Brandenburgische Technische Universität Cottbus - Senftenberg
Fakultät 3 - Maschinenbau, Elektrotechnik und Wirtschaftsingenieurwesen
Lehrstuhl für Kommunikationstechnik
Prof. Dr. Ing. habil. Matthias Wolff

Bachelorarbeit

Festkommaportierung des generalisierten Mel-Cepstrum

Arnulf Becker

Prüfer: Prof. Dr.-Ing. habil. Matthias Wolff, BTU, Cottbus
Betreuer: Dr.-Ing. Frank Duckhorn, Fraunhofer IKTS-MD, Dresden

Cottbus, den 21. Oktober 2015

Aufgabenstellung

Im Rahmen der Arbeit soll das Analyseverfahren des generalisierten Mel-Cepstrums portiert werden. Dieses vereinigt folgende Methoden der Merkmalsanalyse in einem Verfahren: die LPC-Koeffizienten, die Mel-LPC-Koeffizienten, das Cepstrum, das Mel-Cepstrum sowie das generalisierte Cepstrum. Mit einer Festkommaimplementierung des Verfahrens würde eine sehr leistungsfähige Analysemethode zur akustischen Mustererkennung bereit stehen. Die Implementierung sollte einer anschließenden Portierung auf einen digitalen Signalprozessor nicht im Weg stehen. Ausgangspunkt der Arbeit stellt eine bestehende Gleitkommaimplementierung der Analysemethode des generalisierten Mel-Cepstrums in der Programmiersprache C dar. Diese wird zusammen mit Daten zur Verifikation des Verfahrens zur Verfügung gestellt. Folgende Teilaufgaben sollten bearbeitet und in der schriftlichen Arbeit erläutert werden:

- Literaturrecherche zur effizienten Implementierung des generalisierten Mel-Cepstrums
- Analyse und gegebenenfalls Optimierung des bestehenden Gleitkommaprogramms
- Schrittweise Festkommaportierung einzelner Abschnitte
- Begleitende und abschließende Verifikation des Verfahrens mit den zur Verfügung gestellten Daten

Abschließend sollte das portierte Verfahren in Form eines Festkomma-C-Programms übergeben werden.

Selbständigkeitserklärung

Der Verfasser erklärt, dass er die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Cottbus, den 21. Oktober 2015

Arnulf Becker

Vorwort

An dieser Stelle möchte ich allen Personen danken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Mein besonderer Dank gilt Dr.-Ing. Frank Duckhorn, der mit sehr viel Engagement, guten Ideen und unermüdlichem Einsatz meine Bachelorarbeit betreut hat.

Des Weiteren möchte ich mich bei Prof. Dr.-Ing. Matthias Wolff bedanken, der diese Arbeit erst ermöglicht hat und stets für Fragen ansprechbar war.

Großer Dank gebührt auch meiner Freundin und meiner Familie, die mir immer zur Seite gestanden haben und mich beim Korrekturlesen unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Das generalisierte Mel-Cepstrum (englisch Mel-Generalized Cepstrum (MGC)) ist ein Analyseverfahren der Signalverarbeitung, das zur Extraktion von Merkmalsvektoren aus Signalen dient. Diese Signale können unterschiedlichen Ursprungs sein. Ein mögliches Einsatzgebiet dieser Analyse ist die Verarbeitung von Ultraschallsignale, welche bei der Materialdiagnostik von Keramiken eingesetzt werden, um Inhomogenitäten und Fehler in diesen zu erkennen. Ein weiteres Gebiet ist die Spracherkennung und die Sprechererkennung. Hierbei sind Sprachsignale das Analyseobjekt aus denen mithilfe des MGC-Verfahrens entsprechende Merkmalsvektoren extrahiert werden. Diese dienen zum trainieren bestimmter Mustererkennungsalgorithmen.

Dabei ist das generalisierte Mel-Cepstrum bei weitem nicht das einzig eingesetzte Analyseverfahren. Jedoch ist sein Vorteil, dass es bereits fünf häufig verwendete Methoden in sich vereint. Namentlich sind dies die LPC-Koeffizienten, die Mel-LPC-Koeffizienten, das Cepstrum, das Mel-Cepstrum sowie das generalisierte Cepstrum (siehe auch Unterabschnitt 2.1.4). Die Kombination in einem Verfahren fasst die unterschiedlichen Eigenschaften der einzelnen Analysen zusammen und ermöglicht eine kontinuierliche Variation zwischen diesen. Je nach Anforderung im konkreten Anwendungsfall ist somit ein passendes Analyseverfahren verfügbar.

Die Berechnung der MGC-Koeffizienten erfolgt, wie viele andere komplexe Signalverarbeitungsprozesse heutzutage, meist auf digitalen Rechnern. Dafür ist der Algorithmus in einem Programm implementiert, welches je nach Rechnerarchitektur und anderen Randbedingungen wie zum Beispiel der Ausführungsgeschwindigkeit und Latenz einer individuellen Adaption bedarf. Je nach Programmiersprache und Zielplattform übernimmt das Compilerprogramm bereits viele dieser Umsetzungen. Jedoch benötigt es für besondere Anforderungen die Hilfe des Programmierers. Ein solcher Fall ist die Verwendung von Festkommazahlen, statt der sonst üblichen Gleitkommazahlen. Die Motivation dahinter ist, dass Festkommaprogramme auf einer größeren Bandbreite von Architekturen lauffähig sind und dabei häufig weniger Zeit benötigen.

Die in dieser Arbeit beschriebene Festkommaportierung des bereits bestehenden Gleitkommaalgorithmus zur Berechnung des generalisierten Mel-Cepstrums ebnet den Weg dafür, dieses mächtige Analysewerkzeug auch auf spezialisierten Plattformen wie DSPs und FPGAs, in womöglich echtzeitkritischen Anwendungsfällen, einzusetzen. Im Allgemeinen rückt damit ein sehr viel größeres Feld möglicher Prozessorarchitekturen, wie sie zum Beispiel auch in eingebetteten Systemen Verwendung finden, in den Einsatzbereich.

2 Grundlagen

Zu Beginn ist eine kurze theoretische Betrachtung der beiden Teilaspekte dieser Arbeit hilfreich, um die Hintergründe besser zu verstehen. Dafür ist dieses Kapitel in zwei Abschnitte unterteilt. Im ersten Teil werden zunächst alle Einzelkomponenten des generalisierten Mel-Cepstrums betrachtet um anschließend auf das kombinierte Analyseverfahren einzugehen. Im zweiten Teil findet eine Betrachtung von Gleit- bzw. Festkommazahlen und deren Arithmetik statt, die in den Hauptteil zur eigentlichen Festkommaportierung überleitet.

Generell steht dabei mehr das allgemeine Verständnis der Konzepte im Vordergrund als deren Herleitungen, wofür auf die entsprechende Fachliteratur verwiesen wird.

2.1 Analyseverfahren

2.1.1 Mel-Skala

Die Einheit Mel ist mit der psychoakustischen Größe der *Tonheit* verknüpft. Das bedeutet, dass sie auf Basis der menschlichen auditiven Wahrnehmung definiert ist. Nach dieser entspricht eine Verdoppelung der Tonheit einer doppelt so hohen Tonwahrnehmung bei Sinustönen. Aus dem Grund, dass Hörempfindung stark subjektiv und von vielen Faktoren, wie etwa der Hörerfahrung der Probanden, abhängig ist, existieren viele verschiedene Skalen für die Tonheit. Eine in der Sprachverarbeitung häufig verwendete Abbildungsfunktion von der Frequenzachse auf die Tonheitsachse ist die in Gleichung 2.1 definierte Skalentransformation. Ein Ton mit der Frequenz f von 1000 Hz entspricht demnach einer Tonheit z von 1000 mel [SVN37][PK08, S.95]. Zur Veranschaulichung ist diese Funktion in Abbildung 2.1 für den Frequenzausschnitt des menschlichen Hörbereichs von 20 Hz bis 20 kHz aufgetragen.

$$z(f) = 1127 \cdot ln(1 + f/700\,\mathrm{Hz}) \tag{2.1}$$

Motivation für die Verwendung der Tonheit bei der Verarbeitung von Audiosignalen ist, dass die Frequenzachse auf diese Weise einer dem Gehör angepasste Achse abgebildet werden kann. Damit ist es möglich, die für das menschliche Gehör relevanten Daten besser zu den für die digitale Weiterverarbeitung notwendigen Quantisierungsstufen zuzuordnen. Daraus resultieren subjektiv bessere Ausgangssignale bei gleichbleibenden Datenmengen.

In der Praxis wird, statt der aufwändig zu berechnenden Funktion in Gleichung 2.1, häufig eine Approximation der Mel-Skala eingesetzt. Dessen Übertragungsfunktion im

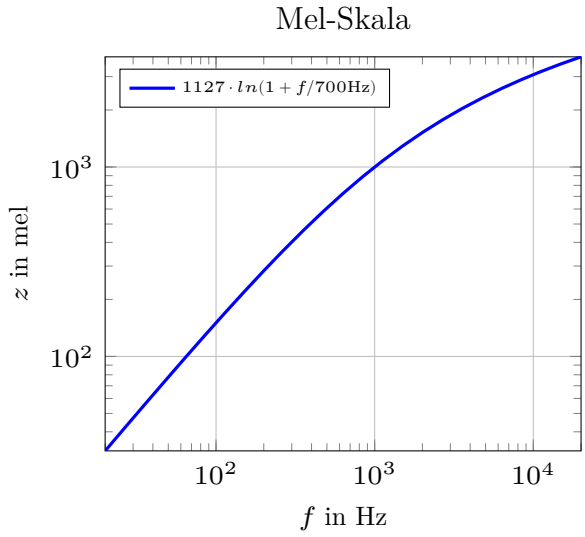

Abbildung 2.1 Zusammenhang zwischen Frequenz und wahrgenommener Tonhöhe.

z-Spektralbereich entspricht der in Gleichung 2.2 dargestellten Form, wobei $\tilde{X}(z)$ für das frequenztransformierte Spektrum steht [Str15, S.7] [Tok+94].

$$X(z) = \tilde{X}(\tilde{z}(z)), \quad \tilde{z}^{-1}(z) = \frac{z^{-1} - \lambda}{1 - \lambda z^{-1}} \qquad \left| \begin{array}{l} z = e^{j\omega} \\ \\ |\lambda| < 1. \end{array} \right. \tag{2.2}$$

Dabei wird der Parameter λ als *Verzerrungsfaktor* (*warping factor*) bezeichnet, da er die Verzerrung der Frequenzachse beschreibt. Ein Wert von $\lambda = 0$ steht für eine unverzerrte Frequenzachse, wohingegen für die beste Approximation der Mel-Skala kein konkreter Wert festgelegt werden kann. Er ist von der Abtastrate des Systems abhängig. Für die häufig verwendeten 16 kHz nähert beispielsweise ein Wert von $\lambda = 0.47$ die Mel-Skala am besten an [Str15, S.8].

2.1.2 Cepstrum

Das sogenannte Cepstrum ist eine Methode der Signalverarbeitung zur Merkmalsextraktion aus Signalen. Die Abbildung 2.2 zeigt eine mögliche Verarbeitungskette, um von einem Signal das Cepstrum zu erhalten.

