universität wien

DIPLOMARBEIT / DIPLOMA THESIS

Titel der Diplomarbeit / Title of the Diploma Thesis

„Mathematische Grundlagen der Signalverarbeitung“

verfasst von / submitted by

Michael Schiller

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Magister der Naturwissenschaften (Mag. rer. nat.)

Wien, 2017 / Vienna, 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 190 406 884
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Lehramtsstudium UF Mathematik UF Informatik und Informatikmanagement
Betreut von / Supervisor:	Priv.-Doz. Dr(USA) Maria Charina

Danksagung

Ich möchte mich ganz herzlich bei meiner Diplomarbeitsbetreuerin Frau Prof. Maria Charina bedanken. Sie hat sich stets viel Zeit genommen, meine Arbeit bei den Zwischentreffen zu besprechen und weiterzuentwickeln. Dabei war sie immer sehr nett und hilfsbereit, was mich beim Schreiben enorm motiviert hat. Ich bin sehr froh, so eine tolle Professorin als Betreuerin gehabt zu haben.

Inhaltsverzeichnis

1 Einleitung **5**

2 Notationen und Definitionen **7**

3 Grundlegende Operationen der Signalverarbeitung **10**
3.1 Laurentpolynome 10
3.2 Upsampling einer Folge 11
3.3 Downsampling einer Folge 13
3.4 Faltung von Folgen 17

4 Zerlegung und Rekonstruktion von Audiodaten **20**
4.1 Einführende Beispiele 20
4.2 Algorithmus zur Zerlegung und Rekonstruktion von Signalen 29
4.2.1 Zerlegung einer MP3-Audiodatei 31
4.2.2 Rekonstruktion einer MP3-Audiodatei 34
4.2.3 Perfekte Rekonstruktion 37
4.2.4 Beispiele mit realen Audiodaten 39
4.3 Konstruktion von Zerleguns- und Rekonstruktionsfolgen . 47
4.3.1 Algorithmus zur Konstruktion von Zerlegungs- und Rekonstruktionsfolgen 47
4.3.2 Beispiele zur Konstruktion von Zerlegungs- und Rekonstruktionsfolgen 50

5 Zusammenfassung **68**

6 Appendix **70**
6.1 Tabelle der Koeffizientenfolgen von $h^*(z)$ 70
6.2 Tabellen der Koeffizientenfolgen von $a^*(z), b^*(z), P^*, Q^*(z)$ 71

7 Quellenverzeichnis **73**
7.1 Literatur 73
7.2 Abbildungen 73

1 Einleitung

Die vorliegende Diplomarbeit beschäftigt sich mit den mathematischen Grundlagen der Signalverarbeitung. Dieses Thema interessiert mich, weil es ein aktuelles Anwendungsgebiet der Mathematik darstellt und eng mit der Informatik, meinem zweiten Unterrichtsfach, verbunden ist. Nachdem ich in meinem zukünftigen Lehrberuf die Mathematik möglichst anschaulich und schülerfreundlich vermitteln möchte, ist es mir auch in meiner Abhandlung wichtig, einen einsteigerfreudlichen Zugang zur faszinierenden Welt der Signalverarbeitung zu bieten. Es werden die Grundlagen der Signalverarbeitung vermittelt, mit Beispielen illustriert und Implementierungen in Matlab präsentiert. Es wird kaum Vorwissen im Bereich der Hochschulmathematik vorausgesetzt. Daher ist die Arbeit für StudienanfängerInnen und auch interessierte SchülerInnen geeignet.

Für die Implementierungen in Matlab wird allerdings die Wavelet Toolbox benötigt.

Die Verarbeitung von Signalen spielt seit einigen Jahrzehnten eine tragende Rolle in den verschiedensten Bereichen: in der Sprach- und Datenkommunikation, der Akustik, der Sonarwissenschaft, der Radartechnik, der Seismologie, der Ölsuche, der Instrumententechnik, in der Robotik, der Kosumelektronik und vielen weiteren. Die Algorithmen und Geräte zur Signalverarbeitung sind in Systemen verbreitet, welche von militärischen Geräten, über industrielle Anwendungen, bis hin zu Massenkonsumartikeln reichen. Faszinierend ist, dass auch bei selbstverständlich gewordenen Unterhaltungsgeräten, wie dem Radio oder dem Fernseher, Signalverarbeitungstechnologie zum Einsatz kommt, die an der Grenze des technisch Möglichen arbeitet.

Signale werden als Kommunikationsmittel zwischen Menschen und zwischen Menschen und Maschinen eingesetzt. Mit Hilfe von Signalen kann Information dargestellt und nutzbar gemacht werden. Sie werden auch verwendet, um die Umgebung zu untersuchen und Details von Strukturen aufzudecken [1, S.1-2].

Es gibt verschiedene Arten von Signalen. Das mathematische Modell eines Signals ist häufig eine Funktion, die Werte in Abhängigkeit von der Zeit beschreibt. In dieser Arbeit wird nur das Modell, des sogenannten zeitdiskreten Signals, verwendet. Der Fokus liegt auf der Verarbeitung von Audiosignalen. Ein diskretes Audiosignal kann beispielsweise eine Folge sein, die aus zu bestimmten Zeitpunkten gemessenen Frequenzen eines Audiotons besteht.

Signalverarbeitung ist definiert als Zerlegung, Analyse, Manipulation und Speicherung von Signalen und den enthaltenen Informationen. Mit den Methoden der Signalverarbeitung können beispielweise zwei oder

mehr Signale, die miteinander verbunden sind, voneinander getrennt oder gewisse Komponenten verstärkt/abgeschwächt werden. Mit dem Verfahren können unter anderem unerwünschte Störsignale aus einer Audioaufnahme herausgefiltert werden.

Das Hauptaugenmerk dieser Diplomarbeit ist auf das Zerlegen und die Rekonstruktion von Audiosignalen gerichtet. Es wird gezeigt, wie dadurch die Frequenzen von Audiosignalen analysiert und verarbeitet werden können.

Zu Beginn der Arbeit, in Kapitel 2, werden einige Notationen und wesentliche Definitionen eingeführt. Es ist ratsam dieses Kapitel sorgfältig zu lesen, um ein flüssiges und unmissverständliches Lesen der Arbeit zu ermöglichen.

In Kapitel 3 werden die grundlegenden Operationen der Signalverarbeitung vorgestellt. Dazu gehören das Up- und Downsampling und die Faltung. Diese Operationen werden für die Algorithmen 4.5 und 4.7, im darauffolgenden Kapitel 4, benötigt.

In Kapitel 4 geht es um die Zerlegung und Rekonstruktion von Audiosignalen. Zuerst werden, in Unterkapitel 4.1, einführende Beispiele gezeigt. Danach wird das mathematische Modell eines Signals vorgestellt und erklärt, was Signalverarbeitung ist. Im Anschluss werden die Algorithmen 4.5 und 4.7, zur Zerlegung und Rekonstruktion von diskreten Audiosignalen, beschrieben und begründet. Im darauffolgenden Unterkapitel 4.2.4 werden die Algorithmen 4.5 und 4.7 auf reale Audiodaten angewandt. Es wird beispielhaft gezeigt, wie die Perfekte Rekonstruktion von Audiodaten funktioniert (Beispiel 4.10) und wie eine Audioaufnahme von störendem Rauschen befreit werden kann (Beispiel 4.11). Im Anschluss wird, in Unterkapitel 4.3, gezeigt, wie bestimmte Zerlegungs- und Rekonstruktionsfolgen konstruiert werden können, die für die Algorithmen 4.5 und 4.7 benötigt werden. Im Appendix befinden sich die Tabellen 6.2 und 6.3, in denen diese Folgen konkret angegeben sind.

Die Arbeit stützt sich vor allem auf moderne Waveletliteratur von Charles Chui und Johan De Villiers und ihr Werk „Wavelet Subdivision Methods“.

2 Notationen und Definitionen

Definition 2.1 (Reelle Folgen mit Indexbereich $\mathbb{Z}$)**.**
Die Gesamtheit der reellwertigen Folgen mit Indexbereich $\mathbb{Z}$ wird mit $\ell(\mathbb{Z})$ bezeichnet.

Beispiel 2.2 (Reelle Folge mit Indexbereich $\mathbb{Z}$)**.**
Die Folge $c = \{c_k\}$, mit $c_k \in \mathbb{R}$, $k \in \mathbb{Z}$, also z.B.

$$\{c_k\} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\},$$

ist eine reellwertige Folge, mit Indexbereich $\mathbb{Z}$, aus $\ell(\mathbb{Z})$.

Definition 2.3 (Träger einer Folge)**.**
Der Träger einer Folge $c = \{c_k\} \in \ell(\mathbb{Z})$ wird mit $supp(c)$ bezeichnet und ist folgendermaßen definiert:

$$supp(c) := \{\mu, \dots, \nu\}, \ falls\ c_k = 0 \quad \forall k \notin \{\mu, ..., \nu\}, \qquad \mu, \nu \in \mathbb{Z}.$$

Beispiel 2.4 (Träger einer Folge)**.**
Die Folge $c = \{c_k\} \in \ell(\mathbb{Z})$, mit

$$c_k := 0 \quad \forall k \notin \{-1, 0, 1\}$$

und

$$c_{-1} := 1, \ c_0 := 2, \ c_1 := 3,$$

hat den Träger $supp(c) = \{-1, 0, 1\}$.

Definition 2.5. *Eine Folge $c \in \ell(\mathbb{Z})$ heißt zentriert, falls $supp(c) = \{\mu, ..., \nu\}$, mit $\mu \leq -1, \nu \geq 1$.*

Die Folge c, aus Beispiel 2.4, ist zentriert.

Definition 2.6 (Endliche Folgen)**.**
Die Gesamtheit der endlichen Folgen aus $\ell(\mathbb{Z})$ wird mit $\ell_0(\mathbb{Z})$ bezeichnet und ist folgendermaßen definiert.

$$\ell_0(\mathbb{Z}) := \{c \in \ell(\mathbb{Z}) : |supp(c)| < \infty\}.$$

Notation 2.7 (Endliche Folgen)**.**
Eine Folge $c = \{c_k\} \in \ell_0(\mathbb{Z})$, mit $supp(c) = \{\mu, \dots, \nu\}$, wird folgendermaßen bezeichnet.

$$\{c_k\}_\mu^\nu = \{c_\mu, \dots, c_\nu\}, \quad \mu, \nu \in \mathbb{Z}.$$

Beispiel 2.8 (Endliche Folge)**.**
Die Folge c, aus Beispiel 2.4, ist eine endliche Folge aus $\ell_0(\mathbb{Z})$ und wird folgendermaßen bezeichnet.

$$\{c_k\}_{-1}^{1} = \{1, 2, 3\}.$$

Notation 2.9 (Komplexe Zahlen ohne Null)**.**
Die Komplexen Zahlen, ohne Null, werden folgendermaßen bezeichnet.

$$\mathbb{C}^\times := \mathbb{C}\backslash\{0\}.$$

Definition 2.10 (Laurentpolynom [2, Def.2.1])**.**
Der Ring der Laurentpolynome, über den komplexen Zahlen ohne Null, wird mit $\Lambda[\mathbb{C}^\times]$ bezeichnet und ist folgendermaßen definiert.

$$\Lambda[\mathbb{C}^\times] := \left\{ f(z) = \sum_{j=-n}^{m} f_j z^j \ : \ z \in \mathbb{C}^\times, f_j \in \mathbb{C}, \ n, m \in \mathbb{N}_0 \right\}.$$

Beispiel 2.11 (Laurentpolynom)**.**

$$P : \mathbb{C}^\times \to \mathbb{C}^\times, \ mit \ P(z) = z^{-1} + 2z^0 + 4z^1$$

ist ein Laurentpolynom aus $\Lambda[\mathbb{C}^\times]$.

Zu jeder endlichen Folge $c \in \ell_0(\mathbb{Z})$ kann eindeutig ein dazugehöriges Laurentpolynom gebildet werden, indem die Folgenglieder als Koeffizienten verwendet werden. Das entsprechende Laurentpolynom c^* ist dann wie folgt definiert.

Definition 2.12 (Laurentpolynome endlicher Folgen)**.**
Das Laurentpolynom $c^* \in \Lambda[\mathbb{C}^\times]$ einer endlichen Folge $c \in \ell_0(\mathbb{Z})$ ist definiert durch

$$c^*(z) := \sum_{k\in\mathbb{Z}} c_k z^k, \qquad z \in \mathbb{C}^\times.$$

In Kapitel 4 werden die modifizierten Laurentpolynome P^* und Q^* verwendet, die den Faktor $\frac{1}{2}$ enthalten und wie folgt definiert sind.

$$P^*(z) := \tfrac{1}{2}p^*(z) = \tfrac{1}{2}\sum_{k\in\mathbb{Z}} p_k(z), \quad p \in \ell_0(\mathbb{Z}), \tag{2.0.1}$$

$$Q^*(z) := \tfrac{1}{2}q^*(z) = \tfrac{1}{2}\sum_{k\in\mathbb{Z}} q_k(z), \quad q \in \ell_0(\mathbb{Z}). \tag{2.0.2}$$

Beispiel 2.13 (Laurentpolynom einer endlichen Folge)**.**
Sei $c = \{c_k\} = \{-5, 7, 4, -1\}_{-2}^{1} \in \ell_0(\mathbb{Z})$.

Das zu c gehörende Laurentpolynom $c^*(z) \in \Lambda[\mathbb{C}^\times]$ lautet

$$c^*(z) = -5z^{-2} + 7z^{-1} + 4 - z, \quad z \in \mathbb{C}^\times.$$

Definition 2.14 (Grad eines Laurentpolynoms)**.**
Der Grad eines Laurentpolynoms $c^*(z) \in \Lambda[\mathbb{C}^\times]$ *wird in dieser Arbeit immer als der kleinste Exponent der in* c^* *vorkommenden Monome angegeben (d.h.* $grad(c^*) = \mu$*, für* $supp(c) = \{\mu, \ldots, \nu\}$*).*

Notation 2.15 (Upsampling von Laurentpolynomen)**.**
Nachdem mit einem Laurentpolynom $c^*(z)$ *Upsampling durchgeführt wurde, notiert man es folgendermaßen:*

$$c^*_{up}(z).$$

Notation 2.16 (Downsampling von Laurentpolynomen)**.**
Nachdem mit einem Laurentpolynom $c^*(z)$ *Downsampling durchgeführt wurde, notiert man es folgendermaßen:*

$$c^*_{down}(z).$$

3 Grundlegende Operationen der Signalverarbeitung

3.1 Laurentpolynome

In diesem Kapitel wird auf die Matlab-Darstellung von Laurentpolynomen eingegangen. Laurentpolynome sind Polynome mit komplexen Koeffizienten, bei denen auch Monome z^k mit negativen Exponenten k zugelassen sind (siehe die Definitionen 2.10 bzw. 2.12).
Für die digitale Signalverarbeitung sind Laurentpolynome ein sehr wichtiges mathematisches Konzept. Viele Operationen zur Signalverarbeitung können als Laurentpolynomoperationen durchgeführt werden. Diese Operationen sind einfach zu berechnen und zu implementieren.

In Matlab gibt es verschiedene Möglichkeiten, um Laurentpolynome darzustellen. Eine davon ist es, eine sogenannte Struktur zu definieren, die einem Laurentpolynom entspricht. In folgender Implementierung wird eine Struktur definiert (in Matlab `struct` genannt), die den (endlichen) Koeffizientenvektor $[c_\mu, \ldots, c_\nu]$, $\mu, \nu \in \mathbb{Z}$, des Laurentpolynoms beinhält, als auch den Grad μ. Für das Laurentpolynom aus Beispiel 2.11 sieht die entsprechende Matlab-Implementierung folgendermaßen aus:

Implementierung in Matlab 3.1 (Laurentpolynome)**.**

Matlab Code

```
P = struct('koeff',[1 2 4],'grad',-1);
```

Die Koeffizienten `[1 2 4]` *werden in dem Vektor* `koeff` *gespeichert und der Grad* $\mu = -1$ *des entsprechenden Laurentpolynoms in einer weiteren Variable* `grad`*. Es ist wichtig zu wissen, dass in Matlab der erste Eintrag eines Vektors mit 1 indiziert ist. Außerdem gilt es zu beachten, dass, wenn man Operationen mit den Koeffizientenvektoren von zwei Laurentpolynomen durchführt, der Grad des Resultats entsprechend anzupassen ist (siehe zum Beispiel die Implementierung in Matlab 3.6). Auf die beiden Komponenten der Sruktur* `P` *(also den Koeffizientenvektor und den Grad des Laurentpolynoms) kann folgendermaßen zugegriffen werden:*

Matlab Code	*Ausgabe*	*Erklärung*
`P.koeff` `P.grad`	`[1 2 4]` `-1`	*Abfrage der Werte*
`P.koeff = [2 4];` `P.grad = 0;`		*Änderung der Werte*
`P`	`P = koeff: [2 4]` `grad: 0`	*Abfrage der Struktur P*

- *Das Zugreifen auf eine Komponente der Struktur wird durchgeführt, indem man* `Strukturname.Strukturkomponente` *eingibt.*
- *Das Zuweisen eines neuen Wertes kann mit dem Befehl* `Strukturname.Strukturkomponente = neuer Wert` *durchgeführt werden.*

3.2 Upsampling einer Folge

In diesem und im darauffolgenden Kapitel wird auf das Up- und Downsampling eingegangen. Das sind zwei essenzielle Operationen der Signalverarbeitung, die dazu dienen, die Abtastrate eines Signals zu erhöhen bzw. zu verringern. Es wird beschrieben, wie das Up- und Downsampling von Folgen und Laurentpolynomen definiert ist und mit Beispielen illustriert. Außerdem wird auf die Implementierung von Up- und Downsampling in Matlab eingegangen.

Das Upsampling $\uparrow$ einer Folge $\{c_k\} \in \ell(\mathbb{Z})$ ist das Einfügen von Nullen zwischen allen Folgengliedern.

Definition 3.2 (Upsampling einer Folge [3, S.17 (1.4.1)])**.**
Sei $c = \{c_k\} \in \ell(\mathbb{Z})$.
Das Upsampling ist eine Abbildung

$$\uparrow\colon \ell(\mathbb{Z}) \to \ell(\mathbb{Z}),\ \uparrow\{c_k\} = \{\tilde{c_j}\}\ mit\ \tilde{c}_j := \begin{cases} c_k,\ für\ j = 2k, \\ 0,\ für\ j = 2k+1. \end{cases}$$

In Definition 3.2 werden die Nullen als ungerade-indizierte Folgegliedern gesetzt. Analog dazu kann man das Upsampling auch durchführen, indem man die Nullen als gerade-indizierte Folgenglieder setzt.

Beispiel 3.3 (Upsampling einer Folge)**.**
Sei $c = \{c_k\} \in \ell_0(\mathbb{Z})$, *mit:*

$$\{c_k\}_{-1}^{1} := \left\{\tfrac{1}{2}, 1, \tfrac{1}{2}\right\}.$$

Dann ist

$$\uparrow \{c_k\}_{-1}^{1} = \uparrow \left\{\tfrac{1}{2}, 1, \tfrac{1}{2}\right\}_{-1}^{1} = \left\{\tfrac{1}{2}, 0, 1, 0, \tfrac{1}{2}\right\}_{-2}^{2}.$$

Das Upsampling einer Folge kann nicht nur, wie in Definion 3.2, bewerkstelligt werden, sondern auch als die folgende Polynomoperation.

Definition 3.4 (Upsampling als Polynomoperation)**.**
Sei $c \in \ell_0(\mathbb{Z})$.
Man betrachte das entsprechende Laurentpolynom $c^* \in \Lambda[\mathbb{C}^\times]$.
Das Upsampling von c wird äquivalent folgendermaßen durchgeführt:

$$c^*_{up}(z) := c^*(z^2).$$

Beachte, dass $\uparrow c$ die Koeffizientenfolge von $c^*_{up}(z)$ ist.

Beispiel 3.5 (Upsampling als Polynomoperation)**.**
Betrachte die Koeffizientenfolge $c \in \ell_0(\mathbb{Z})$ aus Beispiel 3.3. Das entsprechende Laurentpolynom ist dann:

$$c^*(z) := \tfrac{1}{2}z^{-1} + 1 + \tfrac{1}{2}z^{1}\ , \quad z \in \mathbb{C}^\times$$

und

$$c^*_{up}(z) := c^*(z^2) = \tfrac{1}{2}z^{-2} + 1 + \tfrac{1}{2}z^{2}, \quad z \in \mathbb{C}^\times.$$

Die Koeffizientenfolge von $c^*_{up}(z)$ ist somit $\left\{\tfrac{1}{2}\ 0\ 1\ 0\ \tfrac{1}{2}\right\}_{-2}^{2}$
und das Upsampling ist korrekt durchgeführt worden.
Man beachte, dass der Grad des Polynoms $c^*(z)$, durch das Upsampling, verdoppelt worden ist.

Implementierung in Matlab 3.6 (Upsampling)**.**
In Matlab kann das Upsampling mit dem Befehl `dyadup` *(Dyadic Upsampling), aus der Wavelet Toolbox, bewerkstelligt werden. Das Beispiel 3.5 in Matlab umgesetzt:*

Matlab Code	*Ausgabe*
`P = struct('koeff',[1/2 1 1/2],'grad',-1)`	`P = koeff: [0.5 1 0.5]` `grad: -1`
`P.koeff = dyadup(P.koeff,0);` `P.grad = P.grad*2;`	
`P`	`P = koeff: [0.5 0 1 0 0.5]` `grad: -2`

Folgende Punkte sind dabei zu beachten:

- *dem Befehl* `dyadup` *kann als zweiter Parameter eine Zahl übergeben werden, die beschreibt, ob die Nullen in* $\uparrow c$ *gerade/ungerade indiziert werden. Ist diese Zahl gerade (in unserem Beispiel gleich 0), so werden die Nullen als gerade indizierte Einträge eingefügt. Ist die Zahl ungerade, werden die Nullen als ungerade indizierte Einträge eingefügt.*

- *der Grad des Laurentpolynoms* `P.grad` *(die zweite Komponente der Matlabstruktur) muss beim Upsampling verdoppelt werden (folgt aus der Definition 3.4).*

3.3 Downsampling einer Folge

Das Downsampling $\downarrow$ einer Folge $c = \{c_k\} \in \ell(\mathbb{Z})$ ist das Verwerfen jedes ungerade indizierten Folgengliedes.

Definition 3.7 (Downsampling)**.** *Sei* $c = \{c_k\} \in \ell(\mathbb{Z})$*. Das Downsampling ist eine Abbildung*

$$\downarrow: \ell(\mathbb{Z}) \to \ell(\mathbb{Z}), \quad \downarrow \{c_k\} := \{c_{2k}\}.$$

Analog dazu könnte man Downsampling auch als das Verwerfen aller gerade indizierten Folgenglieder durchführen.

Beispiel 3.8 (Downsampling)**.**
Betrachte die Folge $c = \{c_k\} \in \ell_0(\mathbb{Z})$*, mit*

$$\{c_k\}_{-3}^{3} := \{1, 2, 3, 4, 5, 6, 7\}.$$

Aus der Definition 3.7 folgt dann:

$$\downarrow \{c_k\}_{-3}^{3} = \downarrow \{1, 2, 3, 4, 5, 6, 7\}_{-3}^{3} = \{2, 4, 6\}_{-1}^{1}.$$

Analog zum Upsampling als Polynomoperation, gibt es auch für das Downsampling eine analoge Definition.

Definition 3.9 (Downsampling als Polynomoperation)**.**
Seien $\mu, \nu \in \mathbb{Z}, c = \{c_k\}_{\mu}^{\nu} \in \ell_0(\mathbb{Z})$ *und das dazugehörige Laurentpolynom* $c^*(z) \in \Lambda[\mathbb{C}^\times]$.
Dann ist $\downarrow c$ *die Koeffizientenfolge des Polynoms*

$$c^*_{down}(z) := \tilde{c}^*(z^{\frac{1}{2}}),$$

mit

$$\tilde{c}^*(z) := \tfrac{1}{2}\left[c^*(z) + c^*(-z)\right] \quad z \in \mathbb{C}^\times.$$

Der Grad des Polynoms $c^*_{down}(z)$ entspricht dann $\lceil \frac{\mu}{2} \rceil$.

Bemerkung 3.10 (Downsampling als Polynomoperation)**.**
Beim Downsampling durch Polynomoperationen werden, im ersten Schritt, die Monome ungeraden Grades verworfen und im Anschluss die Exponenten der verbliebenen Monome geraden Grades, halbiert.
D.h.: c^*_{down} in Definition 3.9 ist wieder ein Laurentpolynom $\in \Lambda[\mathbb{C}^\times]$, weil $\tilde{c}^*(z)$ ein Laurentpolynom mit Monomen $z^{2j},\ j \in \mathbb{Z}$, ist. Dies ist leicht zu erkennen, indem man die Definition 2.12 einsetzt

$$\tilde{c^*}(z) = \tfrac{1}{2}\left[c^*(z) + c^*(-z)\right] = \frac{1}{2}\left[\sum_{j\in\mathbb{Z}} c_j z^j + \sum_{j\in\mathbb{Z}} c_j (-z)^j\right],$$

die Umformung

$$\frac{1}{2}\left[\sum_{j\in\mathbb{Z}} c_j z^j + \sum_{j\in\mathbb{Z}} c_j (-z)^j\right] = \frac{1}{2}\sum_{j\in\mathbb{Z}} c_j \left[z^j + (-z)^j\right]$$

durchführt und die Summen folgendermaßend aufspaltet

$$\begin{aligned}&\frac{1}{2}\sum_{j\in\mathbb{Z}} c_j \left[z^j + (-z)^j\right]\\ &= \frac{1}{2}\sum_{j\in\mathbb{Z}} c_j \left[\sum_{j\in\mathbb{Z}} z^{2j} + \sum_{j\in\mathbb{Z}} z^{2j+1} + \sum_{j\in\mathbb{Z}} (-z)^{2j} + \sum_{j\in\mathbb{Z}} (-z)^{2j+1}\right], \quad z \in \mathbb{C}^\times.\end{aligned}$$

Weil die Potenz einer negativen Zahl positiv ist, wenn der Exponent gerade ist und die Potenz negativ ist, wenn der Exponent ungerade ist, gilt

$$\begin{aligned} c^*(z) &= \frac{1}{2}\sum_{j\in\mathbb{Z}} c_j \left[\sum_{j\in\mathbb{Z}} z^{2j} + \sum_{j\in\mathbb{Z}} z^{2j+1} + \sum_{j\in\mathbb{Z}} z^{2j} - \sum_{j\in\mathbb{Z}} z^{2j+1}\right]\\ &= \frac{1}{2}\sum_{j\in\mathbb{Z}} c_j \left[2\sum_{j\in\mathbb{Z}} z^{2j}\right]\\ &= \sum_{j\in\mathbb{Z}} c_j z^{2j}, \quad z \in \mathbb{C}^\times.\end{aligned}$$

Beispiel 3.11 (Downsampling als Polynomoperation)**.**
Sei $c = \{c_k\}_\mu^\nu \in \ell_0(\mathbb{Z})$ die Koeffizientenfolge aus dem Beispiel 3.8. Das dazugehörige Laurentpolynom $c^*(z) \in \Lambda[\mathbb{C}^\times]$ ist dann

$$c^*(z) = z^{-3} + 2z^{-2} + 3z^{-1} + 4 + 5z + 6z^2 + 7z^3\ , \quad z \in \mathbb{C}^\times.$$

Damit ist

$$c^*(-z) = -z^{-3} + 2z^{-2} - 3z^{-1} + 4 - 5z + 6z^2 - 7z^3, \quad z \in \mathbb{C}^\times.$$

Laut Definition 3.9 ist dann

$$\tilde{c}^*(z) = \tfrac{1}{2}[c^*(z) + c^*(-z)] = 2z^{-2} + 4 + 6z^2$$

und

$$c^*_{down}(z) = \tilde{c^*}(z^{\frac{1}{2}}) = 2z^{-1} + 4 + 6z^1, \quad z \in \mathbb{C}^\times.$$

Man beachte, dass sich der Grad des Polynoms $c^*(z)$, durch das Downsampling, geändert hat und nun $\lceil \frac{\mu}{2} \rceil = \lceil \frac{-3}{2} \rceil = -1$ beträgt.

Implementierung in Matlab 3.12 (Downsampling)**.**
Das Downsampling kann in Matlab mit dem Befehl `dyaddown` *(Dyadic Downsampling), aus der Wavelet Toolbox, umgesetzt werden. Als ersten Parameter übergibt man dem Befehl den (Koeffizienten-)Vektor, mit dem Downsampling durchgeführt werden soll. Wie beim Upsampling-Befehl, kann auch beim Downsampling-Befehl als zweiter Parameter eine Zahl übergeben werden. Je nachdem, ob diese gerade/ungerade ist, liefert das Downsampling einen Vektor, der die gerade/ungerade indizierten Einträge des ursprünglichen (Koeffizienten-)Vektors enthält.*
Hier das Beispiel 3.11, implementiert in Matlab:

Matlab Code	*Ausgabe*

`koeff = [1 2 3 4 5 6 7];`
`C =`
`struct('koeff',koeff,'grad',-3)`
	`C =`
	`koeff: [1 2 3 4 5 6 7]`
	`grad: -3`

`koeff_down = dyaddown(C.koeff,C.grad+1);` `grad_down = ceil(C.grad/2);`	

`C_down =`
`struct('koeff',koeff_down,'grad',grad_down)`
	`C_down =`
	`koeff: [2 4 6]`
	`grad: -1`

Beim Downsampling sollen (laut Definition 3.9 bzw. Definition 3.7) alle Monome ungeraden Grades verworfen werden. Bei der Implementierung in Matlab muss daher beachtet werden, dass der zweite Parameter von `dyaddown`*, je nach Polynomgrad, korrekt gewählt wird:*

- *Ist der Grad des Polynoms ungerade, müssen mit* `dyaddown`*, in Matlab, alle ungerade indizierten Einträge des Koeffizientenvektors gelöscht werden. Daher wählt man als zweiten Parameter eine gerade Zahl.*
- *Für Polynome mit geradem Grad muss als zweiter Parameter eine ungerade Zahl gewählt werden, damit jeder gerade indizierten Einträge des Koeffizientenvektors gelöscht wird.*

Das bedeutet, dass als zweiter Parameter des `dyaddown`*-Befehls* `.grad+1` *gewählt werden soll, um das Downsampling korrekt durchzuführen.*
Das Anpassen des Polynomgrades kann in Matlab mit Hilfe des Befehls `ceil` *umgesetzt werden.* `ceil` *entspricht der üblichen Gaußklammer* $\lceil\ \rceil$*.*

3.4 Faltung von Folgen

In diesem Kapitel wird die diskrete Faltung von Folgen definiert und darauf eingegangen, dass sie sich wie die Multiplikation der entsprechenden Laurentpolynome verhält.

Die Faltung kann bei der Signalverarbeitung eingesetzt werden, um gewichtete Mittelwerte von Signalen zu bilden und Signalkurven zu glätten. In der digitalen Audiosignalverarbeitung können Faltungsoperationen eingesetzt werden, um Audioaufnahmen von Störgeräuschen zu befreien (siehe Unterkapitel 4.2.4).

Definition 3.13 (Diskrete Faltung zweier Folgen [2, Def.1.7])**.**
Zu $a, b \in \ell_0(\mathbb{Z})$ ist die Faltung $*$ folgendermaßen definiert:

$$* : \ell_0(\mathbb{Z}) \times \ell_0(\mathbb{Z}) \to \ell_0(\mathbb{Z}), \quad (a * b)_k := \sum_{j\in\mathbb{Z}} a_j b_{k-j}, \quad k \in \mathbb{Z}.$$

Beispiel 3.14 (Diskrete Faltung zweier Folgen)**.**
Gegeben: zwei Folgen $a = \{a_k\}, b = \{b_k\} \in \ell_0(\mathbb{Z})$, mit

$$\{a_k\}_{-2}^{1} := \{1, 2, 3, 4\} \quad \textit{und} \quad \{b_k\}_{-1}^{2} := \{4, 3, 2, 1\}.$$

Berechne: $(a * b)_k$ laut Definition 3.13:

$$\begin{aligned}
(a*b)_{-3} = \sum_{j\in\mathbb{Z}} a_j b_{-3-j} &= \cdots + 0 + a_{-2}b_{-1} + 0 + \cdots = 4.\\
(a*b)_{-2} = \sum_{j\in\mathbb{Z}} a_j b_{-2-j} &= \cdots + 0 + a_{-2}b_0 + a_{-1}b_{-1} + 0 + \cdots = 3 + 8 = 11.\\
(a*b)_{-1} = \sum_{j\in\mathbb{Z}} a_j b_{-1-j} &= \cdots + 0 + a_{-2}b_1 + a_{-1}b_0 + a_0 b_{-1} + 0 + \ldots\\
&= 2 + 6 + 12 = 20.\\
(a*b)_{0} = \sum_{j\in\mathbb{Z}} a_j b_{-j} &= \cdots + 0 + a_{-2}b_2 + a_{-1}b_1 + a_0 b_0 + a_1 b_{-1} + 0 + \ldots\\
&= 1 + 4 + 9 + 16 = 30.\\
(a*b)_{1} = \sum_{j\in\mathbb{Z}} a_j b_{1-j} &= \cdots + 0 + a_{-1}b_2 + a_0 b_1 + a_1 b_0 + 0 + \ldots\\
&= 2 + 6 + 12 = 20.\\
(a*b)_{2} = \sum_{j\in\mathbb{Z}} a_j b_{2-j} &= \cdots + 0 + a_0 b_2 + a_1 b_1 + 0 + \cdots = 3 + 8 = 11.\\
(a*b)_{3} = \sum_{j\in\mathbb{Z}} a_j b_{3-j} &= \cdots + 0 + a_1 b_2 + 0 + \cdots = 4.
\end{aligned}$$

$$\begin{aligned}
&(a*b)_k = 0 \quad \forall k \notin \{-3, -2, -1, 0, 1, 2, 3\}\,.\\
&\Rightarrow (a*b)_k = \{4, 11, 20, 30, 20, 11, 4\}_{-3}^{3}.
\end{aligned}$$

Satz 3.15 (Diskrete Faltung als Polynommultiplikation [2, S.31])**.**
Die Diskrete Faltung $*$ zweier Folgen $a, b \in \ell_0(\mathbb{Z})$ verhält sich wie die (Laurent-)Polynommultiplikation von a^* mit b^*:

$$(a * b)^*(z) = a^*(z)b^*(z), \quad z \in \mathbb{C}^\times.$$

Beweis:
Sei $z \in \mathbb{C}^\times$. Laut der Definition 2.10 ist

$$(a * b)^*(z) = \sum_{k\in\mathbb{Z}} (a * b)_k z^k.$$

Setzt man nun die Definition 3.13 ein, ergibt sich

$$\sum_{k\in\mathbb{Z}} (a * b)_k z^k = \sum_{k\in\mathbb{Z}} \left(\sum_{j\in\mathbb{Z}} a_j b_{k-j} \right) z^k.$$

Mit den Rechenregeln für Potenzen folgt

$$\sum_{k\in\mathbb{Z}} \left(\sum_{j\in\mathbb{Z}} a_j b_{k-j} \right) z^k = \sum_{k\in\mathbb{Z}} \left(\sum_{j\in\mathbb{Z}} a_j b_{k-j} \right) z^j z^{k-j};$$

und nach der Umformung

$$\sum_{k\in\mathbb{Z}} \left(\sum_{j\in\mathbb{Z}} a_j b_{k-j} \right) z^j z^{k-j} = \sum_{j\in\mathbb{Z}} a_j z^j \sum_{k\in\mathbb{Z}} b_{k-j} z^{k-j}$$

wird folgende Indexsubstitution $k' = k - j$ durchgeführt.

$$\sum_{j\in\mathbb{Z}} a_j z^j \sum_{k\in\mathbb{Z}} b_{k-j} z^{k-j} = \left(\sum_{j\in\mathbb{Z}} a_j z^j \right) \left(\sum_{k'\in\mathbb{Z}} b_{k'} z^{k'} \right).$$

Setzt man wieder die Definition 2.10 ein, ergibt sich

$$\left(\sum_{j\in\mathbb{Z}} a_j z^j \right) \left(\sum_{k'\in\mathbb{Z}} b_{k'} z^{k'} \right) = a^*(z)b^*(z).$$

□

Beispiel 3.16 (Diskrete Faltung als Polynommultiplikation)**.**
Gegeben sind die beiden Folgen a und b aus Beispiel 3.14.
Betrachte

$$a^*(z) := z^{-2} + 2z^{-1} + 3 + 4z$$

und

$$b^*(z) := 4z^{-1} + 3 + 2z + z^2, \quad z \in \mathbb{C}^\times.$$

Berechne

$$a^*(z)b^*(z) = 4z^{-3} + 11z^{-2} + 20z^{-1} + 30 + 20z + 11z^2 + 4z^3, \quad z \in \mathbb{C}^\times.$$

Nach Satz 3.15 gilt $(a * b)^*(z) = a^*(z)b^*(z)$, was die Ausrechnung von $a^*(z)b^*(z)$ auch bestätigt (vergleiche das Ergebnis mit Beispiel 3.14).

Implementierung in Matlab 3.17 (Faltung)**.**
Die diskrete Faltung kann in Matlab mit dem Befehl `conv(a,b)`*, aus der Wavelet-Toolbox, umgesetzt werden. Die beiden Parameter* `a` *und* `b` *sind Vektoren und* `conv(a,b)` *liefert die Faltung der beiden Vektoren. Hier das Beispiel 3.16, in Matlab implementiert:*

Matlab Code	*Ausgabe*

`A =`
`struct('koeff',[1 2 3 4],'grad',-2)`
 	
`B =`
`struct('koeff',[4 3 2 1],'grad',-1)`
	`A = koeff: [1 2 3 4]`
	`grad: -2`
	`B = koeff: [4 3 2 1]`
	`grad: -1`

`AB_koeff = conv(A.koeff,B.koeff);` `AB_grad = A.grad + B.grad;`	

`C =`
`struct('koeff',AB_koeff,'grad',AB_grad)`
	`C =`
	`koeff: [4 11 20 30 20 11 4]`
	`grad: -3`

Es gilt zu beachten, dass, laut den Standardrechenregeln (für Polynome) aus der Algebra, der Grad von `C` *gleich die Summe der Grade von* `A` *und* `B` *ist.*

4 Zerlegung und Rekonstruktion von Audiodaten

In diesem Kapitel werden die soeben im Kapitel 3 vorgestellten grundlegenden Operationen der Signalverarbeitung angewendet und ihr Nutzen verdeutlicht. Das Hauptaugenmerk dieses Abschnitts ist auf die Algorithmen 4.5 und 4.7 gerichtet, mit deren Hilfe Audiodaten zerlegt und wieder rekonstruiert werden können.
Zu Beginn wird, durch einführende Beispiele, schrittweise an die Algorithmen herangeführt.
Im Anschluss wird erklärt, was ein diskretes Signal ist und welcher Zusammenhang zwischen Audiodaten und Audiosignalen besteht. Schließlich werden die Algorithmen genau erläutert und ihre Implementierung in Matlab vorgestellt. Danach wird, in Unterkapitel 4.2.3, begründet, weshalb die Perfekte Rekonstruktion eines diskreten Audiosignals funktioniert.

Im sehr wichtigen und interessanten Unterkapitel 4.2.4 werden die Operationen und Algorithmen, zur Verarbeitung von Audiosignalen, auf reale Audiodaten angewendet. Dieser Abschnitt ist besonders anschaulich und hat einen hohen Praxisbezug.

Im Unterkapitel 4.3 werden die Zerlegungs- und Rekonstruktionsfolgen, die für die Algorithmen 4.5 und 4.7 notwendig sind, hergeleitet.
Folgende Teile des Kapitels 4 wurden gemeinsam mit dem Kollegen Lukas Weichhart ausgearbeitet und in überarbeiteter Version übernommen: der Algorithmus 4.12, das Beispiel 4.13 und der Satz 4.15.

4.1 Einführende Beispiele

Das folgende Beispiel ist eine Illustration für die Notwendigkeit des Feldes `.grad` bei der in dieser Arbeit verwendeten Laurentpolynom-Struktur (siehe die „Implementierung in Matlab 3.1“). In Beispiel 4.1 wird klar, weshalb es nicht genügt, nur mit den Koeffizienten von Laurentpolynomen zu arbeiten. Es werden zwei Schritte des Algorithmus 4.5 mit dem Datenvektor `c0 = [1 2 3]` durchgeführt, wodurch der Vektor `c0` zerlegt wird. Anschließend wird versucht, mit dem Algorithmus 4.7, die Perfekte Rekonstruktion von `c0` zu bewirken. Allerdings ohne Laurentpolynomstrukturen zu verwenden, sondern nur (Koeffizienten-)Vektoren. Deshalb kommt es im letzten Schritt zur Fehlerbildung. Im Anschluss wird erklärt, wie durch die Verwendung von Strukturen dieser Fehler behoben werden kann. Es ist an dieser Stelle nicht notwendig zu wissen, wie die Algorithmen genau durchgeführt werden (dies wird später, im Unterkapitel 4.2, erklärt).

Beispiel 4.1 (Matlab Beispiel ohne Berücksichtigung der Polynomgrade)**.**

Zuerst werden die, für die Algorithmen 4.5 und 4.7 notwendigen, Vektoren `c0,a,b,p,q` *definiert:*

Matlab Code

```
c0 = [1 2 3];

a = [1];
b = [1/4 -1/2 1/4];
p = [1/2 1 1/2];
q = [-2];
```

Anschließend wird der Vektor `c`*, in zwei Schritten, mit dem Algorithmus 4.5 zerlegt:*

Matlab Code	*Ausgabe*
`% Zerlegen`	
`c1 = dyaddown(conv(c0,a),1)`	`c1 = [1 3]`
`c2 = dyaddown(conv(c1,a),1)`	`c2 = [1]`
`d1 = dyaddown(conv(c0,b),1)`	`d1 = [0.25 0 0.75]`
`d2 = dyaddown(conv(c1,b),1)`	`d2 = [0.25 -1.25]`

Wenn man nun versucht, den Algorithmus 4.7 anzuwenden, um `c0` *zu rekonstruieren, erhält man, von Matlab, im letzten Schritt, einen Error als Rückmeldung:*

Matlab Code	*Ausgabe*

`% Rekonstruieren`
 	
`c2_up = dyadup(c2,0)`
`c2_up_mit_p = conv(c2_up,p)`
 	
`d2_up = dyadup(d2,0)`
`d2_up_mit_q = conv(d2_up,q)`
 	
`c1_rek =`
`c2_up_mit_p + d2_up_mit_q`
 	
`c1_rek_up = dyadup(c1_rek,0)`
`c1_rek_up_mit_p =`
`conv(c1_rek_up,p)`
 	
`d1_up = dyadup(d1,0)`
`d1_up_mit_q = conv(d1_up,q)`
	`c2_up = [1]`
	`c2_up_mit_p = [0.5 1 0.5]`
	
	`d2_up = [0.25 0 -1.25]`
	`d2_up_mit_q = [-0.5 0 2.5]`
	
	`c1_rek = [0 1 3]`
	
	`c1_rek_up = [0 0 1 0 3]`
	`c1_rek_up_mit_p =`
	`[0 0 0.5 1 2 3 1.5]`
	
	`d1_up = [0.25 0 0 0 0.75]`
	`d1_up_mit_q =`
	`[-0.5 0 0 0 -1.5]`

`c0_rek =` `c1_rek_up_mit_p + d1_up_mit_q`	`Error:` `Matrix dimensions must agree.`

Die Vektoren `c1_rek_up_mit_p = [0 0 0.5 1 2 3 1.5]` *und* `d1_up_mit_q = [-0.5 0 0 0 -1.5]` *können in Matlab nicht addiert werden, weil sie unterschiedlich viele Einträge haben. Das Programm kann Vektoren unterschiedlicher Länge nicht addieren, weil die Einträge der beiden Vektoren dann nicht komponentenweise addiert werden können. Um den Fehler zu beheben, müssen die Vektoren derart auf dieselbe Länge gebracht werden, dass dann die „richtigen“ Einträge zusammenaddiert werden.*

Wie geht das? Indem man Laurentpolynomstrukturen verwendet, wie in der „Implementierung in Matlab 3.1“ beschrieben. Dann existiert nämlich zu jedem Koeffizienvektor `.koeff` *ein zugehöriges Feld* `.grad`*. Mit dieser Zusatzinformation, über den Grad des Laurentpolynoms, können die Vektoren geschickt auf dieselbe Länge gebracht werden, um sie anschließend korrekt addieren zu können. Dafür muss mindestens einer der Vektoren* `.koeff` *passend mit Nullen aufgefüllt werden und der Grad* `.grad` *entsprechend angepasst werden. Zum besseren Verständnis wird diese Methode auf das eben gezeigte Beispiel angewendet.*

Definiere die Laurentpolynomstrukturen `C1` *und* `D1` *zu den beiden Koeffizientenvektoren*
`c1_rek_up_mit_p = [0 0 0.5 1 2 3 1.5]` *und*
`d1_up_mit_q = [-0.5 0 0 0 -1.5]`*:*

Matlab Code	*Ausgabe*

`C1 =`
`struct('koeff',c1_rek_up_mit_p,'grad',-3)`
 	
`D1 =`
`struct('koeff',d1_up_mit_q,'grad',-1)`
	`C1 =`
	`koeff: [0 0 0.5 1 2 3 1.5]`
	`grad: -3`
	
	`D1 =`
	`koeff: [-0.5 0 0 0 -1.5]`
	`grad: -1`

Fülle nun den Vektor `D1.koeff` *links mit zwei Nullen auf und ändere den Grad* `D1.grad` *auf* `-3`*:*

Matlab Code	*Ausgabe*

`D1.koeff = [0 0 D1.koeff];`
`D1.grad = -3`
	`D1 =`
	`koeff: [0 0 -0.5 0 0 0 -1.5]`
	`grad: -3`

Die Struktur `D1` *beschreibt noch immer dasselbe Laurentpolynom. Allerdings hat nun der Vektor* `D1.koeff` *dieselbe Länge wie* `C1.koeff` *und sie können addiert werden:*

Matlab Code	*Ausgabe*

`c0_rek = D1.koeff + C1.koeff;`
`C0_rek =`
`struct('koeff',c0_rek,'grad',-3)`
	`C0_rek = koeff: [0 0 0 1 2 3 0]`
	`grad: -3`

Bei der Berechnung von `c0_rek` *tritt nun kein* `Error` *mehr auf. Das Ergebnis* `c0_rek = [0 0 0 1 2 3 0]` *stimmt mit dem ursprünglichen Vektor* `c = [1 2 3]`*, bis auf zusätzliche Nullen, überein. Die überflüssigen Nullen sind allerdings nicht störend. Wenn man bei dem Beispiel von Anfang an Laurentpolynomstrukturen verwendet und die Grade* `.grad`*, in jedem Schritt, anpasst, sieht man, dass die ursprüngliche Struktur* `C0 = struct('koeff',[1 2 3],'grad',0)` *und* `C0_rek` *dasselbe Laurentpolynom* $1 + 2z + 3z^2 \in \Lambda[\mathbb{C}^\times]$ *darstellen.*

Im nachfolgenden Beispiel 4.2 wird das eben gerechnete Beispiel 4.1 noch einmal, unter Verwendung von Strukturen, implementiert.

Beispiel 4.2 (Matlab Beispiel mit Verwendung von Laurentpolynomstrukturen)**.**

Zuerst werden die benötigten Laurentpolynom-Strukturen wie folgt definiert.

Matlab Code	*Ausgabe*

`c0 = [1 2 3];  C0_grad=0;`
 	
`a = [1];  A_grad=0;`
`b = [1/4 -1/2 1/4];  B_grad=0;`
`p = [1/2 1 1/2];  P_grad=-1;`
`q = [-2];  Q_grad=-1;`
 	
`C0= struct('koeff',c0,'grad',C0_grad)`
`A = struct('koeff',a,'grad',A_grad);`
`B = struct('koeff',b,'grad',B_grad);`
`P = struct('koeff',p,'grad',P_grad);`
`Q = struct('koeff',q,'grad',Q_grad);`
	`C0 = koeff: [1 2 3]`
	`grad: 0`

Für dieses einführende Beispiel ist es noch nicht wichtig zu verstehen, weshalb die Grade `.grad` *der Laurentpolynom-Strukturen* `C0,A,B,P,Q` *genau so gewählt werden. Darauf wird im späteren Verlauf der Arbeit eingegangen.*

Im Folgenden werden, analog zu Beispiel 4.1, die Zerlegungs- und Rekonstruktionsalgorithmen angewendet. Beachte, wie die Grade entsprechend angepasst werden.

Matlab Code	*Ausgabe*

`% Zerlegen`
 	
`C0_mit_A=conv(C0.koeff,A.koeff);`
 	
`c1 =`
`dyaddown(C0_mit_A,C0.grad+A.grad+1);`
`C1_grad=ceil((C0.grad+A.grad)/2);`
 	
`C1 =`
`struct('koeff',c1,'grad',C1_grad)`
 	
`C1_mit_A=conv(C1.koeff,A.koeff);`
 	
`c2 =`
`dyaddown(C1_mit_A,C1.grad+A.grad+1);`
`C2_grad=ceil((C1.grad+A.grad)/2);`
 	
`C2 =`
`struct('koeff',c2,'grad',C2_grad)`
 	
`C0_mit_B=conv(C0.koeff,B.koeff)`
 	
`d1 =`
`dyaddown(C0_mit_B,C0.grad+B.grad+1);`
`D1_grad=ceil((C0.grad+B.grad)/2)`
 	
`D1 =`
`struct('koeff',d1,'grad',D1_grad)`
 	
`D1_mit_B=conv(D1.koeff,B.koeff)`
 	
`d2 =`
`dyaddown(C1_mit_B,C1.grad+B.grad+1);`
`D2_grad=ceil((C1.grad+B.grad)/2);`
 	
`D2 =`
`struct('koeff',d2,'grad',D2_grad)`
	`C1 = koeff: [1 3]`
	`grad: 0`
	
	`C2 = koeff: [1]`
	`grad: 0`
	
	`D1 = koeff: [0.25 0 0.75]`
	`grad: 0`
	
	`D2 = koeff: [0.25 -1.25]`
	`grad: 0`

Matlab Code	*Ausgabe*

`% Rekonstruieren`
 	
`c2_up = dyadup(C2.koeff,0);`
`d2_up = dyadup(D2.koeff,0);`
 	
`C1_koeff =`
`conv(c2_up,P.koeff)+conv(d2_up,Q.koeff);`
 	
`C1_grad =`
`C2.grad*2+P.grad;`
 	
`C1_rek =`
`struct('koeff',C1_koeff,'grad',C1_grad)`
	`C1_rek = koeff: [0 1 3]`
	`grad: -1`

`c1_up = dyadup(C1_rek.koeff,0);`
`d1_up = dyadup(D1.koeff,0);`
 	
`c1_up_mit_p=conv(c1_up,P.koeff)`
`d1_up_mit_q=conv(d1_up,Q.koeff)`
	`c1_up_mit_p =`
	`[0 0 0.5 1 2 3 1.5]`
	`d1_up_mit_q =`
	`[-0.5 0 0 0 -1.5]`

`% mit Nullen auffuellen`
`d1_up_mit_q = [0 0 d1_up_mit_q];`
 	
`C0_koeff =`
`c1_up_mit_p+d1_up_mit_q;`
 	
`C0_grad =`
`C1_rek.grad*2+P.grad;`
 	
`C0_rek =`
`struct('koeff',C0_koeff,'grad',C0_grad)`
	`C0_rek= koeff: [0 0 0 1 2 3 0]`
	`grad: -3`

Wie man sehen kann, stellen die beiden Laurentpolynomstrukturen `C0` *und* `C0_rek` *dasselbe Laurentpolynom* $1 + 2z + 3z^2 \in \Lambda[\mathbb{C}^\times]$ *dar und die Perfekte Rekonstruktion hat stattgefunden.*

Im nachfolgenden Beispiel werden die Zerlegungs- und Rekonstruktionsalgorithmen 4.5 und 4.7 ohne Matlab und mittels Laurentpolynomoperationen durgeführt. Es werden dieselben Eingabeparameter wie in den Beispielen 4.1 und 4.2 verwendet. Allerdings wird, damit das Beispiel kompakt ist, nur ein Schritt der Zerlegung und anschließenden Rekonstruktion durchgeführt.

Beispiel 4.3 (Zerlegung und Rekonstruktion durch Polynomoperationen)**.**

Sei $z \in \mathbb{C}^\times$ *und seien*

$$\left.\begin{aligned} c^*(z) &:= 1 + 2z + 3z^3 \\ a^*(z) &:= 1 \\ b^*(z) &:= \tfrac{1}{4} - \tfrac{1}{2}z + \tfrac{1}{4}z^2 \\ p^*(z) &:= \tfrac{1}{2}z^{-1} + 1 + \tfrac{1}{2}z \\ q^*(z) &:= -2z^{-1} \end{aligned}\right\}$$

die Eingabeparameter für die Algorithmen 4.5 und 4.7.

Ein Schritt der Zerlegung von $c^*(z)$***:***

$c^*_{zerl}(z)$ *berechnen, indem man* $c^*(z)$ *mit* $a^*(z)$ *multipliziert und anschließend Downsampling durchführt:*

$$c^*a^*(z) = 1 + 2z + 3z^3,$$

$$\widetilde{c^*a^*}(z) = \tfrac{1}{2}[c^*a^*(z) + c^*a^*(-z)] = 1 + 3z^2,$$

$$c^*_{zerl}(z) := c^*a^*_{down}(z) = \widetilde{c^*a^*}(z^{\frac{1}{2}}) = 1 + 3z.$$

$d^*(z)$ *berechnen, indem man* $c^*(z)$ *mit* $b^*(z)$ *multipliziert und anschließend Downsampling durchführt:*

$$c^*b^*(z) = \tfrac{1}{4} - z^3 + \tfrac{3}{4}z^4,$$

$$\widetilde{c^*b^*}(z) = \tfrac{1}{2}[c^*b^*(z) + c^*b^*(-z)] = \tfrac{1}{4} + \tfrac{3}{4}z^4,$$

$$d^*(z) := c^*b^*_{down}(z) = \widetilde{c^*b^*}(z^{\frac{1}{2}}) = \tfrac{1}{4} + \tfrac{3}{4}z^2.$$

Die Ausgabe des Zerlegungsalgorithmus sind die Laurentpolynome $c^*_{zerl}(z)$ *und* $d^*(z)$*.*

Nun wird der Rekonstruktionsalgorihtmus 4.7 angewendet, für die

Perfekte Rekonstruktion von $c^*(z)$***:***

Upsampling mit $c^*_{zerl}(z)$ *durchführen und anschließend mit* $p^*(z)$ *multiplizieren:*

$$c^*_{zerl_{up}}(z) = c^*_{zerl}(z^2) = 1 + 3z^2,$$

$$c^*_{zerl_{up}} p^*(z) = \tfrac{1}{2}z^{-1} + 1 + 2z + 3z^2 + \tfrac{3}{2}z^3$$

Upsampling mit $d^*(z)$ *durchführen und anschließend mit* $q^*(z)$ *multiplizieren:*

$$d^*_{up}(z) = d^*(z^2) = \tfrac{1}{4} + \tfrac{3}{4}z^4$$

$$d^*_{up} q^*(z) = -\tfrac{1}{2}z^{-1} - \tfrac{3}{2}z^3$$

Das ursprüngliche Polynom $c^*(z)$ *ist wiederhergestellt, nachdem* $c^*_{zerl_{up}} p^*(z)$ *und* $d^*_{up} q^*(z)$ *addiert werden. Tatsächlich gilt*

$$c^*(z) = c^*_{zerl_{up}} p^*(z) + d^*_{up} q^*(z) = 1 + 2z + 3z^3.$$

4.2 Algorithmus zur Zerlegung und Rekonstruktion von Signalen

Bevor man definieren kann, was Signalverarbeitung ist, wird das mathematische Modell eines Signals vorgestellt. Es gibt viele verschiedene Arten von Signalen. Man kann grob zwischen kontinuierlichen und diskreten Signalen unterscheiden. Kontinuierliche Signale sind analoge Signale. Ein analoges Signal ist beispielsweise eine Audioaufnahme, die auf einer Schallplatte gespeichert ist. In dieser Arbeit wird allerdings nur mit diskreten Signalen gearbeitet, die wie folgt definiert sind.

Definition 4.4 (Zeitdiskretes Signal)**.**
Ein zeitdiskretes Signal ist eine Abbildung $c : \mathbb{Z} \to \mathbb{R}$.

Ein zeitdiskretes Signal ist also eine diskrete Funktion (eine reelle Folge), die von einer Variable abhängt und reelle Werte annimmt. Die unabhängige Variable aus $\mathbb{Z}$ wird meist als Zeit bezeichnet.

Im Folgenden geht es um die Verarbeitung von digitalen Audiosignalen. Ein digitales Audiosignal ist ein zeitdiskretes Signal $c = \{c_k\}_\mu^\nu \in \ell_0(\mathbb{Z})$, das jedem Zeitpunkt aus $\{\mu, \ldots, \nu\}$ einen numerischen Wert aus $\mathbb{R}$ zuordnet. Signale werden bei der digitalen Signalverarbeitung durch Zahlen endlicher Genauigkeit dargestellt. Die Verarbeitung geschieht mit Hilfe eines Digitalrechners [1, S.2].

Ein (digitales) zeitdiskretes Signal erhält man durch das Abtasten eines (analogen) kontinuierlichen Signals. Das Abtasten kann als das Messen des kontinuierlichen Signals zu bestimmten Zeitpunkten verstanden werden.

Hier ein Beispiel, zum besseren Verständnis: Ein Lied, das auf einer Schallplatte gespeichert ist, soll in ein digitales Audiosignal umgewandelt werden. Auf der Schallplatte liegt ein (analoges) kontinuierliches Signal vor. Beim Abspielen der Schallplatte kann dieses nun, zu endlich vielen Zeitpunkten $\{0, 1, 2, \ldots, m\}$, gemessen und damit abgetastet werden. Durch das Abtasten erhält man ein zeitdiskretes (Audio-)Signal $c = \{c_k\}_0^m \in \ell_0(\mathbb{Z})$. Dieses Signal c beschreibt die gemessenen Werte des analogen Schallplattensignals, in Abhängigkeit von den Messzeitpunkten $\{0, 1, 2, \ldots, m\}$. Mit Hilfe eines Computers kann das digitale Signal c nun verarbeitet werden. Beispielsweise, um es zu analysieren oder unerwünschte Störgeräusche zu entfernen. Das Verarbeiten des Signals kann, unter anderem, mit Hilfe der in Kapitel 3 definierten Operationen geschehen.

Bisher wurden alle Operationen für Folgen aus $\ell(\mathbb{Z})$ bzw. $\ell_0(\mathbb{Z})$ definiert. Nachdem die Definition 2.1, für reellwertige Folgen mit Indexbereich $\mathbb{Z}$, deckungsgleich mit der Definition 4.4 für ein zeitdiskretes Signal ist, lassen sich die Operationen auf diskrete Signale übertragen.

Unter Signalverarbeitung versteht man die Zerlegung, Analyse und Speicherung von Signalen.

Was bedeutet es überhaupt, ein Audiosignal $c \in \ell_0(\mathbb{Z})$ zu zerlegen? Es geht darum, mit Hilfe bestimmter Zerlegungsfolgen $a, b \in \ell_0(\mathbb{Z})$, Frequenzen in c abzuspalten und zur späteren Verwendung abzuspeichern. Die Frequenzen, die man beim Zerlegen der Datei herausfiltert, werden in dieser Arbeit Details genannt.

Durch einen Schritt der Zerlegung erhält man einerseits eine Approximation $c^{(1)} \in \ell_0(\mathbb{Z})$ des ursprünglichen Signals c, als auch die abgespaltenen Details $d^{(1)} \in \ell_0(\mathbb{Z})$. Das Signal c kann n-mal zerlegt werden, um eine grobe Approximation $c^{(n)} \in \ell_0(\mathbb{Z})$, des ursprünglichen Audiosignals c, und die Details $d^{(i)},\ i = 1, \ldots, n$ zu erhalten.

Die Zerlegung ermöglicht es zum Beispiel, bestimmte Details aus $\left\{d^{(i)}\right\}_1^n$ zu verändern (oder zu verwerfen) und, mit Algorithmus 4.7, wieder an $c^{(n)}$ anzuheften. Dadurch erhält man eine modifizierte Version des Signals c. Wenn das Ursprungssignal c allerdings perfekt rekonstruiert werden soll, müssen die Details alle unverändert bewahrt werden. In dem Algorithmus, der nun vorgestellt wird, geht es darum, ein Audiosignal, in endlich vielen Schritten, zu zerlegen und die Details zu identifizieren. Lässt man die Details unverändert, können sie anschließend, mit dem Algorithmus 4.7, zur Perfekten Rekonstruktion des Signals c verwendet werden.

4.2.1 Zerlegung einer MP3-Audiodatei

Die Zerlegung einer MP3-Audiodatei geschieht mit Hilfe der Zerlegungsfolgen $a, b \in \ell_0(\mathbb{Z})$, die in Kapitel 4.3 konstruiert werden. Erst nachdem die MP3-Datei in ein digitales Audiosignal $c^{(0)} \in \ell_0(\mathbb{Z})$ umgewandelt wird, kann mit dem Algorithmus begonnen werden. Dann wird $n \in \mathbb{N}$, als Anzahl der Zerlegungsschritte, gewählt.

Das Signal $c^{(0)}$ wird mit den vorgegebenen Zerlegungsfolgen a und b gefaltet und anschließend wird der Schritt Downsampling mit $(c^{(0)} * a)$ und mit $(c^{(0)} * b)$ durchgeführt. Dadurch erhält man eine Approximation $c^{(1)} :=\downarrow (c^{(0)} * a) \in \ell_0(\mathbb{Z})$, an das ursprüngliche Signal $c^{(0)}$ und die herausgefilterten Details $d^{(1)} :=\downarrow (c^{(0)} * b) \in \ell_0(\mathbb{Z})$ [3, S.29]. Dieser Vorgang wird dann n-mal iterativ auf $c^{(i)},\ i = 1, \ldots, n$ angwendet.

Die Ausgabe des Algorithmus sind die Folgen $c^{(n)}$ und $d^{(i)},\ i = 1, \ldots, n$.

Algorithmus 4.5 (Algorithmus zur Zerlegung einer MP3-Datei)**.**

Voraussetzungen und Eingabe:

- *Eine MP3-Datei, als digitales Signal* $c^{(0)} = \{c_k\}_0^\nu \in \ell_0(\mathbb{Z}),\ \nu \in \mathbb{N}$,
- *Zerlegungsfolgen* $a \in \ell_0(\mathbb{Z})$ *und* $b \in \ell_0(\mathbb{Z})$*, welche die Identiäten aus dem Satz 4.15 (bzw. 4.20) erfüllen oder laut dem Algorithmus 4.12 konstruiert sind.*
- *Wähle* $n \in \mathbb{N}$*, als Anzahl der Zerlegungsschritte.*

Berechnung:
Berechne für i=1,...,n

- $d^{(i)} := \downarrow (c^{(i-1)} * b)$
- $c^{(i)} := \downarrow (c^{(i-1)} * a)$

Ausgabe:

- $d^{(i)} \in \ell_0(\mathbb{Z}),\ i = 1, \ldots, n$
- $c^{(n)} \in \ell_0(\mathbb{Z})$

Ende.

Der Algorithmus liefert n verschiedene Folgen $d^{(i)}$ $(i = 1, ..., n)$, in denen die Details des Ursprungssignals $c^{(0)}$ enthalten sind und eine Folge $c^{(n)}$, welche eine grobe Approximation an die Audiodatei darstellt.

Implementierung in Matlab 4.6 (Algorithmus zur Zerlegung von Signalen)**.**

Die Implementierung des Zerlegungsalgorithmus 4.5 kann mit einfachen Mitteln umgesetzt werden. Es wird mit den bereits bekannten Laurentpolynomstrukturen (siehe Implementierung in Matlab 3.1) gearbeitet. Zwei neue Befehle müssen allerdings erklärt werden:

- *der Befehl* `audioread`*, zum Einlesen von Audiodateien. Dieser wird folgendermaßen eingegeben:* `audioread('dateiname.dateiendung')` *und liefert einen Vektor* `[c,Fs]`*.* `c` *enthält die Audiodaten und stellt sozusagen das Eingabesignal* $c^{(0)}$ *für den Algorithmus dar.* `Fs` *ist die Abtastrate der Daten* `c`*.*

- *die* `for`*-Schleife, um den Algorithmus mit* `n` *Schritten anwenden zu können. Dieser Befehl wird folgendermaßen eingegeben:*

  ```
  for Index = Wertebereich
      Anweisungen
  end
  ```

  Der `Index` *durchläuft den* `Wertebereich` *und die* `Anweisungen` *werden dementsprechend oft ausgeführt. Im Algorithmus 4.5 wird der Index mit* `i` *bezeichnet und läuft von* 1 *bis* n*. In Matlab kann ein Indexbereich mit dem Operator „*`:`*“ angegeben werden. Die benötigte* `for`*-Schleife sieht daher folgendermaßend aus:*

  ```
  for i = 1:n
      Anweisungen
  end
  ```

Hier der gesamte Code der Implementierung des Algorithmus 4.5:

Eingabeparameter:

```
% Einlesen der MP3-Datei
 [c,Fs] = audioread('audiodatei.mp3');

 C = struct('koeff',c','grad',0);

% Definieren der Zerlegungsfolgen
 a = % gebe hier Zerlegungsfolge a als Vektor ein
 b = % gebe hier Zerlegungsfolge b als Vektor ein

 A = struct('koeff',a,'grad',A_grad);
 B = struct('koeff',b,'grad',B_grad);

% n waehlen
 n = % gebe hier Anzahl der Zerlegungsschritte ein
```

Zerlegung n-mal ausführen:

```
 for i=1:n
      D(i) = zerl_filtern(C,B);
      C = zerl_filtern(C,A);
 end
```

Die Funktion `zerl_filtern` *ist in der separten Matlab-Datei* `zerl_filtern.m` *folgendermaßen definiert:*

Der Code der Funktion `zerl_filtern`

```
function X_ = zerl_filtern(X,F)

  XF_koeff = conv(X.koeff,F.koeff);
  XF_grad = X.grad + F.grad;

  XF_down_koeff = dyaddown(XF_koeff,XF_grad +1);
  XF_down_grad = ceil(XF_grad/2);

  X_ = struct('koeff',XF_down_koeff,'grad',XF_down_grad);

end
```

4.2.2 Rekonstruktion einer MP3-Audiodatei

Für die Rekonstruktion einer Audiodatei benötigt man ein zerlegtes Audiosignal $c = c^{(n)} \in \ell_0(\mathbb{Z})$ und die zugehörigen Details $d^{(i)} \in \ell_0(\mathbb{Z})$ $(i = 1, ..., n)$, welche Schritt für Schritt zu den wiederhergestellten $c^{(i)}$ addiert werden sollen. Wenn die unveränderten Details $d^{(i)}$, aus Algorithmus 4.5, verwendet werden, ist die Ausgabe das ursprüngliche Signal $c^{(0)} \in \ell_0(\mathbb{Z})$, aus Algorithmus 4.5. Dann spricht man von einer Perfekten Rekonstruktion.

Der Algorithmus kann allerdings auch mit modifizierten Details angewendet werden. Dieser Vorgang wird in Beispiel 4.11 illustriert.

Algorithmus 4.7 (Algorithmus zur Rekonstruktion von Signalen)**.**

Voraussetzungen und Eingabe:

- *Zerlegtes Audiosignal* $c^{(n)} \in \ell_0(\mathbb{Z}),\ n \in \mathbb{N}$,
- *Details* $d^{(i)} \in \ell_0(\mathbb{Z}),\ i = 1, \ldots, n$,
- *Rekonstruktionsfolgen* $p \in \ell_0(\mathbb{Z})$ *und* $q \in \ell_0(\mathbb{Z})$*, welche die Identitäten aus dem Satz 4.15 (bzw. 4.20) erfüllen oder laut Algorithmus 4.12 konstruiert sind.*

Berechnung:
Berechne für i=n,...,1

- $c^{(i-1)} := \left[\left(\uparrow c^{(i)} \right) * p \right] + \left[\left(\uparrow d^{(i)} \right) * q \right]$

Ausgabe:

- $c^{(0)}$

Ende.

Implementierung in Matlab 4.8 (Algorithmus zur Rekonstruktion einer MP3-Datei)**.**

Eingabeparameter:

```
C = struct('koeff',c,'grad',C_grad);

D = % gebe hier Details als Vektor mit n Strukturen ein

P = struct('koeff',p,'grad',P_grad);
Q = struct('koeff',q,'grad',Q_grad);

n = % gebe hier Anzahl der Details ein, die angeheftet werden
```

Rekonstruktion n-mal ausführen:

```
for i=n:-1:1
    C = rekonstr_filtern(C,P);
    D(i) = rekonstr_filtern(D(i),Q);
    C = zusammensetzen(C,D(i));
end
```

Die Funktion `rekonstr_filtern` *ist in der separaten Matlab-Datei* `rekonstr_filtern.m` *folgendermaßen definiert:*

Der Code der Funktion `rekonstr_filtern`

```
function X_ = rekonstr_filtern(X,F)

  X_up_koeff = dyadup(X.koeff,0);
  X_up_grad = X.grad * 2;

  X_up_mit_F_koeff = conv(X_up_koeff,F.koeff);
  X_up_mit_F_grad = X_up_grad + F.grad;
  X_=struct('koeff',X_up_mit_F_koeff,'grad',X_up_mit_F_grad);

end
```

Die Funktion `zusammensetzen` *ist in der separaten Matlab-Datei* `zusammensetzen.m` *folgendermaßen definiert:*

Der Code der Funktion `zusammensetzen`

```
function C = zusammensetzen(C,D)

  if (D.grad < C.grad)
    for i=1:(C.grad - D.grad)
      C.koeff = [0 C.koeff];
    end
  end

  if (C.grad < D.grad)
    for i=1:(D.grad - C.grad)
      D.koeff = [0 D.koeff];
    end
  end

  if length(C.koeff) > length(D.koeff)
    for i=1:(length(C.koeff)-length(D.koeff))
      D.koeff = [D.koeff 0];
    end
  end

  if length(C.koeff) < length(D.koeff)
    for i=1:(length(D.koeff)-length(C.koeff))
      C.koeff = [C.koeff 0];
    end
  end

  C.koeff = C.koeff + D.koeff;

end
```

Zur Erinnerung: Vor dem Addieren müssen die Vektoren `C.koeff` *und* `D.koeff` *auf dieselbe Länge gebracht werden, indem sie passend mit Nullen aufgefüllt werden. In dem einführenden Beispiel 4.1 wurde gezeigt, weshalb das nötig ist.*

4.2.3 Perfekte Rekonstruktion

Nachdem nun die Algorithmen zur Zerlegung und Rekostruktion einer Audiodatei bekannt sind, ist es notwendig, zu begründen, weshalb die Perfekte Rekonstruktion funktioniert. Dafür betrachtet man *einen* Schritt der Zerlegung (laut Algorithmus 4.5) und der anschließenden Rekonstruktion (laut Algorithmus 4.7). Allgemein formuliert:

Gegeben:

- MP3-Datei als Folge $c \in \ell_0(\mathbb{Z})$
- Folgen $a, b, p, q \in \ell_0(\mathbb{Z})$, welche die Identitäten aus Satz 4.15 erfüllen (zum Beispiel aus Algorithmus 4.12)

Ausgabe des Algorithmus 4.5, für n=1:

$$c^{(1)} :=\downarrow (c * a) \quad \textit{und} \quad d^{(1)} :=\downarrow (c * b). \tag{4.2.1}$$

Die Ausgabe des Zerlegungsschrittes wird nun als Eingabe für den Rekonstruktionsalgorithmus 4.7 verwendet und liefert die

Ausgabe der Rekonstruktion:

$$\uparrow [\downarrow (c * a)] * p + \uparrow [\downarrow (c * b)] * q. \tag{4.2.2}$$

Jetzt soll gezeigt werden, dass es sich bei (4.2.2) tatsächlich um die ursprüngliche gegebene Folge c handelt und die Perfekte Rekonstruktion stattgefunden hat. Dafür muss folgender Satz bewiesen werden.

Satz 4.9 (Perfekte Rekonstruktion)**.**
Für ein zeitdiskretes Signal $c \in \ell_0(\mathbb{Z})$ und Zerlegungs- und Rekonstruktionsfolgen $a, b, p, q \in \ell_0(\mathbb{Z})$, welche die Identitäten aus Satz 4.15 erfüllen, gilt:

$$\uparrow [\downarrow (c * a)] * p + \uparrow [\downarrow (c * b)] * q = c.$$

Beweis:
Man betrachtet die Folgenoperationen in (4.2.2) als ihre äquivalenten Polynomoperationen und erhält für die Folge

$$f := \uparrow [\downarrow (c * a)] * p + \uparrow [\downarrow (c * b)] * q$$

das entsprechende Laurentpolynom ($z \in \mathbb{C}^{\times}$)

$$\begin{aligned} f^*(z) &:= \sum_{k\in\mathbb{Z}} f_k z^k \\ &= \tfrac{1}{2}\left[c^*(z)a^*(z) + c^*(-z)a^*(-z)\right] p^*(z) + \tfrac{1}{2}\left[c^*(z)b^*(z) + c^*(-z)b^*(-z)\right] q^*(z). \end{aligned}$$

Umgeformt ist

$$f^*(z) = \tfrac{1}{2}\Big[c^*(z)\left[a^*(z)p^*(z) + b^*(z)q^*(z)\right] + c^*(-z)\left[a^*(-z)p^*(z) + b^*(-z)q^*(z)\right]\Big]. \tag{4.2.3}$$

Zur Erinnerung: laut (2.0.1) und (2.0.2) sind die modifizierten Laurentpolynome $P^*(z) = \frac{1}{2}p^*(z)$ und $Q^*(z) = \frac{1}{2}q^*(z)$. Damit ist

$$f^*(z) = c^*(z)\left[a^*(z)P^*(z) + b^*(z)Q^*(z)\right] + c^*(-z)\left[a^*(-z)P^*(z) + b^*(-z)Q^*(z)\right]. \tag{4.2.4}$$

Falls $a^*(z), b^*(z), P^*(z), Q^*(z)$ die Identitäten aus dem Satz 4.15 erfüllen (z.B. aus dem Algorithmus 4.12 sind), ist, laut Satz 4.20, $[a^*(z)P^*(z) + b^*(z)Q^*(z)]$ gleich 1 und $[a^*(-z)P^*(z) + b^*(-z)Q^*(z)]$ gleich 0 und es gilt

$$\begin{aligned} f^*(z) &= c^*(z)\left[a^*(z)P^*(z) + b^*(z)Q^*(z)\right] + c^*(-z)\left[a^*(-z)P^*(z) + b^*(-z)Q^*(z)\right] \\ &= c^*(z) + c^*(-z)0 \\ &= c^*(z). \end{aligned}$$

□

Damit ist gezeigt, dass ein Schritt der Zerlegung und Rekonstruktion einer MP3-Audiodatei funktioniert. Das genügt, um die perfekte Rekonstruktion zu begründen, da dieser Schritt iterativ auf die Signalfolge angewendet wird.

4.2.4 Beispiele mit realen Audiodaten

In diesem Unterkapitel geht es darum, die Algorithmen 4.5 und 4.7, zur Zerlegung und Rekonstruktion, an realen Audiodaten in Matlab anzuwenden. In folgendem Beispiel wird die Perfekte Rekonstruktion demonstriert.

Beispiel 4.10 (Perfekte Rekonstruktion von Audiodaten)**.**

Eingabeparameter:

- *MP3-Datei* `hallo.mp3`
- *Zerlegungsfolgen* $a = \left\{\frac{3}{2}, -\frac{1}{2}\right\}_0^1, b = \left\{\frac{1}{8}, -\frac{3}{8}, \frac{3}{8}, -\frac{1}{8}\right\}_0^3 \in \ell_0(\mathbb{Z})$
- *Rekonstruktionsfolgen* $p = \left\{\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}\right\}_{-1}^2, q = \{-3, -1\}_{-1}^0 \in \ell_0(\mathbb{Z})$
- *Anzahl der Zerlegungsschritte* $n = 3$

Definiere die obigen Eingabeparameter in Matlab:

Matlab Code

```
[c,Fs] = audioread('hallo.mp3'); C_grad = 0;

a = [3/2 -1/2];                  A_grad = 0;
b = [1/8 -3/8 3/8 -1/8];         B_grad = 0;

p = [1/4 3/4 3/4 1/4];           P_grad = -1;
q = [-3 -1];                     Q_grad = -1;

n = 3;
```

Definiere die zugehörigen Laurentpolynomstrukturen:

Matlab Code

```
C = struct('koeff',c,'grad',C_grad);

A = struct('koeff',a,'grad',A_grad);
B = struct('koeff',b,'grad',B_grad);

P = struct('koeff',p,'grad',P_grad);
Q = struct('koeff',q,'grad',Q_grad);
```

Wende den Zerlegungsalgorithmus 4.5 an:

Matlab Code

```
for i=1:n
     D(i) = zerl_filtern(C,B);
     C = zerl_filtern(C,A);
end
```

In der Abbildung 1 ist der Zerlegungsalgorithmus mit den gefilterten Audiodaten $c^{(i)}$ und den zugehörigen Details $d^{(i)}$ grafisch dargestellt.

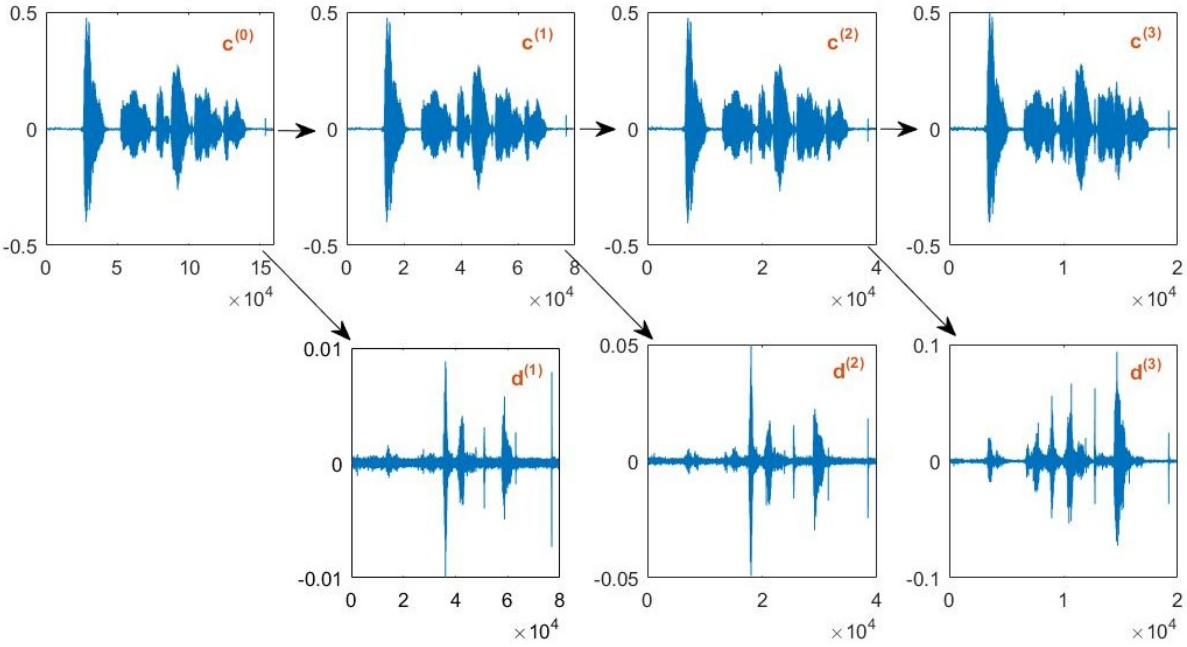

Abbildung 1: Die Zerlegung der Audiodaten $c^{(0)}$ in 3 Schritten

Wie man in der obigen Abbildung sehen kann, wird die Länge der Datenvektoren der $c^{(i)}$ mit jedem Zerlegungsschritt, durch das Downsampling, halbiert (beachte die Skalierung).

Man kann sich die Audiosignale $c^{(i)}$ anhören, indem man den Matlab Befehl `sound` *zum Abspielen verwendet. Dieser wird folgendermaßen eingegeben.*

Matlab Code

```
sound(C.koeff,Fs);
```

Der erste Parameter des Befehls ist der Audiodatenvektor (in diesem Fall `C.koeff`*) und der zweite Parameter ist die Abtastrate.*

Spielt man das einmal gefilterte $c^{(1)}$ mit der Abtastrate `Fs` *des Ursprungsignals $c^{(0)}$ ab, hört sich die Tonhöhe der Audioaufnahme höher als die Originalaufnahme an. Das liegt daran, dass in $c^{(1)}$ nur mehr die Hälfte der Daten enthalten sind und beim Abspielen des Signals die Abtastrate auf* `Fs/2` *angepasst werden muss. Hört man sich $c^{(1)}$ mit der entsprechenden Abtastrate* `Fs/2` *an, stimmt die Tonhöhe mit dem Originalsignal überein; allerdings kann beim Abhören die Kompression der Audiodaten festgestellt werden - umgangssprachlich wird manchmal von einer „schlechteren Audioqualität" gesprochen. Spielt man $c^{(2)}$ mit der entsprechenden Abstastrate* `Fs/4` *ab, kann eine noch stärkere Kompression (noch stärkere „Verschlechterung der Audioqualität") festgestellt werden (usw).*

Für die Perfekte Rekonstruktion von $c^{(0)}$ werden die Details $d^{(3)}, d^{(2)}, d^{(1)}$ und die dreimal gefilterten Audiodaten $c^{(3)}$ benötigt. Es wird $d^{(3)}$ an $c^{(3)}$ angeheftet, um $c^{(2)}$ zu rekonstruieren; $d^{(2)}$ an $c^{(2)}$ angeheftet, um $c^{(1)}$ zu rekonstruieren und schließlich $d^{(1)}$ an $c^{(1)}$ angeheftet, um $c^{(0)}$ perfekt zu rekonstruieren.

Ein weiterer Anwendungsbereich der Algorithmen für die Zerlegung und anschließenden Rekonstruktion ist das Entrauschen von Audiosignalen. Unter Rauschen versteht man meist eine unerwünschte Störgröße (mit unspezifischem Frequenzspektrum), die mit dem Originalsignal verbunden ist (oder die zum Originalsignal addiert wird).

Mit Hilfe der Algorithmen 4.5 und 4.7 kann unerwünschtes Rauschen entfert werden. Dafür muss das Audiosignal zuerst mit Algorithmus 4.5 in endlich vielen Schritten zerlegt werden. Dadurch können die Details $d^{(i)} \in \ell_0(\mathbb{Z})$ verarbeitet werden. Anschließend werden die unerwünschten Details gleich Null gesetzt. Das heißt: wähle einen Schwellenwert $TOL \in \mathbb{R}$ und verwerfe alle Details $d_j^{(i)}$, für die $|d_j^{(i)}| < TOL$ gilt. Durch dieses Nullsetzen ausgesuchter Details können gewisse Frequenzen des ursprünglichen Signals, wie Rauschen, entfernt werden. Im Anschluss wird der Algorithmus 4.7, zur Rekonstruktion, auf das gefilterte Signal $c^{(n)}$ und die verarbeiteten Details angewendet. Dadurch, dass manche, oder alle, Details verworfen wurden, findet keine Perfekte Rekonstruktion statt. Das ist allerdings erwüscht, da eine Rauschverminderung im ursprünglichen Signal angestrebt wird.

Im Folgenden wird der oben geschilderte Vorgang anhand eines Beispiels illustriert. Die Details $d^{(i)}$ werden in Beispiel 4.11 zuerst ohne Schwellenwertbildung allesamt verworfen. Danach wird der Vorgang mit Schwellenwertbildung erneut durchgeführt und die beiden Ergebnisse miteinander verglichen.

Beispiel 4.11 (Audiosignal entrauschen)**.**

Für dieses Beispiel zum Entrauschen eines Audiosignals werden Zerlegungs- und Rekonstruktionsfolgen verwendet, die nicht aus dem Algorithmus 4.12 stammen. Die Folgen erfüllen allerdings die Identitäten aus dem Satz 4.15 und können deshalb für die Algorithmen 4.5 und 4.7 verwendet werden.

Der Grund für die Verwendung anderer Zerlegungs- und Rekonstruktionsfolgen ist, dass sie sich gut für die Entrauschung eignen. Die in Beispiel 4.1 zur Kompression und anschließenden Perfekten Rekonstruktion verwendeten Folgen wären zum Entrauschen nicht so gut geeignet. Es gibt also Zerlegungs- und Rekonstruktionsfolgen, die für bestimmte Anwendungen besser/schlechter geeignet sind.

Eingabe:

- *Audiodatei* `hallo.mp3` *+ Rauschen*
- *Zerlegungsfolgen* $a = \left\{\frac{1}{2}, \frac{1}{2}\right\}_0^1,\ b = \left\{\frac{1}{2}, -\frac{1}{2}\right\}_0^1 \in \ell_0(\mathbb{Z})$
- *Rekonstruktionsfolgen* $p = \{1, 1\}_{-1}^0,\ q = \{-1, 1\}_{-1}^0 \in \ell_0(\mathbb{Z})$

In diesem Beispiel wird eine MP3-Audioaufnahme eines Menschen verwendet, der den Satz:„Hallo, mein Name ist Michael Thomas Schiller“ spricht. Weil diese Audioaufnahme kaum Rauschen enthält, wird, damit das Entrauschen eindrucksvoller ist, in Matlab künstliches Rauschen zu dem Audiosignal addiert. Das kann mit dem Befehl `awgn` *(add white Gaussian noise) bewerkstelligt werden.*
Dieser wird folgendermaßen eingegeben.

```
awgn(Audiosignal,Signal-Rauschen-Verhaeltnis,'measured')
```

Der erste Parameter ist das Audiosignal. Im zweiten Parameter kann das Verhältnis „Signal-zu-Rauschen“ in Db angegeben werden. Als dritter Parameter kann `'measured'` *eingegeben werden, damit die „power“ des Audiosignals gemessen wird, bevor das Rauschen addiert wird.*

Der Befehl `awgn` *angewendet auf die Audioaufnahme* `hallo.mp3` *sieht in Matlab folgendermaßen aus.*

Matlab Code

```
[c,Fs] = audioread('hallo.mp3'); C_grad = 0;
C = struct('koeff',c,'grad',C_grad);

C.koeff = awgn(C.koeff,1,'measured');
```

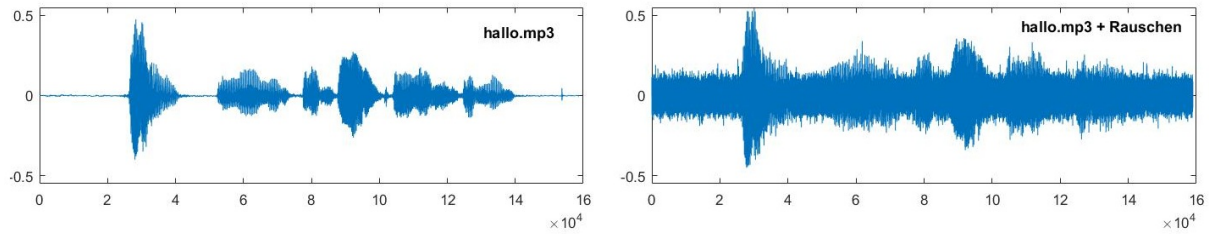

Abbildung 2: Die Auswirkungen des Befehls `awgn` *auf die Aufnahme* `hallo.mp3`

Nun wird der Algorithmus 4.5 zur Zerlegung mit $n = 5$ angewendet, um die Details $d^{(i)}$ zu identifizieren.

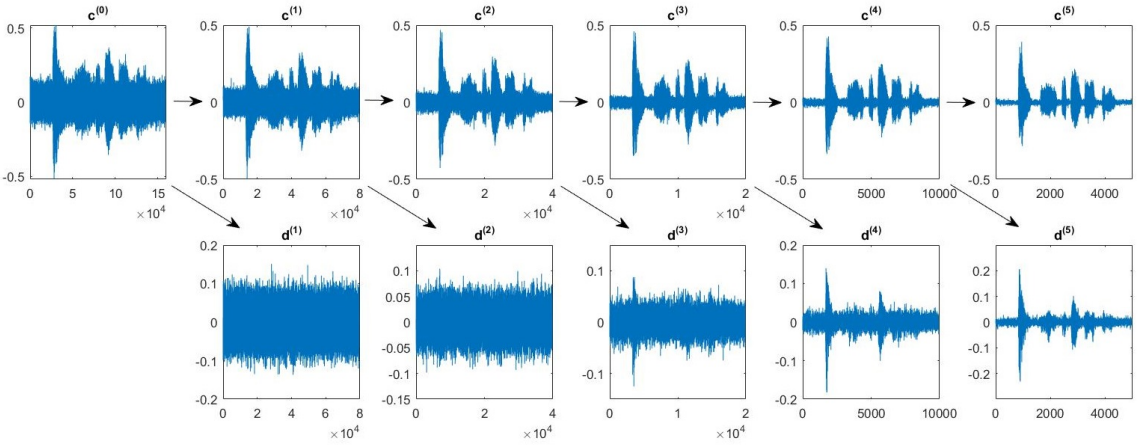

Abbildung 3: Die fünfmalige Zerlegung der verrauschten Aufnahme `hallo.mp3`

Verwirft man nun alle Details, indem man alle $d_j^{(i)}$ gleich Null setzt und anschließend den Rekonstruktionsalgorithmus (mit $c^{(5)}$ und den nullgesetzten $d^{(i)}$) anwendet, erhält man folgendes Resultat.

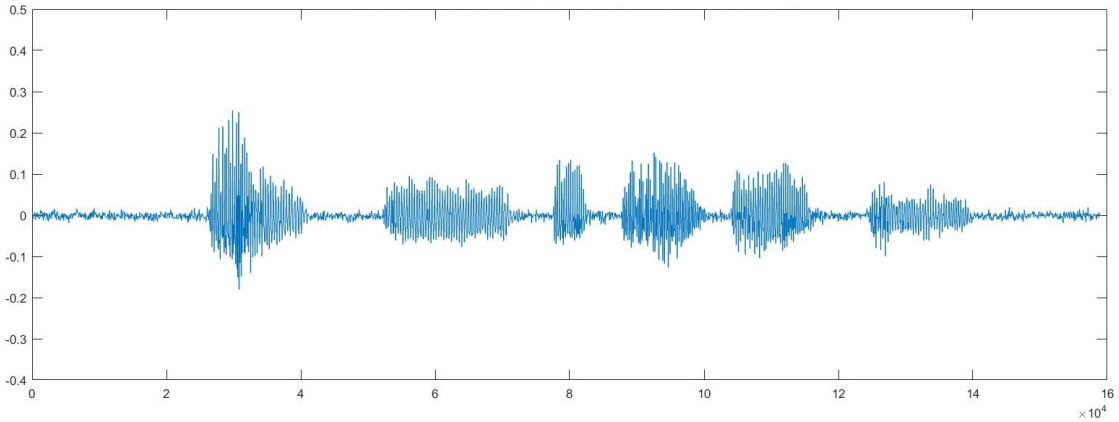

Abbildung 4: Die entrauschte Audioaufnahme `hallo.mp3`*, ohne Schwellenwertbildung*

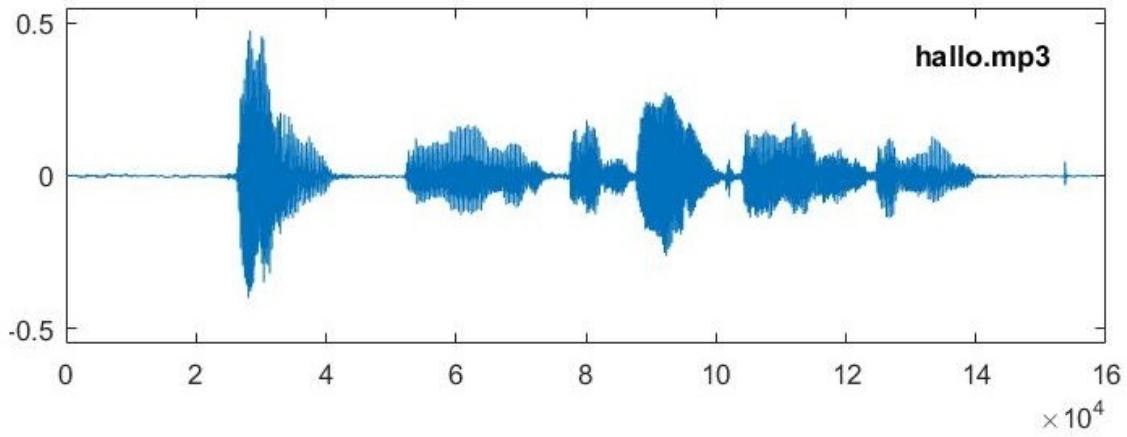

Abbildung 5: Die Originalaufnahme `hallo.mp3`

Wie man sehen kann, ähnelt das entrauschte Audiosignal (Abb.4) dem Originalsignal (Abb.5) und das Rauschen wurde fast komplett entfernt. Allerdings erkennt man in der grafischen Darstellung, dass das rekonstruierte Signal „ausgedünnt" ist. Es sind nämlich einige Frequenzen des Originalsignals, samt dem Rauschen, verworfen worden. Das Entrauschen ohne Schwellenwertbildung führt also dazu, dass zu viele Frequenzen des Originalsignals herausgefiltert werden.

Deshalb wird nun der Vorgang noch einmal mit Schwellenwertbildung wiederholt, sodass nur ausgesuchte $d_j^{(i)}$ nullgesetzt werden und dadurch mehr Frequenzen des Originalsignals rekonstruiert werden können.

Die verwendete Methode ist die des sogenannten „hard threshold". Dabei wird ein Schwellenwert $TOL \in \mathbb{R}$ gewählt und alle $|d_j^{(i)}| < TOL$ nullgesetzt.

Als Schwellenwert wird in diesem Beispiel der „universal threshold" gewählt, der folgendermaßen mit dem Median der Details gebildet wird [5, Kap.3.2].

$$TOL(d^{(i)}) := \frac{\sqrt{(2 \cdot \log(Laenge\ des\ Datenvektors)} \cdot median(|d^{(i)}|)}{0.6745} \in \mathbb{R}$$

Nun folgt die Implementierung des „hard universal threshold" in Matlab.

Matlab Code

```
for i=1:n
 TOL=(sqrt(2*log(length(c')))*median(abs(D(i).koeff)))/0.6745
   for j=1:length(D(i).koeff)
      if abs(D(i).koeff(j)) < TOL
         D(i).koeff(j)=0;
      end
   end
end
```

Es wird also für alle `D(i).koeff` *ein Schwellenwert* `TOL` *gebildet und anschließend alle* `D(i).koeff(j)` *mit* `abs(D(i).koeff(j))<TOL` *nullgesetzt.*

Danach werden die modifizierten `D(i)` *der Reihe nach mit dem Rekonstruktionsalgorithmus 4.7 zu* `C` *addiert.*

Das Resultat sieht folgendermaßen aus.

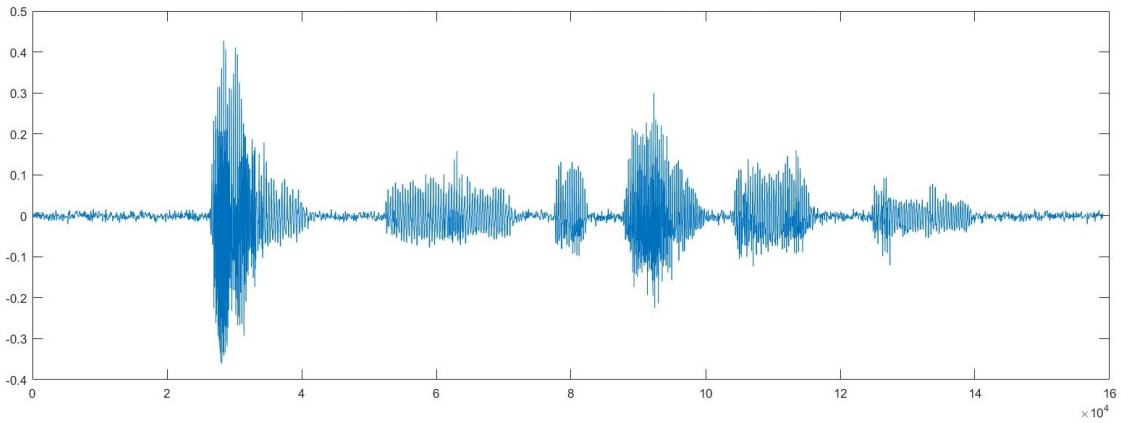

Abbildung 6: Die entrauschte Audioaufnahme `hallo.mp3`*, mit Schwellenwertbildung*

Vergleicht man die beiden Abbildungen 4 und 6, fällt einem sofort auf, dass die mit Schwellenwert rekonstruierte Aufnahme (Abb. 6) mehr Frequenzen des Originalsignals (Abb. 5) enthält. Zum Entrauschen von Audioaufnahmen wird daher die Methode mit Schwellenwertbildung empfohlen. Dadurch können mehr Frequenzen der Originalaufnahme rekonstruiert werden und die Rauschreduktion ist dennoch gegeben.

4.3 Konstruktion von Zerleguns- und Rekonstruktionsfolgen

4.3.1 Algorithmus zur Konstruktion von Zerlegungs- und Rekonstruktionsfolgen

In diesem Kapitel wird der Algorithmus 4.12 zur Kontruktion der Zerlegungs- und Rekontruktionsfolgen (bzw. der zugehörigen Laurentpolynome) vorgestellt. Es wird gezeigt, dass diese Folgen die Identitäten aus den Sätzen 4.15 und 4.20 erfüllen. Dadurch kann die Perfekte Rekonstruktions-Eigenschaft des Algorithmus 4.7, zur Rekonstruktion von Audiodateien, garantiert werden.

Algorithmus 4.12 (Konstruktion von $a^*(z), b^*(z), Q^*(z)$ $[3, S.360]$)**.**

Folgende Vorraussetzungen müssen überprüft werden, um den Algorithmus anwenden zu können.

***Gegeben**:*

- *Eine zentrierte Folge* $p = \{p_j\}_\mu^\nu \in \ell_0(\mathbb{Z})$ *und*
- *eine zentrierte Skalierungsfunktion mit kompaktem Träger* $\phi : \mathbb{R} \to \mathbb{R}$*, mit* $\phi(x) := \sum_{j=\mu}^{\nu} p_j \phi(2x - j), \quad \mu \leq -1, \nu \geq 1.$

Für die numerische Implementierung des Algorithmus 4.7 ist es aus praktischen Gründen ratsam, die Indizes von $\{p_j\}$ *so zu verschieben, dass die Folge um* p_0 *zentriert ist [3, S.78].*

***Überprüfe**:*

- *Die lineare Unabhängigkeit von* $\{\phi(x - j), j \in \mathbb{Z}\}$ *auf* $\mathbb{R}$*. Also*

$$\forall x \in \mathbb{R} : \sum_{j \in \mathbb{Z}} c_j \phi(x - j) = 0 \Leftrightarrow \forall c_j = 0, \quad j \in \mathbb{Z}. \tag{4.3.1}$$

  Diese Eigenschaft impliziert nämlich die numerische Stabilität des Algorithmus 4.7. Laut $[3, Lemma\ 2.4.2]$ *folgt aus der linearen Unabhängigkeit von* $\{\phi(x - j), j \in \mathbb{Z}\}$ *auf dem Intervall* $[0, n]$*,* $n \in \mathbb{N}$*, die lineare Unabhängigkeit auf ganz* $\mathbb{R}$*. Um (4.3.1) nachzuweisen, genügt es also, zum Beispiel, die lineare Unabhängigkeit auf dem Intervall* $[0, 1]$ *zu zeigen.*

- *Die Summenregel*

$$\sum_{j \in \mathbb{Z}} p_{2j} = \sum_{j \in \mathbb{Z}} p_{2j-1} = 1. \tag{4.3.2}$$

Die Summenregel (4.3.2) ist notwendig, damit der Algorithmus 4.7 konvergiert [3, Theorem 4.1.1]. Sie impliziert

$$P^*(1) = \tfrac{1}{2}p^*(1) = \tfrac{1}{2}\sum_{j\in\mathbb{Z}} p_j = 1 \tag{4.3.3}$$

und

$$P^*(-1) = \tfrac{1}{2}\sum_{j\in\mathbb{Z}} p_j(-1)^j = \tfrac{1}{2}\left(\sum_{j\in\mathbb{Z}} p_{2j} - \sum_{j\in\mathbb{Z}} p_{2j-1}\right) = 0. \tag{4.3.4}$$

Aus (4.3.4) folgt, dass das Laurentpolynom $P^*(z)$ die Nullstelle $z = -1$ hat und deshalb den Linearfaktor $(1+z),\ z \in \mathbb{C}^\times$, enthält.

- *$P^*(z) := \frac{1}{2}\sum_{j\in\mathbb{Z}} p_j z^j$, mit $z \in \mathbb{C}^\times$, hat keine symmetrischen Nullstellen in $\mathbb{C}^\times$.*
  D.h.: $\nexists z \in \mathbb{C}^\times : P^*(z) = P^*(-z) = 0$.
  Dies ist notwendig, um die Existenz und Eindeutigkeit von $h^*(z) \in \Lambda[\mathbb{C}^\times]$, aus Schritt 2 des Algorithmus, zu garantieren [3, Satz 7.1.1].

Nach der Überprüfung der obigen Punkte,

Wähle $\ell \in \mathbb{N}_0$.

Führe anschließend die Schritte 1 *bis* 5 *zur Konstruktion von $a^*(b), b^*(z), Q^*(z) \in \ell_0(\mathbb{Z})$ durch.*

1.Schritt: *Definiere das Laurentpolynom $g^*(z) \in \Lambda[\mathbb{C}^\times]$*

$$g^*(z) := \begin{cases} z^{-\mu}(\frac{1+z}{2})^\ell P^*(z), & z \in \mathbb{C}^\times \\ (\frac{1}{2})^{\ell+1} p_\mu, & z = 0. \end{cases}$$

2.Schritt: *Definiere das Polynom $h^*(z) \in \Lambda[\mathbb{C}^\times]$*

als das kleinstgradige Polynom, welches die folgende Identität erfüllt.

$$g^*(z)h^*(z) - g^*(-z)h^*(-z) = z^{2\lfloor\frac{1}{2}(\nu-\mu+\ell)\rfloor-1}, \quad z \in \mathbb{C}^\times. \tag{4.3.5}$$

Die Berechnung von $h^*(z)$ basiert auf dem Euklidischen Algorithmus. Die Existenz, Eindeutigkeit und Berechnung von $h^*(z)$ wird in Kapitel 7 des Buches Wavelet Subdivision Methods [3] behandelt und ist nicht Teil dieser Arbeit.

In der Tabelle 6.1 können allerdings die, für den Algorithmus benötigten, Koeffizienten von $h^*(z)$ nachgesehen werden.

3.Schritt: *Definiere das Laurentpolynom* $a^*(z) \in \Lambda[\mathbb{C}^\times]$

$$a^*(z) = \sum_{j\in\mathbb{Z}} a_j z^j := z^{-2\lfloor\frac{1}{2}(\nu+\ell)\rfloor+\sigma_{\mu,\nu,\ell}+1} \left(\tfrac{1+z}{2}\right)^\ell h^*(z), \quad z \in \mathbb{C}^\times; \quad (4.3.6)$$

mit

$$\sigma_{\mu,\nu,\ell} := \begin{cases} 0, & \text{wenn } \mu \text{ gerade,} \\ 1, & \text{wenn } \mu \text{ ungerade und } \nu+\ell \text{ gerade,} \\ -1, & \text{wenn } \mu \text{ ungerade und } \nu+\ell \text{ ungerade.} \end{cases} \quad (4.3.7)$$

4.Schritt: *Definiere das Laurentpolynom* $b^*(z) \in \Lambda[\mathbb{C}^\times]$

$$b^*(z) = \sum_{j\in\mathbb{Z}} b_j z^j := -z^{-2\left\lfloor\frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} P^*(-z), \quad z \in \mathbb{C}^\times. \quad (4.3.8)$$

5.Schritt: *Definiere das Laurentpolynom* $Q^*(z) \in \Lambda[\mathbb{C}^\times]$

$$Q^*(z) = \tfrac{1}{2}\sum_{j\in\mathbb{Z}} q_j z^j := (-1)^\mu z^\mu \left(\tfrac{1-z}{2}\right)^\ell h^*(-z), \quad z \in \mathbb{C}^\times. \quad (4.3.9)$$

Ende.

4.3.2 Beispiele zur Konstruktion von Zerlegungs- und Rekonstruktionsfolgen

Im Folgenden wird der Algorithmus 4.12 anhand eines einfachen Beispiels veranschaulicht.

Beispiel 4.13 (Konstruktion der Zerlegungs- und Rekonstruktionsfolgen)**.**

***Gegeben**:*

- *Die zentrierte Folge $p = \{p_j\} \in \ell_0(\mathbb{Z})$, mit $\{p_j\}_{-1}^{1} = \left\{\frac{1}{2}, 1, \frac{1}{2}\right\}$, also $\mu = -1$ und $\nu = 1$.*
- *Die sogenannte Hut-Funktion*

$$\phi : \mathbb{R} \to \mathbb{R}, \quad \phi(x) := \begin{cases} x+1, & x \in [-1, 0) \\ -x+1, & x \in [0, 1] \\ 0, & \textit{sonst.} \end{cases} \tag{4.3.10}$$

erfüllt

$$\phi(x) = \sum_{j=-1}^{1} p_j \phi(2x - j), \quad x \in \mathbb{R}.$$

Denn

$$\phi(x) = \tfrac{1}{2}\phi(2x+1) + \phi(2x) + \tfrac{1}{2}\phi(2x-1), \quad x \in \mathbb{R}. \tag{4.3.11}$$

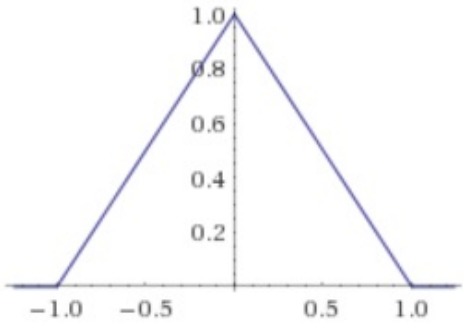

Abbildung 7: die Hut-Funktion

Um die Gleichheit in (4.3.11) nachzuweisen, ist es hilfreich, zuerst

$$\phi(2x+1) = \begin{cases} 2x+2, & x \in \left[-1, -\frac{1}{2}\right) \\ -2x, & x \in \left[-\frac{1}{2}, 0\right] \\ 0, & sonst. \end{cases}$$

$$\phi(2x) = \begin{cases} 2x+1, & x \in \left[-\frac{1}{2}, 0\right) \\ -2x+1, & x \in \left[0, \frac{1}{2}\right] \\ 0, & sonst. \end{cases}$$

$$\phi(2x-1) = \begin{cases} 2x, & x \in \left[0, \frac{1}{2}\right) \\ -2x+2, & x \in \left[\frac{1}{2}, 1\right] \\ 0, & sonst. \end{cases}$$

zu betrachten. Anschließend wird folgende Fallunterscheidung getroffen.

1.Fall: $x \in \left[-1, -\frac{1}{2}\right)$

$\frac{1}{2}\phi(2x+1) + \phi(2x) + \frac{1}{2}\phi(2x-1) = \frac{1}{2}(2x+2) + 0 + \frac{1}{2} \cdot 0 = x+1 = \phi(x).$

2.Fall: $x \in \left[-\frac{1}{2}, 0\right)$

$\frac{1}{2}\phi(2x+1) + \phi(2x) + \frac{1}{2}\phi(2x-1) = \frac{1}{2}(-2x) + 2x + 1 + \frac{1}{2} \cdot 0 = x+1 = \phi(x).$

3.Fall: $x \in \left[0, \frac{1}{2}\right)$

$\frac{1}{2}\phi(2x+1) + \phi(2x) + \frac{1}{2}\phi(2x-1) = \frac{1}{2} \cdot 0 - 2x + 1 + \frac{1}{2}(2x) = -x+1 = \phi(x).$

4.Fall: $x \in \left[\frac{1}{2}, 1\right]$

$\frac{1}{2}\phi(2x+1) + \phi(2x) + \frac{1}{2}\phi(2x-1) = \frac{1}{2} \cdot 0 + 0 + \frac{1}{2}(-2x+2) = -x+1 = \phi(x).$

Damit ist (4.3.11) bewiesen.

Voraussetzungen für den Algorithmus überprüfen:

- *(4.3.1) überprüfen:*
  Weil die lineare Unabhängigkeit von $\{\phi(x-j), j \in \mathbb{Z}\}$ *auf dem Intervall* $[0,1]$ *die lineare Unabhängigkeit auf ganz* $\mathbb{R}$ *impliziert, genügt es, zu beweisen, dass gilt*

$$\forall x \in [0,1] : \sum_{j \in \mathbb{Z}} c_j \phi(x-j) = 0 \Leftrightarrow c_0 = c_1 = 0. \qquad (4.3.12)$$

Beweis:
$\boxed{\Rightarrow}$ Es wird $\sum_{j\in\mathbb{Z}} c_j\phi(x-j) = 0$ für $x \in \{0,1\}$ betrachtet, um ein linear unabhängiges Gleichungssystem zu erhalten.
Für $x = 0$: wegen $\phi(x) = 0\ \forall x \notin [-1,1]$, gilt

$$\begin{aligned}\sum_{j\in\mathbb{Z}} c_j\phi(0-j) &= \cdots + c_{-1}\cdot 0 + c_0\cdot 1 + c_1\cdot 0 + \ldots \\ &= c_0\cdot 1 = 0 \Rightarrow c_0 = 0.\end{aligned}$$

Für $x = 1$: wegen $\phi(x) = 0\ \forall x \notin [-1,1]$, gilt

$$\begin{aligned}\sum_{j\in\mathbb{Z}} c_j\phi(1-j) &= \cdots + c_0\cdot 0 + c_1\cdot 1 + c_2\cdot 0 + \ldots \\ &= c_1\cdot 1 = 0 \Rightarrow c_1 = 0.\end{aligned}$$

$\boxed{\Leftarrow}$ Es gelte $c_0 = c_1 = 0$.
Für $x \in [0,1]$, weil ϕ, laut (4.3.10), auf den Intervallen $(-\infty,-1]$ und $[1,\infty)$ gleich 0 ist, folgt

$$\begin{aligned}&\sum_{j\in\mathbb{Z}} c_j\phi(x-j) \\ &= \cdots + c_{-1}\phi(x+1) + c_0\phi(x) + c_1\phi(x-1) + c_2\phi(x-2) + \ldots \\ &= \cdots + c_{-1}\phi(x+1) + 0\cdot\phi(x) + 0\cdot\phi(x-1) + c_2\phi(x-2) + \ldots \\ &= \cdots + c_{-1}\cdot 0 + 0\cdot\phi(x) + 0\cdot\phi(x-1) + c_2\cdot 0 + \cdots = 0.\end{aligned}$$ □

- *Die Summenregel* (4.3.2)*, welche eine Voraussetzung für den Algorithmus ist, wird erfüllt, denn*

$$\begin{aligned}\sum_{j\in\mathbb{Z}} p_{2j} &= p_0 = 1, \\ \sum_{j\in\mathbb{Z}} p_{2j-1} &= p_{-1} + p_1 = \tfrac{1}{2} + \tfrac{1}{2} = 1.\end{aligned}$$

- *Damit der Algorithmus angewendet werden kann, muss noch gezeigt werden, dass $P^*(z)$ keine symmetrischen Nullstellen in $\mathbb{C}^\times$ hat:*

$$\begin{aligned}P^*(z) = \tfrac{1}{2}\sum_{j\in\mathbb{Z}} p_j z^j &= \tfrac{1}{2}(\tfrac{1}{2}z^{-1} + 1 + \tfrac{1}{2}z) = \tfrac{1}{4}z^{-1}(1 + 2z + z^2) \\ &= \tfrac{1}{4}z^{-1}(z+1)^2.\end{aligned}$$

Daraus folgt, dass $P^*(z)$ die doppelte Nullstelle $z = -1$ und somit keine symmetrischen Nullstellen hat.

1.Fall: *Mit $\ell = 0$ liefert der Algorithmus 4.12*

$$\begin{aligned} g^*(z) &= \tfrac{1}{4} + \tfrac{1}{2} + \tfrac{1}{4}z^2, \\ h^*(z) &= 1, \\ a^*(z) &= 1, \\ b^*(z) &= \tfrac{1}{4} - \tfrac{1}{2}z + \tfrac{1}{4}z^2, \\ Q^*(z) &= -1z^{-1}, \quad z \in \mathbb{C}^\times. \end{aligned}$$

2.Fall: *Mit $\ell = 1$ liefert der Algorithmus 4.12*

$$\begin{aligned} g^*(z) &= \tfrac{1}{4} + \tfrac{3}{8}z + \tfrac{3}{8}z^2 + \tfrac{1}{4}, \\ h^*(z) &= \tfrac{3}{2} - \tfrac{1}{2}z, \\ a^*(z) &= \tfrac{3}{4} + \tfrac{1}{2}z - \tfrac{1}{4}z^2, \\ b^*(z) &= \tfrac{1}{4} - \tfrac{1}{2}z + \tfrac{1}{4}z^2, \\ Q^*(z) &= \tfrac{1}{2}\left(-\tfrac{3}{2}z^{-1} + 1 + \tfrac{1}{2}z\right), \quad z \in \mathbb{C}^\times. \end{aligned}$$

Die Zerlegungs- und Rekonstruktionsfolgen, welche der Algorithmus 4.12 für $\ell = 2$ und $\ell = 3$ liefert, können im Appendix in der Tabelle 6.2 nachgesehen werden.

Hier ein weiteres Beispiel:

Beispiel 4.14 (Konstruktion der Zerlegungs- und Rekonstruktionsfolgen)**.**

Gegeben*:*

- *Die zentrierte Folge* $p = \{p_j\} \in \ell_0(\mathbb{Z})$*, mit* $\{p_j\}_{-1}^{2} = \left\{\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}\right\}$*, also* $\mu = -1$ *und* $\nu = 2$*.*
- *Der zentrierte quadratische B-Spline*

$$\phi : \mathbb{R} \to \mathbb{R}, \quad \phi(x) := \begin{cases} \frac{1}{2}x^2 + x + \frac{1}{2}, & x \in [-1, 0) \\ -x^2 + x + \frac{1}{2}, & x \in [0, 1) \\ \frac{1}{2}x^2 - 2x + 2, & x \in [1, 2] \\ 0, & \textit{sonst.} \end{cases} \tag{4.3.13}$$

erfüllt

$$\phi(x) = \sum_{j=-1}^{2} p_j \phi(2x - j), \quad x \in \mathbb{R}.$$

Denn

$$\phi(x) = \tfrac{1}{4}\phi(2x + 1) + \tfrac{3}{4}\phi(2x) + \tfrac{3}{4}\phi(2x - 1) + \tfrac{1}{4}\phi(2x - 2), \quad x \in \mathbb{R}. \tag{4.3.14}$$

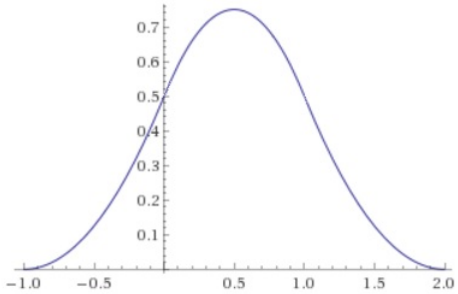

Abbildung 8: der zentrierte quadratische B-Spline

Um die Gleichheit in (4.3.14) nachzuweisen, ist es hilfreich, zuerst

$$\phi(2x+1) = \begin{cases} 2x^2+4x+2, & x \in \left[-1, -\frac{1}{2}\right) \\ -4x^2-2x-\frac{3}{2} & x \in \left[-\frac{1}{2}, 0\right) \\ 2x^2-2x+\frac{1}{2}, & x \in \left[0, \frac{1}{2}\right] \\ 0, & sonst. \end{cases}$$

$$\phi(2x) = \begin{cases} 2x^2+2x+\frac{1}{2}, & x \in \left[-\frac{1}{2}, 0\right) \\ -4x^2+2x+\frac{1}{2}, & x \in \left[0, \frac{1}{2}\right) \\ 2x^2-4x+2, & x \in \left[\frac{1}{2}, 1\right] \\ 0, & sonst. \end{cases}$$

$$\phi(2x-1) = \begin{cases} 2x^2, & x \in \left[0, \frac{1}{2}\right) \\ -4x^2+6x-\frac{3}{2}, & x \in \left[\frac{1}{2}, 1\right) \\ 2x^2-6x+\frac{9}{2}, & x \in \left[1, \frac{3}{2}\right] \\ 0, & sonst. \end{cases}$$

$$\phi(2x-2) = \begin{cases} 2x^2-2x-\frac{1}{2}, & x \in \left[\frac{1}{2}, 1\right) \\ -4x^2+10x-\frac{11}{2}, & x \in \left[1, \frac{3}{2}\right) \\ 2x^2-8x+8, & x \in \left[\frac{3}{2}, 2\right] \\ 0, & sonst. \end{cases}$$

zu betrachten. Anschließend wird folgende Fallunterscheidung getroffen.

1.Fall: $x \in \left[-1, \frac{1}{2}\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{1}{4}\phi(2x+1) = \frac{1}{4}(2x^2+4x+2)$

$= \frac{1}{2}x^2 + x + \frac{1}{2} = \phi(x).$

2.Fall: $x \in \left[-\frac{1}{2}, 0\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x)$

$= \frac{1}{4}(-4x^2-2x-\frac{3}{2}) + \frac{3}{4}(2x^2+2x+\frac{1}{2})$

$= \frac{1}{2}x^2 + x + \frac{1}{2} = \phi(x).$

3.Fall: $x \in \left[0, \frac{1}{2}\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1)$

$= \frac{1}{4}(2x^2 - 2x + \frac{1}{2}) + \frac{3}{4}(-4x^2 + 2x + \frac{1}{2}) + \frac{3}{4}(2x^2)$

$= -x^2 + x + \frac{1}{2} = \phi(x).$

4.Fall: $x \in \left[\frac{1}{2}, 1\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{3}{4}(2x^2 - 4x + 2) + \frac{3}{4}(-4x^2 + 6x - \frac{3}{2}) + \frac{1}{4}(2x^2 - 2x - \frac{1}{2})$

$= -\frac{1}{2}x^2 - 2x + 2 = \phi(x).$

5.Fall: $x \in \left[1, \frac{3}{2}\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{3}{4}(2x^2 - 6x + \frac{9}{2}) + \frac{1}{4}(-4x^2 + 10x - \frac{11}{2})$

$= \frac{1}{2}x^2 - 2x + 2 = \phi(x).$

6.Fall: $x \in \left[\frac{3}{2}, 2\right]$

$\frac{1}{4}\phi(2x+1) + \frac{3}{4}\phi(2x) + \frac{3}{4}\phi(2x-1) + \frac{1}{4}\phi(2x-2)$

$= \frac{1}{4}\phi(2x-2) = \frac{1}{4}(2x^2 - 8x + 8)$

$= \frac{1}{2}x^2 - 2x + 2 = \phi(x).$

Damit ist (4.3.14) bewiesen.

Voraussetzungen für den Algorithmus überprüfen:

- *(4.3.1) überprüfen:*
  Weil die lineare Unabhängigkeit von $\{\phi(x-j), j \in \mathbb{Z}\}$ auf dem Intervall $[0,1]$ die lineare Unabhängigkeit auf ganz $\mathbb{R}$ impliziert, genügt es, zu beweisen, dass gilt

$$\forall x \in [0,1] : \sum_{j\in\mathbb{Z}} c_j \phi(x-j) = 0 \Leftrightarrow c_{-1} = c_0 = c_1 = 0. \qquad (4.3.15)$$

  ***Beweis**:*
  $\boxed{\Rightarrow}$ Es wird $\sum_{j\in\mathbb{Z}} c_j\phi(x-j) = 0$ für $x \in \{0, \frac{1}{2}, 1\}$ betrachtet, um ein linear unabhängiges Gleichungssystem zu erhalten.
  Wegen $\phi(x) = 0\ \forall x \notin (-1,2)$, gilt
  Für $x = 0$:

$$\begin{aligned}\sum_{j\in\mathbb{Z}} c_j\phi(0-j) &= \cdots + 0 + c_{-2}\phi(2) + c_{-1}\phi(1) + c_0\phi(0) + c_1\phi(-1) + 0 + \ldots \\ &= \tfrac{1}{2}c_{-1} + \tfrac{1}{2}c_0 = 0 \Rightarrow c_0 = -c_{-1}.\end{aligned}$$

  Für $x = 1$:

$$\begin{aligned}\sum_{j\in\mathbb{Z}} c_j\phi(1-j) &= \cdots + 0 + c_{-1}\phi(2) + c_0\phi(1) + c_1\phi(0) + c_2\phi(-1) + 0 + \ldots \\ &= \tfrac{1}{2}c_0 + \tfrac{1}{2}c_1 = 0 \Rightarrow c_0 = -c_1.\end{aligned}$$

  Für $x = \frac{1}{2}$:

$$\begin{aligned}\sum_{j\in\mathbb{Z}} c_j\phi(\tfrac{1}{2}-j) &= \cdots + 0 + c_{-1}\phi(\tfrac{3}{2}) + c_0\phi(\tfrac{1}{2}) + c_1\phi(-\tfrac{1}{2}) + 0 + \ldots \\ &= \tfrac{1}{8}c_{-1} + \tfrac{4}{8}c_0 + \tfrac{1}{8}c_1 = 0 \Rightarrow c_0 = \tfrac{1}{4}(-c_1 - c_{-1}).\end{aligned}$$

  Aus

$$\sum_{j\in\mathbb{Z}} c_j(x-j) = 0$$

  folgt also, für $x \in \left\{0, \frac{1}{2}, 1\right\}$

$$-c_{-1} = c_0 = -c_1 \qquad (4.3.16)$$

  und

$$c_0 = \tfrac{1}{4}(-c_1 - c_{-1}). \qquad (4.3.17)$$

  Setzt man (4.3.16) in (4.3.17) ein, erhält man

$$c_0 = \tfrac{1}{4}(c_0 + c_0) = \tfrac{1}{2}c_0. \qquad (4.3.18)$$

  (4.3.18) wird nur von $c_0 = 0$ erfüllt und gemeinsam mit (4.3.16) folgt

$$c_{-1} = c_0 = c_1 = 0.$$

$\boxed{\Leftarrow}$ Es gelte $c_{-1} = c_0 = c_1 = 0$.
Für $x \in [0,1]$, weil ϕ, laut (4.3.13), auf den Intervallen $(-\infty, -1]$ und $[2, \infty)$ gleich 0 ist, folgt

$$\sum_{j\in\mathbb{Z}} c_j \phi(x-j)$$

$$= \cdots + c_{-2}\phi(x+2) + c_{-1}\phi(x+1) + c_0\phi(x) + c_1\phi(x-1) + c_2\phi(x-2) + \ldots$$

$$= \cdots + c_{-2}\phi(x+2) + 0\cdot\phi(x+1) + 0\cdot\phi(x) + 0\cdot\phi(x-1) + c_2\phi(x-2) + \ldots$$

$$= \cdots + c_{-2}\cdot 0 + 0\cdot\phi(x+1) + 0\cdot\phi(x) + 0\cdot\phi(x-1) + c_2\cdot 0 + \cdots = 0.$$

□

- *Die Summenregel (4.3.2) wird erfüllt, denn*

$$\begin{aligned} \sum_{j\in\mathbb{Z}} p_{2j} &= p_0 + p_2 = \tfrac{3}{4} + \tfrac{1}{4} = 1, \\ \sum_{j\in\mathbb{Z}} p_{2j-1} &= p_{-1} + p_1 = \tfrac{1}{4} + \tfrac{3}{4} = 1. \end{aligned}$$

- *Damit der Algorithmus angewendet werden kann, muss noch gezeigt werden, dass $P^*(z)$ keine symmetrischen Nullstellen in $\mathbb{C}^\times$ hat.*

$$\begin{aligned} P^*(z) &= \tfrac{1}{2}\sum p_j z^j = \tfrac{1}{2}\left(\tfrac{1}{4}z^{-1} + \tfrac{3}{4}z^0 + \tfrac{3}{4}z + \tfrac{1}{4}z^2\right) \\ &= \tfrac{1}{8}z^{-1}(1 + 3z + 3z^2 + z^3) = \tfrac{1}{8}z^{-1}(1+z)^3, \quad z \in \mathbb{C}^\times. \end{aligned}$$

Daraus folgt, dass $P^*(z)$ die dreifache Nullstelle $z = -1$ und somit keine symmetrische Nullstelle hat.

1.Fall: *Mit $\ell = 0$ liefert der Algorithmus 4.12*

$$\begin{aligned} g^*(z) &= \tfrac{1}{2}\left(\tfrac{1}{4} - \tfrac{3}{4}z - \tfrac{3}{4}z^2 - \tfrac{1}{4}z^3\right), \\ h^*(z) &= \tfrac{3}{2} - \tfrac{1}{2}z, \\ a^*(z) &= \tfrac{3}{2} - \tfrac{1}{2}z, \\ b^*(z) &= \tfrac{1}{8} - \tfrac{3}{8}z + \tfrac{3}{8}z^2 - \tfrac{1}{8}z^3, \\ Q^*(z) &= \tfrac{1}{2}\left(-3z^{-1} - 1\right), \quad z \in \mathbb{C}^\times. \end{aligned}$$

2.Fall: *Mit $\ell = 1$ liefert der Algorithmus 4.12*

$$\begin{aligned} g^*(z) &= \tfrac{1}{16}\left(1 + 4z + 6z^2 + 4z^3 + z^4\right), \\ h^*(z) &= -\tfrac{1}{2} + 2z - \tfrac{1}{2}z^2, \\ a^*(z) &= -\tfrac{1}{4}z^{-2} + \tfrac{3}{4}z^{-1} + \tfrac{3}{4} - \tfrac{1}{4}z, \\ b^*(z) &= \tfrac{1}{8}z^{-2} - \tfrac{3}{8}z^{-1} + \tfrac{3}{8} - \tfrac{1}{8}z^1, \\ Q^*(z) &= \tfrac{1}{2}\left(\tfrac{1}{2}z^{-1} + \tfrac{3}{2} - \tfrac{3}{2}z - \tfrac{1}{2}z^2\right), \quad z \in \mathbb{C}^\times. \end{aligned}$$

Die Zerlegungs- und Rekonstruktionsfolgen, welche der Algorithmus 4.12 für $\ell = 2$ und $\ell = 3$ liefert, können im Appendix in der Tabelle 6.3 nachgesehen werden.

Satz 4.15 (Eigenschaften von $a^*(z), b^*(z), P^*(z), Q^*(z)$ [3, S.343])**.**
Die Laurent-Polynome $a^*(z), b^*(z), P^*(z), Q^*(z) \in \Lambda[\mathbb{C}^\times]$ aus dem Algorithmus 4.12 erfüllen folgende Identitäten:

$$P^*(z)a^*(z) + P^*(-z)a^*(-z) = 1, \tag{4.3.19}$$

$$Q^*(z)a^*(z) + Q^*(-z)a^*(-z) = 0, \tag{4.3.20}$$

$$P^*(z)b^*(z) + P^*(-z)b^*(-z) = 0, \tag{4.3.21}$$

$$Q^*(z)b^*(z) + Q^*(-z)b^*(-z) = 1. \tag{4.3.22}$$

Um diese Identitäten nachweisen zu können, werden unter anderem folgende Resultate benötigt.
Lemma 4.16 folgt aus dem 1. Schritt des Algorithmus 4.12.

Lemma 4.16.
Seien $\ell \in \mathbb{N}_0$, $\mu \leq -1$. Dann gilt

$$\left(\frac{1+z}{2}\right)^\ell P^*(z) = z^\mu g^*(z), \quad z \in \mathbb{C}^\times.$$

Lemma 4.17.
Seien $\ell \in \mathbb{N}_0$, $\mu \leq -1$, $\nu \geq 1$ und $\sigma_{\mu,\nu,\ell}$ aus (4.3.7). Dann gilt

$$-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell} = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu.$$

Beweis:

1.*Fall* : μ gerade:

$-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu = -2\left\lfloor \frac{1}{2}(\nu+\ell) - \frac{1}{2}\mu\right\rfloor - \mu =$

$-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \mu - \mu = -2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor.$

Aus Definition (4.3.7) folgt, dass $\sigma_{\mu,\nu,\ell} = 0$ und damit

$-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell} = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu.$

2.*Fall* : μ ungerade, $\nu+\ell$ gerade:

Weil $\nu-\mu+\ell$ ungerade ist, gilt

$-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell) - \frac{1}{2}\right\rfloor - \mu =$

$= -2\left\lfloor \frac{1}{2}(v+\ell)\right\rfloor + 1 + \mu - \mu = -2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + 1.$

Aus Definition (4.3.7) folgt, dass $\sigma_{\mu,\nu,\ell} = +1$ und damit

$-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell} = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu.$

<u>3.*Fall*</u> : μ ungerade, $v+\ell$ ungerade:

Weil $\nu - \mu + \ell$ gerade ist, gilt

$-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell) + \frac{1}{2}\right\rfloor - \mu =$

$= -2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor - 1 + \mu - \mu = -2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor - 1.$

Aus Definition (4.3.7) folgt, dass $\sigma_{\mu,\nu,\ell} = -1$ und damit

$-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell} = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu.$ □

***Beweis von Satz 4.15**:*
Sei $z \in \mathbb{C}^\times$.

Um die Identität (4.3.19) nachzuweisen, beachte, dass die Definition 4.3.6 impliziert

$P^*(z)a^*(z) + P^*(-z)a^*(-z) =$
$P^*(z)z^{-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell}+1}(\frac{1+z}{2})^\ell h^*(z) + P^*(-z)(-z)^{-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,\ell}+1}(\frac{1-z}{2})^\ell h^*(-z).$

Mit Hilfe von Lemma 4.16 und Lemma 4.17 folgt
$P^*(z)a^*(z) + P^*(-z)a^*(-z) =$
$z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1} g^*(z)h^*(z) + (-1)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+l)\right\rfloor + 1} z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+l)\right\rfloor + 1} g^*(-z)h^*(-z).$

Durch das Herausheben von $\left(z^{-2\left\lfloor \frac{1}{2}(v-\mu+l)\right\rfloor + 1}\right)$ und die Anwendung der Identität (4.3.5) folgt

$z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1} g^*(z)h^*(z) + (-1)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+l)\right\rfloor + 1} z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+l)\right\rfloor + 1} g^*(-z)h^*(-z)$

$= z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1 + 2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - 1} = 1.$

Um die Identität (4.3.20) nachzuweisen, beachte, dass die Definitionen (4.3.6) und (4.3.9) implizieren

$$Q^*(z)a^*(z) + Q^*(-z)a^*(-z)$$

$$= (-1)^\mu z^\mu (\tfrac{1-z}{2})^\ell h^*(-z) z^{-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,v,1}+1} (\tfrac{1+z}{2})^\ell h^*(z) +$$
$$(-1)^\mu (-z)^\mu (\tfrac{1+z}{2})^\ell h^*(z)(-z)^{-2\left\lfloor \frac{1}{2}(\nu+\ell)\right\rfloor + \sigma_{\mu,\nu,1}+1} (\tfrac{1-z}{2})^\ell h^*(-z).$$

Durch Herausheben von $(-1)^\mu h^*(z)h^*(-z)(\frac{1+z}{2})^\ell(\frac{1-z}{2})^\ell$ und Anwendung von Lemma 4.17 folgt

$$Q^*(z)a^*(z) + Q^*(-z)a^*(-z)$$
$$= (-1)^\mu h^*(z)h^*(-z)(\tfrac{1+z}{2})^\ell(\tfrac{1-z}{2})^\ell \cdot \left[z^\mu z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu + 1} + (-z)^\mu (-z)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu + 1} \right].$$

Der Ausdruck in den eckigen Klammern lässt sich folgendermaßen vereinfachen.

$$\left[z^\mu z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu + 1} + (-z)^\mu (-z)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor - \mu + 1} \right]$$
$$= z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1} + (-z)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1}.$$

Weil der Exponent $-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1$ ungerade ist, gilt

$$z^{-2\left\lfloor \frac{1}{2}(v-\mu+\ell)\right\rfloor + 1} + (-z)^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor + 1} = 0;$$

und damit

$$Q^*(z)a^*(z) + Q^*(-z)a^*(-z) = 0.$$

Um die Identität (4.3.21) nachzuweisen, beachte, dass die Identität (4.3.8) impliziert

$$P^*(z)b^*(z) + P^*(-z)b^*(-z) =$$

$$= P^*(z)(-z)^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor + 1} P^*(-z) + P^*(-z) z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor + 1} P(z).$$

Durch Herausheben von $P^*(z)P^*(-z)$ ist leicht zu sehen, dass

$$P^*(z)P^*(-z)\left[(-z)^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor + 1} + z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor + 1} \right] = P^*(z)P^*(-z) \cdot 0$$
$$= 0.$$

Um die Identität (4.3.22) nachzuweisen, beachte, dass die Identitäten (4.3.8) und (4.3.9) implizieren

$$Q^*(z)b^*(z) + Q^*(-z)b^*(-z)$$

$$= -z^{-2\left\lfloor \frac{1}{2}(\mu+v+\ell)\right\rfloor+1} P^*(-z)(-1)^\mu z^\mu (\tfrac{1-z}{2})^\ell h^*(-z)$$
$$+ z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} P^*(z)(-1)^\mu (-z)^\mu (\tfrac{1+z}{2})^\ell h^*(z)$$

$$= z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot [P^*(-z)(-1)^\mu z^\mu (\tfrac{1-z}{2})^\ell h^*(-z)(-1)^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1}$$
$$+P^*(z)(-1)^\mu (-z)^\mu (\tfrac{1+z}{2})^\ell h^*(z)].$$

Weil $(-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1)$ ungerade ist und wegen Lemma 4.16 ist der obige Ausdruck gleich

$$z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot [(-z)^\mu g^*(-z)(-1)^\mu (-z)^\mu h^*(z) - z^\mu g^*(z)(-1)^\mu z^\mu h^*(-z)]$$

$$= z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot [g^*(z)(-z)^\mu (-z)^\mu h^*(z) - g^*(-z) z^\mu z^\mu h^*(z)].$$

Nach Einsetzen der Identität (4.3.5) folgt

$$z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot [g^*(z)(-z)^\mu (-z)^\mu h^*(z) - g^*(-z) z^\mu z^\mu h^*(z)]$$

$$= z^{2\mu} z^{-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot z^{2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor-1}$$

$$= z^{2\mu-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot z^{2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor-1}.$$

Weil μ ganzzahlig ist, gilt $2\mu - 2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor = -2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor$ und damit

$$z^{2\mu-2\left\lfloor \frac{1}{2}(\mu+\nu+\ell)\right\rfloor+1} \cdot z^{2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor-1}$$

$$= z^{-2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor+1} \cdot z^{2\left\lfloor \frac{1}{2}(\nu-\mu+\ell)\right\rfloor-1} = z^0 = 1. \qquad \square$$

Es wurde nun allgemein gezeigt, dass die Laurentpolynome, aus dem Algorithmus 4.12, sicher die Identitäten aus dem Satz 4.15 erfüllen. Dennoch sollten die Identitäten aus dem Satz 4.15 für konkrete $a^*(z), b^*(z), P^*(z), Q^*(z)$ nachgewiesen werden. Damit können etwaige Fehler, die bei der Berechnung der Laurentpolynome entstanden sein könnten, aufgedeckt werden. In [3, Tabelle 9.4.1] sind zum Beispiel fehlerhafte Laurentpolynome angegeben, welche die Identitäten nicht erfüllen. Nur wenn die Identitäten erfüllt werden, eignen sich die Laurentpolynome, um für die Algorithmen zur Zerlegung 4.5 und Rekonstruktion 4.7 verwendet zu werden.

Die folgenden zwei Beispiele stellen Teil zwei der Beispiele 4.13 und 4.14 dar. Dort werden die Identitäten für die Laurentpolynome aus den Beispielen 4.13 (für den Fall $\ell = 0$) und 4.14 (für den Fall $\ell = 1$) überprüft.

Beispiel 4.18 (Teil zwei des Beispiels 4.13)**.**

Seien $a^*(z), b^*(z), P^*(z), Q^*(z),\ z \in \mathbb{C}^\times$, *aus dem 1.Fall* ($\ell = 0$) *des Beispiels 4.13.*

Überprüfen der Identität (4.3.19) aus Satz 4.15:

$$\begin{aligned} P^*(z)a^*(z) + P^*(-z)a^*(-z) &= \left(\tfrac{1}{4}z^{-1} + \tfrac{1}{2} + \tfrac{1}{4}z\right) + \left(-\tfrac{1}{4}z^{-1} + \tfrac{1}{2} - \tfrac{1}{4}z\right) \\ &= 1. \end{aligned}$$

Überprüfen der Identität (4.3.20) aus Satz 4.15:

$$Q^*(z)a^*(z) + Q^*(-z)a^*(-z) = (-1z^{-1}) + (1z^{-1}) = 0.$$

Überprüfen der Identität (4.3.21) aus Satz 4.15:

$$\begin{aligned} &P^*(z)b^*(z) + P^*(-z)b^*(-z) \\ &= \left(\tfrac{1}{4} + \tfrac{1}{2} + \tfrac{1}{4}z\right)\left(\tfrac{1}{4} - \tfrac{1}{2}z + \tfrac{1}{4}z^2\right) + \left(-\tfrac{1}{4} + \tfrac{1}{2} - \tfrac{1}{4}z\right)\left(\tfrac{1}{4} + \tfrac{1}{2}z + \tfrac{1}{4}z^2\right) \\ &= \left(\tfrac{1}{16}z^{-1} - \tfrac{1}{8}z + \tfrac{1}{16}z^3\right) + \left(-\tfrac{1}{16}z^{-1} - \tfrac{1}{8}z - \tfrac{1}{16}z^3\right) \\ &= 0. \end{aligned}$$

Überprüfen der Identität (4.3.22) aus Satz 4.15:

$$\begin{aligned} Q^*(z)b^*(z) + Q^*(-z)b^*(-z) &= \left(-z^{-1}\right)\left(\tfrac{1}{4} - \tfrac{1}{2}z + \tfrac{1}{4}z^2\right) + \left(z^{-1}\right)\left(\tfrac{1}{4} + \tfrac{1}{2}z + \tfrac{1}{4}z^2\right) \\ &= 1. \end{aligned}$$

Somit ist für Beispiel 4.13 (für $\ell = 0$*) gezeigt, dass die erzeugten Laurentpolynome alle vier Identitäten aus dem Satz* 4.15 *erfüllen.*

Beispiel 4.19 (Teil zwei des Beispiels 4.14)**.**

Seien $a^*(z), b^*(z), P^*(z), Q^*(z),\ z \in \mathbb{C}^\times$, aus dem 2.Fall ($\ell = 1$) des Beispiels 4.14.

Überprüfen der Identität (4.3.19) aus Satz 4.15:

$$\begin{aligned}
&P^*(z)a^*(z) + P^*(-z)a^*(-z)\\
&= \left(\tfrac{1}{8}z^{-1} + \tfrac{3}{8} + \tfrac{3}{8}z + \tfrac{1}{8}z^2\right)\left(-\tfrac{1}{4}z^{-2} + \tfrac{3}{4}z^{-1} + \tfrac{3}{4} - \tfrac{1}{4}z\right)\\
&+ \left(-\tfrac{1}{8}z^{-1} + \tfrac{3}{8} - \tfrac{3}{8}z + \tfrac{1}{8}z^2\right)\left(-\tfrac{1}{4}z^{-2} - \tfrac{3}{4}z^{-1} + \tfrac{3}{4} + \tfrac{1}{4}z\right)\\
&= (-\tfrac{1}{32}z^3 + \tfrac{9}{32}z^{-1} + \tfrac{1}{2} + \tfrac{9}{32}z - \tfrac{1}{32}z^3))\\
&+ (\tfrac{1}{32}z^3 - \tfrac{9}{32}z^{-1} + \tfrac{1}{2} - \tfrac{9}{32}z + \tfrac{1}{32}z^3))\\
&= 1.
\end{aligned}$$

Überprüfen der Identität (4.3.20) aus Satz 4.15:

$$\begin{aligned}
&Q^*(z)a^*(z) + Q^*(-z)a^*(-z)\\
&= \left(\tfrac{1}{4}z^{-1} + \tfrac{3}{4} - \tfrac{3}{4}z - \tfrac{1}{4}z^2\right)\left(-\tfrac{1}{4}z^{-2} + \tfrac{3}{4}z^{-1} + \tfrac{3}{4} - \tfrac{3}{4}z\right)\\
&+ \left(-\tfrac{1}{4}z^{-1} + \tfrac{3}{4} + \tfrac{3}{4}z - \tfrac{1}{4}z^2\right)\left(-\tfrac{1}{4}z^{-2} - \tfrac{3}{4}z^{-1} + \tfrac{3}{4} + \tfrac{3}{4}z\right)\\
&= \left(-\tfrac{1}{16}z^3 + \tfrac{15}{16}z^{-1} - \tfrac{15}{16}z + \tfrac{1}{16}z^3\right)\\
&+ \left(\tfrac{1}{16}z^3 - \tfrac{15}{16}z^{-1} + \tfrac{15}{16}z - \tfrac{1}{16}z^3\right)\\
&= 0.
\end{aligned}$$

Überprüfen der Identität (4.3.21) aus Satz 4.15:

$$\begin{aligned}
&P^*(z)b^*(z) + P^*(-z)b^*(-z)\\
&= \left(\tfrac{1}{8}z^{-1} + \tfrac{3}{8} + \tfrac{3}{8}z + \tfrac{1}{8}z^2\right)\left(\tfrac{1}{8}z^{-2} - \tfrac{3}{8}z^{-1} + \tfrac{3}{8} - \tfrac{1}{8}z\right)\\
&+ \left(-\tfrac{1}{8}z^{-1} + \tfrac{3}{8} - \tfrac{3}{8}z + \tfrac{1}{8}z^2\right)\left(\tfrac{1}{8}z^{-2} + \tfrac{3}{8}z^{-1} + \tfrac{3}{8} + \tfrac{1}{8}z\right)\\
&= (\tfrac{1}{64}z^{-3} - \tfrac{3}{64}z^{-1} + \tfrac{3}{64}z - \tfrac{1}{64}z^3)\\
&+ (-\tfrac{1}{64}z^{-3} + \tfrac{3}{64}z^{-1} - \tfrac{3}{64}z + \tfrac{1}{64}z^3)\\
&= 0.
\end{aligned}$$

Überprüfen der Identität (4.3.22) aus Satz 4.15:

$$\begin{aligned}
&Q^*(z)b^*(z) + Q^*(-z)b^*(-z)\\
&= \left(\tfrac{1}{4}z^{-1} + \tfrac{3}{4} - \tfrac{3}{4}z - \tfrac{1}{4}z^2\right)\left(\tfrac{1}{8}z^{-2} - \tfrac{3}{8}z^{-1} + \tfrac{3}{8} - \tfrac{1}{8}z\right)\\
&+ \left(-\tfrac{1}{4}z^{-1} + \tfrac{3}{4} + \tfrac{3}{4}z - \tfrac{1}{4}z^2\right)\left(\tfrac{1}{8}z^{-2} + \tfrac{3}{8}z^{-1} + \tfrac{3}{8} + \tfrac{1}{8}z\right)\\
&= \left(\tfrac{1}{32}z^{-3} - \tfrac{9}{32}z^{-1} + \tfrac{1}{2} - \tfrac{9}{32}z + \tfrac{1}{32}z^3\right)\\
&+ \left(-\tfrac{1}{32}z^{-3} + \tfrac{9}{32}z^{-1} + \tfrac{1}{2} + \tfrac{9}{32}z - \tfrac{1}{32}z^3\right)\\
&= 1.
\end{aligned}$$

Somit ist für Beispiel 4.14 (für $\ell = 1$) gezeigt, dass die erzeugten Laurentpolynome alle vier Identitäten aus dem Satz 4.15 *erfüllen.*

Die Identitäten aus dem Satz 4.15 kann man äquivalent umschreiben. Diese äquivalente Form von (4.3.19) - (4.3.22) wird in dem Satz 4.20 verwendet und ist für den Beweis von Satz 4.2.3, über die Perfekte Rekonstruktion, wichtig.

Satz 4.20.
Die Laurentpolynome $a^*(z), b^*(z), P^*(z), Q^*(z) \in \Lambda[\mathbb{C}^\times]$ aus dem Algorithmus 4.12 erfüllen folgende Identitäten.

$$a^*(z)P^*(z) + b^*(z)Q^*(z) = 1, \tag{4.3.23}$$

$$a^*(-z)P^*(z) + b^*(-z)Q^*(z) = 0, \tag{4.3.24}$$

$$a^*(z)P^*(-z) + b^*(z)Q^*(-z) = 0, \tag{4.3.25}$$

$$a^*(-z)P^*(-z) + b^*(-z)Q^*(-z) = 1, \quad z \in \mathbb{C}^\times. \tag{4.3.26}$$

Um Satz (4.20) beweisen zu können, wird folgendes Standardresultat der Linearen Algebra verwendet:

Satz 4.21 ([4, Theorem 1.4])**.**
Seien $A, B \in \mathbb{C}^{n\times n}$. Dann gilt

$$AB = I \Leftrightarrow BA = I.$$

***Beweis**:*
$\boxed{\Rightarrow}$ Angenommen, es gilt $AB = I$, für $A, B \in \mathbb{C}^{n\times n}$.
Dann gilt auch $BABA = BIA = BA$ und $BABA - BA = 0$.
Herausheben von BA liefert $BA(BA - I) = 0$. Das bedeutet, dass entweder $BA = 0$ oder $BA - I = 0$.
Wenn $BA = 0$, ist auch $BAB = 0$. Weil $AB = I$, folgt $B = 0$. Das Produkt jeder Matrix mit der Nullmatrix ist gleich der Nullmatrix und daher ist $AB = I$ nicht möglich. Folglich muss $BA - I = 0$ gelten und somit $BA = I$.

$\boxed{\Leftarrow}$ analog. □

Beweis von Satz 4.20 *[3, S.345]:*
Für den Beweis wird der Satz 4.15 benötigt. Dieser kann folgendermaßen in der Matrixschreibweise notiert werden:

$$\begin{bmatrix} a^*(z) & a^*(-z) \\ b^*(z) & b^*(-z) \end{bmatrix} \begin{bmatrix} P^*(z) & Q^*(z) \\ P^*(-z) & Q^*(-z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad z \in \mathbb{C}^\times. \tag{4.3.27}$$

Laut Satz 4.21 ist die Identität (4.3.27) punktweise äquivalent zu

$$\begin{bmatrix} P^*(z) & Q^*(z) \\ P^*(-z) & Q^*(-z) \end{bmatrix} \begin{bmatrix} a^*(z) & a^*(-z) \\ b^*(z) & b^*(-z) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad z \in \mathbb{C}^\times. \tag{4.3.28}$$

□

5 Zusammenfassung

Ziel der Arbeit war es, die mathematischen Grundlagen der Signalverarbeitung so einfach und anschaulich wie möglich zu vermitteln. Es wurden zu allen Grundlagen **Implementierungen in Matlab** gezeigt.
Zu den Grundlagen gehören:

- Das mathematische Modell des **Laurentpolynoms**, welches nach der geeigneten Verschiebung (Multiplikation mit $z^{-\mu}$, $\mu = grad(Laurentpolynom)$) ein Polynom über $\mathbb{C}$ ist, bei dem auch negative Exponenten zugelassen sind. Signale werden aus praktischen Gründen als Koeffizientenfolgen von Laurentpolynomen betrachtet.

- Das **Upsampling**, welches zur Abtastratenerhöhung von Signalen eingesetzt wird.

- Das **Downsampling**, welches zur Abtastratenverminderung von Signalen eingesetzt wird.

- Die **diskrete Faltung**, welche ein Operator $*$ ist, der für ein endliches Signal a und eine endliche Folge b ein neues Signal $(a * b)$ liefert.

  Indem die Folge b passend gewählt wird, erhält man ein Signal $(a * b)$ mit gewünschten Eigenschaften.

Der Fokus der Abhandlung lag vor allem auf der Verarbeitung von Audiosignalen. Diese können mathematisch als diskrete Funktionen definiert werden, die numerische Werte des Signals in Abhängigkeit von Messzeitpunkten beschreiben.

Es wurden **Algorithmen zur Zerlegung und Rekonstruktion von Audiosignalen** vorgestellt, bei denen die oben genannten Grundlagen zur Anwendung kommen. Für die Algorithmen wurden zwei konkrete Anwendungsbereiche vorgestellt und beispielhaft mit realen Audiodaten illustriert:

- Die **Perfekte Rekonstruktion** von Audioaufnahmen:
  Zuerst wird die Audioaufnahme mit dem Zerlegungsalgorithmus komprimiert, indem Frequenzen abgespaltet werden. Danach wird die ursprüngliche Audioaufnahme mit dem Rekonstruktionsalgorithmus wieder Perfekt Rekonstruiert, indem die abgespalteten Frequenzen wieder dazuaddiert werden.

- Die **Entrauschung** von Audioaufnahmen, um unerwünschtes Rauschen aus Audioaufnahmen zu entfernen:
  Dazu werden zuerst, mit dem Zerlegungsalgorithmus, Signal-Frequenzen

abgespaltet und analysiert. Anschließend werden die unerwünschten Störfrequenzen verworfen und mit dem Rekonstruktionsalgorithmus nur begehrenswerte Frequenzen wieder angeheftet.

Außerdem wurde gezeigt, wie bestimmte **Zerlegungs- und Rekonstruktionsfolgen** konstruiert werden können, die für die oben beschriebenen Algorithmen benötigt werden.

Auch mit geringen Vorkenntnissen im Bereich der Hochschulmathematik können Hintergründe und Methoden der modernen Signalverarbeitung verstanden werden. Das Spannende ist, dass sich die Signalverarbeitung ständig weiterentwickelt und ein Forschungsgebiet darstellt, das am Puls der Zeit liegt.

6 Appendix

6.1 Tabelle der Koeffizientenfolgen von $h^*(z)$

Die Koeffizientenfolgen von $h^*(z)$ hängen von ν_g von

$$g^*(z) = \sum_{j=\mu}^{\nu_g} g_j z^j, \quad \{g_j\}_{\mu}^{\nu_g} \in \ell_0(\mathbb{Z}),\ z \in \mathbb{C}^\times,$$

aus Algorithmus 4.12 ab.

Tabelle 6.1 (Koeffizienten von $h^*(z)$ aus Algorithmus 4.12 [3, S.288])**.**

ν_g	*Koeffizientenfolgen* $\{h_k\}_0$ *von* $h^*(z)$
2	$\{1\}$
3	$\{\frac{3}{2}, -\frac{1}{2}\}$
4	$\{-\frac{1}{2}, 2, -\frac{1}{2}\}$
5	$\{-\frac{5}{8}, \frac{25}{8}, -\frac{15}{8}, \frac{3}{8}\}$
6	$\{\frac{3}{8}, -\frac{18}{8}, \frac{38}{8}, -\frac{18}{8}, \frac{3}{8}\}$
7	$\{\frac{7}{16}, -\frac{49}{16}, \frac{126}{16}, -\frac{98}{16}, \frac{35}{16}, -\frac{5}{16}\}$
8	$\{-\frac{5}{16}, \frac{40}{16}, -\frac{131}{16}, 13, -\frac{131}{16}, \frac{40}{16}, -\frac{5}{16}\}$
9	$\{-\frac{45}{128}, \frac{405}{128}, -\frac{1521}{128}, \frac{2889}{128}, -\frac{2535}{128}, \frac{1215}{128}, -\frac{315}{128}, \frac{35}{128}\}$
10	$\{\frac{35}{128}, -\frac{350}{128}, \frac{1520}{128}, -\frac{3650}{128}, \frac{5018}{128}, -\frac{3650}{128}, \frac{1520}{128}, -\frac{350}{128}, \frac{35}{128}\}$

6.2 Tabellen der Koeffizientenfolgen von $a^*(z), b^*(z), P^*, Q^*(z)$

Tabelle 6.2 (Zerlegungs- und Rekonstruktionsfolgen [3, S.376]).
In der folgenden Tabelle sind die Zerlegungs- und Rekonstruktionsfolgen aus Algorithmus 4.12 angegeben, für die Hutfunktion aus Beispiel 4.13.

	$\{p_k\}_{-1}^{1} = \left\{\frac{1}{2}, 1, \frac{1}{2}\right\}$
$\ell = 0:$	$\{q_k\}_{-1}^{-1} = \{-2\}$ $\{a_k\}_{0}^{0} = \{1\}$ $\{b_k\}_{0}^{2} = \left\{\frac{1}{4}, -\frac{1}{2}, \frac{1}{4}\right\}$
$\ell = 1:$	$\{q_k\}_{-1}^{-1} = \left\{-\frac{3}{2}, 1, \frac{1}{2}\right\}$ $\{a_k\}_{0}^{2} = \left\{\frac{3}{4}, \frac{1}{2}, -\frac{1}{4}\right\}$ $\{b_k\}_{0}^{2} = \left\{\frac{1}{4}, -\frac{1}{2}, \frac{1}{4}\right\}$
$\ell = 2:$	$\{q_k\}_{-1}^{3} = \left\{\frac{1}{4}, \frac{1}{2}, -\frac{3}{2}, \frac{1}{2}, \frac{1}{4}\right\}$ $\{a_k\}_{-2}^{2} = \left\{-\frac{1}{8}, \frac{1}{4}, \frac{3}{4}, \frac{1}{4}, -\frac{1}{8}\right\}$ $\{b_k\}_{-2}^{0} = \left\{\frac{1}{4}, -\frac{1}{2}, \frac{1}{4}\right\}$
$\ell = 3:$	$\{q_k\}_{-1}^{5} = \left\{\frac{5}{32}, \frac{5}{16}, -\frac{45}{32}, \frac{7}{8}, \frac{11}{32}, -\frac{3}{16}, -\frac{3}{32}\right\}$ $\{a_k\}_{-2}^{4} = \left\{-\frac{5}{64}, \frac{5}{32}, \frac{45}{64}, \frac{7}{16}, -\frac{11}{64}, -\frac{3}{32}, \frac{3}{64}\right\}$ $\{b_k\}_{-2}^{0} = \left\{\frac{1}{4}, -\frac{1}{2}, \frac{1}{4}\right\}$

Tabelle 6.3 (Zerlegungs- und Rekonstruktionsfolgen [3, S.377]).
In der folgenden Tabelle sind die Zerlegungs- und Rekonstruktionsfolgen aus Algorithmus 4.12 angegeben, für das quadratische B-Spline aus Beispiel 4.14.

	$\{p_k\}_{-1}^{2} = \left\{\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}\right\}$
$\ell = 0:$	$\{q_k\}_{-1}^{0} = \{-3, -1\}$ $\{a_k\}_{0}^{1} = \left\{\frac{3}{2}, -\frac{1}{2}\right\}$ $\{b_k\}_{0}^{3} = \left\{\frac{1}{8}, -\frac{3}{8}, \frac{3}{8}, -\frac{1}{8}\right\}$
$\ell = 1:$	$\{q_k\}_{-1}^{2} = \left\{\frac{1}{2}, \frac{3}{2}, -\frac{3}{2}, -\frac{1}{2}\right\}$ $\{a_k\}_{-2}^{1} = \left\{\frac{3}{4}, \frac{1}{2}, -\frac{1}{4}\right\}$ $\{b_k\}_{-2}^{1} = \left\{\frac{1}{8}, -\frac{3}{8}, \frac{3}{8}, -\frac{1}{8}\right\}$
$\ell = 2:$	$\{q_k\}_{-1}^{4} = \left\{\frac{5}{16}, \frac{15}{16}, -\frac{15}{16}, -\frac{1}{8}, \frac{9}{8}, \frac{3}{16}\right\}$ $\{a_k\}_{-2}^{3} = \left\{-\frac{5}{32}, \frac{15}{32}, \frac{15}{6}, -\frac{1}{16}, -\frac{9}{16}, \frac{3}{32}\right\}$ $\{b_k\}_{-2}^{1} = \left\{\frac{1}{8}, -\frac{3}{8}, \frac{3}{8}, -\frac{1}{8}\right\}$
$\ell = 3:$	$\{q_k\}_{-1}^{6} = \left\{-\frac{3}{32}, -\frac{9}{32}, \frac{7}{32}, \frac{45}{32}, -\frac{45}{32}, -\frac{7}{32}, \frac{9}{32}, \frac{3}{32}\right\}$ $\{a_k\}_{-4}^{3} = \left\{\frac{3}{64}, -\frac{9}{64}, -\frac{7}{64}, \frac{45}{64}, \frac{45}{64}, -\frac{7}{64}, -\frac{9}{64}, \frac{3}{64}\right\}$ $\{b_k\}_{-4}^{-1} = \left\{\frac{1}{8}, -\frac{3}{8}, \frac{3}{8}, -\frac{1}{8}\right\}$

7 Quellenverzeichnis

7.1 Literatur

[1] A. Oppenheim, R. Schafer: *Zeitdiskrete Signalverarbeitung*. München,Oldenbourg, 1999.

[2] T. Sauer: *Digitale Signalverarbeitung*. Vorlesung, Lehrstuhl für Numerische Mathematik, Justus-Liebig-Universität, Gießen, 2003.

[3] C. Chui, J. D. Villiers: *Wavelet Subdivision Methods: GEMS for rendering curves and surfaces*. London, CRC Press, 2011.

[4] W.Ford: *Numerical Linear Algebra with Applications*. Oxford,Academic Press Elsevier, 2015.

[5] G.P. Nason: *Choice of the threshold parameter in wavelet function estimation*. Paper, Department of Mathematics, University of Bristol.

7.2 Abbildungen

Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.

Die Abbildungen 1 bis 6 wurden mit Matlab erstellt.
Die Abbildungen 7 und 8 wurden mit Hilfe der Internetseite
`http://www.wolframalpha.com/` erstellt.