Gleichungssystem

$$\tilde{P}(\lambda^*) = -a(\lambda^* - \lambda_{\max})^2 + \tilde{P}(\lambda_{\max}) \tag{4.6}$$

$$\tilde{P}(\lambda^{**}) = -a(\lambda^{**} - \lambda_{\max})^2 + \tilde{P}(\lambda_{\max}) \tag{4.7}$$

$$\tilde{P}(\lambda^{***}) = -a(\lambda^{***} - \lambda_{\max})^2 + \tilde{P}(\lambda_{\max}) \quad . \tag{4.8}$$

Nach Gleichsetzen von $\tilde{P}(\lambda_{\max})$ aus Formeln (4.6) und (4.7) sowie (4.6) und (4.8) erhält man durch elementare Umformungen zunächst $\lambda_{\max}$:

$$\tilde{P}(\lambda^{**}) + a(\lambda^{**} - \lambda_{\max})^2 = \tilde{P}(\lambda^*) + a(\lambda^* - \lambda_{\max})^2$$

$$\Leftrightarrow \quad \lambda_{\max} = \frac{\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^*)}{2a(\lambda^{**} - \lambda^*)} + \frac{\lambda^{**} + \lambda^*}{2} \tag{4.9}$$

$$\text{und analog auch} \quad \lambda_{\max} = \frac{\tilde{P}(\lambda^{***}) - \tilde{P}(\lambda^*)}{2a(\lambda^{***} - \lambda^*)} + \frac{\lambda^{***} + \lambda^*}{2} \quad . \tag{4.10}$$

Erneutes Gleichsetzen von (4.9) und (4.10) liefert $a$:

$$\frac{\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*})}{2a(\lambda^{**} - \lambda^{*})} + \frac{\lambda^{**} + \lambda^{*}}{2} = \frac{\tilde{P}(\lambda^{***}) - \tilde{P}(\lambda^{*})}{2a(\lambda^{***} - \lambda^{*})} + \frac{\lambda^{***} + \lambda^{*}}{2}$$

$$\begin{aligned} \Leftrightarrow \quad & a(\lambda^{**} - \lambda^{***})(\lambda^{**} - \lambda^{*})(\lambda^{***} - \lambda^{*}) = \\ & (\tilde{P}(\lambda^{***}) - \tilde{P}(\lambda^{*}))(\lambda^{**} - \lambda^{*}) - (\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*}))(\lambda^{***} - \lambda^{*}) \end{aligned}$$

$$\Leftrightarrow \quad a = \frac{\dfrac{\tilde{P}(\lambda^{***}) - \tilde{P}(\lambda^{*})}{\lambda^{***} - \lambda^{*}} - \dfrac{\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*})}{\lambda^{**} - \lambda^{*}}}{\lambda^{**} - \lambda^{***}} \quad . \tag{4.11}$$

Weil $\lambda^{*}$, $\lambda^{**}$ und $\lambda^{***}$ benachbarte Fourier Frequenzen sind, ist

$$\begin{aligned} & \lambda^{***} - \lambda^{*} = -(\lambda^{**} - \lambda^{*}) \\ \text{und} \quad & \lambda^{**} - \lambda^{***} = \lambda^{**} - \lambda^{*} + \lambda^{*} - \lambda^{***} = 2(\lambda^{**} - \lambda^{*}) \quad . \end{aligned}$$

Einsetzen in (4.11) vereinfacht $a$ dann wie folgt:

$$\begin{aligned} a &= \frac{\dfrac{\tilde{P}(\lambda^{***}) - \tilde{P}(\lambda^{*})}{-(\lambda^{**} - \lambda^{*})} - \dfrac{\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*})}{\lambda^{**} - \lambda^{*}}}{2(\lambda^{**} - \lambda^{*})} \\ &= \frac{2\tilde{P}(\lambda^{*}) - \tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{***})}{2(\lambda^{**} - \lambda^{*})^{2}} \quad . \end{aligned} \tag{4.12}$$

Aus (4.12) und (4.9) ergibt sich dann als eine Interpolation gemäß Modell (4.5):

$$\begin{aligned} \lambda_{\max} &= \frac{\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*})}{\dfrac{2\tilde{P}(\lambda^{*}) - \tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{***})}{\lambda^{**} - \lambda^{*}}} + \frac{\lambda^{**} + \lambda^{*}}{2} \\ &= \frac{\lambda^{**} + \lambda^{*}}{2} + \frac{(\lambda^{**} - \lambda^{*})(\tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{*}))}{2\tilde{P}(\lambda^{*}) - \tilde{P}(\lambda^{**}) - \tilde{P}(\lambda^{***})} \quad . \end{aligned} \tag{4.13}$$

Zur Überprüfung der Schätzgenauigkeit wurden reine Sinusschwingungen (11025 Hertz, 16 Bit) simuliert mit den 117 Frequenzen $f_j = 440 \cdot 2^{j/12}$ Hertz mit

$j \in \{-31, -30.5, -30, \ldots, 26.5, 27\}$. Das sind alle Halbtöne von $D$ (73.4 Hertz, tiefster Ton Bass) bis $c''''$ (2093 Hertz, weit höher als die höchsten für Soprane singbaren Grundfrequenzen) bei 440 Hertz Stimmung sowie jeweils eine Frequenz genau zwischen den Halbtönen (siehe dazu auch Abschnitte 2.1.2 und 2.1.3).

Für diese 117 erzeugten Schwingungen ist die Abweichung der Interpolation mit Formel (4.13) im Mittel zwar etwa Null, es zeigt sich allerdings eine für das zugrunde liegende Problem inakzeptabel große Standardabweichung von 3.7 Hertz.

Gallant et al. (1974) schlagen zur Verbesserung eine Transformation $g(\alpha, \tilde{P}(\lambda)) := (\tilde{P}(\lambda))^\alpha$ mit $\alpha = \frac{1}{3}$ oder $\alpha = \frac{1}{4}$ vor, so dass sich als neuer Schätzer

$$\lambda_{\max,\alpha} = \frac{\lambda^{**} + \lambda^*}{2} + \frac{(\lambda^{**} - \lambda^*)((\tilde{P}(\lambda^{**}))^\alpha - (\tilde{P}(\lambda^*))^\alpha)}{2(\tilde{P}(\lambda^*))^\alpha - (\tilde{P}(\lambda^{**}))^\alpha - (\tilde{P}(\lambda^{***}))^\alpha} \tag{4.14}$$

ergibt. Für die simulierten Schwingungen führte der Wert $\alpha = \frac{1}{9}$ zu einer Standardabweichung von 1.5 Hertz, die kleiner ist als die Standardabweichung der Schätzungen, die mit den von Gallant et al. (1974) vorgeschlagenen $\alpha$ Werten durchgeführt wurden.

Eine weitere Verbesserung der Schätzergebnisse konnte durch leichte Abänderung und Vereinfachung von Formel (4.14) erreicht werden, indem im Nenner der Anteil $-(\tilde{P}(\lambda^{**}))^\alpha - (\tilde{P}(\lambda^{***}))^\alpha$ als vernachlässigbar klein gegenüber $2(\tilde{P}(\lambda^*))^\alpha$ angenommen wird. Dabei hat der Schätzer

$$\begin{aligned} f_{\text{heur},\alpha} &= \frac{\lambda^{**} + \lambda^*}{2} + \frac{(\lambda^{**} - \lambda^*)}{2}\left(\frac{(\tilde{P}(\lambda^{**}))^\alpha}{(\tilde{P}(\lambda^*))^\alpha} - 1\right) \\ &= \lambda^* + \frac{\lambda^{**} - \lambda^*}{2}\left(\frac{\tilde{P}(\lambda^{**})}{\tilde{P}(\lambda^*)}\right)^\alpha \end{aligned} \tag{4.15}$$

mit $\alpha = \frac{1}{2}$ sehr gute Ergebnisse gezeigt. Für die mit der oben erwähnten Simulation erzeugten 117 Schwingungen ist die Abweichung der Schätzung von der wahren Frequenz nie größer als 2.23 Hertz:

$$\max_{j \in \{-31, -30.5, \ldots, 26.5, 27\}} |f_j - f_{\text{heur},\alpha=\frac{1}{2},j}| < 2.23.$$

Die Abweichung hat einen Mittelwert von fast 0 und eine Standardabweichung von 1.4. Bei 115 von den 117 Schätzungen ist sie sogar kleiner als 2. Diese Genauigkeit sollte ausreichen, selbst zwischen sehr tiefen Halbtönen mit nur 4.34 Hertz Abstand zu unterscheiden. Die in Abschnitt 4.1.2 vorgeschlagene Interpolationsformel (4.4) folgt direkt aus Formel (4.15) mit $\alpha = \frac{1}{2}$.

### 4.1.4 Klassifikation der Notenhöhe

Aus den für jedes Fenster $i$ mit $f_{\text{heur},i}$ nach Formel (4.4) aus Abschnitt 4.1.2 geschätzten Grundfrequenzen $f_{0,i}$ sollen als nächstes Notenhöhen klassifiziert werden. Dazu wird vorausgesetzt, dass eine temperierte Stimmung (siehe Abschnitt 2.1.3) vorliegt und die Frequenz des Kammertons $a'$ bekannt ist oder geschätzt werden konnte. Bei der Erzeugung der hier verwendeten Daten war das begleitende Klavier auf $f_{a'} = 443.5$ Hertz gestimmt. Die Idee der Notenklassifikation wird in einem globaleren Zusammenhang in Abschnitt 3.3 vorgestellt.

Aus der durch $\hat{f_{0,i}} = f_{\text{heur},i}$ geschätzten Grundfrequenz eines Abschnitts $i$ wird die Notenklassifikation $n_i^\circ$ bestimmt (siehe Abschnitt 2.1.3, Formel (2.2)):

$$n_i^\circ := \left\lfloor 12 \cdot log_2 \left( \frac{f_{\text{heur},i}}{f_{a'}} \right) + 0.5 + n^{\text{cor}} \right\rfloor . \tag{4.16}$$

Dabei ist $n^{\text{cor}}$ ein Korrekturwert, der angibt, ab wo eine Frequenz dem nächsten Halbton zugeordnet wird. Als allgemein geeigneter Wert wird $n^{\text{cor}} = 0$ verwendet, so dass genau auf der Hälfte zwischen zwei Halbtönen die Klassifikation von einem zum nächsten Halbton wechselt. Das ist dann bei 50 cents ($\hat{=}$0.5 Halbtönen) der Fall. Der Wert von $n_i^\circ$ ist eine ganze Zahl, die den Abstand in Halbtönen vom Kammerton $a'$ angibt. Bei $n_i^\circ = 3$ liegt damit ein $c''$ vor.

Da jedoch nicht zu jedem Zeitpunkt Klang vorhanden ist, sondern auch Stille (siehe Abschnitt 2.1.5) in einem Musikstück auftritt, die als Pause komponiert ist oder auch durch den Interpreten verursacht wird (Atempause, Phrasierung, zu kurzes Halten von Tönen), muss auch die Klassifikation von Pausen möglich sein. Ebenso darf auch leises Rauschen, wie es bei Zischlauten entsteht, nicht zu der Klassifikation

einer falschen Note führen. Dazu wird zunächst zu jedem gefensterten Abschnitt $X_{w,o}(x,i)$ der Zeitreihe die *Energie*

$$e_i := 20 \cdot \log_{10}\left(\sum_{t=i\cdot o+1}^{i\cdot o+w} |x_t|\right) \tag{4.17}$$

berechnet (siehe auch Cano et al., 1999), die hier als Analogon zum Schalldruck Lp gebildet wird, der wie folgt (mit Einheit dB) definiert ist:

$$\text{Lp} := 20 \cdot \log_{10}\left(\frac{\text{Druck in Pascal}}{0.00002\ \text{Pa}}\right).$$

Ein Abschnitt wird dann als Pause klassifiziert, wenn er zu dem $\varphi$-ten Teil der Abschnitte mit kleinster Energie $e_i$ gehört, es also sehr leise ist, und zugleich mindestens $\psi$ interessante Peaks (siehe Formel (4.1)) im zugehörigen Periodogramm enthalten sind. Letzteres ist ein Indiz dafür, dass recht viele Frequenzen am Klang beteiligt sind, also Rauschen vorhanden ist. Als günstige Einstellungen haben sich bei den vorliegenden Daten $\varphi = 0.2$ und $\psi = 7$ erwiesen.

Aus algorithmischer Sicht wird im Fall von „Pause“ als Note ein Wert von -100 notiert, um Pausen so repräsentieren zu können, dass auch Glätter einfach mit ihnen umgehen können. Daher erhält man als Gesamtklassifikation $n_i^*$ der Notenhöhe:

$$n_i^* := \begin{cases} -100\,, & \text{falls Pause klassifiziert wurde,} \\ n_i^\circ\,, & \text{sonst.} \end{cases} \tag{4.18}$$

Abbildung 4.2 zeigt die Anwendung der Klassifikation auf die in Abschnitt 4.1.2 geschätzten Grundfrequenzen für die ersten 96 Abschnitte der Fensterung des zweiten `A` Teils von „Tochter Zion“, gesungen von Sopranistin S5. Anstelle von -100 steht hier der Wert `NA` für eine klassifizierte Pause, andere Werte geben die Entfernung in Halbtönen von $a'$ an.

### 4.1.5 Glättung und Segmentierung

In Abbildung 4.2 wird am Beispiel von Sopranistin S5 das Ergebnis der Klassifikation dargestellt. In den ersten 48 Abschnitten wechselt die Klassifikation rasch

```
NA NA -30 NA  2  2 15 15 15 15 15 15  2  3  3  3  3  3  2  2   2  2  4  4
 3  2   2  2  2  4  4  3  2  2  2  2  3  2  2  2  2  2  2  2 -29 NA NA NA
NA NA  NA -2 -1  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0   0 -1 -1 -1
 0  0   0  0 -1 -1 -1  0  0  0  0  0 -1  1  1  1  1  1  1  1   1  0 NA NA
```

Abbildung 4.2: ungeglättete Klassifikation des ersten Taktes von Sopranistin S5

zwischen den Werten 2 ($b'$), 3 ($c''$) und 4 ($cis''$). Tatsächlich sollte das $c''$ (Wert 3) gesungen werden. Die scheinbaren Fehlklassifikationen treten insbesondere deswegen auf, weil Sopranistin S5 mit sehr intensivem Vibrato (siehe Abschnitt 2.1.4) singt. Tatsächlich schwankt die Grundfrequenz aber wegen des ausgeprägten Vibratos sehr stark um das $c''$, und zwar geschätzt zwischen 489.68 und 572.60 Hertz, also um $12 \log_2(572.6/489.68) \approx 2.7$ Halbtonschritte. Während nur Schwankungen von bis zu einem Halbton üblich sind, sind jedoch auch solche von bis zu 4 Halbtonschritten beobachtet worden (siehe Abschnitt 2.1.4), so dass der geschätzte Wert durchaus noch plausibel ist.

Eine Glättung $\tilde{n}_i^*$ der Zeitreihe der klassifizierten Noten $n_i^*$ hilft in vielen Fällen, die durch Vibrato verursachten Abweichungen um einen Halbton nach oben oder unten zu kompensieren, da die Abweichungen in regelmäßigen Abständen einer Schwingung folgend in beide Richtungen auftreten (siehe Abschnitt 2.1.4).

Als häufig geeignet erweist sich ein $M_w$-fach ($M_w \in \mathbb{N}$) angewandter gleitender Median der Ordnung $M_o \in \mathbb{N}$, der damit eine Fensterbreite von $2M_o + 1$ aufweist. Für die Transkription der vorliegenden Daten wurden die Einstellungen $M_w = 2$ und $M_o = 3$ (also Fensterbreite 7) verwendet. Grund für die Wahl ist, dass die Frequenz des Vibratos, also die Geschwindigkeit des Wechsels vom hohen zum niedrigen und wieder zum hohen Teil des Vibratos, meist zwischen 5 Hertz und 7 Hertz liegt. Eine Vibratoschwingung bei 5 Hertz umfasst $\frac{1}{5}$ Sekunde und damit bei der vorliegenden Samplingrate $\frac{11025}{5} = 2205$ Beobachtungen. Bei einer Fensterbreite von 512 mit halber Überlappung (also jeweils nach 256 Beobachtungen eine neue Fensterung) sind darin $\frac{2205}{256} \approx 8.6$ Abschnitte enthalten. Analog sind bei einer Vibratoschwingung von 7 Hertz $\frac{11025}{7} = 1575$ Beobachtungen und somit $\frac{1575}{256} \approx 6.2$ Abschnitte inner-

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | NA | NA | NA |
| NA | NA | NA | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | NA | NA |

Abbildung 4.3: geglättete Klassifikation des ersten Takts von Sopranistin S5

halb einer Vibratoschwingung Eine Vibratoschwingung umfasst also etwa 6 bis 9 Abschnitte der Fensterung, so dass das Fenster der Breite 7 des Glätters *etwa* eine vollständige Vibratoschwingung umfasst.

Abbildung 4.3 zeigt das Ergebnis der durchgeführten Glättung von der Zeitreihe der Notenhöhen aus Abbildung 4.2.

Leider führt auch die Glättung bei der ersten Note nicht zur richtigen Erkennung der 3 ($c''$). Vielmehr wird der Ton zunächst genau eine Oktave zu hoch klassifiziert. Das in Abbildung 4.4 (linke Seite) dargestellte Periodogramm, für dessen Berechnung wie im Algorithmus beschrieben 512 Beobachtungen eines Teils dieses Tons verwendet wurden, zeigt den Grund: Es ist ausschließlich die Frequenz des ersten Obertons (OT1, $c'''$) im Periodogramm so präsent, dass sie sich aus dem Grundrauschen abhebt. Weder die Grundfrequenz (GF, $c''$) noch der zweite Oberton (OT2) sind zu erkennen, so dass der Algorithmus hier nur falsch schätzen kann. Erst die starke Vergrößerung in der rechten Seite von Abbildung 4.4 läßt einen minimalen „Peak“ bei der Grundfrequenz erahnen. Das Rechteck im linken Teil der Abbildung 4.4 zeigt den Ausschnitt der Vergrößerung an.

Nach der Glättung wird die Reihe so segmentiert wie es in Abschnitt 3.3 beschrieben wurde: Die geglättete Zeitreihe der klassifizierten Noten, die zu äquidistanten Punkten gemäß der Breite der Verschiebung bei der Fensterung vorliegen, wird an den Stellen, an denen sich der Notenwert ändert, segmentiert. Damit liegen an dieser Stelle die Informationen über die Höhe und die Länge der einzelnen Noten vor. Es ist zu beachten, dass diese Art der Segmentierung nicht einen langen von zwei halb so lang gesungenen Tönen gleicher Höhe unterscheiden kann.

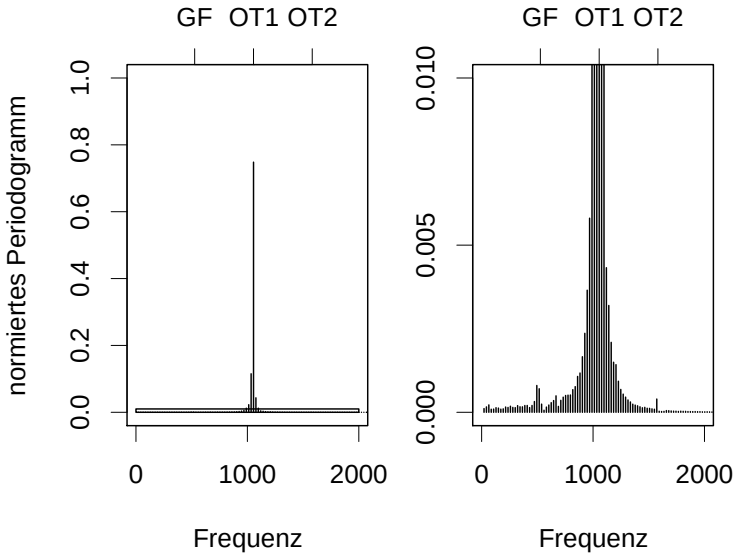


Abbildung 4.4: erstes Periodogramm des ersten Tons von Sopranistin S5
*links*: nur der erste Oberton ist erkennbar
*rechts*: Vergrößerung des Kastens aus linker Grafik

Abbildung 4.5 zeigt das Ergebnis nach Glättung und Segmentierung des zweiten von Sopranistin S5 gesungenen `A` Teils von „Tochter Zion“. Dort ist die klassifizierte Notenhöhe (einschließlich Stille oder Pausen) gegen die Zeit (hier bereits eingeteilt in Takte) abgetragen. Grau hinterlegte Kästen zeigen die Notenhöhe, die durch die wahre Notenschrift vorgegeben sind, die schwarze Linie mit längeren horizontalen Elementen zeigt den mittels der Heuristik geschätzten Verlauf der Notenhöhen. Die am unteren Rand der Grafik verlaufende schwarze Linie zeigt den aus der Originalreihe extrahierten Energieverlauf an.

Der erste Ton wird zunächst, wie bereits erwähnt, um eine Oktave zu hoch geschätzt. Danach wird die Notenhöhe nie um mehr als einen Halbton zu hoch oder zu niedrig geschätzt. Zu Beginn des vierten Takts beginnt die Sängerin den nächsten Ton deutlich zu früh. Es wird somit richtig der zu frühe Einsatz erkannt.

Auch Pausen werden immer „richtig“ angezeigt. Die Sängerin macht nämlich, bedingt durch Atmung oder das Singen stimmloser Konsonanten, deutliche Pausen am

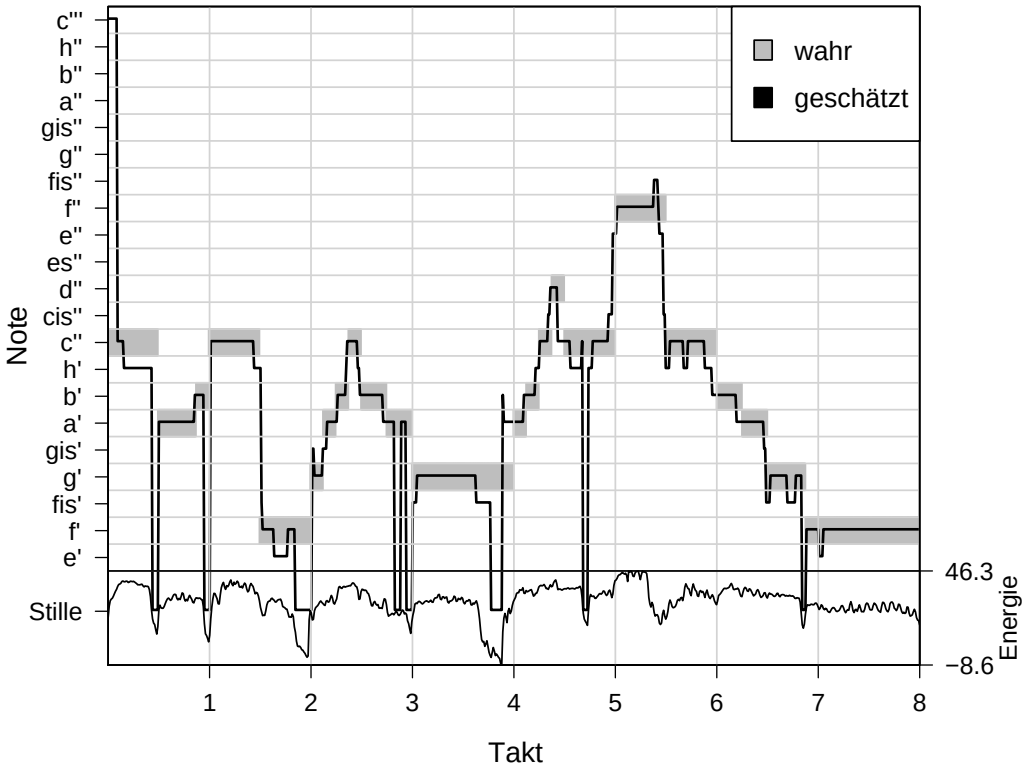


Abbildung 4.5: Ergebnis nach Glättung und Segmentierung

Ende der Takte 1, 2, 3, 4 und 7. Am Ende des dritten Takts wird die Pause jedoch etwas zu früh erkannt. Die Pause in der Mitte des ersten Takts kommt zustande, weil die Sängerin ein „ch“ (aus „Tochter“) singt, das zu wenig Energie und hohem Rauschen ohne erkennbare Grundfrequenz führt. Die Pause in der zweiten Hälfte des vierten Takts entsteht dadurch, dass die Sängerin zwei gleich hohe Töne deutlich voneinander absetzt.