In Worten entspricht das Cepstrum demnach der (inversen) FOURIER-Transformierten des logarithmierten Betragsspektrums. Die Logarithmierung bewirkt, dass Faltungsoperationen im Zeitbereich, also Multiplikationen im Frequenzbereich in Additionen im Cepstralbereich, welcher auch als *Quefrenzbereich* bezeichnet wird, überführt werden. In der Praxis ist dies von Vorteil, da additiv zusammengesetzte Komponenten im Allgemeinen leichter getrennt werden können. Da jede Anregung eines Übertragssystems, was zum Beispiel der Vokaltrakt des Menschen, ein Raum oder eine Keramikkomponen-

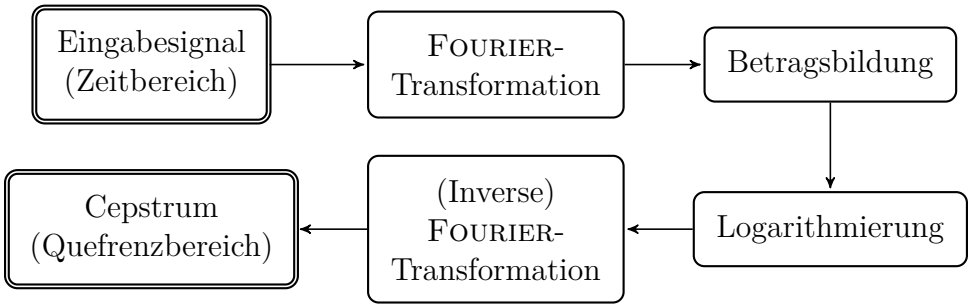

Abbildung 2.2 Schritte zur Erzeugung des Cepstrums aus einem Signal.

te sein kann, eine Faltungsoperation darstellt, ist die Relevanz dieser Analysemethode erkennbar, um das Anregungssignal von der Übertragungsfunktion des Systems trennen zu können [HW14, S.339f].

2.1.3 Linear Predictive Coding

Das Linear Predictive Coding (LPC) ist ein Analyseverfahren, welches hauptsächlich für Audiosignale eingesetzt wird. Grundlage hierfür ist, wie bei dem gerade besprochenen Cepstrum, das Quelle-Filter-Modell, welches versucht Signale auf eine Faltung von einem Anregungssignal (Quelle) mit einer Filterfunktion zurückzuführen.

In Gleichung 2.3 ist der Ansatz der linearen Prädiktion angegeben. Dabei wird der Abtastwert x zum Zeitpunkt n mit der Summe der mit den LPC-Koeffizienten a_k gewichteten K vorangegangenen Abtastwerten $x(n-k)$ abgeschätzt.

$$\hat{x}(n) = \sum_{k=1}^{K} a_k x(n-k) \tag{2.3}$$

Um die für die Signalverarbeitung interessanten Koeffizienten a_k zu erhalten, existiert eine Vielzahl von Berechnungswegen, deren Beschreibung in der Literatur zu finden ist [Str15, S.16-19] [HW14, S.363-371].

2.1.4 Generalisiertes Mel-Cepstrum

Wie bereits zuvor angedeutet, vereint das generalisierte Mel-Cepstrum insgesamt fünf Analyseverfahren. Zwischen diesen kann durch die beiden Parameter λ und γ ein kombiniertes Verfahren gewählt werden, das den gewünschten Eigenschaften der resultierenden Merkmale entspricht. In der Abbildung 2.3 sind die eingeschlossenen Analysemethoden und deren Abhängigkeit von der Wahl der Parameter dargestellt.

Der als Generalized Cepstrum (GC)-Faktor bezeichnete Parameter γ fließt direkt in die Übertragungsfunktion des generalisierten Mel-Cepstrums in Gleichung 2.4 ein [Str15, S.60][Tok+94]. Er stellt die Tendenz der Merkmale zwischen dem LPC und dem Cepstrum ein. Dabei entsprechen die MGC-Koeffizienten im Fall $\gamma = -1$ den (Mel-)LPC

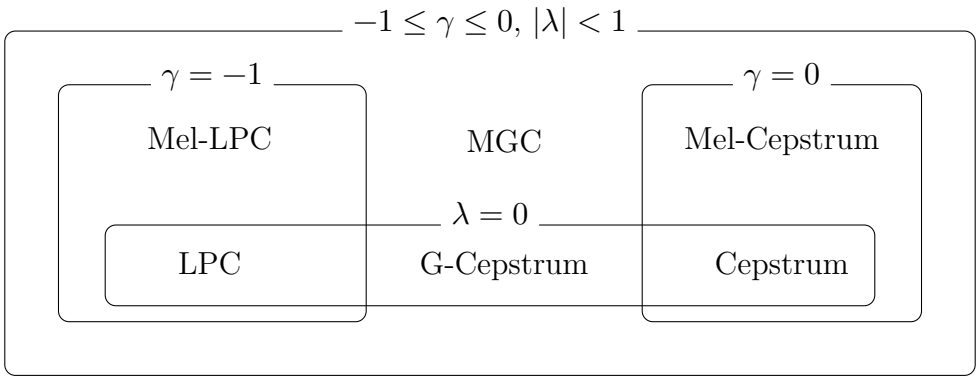

Abbildung 2.3 Die im generalisierten Mel-Cepstrum eingeschlossenen Analyseverfahren bei entsprechender Wahl der Parameter γ und λ [Str15, S.61].

Koeffizienten, im Fall $\gamma = 0$ dagegen den (Mel-)Cepstrum Koeffizienten. Der Parameter λ stellt die in Unterabschnitt 2.1.1 besprochene Skalentransformation zwischen unverzerrter Frequenzachse und Mel-Skala ein.

$$\tilde{H}_\gamma(\tilde{z}) = \left(1 + \gamma \sum_{k=0}^{K} \tilde{c}_{\gamma,k}\, \tilde{z}^{-k}\right)^{1/\gamma} \quad \left| \begin{array}{c} 0 < |\gamma| \leq 1 \\ \\ \tilde{z}^{-1}(z) = \dfrac{z^{-1} - \lambda}{1 - \lambda z^{-1}}. \end{array} \right. \tag{2.4}$$

2.2 Festkommaportierung

2.2.1 Darstellung von rationalen Zahlen im Computer

Die Darstellung von Zahlen erfolgt auf den heutzutage am weitest verbreiteten digitalen, binären Rechnern im Dualsystem. In diesem können Zahlenwerte unterschiedlich abgebildet werden.

Dabei sind die zwei am häufigsten verwendeten Repräsentationen für Zahlen, welche fast alle Programmiersprachen dem Anwender abstrahiert zur Verfügung stellen, die Ganzzahlen (englisch: *integer*) und die Gleitkommazahlen (englisch: *floating point number*).

Bei letzteren wird die Zahl normalisiert und Mantisse und Exponent separat, aber in einem Datenwort gespeichert. Operationen auf den Gleitkommazahlen müssen beide Komponenten getrennt berechnen [Gol91].

Die Ganzzahlen stellen einen Spezialfall der sogenannten *Festkommazahlen* dar. Bei diesen wird eine bestimmte Anzahl an niederwertigen Bits als Vorkommateil und die restlichen höheren Bits als Nachkommateil festgelegt. Dies bedeutet für die dargestellte Zahl, dass sich die Position des Kommas immer an einer bestimmten Stelle befindet, woraus sich auch die Bezeichnung ableitet. Ganze Zahlen besitzen keine Nachkommastellen, das Komma befindet sich also hinter der letzten Ziffer.

Die Werte, die eine Festkommazahl in Zweierkomplementdarstellung annehmen kann, sind in der Menge P enthalten, wie sie in Gleichung 2.5 definiert ist. Dabei entspricht N der Gesamtwortlänge und m den Nachkomma-Bits (vgl. [Yat13]).

$$P = \left\{ \frac{p}{2^m} \;\middle|\; -2^{N-1} \leq p \leq 2^{N-1} - 1,\ p \in \mathbb{Z} \right\} \tag{2.5}$$

Da die Position des Kommas nicht mit abgespeichert wird, ist dem Rechner der vom Programmierer angedachte Wert einer bestimmten Binärzahl nicht direkt zugänglich. Dennoch funktionieren mit ihnen die von der Recheneinheit bereitgestellten atomaren Operationen, wenn vom Programmierer die Position des Kommas in dem Programm berücksichtigt wird. Nach einigen Operationen wie z.B. der Multiplikation ist teilweise eine Neuberechnung der Kommaposition notwendig, da die Anzahl der Vor- und Nachkommastellen der jeweiligen Operanden addiert wird. Bei der Addition und Subtraktion müssen die Summanden das gleiche Festkommaformat besitzen, um korrekte Ergebnisse zu erhalten.

2.2.2 Vor- und Nachteile von Gleitkomma- bzw. Festkommazahlen

Wie bereits in Unterabschnitt 2.2.1 angedeutet, müssen bei Gleitkommaoperationen Mantisse und Exponent getrennt berechnet werden. Daraus ergibt sich ein höherer Rechenaufwand. Viele der heutigen Prozessoren besitzen eine sogenannte Floating Point Unit (FPU), welche für Gleitkommaoperationen mit vergleichbarer Wortbreite ähnliche Ausführungzeiten wie für Operationen mit ganzen Zahlen benötigt. Dies wird meist durch die Parallelisierung von Berechnungen erreicht. Aber gerade in DSPs und anderen spezialisierten Prozessoren fehlt diese FPU in vielen Fällen, was für Gleitkommaoperationen deutlich höhere Rechenzeiten als für Festkommaoperationen ergibt.

Ein weiterer Vorteil der Festkommazahlen ist es, dass sie für eine bestimmte Darstellungsgenauigkeit eine geringere Wortbreite benötigen. Dieser Zusammenhang ist beispielhaft in Abbildung 2.4 verdeutlicht, in welcher die möglichen Zahlenwerte bei gleich großer Wortbreite von Fest- bzw. Gleitkommazahl veranschaulicht sind. Aus diesem Grund wird für Gleitkommazahlen meist eine höhere Wortbreite eingeräumt, was wiederum zu längeren Zugriffszeiten auf solche Zahlen im Speicher führt.

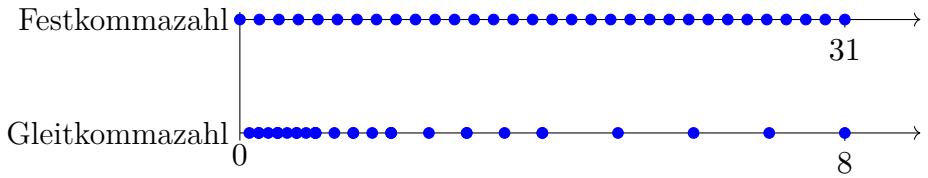

Abbildung 2.4 Vergleich der exakt abbildbaren Maschinenzahlen von vorzeichenlosen Gleit- und Festkommazahlen mit gleicher Wortlänge von 5 Bit. Bei der Gleitkommazahl entfallen davon 3 Bit auf die Mantisse und 2 Bit auf den Exponenten.

Ein entscheidender Nachteil von Festkommazahlen ist, dass deren Verwendung einen

deutlich höheren Aufwand für den Programmierer nach sich zieht. Zum einen müssen die Werte von Hand skaliert werden, um den vollen Wertebereich auszuschöpfen, wohingegen dies bei Gleitkommazahlen automatisch geschieht. Daher eignen sich letztere besser für die Darstellung von Signalen mit einem dynamischen Wertebereich, wie es zum Beispiel bei Sprachsignalen der Fall ist.

Zusätzlich muss bei Festkommazahlen die Position des Kommas berücksichtigt und nach Operationen eventuell korrigiert werden (siehe Unterabschnitt 2.2.1). Daraus ergeben sich viele potentielle Fehlerquellen und ein erhöhter Zeitaufwand für die Implementierung. Einige dieser Fehler können durch die Wahl des Festkommaformats vermieden werden. Auf ein solches wird in dem folgenden Unterabschnitt 2.2.3 eingegangen.

2.2.3 Festkommaarithmetik mit fraktionalen Zahlen

Als fraktionale Zahlen bezeichnet man einen Spezialfall der Festkommazahlen. Bei diesen steht nur ein Bit für die Vorkommastelle zur Verfügung. Dies impliziert nach Gleichung 2.5, dass deren Betrag stets kleiner gleich eins ist. In Tabelle 2.1 sind beispielhaft die Werte, die eine solche Zahl mit einer Wortbreite von drei Bit annehmen kann, abgedruckt.

Bitfolge	Dezimalwert
100	-1
101	-0.75
110	-0.5
111	-0.25
000	0
001	0.25
010	0.5
011	0.75

Tabelle 2.1 Fraktionale 3-Bit Zahl in Zweierkomplementdarstellung.

Die Wortbreite der verwendeten fraktionalen Zahlen ist an die geforderte Genauigkeit anpassbar, solange Operationen für das gewünschte Format der Operanden existieren bzw. vom Programmierer bereitgestellt werden. In der Praxis ist die Verwendung von 16 oder 32 Bit verbreitet.

Fraktionale Zahlen eignen sich zur Darstellung von Signalen im Festkommabereich besonders gut, da bei der in der Signalverarbeitung häufig verwendeten Multiplikationen ein Überlauf ausgeschlossen ist. Somit entfällt diese Fehlerquelle. Auch alle anderen Operationen mit diesen Zahlen werden so implementiert, dass diese statt einem Überlauf in die *Sättigung* gehen, also den höchsten bzw. niedrigsten Signalwert annehmen. Daraus ergibt sich ein kleinerer Fehler bei Überschreiten des zulässigen Wertebereiches.

Die konsequente Verwendung von fraktionalen Zahlen als alleinige Darstellungsform von Signalwerten vereinfacht zudem die Implementierung, da die Position des Kommas immer klar ist. Dies schließt weitere mögliche Fehler aus.

3 Ausgangspunkt und Portierung

In diesem Kapitel wird die Festkommaportierung des generalisierten Mel-Cepstrums genauer beleuchtet. Das Ziel ist ein Programm, dessen Operationen alle im Festkommabereich ausgeführt werden. Noch vorhandene Gleitkommaoperationen werden so gekapselt, dass sie bei Bedarf einfach mit Funktionsbibliotheken ausgetauscht werden können.

Zu Beginn wird das Programm kurz in den Kontext seiner Programmumgebung eingeordnet. Anschließend findet eine Analyse des bestehenden Programms und dessen Algorithmus statt, das weiter zur Beschreibung der Vorgehensweise mit einer Besprechung aller Schwierigkeiten und Lösungsstrategien führt. Dies leitet zum nächsten Kapitel über, in dem die Ergebnisse analysiert werden.

3.1 Der Algorithmus und das Programm

Das Programm zur Berechnung der Mel-Generalized Cepstrum (MGC)-Koeffizienten ist Teil von dLabPro[1], einer Skriptsprache, die besonders für Aufgaben der digitalen Signalverarbeitung und der Mustererkennung in akustischen Signalen entwickelt wurde. Die Sprache sowie alle Bibliotheken sind in der Programmiersprache C implementiert.

Der Aufruf der Festkommaimplementierung des generalisierten Mel-Cepstrums erfolgt aus Sicht des Programmierers wie die Gleitkommaimplementierung. Der einzige Unterschied besteht darin, dass der übergebene Signalvektor aus ganzzahligen Daten besteht. In Listing 3.1 ist ein Aufruf beider Funktionen dargestellt.

Listing 3.1 Beispiel für Aufruf des MGC-Programms aus dLabPro.

```
### Variable definition ###
data idSigD;
data idSigI;
data idFeaD;
data idFeaI;

### Signal import ###
"test.wav" "wav" idSigD stdfile -import;
idSigD 32768 scale idSigD =;

### Signal framing ###
idSigD 400 160 frame 400 512 "blackman" FALSE window idSigD =;

### Convert signal to fixed point ###
:round(idSigD*32767): -type short idSigI -tconvert;
```

[1] https://github.com/matthias-wolff/dLabPro

```
16
17 ### Do MGC analysis in fixed and floating point ###
18 idSigD -0.5 0.47 24 mgcep idFeaD =;  # floating
19 idSigI -0.5 0.47 24 mgcep idFeaI =;  # fixed
20
21 quit;
```

Um Veränderungen an dem vorhandenen Programmcode durchzuführen, oder wie in diesem Fall, den verwendeten Datentyp und damit alle Operationen auf ein anderes Arithmetiksystem umzustellen, ist ein tiefgehendes Verständnis des zugrunde liegenden Algorithmus notwendig. Hierfür werden in den folgenden Unterkapiteln die Schritte des Algorithmus zur Berechnung der MGC-Koeffizienten genau analysiert.

3.1.1 Algorithmus und Programmablauf

Nach [Str15, S.6f] existiert eine Vielzahl von Wegen, um aus einem Eingabesignal die MGC-Koeffizienten zu erhalten. Dabei wird zwischen einer direkten Berechnung aus dem Signal und einer Transformation aus anderen Merkmalen unterschieden. Der in der gegebene Gleitkommaimplementierung gewählte Weg ist abstrahiert in Abbildung 3.1 dargestellt. Er geht über die Transformation aus den (Mel)-LPC-Koeffizienten. Dafür werden diese zunächst mittels der Latticemethode berechnet (vgl. [Str15, S.18f]). Die daran anschließende Transformation in die in MGC-Koeffizienten macht den Großteil der Implementierung aus.

Hierbei handelt es sich um einen iterativen Prozess zur Annäherung an ein relatives Minimum, weswegen sich dieser Teil in einer Schleife befindet. In der weiteren Arbeit wird diese auch als „Hauptschleife“ bezeichnet. Deren Abbruch erfolgt bei ausreichender Annäherung an das Minimum.

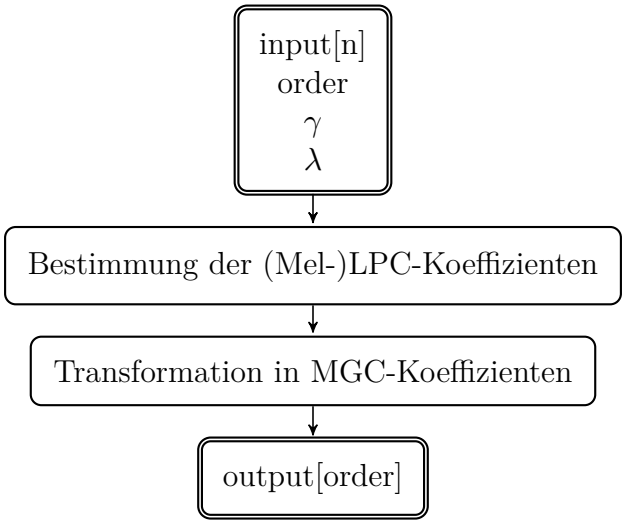

Abbildung 3.1 Skizzierter Ablauf des Algorithmus.

3.1.2 Datenfluss

Für das Verständnis des Algorithmus und der Zusammenhänge von Zwischenwerten ist die grafische Veranschaulichung dieser in einem Datenflussdiagramm besonders hilfreich. Dazu wurde das Diagramm in Abbildung 3.2 erstellt, welches alle Abhängigkeiten darstellt, dabei aber viele Operationen vereinfacht oder in jeweils einer zusammenfasst. Die Bezeichnung der Funktionsknoten orientiert sich dabei an den Funktionsnamen der ursprünglichen Implementierung oder an deren Operationen. Die Initialsierung bestimmter Strukturen ist nicht dargestellt. Deswegen sind gewisse Abhängigkeiten von den Eingangsparametern nicht direkt ersichtlich. Hinsichtlich der als „Mel Filter“ und „inv Mel Filter“ bezeichneten Knoten ist anzumerken, dass diese, abhängig vom Parameter λ, auch andere Filtercharakteristiken aufweisen können (siehe dazu Gleichung 2.2). Einfache mathematische Operationen sind kreisförmig eingefasst. Wenn Register oder Zwischenwerte in einer bestimmten Variable oder Vektor gespeichert werden, so sind die Kanten dementsprechend beschriftet. Die Hauptschleife, wie sie bereits in dem vorherigen Unterabschnitt 3.1.1 besprochen wurde, ist als gestrichelte Linie verdeutlicht. Ein- und Ausgabefelder sind gelb hinterlegt.

3.2 Vorgehensweise bei der Portierung

Die bereits vor der Arbeit existierende Implementierung zur Berechnung des generalisierten Mel-Cepstrums verwendet Gleitkommaarithmetik. Diese wurde iterativ auf Festkommaarithmetik umgestellt. In den folgenden Unterkapiteln werden die dafür gewählten Schritte, Strategien und die dabei auftretenden Schwierigkeiten besprochen.

3.2.1 Arbeitsablauf

Aus dem Grund, dass in der Literatur Empfehlungen zum Vorgehen bei Festkommaimplementierungen nur spärlich zu finden sind, wurden die aus dem Kenntnisstand resultierenden Strategien kontinuierlich überarbeitet und teilweise durch neue ersetzt. Die folgenden Erläuterungen beziehen die während der Arbeit gewonnenen Erkenntnisse mit ein und beschreiben somit den letztendlich erfolgreichen Ansatz.

Damit für die Zwischenwerte des Programms Referenzen verfügbar sind, bleibt die Gleitkommaimplementierung zunächst parallel bestehen. Daher werden für alle Gleitkommavariablen äquivalente Festkommavariablen angelegt und benötigter Speicher alloziert.

Wie bereits in Abschnitt 2.2 angedeutet, verwenden Gleitkommazahlen andere Operationen als die stattdessen eingesetzten fraktionalen Zahlen. In einem weiteren Schritt erfolgt daher die zeilenweise Umsetzung der entsprechenden Operationen in Funktionsaufrufe der fraktionalen Operationen. In Listing 3.2 ist dazu beispielhaft die Bildung des Betragsquadrates eines Vektors mit komplexen Komponenten abgedruckt. Auf Besonderheiten und häufige Fehler in diesem Schritt wird in Unterabschnitt 3.2.4 eingegangen.

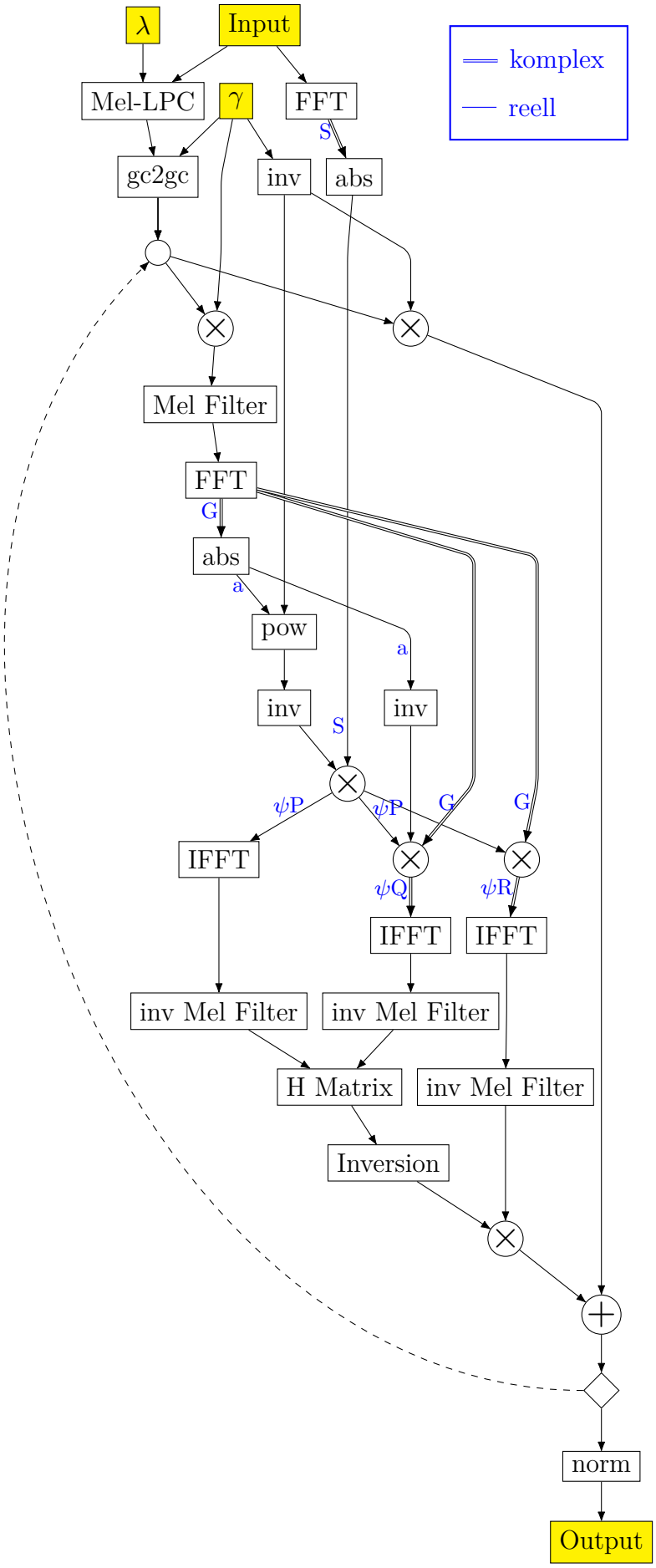

Abbildung 3.2 Vereinfachtes Datenflussdiagramm des Algorithmus.

Listing 3.2 Äqivalente Rechnungen in Gleit- und Festkommaarithmethik.

```
for (i = 0; i <= n / 2; i++) {
        /* Gleitkommaarithmetik */
        lpSx[i] = lpSx[i] * lpSx[i] + lpSy[i] * lpSy[i]; //<<
        /* Festkommaarithmetik */
        lpSxI32[i] = add32(mul16_32(lpSxI16[i], lpSxI16[i]),
                                    mul16_32(lpSyI16[i], lpSyI16[i]));
}
```

Für aufgerufene Unterprogramme mit Gleitkommazahlen als Parameter werden entsprechende Festkomma-Unterprogramme angelegt, welche zunächst nur eine Kapselung mit entsprechenden Konvertierungen darstellen. Ziel dieses Schrittes ist es, das Hauptprogramm komplett von Gleitkommaarithmetik zu befreien. Außerdem können so die Unterprogramme bei Bedarf ebenfalls portiert oder durch andere Programmbibliotheken ersetzt werden.

Sind diese Schritte abgeschlossen, so ist eine Anpassung der Signalwerte durch *Skalierungen* notwendig. Wegen ihrer zentralen Rolle bei der Portierung ist ihnen der folgende Abschnitt zugeteilt.

3.2.2 Skalierungen

Skalierungen sind Multiplikationen von Zahlenwerten mit konstanten Faktoren. Bei Festkommaprogrammen sind sie untrennbarer Bestandteil der Implementierung, da hier durch den stark beschränkten Wertebereich der Festkommazahlen die Signalwerte einer entsprechenden Anpassung bedürfen, um diesen voll auszuschöpfen. Dies ist notwendig, da es ansonsten zu Fehlern durch die Nichtbeachtung von Bits kommt. Dabei gibt es grundsätzlich zwei Fälle der Fehlanpassung der Signalwerte an den zur Verfügung stehenden Wertebereich. Der Fall in dem die Werte in die Sättigung geraten wird auch *Überanpassung* genannt. In dem entgegengesetzten Fall, in welchem die Signalwerte nur einen Teil des Wertevorrates nutzen, spricht man von einer *Unteranpassung*.

Wie im Datenflussdiagramm in Unterabschnitt 3.1.2 zu sehen ist, besteht eine starke Abhängigkeit zwischen den aufeinander folgenden Operationen. Ausgangspunkt der Anpassung der Skalierungen sind deswegen die Eingangswerte von denen eine iterativ zu den jeweils nächsten Operationen fortzuschreiten ist. Im Idealfall kann der gesamte Prozess demnach auf das in Abbildung 3.3 dargestellte Vorgehen reduziert werden. Die hier erwähnte Auswertungsfunktion wird ausführlich im Unterabschnitt 3.2.3 besprochen.

Dieses lineare Prozedere ist in der Praxis teilweise unzureichend, da bestimmte Werteverteilungen oder Programmkonstruktionen teilweise andere Wege erfordern. In den folgenden Unterabschnitte werden zwei dieser Spezialfälle behandelt.

Separate Skalierung von Koeffizienten

In der Implementierung kommen Operationen vor nach denen der Ausgabevektoren im erste Koeffizient stark von den übrigen abweicht. Im Extremfall ist er drei Größenordnungen außerhalb des Bereichs der anderen, wie sich in Abbildung 3.4 sehen lässt.

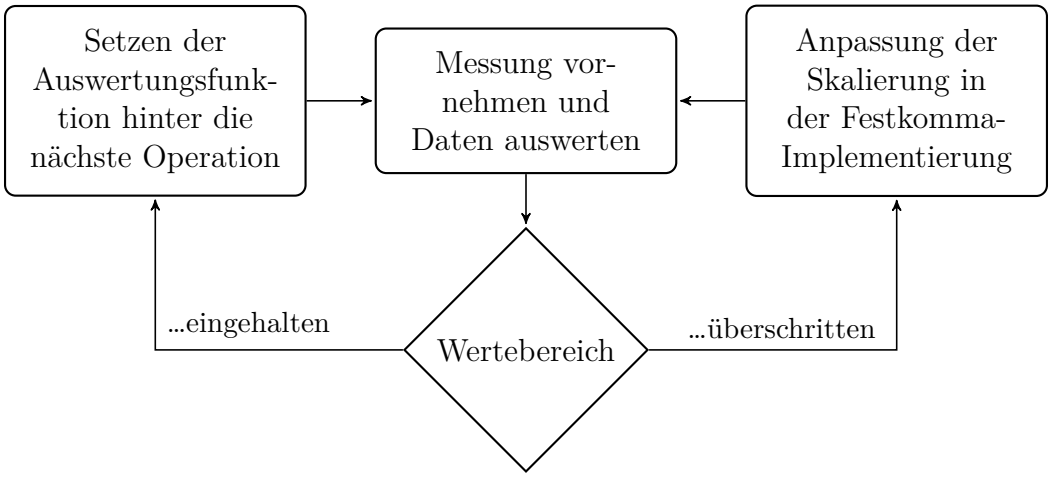

Abbildung 3.3 Vorgehensweise zur Einstellung der Skalierungen.

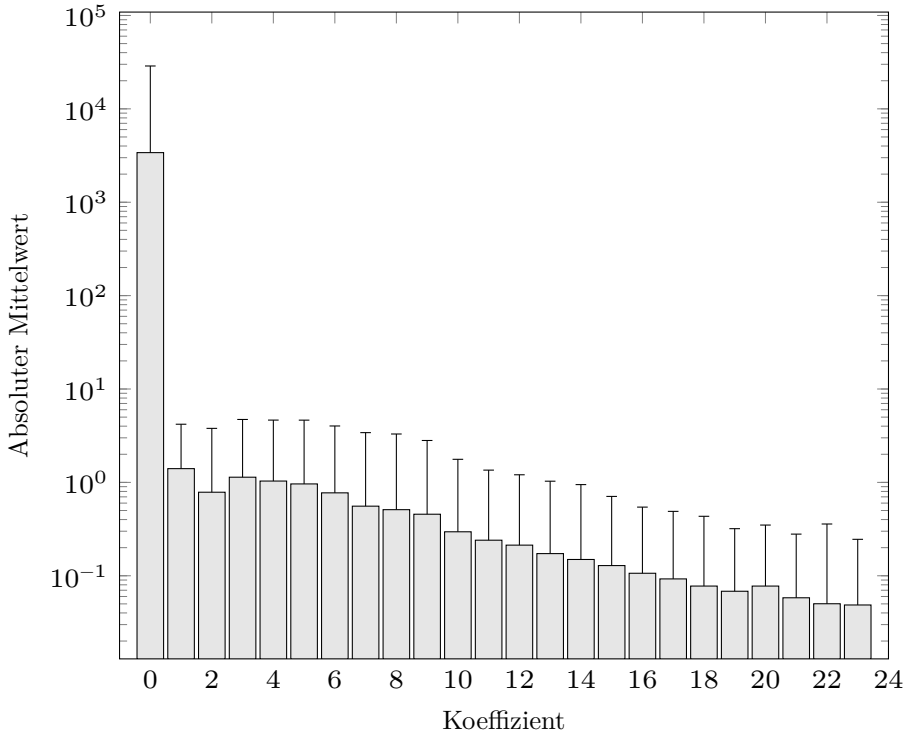

Abbildung 3.4 Werte der Koeffizienten nach LPC-Abschätzung und GC-Transformation in der Gleitkommaimplementierung. Sprachdatei: M_3. Anzahl Fenster = 400.

In solchen Fällen ist es nur schwer möglich einen gemeinsamen Skalierungsfaktor für alle Koeffizienten zu finden. Wird anhand des größten Wertes normiert, so können die anderen Koeffizienten nur die unteren Bits des Wortes nutzen und erhalten somit große Rundungsfehler. Im anderen Fall, in dem anhand des Mittelwertes aller Koeffizienten

normiert wird, verlieren die hoch ausgesteuerten viel Information, da sie in die Sättigung geraten. Gerade in dem vorliegenden Algorithmus stellt dies eine signifikante Fehlerquelle dar. Zumal hier dieser Wert in den nachfolgenden Operationen in die anderen Koeffizienten mit einfließt.

Aus diesem Grund ist hier ein anderer Ansatz notwendig, der als *gewichtete Skalierung* bezeichnet wird. Hierbei werden den einzelnen Komponenten des Vektors unterschiedliche Skalierungsfaktoren zugeordnet. Im einfachsten Fall sind dies zwei Faktoren. Dies sind in unserem Beispiel, nach Betrachtung von Abbildung 3.4, ausreichend, da hier nur zwei signifikante Werteniveaus zu erkennen sind. Die unterschiedlichen Skalierungen erfordern in nachfolgenden nichtlinearen Operationen zusätzliche Rechenschritte, um die korrekten Ergebnisse zu erhalten. Dies ist der Fall, wenn Koeffizienten mit verschiedenen Skalierungsfaktoren verknüpft werden. Hier ist vor Ausführung der Operation eine Kompensation der Skalierungsdifferenzen notwendig.

Skalierungen in Schleifen

Eingeführte Skalierungen wirken sich selbstverständlich auf die nachfolgenden Operationen aus, da sie immer auch einen Verstärkungsfaktor darstellen. Insbesondere in Schleifen, in welchen Werte von vorangehenden Durchläufen abhängen, hat das den Effekt, dass sich diese „aufschaukeln“ können. Um diesen Rückkopplungseffekt zu minimieren, muss entweder der Faktor sehr sorgfältig ausgewählt werden, was aber bei der großen Variation in den Eingangswerten oft nicht möglich ist, oder die Skalierung an geeigneter Stelle kompensiert werden.

In dem vorliegenden Programm betrifft dieser Effekt die Hauptschleife. Wie in dem Algorithmus beabsichtigt, werden die Werte bei jedem Durchlauf kleiner. Dadurch kommt es vor, dass nach einigen Iterationen die entsprechenden Variablen nicht mehr richtig skaliert sind.

Ein Lösungsansatz hierfür ist die Einführung einer von der Schleifeniteration abhängigen Skalierung. Sie ermöglicht den vollen Wertebereich für alle Operationen zu benutzen und erst bei Bedarf an geeigneter Stelle den eigentlichen Wert zu berechnen. Allerdings ist dieses Vorgehen mit zahlreichen Schwierigkeiten verbunden, da der dafür benötigte Skalierungsfaktor von den Werten einiger Variablen abhängt. Darüber hinaus kommen in der Hauptschleife zahlreiche nichtlineare Operationen vor, die die Umsetzung dieses Konzepts weiter erschweren. Aus diesen Gründen wurde zunächst von einer Implementierung dieser Technik abgesehen und stattdessen ein vorzeitiger Abbruch der Schleife in Kauf genommen. Wie in Abschnitt 4.2 in Tabelle 4.3 ermittelt, hat dies in der Praxis keine signifikante Auswirkung.

3.2.3 Datenauswertung

Zur Portierung eines Algorithmus ist die Betrachtung und Auswertung von Zwischenergebnissen ein wichtiger Schritt. Besonders für die in dem vorherigen Unterabschnitt 3.2.2 besprochenen Einstellung von Skalierungen ist sie wichtig. Dazu kann beispielsweise ein bereits vorhandenes Fehlersuchprogramm, meist als *Debugger* bezeichnet, verwendet

werden. Diese sind jedoch meist nicht dafür ausgelegt, große Datenmengen zu verarbeiten, welche jedoch bei diesem Algorithmus an vielen Stellen anfallen. Aus diesem Grund wurde ein Programm geschrieben, um Daten in eine separate CSV-Datei zu schreiben. Dieser sogenannte *Datensammler* (englisch *data logger*) stellt entsprechende Funktionen zur Verfügung, die an beliebige Stellen des Quelltextes eingefügt werden können. Von diesen Funktionen ist jeweils eine für jeden der drei im Programm verwendeten Datentypen, `FLOAT64`, `INT32` und `INT16`, implementiert. Des Weiteren stehen eine Initialisierungsfunktion (`data2csv_init`) und eine Funktion zur Wiederfreigabe (`data2csv_free`) zur Verfügung, welche vor bzw. nach der Benutzung aufgerufen werden müssen. In dem Fall des vorliegenden Programms geschieht dies in den bereits vorhandenen Funktionen `dlm_mgcepfix_init` und `dlm_mgcepfix_free`.

Listing 3.3 Beispielhafte Aufrufe des Datensammlers.

```
1 DataLog logger = { .file_path = "mgcepfix_log.csv" };
2 data2csv_init(&logger);
3 data2csv_FLOAT64(&logger, "FreqDomain PsiRy", lpPsiRy, n/2);
4 data2csv_INT16(&logger, "FreqDomain PsiRyI16", lpPsiRyI16, n/2);
5 data2csv_INT32(&logger, "tmp1I32", &tmp1I32, 1);
6 data2csv_free(&logger);
```

Wie in Listing 3.3 zu sehen ist, wird den `data2csv`-Funktionen neben dem Zeiger auf die initialisierte `DataLog`-Struktur noch eine frei wählbare Beschreibung, ein Zeiger auf die auszulesenden Daten und deren Anzahl übergeben.

Um einen möglichen Genauigkeitsverlust der Binärdaten zu vermeiden, werden sie in der CSV-Datei als Hexadezimalzahlen abgespeichert. Somit kann immer der exakte Registerinhalt reproduziert werden, ohne Fehler durch Rundung auf das Dezimalsystem zu erhalten (vgl. [Gol91]).

Die Speicherung der aufgezeichneten Daten in einer separaten Datei bringt diverse Vorteile mit sich. Zum einen ist der direkte Vergleich verschiedener Testdurchläufe möglich, zwischen denen auch ein größerer Zeitraum liegen kann. Zum anderen sind die Daten in dieser Form sehr portabel, d.h. sie können je nach Anforderung mit unterschiedlichen Programmen ausgewertet werden.

In Abbildung 3.5 ist das typische Anwendungsszenario des Datensammlers zu sehen. Die auf die CSV-Datei folgenden Verarbeitungsstufen werden in dem folgenden Unterabschnitt genauer betrachtet.

Skripte und Programme zur Datenverarbeitung

Um die benötigten Informationen aus der meist großen Datenmenge in der durch den Datensammler geschriebenen CSV-Datei ziehen zu können, ist eine Aufbereitung der Daten notwendig. Dabei ist es eine Grundanforderung, diese Stufe schnell und flexibel an neue Auswertungsaufgaben anpassen zu können. Außerdem soll sich die Anwendung nahtlos in den Arbeitsablauf und somit in die integrierte Entwicklungsumgebung einfügen. Aus diesen Gründen wurde die Programmiersprache Python[2] gewählt, da sie über

[2] https://www.python.org/

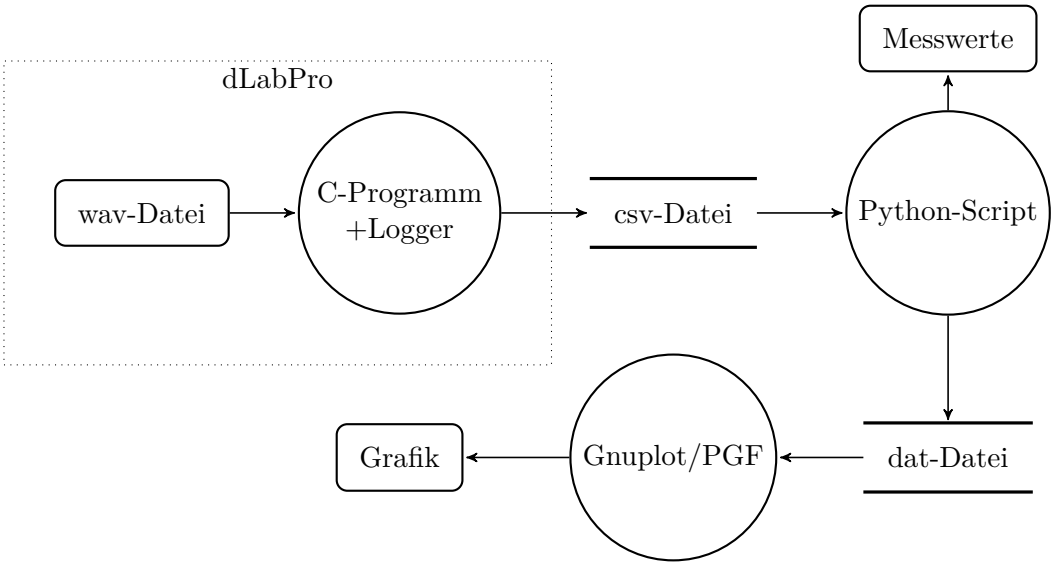

Abbildung 3.5 Datenfluss zur Evaluierung von Zwischenergebnissen.

viele Bibliotheken zur Datenverarbeitung verfügt und als kompakte Skriptsprache eine einfache Syntax besitzt.

Für die Verarbeitung der Daten müssen diese zunächst aus der CSV-Datei in die spracheigenen Datenstrukturen eingelesen werden. Hierfür wurde ein sogenannter *Parser* geschrieben, der diese dann in Form eines assoziativen Feldes (*Dictionary*) mit dem Datentyp und der frei wählbaren Bezeichnungen als Schlüssel ablegt. Alle Auswertungsfunktionen greifen auf diese Funktion zu, von denen die am häufigst verwendeten nun genauer betrachtet werden sollen.

Statistische Auswertung Hierbei erfolgt die für die Programmierung hilfreiche Ausgabe von Eckdaten einer Messreihe. Besonders der Maximal- und Minimalwert sind für die in Unterabschnitt 3.2.2 besprochene Skalierung von Werten unerlässlich. Die zusätzliche Ausgabe des Betragsmittelwertes ist hilfreich, um einen schnellen Überblick über die Werteverteilung zu erhalten.

Grafische Auswertung Um mehrere Merkmale einer große Datenmenge schnell zu erfassen, ist eine Aufbereitung der Daten zu grafischen Darstellungen unersetzlich. Für die in der digitalen Signalverarbeitung häufig angewendete Verarbeitung nach vorheriger Fensterung eignen sich besonders dreidimensionale Darstellungsformen, da hier mehrere Fenster mit deren Werten abbildbar sind.

Die programmtechnische Umsetzung sieht zuerst ein Python-Skript mit einer grafischen Benutzeroberfläche vor, wie sie beispielhaft in Abbildung 3.6 aufgezeigt ist. In dieser kann eine CSV-Datei mit den zuvor aufgezeichneten Daten ausgewählt werden. Nach dem Einlesen der Datei ist noch die zu verarbeitende Datengruppe auszuwählen. Es existiert jeweils eine Gruppe für jeden im Programm eingesetzten Datensammler

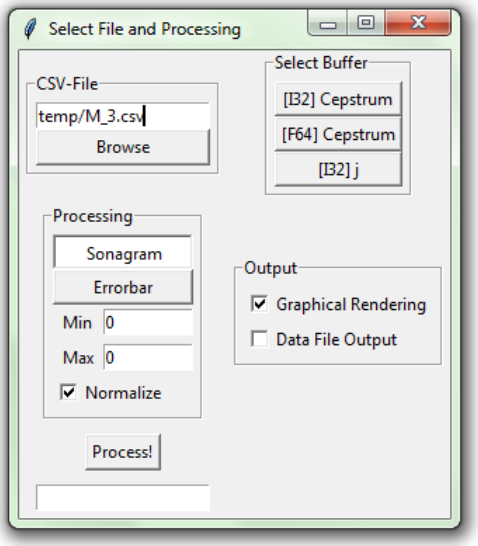

Abbildung 3.6 Benutzeroberfläche zur Datenverarbeitung.

(siehe dazu Unterabschnitt 3.2.3). Anschließend ist das gewünschte Ausgabeformat zu wählen. Hier ist der Wunsch für eine direkte grafische Ausgabe wählbar, welche mit den eigenen Bibliotheken der Programmiersprache eine Darstellung erzeugt. Ebenso kann die Ausgabe als Datei gewählt werden, wobei die Daten für die nachfolgenden Programme aufbereitet werden und anschließend in eine separate Textdatei mit der Endung `.dat` geschrieben werden. Um den Auswahlvorgang abzuschließen, muss das gewünschte Erscheinungsbild ausgewählt werden. Des Weiteren steht optional die Möglichkeit zur Verfügung nur einen gewissen Bereich des Fensters in die Bearbeitung einzubeziehen, um beispielsweise den ersten Koeffizienten auszuschließen. Dazu dienen die Eingabefelder mit der Beschriftung `Min` bzw. `Max`. Zusätzlich gibt es die Option die Daten auf den Wertebereich zwischen +1 und -1 zu normalisieren. Die Verarbeitung wird anschließend über die Schaltfläche mit der Aufschrift `Process!` gestartet. Wurde eine Ausgabe in eine Datei gewählt, erscheint anschließend der absolute Pfad zu dieser Datei in dem darunter liegenden Textfeld. Diese Dateien können anschließend z.B. mit Gnuplot[3] weiter verarbeitet werden.

3.2.4 Schwierigkeiten der Konvertierung

Bei der in Unterabschnitt 3.2.1 besprochenen Umsetzung von Gleitkommaberechnungen in deren äquivalente Festkommaberechnungen sind zahlreiche Details zu beachten. Im Folgenden sind einige dieser Besonderheiten, teils mit Beispielen aus dem Programm, aufgeführt.

[3] http://gnuplot.info/

Besonderheiten der Addition

Bei der Addition ist zu beachten, dass sich die Zahl der benötigten Bits um eins erhöht wird, dieses jedoch nicht in einem Wort mit gleicher Länge abgespeichert werden kann. Eine Auswirkung hat dies erst bei zwei Operanden, die beide das Bit mit dem höchsten Stellenwert verwenden, da statt dem zusätzlichen Bit das Resultat auf den zur Verfügung stehenden Maximalwert gesetzt wird. Eine mögliche Lösung ist es, beide Summanden vor der Addition um ein Bit nach rechts zu verschieben, um dieses zusätzliche Bit zur Verfügung zu stellen. Dabei geht natürlich die Information des niedrigsten Bits verloren. Dies ist jedoch weit weniger kritisch, als ein mögliche Sättigung des Wertebereiches, bei der Informationen von vielen Größenordnungen verschwinden würden. Das skalierte Ergebnis muss für den eigentlichen Wert der Addition um ein Bit nach links verschoben werden. Dies ist meist nicht direkt nach der Operation möglich, da ansonsten die vorherige Skalierung hinfällig werden würde. In dem hier aufgeführten Beispiel erfolgt die Rückskalierung nach der darauf folgenden Multiplikation:

Listing 3.4 Addition von Festkommazahlen.

```
1 /* Operation: lpPsiQy[i] *= (1. + gamma); */
2 lpPsiQyI16[i] = mul16(lpPsiQyI16[i],
3         add16(INT16_MAX >> 1, gammaI16 >> 1));
4 lpPsiQyI16[i] <<= 1; /* Rückskalierung */
```

Zwischenergebnisse

Ein zusätzliche Schwierigkeit ergibt sich aus der Tatsache, dass für Zwischenergebnisse von Berechnungen nur ein genau so großer Wertebereich wie für den Operanden der darauf folgenden Operation zur Verfügung steht. So genügt es zum Beispiel nicht, ein Ergebnis erst nach dem Abschluss der Operation zu skalieren, es müssen vielmehr bereits die Operanden entsprechend skaliert werden. Auch bei mehreren hintereinander ausgeführten Operationen müssen die Zwischenergebnisse immer evaluiert und entsprechend skaliert werden. Dies betrifft insbesondere die Verknüpfung von vielen Operationen, wie sie in Listing 3.5 aufgeführt ist. Hier erfolgt die Zerlegung in mehrere Teilschritte, um bei Bedarf die Zwischenwerte überprüfen zu können.

Listing 3.5 Zerlegung von verknüpften Operationen.

```
1 /* Operation: lpPsiQx[i]=(lpGx[i]*lpGx[i] - lpGy[i]*lpGy[i]) * tmp1/a */
2 INT32 intermRes = sub32(mul16_32(lpGxI16[i], lpGxI16[i]),
3                         mul16_32(lpGyI16[i], lpGyI16[i]));
4 intermRes = mul32(intermRes, tmp1I32);
5 lpPsiQxI32[i] = mul32(intermRes, aInvI32);
```

3.2.5 Nichtlineare Funktionen

Alle nichtlineare Funktionen stellen eine große Schwierigkeit bei der Portierung dar. Grund dafür ist, dass die Skalierungsfaktoren nach solchen Operationen nicht mehr

einfach aus dem Resultat heraus gerechnet werden können, falls an einer anderen Stelle eine umgekehrte Skalierung notwendig ist.

Zum Beispiel muss dazu bei Exponentialfunktionen die Wurzel berechnet werden. Besonders problematisch ist dies, weil meist sowohl Basis als auch Exponent Skalierungsfaktoren mitbringen und somit gleich zwei Wurzeloperationen notwendig sind, um das unskalierte Ergebnis zu erhalten.

$$(ax)^{by} = (a^y \cdot x^y)^b$$
$$ax^y = \sqrt[b]{(\sqrt[y]{a} \cdot x)^{by}}$$

In der Arbeit wurde dieses Problem gelöst, indem der Exponentialfunktion die Skalierungsfaktoren für die Eingabewerte der Mantisse und des Exponenten mitgeteilt wird. Weiterhin ist natürlich eine Skalierung der Ausgabewerte notwendig. So kann theoretisch immer das richtige Ergebnis berechnet werden. Für die Berechnung von Exponentialfunktionen in Festkommaarithmethik existieren viele Ansätze, wie zum Beispiel Nachschlagetabellen (*lookup table*), lineare Interpolation, Reihenentwicklungen oder daraus kombinierte Verfahren. Da diese Funktionen in der Arbeit durch Kapselung von bestehenden Gleitkommafunktionen realisiert wurden, soll im Folgenden jedoch nicht weiter auf diese eingegangen werden, da eine genauere Betrachtung den Rahmen sprengen würde.

3.2.6 Anpassung der Abbruchbedingung der Hauptschleife

In der ursprünglichen Implementierung wird zur Feststellung des Abbruchs der Hauptschleife die Veränderung des ersten Koeffizienten des momentanen Durchlaufs gegen den vorangegangenen verglichen. Unterschreitet diese einen sehr kleinen Wert, wird eine Markierung gesetzt, die daraufhin zur Beendigung der Schleife führt:

Listing 3.6 Ursprüngliche Abbruchbedingung.

```
1 /* ep = out[0] aus vorherigem Durchlauf */
2 if ((ep - out[0]) / out[0] < 0.00001) {
3         flag = 1; /* Abbruch */
4 }
```

Diese Überprüfung kann so direkt nicht in eine Festkommavariante überführt werden, da hier ein Inverses verwendet wird und die Werte alle sehr stark skaliert werden müssten. Jedoch lässt sich der Ausdruck unter Beachtung der Randbedingungen in einen äquivalenten Ausdruck umformen. Dies ist möglich, da der Wert `out[0]` aus einer Wurzeloperation hervorgeht und diese Bedingung somit einhält.

Listing 3.7 Abbruchbedingung der Festkommaimplementierung.

```
1 /* Randbedingung: out[0] >= 0 */
2 if (100000 * (ep - out[0]) < out[0]) {
3         flag = 1; /* Abbruch */
4 }
```

Hierbei tritt jedoch ein Problem mit den Wertebereichen auf. Die Bedingung überprüft gegen eine Zahl mit einer Genauigkeit von sechs Dezimalstellen, wobei die Operanden aus den vorhergehenden Operationen nur eine Wortbreite von 16 Bit mitbringen, die diese geforderte Präzision nicht abbilden können. Als eine mögliche Lösung stellt sich die Umstellung dieser auf 32 Bit heraus.

Dennoch bleibt das Problem bestehen, dass die vorherigen Operationen wie z.B. die FFTs oftmals nicht mit einer ausreichenden Genauigkeit berechnet werden. Zusammen mit der in Abschnitt 3.2.2 besprochenen unzureichenden Skalierung ergibt sich stets ein frühzeitiger Abbruch der Hauptschleife.

3.3 Einstellung der Skalierungen für veränderte Parameter

Wie bereits in Unterabschnitt 3.2.2 besprochen, benötigen Festkommaprogramme für geänderte Parameter im Regelfall neue Skalierungsfaktoren. Dies betrifft insbesondere die Anpassung an Eingangssignale, die von anderen Signalquellen stammen oder unter anderen Bedingungen entstanden sind. Um diesen Schritt schneller durchführen zu können, sind in dem Programm an den Stellen, an denen einstellbare Skalierungsfaktoren verwendet werden, Funktionen zur Datensammlung eingefügt, wie sie in Unterabschnitt 3.2.3 erläutert sind. Die Kennzeichnung der Daten entspricht dabei der Bezeichung des zugeordneten Skalierungsfaktors. Für einen Überblick über die Positionen und damit über die jeweiligen Abhängigkeiten dieser Faktoren sind diese in Abbildung 3.7 den ihnen vorangehenden Operationen zugeordnet.

Für die Auswertung der Daten können die in Abschnitt 3.2.3 aufgeführten Programme verwendet werden. Das Vorgehen ist ähnlich zu dem in Unterabschnitt 3.2.2 erläuterten iterativen Prozess. Dieser kann jedoch bei Signalquelle gleicher Natur häufig verkürzt werden. Im Idealfall genügt hier bereits eine Anpassung der Eingangsskalierung. Bei Änderungen der Parameter λ und γ ist der jeweilige Datenfluss zu verfolgen und die Skalierungen nach Überprüfung an den relevanten Stellen abzuändern.

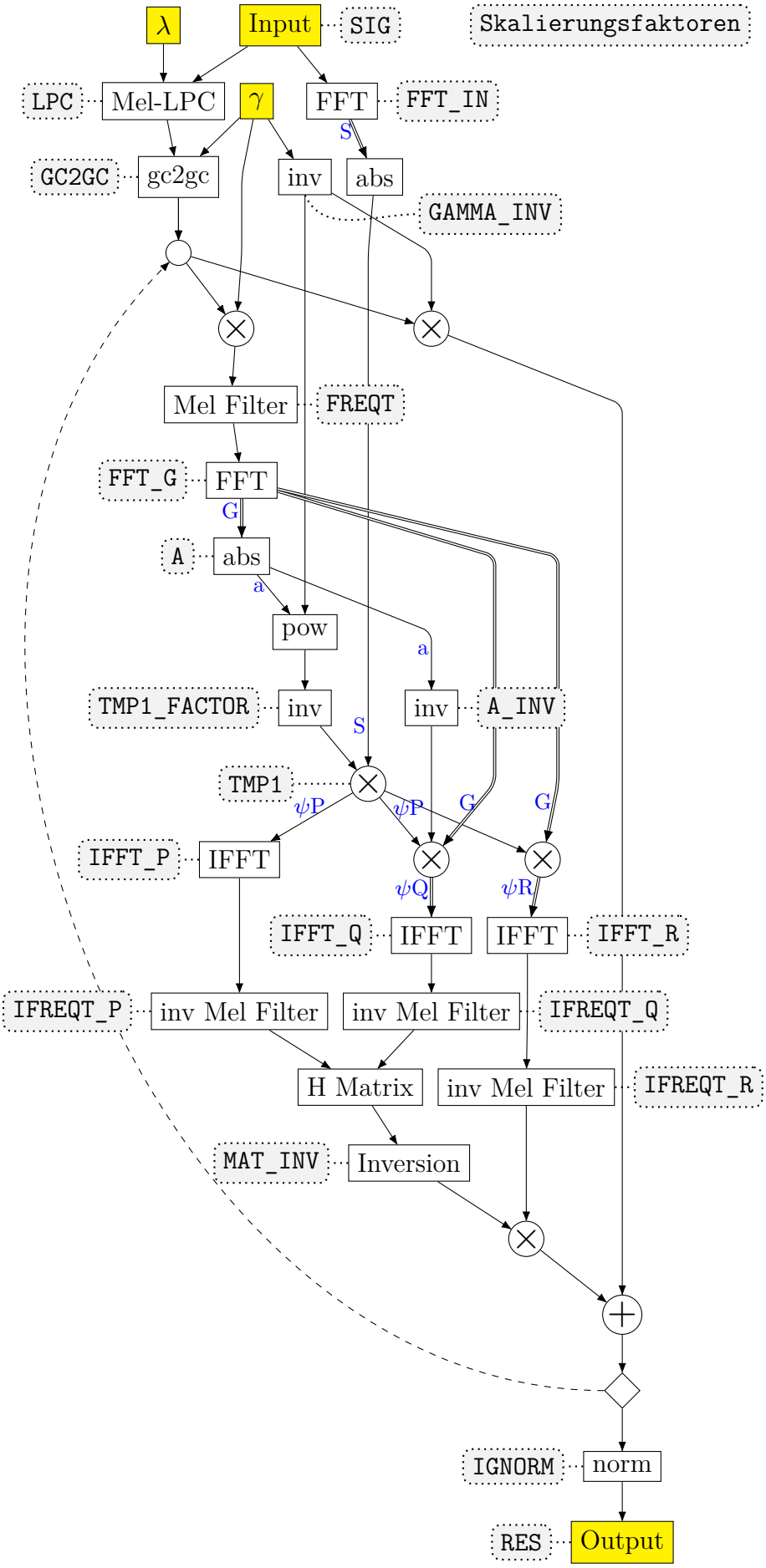

Abbildung 3.7 Vereinfachtes Datenflussdiagramm des Algorithmus mit den zugeordneten Skalierungsfaktoren.

4 Ergebnisse der Portierung

Eine umfassende Aussage über das Ausmaß des Fehlers in den mit dem portierten Verfahren berechneten Merkmalsvektoren ist, aufgrund der Komplexität des Programms, nur eingeschränkt möglich. Dennoch sind Aussagen für einzelne konkrete Beispiele möglich, aus denen sich dann grobe Tendenzen ableiten lassen.

Im ersten Abschnitt dieses Kapitels werden einige Merkmalsvektoren des ursprünglichen und des portierten Verfahrens verglichen. Wegen der geringen Aussagekraft dieser Vergleiche auf reelle Anwendungsszenarien, bleibt diese Untersuchungen in ihrem Umfang jedoch auf ein Beispiel beschränkt. Um die Qualität der berechneten MGC-Merkmale im praxisnahen Anwendungsfall zu überprüfen, werden diese im zweiten Abschnitt in einem Phonemerkennungsexperiment verwendet. Anhand der Auswertungen der daraus resultierenden Erkennerergebnissen werden Modifikation des Verfahrens zur Optimierung eingebracht.

4.1 Vergleich von Merkmalsvektoren

Wie in Unterabschnitt 3.2.3 besprochen, wurde für eine erste Beurteilung der Ergebnisse während der Portierung die Cepstren teils visuell ausgewertet. Beispielhaft sind solche für den Ausschnitt einer Sprachdatei in Abbildung 4.1 abgedruckt. Dabei bezieht sich die linke Darstellung auf das Ergebnis der ursprünglichen Gleitkommaimplementierung und die rechte Darstellung auf das des portierten Verfahrens. Die Dimension der Merkmalsvektoren ist dabei 24 und der Wert der Parameter beträgt für $\gamma = -0.1$ und für $\lambda = 0.47$.

Dabei wirkt das Cepstrum des portierten Verfahrens eher „abgeschliffen“ und die Dynamik der Koeffizienten weniger ausgeprägt. Zur Verdeutlichung der Abweichungen ist die Differenzbildung der Merkmalsvektoren eine geeignete Verarbeitungsmethode. Dies erfordert die Normierung der Merkmalsvektoren auf das jeweilige Maximum, um aussagekräftige Resultate zu erhalten. Die Abbildung 4.2 stellt diese Differenz für die in Abbildung 4.1 abgebildeten Vektoren dar.

Hier sind die Abweichungen und deren Ausprägung klar erkennbar. Besonders signifikant ist der Unterschied in den unteren Koeffizienten. Daraus lassen sich bereits Ansätze zur Optimierung ableiten, die versuchen diese Differenz zu minimieren.

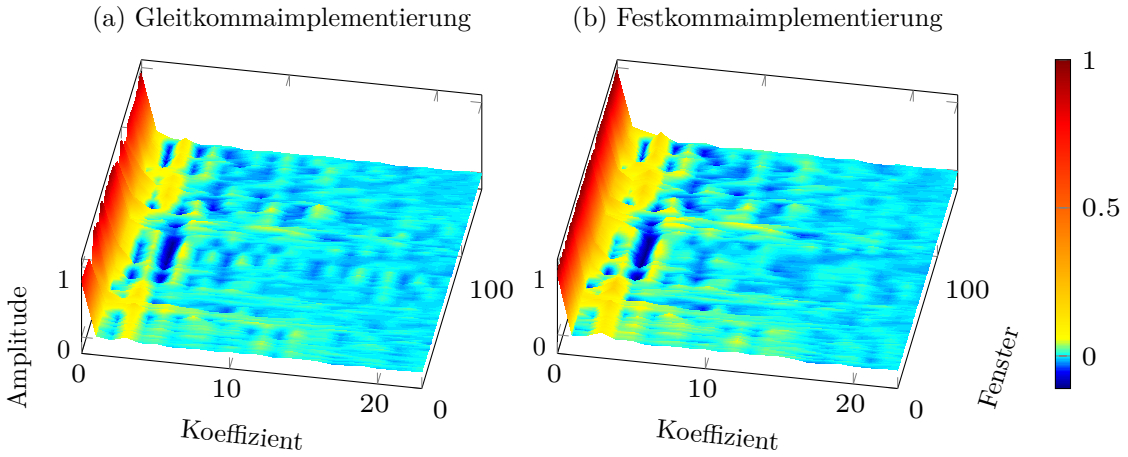

Abbildung 4.1 Vergleich einer Folge von Merkmalsvektoren vor und nach der Portierung (normiert). Sprachdatei: M_3. Ausschnitt: *„Ich habe auch an diesem Lehrstuhl… “*.

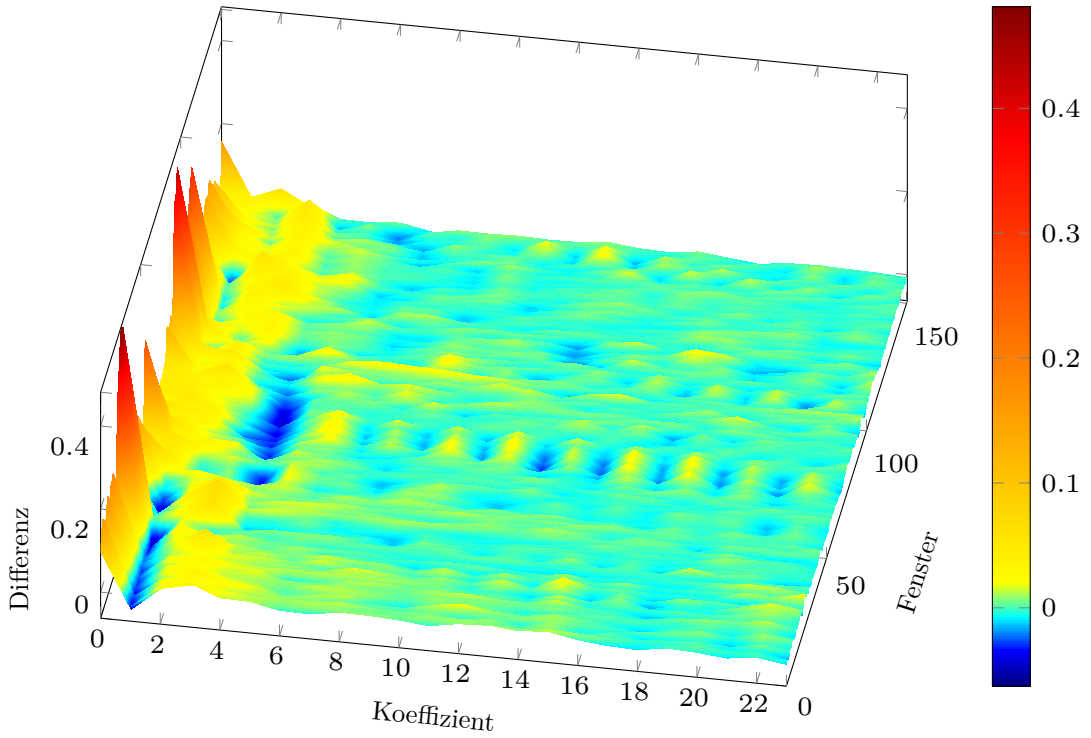

Abbildung 4.2 Differenz zwischen den normierten Merkmalsvektoren des portierten Verfahrens und der Gleitkommaimplementierung.

4.2 Erkennergebnisse

Eine Qualitätsbeurteilung der resultierenden Merkmalsvektoren anhand der Auswertung von reinen Zahlenwerten, wie sie im vorherigen Abschnitt erfolgt ist, besitzt für die praktische Anwendung der MGC-Koeffizienten nur eine beschränkte Aussagekraft. Wie bereits in der Einleitung angesprochen, finden aus Sprachsignalen gewonnene Merkmalsvektoren unter anderem bei der Spracherkennung Verwendung. Aus diesem Grund wurde ein Experiment zur Phonemerkennung aufgesetzt, aus dessen Ergebnissen wiederum Rückschlüsse auf die Qualität der verwendeten Merkmale möglich sind.

Um eine Referenz für die Erkennerergebnisse der Portierung zu haben, wurden Merkmale aus 1537 Sprachdateien[1] mit der ursprünglichen Implementierung berechnet. Bei diesen, und allen folgenden Experimenten, sind die Parameter und Einstellungen konstant. Dabei ist $\gamma = -0.1$, $\lambda = 0.47$ und die Anzahl der MGC-Koeffizienten gleich 24. Für das Training werden aber nur die ersten 15 Koeffizienten verwendet. Grund dafür sind Experimente[2], die gezeigt haben, dass die herkömmliche Anwendung der Hauptkomponentenanalyse auf alle Koeffizienten schlechtere Erkennerergebnisse liefert, als die Einbeziehung von sogenannten Deltas[3] in die Merkmalsvektoren. Um die Dimension dieser niedrig zu halten, werden nur die unteren Koeffizienten für die Erkennung verwendet, da diese für die Spracherkennung am wichtigsten sind.

Nach dem Training mit den Referenz-Merkmalen ergeben sich die aus Tabelle 4.1 zu entnehmenden Genauigkeiten der Phonemerkennung mit 95%-Konfidenzintervall. Zum Vergleich sind die Erkennerergebnisse einer einfachen Festkommaportierung abgedruckt. Bei dieser erfolgt die gesamte Berechnung mit Gleitkommazahlen, nur die Ein- und Ausgabe ist mit Festkommazahlen realisiert. Wie zu sehen ist, stellt dieser erste Schritt zum Festkommaprogramm nur eine kleine Beeinträchtigung der Genauigkeit der Phonemerkennung dar.

In den folgenden Durchläufen wurden nicht alle Modelle geprüft, da bereits nach den ersten fünf eine deutliche Tendenz für die Genauigkeit erkennbar ist. Eine vollständige Evaluation ist darüber hinaus mit einem sehr viel höheren Zeitaufwand verbunden.

Nach Abschluss der vollständigen Portierung des Hauptprogramms wurde ein erstes Training mit den damit berechneten MGC-Koeffizienten durchgeführt. Dieser erste Durchlauf lieferte die in Tabelle 4.2 ersichtlichen Ergebnisse.

Auffallend ist hier die sehr niedrige Genauigkeit. Die Gründe dafür sind vielfältig. Zum einen wurden die Normierungsfaktoren nicht explizit an die für das Training genutzten Audiodateien angepasst, sodass hier große Fehler durch Über- bzw. Unterschreiten des Wertebereichs entstehen. Zum anderen existierte hier noch ein Fehler in der Berechnung, der bewirkte, dass bestimmt Fenster stark von den ursprünglichen Koeffizienten abwichen. Vermutet wurde auch ein Einfluss der zu früh abbgebrochenen Fehlerminimierung der Hauptschleife (siehe dazu Unterabschnitt 3.2.6). Dieser Einflussfaktor konnte jedoch durch einen Test der Gleitkommaimplementierung mit modifizierter Abbruchbedingung nahezu ausgeschlossen werden, da die Erkennerergebnisse, wie aus Tabelle 4.3 ersichtlich,

[1] VMV Experiment der Verbmobil Datenbasis.
[2] Durchgeführt von Frank Duckhorn.
[3] Entspricht Ableitungen der Merkmale.

	Accuracy	
Modell	`VMV_mgcep.cfg`	`VMV_mgcepfix.cfg`
0_0	46.5 % ±2.4 %	40.8 % ±2.6 %
0_1	48.8 % ±2.5 %	47.5 % ±2.8 %
0_2	49.0 % ±2.6 %	48.3 % ±2.5 %
0_3	48.9 % ±2.4 %	48.3 % ±2.6 %
0_4	48.8 % ±2.3 %	48.7 % ±2.5 %
0_5	48.8 % ±2.4 %	48.9 % ±2.5 %
1_0	48.5 % ±2.5 %	48.4 % ±2.6 %
1_1	50.2 % ±3.0 %	50.7 % ±2.8 %
1_2	51.4 % ±3.1 %	51.9 % ±2.6 %
1_3	52.2 % ±3.0 %	52.1 % ±2.7 %
1_4	52.1 % ±2.8 %	51.8 % ±2.6 %
1_5	52.3 % ±2.8 %	51.5 % ±2.7 %
1_6	52.0 % ±2.8 %	51.2 % ±2.7 %
1_7	52.0 % ±2.8 %	51.3 % ±2.7 %
1_8	51.6 % ±2.7 %	50.8 % ±2.6 %
1_9	51.5 % ±2.9 %	50.8 % ±2.6 %
1_10	51.2 % ±2.9 %	50.8 % ±2.5 %

Tabelle 4.1 Vergleich der Erkennerergebnisse mit den Merkmalen des unmodifizierten Verfahrens mit denen einer einfachen Konvertierung, aber Berechnungen in Gleitkommaarithmetik.

	Accuracy
Modell	`VMV_mgcepfix.cfg`
0_0	15.3 % ±1.3 %
0_1	17.1 % ±1.3 %
0_2	16.2 % ±1.4 %
0_3	15.3 % ±1.3 %
0_4	17.6 % ±1.3 %
0_5	15.7 % ±1.2 %

Tabelle 4.2 Erste Erkennerergebnisse mit den Merkmalen des portierten Verfahrens.

	Accuracy	
Modell	`dd = 0.000001`	`dd = 0.1`
0_0	46.5 % ±2.4 %	46.3 % ±2.3 %
0_1	48.8 % ±2.5 %	48.6 % ±2.5 %
0_2	49.0 % ±2.6 %	48.9 % ±2.5 %
0_3	48.9 % ±2.4 %	48.7 % ±2.4 %
0_4	48.8 % ±2.3 %	48.8 % ±2.3 %
0_5	48.8 % ±2.4 %	49.0 % ±2.3 %

Tabelle 4.3 Erkennerergebnisse mit unterschiedlichen Abbruchbedingungen in der Gleitkommaimplementierung.

	Accuracy
Modell	`VMV_mgcepfix.cfg`
0_0	12.4 % ±2.9 %
0_1	19.5 % ±1.8 %
0_2	25.4 % ±1.7 %
0_3	25.6 % ±1.8 %
0_4	26.3 % ±1.8 %
0_5	25.5 % ±1.9 %

Tabelle 4.4 Erkennerergebnisse nach provisorischer Lösung des Fehlers mit dem ersten Koeffizienten.

selbst bei einem Wert von `dd = 0.1` nur eine marginale Abweichung von der Referenz zeigen.

Der Fehler der gänzlich abweichenden Fenstern wurde darauf zurückgeführt, dass in diesen der erste Koeffizient den Wert Null annahm. In der nachfolgende Normierung ergab sich daraus ein sehr großer Faktor für alle Koeffizienten, welcher zur vollen Aussteuerung führte. Beispielhaft ist eine solche Abfolge mit fehlerhaften Merkmalsvektoren in Abbildung 4.3 dargestellt.

Provisorisch wurde dieses Problem dadurch gelöst, dass nach Detektion eines solchen Falles der erste Koeffizient durch den des vorherigen Fensters ersetzt wurde. Nach dieser Modifikation ergaben sich die Tabelle 4.4 zu entnehmenden Ergebnisse.

Um den Einflussfaktor der ersten Koeffizienten ausschließen zu können, wurde nach der Merkmalextraktion der erste Koeffizient verworfen und das Training mit den übrigen 14 Koeffizienten durchgeführt. Dabei ergab sich die in Tabelle 4.5 abgedruckten Erkennerergebnisse. Wie zu sehen ist, verschlechterte sich im Vergleich zu dem vorherigen Experiment die Erkennungsrate deutlich. Dies bedeutet, dass der erste Koeffizient, trotz der im vorherigen Absatz besprochenen Maßnahme, immer noch viel Information trägt.

Die in der Analyse verwendeten Sprachdateien sind weit weniger ausgesteuert, als die

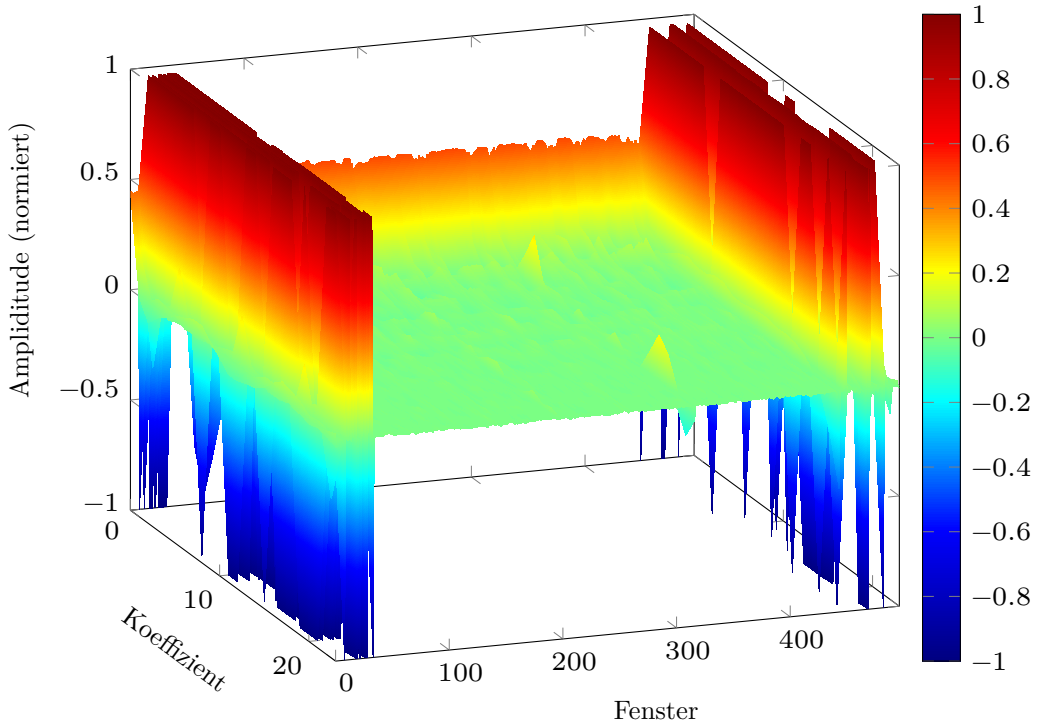

Abbildung 4.3 Auswirkung des Fehlers im ersten Koeffizienten auf die betroffenen Merkmalsvektoren. Sprachdatei: M_2.

	Accuracy	
Modell	`VMV_mgcep.cfg`	`VMV_mgcepfix.cfg`
0_0	37.4 % ±3.4 %	8.8 % ±2.4 %
0_1	44.2 % ±2.3 %	16.7 % ±1.6 %
0_2	45.1 % ±2.3 %	16.3 % ±1.5 %
0_3	44.7 % ±2.5 %	20.7 % ±1.4 %
0_4	44.8 % ±2.6 %	20.5 % ±1.5 %
0_5	44.8 % ±2.6 %	19.1 % ±1.6 %

Tabelle 4.5 Vergleich der Erkennerergebnisse nach Verwerfen der ersten Koeffizienten.

Modell	Accuracy VMV_mgcepfix.cfg
0_0	14.1 % ±3.4 %
0_1	20.7 % ±2.7 %
0_2	26.4 % ±2.1 %
0_3	25.7 % ±2.3 %
0_4	27.1 % ±2.4 %
0_5	27.5 % ±2.5 %

Tabelle 4.6 Erkennerergebnisse nach Eingangsskalierung.

für die Skalierung verwendeten Dateien. Hieraus ergeben sich zusätzliche Fehler. Aus diesem Grund wurde die Aussteuerung aller Dateien evaluiert, um einen festen Skalierungsfaktor für die Eingangssignale festlegen zu können. Nach deren Auswertung ergab sich eine mögliche Eingangsskalierung vom Faktor zwei. Ein erneuter Erkennerdurchlauf mit dieser Modifikation führte zu den in Tabelle 4.6 aufgeführten Ergebnissen.

An dieser Stelle wurden, aufgrund von mangelnder Zeit, keine weiteren Optimierungen hinsichtlich der Erkennerrate durchgeführt. Hier ist jedoch noch einiges an Potential vorhanden. Mögliche Verbesserungsvorschläge sind deswegen im folgenden Kapitel 5 aufgeführt.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurden das Vorgehen zur schrittweisen Portierung des Analyseverfahrens des generalisierten Mel-Cepstrums von einem Gleitkommaalgorithmus zu einem Festkommaalgorithmus untersucht, umgesetzt und die Ergebnisse ausgewertet. