Wie trennt man perzeptiv erfolgreich Diffus- von Direktschallanteilen in Stereo/Surround/Ambisonics-Aufnahmen?

Seminararbeit aus Algorithmen in Akustik und Computermusik 2

Theresa Loss
Jonathan Arweck

Betreuung: Dr. Franz Zotter, Dr. Matthias Frank

Graz, 1. Mai 2016

kunst uni graz

institut für elektronische musik und akustik

iem

Zusammenfassung

Beim Bearbeiten von Audiosignalen kann man heutzutage eine trockene Aufnahme mit einem beliebigen Nachhall versehen, um einen erwünschten Raumeindruck zu erreichen. Doch ist es auch möglich, diese Prozedur umzukehren, ein Signal sozusagen zu „enthallen“, ohne weitere Informationen über den Nachhall zu kennen? Mit dieser Fragestellung beschäftigen wir uns in der dieser Seminararbeit.

Mögliche Anwendungen für ein solches Verfahren liegen überall dort, wo ein Nachhall zwar durch die Aufnahmesituation zwar unvermeidlich, aber letztlich unerwünscht ist. So eignet sich zum Beispiel ein vom Diffusanteil befreites Signal wesentlich besser zur Richtungsdetektion.

Im Folgenden stellen wir einen Algorithmus aus der jüngeren Fachliteratur vor, der diese Aufgabe mithilfe einer Eigenwertzerlegung erfüllt. Wir beschreiben unsere Implementierung und Erweiterung dieses Algorithmus und untersuchen die Methode auf ihren perzeptiven Erfolg. Ferner zeigen wir auf, wie die Aufteilung in Diffus- und Direktschall durch die Parameterwahl beeinflusst werden kann.

Unsere Untersuchungen ergeben, dass die Methode hauptsächlich für Ambisonics-Signale höherer Ordnung, für Stereo- und 5.1-Surround-Signale jedoch nur bedingt geeignet ist.

Inhaltsverzeichnis

1 Einleitung

Das Ziel dieser Seminararbeit ist es, Audio-Signale perzeptiv erfolgreich in ihre Diffus- und Direktschallanteile zu zerlegen.

In der Praxis hat man nur selten mit reinen Direktschall-Signalen zu tun, da man bei Tonaufnahmen grundsätzlich immer auch den Nachhall des Raumes mit aufnimmt. Ausgenommen davon sind nur Aufnahmen im Freifeld oder im reflexionsarmen Raum. Jedoch sind Freifeld-Aufnahmen aus praktischen Gründen (Witterung etc.) meist nicht durchführbar, während breitbandig reflexionsarme Räume aufgrund des hohen Materialaufwands rar und relativ kostspielig sind. Abgesehen davon sind Tonaufnahmen oft auch örtlich gebunden (zum Beispiel Livekonzerte) oder einfach bereits vorhanden. Daher ist auf den meisten Aufnahmen bereits ein gewisser Nachhall vorhanden.

Aus verschiedenen Gründen kann es nun erstrebenswert sein, ein reines Direktschallsignal zur Verfügung zu haben. Dieses könnte zum Beispiel mit einem neuen künstlichen Nachhall versehen werden, um einen nahezu beliebigen Raumeindruck zu vermitteln. Auch bei weiteren Anwendungsbereichen, wie zum Beispiel der Richtungslokalisation, werden möglichst nachhallfreie Aufnahmen benötigt, um die Auflösung und Präzision solcher Verfahren zu verbessern.

Für diese Arbeit haben wir, gestützt auf [JE13], ein Verfahren zur Trennung von Diffus- von Direktschallanteilen implementiert, getestet und erweitert. Wir untersuchen im Verauf der Arbeit, ob diese Methode eine perzeptiv überzeugende Lösung dieses Problems bietet und inwiefern der Algorithmus adaptiert werden kann, um das Ergebnis weiter zu verbessern.

Als Testmaterial verwenden wir dabei drei verschiedene Audioformate: Stereo, 5.1 Surround und Ambisonics 4. und 5. Ordnung. Für alle Formate wird überprüft, wie gut die Trennung von Diffus- und Direktschallanteilen gelingt. Durch die Verwendung von einer künstlichen Raumimpulsantwort im Ambisonics-Format haben wir auch die Möglichkeit, ein zunächst trockenes Sprachsignal künstlich zu verhallen und anschließend mit dem Separationsalgorithmus den Nachhall wieder zu entfernen. Dies ermöglicht uns einen einfachen auditorischen Vergleich.

2 Umsetzung des Algorithmus

Der prinzipielle Ansatz zur Separation von Diffus- und Direktschall stammt aus einem Artikel von C.T. Jin und N. Epain [JE13], das uns für diese Arbeit zur Verfügung steht. In diesem Artikel wird eine Methode zur Zerlegung von HOA-Signalen (*higher order Ambisonics*, zu deutsch *Ambisonics-Signale höherer Ordnung*) beschrieben.

Diese Methode basiert auf einer gewichteten Projektion des Ambisonics-Signals auf einen durch Eigenwertzerlegung gefundenen Unterraum und soll im Abschnitt 2.1 kurz erläutert werden.

Diesen Ansatz erweiterten wir in mehrerlei Hinsicht, um ihn für eine größere Bandbreite von Signalen anzupassen. So untersuchten wir, ob eine Zerlegung des Signal im Zeit- und/oder Frequenzbereich und anschließende Anwendung des Separations-Algorithmus auf die einzelnen Bestandteile Vorteile gegenüber der Verarbeitung des Signals als Ganzes hat. Dazu implementierten wir eine Filterbank und eine Funktion zur Blockverarbeitung, die in den Abschnitten 3.4.1 und 3.4.2 dokumentiert werden.

Weiterhin verglichen wir verschiedene Implementierungen der sogenannten *Separationsmatrix*, eine Diagonalmatrix zur Gewichtung der Eigenwerte, mit mehreren Methoden und untersuchten die Auswirkungen. Diese verschiedenen Implementierungen werden in Abschnitt 2.5 beschrieben.

2.1 Ansatz

Der in diesem Absatz beschriebene Ansatz bezieht sich auf die in [JE13] beschriebene Methode zur Trennung von Diffus- und Direktschall.
Als ersten Schritt der Verarbeitung wird die Signal-Matrix $\mathbf{B}_L$ aus den vorgegebenen Signalen zusammengesetzt. Dazu werden alle K Kanäle des Signals, als Vektoren $b_k[n]$ dargestellt, übereinander gelegt und zu einer Matrix zusammengefasst:

$$\mathbf{B}_L = [b_1[n], b_2[n], \ldots, b_K[n]]^T. \tag{1}$$

Ein Ambisonics-Signal der Ordnung L besteht aus $K = (L+1)^2$ Kanälen, wobei jedes Signal $b_k[n]$ in Vektorform die zeitlichen Abtastwerte zu den Zeitpunkten $n = 1,2,\ldots,N$ darstellt. Dementsprechend erhält man z.B. für Ambisonics 4. Ordnung $K = 25$ Kanäle, während für Stereo $K = 2$ und für 5.1-Surround $K = 6$.

Als nächsten Schritt multiplizieren wir die Matrix $\mathbf{B}_L$ mit der transponierten Form ihrer selbst und erhalten dadurch $\mathbf{C}_L$ der Größe $K \times K$, die wir im Folgenden als Korrelationsmatrix bezeichnen.

Die Korrelations-Matrix $\mathbf{C}_L$ lässt sich mittels Eigenwertzerlegung in ihre Eigenvektor-Matrix $\mathbf{V}$ und die Diagonalmatrix $\mathbf{S}$, welche auf der Hauptdiagonale die Eigenwerte s_i enthält, zerlegen. Dadurch ergibt sich

$$\mathbf{C}_L = \mathbf{B}_L\mathbf{B}_L^T = \mathbf{V}\mathbf{S}\mathbf{V}^T. \tag{2}$$

Die Eigenwerte s_1 bis $s_{(L+1)^2}$ sind dabei in absteigender Reihenfolge nach ihren Wert sortiert, sodass $s_1 \geq s_2 \geq s_3 \geq \cdots \geq s_{(L+1)^2}$:

$$\mathbf{S} = \begin{pmatrix} s_1 & 0 & \dots & 0 \\ 0 & s_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & s_{(L+1)^2} \end{pmatrix}. \tag{3}$$

Der wichtigste Schritt des Algorithmus ist die nun folgende Berechnung der Separations-Matrix $\mathbf{A}$ mithilfe der Gewichtungsmatrix $\mathbf{\Gamma}$:

$$\mathbf{A} = \mathbf{V}\mathbf{\Gamma}\mathbf{V}^T. \tag{4}$$

$\mathbf{\Gamma}$ ist eine Diagonalmatrix und berechnet sich wie folgt:

$$\mathbf{\Gamma} = \begin{pmatrix} \gamma_1 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & \dots & 0 & \gamma_{(L+1)^2} \end{pmatrix}, \tag{5}$$

$$\text{mit } \gamma_i = \sin\left(\frac{\pi s_{(L+1)^2}}{2 s_i}\right)^2. \tag{6}$$

Dabei ist $\gamma_i = v(s_i)$ eine Funktion der Eigenwerte s_i. Die in Gleichung 6 beschriebene Funktion $v(s_i)$ ist die von [JE13] vorgeschlagene Kennlinie. Im unseren Versuchen haben wir noch weitere Kennlinien implementiert und verglichen, vgl. Kapitel 2.5.

Durch $v(s_i)$ werden die Eigenwerte s_i so manipuliert, dass die daraus folgenden γ_i Werte nahe 1 für kleine s_i ergeben und kleine Werte γ_i (nahe 0) für große s_i. Große Eigenwerte stehen im Zusammenhang mit einer hohen Korrelation der Kanäle und lassen sich demnach Direktschallkomponenten zuordnen, für kleine Eigenwerte ist das Gegenteil der Fall und sie lassen sich Diffusschallkomponenten zuordnen. Durch gewichtete Projektion erreichen wir eine Aufteilung in zwei Unterräume. Dadurch gewichtet $\mathbf{\Gamma}$ die Diffusschallkomponenten stärker als die Direktschallkomponenten.

Als letzten Schritt wird die Separationsmatrix $\mathbf{A}$ dazu benutzt, den Diffusschallanteil $\mathbf{B}_L^{(dif)}$ zu berechnen. Mit $\mathbf{I} - \mathbf{A}$ wird hingegen der Direktschallanteil berechnet.

Diese Vorgehensweise zeigt Gleichung 7:

$$\mathbf{B}_L^{(dif)} = \mathbf{A} \cdot \mathbf{B}_L, \qquad \mathbf{B}_L^{(dir)} = (\mathbf{I} - \mathbf{A}) \cdot \mathbf{B}_L. \tag{7}$$

Man sieht, dass $\mathbf{A}$ und $\mathbf{I} - \mathbf{A}$ zueinander komplementäre Projektionsmatrizen sind.

2.2 Filterbank

2.2.1 Motivation

Mit der in Abschnitt 2.1 beschriebenen Methode wird nun ein Signal in Komponenten aus Richtungen zerlegt, die dem Direktschall zugeordnet werden, sowie in solche, die dem Diffusschall zugeordnet werden. Dies erfolgt gleichförming über den gesamten Frequenzbereich; möglicherweise verschiedene Direktschallrichtungen bei verschiedenen Frequenzen werden dabei nicht berücksichtigt. Das kann zum Beispiel dann zu Problemen führen, wenn entweder der Nachhall oder die Abstrahlcharakteristik der Schallquelle frequenzabhängig ist, was beides mehr die Regel als die Ausnahme ist. Dies ist der Tatsache geschuldet, dass die meisten Schallquellen allein durch ihre physikalischen Abmessungen zu hohen Frequenzen hin bündeln, sobald die Wellenlänge des abgestrahlten Schalls in die Größenordnung der Schallquellenabmessung kommt. Weiterhin kann es zu Problemen kommen, wenn auf einer Musikaufnahme zwei Instrumente mit unterschiedlichem Frequenzbereich, beispielsweise eine Flöte und ein Cello, aus zwei unterschiedlichen Richtungen spielen. In diesem Fall ist der Algorithmus nicht in der Lage, zwischen dem Direktschall der Flöte und demjenigen Anteil des Cello-Diffusschalls, der aus der Richtung der Flöte kommt (und umgekehrt), zu unterscheiden.

Aufgrund dieser Überlegungen entschieden wir uns dazu, eine Filterbank zu implementieren und anschließend den Separationsalgorithmus getrennt auf die einzelnen Frequenzbänder anzuwenden.

2.2.2 Implementierung

Die Filterbank wurde in der MATLAB-Funktion

```
function Y=filterbank(X,fs,no_of_bands,length_factor)
```

implementiert. Diese Funktion bearbeitet eine Signalmatrix `X`, deren Spalten die zu bearbeitenden Signalvektoren enthalten. Weitere Parameter sind die Signal-Abtastrate `fs`, die gewünschte Anzahl an Frequenzbändern `no_of_bands` sowie der Faktor `length_factor`. Die Funktion gibt die dreidimensinale Signalmatrix `Y` zurück. In der ersten Dimension dieser Matrix verläuft die Zeitachse; die Länge der Matrix in der ersten Dimension beträgt dabei das `length_factor`-fache der ursprünglichen Länge von `X`, um den Filter-Impulsantworten Zeit zum Abklingen zu geben. Die zweite Dimension indiziert wie im Eingangssignal unverändert die einzelnen Kanäle. In der dritten Dimension sind schließlich in aufsteigender Reihenfolge die verschiedenen Frequenzbänder untergebracht.

Die `filterbank`-Funktion liefert `no_of_bands` Frequenzbänder im akustisch relevanten Bereich von 20 Hz bis 20 kHz. Der Frequenzbereich wird in Bänder mit konstanter relativer Bandbreite zerlegt. In einer Darstellung mit logarithmischer Frequenzachse decken die Bänder damit gleiche Abschnitte ab, wie in Abbildung 1 zu sehen ist.

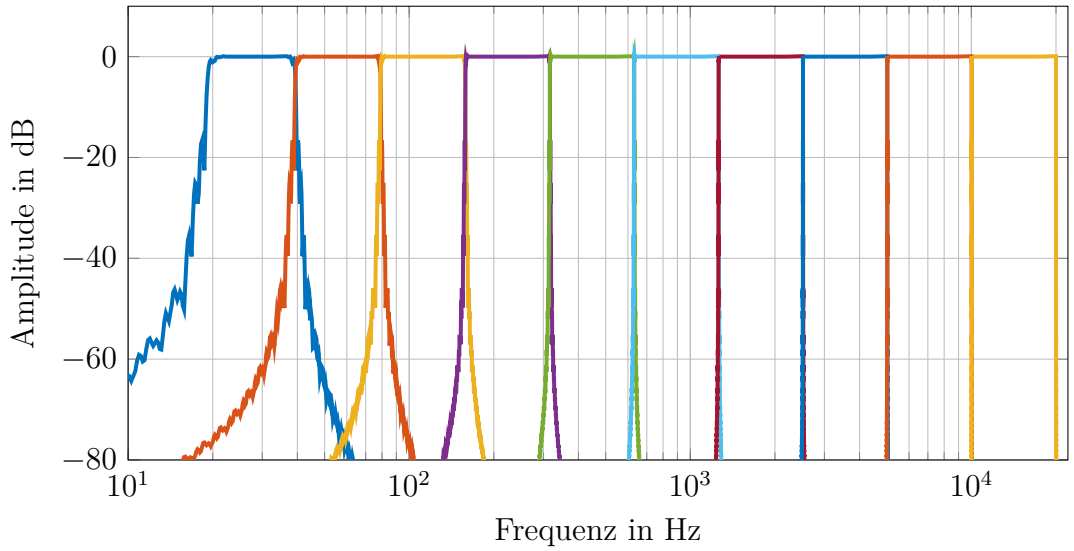

Abbildung 1 – Frequenzgänge der im Zeitbereich gefensterten Bänder mit $2^{15} = 32k$ Samples Filterlänge und zehn Frequenzbändern

Die eigentliche Filterung erfolgt im Frequenzbereich. Daher werden zunächst alle Eingangssignale durch das Auffüllen mit Nullen (Zero-Padding) auf `length_factor`-fache Länge gebracht und mithilfe der Fast Fourier Transformation (FFT) in den Frequenzbereich transformiert. Wir haben in diesem Projekt den `length_factor` $= 2$ gesetzt. Im Frequenzbereich erfolgt die spektrale Filterung über einfache Rechteckfenster, die für jedes Frequenzband alle spektralen Komponenten, deren Frequenz außerhalb des Bandes liegt, zu Null setzen. Anschließend erfolgt die Rücktransformation über eine symmetrische Inverse Fourier Transformation (iFFT). Zum Ausblenden der Sinc-Funktionen, die als Folge der Rechteckfensterung im Frequenzbereich im Zeitbereich als Filter-Impulsantworten auftreten, wird am Anfang und am Ende des Zeitsignals jeweils ein halbes Hann-Fenster angewendet. Dieses wird über die MATLAB-Funktion `tukeywin` realisiert (siehe Abbildung 2). Diese Fensterung hat eine vernachlässigbare Auswirkung auf die Frequenzantworten der Filterbankkanäle.

Die Filterwirkung ist im Einsatz mit endlichen Impulsantworten niemals ideal aber für unseren Einsatzzweck ausreichend genau. Die Filterflanken sind steil und weisen nur leichte Nebenkeulen auf, wobei die erste schon mehr als 15 dB gedämpft ist (siehe Abbildung 3).

Nach der Summation aller Bänder ergibt sich über den Frequenzbereich von 20 Hz bis 20 kHz ein flacher Frequenzgang. Die Rekonstruktion des Signales funktioniert also ordnungsgemäß (Abbildung 4). Durch die Verwendung von symmetrischen Impulsantworten ist die Filterbank komplett linearphasig.

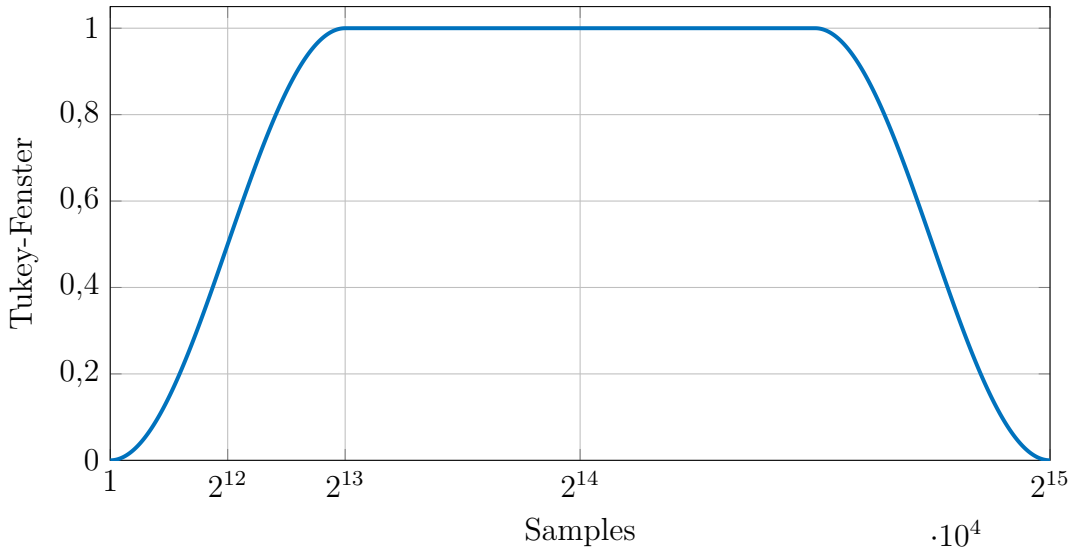

Abbildung 2 – Fenster zum Ausblenden der Filter-Impulsantwort

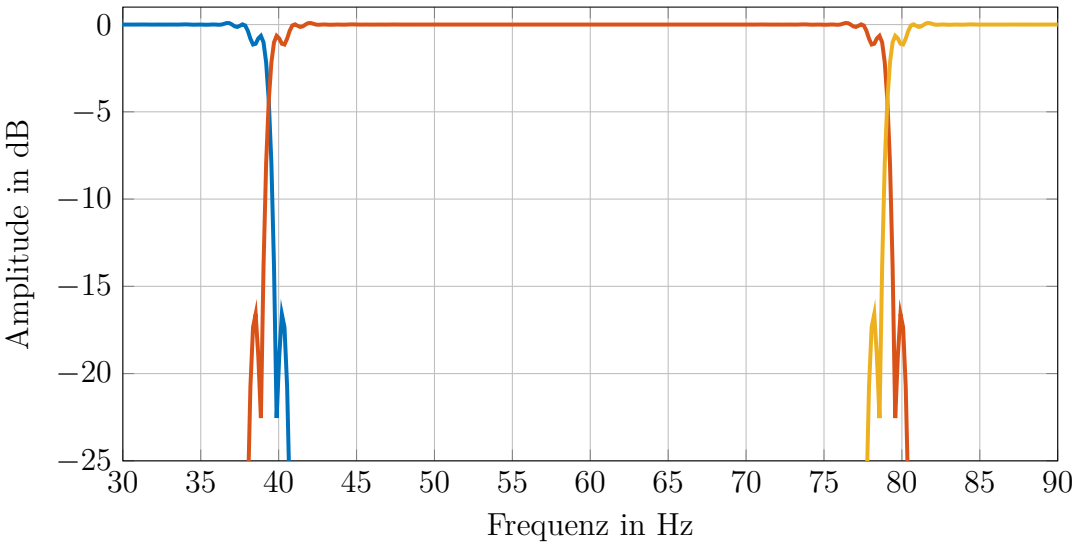

Abbildung 3 – Detailansicht eines Frequenzbandes mit $2^{15} = 32k$ Samples Filterlänge und zehn Frequenzbändern

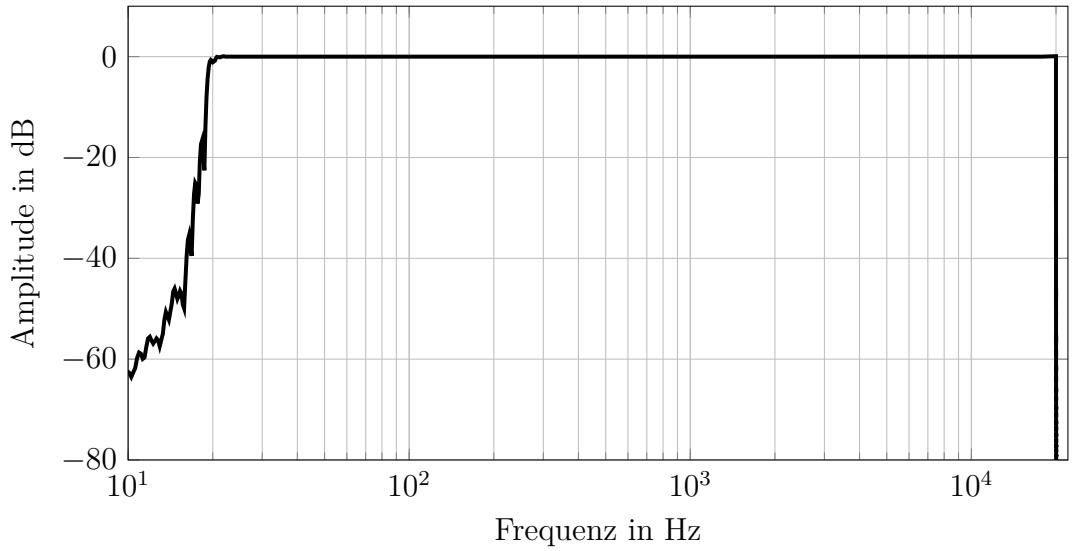

Abbildung 4 – Gesamtfrequenzgang der Filterbank mit $2^{15} = 32k$ Filterlänge und zehn Frequenzbändern

2.3 Blockverarbeitung

2.3.1 Motivation

Der in Abschnitt 2.1 beschriebene Algorithmus geht von langzeitstationären Signalen aus, zeitliche Veränderungen der Einfallsrichtungen von Direkt- und Diffusschall werden also bei der Verarbeitung nicht berücksichtigt.

Ein weiteres Problem, das mehr der praktischen Implementierung entspringt, ist, dass FFTs über Signale mit mehreren Millionen Abtastwerten von heutigen Rechnern nur relativ ineffizient zu berechnen sind. Dies hat zwei Gründe: Zum einen ist die Berechnung sehr langer FFTs durch limitierte Speichergrößen in aktuellen Rechnern ineffizient. Zum anderen benötigen FFTs mit größerer Längen zunehmend mehr Operationen pro Zeitabschnitt. So beträgt mit $N = 1024$ die Komplexität der Berechnung von zwei FFTs $\mathcal{O}(N) = 2 \cdot N \operatorname{ld}(N) = 2 \cdot 1024 \cdot 10 = 20480$, während mit $N = 2048$ die Berechnung einer einzelnen FFT die Komplexität $\mathcal{O}(N) = N \operatorname{ld}(N) = 2048 \cdot 11 = 22528$ beträgt.

Wie in Kapitel 3.4.1 beschrieben, ist die von uns implementierte Filterbank FFT-basiert, und daher (zumindest auf aktuellen Rechnern) nur eingeschränkt für sehr lange Signale geeignet. Daher ermöglicht eine Zerlegung des Signals in kürzere zeitliche Blöcke die effiziente Anwendung unserer Filterbank.

Aufgrund dieser Überlegungen haben wir eine zeitliche Blockzerlegung sowie ein überlappendes Summierungsverfahren implementiert und den Separationsalgorithmus auf einzelne zeitliche Blöcke des Signals angewendet, um sie hinterher wieder aufzusummieren.

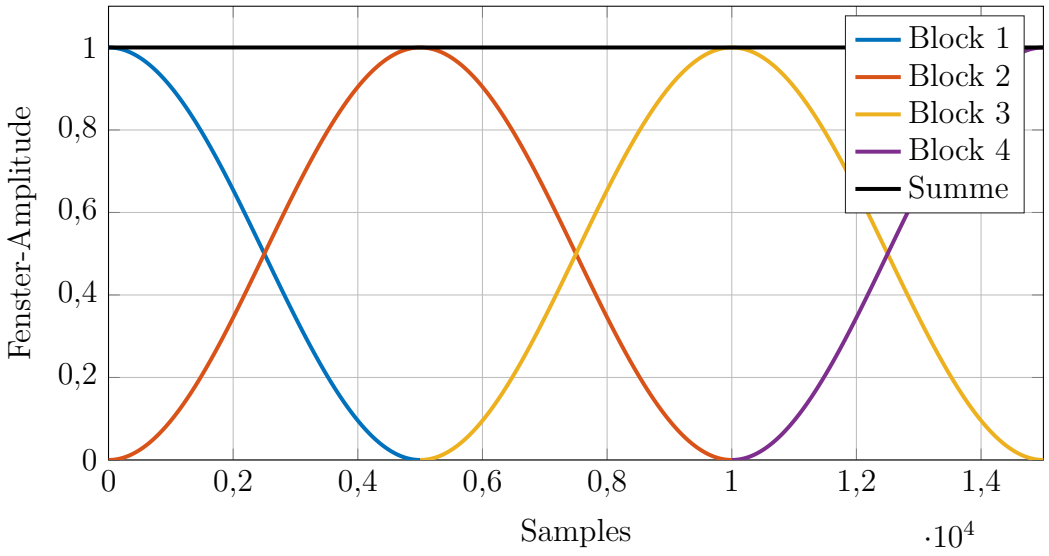

Abbildung 5 – Zeitlicher Verlauf einiger Blöcke mit einer Blockgröße von 10000 Samples

2.3.2 Implementierung

Die Implementierung erfolgt in der `ProcessSignal`-Funktion, für Informationen zur Struktur dieser Funktion siehe Kapitel 2.4.

Die Blockzerlegung erfolgt über zeitliche Hann-Fenster mit einem Überlappungsfaktor von 50 %. Daraus ergibt sich eine perfekte Rekonstruktion nach der Verarbeitung (siehe Abbildung 5).

Nach der Aufteilung in Blöcke erfolgt die weitere Verarbeitung wie gehabt mit oder ohne Filterbank. Anschließend werden die Ergebnisse wieder aufsummiert. Falls die Verarbeitung mit Filterbank erfolgt, muss dabei darauf geachtet werden, dass die verarbeiteten Blöcke doppelt so lang sind wie die Originalblöcke.

2.4 `ProcessSignal`

Als zentraler Funktionsaufruf wurde von uns die `ProcessSignal`-Funktion geschrieben, welche die diversen Parameter und Optionen des Separations-Algorithmus und der Vorverarbeitungs-Verfahren verwaltet und zentralisiert. Sie ist wie folgt definiert:

```
function [Y_dif, Y_dir]=ProcessSignal(X,fs,separation_method,
        block_length,no_of_bands,filename,energy, ambisonics)
```

Diese Funktion dient als Ausgangspunkt für alle Separations-Aufrufe und implementiert eine Reihe von Parametern. Zunächst wird über `X` das Eingangssignal in gewohnter Form mit den Einzelsignalen als Spaltenvektoren übergeben. `fs` legt die Abtastrate fest.

`separation_method` definiert, welche Separationsmatrix verwendet wird (siehe Kapitel 2.5). Über `block_length` wird die Länge für die zeitliche Blockverarbeitung definiert, ist `block_length` = 0 wird die Blockverarbeitung deaktiviert. Die Anzahl der Frequenzbänder wird über `no_of_bands` festgelegt, siehe dazu Kapitel 3.4.1. Für `no_of_bands` = 1 wird die Filterbank deaktiviert. Die letzten drei Parameter wurden im Wesentlichen für diverse Experimente mit der Separationsmethode benötigt, sind aber für den eigentlichen Algorithmus nicht weiter interessant.

`ProcessSignal` besteht aus einer Fallunterscheidung, welche je nach übergebenen Parametern aus verschiedenen Verarbeitungsmethoden auswählt und die zugehörigen Funktionen aufruft. Ferner implementert `ProcessSignal` die Blockverarbeitung, siehe dazu Kapitel 3.4.2.

2.5 Separationsmatrix

Ein wichtiger Punkt des Separationsalgorithmus ist die Wahl der Separationskennlinie $\gamma_i = v(s_i)$ zur Bestimmung der Gewichtungsmatrix $\mathbf{\Gamma}$. Diese spielt eine wichtige Rolle für den perzeptiven Erfolg des Algorithmus, da sie bestimmt, welche Anteile des Signals dem Direkt- und welche dem Diffusschall zugeordnet werden.

In dem Paper [JE13] von Jin und Epain wird vorgeschlagen, alle Eigenwerte invers auf den kleinsten zu normalisieren und das Ergebnis mit einer angehobenen Cosinus-Funktion zu gewichten. Daraus ergibt sich als Gewichtungsfunktion $v(s_i)$ (siehe Gleichung 6):

$$v_{\texttt{sine}}(s_i) = \sin\left(\frac{\pi s_{(L+1)^2}}{2s_i}\right)^2. \tag{8}$$

Bei unseren stark verhallten Testsignalen zeigte sich nach ersten Tests, dass im Direktanteil noch recht viel Hall vorhanden war. Eine Kennlinie, die insgesamt mehr Signalanteile dem Diffusschall zuordnet, erschien daher ein sinnvoller Lösungsansatz zu sein. Daher haben wir die Gewichtungsfunktion mit Cosinus-Termen höherer Potenzen ersetzt, sodass

$$v_{\texttt{sinX}}(s_i) = 1 - \cos\left(\frac{\pi s_{(L+1)^2}}{s_i}\right)^X. \tag{9}$$

Ein Vergleich dieser Gewichtungsfunktionen ist in Abbildung 6 zu sehen.

Wir haben mit einigen anderen Gewichtungsfunktionen experimentiert, im MATLAB-Code z.B. mit `norm`, `threshold` oder `cutoff` bezeichnet, die aber meist zu keinen zufriedenstellenden Ergebnissen führten und daher nicht weiter dokumentiert werden.

Ein alternativer Ansatz führt aufgrund seiner Einfachheit überraschenderweise zu brauchbaren Ergebnissen, weshalb er kurz beschrieben wird: Bei der `constant`-Methode wird, anders als bei den bisher diskutierten Ansätzen, keine Gewichtungsfunktion auf die invers normalisierten Eigenwerte gelegt; die Eigenwerte werden hier vielmehr nur bestimmt, um die Eigenvektoren nach der Größe der zugehörigen

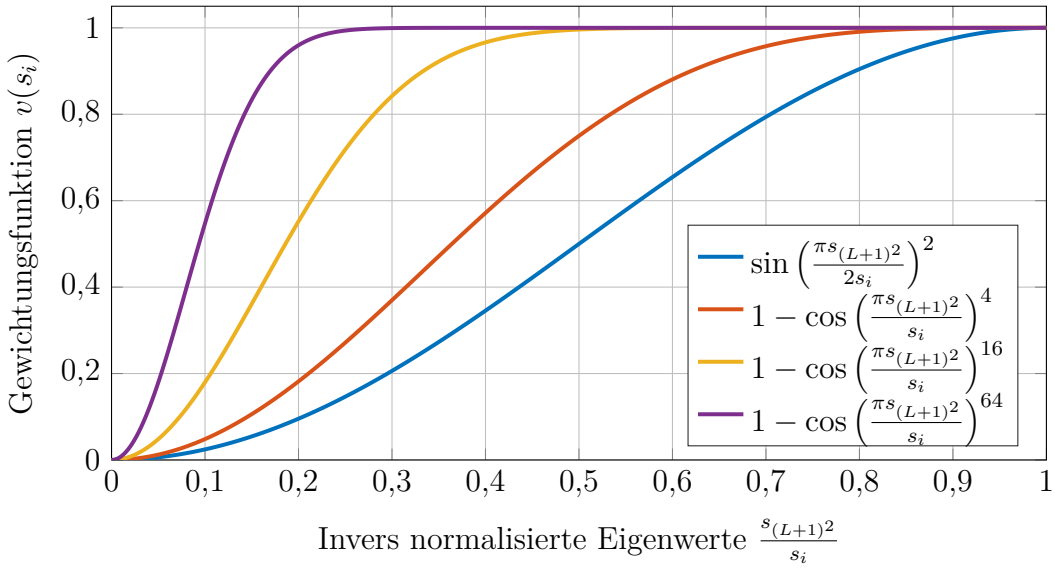

Abbildung 6 – Verschiedene Gewichtungsfunktionen $v(s_i)$

Eigenwerte zu sortieren. Die Separationsmatrix $\mathbf{\Gamma}$ ist hier komplett unabhängig von den Eigenwerten und somit eine Konstante, die rein durch grobes Abschätzen festgelegt wird.

3 Evaluation

In diesem Abschnitt möchten wir überprüfen, ob die beschriebene Umsetzung des Algorithmus zu einer perzeptiv erfolgreichen Trennung von Direkt- und Diffusschallanteilen führt. In den Abschnitten 3.2 und 3.3 wird die Anwendung des Algorithmus auf Stereo- und Surround-Signale evaluiert, Abschnitt 3.4 beschäftigt sich mit der Anwendung des Algorithmus auf Ambisonics-Signale. Die Evaluation beschränkt sich auf unseren perzeptiven Eindruck und wird ergänzt durch die Meinung einiger Testpersonen. Ein formeller Hörversuch war nicht Teil der Seminararbeit.

3.1 Lineare vs. quadratische Trennung

Wie Gleichung 7 zeigt, wird in [JE13] eine lineare Trennung zwischen Direkt- und Diffusschall vorgeschlagen, sodass die Summe der beiden Signalanteile wieder dem Ausgangssignal entspricht. Wir untersuchen zudem, ob es perzeptive Vorteile mit sich bringt, die Trennung energieerhaltend durchzuführen, also mit quadrierten Faktoren. Zu diesem Zweck berechnen wir für den Direktschall eine eigene Gewichtungsmatrix nach dem folgenden Schema:

$$\mathbf{\Gamma}_e = \begin{pmatrix} \sqrt{1-\gamma_1^2} & 0 & \dots & 0 \\ 0 & \sqrt{1-\gamma_2^2} & \dots & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & \dots & 0 & \sqrt{1-\gamma_{(L+1)^2}^2} \end{pmatrix}, \tag{10}$$

und

$$\mathbf{A}_e = \mathbf{V}\mathbf{\Gamma}_e\mathbf{V}^T \tag{11}$$

ergibt folgende Berechnung für Direkt- und Diffusschallanteile:

$$\mathbf{B}_L^{(dif)} = \mathbf{A} \cdot \mathbf{B}_L \qquad\qquad \mathbf{B}_L^{(dir)} = \mathbf{A}_e \cdot \mathbf{B}_L \tag{12}$$

Davon ist im Vergleich zu Gleichung 7 nur $\mathbf{B}_L^{(dir)}$ betroffen.

In Abbildung 7 ist ein Vergleich von linearer und quadratischer Trennung zu sehen. Vergleicht man beide Kurven mit den entsprechenden Plots für andere Separationsmatrizen (Abbildungen 10, 11 und 12), so wird klar, dass die Wahl der Separationsmatrix die Auftrennung in Diffus- und Direktschall wesentlich stärker beeinflusst als die Wahl zwischen linearer oder quadratischer Trennung.

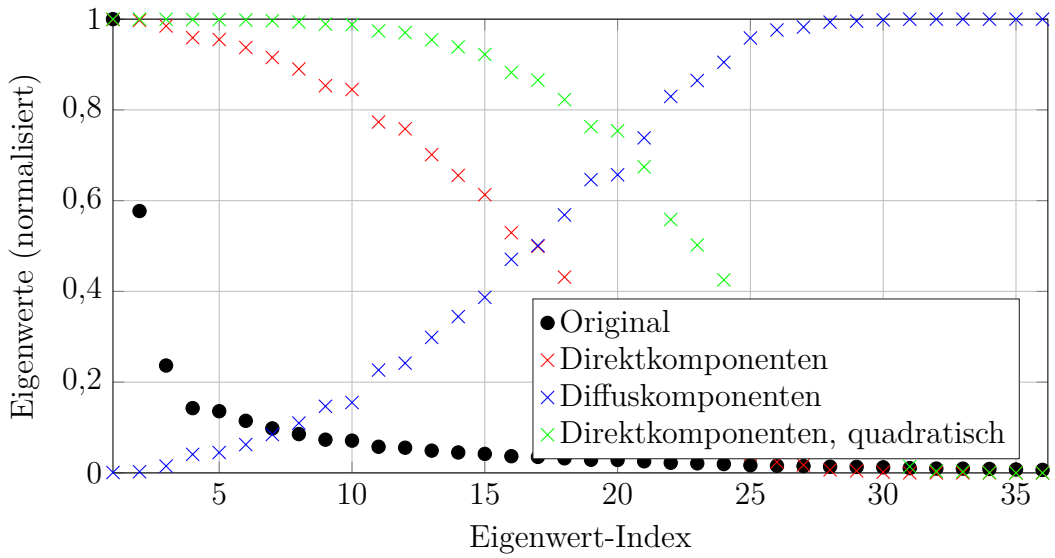

Abbildung 7 – Vergleich von linearer und quadratischer (energetischer) Trennung, Separationsmethode: `sin16`

Wir haben uns daher zunächst für die lineare Variante entschieden, zumal das auch in [JE13] so vorgeschlagen wird, und uns weiters auf die Optimierung der Separationsmatrix konzentriert.

3.2 Evaluation für Stereo-Signale

Im Falle von Stereosignalen ist die Korrelationsmatrix $\mathbf{C}_L$ nur eine 2×2-Matrix, und besitzt daher auch nur zwei Eigenwerte und -vektoren, einer für Direkt- und einer für Diffusschall (vgl. Abbildung 8). Eine solche binäre Trennung wird natürlich in den meisten Fällen den Eigenschaften eines typischen Signals nicht gerecht.

Der Direktschall-Eigenvektor repräsentiert hierbei diejenigen Signalanteile, die korrelliert auf beiden Kanälen vorhanden sind, während der Diffusschall-Eigenvektor alle Signalanteile enthält, welche auf nur auf einem der beiden Kanäle enthalten sind. Damit entspricht das Verfahren dem Vorgehen bei einer M/S-Kodierung. Deshalb ist der Separationsalgorithmus für Stereo-Signale zwar lehrreich aber nicht sonderlich interessant, da die M/S-Kodierung schon lange bekannt ist und wesentlich einfacher umzusetzen ist.

Unabhängig davon gelingt eine ansatzweise Trennung zwischen Direkt- und Diffusschall für exakt mittig ausgerichtete Schallquellen („Karaoke-Effekt“). Für außermittig panoramisierte Schallquellen funktioniert die Trennung zwischen Direkt- und Diffusschall hingegen erwartungsgemäß nicht.

Abbildung 8 – Eigenwerte der Korrelationsmatrix für Stereo, Separationsmethode: `constant`

3.3 Evaluation für Surround-Signale

Wir stellten fest, dass die vorgeschlagene Methode zur Direkt-/ Diffusschall-Trennung für 5.1-Surround-Signale nur bedingt funktioniert. Die Korrelationsmatrix $\mathbf{C}_L$ ist hier eine 6×6-Matrix; bei so wenigen Eigenvektoren ist es schwierig, eine sinnvolle Trennung zwischen Diffus- und Direktanteilen zu finden (vgl. Abbildung 9). In unseren Versuchen konnte zwar perzeptiv ein gewisser eingeschränkter Erfolg festgestellt werden, eine wirklich zufriedenstellende Trennung von Diffus- und Direktanteilen kann jedoch nicht erzielt werden.

3.4 Evaluation für Ambisonics-Signale

Allgemein liefert der Separationsalgorithmus für Ambisonics-Signale höherer Ordnung zufriedenstellende Ergebnisse. Es kann bei korrekter Parameterwahl eine deutliche Trennung zwischen Diffus- und Direktschall bei geringer Artefaktbildung festgestellt werden. Über die Wahl der Separationsmatrix kann der Algorithmus entweder die Qualität des Diffus- oder des Direktschallanteils favorisieren.

In den nächsten Abschnitten beschreiben wir die die zentralen Komponenten unseres Algorithmus und unseren perzeptiven Einfluss auf das Ergebnis.

3.4.1 Filterbank

Mit den von uns verwendeten Testsignalen konnte der Einfluss und Nutzen der Filterbank nicht ausreichend untersucht werden. In der Theorie sollte die Verwendung

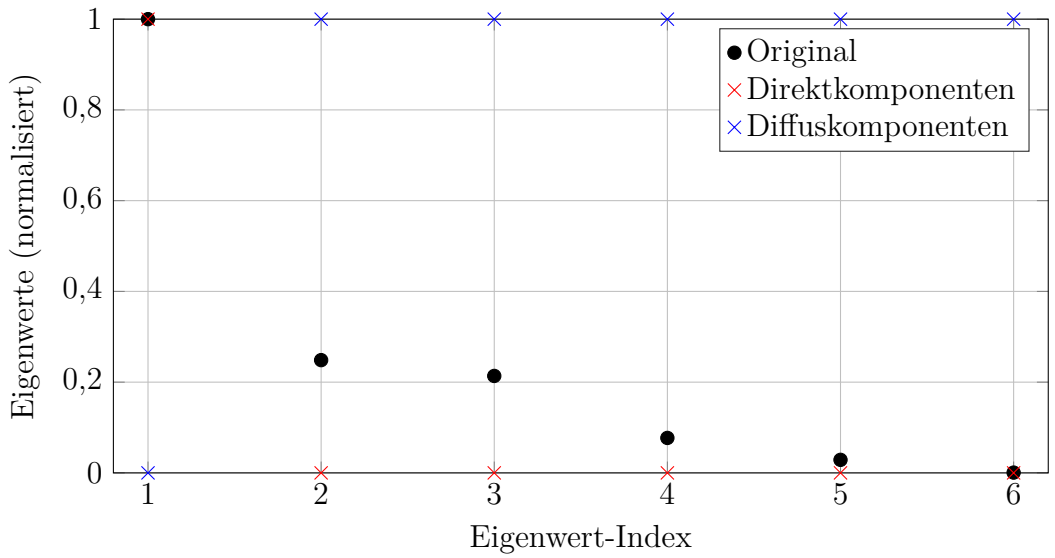

Abbildung 9 – Eigenwerte der Korrelationsmatrix für 5.1, Separationsmethode: `constant`

der Filterbank Vorteile bei dichteren Aufnahmen mit verschiedenen Instrumenten und/oder ungleichmäßigem Nachhall bieten.

Diese Erwartung kann mit den uns zur Verfügung stehenden Ambisonics-Signalen kaum überprüft werden: Das künstlich ambisonisch verhallte Sprachsignal enthält nur eine einzelne Schallquelle, während die Eigenmike-Aufnahme (*Asteroid*) nur relativ wenig Nachhall enthält, was die Evaluation erschwert.

Prinzipiell verschlechtert die Filterbank das Ergebnis in den von uns getesteten Fällen jedoch nicht, auch wenn sie keinen wirklichen Vorteil bietet. Ob die Filterbank für komplexere Signale tatsächliche Vorteile bietet, müsste weitergehend untersucht werden.

3.4.2 Evaluation der Blockverarbeitung

Die Blockverarbeitung ermöglicht es der Theorie nach, auch für nicht-stationäre Signale die Trennung von Direkt- und Diffusschallanteilen zeitvariant durchzuführen. Bei nicht-stationären Signalen ändern sich die Einfallsrichtungen des Direktschallsignal im Verlauf der Zeit. Dies hat Auswirkungen auf die Eigenwerte des Signals, die sich ebenfalls zeitlich stark ändern. Ohne Blockverarbeitung könnte so der zeitliche Verlauf nicht berücksichtigt werden und es würden Direkt- und Diffusschallrichtungen für die gesamte Länge des Signals festgelegt.

Für das Sprachsignal brachte die Verwendung der Blockverarbeitung keinen wirklichen Vorteil, da die Sprecherposition konstant ist und dieses Signal damit stationär im Sinne der Direkt- und Diffusschallrichtungen ist. Bei der Eigenmike-Aufnahme ergab sich bei der Evaluation wiederum das Problem, dass nur relativ wenig Nachhall enthalten ist.

Allgemein führt die Verwendung der Blockverarbeitung in Verbindung mit einer ungeeigneten Separationsmatrix schnell zu Artefakten in Form eines „Pumpens“, da sich die Separationsmatrizen aufeinanderfolgender Blöcke mitunter zu stark unterscheiden. Die Blocklänge sollte also nicht zu kurz gewählt werden, um diese Artefakte zu minimieren, andererseits jedoch aber nicht zu lang um eine ausreichend schnelle Adaption auf geänderte Schalleinfallsrichtungen zu gewährleisten. Damit hängt die optimale Blocklänge stark vom Signal ab, bewährt hat sich bei uns der Bereich von knapp einer Sekunde bis zu wenigen Sekunden. Auch hier könnten weitere Experimente mit geeigneteren Testsignalen durchgeführt werden, um die optimale Blocklänge genauer spezifizieren zu können.

3.4.3 Evaluation der Separationsmatrix

Bei unseren Versuchen wurde schnell klar, dass die Separationsmatrix, bzw. die Art wie diese berechnet wird, wesentlich über den perzeptiven Erfolg des Algorithmus entscheidet. Die in [JE13] vorgeschlagenen Separationsmatrix erschien uns für unser stark verhalltes Sprachsignal nur eingeschränkt geeignet, da der Direktschallanteil des Signals noch sehr viel Diffusschall enthielt. Dies sieht man auch deutlich im entsprechenden Eigenwertplot (siehe Abbildung 10).

Daher haben wir einige alternative Methoden zur Gewinnung der Separationsmatrix implementiert. In unseren Experimenten hat sich die in [JE13] vorgeschlagene inverse Normierung auf den kleinsten Eigenwert bewährt, allerdings haben wir die Gewichtungskennlinie (vgl. Abschnitt 2.5) modifiziert, um die Trennung in Richtung Direktschall zu verschieben (vgl. Abbildung 11). Dadurch ist im Direktschallsignal deutlich weniger Diffusschall enthalten.

Allgemein lässt sich sagen, dass die Genauigkeit der Trennung vom Verlauf der Eigenwertgewichtung abhängt, beziehungsweise davon, wie aprupt die Trennung der Eigenwerte verläuft. Man kann so zum Beispiel ein sehr trockenes Direktschallsignal und ein Diffusschallsignal mit einigen Direktschallanteilen erzeugen, oder mit einer anderen Gewichtungsfunktion ein Direktschallsignal mit einigen Diffusschallanteilen und dazu ein Signal, das fast ausschließlich aus Diffusschallanteilen besteht.

Die von uns am besten bewerteten Kennlinien sind `sin32` und `sin16`, da sie ein relativ trockenes Direktschallsignal liefern, ohne zu viel Direktschall in den Diffusschallanteil zu verschieben.

Eine weitere getesteter Ansatz ist die `constant` Kennlinie (Abbildung 12), die einen fest definierte Separationsmatrix verwendet, bei der die Größe der Eigenwerte keine Rolle mehr spielt. Diese Methode erzielte zwar auch perzeptiv überzeugende Ergebnisse, muss jedoch vermutlich für jedes Signal individuell festgelegt werden, um die Trennung auf die räumliche Situation der Aufnahme anzupassen.

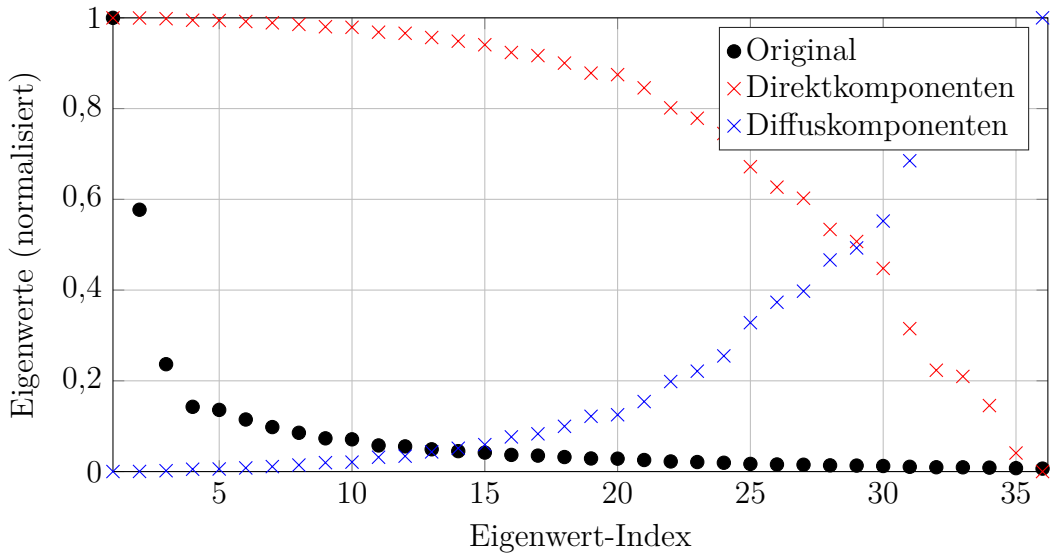

Abbildung 10 – Eigenwerte der Korrelationsmatrix und deren Gewichtung, Ambisonics 5. Ordnung, Separationskennlinie: `sine`

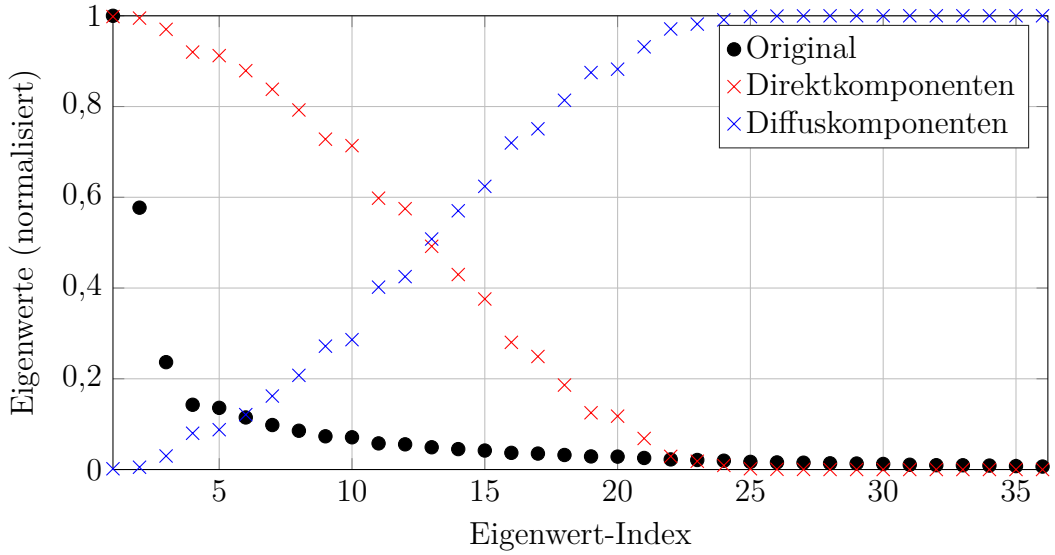

Abbildung 11 – Eigenwerte der Korrelationsmatrix und deren Gewichtung, Ambisonics 5. Ordnung, Separationskennlinie: `sin32`

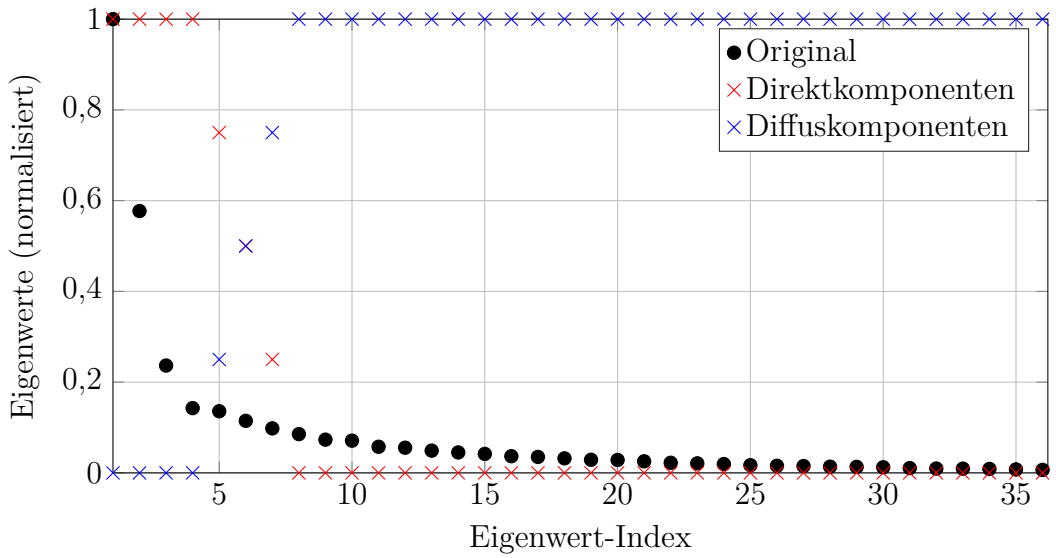

Abbildung 12 – Eigenwerte der Korrelationsmatrix und deren Gewichtung, Ambisonics 5. Ordnung, Separationskennlinie: `constant`

4 Schlussfolgerung

Zusammenfassend können wir festhalten, dass die vorgeschlagene Methode für Ambisonics Signale überzeugende Ergebnisse liefert. Um den Einfluss und die optimale Parametrisierung der Blockverarbeitung und der Filterbank genauer analysieren zu können, wären weitere Versuche mit geeigneten Ambisonics-Signalen nötig.

Der Wahl der Separationsmatrix ist von zentraler Bedeutung, da sie wesentlich über den perzeptiven (Miss-)Erfolg des Algorithmus entscheidet.

Prinzipiell liefert der Algorithmus interessante Ergebnisse, funktioniert aber nicht unbedingt automatisch zur perzeptiv erfolgreichen Enthallung beliebige Signale. Ursprünglich war er dafür aber auch nicht vorgesehen, sondern wurde als Vorverarbeitungsschritt für Richtungslokalisation entworfen.

5 Ausblick

Bei einem allgemeinen, nicht-stationären Signal verändert sich die Struktur der Eigenwerte mit der Zeit, was es erforderlich macht, zu unterschiedlichen Zeitpunkten unterschiedliche Separationsmatrizen anzuwenden. Bei der Blockverarbeitung haben wir durch Zerlegung in einzelne zeitliche Blöcke versucht, dieses Problem zu lösen. Dabei haben wir bisher aber nur fixierte Blocklängen verwendet ohne dabei genauer auf den tatsächlichen Verlauf der Eigenwertstruktur einzugehen.

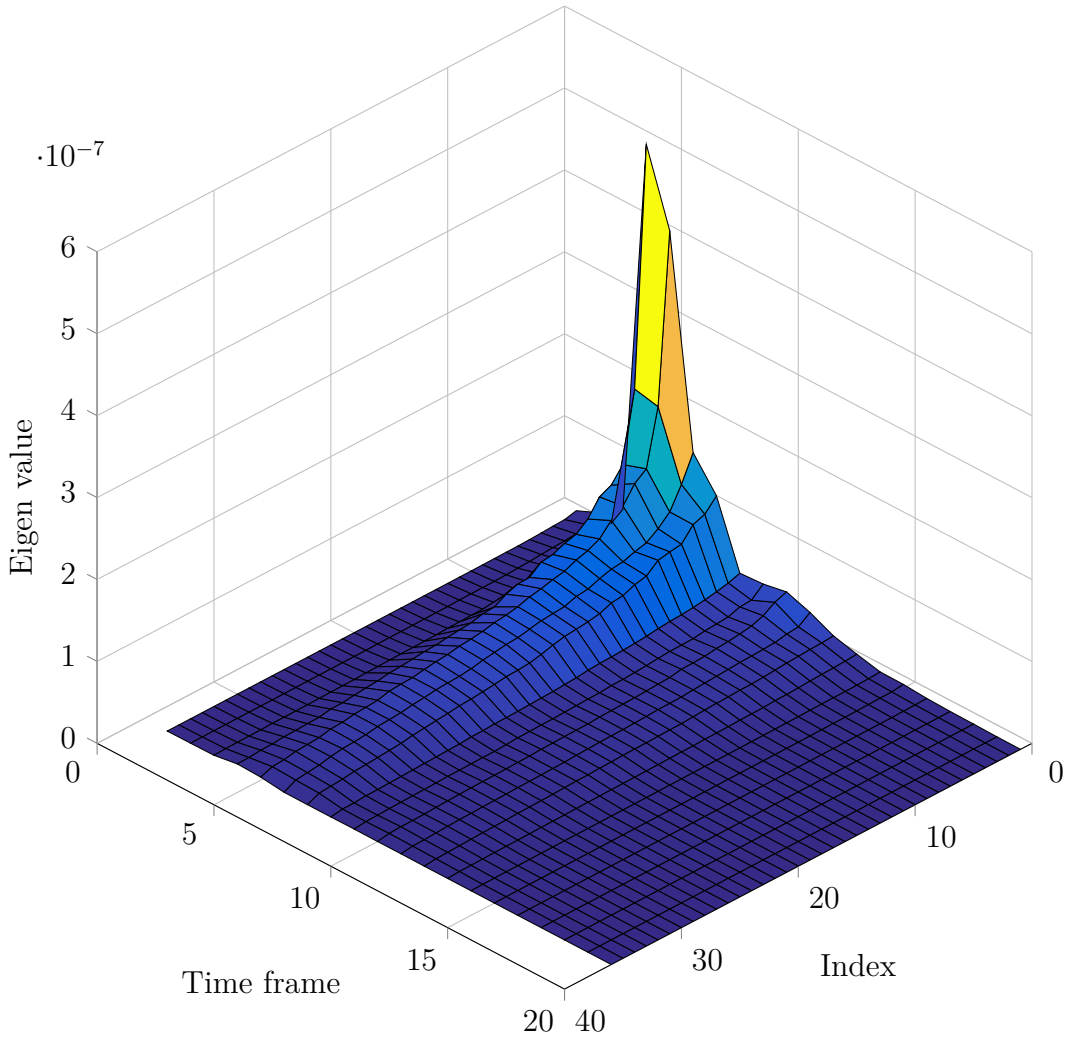

Abbildung 13 – Zeitlicher Verlauf der Eigenwerte, Sprach-Signal, Ambisonics 5. Ordnung

Der zeitliche Verlauf der Eigenwerte eines Ausschnitts aus dem Ambisonics-Sprachsignal ist in Abbildung 13 dargestellt. Es handelt sich dabei um eine Aufnahme des Wortes „Christmas“ mit Ein- und Ausklingvorgang des Nachhalls, dessen Eigenwerte über dem zeitlichen Verlauf dargestellt sind. Man sieht deutlich, dass sich das Verhältnis von größtem zu kleinstem Eigenwert mit der Zeit ändert. Zur Quantifizierung dieses Verhältnisses kann die Konditionszahl κ verwendet werden. Sie wird häufig für die Beschreibung von Eigenwerten verwendet und beschreibt allgemein das Verhältnis zwischen größtem und kleinstem Eigenwert einer Matrix,

$$\kappa = \frac{s_{max}}{s_{min}}. \tag{13}$$

Wie man erkennen kann, ändert sich die Konditionszahl in Abbildung 13 stark mit der Zeit. Eine mögliche Idee zur Verbesserung des Algorithmus wäre nun, die Berechnung der Separationsmatrix dynamisch zu gestalten und z.B. von der Konditionszahl κ abhängig zu machen.

Für eine dynamische Berechnung der Separationsmatrix könnte man bei einer hohen Konditionszahl von einem hohen Direktschallanteil ausgehen und demnach einen größeren Teil der Eigenwerte für dem Direktschallsignal zuordnen. Umgekehrt kann bei einer niedrigen Konditionszahl davon ausgegangen werden, dass das Signal großteils aus Diffusschall besteht und demnach sollte die Mehrheit der Eigenwerte dem Diffusschallanteil zugeordnet werden. Die praktische Implementierung dieser Überlegungen wurde jedoch im zeitlich begrenzten Rahmen dieser Seminararbeit nicht durchgeführt.

Anhang

Literatur

[JE13] C. Jin and N. Epain, "Super-resolution sound field analysis," *Cutting Edge in Spatial Audio*, 2013.

Matlab Code

Main.m

Listing 1 – Main.m

```
close all;
clear variables;
clc;

ambisonics = false;
%Read audio file
%file = '01 Les passants.wav';
%filename = '2channel';

%file = 'maroni_em32_ambix_o4.wav';
%filename = 'maroni';
%ambisonics = true;

%file = 'Pink Floyd - Time (5.1).wav';
%filename = 'pf5.1';

%file = 'SDM_Signal.wav';
%filename = 'SDM';
%ambisonics = true;

%file = 'asteroid_part.wav';
%filename = 'asteroid';
%ambisonics = true;

file = 'the_coach_was_crowded.wav';
filename = 'coach';
ambisonics = true;

[X,fs] = audioread(['Files\',file]);

%cut off empty channels
if strcmp(file,'maroni_em32_ambix_o4.wav') == 1
    X = X(:,1:25);
end

energy = 'linear';   %linear/quadratic, method for separation

[signal_length,channels] = size(X);

```

```
%% Processing
%  Conduct the separation with different approaches, render all
    results as
%  different files

N = 4;
TYPE = cell(N,1);
TYPE{1}= 'allin1';
%TYPE{2}= 'block';
%TYPE{3}= 'filter';
%TYPE{4}= 'blockfilter';
separation_method = 'sin16';

%% Part 1, all in one
if sum(strcmp(TYPE,'allin1')) == 1
    [B_dif, B_dir] = ProcessSignal(X,fs,separation_method,0,1,
       filename,energy,ambisonics);
    audiowrite(['Results\Bdif-allinone-',filename,'-',
       separation_method,'-',energy,'.wav'],B_dif,fs);
    audiowrite(['Results\Bdir-allinone-',filename,'-',
       separation_method,'-',energy,'.wav'],B_dir,fs);
end

%% Part 2, block-based
if sum(strcmp(TYPE,'block')) == 1
    [B_dif, B_dir] = ProcessSignal(X,fs,separation_method,1000,1,
       filename,energy,ambisonics);
    audiowrite(['Results\Bdif-block-',filename,'-',separation_method,
       '-',energy,'.wav'],B_dif,fs);
    audiowrite(['Results\Bdir-block-',filename,'-',separation_method,
       '-',energy,'.wav'],B_dir,fs);
end

%% Part 3, Filterbank
if sum(strcmp(TYPE,'filter')) == 1
    [B_dif, B_dir] = ProcessSignal(X,fs,separation_method,0,10,
       filename,energy,ambisonics);
    audiowrite(['Results\Bdif-filter-',filename,'-',separation_method
       ,'-',energy,'.wav'],B_dif,fs);
    audiowrite(['Results\Bdir-filter-',filename,'-',separation_method
       ,'-',energy,'.wav'],B_dir,fs);
end

%% Part 4, Filterbank + Block processing
if sum(strcmp(TYPE,'blockfilter')) == 1
```

```
75
76      block_length = 1000;
77      [B_dif, B_dir] = ProcessSignal(X,fs,separation_method,
            block_length,10,filename,energy,ambisonics);
78      audiowrite(['Results\Bdif-filter-block-',filename,'-',
            separation_method,'-',energy,'-ambi.wav'],B_dif,fs);
79      audiowrite(['Results\Bdir-filter-block-',filename,'-',
            separation_method,'-',energy,'-ambi.wav'],B_dir,fs);
80
81  end
```

separate.m

Listing 2 – separate.m

```
1   function [B_dif, B_dir,s] = separate(X,method,filename,energy)
2
3   %Function for separating direct and diffuse components of a
        multichannel signal,
4   %IN:  X      = input signal
5   %     method = method for separation matrix
6   %     file   = path for saving
7   %     energy = criterion for separation, linear/ quadratic
8   %
9   %OUT: B_dif  = diffuse part
10  %     B_dir  = direct part
11
12  %to match the notation in the paper
13  B = X';
14
15  %Calculate correlation matrix
16  C = B * B';
17
18  %Eigenvalue decomposition
19  [V,S] = eig(C);
20
21  %sort eigenvalues and vectors
22  [s,I] = sort(diag(S), 'descend');
23  V = V(:, I);
24
25  %inverse-normalize to the smallest eigenvalue
26  sn = s(end) ./ s;
27
28  if strcmp(method,'sine') == 1
```

```
    v = sin((pi/2)*sn).^2;
elseif strncmp(method,'sin',3) == 1
    exponent = sscanf(method,'%*3c %d');
    v = 1-cos((pi/2)*sn).^exponent;
elseif strcmp(method,'norm') == 1
    v = (1-s./s(1));
elseif strcmp(method,'normsin') == 1
    v = s./s(1);
    v = cos((pi/2).*v).^16;
elseif strcmp(method,'cutoff') == 1
    ratio = 0.1;
    len = length(sn);
    v = [zeros(floor(len*(1-ratio)),1); ones(ceil(len*ratio),1)];
elseif strcmp(method,'threshold') == 1
    ratio = 0.1;
    v = zeros(length(sn),1);
    v(s < ratio * s(1)) = 1;
elseif strcmp(method,'linear') == 1
    v = sn;
elseif strcmp(method,'constant') == 1
    direigs = ceil(0.1*length(sn));

    v = [zeros(direigs,1); (1/direigs:1/direigs:1)'; ones(length(sn)
       - 2*direigs,1)];
end

%calculate inverse v for quadratic computation
v_= sqrt(1 - v.^2);

%create Gamma-Matrix
G=diag(v);
G_=diag(v_);

%Plot of eigen values and separation; leave commented for normal use
%plot_eigs(s,v,filename,method,energy);

%Compute A
A = V * G * V';
A_ = V * G_ * V';


%Render signals

B_dif = (A * B)';

```

```
73 if strcmp(energy,'linear') == 1
74     I = eye(size(A,1));
75     B_dir = ((I - A) * B)';
76 elseif strcmp(energy,'quadratic') == 1
77     B_dir = (A_ * B)';
78 else
79     error('Method for separation undefined')
80 end
81 end
```

Filterbank.m

Listing 3 – Filterbank.m

```
function Y=filterbank(X,fs,no_of_bands,length_factor)
%Decomposes a multichannel signal into multiple constant-interval frequency
%bands.
%IN:     X = input matrix where the columns are individual signal vectors
%        fs= sample rate of input signal
%        no_of_bands = number of frequency bands
%        length_factor = the output signal will be longer than the input
%        signal by this factor to allow space for the filter impulse
%        response
%OUT:    Y = three-dimensional filtered output signal; the frequency bands
%        are indexed by the third dimension in ascending order
    f0 = 20;
    f1 = 20000;
    [len, channels] = size(X);

    len = len*length_factor;

    %calculations are done with logarithmic frequencies, normalized to the
    %signal length
    logf0 = log10(f0/fs*len);
    logf1 = log10(f1/fs*len);

    bandwidth = (logf1 - logf0) / no_of_bands;
    offset = logf0;

    %allocate output vector
    Y = zeros(len, channels, no_of_bands);

    %zeropad X for the appropriate oversampling and coumpute its spectrum
    Xpad = [zeros(ceil((len - length(X))/2),channels); X; zeros(floor((len - length(X))/2),channels)];
    spectrum = fft(Xpad, len);

    window = tukeywin(len);

    %loop through all frequency bands of all channels
```

```
36      for i=1:no_of_bands
37          %calculate the sample boundaries of the current frequency
               band
38          start = ceil(10^offset);
39          offset = offset + bandwidth;
40          stop = floor(10^offset);
41          for i1 = 1:channels
42              %create a vector that exclusively contains the current
                   frequency band
43              tmp = zeros(len,1);
44              tmp(start:stop) = spectrum(start:stop,i1);
45
46              %'symmetric' option ignores second half of tmp and
                   assumes
47              %hermitian symmetry -> real-valued output
48              Y(:,i1,i) = window .* ifft(tmp, 'symmetric');
49          end
50      end
51  end
```

separateFilterbank.m

Listing 4 – separateFilterbank.m

```
1   function [Y_dif,Y_dir]=separateFilterbank(X,separation_method,
       no_of_bands,fs,oversampling,file,energy)
2   %individually separate diffuse from direct components of a
       multichannel
3   %signal for multiple frequency bands.
4       if no_of_bands < 2  %no filterbank
5           [Y_dif,Y_dir] = separate(X,separation_method,file,energy);
6       else
7           Xf = filterbank(X,fs,no_of_bands,oversampling);
8           [len, channels, ~] = size(Xf);
9
10          Y_dir = zeros(len,channels);
11          Y_dif = zeros(len,channels);
12
13          %separate all frequency bands individually and sum up the
               results
14          for i=1:no_of_bands
15              [Yfdif,Yfdir] = separate(Xf(:,:,i),separation_method,file
                   ,energy);
16              Y_dir = Y_dir + Yfdir;
```

```
            Y_dif = Y_dif + Yfdif;
        end

    end
end
```

ProcessSignal.m

Listing 5 – ProcessSignal.m

```
function [Y_dif, Y_dir]=ProcessSignal(X,fs,separation_method,
   block_length,no_of_bands,filename,energy, ambisonics)
%Function for processing the input signal
%IN:     X = input signal
%        fs= sample rate
%        separation_method   = method for calculating separation
   matrix
%        block_length   = distinguish between block processing or all
    in 1
%        no_of_bands    = bands for filterbank
%        file  = name of input file
%        energy = method for separation matrix, linear/ quadratic
%        ambisonics = should be true for ambisonics input, false
   otherwise
%
%OUT:   Y_dif  = diffuse part
%       Y_dir  = direct part

    if ambisonics
        X=ambiGain(X,true);
    end

    if block_length == 0 && no_of_bands == 1     %all in one
        [Y_dif,Y_dir,eigv] = separate(X,separation_method,filename,
           energy);

    elseif block_length == 0 && no_of_bands > 1  %filter bank, no
       block processing
        [Y_dif,Y_dir,eigv] = separateFilterbank(X,separation_method,
           no_of_bands,fs,2,filename,energy);

    elseif block_length ~= 0 %block processing, with/without
       filterbank

```

```
27          %if the filterbank is active, we have to be aware that it
               increases
28          %the length of the block, in this case by a factor of 2
29          if no_of_bands == 1
30              filter_overlap = 1;
31          else
32              filter_overlap = 2;
33          end
34
35          [len, channels] = size(X);
36
37
38
39
40          %calculate window for block separation
41          window_length = (block_length * fs) / 1000;
42          window = hann(window_length);
43
44          number_of_blocks = ceil((len/(window_length*0.5)) + 1);
45
46          %zero pad beginning and end of signal for an integer block
               count
47          Xpad = zeros((number_of_blocks+1) * window_length * 0.5,
               channels);
48          Xpad(window_length * 0.5 + 1:window_length * 0.5 + len,:) = X
               ;
49
50          %preallocate output matrices
51          Y_dif = zeros(length(Xpad),channels);
52          Y_dir = zeros(length(Xpad),channels);
53
54          %define eigenvalue matrix
55          Eig  = zeros(number_of_blocks,channels);
56
57          %First block
58          block = Xpad(1:window_length,:);
59
60          %multiply the columns of the 'block' matrix elementwise with
               the
61          %'window' vector
62          block = bsxfun(@times, block, window);
63
64          %separate diffuse from direct components
```

```
65          [result_dif,result_dir,Eig(1,:)] = separateFilterbank(block,
                separation_method,no_of_bands,fs,filter_overlap,filename,
                energy);
66
67          %compute start point of actual signal
68          offset = window_length * (filter_overlap - 1) * 0.5;
69
70          %write result to output vector, ignore (i.e. truncate) filter
71          %overlap at the beginning
72          Y_dif(1:window_length + offset,:) = result_dif(offset+1:end
                ,:);
73          Y_dir(1:window_length + offset,:) = result_dir(offset+1:end
                ,:);
74
75          %repeat steps for all other blocks
76          for k = 1:number_of_blocks - 2;
77              start = k * (window_length*0.5) + 1;
78              stop = (k+2) * (window_length*0.5);
79              block = bsxfun(@times, Xpad(start:stop,:), window);
80
81              [result_dif,result_dir,Eig(k,:)] = separateFilterbank(
                    block,separation_method,no_of_bands,fs,filter_overlap,
                    filename,energy);
82
83              start = start - offset;
84              stop = stop + offset;
85              %add result to previous ones (overlap&add)
86              Y_dif(start:stop,:) = Y_dif(start:stop,:) + result_dif;
87              Y_dir(start:stop,:) = Y_dir(start:stop,:) + result_dir;
88          end
89
90          %last block
91          start = (number_of_blocks-1) * (window_length*0.5) + 1;
92          stop = length(Xpad);
93          block = bsxfun(@times, Xpad(start:stop,:), window);
94          [result_dif,result_dir,Eig(end,:)] = separateFilterbank(block
                ,separation_method,no_of_bands,fs,filter_overlap,filename,
                energy);
95          start = start - offset;
96          %ignore (i.e. truncate) filter overlap at the end
97          Y_dif(start:stop,:) = Y_dif(start:stop,:) + result_dif(1:end-
                offset,:);
98          Y_dir(start:stop,:) = Y_dir(start:stop,:) + result_dir(1:end-
                offset,:);
99      else
```

```
        error('Specified separation method is not specified')
    end
        %save('eigs_time.mat','Eig')
        %load('eigs_time.mat');
       % plot_eigs_time(Eig);

    if ambisonics
        Y_dif=ambiGain(Y_dif,false);
        Y_dir=ambiGain(Y_dir,false);
    end
end
```

ambiGain.m

Listing 6 – ambiGain.m

```
function Y=ambiGain(X,inverse)
%Applies a gain of sqrt(n) to the n-th order signal components of an
%ambisonics signal
%IN:     X = input ambisonics signal
%        inverse = if true, the inverse gain is applied such that
   ambiGain(ambiGain(X,false),true)=X
%
%OUT:    Y = output signal
    [length,channels] = size(X);
    Y = zeros(length,channels);

    %For zero-order, gain is 1
    Y(:,1) = X(:,1);

    c=1;
    l=1;
    for i=2:channels
        if c==l     %next order
            c=0;
            l = l+2;
            if inverse
                f = 1/sqrt(l);
            else
                f = sqrt(l);
            end
        end
        c = c + 1;
        Y(:,i) = X(:,i) .* f;
```

```
28      end
29  end
```