### 4.1.6 Quantisierung

Mit Hilfe einer Quantisierung (siehe Abschnitt 3.4) werden als nächstes die gefundenen Längen zu plausiblen Notenwerten zusammengefasst. Es ist bekannt, dass Achtelnoten die kürzesten Werte für die vorliegenden Daten darstellen und das Stück im $\frac{4}{4}$ Takt vorliegt. Außerdem wurde durch die Klavierbegleitung konstantes Tempo

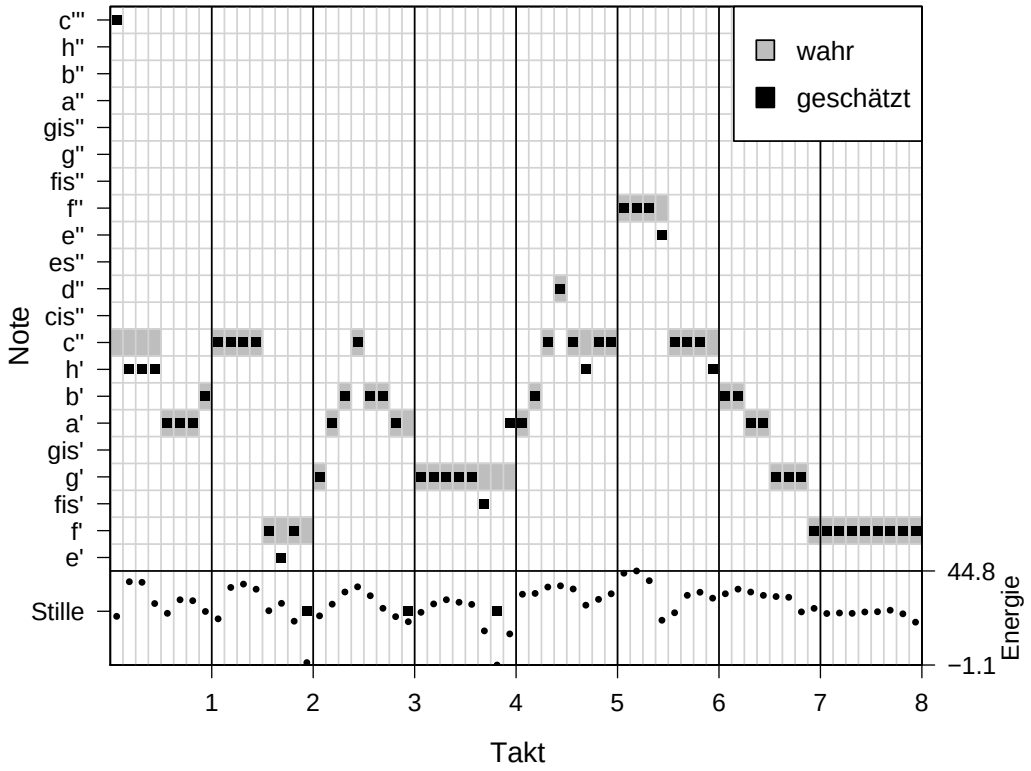


Abbildung 4.6: Ergebnis nach Quantisierung

vorgegeben. Daher wird ein statisches Quantisierungverfahren verwendet, das die Zeitreihe der geglätteten klassifizierten Notenwerte $\tilde{n}_i^*$ in so viele Segmente aufteilt, wie Achtelnoten zu erwarten sind. Im Fall der Beispieldaten sind das bei 8 Takten mit jeweils 8 Achtelnoten insgesamt 64 Segmente. In jedem Segment wird der Modus aller enthaltener $\tilde{n}_i^*$ als Notenwert verwendet. Mehrere Segmente, denen die gleiche Notenhöhe zugeordnet worden ist, werden zu einer entsprechend längeren Note zusammengefasst, sofern sie nicht eine Taktgrenze überschreiten.

Analog zu Abbildung 4.5 vor der Quantisierung wird in Abbildung 4.6 das Ergebnis nach der Quantisierung mit den Originalnoten verglichen. Die Aufteilung der Darbietung von Sopranistin S5 erfolgt dabei in Achtelnoten. Die gewählte Visualisierung zeigt noch nicht die Zusammenfassung von Achtelnoten an. Nach wie vor ist insbesondere der erste Ton fehlerhaft erkannt worden.

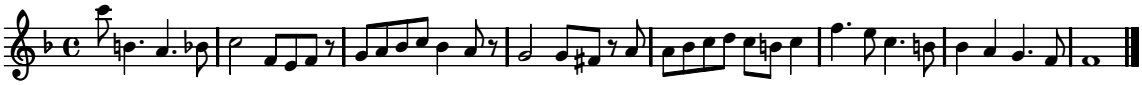

Abbildung 4.7: Ergebnis der Transkription mittels heuristischem Algorithmus von der Darbietung der Sopranistin S5

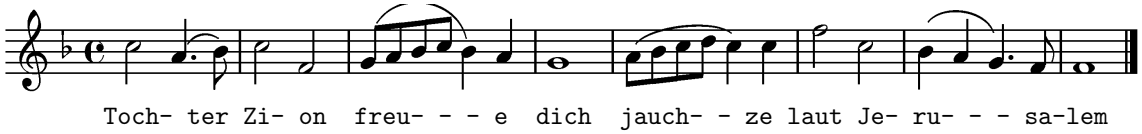


Abbildung 4.8: Originalnoten von „Tochter Zion“

### 4.1.7 Umsetzung in Notenschrift

Die Umsetzung in Notenschrift aus den zuvor ermittelten Informationen erfolgt wie in Abschnitt 3.6 beschrieben. Mit Hilfe der Funktion `lilyinput()` des R Pakets *tuneR* (siehe Kapitel 7) wird eine Eingabe für LilyPond erzeugt und damit in Notenschrift umgesetzt. Die Ergebnis ist in Abbildung 4.7 abgedruckt. Sie geht direkt aus den in Abbildung 4.6 dargestellten quantisierten Werten hervor und kann mit den Originalnoten in Abbildung 4.8 verglichen werden.

### 4.1.8 Ergebnisse der Transkription

Die Zeitreihen aller in Abschnitt 2.3 beschriebenen Sängerinnen und Sänger wurden mit der oben beschriebenen Methode transkribiert. Um nun Aussagen über die Güte der Transkriptionen machen zu können, muss ein Maß für die Fehler gefunden werden.

Für die vorliegenden Daten erscheint ein Fehlermaß geeignet zu sein, das auf falsch klassifizierten Noten der kürzesten Zeiteinheit basiert. Weil dabei Sängerinnen und Sänger häufig Noten zu früh beenden oder zu lange atmen, sollen klassifizierte Pausen nicht als Fehler gewertet werden. Der Fehler wird deswegen wie folgt gemessen:

Tabelle 4.1: Fehlerraten aller Transkriptionen

| Bass | B1 | B3 | B6 | B7 | | |
|---|---|---|---|---|---|---|
| Fehler | 0.214 | 0.436 | 0.105 | 0.353 | | |
| Tenor | T3 | T6 | T7 | | | |
| Fehler | 0.017 | 0.121 | 0.018 | | | |
| Alt | A2 | A4 | A5 | A8 | A9 | A10 |
| Fehler | 0.200 | 0.160 | 0.036 | 0.053 | 0.069 | 0.105 |
| Sopran | S1 | S2 | S4 | S5 | | |
| Fehler | 0.131 | 0.077 | 0.109 | 0.164 | | |

$$e := \frac{\#\text{ falsch klassifizierte Noten (ohne Pausen)}}{\#\text{ alle Noten} - \#\text{ klassifizierte Pausen}}, \tag{4.19}$$

wobei jeweils die kürzeste Zeiteinheit zu betrachten ist. In den Daten müssen jeweils 64 Achtel in 8 Takten betrachtet werden. Eine falsch klassifizierte Viertelnote ginge also als 2 falsch klassifizierte Achtelnoten in die Wertung ein.

Für Sopranistin S5 lässt sich die Fehlerrate gut an Abbildung 4.6 ablesen: Es gibt 10 falsch klassifizierte Achtel und 3 Achtelpausen, also $e = \frac{10}{64-3} = 0.164$. Es ist zu beachten, dass damit eine gemeinsame Fehlerrate von Sängerin oder Sänger und dem Algorithmus geschätzt wird, denn neben dem Algorithmus machen natürlich auch die Interpreten Fehler. Singt ein Interpret zu früh, zu spät, zu hoch oder zu tief, wird sich dies in der Transkription niederschlagen und auch als Fehler gewertet. Tabelle 4.1 enthält die Fehlerraten aller Transkriptionen.

Einige Fehlerraten sind hervorragend klein (für Tenöre T3, T7 und Alt A5 jeweils kleiner als 2%), andere allerdings inakzeptabel hoch (Bässe B3 und B7). Gerade bei den eng liegenden Halbtonabständen der Bässe liegen jedoch auch die zuvor beschriebenen Probleme bei der Auflösung des Periodogramms an den Fourier Frequenzen. Es wurde daher versucht, die Klassifikation mit Hilfe von a-priori Informationen zu verbessern (Weihs et al., 2005a), nämlich Informationen über die Stimmlage der

Sängerin oder des Sängers und Informationen über das vorliegende Musikstück.

Es ist durchaus realistisch, dass die Stimmlage der Sängerin oder des Sängers einem Algorithmus vor Beginn der Transkription „mitgeteilt“ werden kann. Der Algorithmus sucht dann nur nach solchen Grundfrequenzen, die auch von der entsprechenden Stimme gesungen werden können und verwirft andere Frequenzen. Leider führt diese a-priori Information über die Stimmlage nicht zu einer bedeutenden Verbesserung bei den untersuchten Datensätzen, so dass sie nicht weiter verwendet wird. Im Beispiel der Transkription von Sängerin S5 hat der Algorithmus auch ohne diese Information nur Grundfrequenzen erkannt, die auch von einer Sopranstimme gesungen werden können.

In Tabelle 4.2 werden die Fehlerraten aus Weihs et al. (2005a) (unter Verwendung des Glätters wie in Abschnitt 4.1.5 beschrieben) angegeben, um zu zeigen, dass die Verbesserung durch Angabe der Stimmlage nicht wesentlich ist. Es ist zu beachten, dass diese Fehlerraten sich noch auf einen weniger gut entwickelten heuristischen Algorithmus beziehen, so dass sie nicht direkt mit den in Tabelle 4.1 oder später in dieser Arbeit veröffentlichten Fehlerraten vergleichbar sind. Wegen des hohen Aufwands wurde auf eine Anpassung der Werte aus Tabelle 4.2 an den neuen Algorithmus verzichtet, zudem innerhalb der Tabelle das (zum Teil fehlende) Potenzial der Nutzung von a-priori Information deutlich wird.

Als wirksamer stellte sich die Benutzung der a-priori Information über das Musikstück heraus (siehe Tabelle 4.2). Der Algorithmus sucht dabei nur nach solchen Grundfrequenzen, die den in dem Musikstück vorkommenden Notenhöhen entsprechen. Allerdings ist es sehr unrealistisch, dass solch eine a-priori Information vorliegt, da dann ohnehin das Musikstück bekannt sein dürfte und die Transkription damit unnötig wäre. In Abschnitt 4.2 wird daher ein ganz anderer Ansatz zur Verbesserung der Fehlerraten vorgestellt.

Tabelle 4.2: Fehlerraten der Transkription nach Weihs et al. (2005a) unter Ausnutzung verschiedener a-priori Informationen

| Sänger(in) | keine Info | Stimmlage | Musikstück |
|---|---|---|---|
| B1 | 0.27 | 0.29 | 0.17 |
| B3 | 0.52 | 0.52 | 0.34 |
| B6 | 0.20 | 0.20 | 0.11 |
| B7 | 0.48 | 0.33 | 0.23 |
| T3 | 0.13 | 0.13 | 0.07 |
| T6 | 0.23 | 0.24 | 0.14 |
| T7 | 0.16 | 0.16 | 0.10 |
| A2 | 0.28 | 0.29 | 0.11 |
| A4 | 0.24 | 0.22 | 0.14 |
| A5 | 0.10 | 0.10 | 0.04 |
| A8 | 0.12 | 0.11 | 0.09 |
| A9 | 0.12 | 0.12 | 0.08 |
| A10 | 0.17 | 0.16 | 0.05 |
| S1 | 0.19 | 0.17 | 0.11 |
| S2 | 0.18 | 0.17 | 0.08 |
| S4 | 0.25 | 0.22 | 0.19 |
| S5 | 0.22 | 0.20 | 0.19 |
| MIN | 0.10 | 0.10 | 0.04 |
| MED | 0.20 | 0.20 | 0.11 |
| MAX | 0.52 | 0.52 | 0.34 |

## 4.2 Parameteroptimierung für die Heuristik

Die zuvor beschriebene Heuristik kann noch auf anderem Wege für eine bestimmte Sängerin oder einen bestimmten Sänger optimiert werden, deren in Wave Format vorliegende Darbietung transkribiert werden soll. Dazu müssen einige Parameter, darunter einige der bereits beschriebenen Schwellenwerte (eine Zusammenfassung folgt auf Seite 51), optimal an die Charakteristiken der jeweiligen Stimme angepasst

werden. Solche Charakteristiken sind zum Beispiel die unterschiedlichen Anteile der Obertöne, die den Klang prägen (siehe Abschnitt 2.1.2), und das Volumen und die Geschwindigkeit des Vibratos (siehe Abschnitt 2.1). Entsprechende eigene Arbeiten zu Eigenschaften des Klangs einer Singstimme sind beispielsweise Weihs und Ligges (2003), Weihs et al. (2005b) und Weihs und Ligges (2005). All diese Arbeiten zeigen, dass sich die Klangeigenschaften durch Änderungen des Periodogramms zeigen. Daher sind die Anpassungen der Schwellenwerte und anderer Parameter notwendig.

Für die Optimierung der Transkription und die Anpassung an die Charakteristika einer Stimme wird mit Hilfe des Optimierungsverfahrens von Nelder und Mead (1965) die Fehlerrate auf dem dritten Teil von „Tocher Zion“ durch Optimierung der Paramatereinstellungen minimiert. Die Optimierung verlangt, dass Sängerinnen und Sänger ein dem Algorithmus bekanntes Stück, zu dem die Noten vorliegen, als Trainingsbeispiel singen. Um die Heuristik zu verbessern, werden jetzt also a-priori Informationen genutzt, die es zwar nötig machen, das Programm zu trainieren, dafür aber letztendlich die Ergebnisse verbessern. Ein solches Training mit a-priori Informationen auf festgelegten Mustern ist auch im Bereich der automatischen Schrifterkennung (OCR) und Spracherkennung durch Algorithmen üblich.

Die einzelnen zu optimierenden Parameter sind im Folgenden aufgeführt. Für ausführliche Erläuterungen wird auf Abschnitt 4.1 verwiesen. Die Standardwerte (ohne Optimierung) des Algorithmus sind in Klammern angegeben:

- $h$: *Interessante Peaks* im Periodogramm müssen größer als der $h$-te (0.015) Teil des höchsten Peaks des Periodogramms sein.
- $\varphi$: Nur die $\varphi$ (0.2) Abschnitte mit kleinster Energie werden als Kandidaten für „Stille“ (Rauschen) in Betracht gezogen.
- $\psi$: „Stille“ (Rauschen) wird definiert als Abschnitte mit niedriger Energie (siehe $\varphi$), bei denen mindestens $\psi$ (7) interessante Peaks vorhanden sind.

  $m, q, l, r$: Parameter, die über die „Vernünftigkeit“ einer möglichen Grundfrequenz $\tilde{\lambda}$ entscheiden (z.T. auf Obertönen basierend), sind im Einzelnen:

- $m$: Mit diesem Faktor (1.8) wird die Frequenz des ersten interessanten Peaks $\min(\tilde{\lambda})$ so multipliziert, dass gilt: $\lambda^* \in [\min(\tilde{\lambda}),\ m \cdot \min(\tilde{\lambda})]$.
- $q$: Diese Anzahl (10) der kleinsten Fourier Frequenzen soll nicht beachtet werden.
- $l, r$: Falls ein interessanter Peak für die Faktoren $l$ (1.35) und $r$ (1.65) in dem Intervall $[l\lambda^*,\ r\lambda^*]$ liegt, ist es plausibel, dass als bisheriges $\lambda^*$ der erste Oberton statt der Grundfrequenz $f_0$ gefunden wurde.
- $M_o$: Die Ordnung des Median Glätters (3) führt zu der Fensterbreite $2M_o + 1$.
- $M_w$: Durch $M_w$ (2) ist die Anzahl der Wiederholungen des Median Glätters gegeben.
- $n^{\text{cor}}$: Dieser Halbton Schwellenwert (0) gibt die Abweichung von 0.5 Halbtönen an, ab der der nächst höhere Halbton klassifiziert wird.

Fehlerraten werden wie im letzten Abschnitt (siehe Formel (4.19), Seite 48), basierend auf falsch klassifizierten Noten der kürzesten Zeiteinheit, berechnet:

$$e := \frac{\#\text{ falsch klassifizierte Noten (ohne Pausen)}}{\#\text{ alle Noten} - \#\text{ klassifizierte Pausen}}.$$

In dem Beispiel werden 64 Achtel in 8 Takten betrachtet. Um übersichtliche Notenschrift zu erzeugen, werden aufeinanderfolgende gleichartige Achtel zusammengefasst.

Tabelle 4.3 zeigt die Ergebnisse der Optimierung. In der ersten Zeile sind die Standardeinstellungen für die Heuristik angegeben, in den Zeilen zwei bis fünf sind die Startwerte für das Optimierungsverfahren angegeben, wobei in den weiteren Zeilen das beste Resultat der vier Suchläufe als Ergebnis verwendet wird.

Bei fast allen Sängerinnen und Sängern konnte durch Lernen von Charakteristiken der Stimmen mit Hilfe von Parameteroptimierung die Fehlerrate der Transkription mitunter stark verbessert werden. In einigen Fällen konnte die aus der nicht optimierten Heuristik resultierende Fehlerrate $e_h$ durch die Optimierung (Fehlerrate $e_o$)

Tabelle 4.3: Ergebnisse der Nelder-Mead Optimierung

| | $h$ | $\varphi$ | $\psi$ | $m$ | $q$ | $l$ | $r$ | $M_o$ | $M_w$ | $n^{\text{cor}}$ | $e_o$ | $e_h$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Standard | 0.015 | 0.20 | 7 | 1.80 | 10 | 1.35 | 1.65 | 3 | 2 | 0.0000 | | |
| start1 | 0.016 | 0.15 | 10 | 1.80 | 22 | 1.30 | 1.65 | 5 | 3 | 0.0000 | | |
| start2 | 0.012 | 0.25 | 6 | 1.80 | 9 | 1.36 | 1.70 | 3 | 2 | 0.0000 | | |
| start3 | 0.018 | 0.27 | 9 | 1.80 | 20 | 1.32 | 1.67 | 5 | 2 | 0.0000 | | |
| start4 | 0.012 | 0.23 | 6 | 1.90 | 9 | 1.40 | 1.71 | 3 | 2 | 0.0100 | | |
| | | | | | | *Bass* | | | | | | |
| B1 | 0.013 | 0.24 | 6 | 1.80 | 10 | 1.36 | 1.69 | 3 | 2 | *0.1678* | 0.115 | 0.214 |
| B3 | 0.013 | 0.23 | 7 | 1.89 | 9 | 1.41 | 1.72 | 2 | 2 | 0.1495 | 0.333 | 0.436 |
| B6 | *0.028* | 0.30 | 7 | 1.84 | 12 | 1.35 | 1.65 | 6 | 2 | 0.0849 | 0.083 | 0.105 |
| B7 | *0.005* | 0.22 | 8 | 1.90 | 9 | *1.50* | *1.76* | 2 | 2 | 0.0404 | 0.122 | 0.353 |
| | | | | | | *Tenor* | | | | | | |
| T3 | 0.017 | 0.25 | 6 | 1.81 | 9 | 1.45 | 1.70 | 3 | 2 | 0.0089 | 0.017 | 0.017 |
| T6 | 0.003 | 0.20 | 9 | 1.81 | 14 | 1.45 | 1.73 | 5 | 2 | 0.1203 | 0.060 | 0.121 |
| T7 | 0.022 | 0.24 | 6 | 1.82 | 11 | 1.38 | 1.68 | 3 | 2 | 0.0182 | 0.017 | 0.018 |
| | | | | | | *Alt* | | | | | | |
| A2 | 0.014 | 0.21 | 6 | 1.93 | 12 | 1.43 | 1.74 | 3 | 2 | 0.0147 | 0.170 | 0.200 |
| A4 | 0.015 | 0.25 | 5 | 1.80 | 11 | 1.39 | 1.73 | 6 | 2 | 0.0296 | 0.075 | 0.160 |
| A5 | 0.016 | 0.23 | 6 | 1.83 | 10 | 1.39 | 1.67 | 4 | 2 | 0.0213 | 0.017 | 0.036 |
| A8 | 0.013 | 0.26 | 7 | 1.81 | 10 | 1.37 | 1.71 | *7* | 2 | 0.0141 | 0.034 | 0.053 |
| A9 | 0.021 | 0.25 | 6 | 1.80 | 9 | 1.36 | 1.70 | 3 | 2 | 0.0000 | 0.069 | 0.069 |
| A10 | 0.017 | 0.33 | 2 | 1.86 | *23* | 1.33 | 1.71 | 3 | 2 | 0.0150 | 0.022 | 0.105 |
| | | | | | | *Sopran* | | | | | | |
| S1 | 0.021 | 0.30 | *2* | 1.83 | *23* | 1.35 | 1.70 | 6 | *3* | 0.0273 | 0.044 | 0.131 |
| S2 | 0.017 | 0.25 | 6 | 1.80 | 9 | 1.36 | 1.70 | 4 | 2 | 0.0035 | 0.039 | 0.077 |
| S4 | 0.012 | 0.25 | 6 | *1.97* | 9 | 1.36 | 1.70 | 3 | 2 | 0.0000 | 0.075 | 0.109 |
| S5 | 0.016 | 0.24 | *10* | 1.81 | *23* | *1.31* | 1.66 | 5 | *3* | 0.0441 | 0.078 | 0.164 |

mehr als halbiert werden. Keine Verbesserung gab es bei der Fehlerrate von Tenor T3, der schon bei Standardeinstellungen mit nur 1.7% Fehlerrate transkribiert wurde.

Die schlechteste Transkription ist bei Bass B3 festzustellen. Die zweit schlechteste Transkription vor der Optimierung liegt bei dem Kammersänger (!) Bass B7 vor, bei dem nach der Optimierung eine deutliche Verbesserung und die extrem kleine Einstellung $h = 0.005$ auffällig ist. Dies liegt daran, dass die Stimmen professioneller Sänger sehr viel Stärke in den Obertönen aufweisen, der Grundton also oft im Periodogramm durch einen sehr kleinen Peak im Vergleich zu den Peaks der Obertönen repräsentiert wird. Bass B1 fällt durch den Wert $n^{\text{cor}} = 0.1678$ auf. Dieser Sänger intoniert offensichtlich tendenziell zu tief, so dass bei der Transkription die Grundfrequenzschätzung um etwa $\frac{1}{6}$ Halbton nach oben korrigiert werden muss.

Die professionelle Sopranistin S5 ist am auffälligsten, nicht nur wegen der größten Fehlerrate der Soprane, sondern vor allem durch extreme Einstellungen einiger Parameter und die deutliche Verringerung der Fehlerrate nach Optimierung. Für diese Sopranistin werden in den Abbildungen 4.9 und 4.10 die Verringerungen der Fehlerrate im Einzelnen deutlich, wenn sie mit den Abbildungen 4.5 und 4.6, die das Ergebnis vor der Optimierung wiedergeben, verglichen werden.

Das Ergebnis der Transkription der Sopranistin S5 vor der Optimierung (Abbildung 4.11) kann mit dem Ergebnis nach der Optimierung (Abbildung 4.12) und den Originalnoten von „Tochter Zion“ (Abbildung 4.13) verglichen werden.

Neben den Fehlerraten der Transkription von Sopranistin S5 konnten auch die von Bass B7, Tenor T6, Alt A4, A5, A10 und Sopranistin S1 mehr als halbiert werden. Das „Trainieren“ der Heuristik (d.h. die Optimierung der Parameter) führt also gerade bei Sängerinnen und Sängern, die aufgrund der Klangcharakteristika ihrer Stimme schlecht zu transkribieren sind, zu starken Verbesserungen. Ob das Training möglich und sinnvoll ist, hängt allerdings vom Einsatzgebiet des Verfahrens ab. Sicherlich sollte ein Anwender, der immer wieder selbst gesungene Töne transkribieren möchte, die Parameter für seine eigene Stimme optimieren. Sollen von verschiedenen

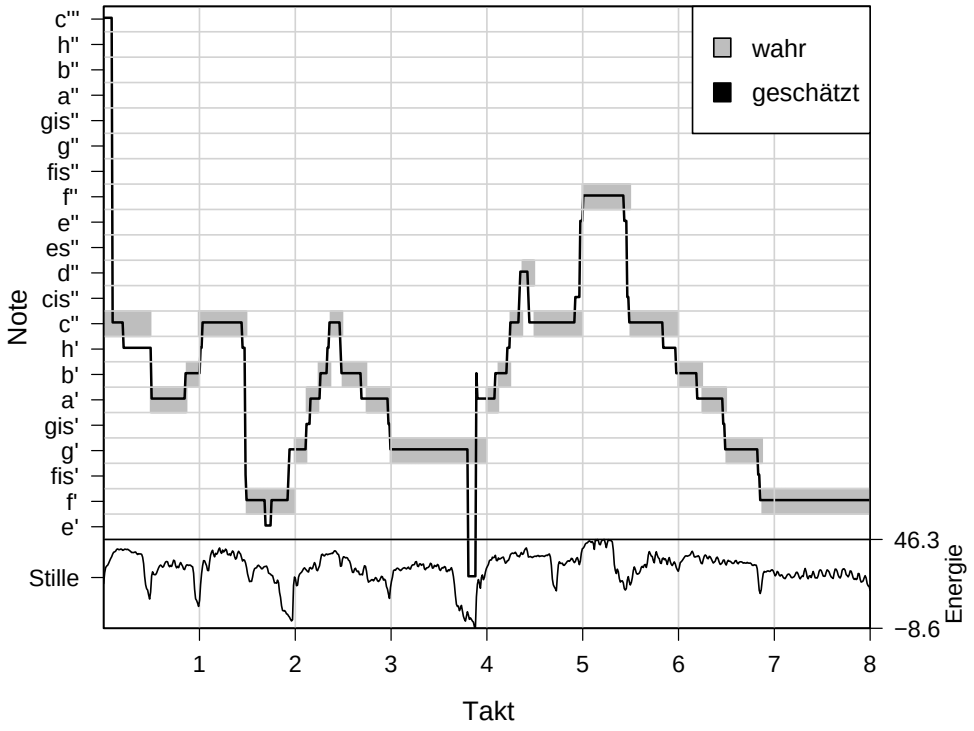


Abbildung 4.9: Ergebnis der Transkription nach Glättung und Segmentierung mit optimierten Parametern bei Sängerin S5

Personen einmalig gesungene Töne in Noten umgesetzt werden, ist die Frage, ob die Verringerung der Fehlerrate die durch das Training entstehenden Kosten aufwiegt. Solche Kosten sind beispielsweise neben der Rechenzeit für die Optimierung vor allem die Zeit, die Personen durch das Singen des Traingsliedes aufwenden müssen.

Im folgenden Kapitel werden modellbasierte Vorschläge zur Schätzung der Grundfrequenz gemacht, um solche Fehler wie bei Sopranistin S5 zu vermeiden: In einem Fall wurde statt der Frequenz des Grundtons die Frequenz des ersten Obertons geschätzt. Auch das intensive Vibrato führt bei der Heuristik noch zu einigen Fehlern, so dass Notenhöhen um einen Halbton über- oder unterschätzt werden. Außerdem kann die Heuristik nur für monophonen Klang eingesetzt werden. Die Transkription polyphoner Klänge ist nicht möglich.

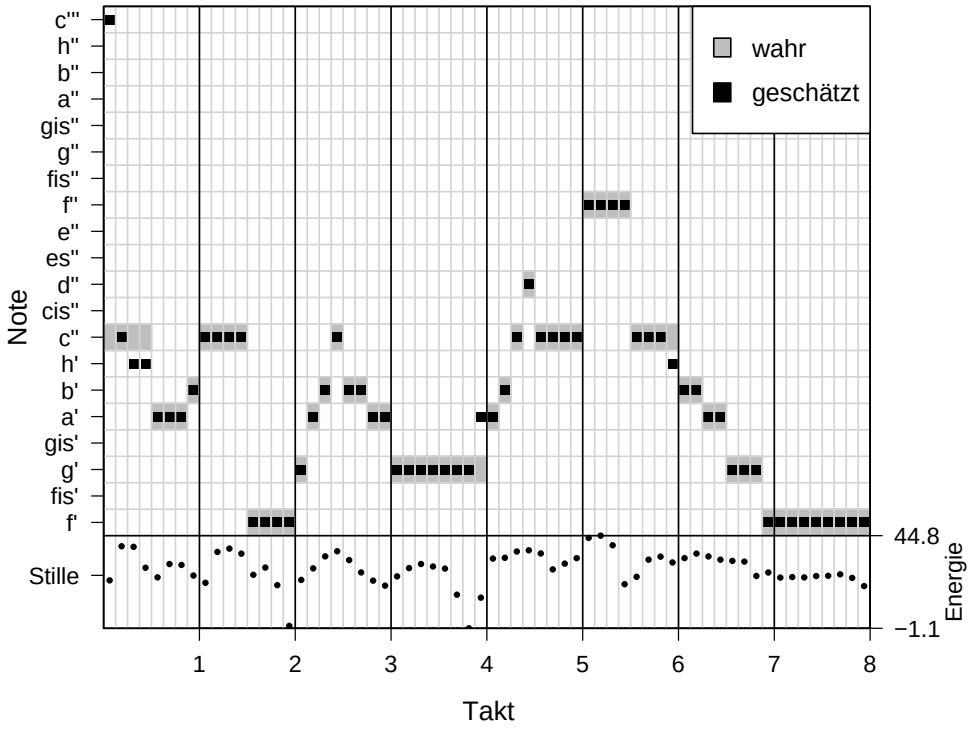


Abbildung 4.10: Ergebnis der Transkription nach Quantisierung mit optimierten Parametern bei Sängerin S5

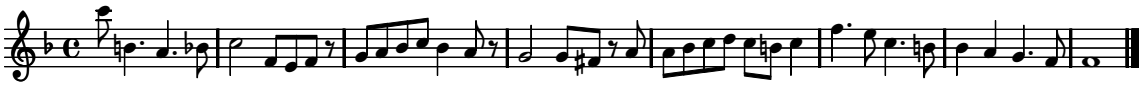

Abbildung 4.11: Ergebnis der Transkription von Sängerin S5 ohne optimierte Parameter

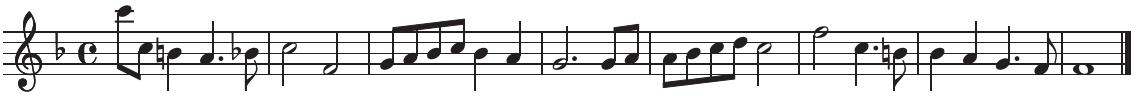

Abbildung 4.12: Ergebnis der Transkription von Sängerin S5 mit optimierten Parametern

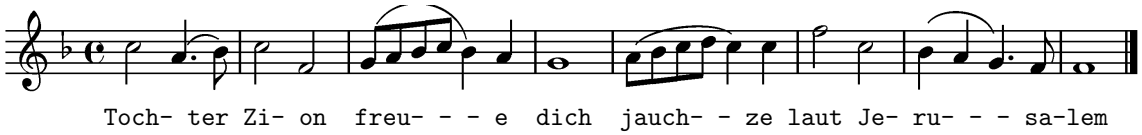


Abbildung 4.13: Originalnoten von „Tochter Zion“

# Kapitel 5

# Modellbasierte Optimierung zur Grundfrequenzschätzung

Die Grundfrequenzschätzung mit Hilfe der Heuristik schlägt in einigen Fällen fehl. Das ist zum einen der Fall, wenn von allen Partialtönen (Grundton und Obertöne) nur einer ausgeprägt ist und der Grundton somit nicht zu erkennen ist, und zum anderen, wenn ein Sänger ein sehr stark ausgeprägtes Vibrato besitzt. Dann sind durch die Fensterung nicht mehr einzelne kurze Töne mit Halbtonabstand von Vibrato zu unterscheiden. Die Hoffnung besteht nun darin, durch modellgestützte Verfahren Parameter für solche „Begleiterscheinungen“ mitzuschätzen, um sie so bei der Grundfrequenzschätzung als Einflussfaktoren auszuschließen. Wenn also das Vibrato als Einflussfaktor im Modell mitgeschätzt wird, so sollte die resultierende Schätzung der Grundfrequenz frei sein von den Einflüssen des Vibratos, so dass die dadurch verursachten Schwankungen „herausgerechnet“ werden.

Ein sehr allgemeines Modell für Zeitreihen monophoner musikalischer Klänge stammt von Rossignol et al. (1999a), die ein Modell zur Tonhöhenverfolgung mit Hilfe der Schätzung lokaler Frequenzen und Segmentierung vorschlagen, das auch Vibrato mit modelliert, wie es zum Beispiel bei professionellen Sängern und vielen Instrumenten auftritt. Davy und Godsill (2002) schlagen dagegen ein Modell oh-

ne Vibrato vor, das aber polyphone Klänge und Lautstärkeschwankungen erfassen kann. In diesem Kapitel werden zunächst beide Modelle zu einem allgemeineren Modell für den monophonen Fall vereinigt. In Abschnitt 5.2 werden Optimierungsverfahren auf Basis von Nelder und Mead für die Modellparameter zur Anwendung im Zeit- und im Frequenzbereich vorgestellt, während in Abschnitt 5.3 durch Erweiterung zu einem hierarchischen Bayes Modell eine Optimierung mit Hilfe von MCMC Verfahren erfolgt.

Die modellbasierten Schätzungen sollen bei Erfolg in den in Kapitel 4 vorgestellten Algorithmus zur Transkription eingebaut werden. Sie ersetzen dann die in Abschnitt 4.1.2 vorgestellte heuristische Grundfrequenzschätzung.

Die Anwendung der hier vorgestellten Schätzverfahren für die Modellparameter und Vergleiche der Schätzgenauigkeit für die Grundfrequenz mit den Ergebnissen der Heuristik erfolgen in Kapitel 6.

## 5.1 Das Modell

Davy und Godsill (2002) lassen Phasenverschiebungen und Frequenzverschiebungen von Obertönen zu und modellieren Amplitudenvariationen mit Hilfe von trigonometrischen Basisfunktionen. Als Erweiterung im monophonen Fall wird in dem folgenden allgemeinen Modell das Vibrato wie in Rossignol et al. (1999a) durch eine Sinusschwingung um die mittlere wahrgenommene Frequenz und deren Obertöne modelliert:

$$\begin{aligned} y_t \quad = \sum_{h=1}^{H} \sum_{i=0}^{I} \quad & \Phi_i(t) B_{h,i} \cos\left[2\pi(h+\delta_h) f_0 t + \phi_h \right. \\ & \left. +(h+\delta_h) A_v \sin(2\pi f_v t + \phi_v)\right] + \varepsilon_t, \end{aligned} \tag{5.1}$$

wobei

- $t \in \left\{\frac{0}{S}, \frac{1}{S}, \ldots, \frac{T-1}{S}\right\}$ Zeitindex mit der Anzahl Beobachtungen $T$ und Samplingrate $S$,

- $i$ = Index der $I+1$ Basisfunktionen,
- $\Phi_i(t) := \cos^2\left[\pi\frac{tS-i\Delta}{2\Delta}\right]\mathbf{1}_{[(i-1)\Delta,(i+1)\Delta]}(t)$ ist die $i$-te Basisfunktion definiert in gleichbreiten Fenstern mit 50% Überlappung, wobei $\Delta := \frac{T-1}{I}$, $S$ die Samplingrate und $\mathbf{1}$ die Indikatorfunktion ist;
- $f_0$ = Grundfrequenz, der hauptsächlich interessierende Parameter,
- $H$ = Anzahl der Partialtöne (Grundton + $(H-1)$ Obertöne),
- $B_{h,i}$ = Amplitude des $h$-ten Partialtons bzgl. der $i$-ten Basisfunktion,
- $\delta_h$ = Frequenzverschiebung des $h$-ten Partialtons mit $\delta_1 := 0$ (keine Verschiebung des Grundtons),
- $\phi_h$ = Phasenverschiebung des $h$-ten Partialtons,
- $f_v$ = Vibratofrequenz,
- $A_v$ = Vibratoamplitude,
- $\phi_v$ = Phasenverschiebung des Vibratos und
- $\varepsilon_t$ = Modellfehler.

Für die folgenden Betrachtungen wird die Reihe $y_t$ normiert auf [-1,1]. Damit kann die absolute Lautstärke eines Tons nicht mehr bestimmt werden, aber weiterhin die relative Lautstärke.

### 5.1.1 Modellbildung

Zum besseren Verständnis werden jetzt die einzelnen Bestandteile des Modells vorgestellt und motiviert. Das Modell

$$y_t = b_1\cos\left[2\pi f_0 t\right] + c_1\sin\left[2\pi f_0 t\right].$$

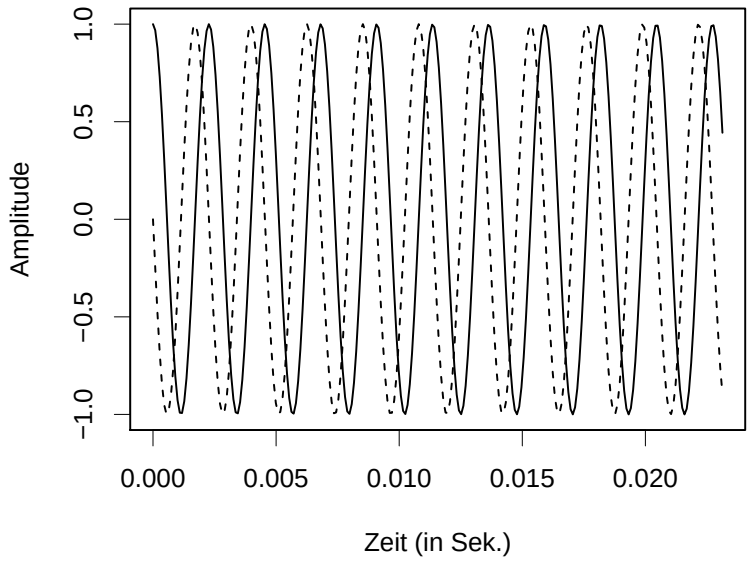


Abbildung 5.1: harmonische Schwingung mit 440 Hz, ohne (durchgehend) und mit (gestrichelt) Phasenverschiebung

stellt eine harmonische Schwingung ohne Obertöne dar. Abbildung 5.1 (durchgehende Zeitreihe) zeigt eine solche harmonische Schwingung mit 440 Hz. Eine andere Darstellung (Bronstein et al., 2000, S. 83) mit einer sogenannten Phasenverschiebung $\phi$ ist:

$$y_t = B_1 \cos\left[2\pi f_0 t + \phi\right],$$

wobei $B_1 = \sqrt{b_1^2 + c_1^2}$ und $\tan(\phi + \pi/2) = b_1/c_1$.

Diese Phasenverschiebung lässt zu, dass der Ton zu Beginn des gewählten Ausschnitts nicht unbedingt mit dem Beginn einer Kosinusschwingung starten muss. Abbildung 5.1 (gestrichelte Zeitreihe) zeigt dieselbe Schwingung erneut, aber mit einer Phasenverschiebung von $\phi = \pi/2$.

Eine Verallgemeinerung stellt die Berücksichtigung sogenannter Obertöne dar, d.h. von Vielfachen der Grundfrequenz $f_0$:

$$y_t = \sum_{h=1}^{H} B_h \cos\left[2\pi h f_0 t + \phi_h\right].$$

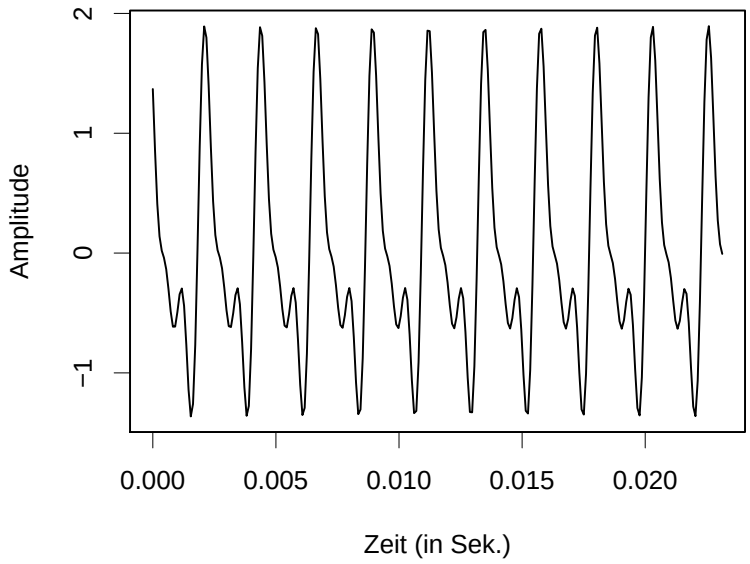


Abbildung 5.2: harmonische Schwingung mit 440 Hz, 2 Obertöne

Abbildung 5.2 zeigt den Effekt der Überlagerung mit Obertönen, wobei $B_1 = 1$, $B_2 = 0.7$, $B_3 = 0.4$. Die Grundfrequenz legt die gehörte Tonhöhe fest, die Amplituden $B_h$ der Obertöne die Klangfarbe (Reuter, 2002). Bei professionellen Sängern wird häufig unterstellt, dass sie ihre Stimme so trainieren, dass die Spektralanteile im sogenannten „Sängerformanten“ (Frequenzregionen mit starken Anteilen im Periodogramm) zwischen 2000 und 3000 Hz möglichst groß sind, um sich gegenüber dem Orchester durchzusetzen (Seidner und Wendler, 1997, S. 122).

Frequenzverschiebungen $\delta_h f_0$ sind insbesondere deshalb zu modellieren, weil bei einigen Instrumenten gewisse Obertöne unharmonisch sein können (Blackham, 1988):

$$y_t = \sum_{h=1}^{H} B_h \cos\left[2\pi(h + \delta_h) f_0 t + \phi_h\right], \quad \delta_1 := 0.$$

Ein Beispiel für solche Verschiebungen findet sich in Abb. 5.3. Hier wird die Originalreihe einer professionellen Sopranistin (durchgehend) modelliert durch eine harmonische Schwingung (465 Hz, gestrichelt) mit einem Oberton, der um $\delta_2 = 0.02$ verschoben ist. Analog dazu zeigt Abb. 5.4 zum Vergleich die Schwingung mit um $\delta_2 = 0.02$ verschobenen Oberton (wieder gestrichelt) und eine analog künstlich

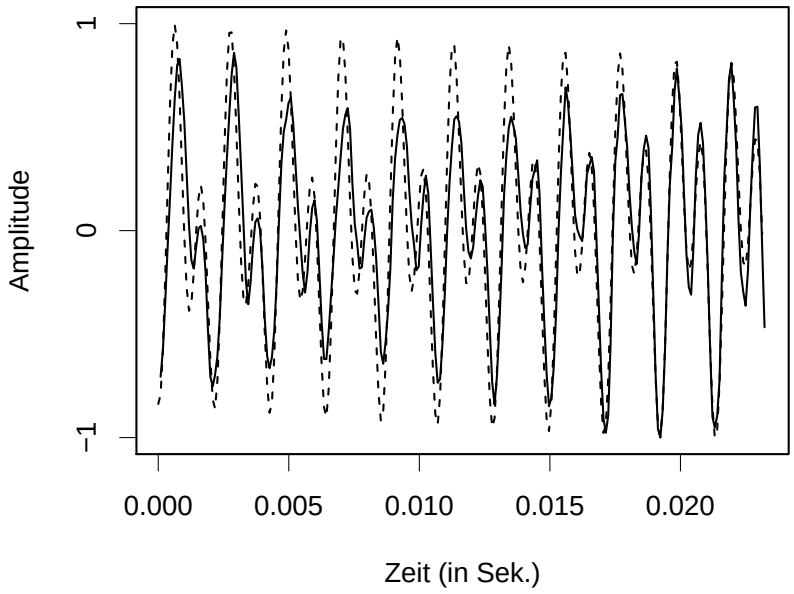


Abbildung 5.3: reale Schwingung (durchgehend) und Annäherung (gestrichelt) durch harmonische Schwingung mit 465 Hz und einen um $\delta_2 = 0.02$ verschobenen Oberton

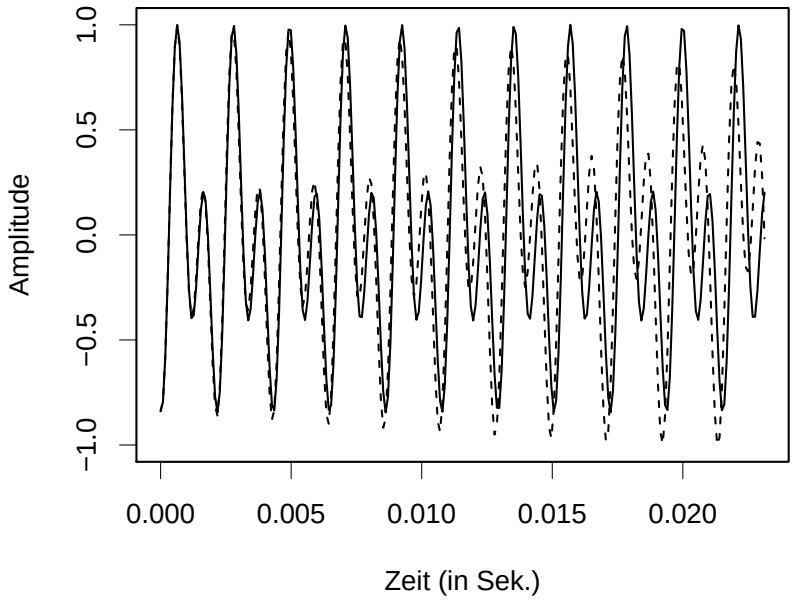


Abbildung 5.4: simulierte Schwingungen ohne verschobenen (durchgehend) und mit um $\delta_2 = 0.02$ (gestrichelt) verschobenem Oberton

erzeugte Schwingung ohne verschobenen Oberton (durchgehend). Man sieht, dass eine kleine Verschiebung schnell zu unterschiedlichen Schwingungen führen kann. Um Fehlschätzungen der Grundfrequenz zu vermeiden, ist daher die Aufnahme der Frequenzverschiebung in das Modell sinnvoll.

Zur Modellierung von Vibrato wird eine Sinusschwingung mit einer Frequenz $f_v$ von bis zu 12 Hz mit Amplitude $A_v$ und Phasenverschiebung $\phi_v$ verwendet:

$$y_t = \sum_{h=1}^{H} B_h \cos\left[2\pi(h+\delta_h)f_0 t + \phi_h + (h+\delta_h)A_v \sin(2\pi f_v t + \phi_v)\right].$$

Der Sinus im Kosinus führt dazu, dass die Grundfrequenz, mit der der Grundton schwingt, selbst mit der Vibratofrequenz um die eigentliche Grundfrequenz schwingt. Es wird also angenommen, dass Vibrato durch eine sinusförmige Wellenbewegung modelliert werden kann.

Abbildung 5.5 zeigt zwei Reihen mit jeweils 100 Hz Grundfrequenz, wobei der mit durchgehender Linie gezeichneten Zeitreihe kein Vibrato und der gestrichelten Reihe eine Vibratofrequenz von $f_v = 5$ und eine Vibratoamplitude von $A_v = 2$ zugrunde liegt. Man sieht, dass die gestrichelte Schwingung im Mittel gleich viele Schwingungen macht wie die durchgehende Linie, die Schwingungen sich aber gegeneinander verschieben. Ausschnitte der zugehörigen Periodogramme an allen Fourierfrequenzen zwischen 0 und 200 Hz beider Reihen sind in Abbildung 5.6 dargestellt, wobei an der Stelle der wahren Frequenz eine gepunktete Linie eingezeichnet ist. Auf dem gezeigten Abschnitt führt das Vibrato zu einer Verstärkung der Fourierfrequenz links neben der wahren Frequenz.

Man kann entweder ganz einfach annehmen, dass das Rauschen $\varepsilon_t$ einer Normalverteilung folgt, wenn unter anderem keine zeitliche Abhängigkeit im Rauschen mehr vorliegt, oder weiter modellieren. An dieser Stelle soll jedoch das Modell nicht weiter aufgebläht werden. Damit ist das Modell in Formel (5.1) vollständig mit Ausnahme des Modellteils über Amplituden-Schwankungen (= relative Lautstärken), die hier nicht weiter untersucht werden, erklärt.

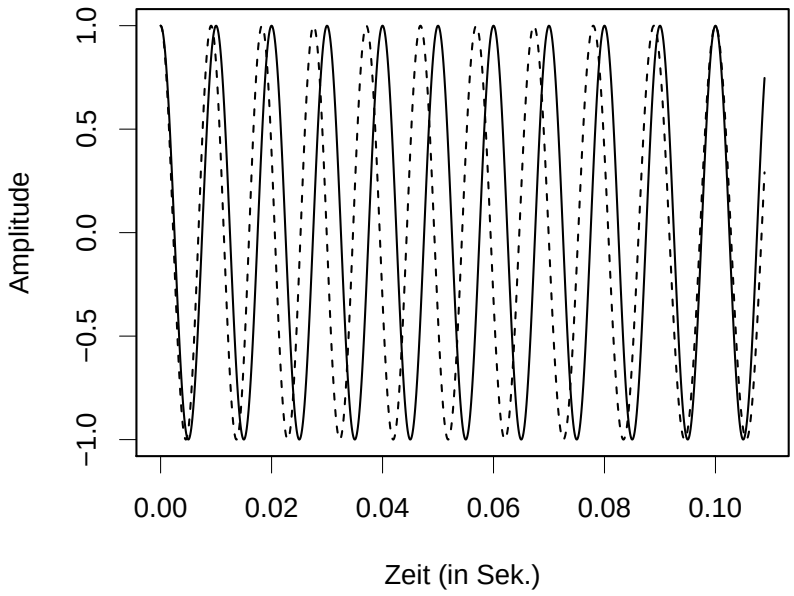


Abbildung 5.5: Schwingung bei 100 Hz Grundfrequenz ohne (durchgehend) und mit (gestrichelt, $f_v = 5, A_v = 2$) Vibrato

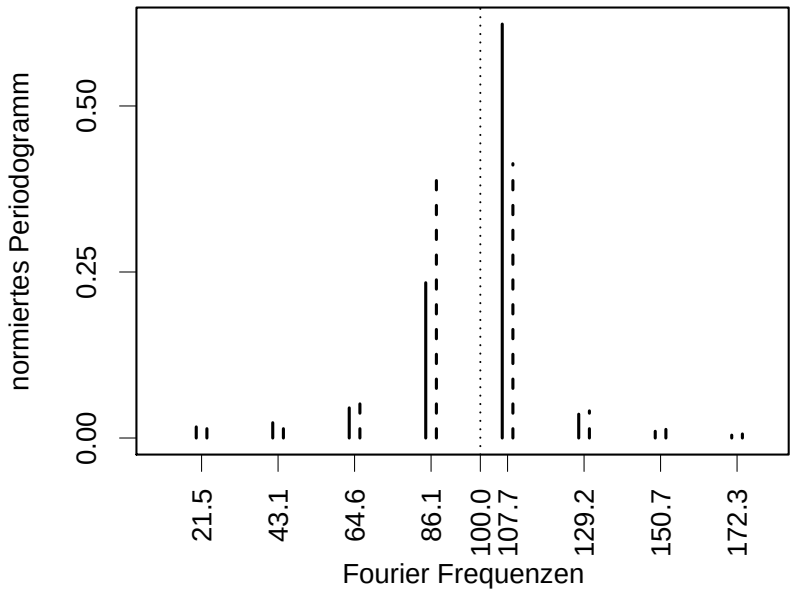


Abbildung 5.6: Periodogramme der Zeitreihen aus Abb. 5.5

## 5.2 Schätzungen im Zeit- und Frequenzbereich mittels Nelder-Mead Optimierung

### 5.2.1 Zeitbereich

Eine erste Idee der Parameterschätzung für das in Formel (5.1) spezifizierte Modell ist es, die Parameter simultan mit Hilfe des Verfahrens von Nelder und Mead (1965) direkt zu schätzen. Dazu wurde jeweils ein Teil (Fensterbreite $w$) der Originalzeitreihe $(x_t, \ldots, x_{t+w-1})$ hergenommen und versucht, die Quadratsumme der Differenz zwischen der Originalzeitreihe und einer mit dem Modell erzeugten Zeitreihe $(y_t, \ldots, y_{t+w-1})$ durch geeignete Wahl der Parameter $\vec{B}$, $\vec{\delta}$, $f_0$, $\vec{\phi}$, $A_v$, $f_v$, und $\phi_v$ zu minimieren:

$$\text{argmin}_{\vec{B},\vec{\delta},f_0,\vec{\phi},A_v,f_v,\phi_v} \sum_{j=1}^{w} (x_{t+j-1} - y_{t+j-1}(\vec{B}, \vec{\delta}, f_0, \vec{\phi}, A_v, f_v, \phi_v))^2.$$

Zur Vereinfachung der Notation wird hier die Vektorschreibweise eingeführt. Die Vektoren $\vec{B}$, $\vec{\delta}$ und $\vec{\phi}$ enthalten jeweils $H$ Elemente, also so viele, wie Partialtöne modelliert werden.

Es stellt sich jedoch heraus, dass die Struktur der Hyperebene, in der das globale Minimum über mehr als 7 Parameter gefunden werden muss, sehr schlecht konditioniert ist. Selbst bei Auswahl von nur zwei Dimensionen ähnelt sie einer Hügellandschaft über den gesamten Parameterraum. In Abbildung 5.7 ist beispielhaft der Fehler gegen die Parameter $f_0$ (Grundfrequenz) und $\phi_1$ (Phasenverschiebung der Grundtonschwingung) abgetragen.

Selbst bei hervorragenden Startwerten, die in der Nähe des globalen Optimums liegen, konvergierte der Simplex bei simulierten Reihen mit bekannten Parametern oft zu einem der vielen lokalen Optima, so dass der vor allem interessierende Parameter $f_0$ fast nie richtig geschätzt wurde. Auch die Verringerung der Parameter durch Wahl einer kleinen Anzahl an Partialtönen $H$ und das Festsetzen der Fre-

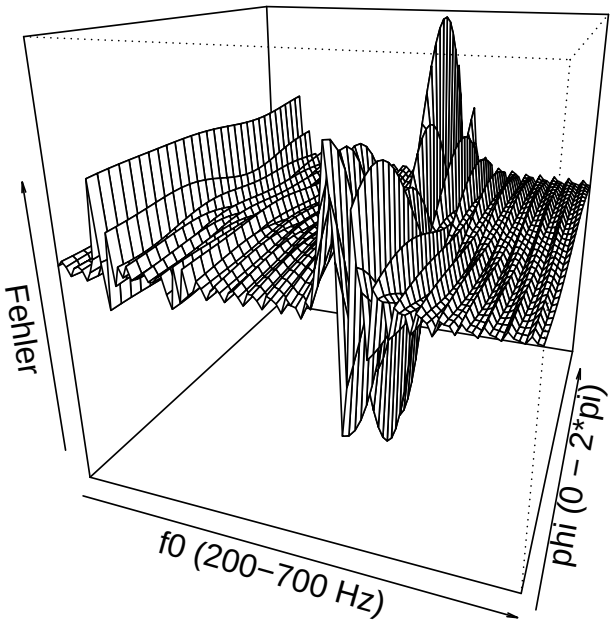


Abbildung 5.7: Hügellandschaft des Fehlers für Parameter $f_0$ und $\phi_1$

quenzverschiebungen $\vec{\delta} := \vec{0}$ konnte die Ergebnisse nicht verbessern. Die Wahl eines Gradientenverfahrens (BFGS) für die Optimierung zeigte ebenfalls keine Vorteile.

### 5.2.2 Frequenzbereich

Es besteht die Hoffnung, dass durch Übergang in den Frequenzbereich eine Verbesserung erzielt werden kann, da dort weniger Parameter zu schätzen sind als im Zeitbereich. Insbesondere gilt das für die Phasenverschiebungen $\vec{\phi}$, denn es ist (Brockwell und Davis, 1991)

$$P(\lambda_k) = \frac{1}{N}\left|\left|\sum_{t=1}^{N}(x_t e^{-it\lambda_k})\right| e^{i\phi}\right|^2 = \frac{1}{N}\left|\sum_{t=1}^{N}(x_t e^{-it\lambda_k})\right|^2,$$

weil $e^{i\phi}$ auf dem Einheitskreis liegt und daher gilt: $\left|e^{i\phi}\right| = 1$.

Für diesen Übergang in den Frequenzbereich wird das Periodogramm $P_{t,w}(\lambda, x)$ des Fensters der Größe $w$ der Originalzeitreihe $x$ berechnet:

$$P_{t,w}(\lambda, x) := P(\lambda, x_t, \ldots, x_{t+w-1}).$$

Analog wird auch bei der Parameteroptimierung stets das Periodogramm $P_{t,w}(\lambda, y)$ der Zeitreihe $y_t$ aus dem Modell berechnet. Beide Periodogramme werden jeweils auf Summe 1 normiert, so dass $\sum_\lambda P(\lambda, \cdot) = 1$. Nun wird die Quadratsumme der Differenzen dieser Periodogramme für die Optimierung der Parameter $\vec{B}$, $\vec{\delta}$ und $f_0$ verwendet:

$$\text{argmin}_{\vec{B},\vec{\delta},f_0} \sum_\lambda (P_{t,w}(\lambda, x) - P_{t,w}(\lambda, y))^2.$$

Damit Vibrato modelliert werden kann, also auch die Parameter $A_v$, $f_v$ und $\phi_v$, reicht es offensichtlich nicht aus, nur ein Periodogramm von einem Zeitabschnitt zu verwenden, da über den zeitlichen Verlauf der Tonhöhenänderung (also das Vibrato) beim Übergang in den Frequenzbereich gemittelt wird. Statt dessen werden mehrere Periodogramme benachbarter (oder besser sogar überlappender) Fenster mit äquidistanten Startzeitpunkten $t_1, t_2, \ldots, t_n$ hergenommen und alle Werte dieser Periodogramme als ein Vektor aufgefasst (hintereinander geschrieben):

$$P_{\vec{t},w}(\lambda, x) = (P_{t_1,w}(\lambda, x), P_{t_2,w}(\lambda, x), \ldots, P_{t_n,w}(\lambda, x))'.$$

Auf diese Weise wird die Tonhöhenveränderung also wieder schätzbar:

$$\text{argmin}_{\vec{B},\vec{\delta},f_0,A_v,f_v,\phi_v} \sum_\lambda \sum_{t=t_1}^{t_n} (P_{t,w}(\lambda, x) - P_{t,w}(\lambda, y))^2.$$

Mit geeigneten Startwerten, die etwa durch die in Kapitel 4 beschriebene Heuristik berechnet werden können, führt dieses Verfahren oft zu sehr genauen Schätzergebnissen. Für Details zur Genauigkeit sei auf Kapitel 6 verwiesen.

## 5.3 Schätzung mit Hilfe von Bayes Methoden

Bei einer stochastischen Erweiterung des Modells aus Formel (5.1),

$$\begin{aligned} y_t \quad = \sum_{h=1}^{H} \quad & B_h \cos\left[2\pi(h+\delta_h) f_0 t + \phi_h \right. \\ & \left. + (h+\delta_h) A_v \sin(2\pi f_v t + \phi_v)\right] + \varepsilon_t, \end{aligned}$$

zu einem hierarchischen Bayes Modell wird zusätzlich angenommen:

- $f_0$, Grundfrequenz, gleichverteilt auf dem Intervall $[0, 3000]$ Hz, das also mehr als den notwendigen Bereich umfasst (vgl. Tabelle 2.1, S. 8),
- $H - 1$, Anzahl der Obertöne, abgeschnitten Poisson verteilt mit Maximum 11 und Gamma$(H, 1)$ verteiltem Erwartungswert (Faustregel: je besser der Sänger, desto mehr Obertöne),
- $B_h$, Amplituden, normalverteilt mit Erwartungswert 0.5 und kleiner Gamma$(0.01, 0.01)$ verteilter Präzision (Kehrwert der Varianz, also große Varianz, da kaum Wissen über Stärke der Obertöne vorhanden ist),
- $\delta_h$, Frequenzverschiebungen, normalverteilt mit Erwartungswert 0 und großer Gamma$(100, 1)$ verteilter Präzision, da Frequenzverschiebungen nur sehr klein sind,
- $\phi_h$, Phasenverschiebungen, gleichverteilt auf dem Intervall $[-\pi/2, \pi/2)$, da keine Information über die Phase vorhanden ist,
- $f_v$, Vibratofrequenz, gleichverteilt auf dem Intervall $[0, 12]$ Hz (vgl. Abschnitt 2.1.4),
- $A_v$, Vibratoamplitude, normalverteilt mit Erwartungswert 0 und kleiner Gamma(0.01, 0.01) verteilter Präzision,
- $\phi_v$, Phasenverschiebung des Vibratos, gleichverteilt auf dem Intervall $[-\pi/2, \pi/2)$,
- $\varepsilon_t$, Modellfehler, normalverteilt mit Erwartungswert 0 und nicht zu großer Gamma$(0.5, 2)$ verteilter Präzision.

Die Wahl der einzelnen Verteilungen ist an Davy und Godsill (2002) angelehnt.

Die Grundidee dieser Modellierung ist die stochastische Suche nach den besten Koeffizienten in einem vorgegebenen Bereich mit unterschiedlichen Wahrscheinlichkeiten.

Beide, Bereich und Wahrscheinlichkeiten, werden durch die Verteilungen spezifiziert. Je weiter der Bereich, man sagt, je kleiner die Präzision, desto weniger legt man sich a-priori fest. Die Grundfrequenz und die Vibratofrequenz werden gleichverteilt in sinnvollen Frequenzbereichen angenommen, die Phasenverschiebungen gleichverteilt auf dem Einheitshalbkreis. Die Amplituden $B_h$ werden normalverteilt mit kleiner Präzision unterstellt, wobei der Erwartungswert auf 0.5 gesetzt wird als ein Wert zwischen Stille ($B_h = 0$ für alle $h$) und höchstmöglicher Lautstärke (z.B. $B_h = 1$ für $H = h = 1$). Diese höchstmögliche Lautstärke ist beschränkt, da die Zeitreihe $y_t$ so normiert ist, dass $y_t \in [-1, 1]$ ist für alle $t$.

Die Frequenzverschiebung sollte klein sein, also in der Nähe von Null mit großer Präzision. Der Modellfehler sollte nicht zu groß sein. Man beachte, dass dieses Modell auch die Anzahl Obertöne mitlernt. Dabei wird eine Poissonverteilung angenommen, deren Erwartungswert mit mittlerer Präzision in der Nähe eines vorgegebenen Wertes liegt. Die Schätzung der unbekannten Koeffizienten verläuft bei diesem Modell iterativ. Dabei wird durch die Modellierung erzwungen, dass sich der Erwartungswert der Anzahl Obertöne nur langsam ändert.

### 5.3.1 Modellschätzung mit Hilfe von BRugs

Zum Zweck der Modellschätzung des hier beschriebenen Modells in nicht allzu großer Rechenzeit wurde intensiv an dem R Paket *BRugs* (Thomas et al., 2006b) mitentwickelt, das ein Interface zwischen R (R Development Core Team, 2005) und OpenBUGS (Thomas, 2004) implementiert. Dieses Paket ermöglicht es, mit Hilfe von Funktionen aus R heraus OpenBUGS zu steuern. Es bietet also die transparente Einbindung der speziellen BUGS Funktionalität für MCMC sampling und hierarchische Modellierung in die Umgebung von R.

Im Detail stellt OpenBUGS mit Hilfe einer *dynamic link library* ein `.C()` Interface zu seiner *command language* bereit. Zu fast allen Funktionen (außer zur Steuerung der nicht benötigten GUI) dieser *command language* gibt es in BRugs eine korrespondierende R Funktion, die das Kommando und die jeweils anzugebenden Parameter an

OpenBUGS übergibt. Die Rückgabe von Information von OpenBUGS an R geschieht entweder auch über das `.C()` Interface oder mit Hilfe einer Pufferdatei.

Das *BRugs* Paket wurde auf CRAN[1], dem Comprehensive R Archive Network, veröffentlicht und kann in R unter Windows bei vorhandener Netzwerkverbindung einfach mit `install.packages("BRugs")` installiert und daraufhin mit `library("BRugs")` geladen werden. Erste Hilfe erhält man bei Eingabe von `?BRugs`. Weiterhin ist die Hilfe auch im Online-manual[2] einzusehen. Eine Auswahl von Hilfeseiten zu den wichtigsten Funktionen ist in Anhang A.2 abgedruckt.

Nur durch die Entwicklung von *BRugs* war es möglich, die in Kapitel 6 beschriebenen Abbruchkriterien für die Optimierung automatisch zu überprüfen, so dass Simulationen in der derartigen Größe möglich sind: Während das MCMC sampling von dem integrierten BUGS Teil des Pakets BRugs ausgeführt wird, wird das Stoppkriterium, das die Berechnung einer Regression voraussetzt, von R bestimmt. Sollte das Stoppkriterium nicht erfüllt sein, kann R den integrierten BUGS Teil veranlassen, weiter zu iterieren. In dem klassischen WinBUGS (Spiegelhalter et al., 2003) und selbst mit dem zuvor mitentwickelten R Paket R2WinBUGS (Sturtz et al., 2005) wäre das iterative Vorgehen unmöglich gewesen. Die Datenvorbereitung wurde stets mit Hilfe des Pakets tuneR (vgl. Kapitel 7) durchgeführt.

[1] http://CRAN.R-project.org

[2] http://CRAN.R-project.org/src/contrib/Descriptions/BRugs.html

# Kapitel 6

# Vergleich der Verfahren

Die verschiedenen vorgestellten Verfahren zur Grundfrequenzschätzung werden in diesem Kapitel verglichen. Die Grundfrequenzschätzung mittels Heuristik aus Abschnitt 4.1.2 wird verglichen mit der modellbasierten Schätzung per Optimierung im Frequenzbereich aus Abschnitt 5.2 und der Schätzung des hierarchischen Bayes Modells mit Hilfe von MCMC Verfahren aus Abschnitt 5.3. Wie die jeweiligen Berechnungen für den Vergleich der Schätzer genau erfolgen, wird in den entsprechenden Abschnitten 6.2 bis 6.4 beschrieben.

Für den Vergleich der Verfahren werden zunächst Klänge simuliert, damit die wahren Werte bekannt sind – bei Sängerinnen und Sängern ist der wahre Wert der Grundfrequenzen leider nicht bekannt. Die simulierten Klänge sollen möglichst gut Gesang verschiedener Stimmarten repräsentieren. Daher wird in Abschnitt 6.1 ein Versuchsplan aufgestellt, der verschiedene Faktoreinstellungen von Variablen annimmt, die für verschiedene Stimmen charakteristisch sind.

Schließlich werden in Abschnitt 6.5 die Ergebnisse der Vergleiche dargestellt. Wegen der hohen Komplexität des den modellbasierten Verfahren zugrundeliegenden Modells ist dabei die Güte nichtlinearer Lern- bzw. Schätzverfahren zur Bestimmung der unbekannten Koeffizienten $I$, $f_0$, $H$, $B_{h,i}$, $\delta_h$, $\phi_h$, $f_v$, $A_v$ und $\phi_v$ zu untersuchen

– siehe Formel (5.1) aus Kapitel 5.3:

$$\begin{aligned} y_t &= \sum_{h=1}^{H} \sum_{i=0}^{I} \Phi_i(t) B_{h,i} \cos\left[2\pi(h+\delta_h) f_0 t + \phi_h \right. \\ &\quad \left. + (h+\delta_h) A_v \sin(2\pi f_v t + \phi_v)\right] + \varepsilon_t. \end{aligned}$$

Da hier nur synthetische Klänge mit konstanter Lautstärke betrachtet werden, wird genau eine konstante Basisfunktion mit dem Wert 1 verwendet.

Insbesondere ist die Genauigkeit der Schätzung der Grundfrequenz von Interesse. Daneben ist interessant, ob das Vibrato gut geschätzt wurde und sich dessen Modellierung durch verbesserte Grundfrequenzschätzung ausgezahlt hat.

## 6.1 Versuchsplan

Zur Datengenerierung für den Vergleich wurde, um systematisch die ganze Breite sinnvoller Klänge abzudecken, ein vollfaktorieller statistischer Versuchsplan mit den folgenden 5 Variablen benutzt: Art der Sängerin (Profi oder Amateur), Tonhöhe (hoch oder niedrig, d.h. 1000 oder 250 Hz), Vibratofrequenz (5 oder 9 Hz), Quotient von Vibratoamplitude und Vibratofrequenz (5 oder 15), Phasenverschiebung des Vibrato (0 oder 3). Die Datengenerierung folgt damit auch der Formel (5.1). In dem Versuchsplan kommen alle Kombinationen der je 2 Faktoreinstellungen der 5 Variablen vor. Es werden also $2^5 = 32$ Zeitreihen untersucht. In weiteren 4 Versuchen ist die Vibratoamplitude auf 0 gesetzt worden, wobei auch Vibratofrequenz auf 1 Hertz und die zugehörige Phasenverschiebung des Vibrato o.B.d.A. auf 0 gesetzt wurden. Der vollständig aufgelistete Versuchsplan ist im Anhang A.3 abgedruckt.

Professionelle Sängerinnen wurden modelliert durch Grundfrequenz und zwei Obertöne mit $B_1 = 3$, $B_2 = 2$ und $B_3 = 1$, Amateure durch Grundfrequenz und einen Oberton mit $B_1 = 3$ und $B_2 = 1$. Phasenverschiebung und Rauschen wurden generell auf 0 gesetzt.

Die Datenvorbereitung (z.B. Simulation der Waves und deren Erzeugung) wurde auch hier mit Hilfe des Pakets tuneR (siehe Kapitel 7) durchgeführt.

Die unbekannten Koeffizienten der so generierten Zeitreihen wurden mit Hilfe von drei unterschiedlichen Verfahren (Heuristik, Nelder-Mead Optimierung und Bayes Verfahren) geschätzt. Für die Heuristik wurden jeweils einmal 512 und einmal 2048 Beobachtungen verwendet, Nelder-Mead Optimierung fand auf 2048 Beobachtungen statt und das Bayes Verfahren wurde wegen der hohen Rechenzeit ausschließlich auf 512 Beobachtungen angewendet.

## 6.2 Berechnung der Heuristik

Zunächst wurden heuristische Schätzungen ($f_{\text{heur}}$) der Grundfrequenz $f_0$ (siehe Kapitel 4, Formel (4.4)) auf Abschnitten von 512 Beobachtungen (Heuristik (1)) oder als der Median von 7 halb überlappenden Abschnitten von je 512 Beobachtungen berechnet (Heuristik (Median)). Vorversuche haben gezeigt, dass diese Heuristik die Grundfrequenz bei reinen Sinusschwingungen mit relevanten Frequenzen mit einem Fehler von höchstens 2 Hz bestimmt. Vibrato wird dabei im Fall von Heuristik (1) ignoriert und im Fall von Heuristik (Median) nur geglättet. Zum Beispiel würde mit der Heuristik (1) die Grundfrequenz im Fall des Periodogramms in Abbildung 5.6 (Seite 64) bei der Reihe ohne Vibrato mit 101 Hz nur wenig überschätzt, mit Vibrato (gestrichelt) mit 97 Hz aber deutlich unterschätzt. Es besteht aber die Hoffnung, dass eine Mitmodellierung des Vibratos zu einer besseren Schätzung der Grundfrequenz führt.

## 6.3 Schätzung der Modellparameter mit Nelder-Mead Optimierung

Als zweites wurden Schätzungen im Frequenzbereich (siehe Abschnitt 5.2) auf der Basis von Periodogrammen der 7 halb überlappenden Abschnitten von jeweils 512 Beobachtungen durchgeführt. Die resultierenden $1792 = 256 \cdot 7$ Werte an den Fourierfrequenzen bilden die Datenbasis für die Schätzung der unbekannten Koeffizienten des Zeitreihenmodells ohne stochastische Erweiterungen mit Hilfe des nichtlinearen Verfahrens von Nelder und Mead (1965). Dazu werden für feste Werte der Koeffizienten die Werte des Periodogramms an den Fourierfrequenzen berechnet und mit dem Periodogramm der Daten verglichen. Das geschieht iterativ so lange, bis sich keine Verbesserung mehr ergibt.

Die folgenden drei Startvektoren $S_k$, die jeweils aus Startwerten für die Parameter $\tilde{f}_0, B_h, f_v, A_v$ und $\phi_v$ bestehen, wurden dazu verwendet:

$$S_k = (\tilde{f_{0,k}}, B_h, f_v, A_v, \phi_v)'.$$

Dabei sind $B_h = 0.5$ für alle $h > 1$, $f_v = 7$, $A_v = 5$ und $\phi_v = 0$ fest. Die Startvektoren unterscheiden sich für $k = 1, 2, 3$ durch $\tilde{f_{0,k}}$ mit

$$\begin{aligned} \tilde{f_{0,1}} &= \text{Median}(f_{\text{heur}}) + 2, \\ \tilde{f_{0,2}} &= \text{Median}(f_{\text{heur}}) + 0, \\ \tilde{f_{0,3}} &= \text{Median}(f_{\text{heur}}) - 2. \end{aligned}$$

Als Startwerte für die Grundfrequenz werden also Werte in der Nähe des Schätzergebnisses der Heuristik verwendet.

In allen Fällen wurde ein Modell mit Grundton ($\tilde{f}$) und zwei Obertönen geschätzt. Da standardisierte Periodogramme genutzt wurden, wurde $B_1 = 1$ gesetzt, um die Identifizierbarkeit des Modells zu gewährleisten. Daher sind die geschätzten Amplituden $B_h$ für $h > 1$ als relativ zu $B_1$ zu betrachten. Die voreingestellten Stoppkrite-

rien der R Funktion `optim` wurden benutzt, wobei jedoch die maximale Anzahl der Iterationen auf 5000 erhöht wurde.

Als globales Ergebnis der Optimierung werden die Parameterschätzer desjenigen der drei Optimierungläufe verwendet, der den kleinsten Wert der Zielfunktion (siehe Abschnitt 5.2)

$$Z(\vec{B}, f_0, A_v, f_v, \phi_v) = \sum_{\lambda} \sum_{t=t_1}^{t_n} (P_{t,w}(\lambda, x) - P_{t,w}(\lambda, y(\vec{B}, f_0, A_v, f_v, \phi_v)))^2.$$

ergeben hat.

## 6.4 Schätzung des hierarchischen Bayes Modells

Für die Schätzung des hierarchischen Bayes Modells (siehe Kapitel 5.3) wurde eine Optimierung mit BUGS (Spiegelhalter et al., 2005), R und Unterstützung mit BRugs auf 512 Beobachtungen durchgeführt. Das verwendete BUGS Modell ist in Abbildung 6.1 angegeben.

Die Startvektoren der insgesamt 3 verwendeten Ketten $k$ sind identisch mit den Startvektoren der oben beschriebenen Optimierung (siehe Abschnitt 6.2), wobei allerdings $B_1$ als weiterer zu optimierender Parameter hinzukommt und die Anzahl der Obertöne $H - 1$ auch mitgeschätzt wird:

$$S_k = (\tilde{f_{0,k}}, B_h, f_v, A_v, \phi_v, H)'.$$

Alle Startwerte sind wie in Abschnitt 6.2 gesetzt und zusätzlich $B_1 = 1$ (nach wie vor ist $B_h = 0.5$ für alle $h > 1$) sowie $H = 2$.

Das übliche Vorgehen bei MCMC Simulationen ist, nach einem ausreichend langen Burn-In eine große Anzahl an Iterationen zu genereiren, dann Konvergenzkriterien zu überprüfen und anschließend daraus Schätzungen abzuleiten. Wegen der extrem großen Laufzeit (siehe Abschnitt 6.5) muss für die vorliegende Anwendung sowohl auf lange Burn-In Vorgänge als auch auf eine große Iterationsanzahl verzichtet werden.

```
model
{
    M0 ~ dpois(lambda)
    M <- M0 + 1
    lambda ~ dgamma(M, 1)
    for(m in 1:12){
        arg[m] <- (m + (delta[m] * step(m - 2 + 0.01))) *
            step(M - m + 0.01)
        a[m] <- (0.5 + ampla[m]) * step(M - m + 0.01)
        phi[m] ~ dunif(-1.57, 1.57)
        delta[m] ~ dnorm(mu, Sigma.delta)
        ampla[m] ~ dnorm(mua, Sigma.ampl)
    }
    for(t in 1:N){
        y[t] ~ dnorm(mue[t], sigma.epsilon.quadrat)
        mue[t] <- sum(Kosinus[,t])
        e[t] <- (y[t] - mue[t]) * (y[t] - mue[t])
        for(m in 1:12){
            vibrato[m,t] <- arg[m] * amplv *
                    sin(6.283185 * omegav / 11025 * t + phiv)
            Kosinus[m,t] <- a[m] *
                    cos(arg[m] * omega * t + phi[m] + vibrato[m,t])
        }
    }
    error <- sum(e[])
    sigma.epsilon.quadrat ~ dgamma(0.5, 2)
    amplv ~ dnorm(mua, Sigma.amplv)
    omega ~ dunif(0, 1.7)
    omegav ~ dunif(0, 12)
    phiv ~ dunif(-1.57, 1.57)
    mu ~ dnorm(0, 10000)
    mua ~ dnorm(0, 10000)
    Sigma.delta ~ dgamma(100, 1)
    Sigma.ampl ~ dgamma(0.01, 0.01)
    Sigma.amplv ~ dgamma(0.01, 0.01)
}
```

Abbildung 6.1: BUGS Modell zur Parameterschätzung

Ebenso signalisieren die üblichen Konvergenzkriterien erst sehr spät die Konvergenz der Ketten.

Ein geeignetes Stoppkriterium zum Beenden des MCMC Sampling muss also entwickelt werden. Folgendes Stoppkriterium hat sich in der Simulationsstudie bewährt: Es wird nach jeweils 100 Beobachtungen überprüft, ob eine lineare Regression der letzten 50 Residuen gegen die zugehörige Iterationsnummer keine signifikante negative Steigung zum Niveau $\alpha = 0.1$ mehr liefert. Weiterhin wird die maximale Anzahl an Iterationen auf 2000 begrenzt.

Als globales Ergebnis der Optimierung werden die Parameterwerte derjenigen Kette verwendet, die die kleinste Fehlerquadratsumme

$$E(\vec{B}, f_0, A_v, f_v, \phi_v, H) = \sum_t (x_t - y_t(\vec{B}, f_0, A_v, f_v, \phi_v, H))^2$$

zwischen Modellreihe und Originalreihe erzeugen.

## 6.5 Vergleich der Ergebnisse

Bei der Darstellung der Ergebnisse müssen nun die Schätzungen der Grund- und Vibratofrequenz betrachtet werden. Die Ergebnisse werden mit Hilfe der mittleren absoluten Abweichung (MAD) und der Wurzel der mittleren quadratischen Abweichung (RMSD) der geschätzten Grundfrequenz von der bekannten wahren (simulierten) Grundfrequenz verglichen (siehe Tabelle 6.1). Auch die sehr unterschiedliche Laufzeit der Algorithmen muss betrachtet werden.

Nur eine größere Anzahl an Beobachtungen kann zu einer erkennbaren Verbesserung führen. Dabei ist zu bedenken, dass die Optimierung mit BUGS auf dem verwendeten Compute-Server schon mit 512 Beobachtungen 31 Stunden für den gesamten Versuchsplan (36 Reihen) benötigt, also 31/36 Stunden pro Reihe. Für ein einminütiges Musikstück mit Samplingrate 11025, Fensterbreite 512 und halb überlappenden Fen-

Tabelle 6.1: Abweichungen (in cent) der geschätzten Grundfrequenz von der wahren Grundfrequenz für die verschiedenen Verfahren mit jeweils allen 36 Versuchen

| | $\hat{f}_0$ MAD | $\hat{f}_0$ RMSD | Laufzeit |
|---|---|---|---|
| Heur. (1) | 5.06 | 6.06 | $< 1$ Sek. |
| BUGS | 4.88 | 6.44 | 31 Std. |
| Heur. (Median) | 2.38 | 2.74 | 2 Sek. |
| NM (spektral) | 1.29 | 3.35 | 4 Std. |

stern ergäbe sich eine Rechenzeit von

$$60 \cdot \frac{11025}{512/2} \cdot \frac{31}{36} = 2225.098 \text{ Stunden},$$

also mehr als 3 Monaten.

Die Ergebnisse der Optimierungen werden mit den Ergebnissen der heuristischen Grundfrequenzschätzung mit Hilfe von Boxplots in Abbildung 6.2 verglichen, wobei dort die Differenzen zum wahren Wert abgetragen worden sind. Die horizontalen Linien an den Stellen $\pm 50$ cents ($= \pm 0.5$ Halbtöne) entsprechen den natürlichen Schwellenwerten zum nächsten Halbton (nach oben bzw. unten). Die auf 2048 Beobachtungen basierende Heuristik (Median) führt zu perfekter Klassifikation, denn keine Punkte liegen außerhalb des Intervalls $[-50, 50]$. Die (spektrale) Nelder-Mead Optimierung ist in der Schätzgenauigkeit Grundfrequenz zwar meistens viel exakter (fast alle Werte nahe 0), liefert in seltenen Fällen aber trotzdem falsche Klassifikationsergebnisse (zwei Werte größer als 50). Die BUGS Ergebnisse sind vergleichbar mit den Ergebnissen der auf 512 Beobachtungen basierenden Heuristik.

Die Schätzungen der Vibratofrequenz werden in Abbildung 6.3 verglichen. Um zur Übersichtlichkeit beizutragen, wurden die Werte für die wahre Vibratofrequenz auf der y-Achse etwas über und unter die jeweilige Ideallinie gezeichnet. Die (spektrale) Nelder-Mead Optimierung ist sehr gut im Fall von 9 Hertz, aber inakzeptabel für wahre 5 Hertz. Auch die BUGS Ergebnisse variieren bei einer wahren Frequenz von 5 Hertz weniger als die Nelder-Mead Ergebnisse. Die Heuristik bleibt hier außen vor, da sie keine Schätzwerte für die Vibratofrequenz liefert.

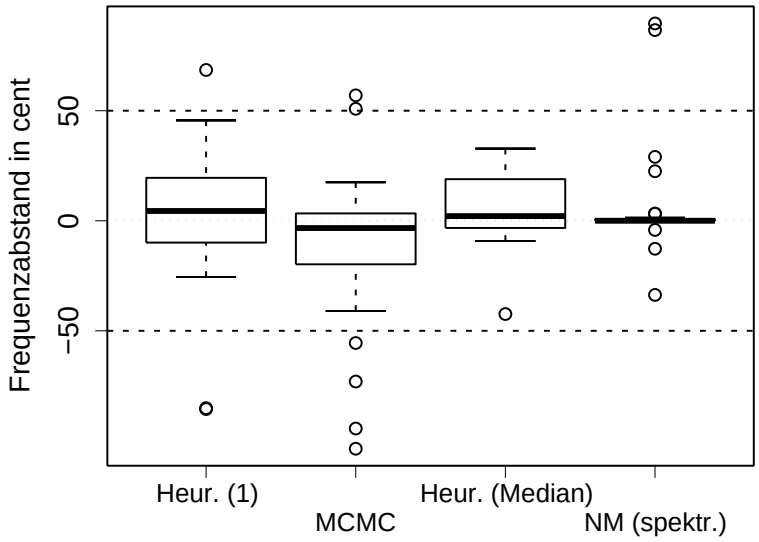


Abbildung 6.2: Boxplots der Abweichungen der geschätzten Grundfrequenzen von den wahren Grundfrequenzen, jeweils gemäß den Einstellungen der jeweils 36 Versuche

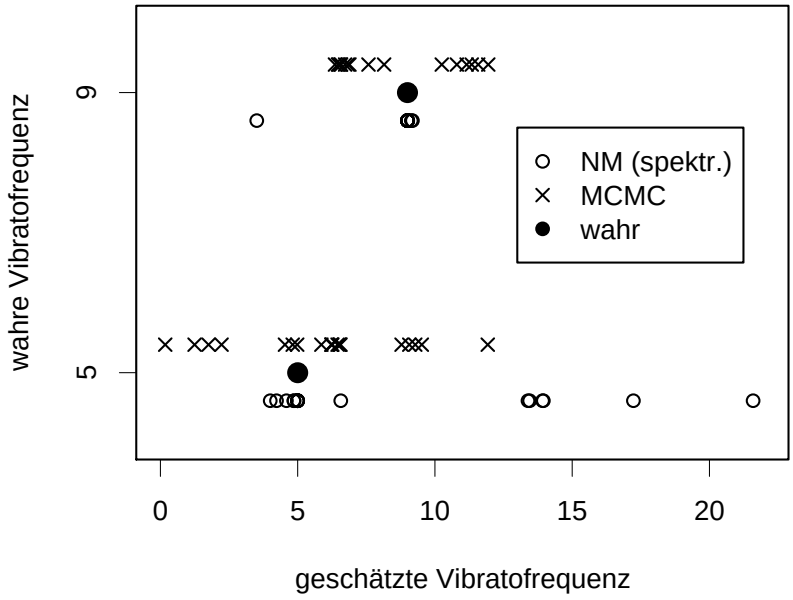


Abbildung 6.3: Schätzung der Vibratofrequenz

Wegen der insgesamt enttäuschenden Verbesserung der Genauigkeit der Schätzung der Grundfrequenz durch die Modell basierten Verfahren im Vergleich zur Heuristik unter gleichzeitig beträchtlicher Zunahme der Rechenzeit wird hier für praktische Anwendungen die Heuristik empfohlen.

Die Idee, durch Mitschätzung der Vibratofrequenz eine Verbesserung der Schätzung zu erreichen, kann offensichtlich erst unter Einsatz sehr großer Datenmengen zum Erfolg führen, denn die Vibratofrequenzschätzung bei sehr niedriger Vibratofrequenz ist noch sehr schlecht. Das kann auch leicht begründet werden: Die vom Bayes Verfahren verwendete Datenmenge ist mit 512/11025 nur gerade etwa 1/20 Sekunde lang, also nur 1/4 einer mit 5 Hertz verlaufenden Vibratoschwingung. Eine Erhöhung der Datenmenge würde die Rechenzeit allerdings noch weiter erhöhen. Die bei der Nelder-Mead Optimierung verwendete Datenmenge, $2048/11025 \approx 1/6$ Sekunde, scheint ausreichend für die schnelle Vibratofrequenz von 9 Hertz zu sein, jedoch nicht für die langsame von 5 Hertz. Für die schnelle Frequenz überdecken die Daten eine Vibratoschwingung komplett, während das für die langsame nicht der Fall ist. Eine Ausweitung der Datenbasis ist allerdings recht riskant, weil dadurch bei schnellen Musikstücken sogar mehr als zwei verschiedene Töne überdeckt werden könnten, die Schätzung dann also aus anderem Grunde ungenau oder verzerrt wird.

Der Vergleich führt zu der Schlussfolgerung, dass die Heuristik zur Grundfrequenzschätzung ähnlich genau wie die deutlich komplexeren Methoden ist, dabei aber deutlich schneller, so dass in der Praxis oder gar bei der Online-Transkription der auf der Heuristik basierende Transkriptionsalgorithmus zu bevorzugen ist.

# Kapitel 7

# tuneR – Software für die Musikanalyse

Das R (R Development Core Team, 2005) Paket *tuneR* (Ligges, 2005b) stellt ein Framework für die statistische Analyse von Musikdaten mit Hilfe von R bereit. Es wurde zur Unterstützung der Entwicklung der in dieser Dissertation vorgestellten Methoden programmiert. Der Einsatzbereich des Pakets ist jedoch wegen der Offenheit des Systems und der Allgemeinheit der in dem Paket implementierten Datenklassen deutlich größer. Klassen und Methoden sind in formaler Form nach dem S4 Standard (Chambers, 1998) implementiert worden. Diese Implementierungsmöglichkeit mit Hilfe von S4 wird anhand von Beispielen des *tuneR* Pakets in Ligges (2005a) beschrieben.

Das *tuneR* Paket wurde auf CRAN[1], dem Comprehensive R Archive Network, veröffentlicht und kann in R bei vorhandener Netzwerkverbindung einfach mit `install.packages("tuneR")` installiert und daraufhin mit `library(tuneR)` geladen werden. Erste Hilfe erhält man bei Eingabe von `?tuneR`.

Im Folgenden werden eine Reihe von Funktionen kurz vorgestellt. Ausführliche Hilfe erhält man in R wie üblich durch Eingabe des Fragezeichens gefolgt vom Funktions-

[1] http://CRAN.R-project.org

namen. Weiterhin ist die Hilfe auch im Online-manual (Ligges, 2005b)[2] einzusehen. Hilfeseiten zu den wichtigsten Funktionen sind auch in Anhang A.1 abgedruckt.

## 7.1 Die Klasse `Wave`

Wie bereits in Abschnitt 2.1.1 beschrieben wurde, ist das *Wave* Format (Microsoft Corporation, 1991) eine auf dem Computer „natürliche“ Repräsentation für Musikzeitreihen. Um Musikzeitreihen mit Hilfe von R untersuchen zu können, sollte also auch in R ein Objekt die Daten repräsentieren, das dem Wave Format ähnlich ist. Auf diese Weise ist es dann leicht möglich, entsprechende Daten in R zu importieren und auch wieder nach Bearbeitung zu schreiben, so dass sie angespielt werden können. Daten können, beispielsweise zu Simulationszwecken, auch neu erzeugt werden.

Für all dies wurde die S4 Klasse `Wave` implementiert, die das Wave Format mit folgenden Slots repräsentiert:

`left`: numerischer Vektor für den linken Kanal

`right`: numerischer Vektor für den rechten Kanal

`stereo`: logischer Skalar zur Anzeige, ob Mono (nur linker Kanal) oder Stereo vorliegt (z.B. `TRUE` bei CD Aufnahme)

`samp.rate`: numerischer Skalar, der die zugrundeliegende Samplingrate angibt (z.B. 44100 für CD Qualität)

`bit`: numerischer Skalar, der die Auflösung der Wave angibt (z.B. 16 bit für CD Qualität)

Ein Objekt der Klasse `Wave` kann direkt mit der Konstruktions-Funktion `Wave()` erzeugt werden. Alternativ lassen sich `Waves` bestimmter Wellenformen auch komfortabel mit folgenden Meta-Funktionen erzeugen:

[2] http://CRAN.R-project.org/src/contrib/Descriptions/tuneR.html

`noise()`: weißes oder rosa Rauschen

`sawtooth()`: Sägezahn-Schwingung

`silence()`: Stille (also konstante Werte auf der 0 Linie)

`sine()`: reine Sinus-Schwingung

`square()`: Rechteck-Schwingung

Da man zur Datenanalyse Wave Dateien einlesen möchte, steht die Funktion `readWave()` bereit. Das Schreiben eines erzeugten (z.B. simulierten) `Wave` Objekts in eine Wave Datei geschieht mit Hilfe von `writeWave()`.

Zur Daten(vor)verarbeitung von `Wave` Objekten stehen Funktionen zur Verfügung, die häufig für Musikzeitreihen notwendige Arbeiten in einem Schritt oder wenigen Schritten möglich machen. Dazu gehören zum Beispiel:

`Arith`: Es stehen für passende `Wave` Objekte die üblichen arithmetischen Operatoren (z.B. `"+"`) zur Verfügung.

`as()`: Umwandlung von bzw. in andere Objekte

`bind()`: Verbinden mehrerer `Wave` Objekte, siehe auch `prepComb()`

`channel()`: Extraktion eines Kanals oder beider Kanäle, auch ein Tausch der Kanäle ist möglich

`downsample()`: Datenreduzierung durch Umformung auf eine niedrigere Samplingrate

`extractWave()` und `"["()`: Extraktion eines Teils der Wave in zeitlicher Richtung, Angabe ist in den Einheiten „Sekunden“ oder „Samples“ möglich, ebenso ist die interaktive Verwendung möglich (Aufforderung zum Klicken in den Plot erscheint)

`mono()`: Umwandlung in eine Mono `Wave` (ein Kanal oder Mittelung beider Kanäle)

`normalize()`: Funktion zur Normierung der `Wave` auf die Intervalle [0,1] (reelle Zahlen), [0,254] (für 8-bit, ganze Zahlen) oder [-32767,32767] (für 16-bit, ganze Zahlen); optional wird das Signal auch auf die Nulllinie (127 für 8-bit, 0 für 16-bit) der entsprechenden Darstellungsform zentriert

`noSilence()`: automatisches Abschneiden von Stille

`panorama()`: Mischung der Kanäle durch Gewichtung

`prepComb()`: bereitet das Verbinden von `Wave` Objekten vor, so dass „Knacken“ durch extreme Amplitudensprünge vermieden wird

`stereo()`: zwei Mono `Wave` Objekte zu einem Stereo `Wave` Objekt zusammenfügen

Eine Reihe weiterer Funktionen ermöglicht es, `Wave` Objekte geeignet darzustellen:

`play()`: `Wave` Objekt mit einem geeigneten Wiedergabegerät abspielen (`setWavPlayer()` und `getWavPlayer()` sind Hilfsfunktionen)

`plot()`: Grafik mit allen vorhandenen Kanälen erzeugen, dabei wird die Datenmenge in Vektorgrafiken durch geschicktes Auslassen von Überlagerungen reduziert und die Grafikerzeugung beschleunigt

`show()` und `print()`: Kurzinformationen in der Konsole anzeigen

`summary()`: etwas ausführlichere Information in der Konsole ausgeben

Die hier genannten Funktionen zur Datenverarbeitung von Waves bieten für die in dieser Arbeit vorgestellten Analysen ein mehr als ausreichendes Handwerkszeug. Sollten weitere Funktionen gebraucht werden, so können diese sehr einfach in dem Framework ergänzt werden. Insbesondere kann die Klasse `Wave` mit Hilfe von S4 Programmiertechniken sehr einfach erweitert werden, falls eine erweiterte Repräsentation erforderlich ist.

## 7.2 Die Klassen Wspec und WspecMat

Ein übliches Hilfsmittel für die Frequenzanalyse ist der Übergang vom Zeit- in den Frequenzbereich mit Hilfe der schnellen Fouriertransformation. Dabei ist es gerade bei Musik- und Sprachzeitreihen üblich, die *Short Time Fast Fourier Transformation* (siehe dazu auch Abschnitt 4.1.1) zu verwenden, d.h. Fouriertransformationen (siehe Abschnitt 2.2.2) auf Fenstern, wie es auch in dieser Arbeit immer wieder geschehen ist. Dazu wurden die S4 Klassen `Wspec` und `WspecMat` implementiert, die die Periodogramme bzw. das Spektrogramm geeignet repräsentieren. Dazu enthalten diese Klassen folgende Slots:

`freq`: numerischer Vektor der Fourierfrequenzen

`spec`: für `Wspec` Objekte eine Liste mit einem Vektor der Schätzungen pro Fenster; für `WspecMat` Objekte eine numerische Matrix der Schätzergebnisse (zur Darstellung in einem Spektrogramm)

`kernel`: Kern, der für die Glättung verwendet worden ist

`df`: Die Verteilung der Spektraldichteschätzung kann mit einer $\chi^2$-Verteilung mit dieser Anzahl Freiheitsgrade approximiert werden.

`taper`: numerischer Wert der Stärke des „tapering“ am Rand des Fensters

`width`: Breite eines Fensters in Anzahl Samples

`overlap`: Überlappung aneinander grenzender Fenster

`normalize`: logischer Wert, im Fall von `TRUE` wurden die Periodogramme bereits auf Summe 1 normiert

`starts`: Index der Startwerte auf der linken Seite der Fenster

`stereo`, `samp.rate`: wird aus dem entsprechenden `Wave` Objekt übernommen, aus dem die Periodogramme berechnet werden

`variance`: numerischer Vektor der Varianzen der Fenster (ein Maß für Lautstärke)

`energy`: numerischer Vektor der Energiewerte der Fenster (anderes Maß für Lautstärke, siehe Formel (4.17))

Ein Objekt der Klasse `Wspec` wird in der Regel mit Hilfe der Funktion `periodogram()` unter Angabe geeigneter Argumente zur Beschreibung der Details aus einem `Wave` Objekt erzeugt. Diese Funktion erzeugt dabei automatisch die geforderte Fensterung und berechnet die Fouriertransformationen, falls gefordert auch nach Anwendung von Glättern und „tapering“, auf allen Fenstern. Beispielsweise erzeugt der Befehl

```
R> WspecObjekt <- periodogram(WaveObjekt, width = 512, overlap = 256,
+                             normalize = TRUE)
```

ein `Wspec` Objekt, das Periodogramme des `WaveObjekt` enthält. Und zwar werden mit dem Aufruf die Fouriertransformationen auf Fenstern mit einer Breite von 512 Samples bei halber Überlappung (256 Samples) berechnet und auf Summe 1 normiert.

Eine Reihe weiterer Funktionen ermöglicht es auch hier, `Wspec` Objekte geeignet darzustellen:

`image()`: Grafik mit Spektrogramm erzeugen

`plot()`: Grafik mit Periodogramm erzeugen unter Angabe der Nummer des Periodogramms

`show()` und `print()`: Kurzinformationen in der Konsole anzeigen

`summary()`: etwas ausführlichere Information in der Konsole ausgeben

## 7.3 Auf dem Weg zur Note

Aus einem `Wspec` Objekt können mit der Funktion `FF()` (und der Hilfsfunktion `FFpure()`) die Grundfrequenzen von Tönen geschätzt werden, die zu jeweils einem Periodogramm gehören. Dazu wird die in Kapitel 4 beschriebene Heuristik verwendet.

Eine Klassifikation, die jeder Grundfrequenz einen Notenwert zuordnet, wird durch die Funktion `noteFromFF()` durchgeführt, die dabei u.a. auf die Notentafel mit `notenames()` zurückgreift. Idealerweise wird anschließend die Funktion `smoother()`, also Glättung, zur Entfernung von Ausreißern verwendet. Die so geglättete Reihe kann nun durch `melodyplot()` visualisiert werden.

Eine statische Quantisierung kann von `quantize()` erledigt werden mit entsprechender Visualisierung mit Hilfe von `quantplot()`. Das abschließende Zusammenfassen zu Noten erfolgt mit `quantMerge()`. Zur Übersetzung in Notenschrift wird die Funktion `lilyinput()` von Preusser et al. (2002) verwendet, die eine für das Musiksatzsystem LilyPond (Nienhuys et al., 2005) geeignete Eingabe erzeugt.

## 7.4 Beispielsitzung mit tuneR zur Transkription

Es folgt die Beispielsitzung einer Transkription mit tuneR. Dazu sind die verwendeten Daten hier einfach zwei simulierte kurze Schwingungen, so dass keine großen Datenmengen zur Anwendung benötigt werden. Die Funktionsweise wird durch Kommentare (mit „`#`“ beginnend) im Code verdeutlicht. Die im Code erzeugten Grafiken sind auch im Code erklärt und in Abbildungen 7.1 bis 7.4 am Ende des Beispiels abgedruckt. Dabei sind die Beschriftungen mit Absicht unverändert belassen worden, um die Resultate des Codes zu zeigen. Leicht angepasst wurden hingegen die Liniendicken, Schriftgrößen und Farben für die bessere Lesbarkeit im Schwarzweißdruck.

```
R> library(tuneR) # Laden von tuneR
R> ## Erzeugung eines Mono Wave Objekts (10 Sek.) mit einer
R> ## Sinus Schwingungen (440Hz gefolgt von 220Hz):
R> Wobj <- bind(sine(440, bit = 16, duration = 5, xunit = "time"),
+               sine(220, bit = 16, duration = 5, xunit = "time"))
R> show(Wobj)             # Anzeigen der Information:

    Wave Object
            Number of Samples:      441002
            Duration (seconds):     10
            Samplingrate (Hertz):   44100
            Channels (Mono/Stereo): Mono
            Bit (8/16):             16

R> ## es wird nur ein schwarzer Kasten zu sehen sein,
R> ## denn es gibt 5*220 + 5*440 = 3300 Schwingungen
R> ## mit voller Amplitude:
R> plot(Wobj)                                       # Abb. 7.1 (a)
R> ## Verkleinerung des Ausschnitts, damit etwas zu sehen ist:
R> plot(extractWave(Wobj, from = 1, to = 500))     # Abb. 7.1 (b)
R> ## leider kann kein Klang gedruckt werden:
R> play(Wobj)

R> ## Wave object in eine temporäre Wave Datei schreiben:
R> tmpfile <- file.path(tempdir(), "testfile.wav")
R> writeWave(Wobj, tmpfile)
R> ## und wieder einlesen:
R> Wobj2 <- readWave(tmpfile)

R> ## nur den linken Kanal verwenden:
R> Wobjm <- mono(Wobj, "left")
```

```
R> ## 11025 Samples/Sekunde reichen aus:
R> Wobjm11 <- downsample(Wobjm, 11025)
R> ## für interaktives Extrahieren eines Teils des Signals
R> ## auf linke/rechte Seite des Ausschnitts im Plot klicken:
R> Wobjm11s <- extractWave(Wobjm11)
R> ## oder Grenzen reproduzierbar angeben:
R> Wobjm11s <- extractWave(Wobjm11, from=101, to=110150)

R> ## Berechnung von Periodogrammen auf Fenstern der Breite 1024
R> ## mit einer Überlappung von 512 Beobachtungen:
R> WspecObject <- periodogram(Wobjm11s, normalize = TRUE,
+                             width = 1024, overlap = 512)
R> ## Visualisierung des ersten Periodogramms:
R> plot(WspecObject, xlim = c(0, 2000), which = 1) # Abb. 7.2 (a)
R> ## Visualisierung des ganzen Spektrogramms:
R> image(WspecObject, ylim = c(0, 1000))           # Abb. 7.2 (b)

R> ## Schätzung der Grundfrequenzen:
R> ff <- FF(WspecObject)
R> print(ff)       # Anzeigen der Werte (Ausgabe gekürzt):
   [1] 440.6094 440.6011 ...
 [211] ... 219.4161 219.4883

R> ## Klassifikation der Noten gegeben a'=440:
R> notes <- noteFromFF(ff, 440)
R> snotes <- smoother(notes) # Glättung
R> ## Ergebnis sollte 0 sein für "a'" und -12 für "a"
R> print(snotes)   # Anzeigen der Werte (Ausgabe gekürzt):
   [1]   0   0   0   0   ...
 [208] ... -12 -12 -12 -12
```

```
R> ## Grafik der Melodie und Energiekurve:
R> melodyplot(WspecObject, snotes)                  # Abb. 7.3 (a)

R> ## Quantisierung (in 8 Teile):
R> qnotes <- quantize(snotes, WspecObject@energy, parts = 8)
R> ## Grafik der quantisierten Melodie, der Energiekurve
R> ## und der erwarteten Werte:                     # Abb. 7.3 (b)
R> quantplot(qnotes, expected = rep(c(0, -12), each = 4), bars = 2)

R> ## Vorbereitung für LilyPond:
R> qlily <- quantMerge(snotes, 4, 4, 2)
R> ## Also Ausgabe von qlily in C-Dur im Violinschlüssel, 4/4 Takt,
R> ## mit Abschlussbalken in Datei "tuneR.ly"      # Abb. 7.4:
R> lilyinput(qlily, file = "tuneR.ly", Major = TRUE, key = "c",
+      clef = "treble", time = "4/4", endbar = TRUE)
```

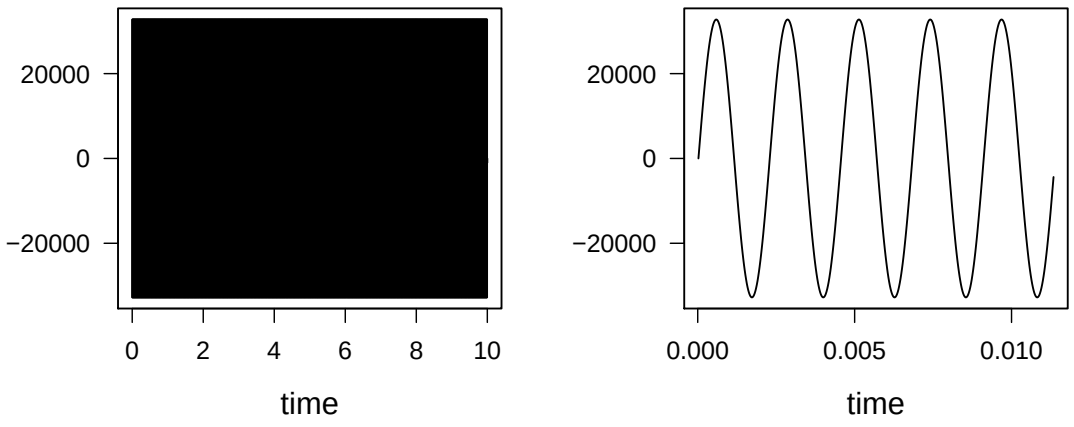


Abbildung 7.1: aus der Beispielsitzung: (a) ganze Zeitreihe und (b) ein kleiner Ausschnitt

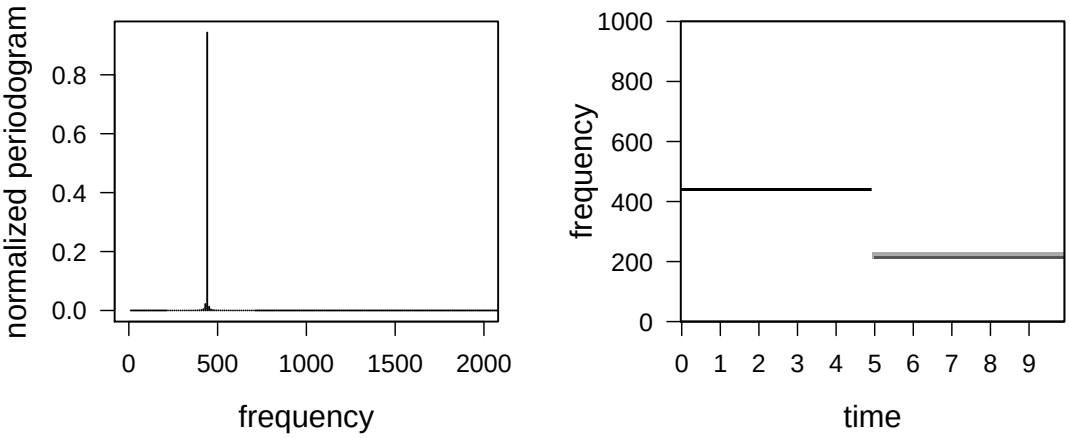


Abbildung 7.2: aus der Beispielsitzung: (a) Periodogramm und (b) vollständiges Spektrogramm (zunächst (440Hz) ist nur eine Frequenz stark beteiligt, später (220Hz) sind 2 Frequenzen mit geringeren Gewichten beteiligt)

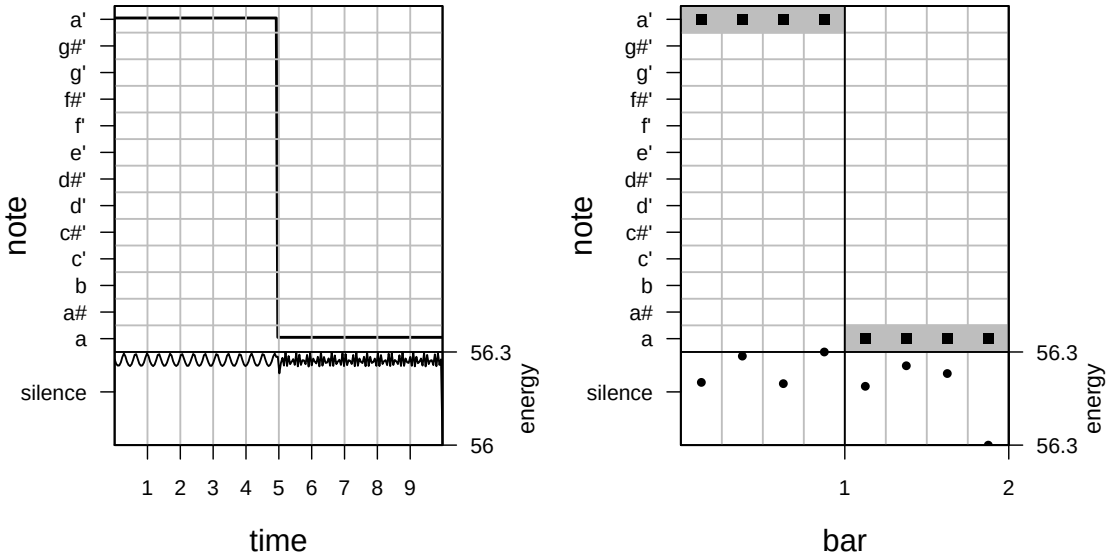


Abbildung 7.3: aus der Beispielsitzung: geschätzter Tonhöhenverlauf (a) vor und (b) nach der Quantisierung

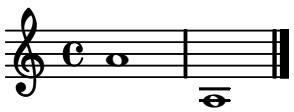

Abbildung 7.4: aus der Beispielsitzung: von LilyPond aus der Datei `tuneR.ly` erzeugte Noten

# Kapitel 8

# Zusammenfassung und Ausblick

Es wurden sowohl eine Heuristik (siehe Kapitel 4) als auch ein allgemeines Modell (siehe Kapitel 5) zur Darstellung und Schätzung von monophonem Klang entwickelt. Die unbekannten Koeffizienten können mit jeweils verschiedenen Techniken geschätzt werden.

Bei dem Modell zeigt sich das auf MCMC sampling beruhende Bayes Verfahren als genau, ist aber wegen der sehr großen Computer-intensiven Berechnungen sehr langsam (siehe Kapitel 6). Rechenzeiten von mehr als einem Tag für die Transkription eines Liedes von einer Minute Länge sind sicher nicht akzeptabel. Für zukünftige Arbeiten im Bereich der Transkription polyphoner Musikzeitreihen könnte der Ansatz allerdings einer aus nur recht wenigen sich anbietenden Möglichkeiten sein, denn das Modell aus Kapitel 5 ist sehr einfach um weitere Parameter erweiterbar, so dass weitere gleichzeitig erklingende Töne mitmodelliert werden können. Zur Reduzierung der Rechenzeit könnte die Implementierung eigener Sampler weiterhelfen, die für das Problem besser angepasst sind als der von BUGS automatisch gewählte Sampler.

Die Optimierung mit Hilfe des Nelder-Mead Verfahrens auf der Originalreihe im Zeitbereich wurde sogleich als zu anfällig für Konvergenz zu lokalen Optima erkannt. Die Optimierung im Frequenzbereich war hingegen deutlich besser. Die Fehler in der Grundfrequenzschätzung sind hier meist deutlich kleiner als bei der Heuristik

und dem Bayes Verfahren. Leider benötigt auch dieses Optimierungsverfahren so deutlich länger als die Heuristik, dass man sich aus praktischen Überlegungen für die Heuristik entscheiden wird.

Die in Kapitel 4 vorgestellte Heuristik ist nämlich nicht nur in extrem kurzer Zeit berechenbar, sondern sie liefert auch so genaue Schätzergebnisse der Grundfrequenz, dass bei den in Kapitel 6 vorgestellten simulierten Daten eine aus der Grundfrequenzschätzung hervorgehende Klassifikation der Notenhöhe ohne Fehler blieb. Sogar die Transkription echter Gesangsdaten gelang für eine Sängerin und einen Sänger mit einer Fehlerrate von jeweils nur 1.7%. Die Fehlerrate von 17% wurde in nur einem Beispiel überschritten. In diesen Fehlerraten sind dabei nicht nur die durch den Transkriptionsalgorithmus verursachten Fehler enthalten, sondern auch individuelle Fehler der Sängerinnen und Sänger, da die Fehlerraten durch Vergleich mit der Originalnotation ermittelt wurden.

Die Schätzverfahren zur Bestimmung der Vibratofrequenz sind allesamt nicht ganz zufriedenstellend, abgesehen von dem Schätzer auf spektralen Daten für eine recht hohe Vibratofrequenz. Letzterer wird voraussichtlich durch Hinzunahme weiterer gefensterter Abschnitte auch für geringe Vibratofrequenzen bessere Schätzungen für die Vibratofrequenz und damit auch für die Grundfrequenz liefern. Als Kosten stehen dem noch längere Rechenzeiten gegenüber.

Die Forschung auf dem Gebiet der Transkription von Musikzeitreihen muss sicherlich in Zukunft weitergeführt werden. Für ein vollständiges autonomes Transkriptionsverfahren fehlt die Hinzunahme eines Quantisierers, der nicht statisch ist, sondern sich an Tempoänderungen anpasst (siehe Kapitel 3). Leider sind fast alle in der Literatur beschriebenen Quantisierer für Schlag- oder Zupfinstrumente entwickelt worden, die zum Tonbeginn einen plötzlichen Amplitudenanstieg im Signal auslösen und für Gesangszeitreihen damit ungeeignet sind.

Ein weiterer Punkt ist die Modellierung von Pausen, also Stille oder leisem Rauschen. Bei Pausen zeigt das Periodogramm der Zeitreihe typischerweise rosa Rauschen (siehe Abschnitt 2.1.5). Durch Entwicklung geeigneter Tests könnte versucht werden,

das Signal harmonischer Schwingungen gegenüber rosa Rauschen zu identifizieren.

Insgesamt zeigt das hier vorgestellte Transkriptionsverfahren mittels heuristischer Grundfrequenzschätzung und statischer Quantisierung sehr gute Ergebnisse. Der entwickelte Algorithmus ist ein sehr hilfreiches Mittel zur Unterstützung und Arbeitserleichterung bei manueller Transkription. Darüber hinaus wurden Grundlagen für die Modellierung, Schätzung und Transkription bei polyphonen Musikzeitreihen gelegt.

Das mitentwickelte R Paket *BRugs* ermöglicht die transparente Einbindung der speziellen BUGS Funktionalität für MCMC Sampling und hierarchische Modellierung in die Umgebung von R. Unter den BUGS Anwendern wurde dieses Paket mit großer Resonanz aufgenommen.

Nicht zuletzt bietet das für die Forschungsarbeiten entwickelte R Paket *tuneR* (siehe Kapitel 7) eine Umgebung für die einfache und komfortable Analyse von Musikzeitreihen mit Hilfe statistischer Verfahren. Das Paket hat inzwischen zahlreiche Nutzer, die es unter anderem auch zur Sprachanalyse verwenden oder in Vorlesungen einsetzen.

# Anhang

## A.1 Auszug der Dokumentation des R Pakets tuneR

Der folgende englischsprachige Auszug aus der Original-Dokumentation des R-Pakets *tuneR* (Stand: Version 0.2-1, vgl. Kapitel 7) wurde aus Gründen der Platzersparnis leicht gekürzt und ist aus demselben Grunde in kleinerer Schrift und mit reduziertem Zeilenabstand abgedruckt. Für die vollständige Hilfe wird auf Ligges (2005b)[1] verwiesen.

---

`melodyplot` *Plotting a melody*

---

### Description

Plot a observed melody and (optional) an expected melody, as well as corresponding energy values (corresponding to the loudness of the sound).

### Usage

```
melodyplot(object, observed, expected = NULL, bars = NULL,
    main = NULL, xlab = NULL, ylab = "note", xlim = NULL, ylim = NULL,
    observedcol = "red", expectedcol = "grey", gridcol = "grey",
    lwd = 2, las = 1, cex.axis = 0.9, mar = c(5, 4, 4, 4) + 0.1,
```

[1] http://CRAN.R-project.org/src/contrib/Descriptions/tuneR.html

```
notenames = NULL, silence = "silence", plotenergy = TRUE, ...,
axispar = list(ax1 = list(side=1),
               ax2 = list(side=2),
               ax4 = list(side=4)),
boxpar = list(),
energylabel = list(text = "energy", side = 4, line = 2.5,
                   at = rg.s-0.25, las = 3),
energypar = list(),
expectedpar = list(),
gridpar = list(col = gridcol),
observedpar = list(col = observedcol, lwd = 2))
```

**Arguments**

| | |
|---|---|
| `object` | An object of class `Wspec`. |
| `observed` | Observed notes, probably as a result from `noteFromFF` (or a smoothed version). This should correspond to the `Wspec object`. |
| `expected` | Expected notes (optional; in order to compare results). |
| `bars` | Number of bars to be plotted (a virtual static segmentation takes place). If `NULL` (default), time rather than bars are used. |
| `main` | Main title of the plot. |
| `xlab, ylab` | Annotation of x/y-axes. |
| `xlim, ylim` | Range of x/y-axis. |
| `observedcol` | Colour for the observed melody. |
| `expectedcol` | Colour for the expected melody. |
| `gridcol` | Colour of the grid. |
| `lwd` | Line width, see `par` for details. |
| `las` | Orientation of axis labels, see `par` for details. |
| `cex.axis` | Size of tick mark labels, see `par` for details. |
| `mar` | Margins of the plot, see `par` for details. |
| `notenames` | Optionally specify other notenames (character) for the y axis. |
| `silence` | Character string for label of the 'silence' (default) axis. |
| `plotenergy` | Logical (default: `TRUE`), whether to plot energy values in the bottom part of the plot. |
| `...` | Additional graphical parameters to be passed to underlying `plot` function. |
| `axispar` | A named list of three other lists (`ax1`,`ax2`, and `ax4`) containing parameters passed to the corresponding `axis` calls for the three axis time (`ax1`), notes (`ax2`), and energy (`ax4`). |

| | |
|---|---|
| `boxpar` | A list of parameters to be passed to the box generating functions. |
| `energylabel` | A list of parameters to be passed to the energy-label generating `mtext` call. |
| `energypar` | A list of parameters to be passed to the `lines` function that draws the energy curve. |
| `expectedpar` | A list of parameters to be passed to the `rect` function that draws the rectangles for expected values. |
| `gridpar` | A list of parameters to be passed to the `abline` function that draws the grid lines. |
| `observedpar` | A list of parameters to be passed to the `lines` function that draws the observed values. |

### See Also

`noteFromFF`, `FF`, `quantplot`; for an example, see the help in tuneR.

---

## `periodogram` *Periodogram (Spectral Density) Estimation on Wave objects*

---

### Description

This function estimates one or more periodograms (spectral densities) of the time series contained in an object of class `Wave` using a window running through the time series (possibly with overlapping). It returns an object of class `Wspec`.

### Usage

```
periodogram(object, width = length(object@left), overlap = 0,
    starts = NULL, ends = NULL, taper = 0, normalize = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `object` | An object of class `Wave`. |
| `width` | A window of width '`width`' running through the time series selects the samples from which the periodograms are to be calculated. |
| `overlap` | The window can be applied by each overlapping `overlap` samples. |
| `starts` | Start number (in samples) for a window. If not given, this value is derived from argument `ends`, or will be derived `width` and `overlap`. |

`ends` End number (in samples) for a window. If not given, this value is derived from argument `starts`, or will be derived from `width` and `overlap`.

`taper` proportion of data to taper. See `spec.pgram` for details.

`normalize` Logical; if `TRUE` (default), two steps will be applied: (i) the input signal will be normalized to amplitude `max(abs(amplitude)) == 1`, (ii) the resulting `spec` values will be normalized to sum up to one for each periodogram.

`...` Further arguments to be passed to the underlying function `spec.pgram`.

**Value**

An object of class `Wspec` is returned containing the following slots.

`freq` Vector of frequencies at which the spectral density is estimated. See `spectrum` for details. (1)

`spec` List of vectors or matrices of the `spec` values returned by `spec.pgram` at frequencies corresponding to `freq`. Each element of the list corresponds to one periodogram estimated from samples of the window beginning at `start` of the `Wave` object.

`kernel` The kernel argument, or the kernel constructed from spans returned by `spec.pgram`. (1)

`df` The distribution of the spectral density estimate can be approximated by a chi square distribution with `df` degrees of freedom. (1)

`taper` The value of the `taper` argument. (1)

`width` The value of the `width` argument. (1)

`overlap` The value of the `overlap` argument. (1)

`normalize` The value of the `normalize` argument. (1)

`starts` If the argument `starts` was given in the call, its value. If the argument `ends` was given in the call, '`ends - width`'. If neither `starts` nor `ends` was given, the start points of all periodograms. In the latter case the start points are calculated from the arguments `width` and `overlap`.

`stereo` Whether the underlying `Wave object` was stereo ot not. (1)

`samp.rate` Sampling rate of the underlying `Wave object`. (1)

`variance` The variance of samples in each window, corresponding to amplitude / loudness of sound.

`energy` The "energy" $E$, also an indicator for the amplitude / loudness of sound:

$$E(x_I) := 20 * log_{10} \sum_{j \in I} |x_j|,$$

where $I$ indicates the interval $I :=$ `start[i]:end[i]` for all $i := 1, \ldots,$ `length(starts)`.

Those slots marked with "(1)" contain the information once, because it is unique for all periodograms of estimated by the function call.

### Note

This function is still in development. Support for processing of stereo `Wave` objects has not yet been implemented.

### See Also

- for the resulting objects' class: `Wspec`,
- for plotting: `plot-Wspec`,
- for the underlying periodogram calculations: `spec.pgram`,
- for the input data class: `Wave`.

### Examples

```
# constructing a Wave object (1 sec.) containing sinus sound
# with 440Hz:
Wobj <- sine(440, bit = 16)
Wobj

# Calculate periodograms in windows of 4096 samples each - without
#   any overlap - resulting in an Wspec object that is printed:
Wspecobj <- periodogram(Wobj, width = 4096)
Wspecobj

# Plot the first periodogram from Wspecobj:
plot(Wspecobj)
# Plot the third one and choose a reasonable xlim:
plot(Wspecobj, which = 3, xlim = c(0, 1000))
# Mark frequency that has been generated before:
abline(v = 440, col="red")

FF(Wspecobj)              # all ~ 440 Hertz
noteFromFF(FF(Wspecobj))  # all diapason a
```

---

`plot-Wave` *Plotting Wave objects*

---

### Description

Plotting objects of class `Wave`.

### Usage

```
## S4 method for signature 'Wave, missing':
plot(x, info = FALSE, xunit = c("time", "samples"),
    ylim = NULL, main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    simplify = TRUE, nr = 1500, ...)

plot.Wave.channel(x, xunit, ylim, xlab, ylab, main, nr, simplify, ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class `Wave`. |
| `info` | Logical, whether to include (written) information on the `Wave` object within the plot. |
| `xunit` | Character indicating which units are used for setting up user coodinates (see `par`) and x-axis labeling. If `xunit = "time"`, the unit is time in seconds, otherwise the number of samples. |
| `ylim` | The `y` (amplitude) limits of the plot. |
| `main, sub` | A main / sub title for the plot. |
| `xlab, ylab` | Label for x-/y-axis. |
| `simplify` | Logical, whether the plot should be "simplified". If `TRUE` (default), not all (thousand/millions/billions) of points (samples) of the `Wave` object are drawn, but the `nr` (see below) ranges (in form of segments) within `nr` windows of the time series.<br>Plotting with `simplify = FALSE` may take several minutes (depending on the number of samples in the `Wave`) and output in any vector format may be really huge. |
| `nr` | Number of windows (segments) to be used *approximately* (an appropriate number close to `nr` is selected) to `simplify` (see above) the plot. Only used if `simplify = TRUE` and the number of samples of `Wave` object `x` is larger. |
| `...` | Further arguments to be passed to the underlying plot functions. |

### Details

Function `plot.Wave.channel` is a helper function to plot a single (left!) channel; in particular it is *not* intended to be called by the user directly.

### See Also

`Wave` and tuneR

---

`plot-Wspec` *Plotting Wspec objects*

---

### Description

Plotting a periodogram contained in an object of class `Wspec`.

### Usage

```
## S4 method for signature 'Wspec, missing':
plot(x, which = 1, type = "h", xlab = "Frequency",
    ylab = NULL, log = "", ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class `Wspec`. |
| `which` | Integer indicating which of the periodograms contained in object `x` to plot. Default is to plot the first one. |
| `type` | The default is to plot horizontal lines, rather than points. See `plot.default` for details. |
| `xlab, ylab` | Label for x-/y-axis. |
| `log` | Character - `"x"` if the x axis is to be logarithmic, `"y"` if the y axis is to be logarithmic (quite typical for some visualizations of periodograms), and `"xy"` or `"yx"` if both axes are to be logarithmic. |
| `...` | Further arguments to be passed to the underlying plot functions. See `plot.default` for details. |

### See Also

see `Wspec`, `periodogram` and tuneR for the constructor function and some examples.

---

`plot-WspecMat` *Plotting WspecMat objects*

---

### Description

Plotting a spectogram (image) of an object of class `Wspec` or `WspecMat`.

### Usage

```
## S4 method for signature 'WspecMat, missing':
plot(x, xlab = "time", ylab = "Frequency",
    xunit = c("samples", "time"), ...)
## S4 method for signature 'Wspec':
image(x, xlab = "time", ylab = "Frequency",
    xunit = c("samples", "time"), ...)
```

### Arguments

| | |
|---|---|
| `x` | Object of class `WspecMat` (for `plot`) or `Wspec` (for `image`). |
| `xlab, ylab` | Label for x-/y-axis. |
| `xunit` | Character indicating which units are used to annotate the x axis. If `xunit = "time"`, the unit is time in seconds, otherwise the number of samples. |
| `...` | Further arguments to be passed to the underlying `image` function. See `image` for details. |

### Details

Calling `image` on a `Wspec` object converts it to class `WspecMat` and calls the corresponding `plot` function.
Calling `plot` on a `WspecMat` object generates an `image` with correct annotated axes.

### See Also

see `image`, `Wspec`, `WspecMat`, `periodogram` and tuneR for the constructor function and some examples.

`tuneR` *tuneR*

## Description

tuneR, a collection of examples

## Functions in tuneR

*tuneR* consists of several functions to work with and to analyze Wave files. In the following examples, some of the functions to generate some data (such as `sine`), to read and write Wave files (`readWave`, `writeWave`), to represent or construct Wave files (`Wave`), to transform Wave objects (`bind`, `channel`, `downsample`, `extractWave`, `mono`, `stereo`), and to `play` Wave objects are used.

Other functions and classes are available to calculate several periodograms of a signal (`periodogram`, `Wspec`), to estimate the corresponding fundamental frequencies (`FF`, `FFpure`), to derive the corresponding notes (`noteFromFF`), and to apply a `smoother`. Now, the melody and corresponding energy values can be plotted using the function `melodyplot`.

A next step is the quantization (`quantize`) and a corresponding plot (`quantplot`) showing the note values for binned data. Moreover, a function called `lilyinput` (and a data-preprocessing function `quantMerge`) can prepare a data frame to be presented as sheet music by postprocessing with the music typesetting software LilyPond.

Of course, print (show), plot and summary methods are available for most classes.

## Examples

```
library(tuneR) # in a regular session, we are loading tuneR

# constructing a mono Wave object (2 sec.) containing sinus
# sound with 440Hz and folled by 220Hz:
Wobj <- bind(sine(440, bit = 16), sine(220, bit = 16))
show(Wobj)
plot(Wobj) # it does not make sense to plot the whole stuff
plot(extractWave(Wobj, from = 1, to = 500))
## Not run:
play(Wobj) # listen to the sound
## End(Not run)

tmpfile <- file.path(tempdir(), "testfile.wav")
# write the Wave object into a Wave file
```

```
# (can be played with any player):
writeWave(Wobj, tmpfile)
# reading it in again:
Wobj2 <- readWave(tmpfile)

Wobjm <- mono(Wobj, "left") # extract the left channel
# and downsample to 11025 samples/sec.:
Wobjm11 <- downsample(Wobjm, 11025)
# extract a part of the signal interactively
# (click for left/right limits):
## Not run:
Wobjm11s <- extractWave(Wobjm11)
## End(Not run)
# or extract some values reproducibly
Wobjm11s <- extractWave(Wobjm11, from=1000, to=17000)

# calculating periodograms of sections each consisting of
# 1024 observations, overlapping by 512 observations:
WspecObject <- periodogram(Wobjm11s, normalize = TRUE,
                           width = 1024, overlap = 512)
# Let's look at the first periodogram:
plot(WspecObject, xlim = c(0, 2000), which = 1)
# or a spectrogram
image(WspecObject, ylim = c(0, 1000))
# calculate the fundamental frequency:
ff <- FF(WspecObject)
print(ff)
# derive note from FF given diapason a'=440
notes <- noteFromFF(ff, 440)
# smooth the notes:
snotes <- smoother(notes)
# outcome should be 0 for diapason "a'" and -12 (12 halftones lower)
# for "a":
print(snotes)
# plot melody and energy of the sound:
melodyplot(WspecObject, snotes)

# apply some quantization (into 8 parts):
qnotes <- quantize(snotes, WspecObject@energy, parts = 8)
# an plot it, 4 parts a bar (including expected values):
quantplot(qnotes, expected = rep(c(0, -12), each = 4), bars = 2)
# now prepare for LilyPond
qlily <- quantMerge(snotes, 4, 4, 2)
qlily
```

---

`Wave-class` *Class Wave and constructors*

---

### Description

Class "Wave" and its constructor functions

### Usage

```
Wave(left, ...)
## S4 method for signature 'numeric':
Wave(left, right = numeric(0), samp.rate = 44100, bit = 16, ...)
```

### Arguments

| | |
|---|---|
| `left, right, samp.rate, bit` | See Section "Slots". |
| `...` | Further arguments to be passed to the default method `Wave.default`. |

### Objects from the Class

Objects can be created by calls of the form `new("Wave", ...)`, or more conveniently using the function `Wave`.

### Slots

`left:` Object of class `"numeric"` representing the left channel.

`right:` Object of class `"numeric"` representing the right channel, `NULL` if mono.

`stereo:` Object of class `"logical"` indicating whether this is a stereo (two channels) or mono representation.

`samp.rate:` Object of class `"numeric"` - the sampling rate, e.g. 44100 for CD quality.

`bit:` Object of class `"numeric"`, common is 16 for CD quality, or 8 for a rather rough representation.

### See Also

`writeWave`, `readWave`

### Examples

```
# constructing a Wave object (1 sec.) containing sinus sound with
# 440Hz:
x <- seq(0, 2*pi, length = 44100)
channel <- round(32000 * sin(440 * x))
Wobj <- Wave(left = channel)
Wobj

# or more easily:
Wobj <- sine(440, bit = 16)
```

---

## WaveIO *Reading and writing Wave files*

---

### Description

Reading and writing Wave files.

### Usage

```
readWave(filename)
writeWave(object, filename)
```

### Arguments

| | |
|---|---|
| `filename` | Filename of the file to be read or written. |
| `object` | Object of class `Wave` to be written to a Wave file. |

### Value

`readWave` returns an object of class `Wave`.

`writeWave` creates a Wave file, but returns nothing.

### See Also

`Wave`, `normalize`

**Examples**

```
Wobj <- sine(440, bit = 16)

tdir <- tempdir()
tfile <- file.path(tdir, "myWave.wav")
writeWave(Wobj, filename = tfile)
list.files(tdir, pattern = "\.wav$")
newWobj <- readWave(tfile)
newWobj
file.remove(tfile)
```

## A.2 Auszug der Dokumentation des R Pakets BRugs

Der folgende englischsprachige Auszug aus der Original-Dokumentation des R-Pakets *BRugs* (Stand: Version 0.2-7, vgl. Abschnitt 5.3.1) wurde aus Gründen der Platzersparnis leicht gekürzt und ist aus demselben Grunde in kleinerer Schrift und mit reduziertem Zeilenabstand abgedruckt. Für die vollständige Hilfe verweise ich auf Thomas et al. (2006a)[2].

---

`BRugs` *Introduction to BRugs*

---

### Description

This manual describes how to use the BRugs software

### Usage

```
help.BRugs(browser = getOption("browser"))
```

### Arguments

| | |
|---|---|
| `browser` | the name of the program to be used as hypertext browser. It should be in the PATH, or a full path specified. |

### Details

BRugs is a collection of R functions that allow users to analyze graphical models using MCMC techniques. Most of the R functions in BRugs provide a interface to the BRugs dynamic link library (shared object file). The BRugs dynamic link library is able to make use of many of the WinBUGS components, in particular those components concerned with graphical models and MCMC simulation. BRugs lacks the GUI interface of WinBUGS but is able to use R to create graphical displays of the MCMC simulation. BRugs uses the same model specification language as WinBUGS and the same format for data and initial values. However BRugs always uses plain text files for

[2] http://CRAN.R-project.org/src/contrib/Descriptions/BRugs.html

input inplace of WinBUGS compound documents. The BRugs functions can be split into two groups: those associated with setting up and simulating the graphical model and those associated with making statistical inference. In general the R functions in BRugs correspond to the command buttons and and text entry fields in the menus of WinBUGS. Each WinBUGS text entry field splits into two R functions, one to set the quantity and the other to get the value of the quantity.

### Permission and Disclaimer

BRugs is released under the GNU GENERAL PUBLIC LICENSE. For details see http://mathstat.helsinki.fi/openbugs/ or type `help.BRugs()`.

More informally, potential users are reminded to be extremely careful if using this program for serious statistical analysis. We have tested the program on quite a wide set of examples, but be particularly careful with types of model that are currently not featured. If there is a problem, BRugs might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse. Please let us know of any successes or failures.

### See Also

`help.WinBUGS` and the meta function `BRugsFit`

### Examples

```
###    Step by step example:    ###
library(BRugs) # loading BRugs

## Now setting the working directory to the examples' one:
oldwd <- getwd()
setwd(system.file("OpenBUGS", "Examples", package="BRugs"))

## some usual steps (like clicking in WinBUGS):
modelCheck("ratsmodel.txt")          # check model file
modelData("ratsdata.txt")            # read data file
modelCompile(numChains=2)            # compile model with 2 chains
modelInits(rep("ratsinits.txt", 2))  # read init data file
modelUpdate(1000)                    # burn in
samplesSet(c("alpha0", "alpha"))     # alpha0, alpha to be monitored
modelUpdate(1000)                    # 1000 more iterations ....

samplesStats("*")                    # the summarized results

## some plots
samplesHistory("*", mfrow = c(4, 2)) # plot the chain,
```

```
samplesDensity("alpha")              # plot the densities,
samplesBgr("alpha[1:6]")             # plot the bgr statistics, and
samplesAutoC("alpha[1:6]", 1)        # plot autocor. of first chain

## switch back to the previous working directory:
setwd(oldwd)
## Not run:
# Getting more (online-)help:
help.BRugs()
## End(Not run)
```

---

`BRugsFit` *BRugs' meta function*

---

### Description

This function takes model, data and starting values as input and automatically runs a simulation in BRugs.

### Usage

```
BRugsFit(modelFile, data, inits, numChains = 3, parametersToSave,
    nBurnin = 1000, nIter = 1000, nThin = 1,
    DIC = TRUE, working.directory = NULL, digits = 5)
```

### Arguments

| | |
|---|---|
| `modelFile` | File containing the model written in OpenBUGS code. |
| `data` | Either a named list (names corresponding to variable names in the `modelFile`) of the data for the OpenBUGS model, *or* a vector or list of the names of the data objects used by the model. In these cases data are written into a file 'data.txt' into the working directory.<br>If a filename of an existing file is given, data are read from that file. |
| `inits` | A list with `numChains` elements; each element of the list is itself a list of starting values for the OpenBUGS model, *or* a function creating (possibly random) initial values. In these cases inits are written into files 'inits1.txt', ..., 'initsN.txt' into the working directory.<br>If a vector of filenames of existing files is given, inits are read from those files. Alternatively, if `inits` is not specified, initial values are generated by OpenBUGS. |

`numChains` Number of Markov chains (default: 3).

`parametersToSave`
Character vector of the names of the parameters to save which should be monitored.

`nBurnin` Length of burn in (before `nIter` iterations start).

`nIter` Number of iterations (without burn in).

`nThin` Every `nThin`-th iteration of each chain is stored.

`DIC` Logical, whether to calculate and return the DIC.

`working.directory`
Sets working directory during execution of this function; `data`, `inits` and other files are written to / read from this directory if no other directory is explicitly given in those arguments. If `NULL`, the current working directory is chosen.

`digits` Number of significant digits used for OpenBUGS input, see `formatC`.

## Value

A list containg components

`Stats` A data frame containing sample statistics. See `samplesStats`.

`DIC` The DIC statistics, if `DIC=TRUE`, else `NULL`. See `dicStats`.

## See Also

`BRugs`, `help.WinBUGS`

## Examples

```
BRugsFit(data = "ratsdata.txt", inits = "ratsinits.txt",
    para = c("alpha", "beta"), modelFile = "ratsmodel.txt",
    numChains = 1,
    working.directory = system.file("OpenBUGS", "Examples",
                                    package = "BRugs"))
```

---

`modelCheck` *Checking the model file*

---

### Description

This function parses a BUGS language description of the statistical model.

### Usage

```
modelCheck(fileName)
```

### Arguments

| | |
|---|---|
| `fileName` | file containing the BUGS language description of the statistical model. |

### Value

If a syntax error is detected the position of the error and a description of the error is printed, otherwise the 'model is syntaxicaly correct' message is displayed.

### Note

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

### See Also

`BRugs`, `help.WinBUGS`

---

`modelCompile` *Compiling the model*

---

### Description

This function builds the data structures needed to carry out MCMC sampling.

### Usage

```
modelCompile(numChains = 1)
```

### Arguments

| | |
|---|---|
| `numChains` | Simulation is carried out for `numChains` chains. |

### Details

The model is checked for completeness and consistency with the data. A node called 'deviance' is automatically created which calculates minus twice the log-likelihood at each iteration, up to a constant. This node can be used like any other node in the graphical model.

### Value

When the model has been successfully compiled, 'model compiled' message should be printed.

### Note

This command becomes active once the model has been successfully checked (see `modelCheck`).

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

### See Also

`BRugs`, `help.WinBUGS`

---

## `modelData` *Loading the data*

---

### Description

This function loads data into the statistical model.

### Usage

```
modelData(fileName = "data.txt")
```

### Arguments

| | |
|---|---|
| `fileName` | Filename(s) of file(s) containing the data in OpenBUGS format. |

### Value

If any syntax errors or data inconsistencies are detected an error message is displayed. Corrections can be made to the data without returning to the 'check model' stage. When the data have been loaded successfully the message 'data loaded' should appear.

### Note

This function can be executed once a model has been successfully checked (see `modelCheck`), it can no longer be executed once the model has been successfully compiled.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

### See Also

`BRugs`, `help.WinBUGS`

---

## `modelInits` *Loading initial values*

---

### Description

This function loads initial values for the MCMC simulation.

### Usage

```
modelInits(fileName, chainNum = NULL)
```

### Arguments

| | |
|---|---|
| `fileName` | Character vector of filenames containing the initial values in OpenBUGS format. |
| `chainNum` | The initial values will be loaded for the chain number `chainNum`. By default `chainNum` is one the first time `modelInits` is executed and incremented by one after each call modula the number of chains `numChains` being simulated (and restarts at 1 after that). If `fileName` is a vector, `chainNum` is increased automatically by default after processing each file. If there is more than one file containing initial values for one chain, either set `chainNum` explicitly, or wait until cycle restarts at chain 1. |

### Details

This function checks that initial values are in the form of an appropriate R object or rectangular array and that they are consistent with any previously loaded data. If some of the elements in an array are known (say because they are constraints in a parameterisation), those elements should be specified as missing (NA) in the initial values file.

Generally it is recommended to load initial values for all fixed effect nodes (founder nodes with no parents) for all chains, initial values for random effects can be generated using the `modelGenInits` function.

### Value

Any syntax errors or inconsistencies in the initial value are displayed. If, after loading the initial values, the model is fully initialized this will be reported by displaying the message 'model initialized'. Otherwise the message 'initial values loaded but this or another chain contain uninitialized variables' will be displayed. The second message can have several meanings:

| | |
|---|---|
| `a)` | If only one chain is simulated it means that the chain contains some nodes that have not been initialized yet. |
| `b)` | If several chains are to be simulated it could mean (a) or that no initial values have been loaded for one of the chains. |

In either case further initial values can be loaded, or `modelGenInits` can be executed to try and generate initial values for all the uninitialized nodes in all the simulated chains.

### Note

This function can be executed once the model has been successfully compiled. It can still be executed once MCMC sampling has been started having the effect of starting

the sampler out on a new trajectory.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

### See Also

`BRugs`, `help.WinBUGS`

---

`modelUpdate` *Updating the model*

---

### Description

This function updates the model.

### Usage

```
modelUpdate(numUpdates, thin = 1, overRelax = FALSE)
```

### Arguments

| | |
|---|---|
| `numUpdates` | This function updates the model by carrying out `thin * numUpdates` MCMC iterations for each chain. |
| `thin` | The samples from every k*th* iteration will be used for inference, where k is the value of `thin`. Setting `thin > 1` can help to reduce the auto-correlation in the sample, but there is no real advantage in thinning except to reduce storage requirements. |
| `overRelax` | If `overRelax` is `TRUE` an over-relaxed form of MCMC (Neal, 1998) which will be executed where possible. This generates multiple samples at each iteration and then selects one that is negatively correlated with the current value. The time per iteration will be increased, but the within-chain correlations should be reduced and hence fewer iterations may be necessary. However, this method is not always effective and should be used with caution. The auto-correlation function may be used to check whether the mixing of the chain is improved. |

### Note

This function can be executed once the model has been compiled and initialized.

If an attempt is made to execute this function in an inappropriate context the generic error message 'command is not allowed (greyed out)' is displayed.

### References

Neal, R. (1998): Suppressing random walks in Markov chain Monte Carlo using ordered over-relaxation. In M.I. Jordan (Ed.): *Learning in Graphical Models*, Kluwer Academic Publishers, Dordrecht, 205-230. `http://www.cs.utoronto.ca/~radford/publications.html`

### See Also

`BRugs`, `help.WinBUGS`

---

## `samplesSample` *Stored values*

---

### Description

This function returns an array of stored values.

### Usage

```
samplesSample(node)
```

### Arguments

| | |
|---|---|
| `node` | Character vector of length 1, name of a variable in the model. |

### Value

Values of the stored sample.

### Note

If sampling a vector of parameters, the function must be called for each parameter separately such as `samplesSample(node[1])`.

### See Also

`BRugs`, `help.WinBUGS`

## A.3 Versuchsplan

Tabelle A.3: Versuchsplan aus Abschnitt 6.1

| $H$ | $f_0$ | $f_v$ | $A_v/f_v$ | $\phi_v$ |
|---|---|---|---|---|
| + (3) | + (1000 Hz) | + (9 Hz) | + (15) | + (3) |
| + (3) | + (1000 Hz) | + (9 Hz) | + (15) | – (0) |
| + (3) | + (1000 Hz) | + (9 Hz) | – (5) | + (3) |
| + (3) | + (1000 Hz) | + (9 Hz) | – (5) | – (0) |
| + (3) | + (1000 Hz) | – (5 Hz) | + (15) | + (3) |
| + (3) | + (1000 Hz) | – (5 Hz) | + (15) | – (0) |
| + (3) | + (1000 Hz) | – (5 Hz) | – (5) | + (3) |
| + (3) | + (1000 Hz) | – (5 Hz) | – (5) | – (0) |
| + (3) | – (250 Hz) | + (9 Hz) | + (15) | + (3) |
| + (3) | – (250 Hz) | + (9 Hz) | + (15) | – (0) |
| + (3) | – (250 Hz) | + (9 Hz) | – (5) | + (3) |
| + (3) | – (250 Hz) | + (9 Hz) | – (5) | – (0) |
| + (3) | – (250 Hz) | – (5 Hz) | + (15) | + (3) |
| + (3) | – (250 Hz) | – (5 Hz) | + (15) | – (0) |
| + (3) | – (250 Hz) | – (5 Hz) | – (5) | + (3) |
| + (3) | – (250 Hz) | – (5 Hz) | – (5) | – (0) |
| – (2) | + (1000 Hz) | + (9 Hz) | + (15) | + (3) |
| – (2) | + (1000 Hz) | + (9 Hz) | + (15) | – (0) |
| – (2) | + (1000 Hz) | + (9 Hz) | – (5) | + (3) |
| – (2) | + (1000 Hz) | + (9 Hz) | – (5) | – (0) |
| – (2) | + (1000 Hz) | – (5 Hz) | + (15) | + (3) |
| – (2) | + (1000 Hz) | – (5 Hz) | + (15) | – (0) |
| – (2) | + (1000 Hz) | – (5 Hz) | – (5) | + (3) |
| – (2) | + (1000 Hz) | – (5 Hz) | – (5) | – (0) |
| – (2) | – (250 Hz) | + (9 Hz) | + (15) | + (3) |
| – (2) | – (250 Hz) | + (9 Hz) | + (15) | – (0) |
| – (2) | – (250 Hz) | + (9 Hz) | – (5) | + (3) |
| – (2) | – (250 Hz) | + (9 Hz) | – (5) | – (0) |
| – (2) | – (250 Hz) | – (5 Hz) | + (15) | + (3) |
| – (2) | – (250 Hz) | – (5 Hz) | + (15) | – (0) |
| – (2) | – (250 Hz) | – (5 Hz) | – (5) | + (3) |
| – (2) | – (250 Hz) | – (5 Hz) | – (5) | – (0) |
| + (3) | + (1000 Hz) | 0 (1 Hz) | 0 (0) | 0 (0) |
| + (3) | – (250 Hz) | 0 (1 Hz) | 0 (0) | 0 (0) |
| – (2) | + (1000 Hz) | 0 (1 Hz) | 0 (0) | 0 (0) |
| – (2) | – (250 Hz) | 0 (1 Hz) | 0 (0) | 0 (0) |

# Literaturverzeichnis


Adak, S. (1998): Time-Dependent Spectral Analysis of Nonstationary Time Series. *Journal of the American Statistical Association*, 93, 1488–1501.

Beran, J. (2004): *Statistics in Musicology*. Boca Raton: Chapman & Hall / CRC.

Berg, R. und Stork, D. (1982): *The Physics of Sound*. New Jersey: Prentice-Hall.

Blackham, E. (1988): *Die Physik der Musikinstrumente*, Kapitel Klaviere. Heidelberg: Spektrum der Wissenschaft.

Bloomfield, P. (2000): *Fourier Analyis of Time Series – An Introduction*. John Wiley and Sons, 2. Auflage.

Brillinger, D. (1975): *Time Series: Data Analysis and Theory*. New York: Holt, Rinehart & Winston Inc.

Brockwell, P. und Davis, R. (1991): *Time Series: Theory and Methods*. New York: Springer-Verlag, 2. Auflage.

Bronstein, I., Semendjajew, K., Musiol, G., und Mühlig, H. (2000): *Taschenbuch der Mathematik*. Frankfurt am Main: Verlag Harri Deutsch, 5. Auflage.

Brown, H., Butler, D., und Jones, M. (1994): Musical and Temporal Influences on Key Discovery. *Music Perception*, 11 (4), 371–407.

Brown, J. und Puckette, M. (1992): An efficcient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92 (5), 2698–2701.

Brown, J. und Puckette, M. (1993): A high resolution fundamental frequency determination based on phase changes of the Fourier transform. *The Journal of the Acoustical Society of America*, 94 (2), 662–667.

Cano, P., Loscos, A., und Bonada, J. (1999): Score-Performance Matching using HMMs. In: *Proceedings of the International Computer Music Conference*. Beijing, China.

Cemgil, A. und Kappen, B. (2003): Monte Carlo Methods for Tempo Tracking and Rhythm Quantization. *Journal of Artificial Intelligence Research*, 18, 45–81.

Cemgil, A., Kappen, B., Desain, P., und Honing, H. (2001): On tempo tracking: Tempogram representation and Kalman filtering. *Journal of New Music Research*, 29 (4), 259–273.

Cemgil, T., Desain, P., und Kappen, B. (2000): Rhythm Quantization for Transcription. *Computer Music Journal*, 24 (2), 60–76.

Chambers, J. (1998): *Programming with Data. A Guide to the S Language*. New York: Springer-Verlag.

Cooley, J. und Tukey, J. (1965): An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19 (90), 297–301.

Dahlhaus, R. (1997): Fitting Time Series Models to Nonstationary Processes. *The Annals of Statistics*, 25, 1–37.

Davies, M. und Plumbley, M. (2004): Causal Tempo Tracking of Audio. In: *Proceedings of the International Conference on Music Information Retrieval*. Audiovisual Institute, Universitat Pompeu Fabra, Barcelona, Spain.

Davy, M. und Godsill, S. (2002): Bayesian Harmonic Models for Musical Pitch Estimation and Analysis. *Technical Report 431*, Cambridge University Engineering Department, Cambridge.

Dixon, S. (1996): Multiphonic Note Identification. *Australian Computer Science Communications*, 17 (1), 318–323.

Evangelista, G. (2001): Flexible Wavelets for Music Signal Processing. *Journal of New Music Research*, 30 (1), 13–22.

Gallant, A., Gerig, T., und Evans, J. (1974): Time Series Realizations Obtained According to an Experimental Design. *Journal of the American Statistical Association*, 69, 639–645.

Godsill, S. und Davy, M. (2003): Bayesian Modelling of Music Audio Signals. In: *Bulletin of the International Statistical Institute, 54th Session*, Ausgabe LX, book 2, 504–506. Berlin.

Godsill, S. und Davy, M. (2005): Bayesian Computational Models for Inharmonicity in Musical Instruments. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. New Paltz, NY.

Hyvärinen, A., Karhunen, J., und Oja, E. (2001): *Independent Component Analysis*. New York: John Wiley and Sons.

Klapuri, A. (2001): Multipitch Estimation and Sound Separation by the Spectral Smoothness Principle. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Klein, W., Plomp, R., und Pols, L. (1970): Vowel Spectra, Vowel Spaces, and Vowel Identification. *The Journal of the Acoustical Society of America*, 48 (4), 999–1009.

Knuth, D. (1984): *The TEXbook*. Addison-Wesley.

Lamport, L. (1994): *LATEX, a Document Preparation System*. Addison-Wesley, 2. Auflage.

Lesaffre, M., Tanghe, K., Martens, G., Moelants, D., Leman, M., De Baets, B., De Meyer, H., und Martens, J.-P. (2003): The MAMI Query-By-Voice Experiment: Collecting and annotating vocal queries for music information retrieval. In: *Proceedings of the International Conference on Music Information Retrieval*. Baltimore, Maryland, USA and Library of Congress, Washington, DC, USA.

Ligges, U. (2000): *Identifikation lokal stationärer Anteile in Gesangszeitreihen*. Diplomarbeit, Fachbereich Statistik, Universität Dortmund, Dortmund, Germany.

Ligges, U. (2005a): *Programmieren mit R*. Heidelberg: Springer-Verlag. ISBN 3-540-20727-9. URL http://www.statistik.uni-dortmund.de/~ligges/PmitR/.

Ligges, U. (2005b): *tuneR: Analysis of music*. URL `http://cran.r-project.org/src/contrib/Descriptions/tuneR.html`. R package version 0.2-1.

Ligges, U., Weihs, C., und Hasse-Becker, P. (2002): Detection of Locally Stationary Segments in Time Series. In: W. Härdle und B. Rönz (Hrsg.) *COMPSTAT 2002 - Proceedings in Computational Statistics - 15th Symposium held in Berlin*, 285–290. Heidelberg: Physika Verlag.

Meyer, J. (1995): *Akustik und musikalische Aufführungspraxis*. Frankfurt am Main: Bochinsky.

Microsoft Corporation (1991): *Multimedia Programming Interface and Data Specification, 1.0*. Joint design by IBM Corporation and Microsoft Corporation.

MIDI Manufacturers Association (2001): *Complete MIDI 1.0 Detailed Specification*, 2. Auflage. URL http://www.midi.org.

Müllensiefen, D. und Frieler, K. (2004): Optimizing Measures of Melodic Similarity for the Exploration of a Large Folk Song Database. In: *5th International Conference on Music Information Retrieval*. Barcelona, Spain: Audiovisual Institute, Universitat Pompeu Fabra.

Nelder, J. und Mead, R. (1965): A Simplex Method for Function Minimization. *The Computer Journal*, 7, 308–313.

Nienhuys, H.-W., Nieuwenhuizen, J., et al. (2005): *GNU LilyPond – The Music Typesetter*. Free Software Foundation. URL http://www.lilypond.org/. Version 2.6.5.

Ombao, H., Raz, J., von Sachs, R., und Malow, B. (2001): Automatic Statistical Analysis of Bivariate Nonstationary Time Series. *Journal of the American Statistical Association*, 96 (454), 543–560.

Pang, H. und Yoon, D. (2005): Automatic detection of vibrato in monophonic music. *Pattern Recognition*, 38 (7), 1135–1138.

Picard, D. (1985): Testing and Estimating Change–Points in Time Series. *Advances in Applied Probability*, 17, 841–867.

Polotti, P. und Evangelista, G. (2000): Harmonic-Band Wavelet Coefficient Modeling for Pseudo-Periodic Sound Processing. In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*. Verona, Italy.

Polotti, P. und Evangelista, G. (2001): Multiresolution Sinusoidal/Stochastic Model for Voiced-Sounds. In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01)*. Limerick, Ireland.

Pressing, J. und Lawrence, P. (1993): Transcribe: A Comprehensive Autotranscription Program. In: *Proceedings of the International Computer Music Conference*, 343–345. Tokyo, Japan.

Preusser, A., Ligges, U., und Weihs, C. (2002): Ein R Exportfilter für das Notations- und Midi-Programm LilyPond. *Arbeitsbericht 35*, Fachbereich Statistik, Universität Dortmund, Dortmund, Germany. URL `http://www.statistik.uni-dortmund.de/de/content/forschung/publikationen/arbeitsberichte.html`.

Priestley, M. und Subba Rao, T. (1969): A Test for Non-Stationarity of Time Series. *Journal of the Royal Statistical Society, Series B*, 31, 140–149.

R Development Core Team (2005): *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Radeka, V. (1969): 1/f noise in physical measurements. *IEEE Transactions of Nuclear Science*, 16, 17–35.

Raphael, C. (2001): A Probabilistic Expert System for Automatic Musical Accompaniment. *Journal of Computational and Graphical Statistics*, 10 (3), 487–512.

Reuter, C. (2002): *Klangfarbe und Instrumentation - Geschichte - Ursachen - Wirkung*. Frankfurt am Main: Peter Lang.

Rossignol, S., Depalle, P., Soumagne, J., Rodet, X., und Collette, J.-L. (1999a): Vibrato: Detection, Estimation, Extraction, Modification. In: *Proceedings 99 Digital Audio Effects Workshop*.

Rossignol, S., Rodet, X., Soumagne, J., Collette, J.-L., und Depalle, P. (1999b): Automatic Characterisation of Musical Signals: Feature Extraction and Temporal Segmentation. *Journal of New Music Research*, 28 (4), 281–295.

Röver, C. (2003): *Musikinstrumentenerkennung mit Hilfe der Hough-Transformation*. Diplomarbeit, Fachbereich Statistik, Universität Dortmund, Dortmund, Germany.

Schlittgen, R. und Streitberg, B. (1997): *Zeitreihenanalyse*. München: Oldenbourg.

Seidner, W. und Wendler, J. (1997): *Die Sängerstimme*. Berlin: Henschel.

Smaragdis, P. und Brown, J. (2003): Non-Negative Matrix Factorization for Polyphonic Music Transcription. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 177–180.

Spiegelhalter, D., Thomas, A., Best, N., und Lunn, D. (2003): *WinBUGS Version 1.4 Users Manual*. Cambridge. URL http://www.mrc-bsu.cam.ac.uk/bugs/.

Spiegelhalter, D., Thomas, A., Best, N., und Lunn, D. (2005): *WinBUGS: User Manual, Version 2.10*. Medical Research Council Biostatistics Unit, Cambridge.

Sturtz, S., Ligges, U., und Gelman, A. (2005): R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software*, 12 (3), 1–16. URL http://www.jstatsoft.org/.

Thomas, A. (2004): *BRugs User Manual, Version 1.0*. Dept of Mathematics & Statistics, University of Helsinki.

Thomas, A., O'Hara, B., Ligges, U., und Sturtz, S. (2006a): *BRugs: OpenBUGS and its R interface BRugs*. URL `http://cran.r-project.org/src/contrib/Descriptions/BRugs.html`. R package version 0.2-7.

Thomas, A., O'Hara, B., Ligges, U., und Sturtz, S. (2006b): Making BUGS Open. *R News*, 6 (1). ISSN 1609-3631. URL http://CRAN.R-project.org/doc/Rnews/.

Van Trees, H. (2001): *Detection, Estimation, and Modulation Theory, Part I*. Melbourne, FL, USA: Wiley-Interscience, reprint Auflage.

Viste, H. und Evangelista, G. (2001): Sounds Source Separation: Preprocessing for Hearing Aids and Structured Audio Coding. In: *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-01)*. Limerick, Ireland.

Viste, H. und Evangelista, G. (2002): An Extension for Source Separation Techniques Avoiding Beats. In: *Proceedings of the 5th Int. Conference on Digital Audio Effects (DAFx-02)*. Hamburg, Germany.

von Ameln, F. (2001): *Blind source separation in der Praxis*. Diplomarbeit, Fachbereich Statistik, Universität Dortmund, Dortmund, Germany.

Walmsley, P., Godsill, S., und Rayner, P. (1999): Polyphonic Pitch Tracking Using Joint Bayesian Estimation of Multiple Frame Parameters. In: *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*. New Paltz, NY.

Weihs, C. und Ligges, U. (2003): Voice Prints as a Tool for Automatic Classification of Vocal Performance. In: R. Kopiez, A. C. Lehmann, I. Wolther, und C. Wolf (Hrsg.) *Proceedings of the 5th Triennial ESCOM Conference*, 332–335. Hanover University of Music and Drama, Germany.

Weihs, C. und Ligges, U. (2005): From Local to Global Analysis of Music Time Series. In: K. Morik, J.-F. Boulicaut, und A. Siebes (Hrsg.) *Local Pattern Detection*, Lecture Notes in Artificial Intelligence 3539, 217–231. Berlin: Springer-Verlag.

Weihs, C., Ligges, U., und Garczarek, U. (2005a): Prediction of Notes from vocal Time Series: An Overview. In: D. Baier und K.-D. Wernecke (Hrsg.) *Innovations in Classification, Data Science, and Information Systems*, 283–294. Berlin: Springer-Verlag.

Weihs, C., Reuter, C., und Ligges, U. (2005b): Register Classification by Timbre. In: C. Weihs und W. Gaul (Hrsg.) *Classification: The Ubiquitous Challenge*, 624–631. Berlin: Springer-Verlag.

Wolfe, P., Godsill, S., und Ng, W.-J. (2004): Bayesian variable selection and regularization for time-frequency surface estimation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66 (3), 575–589.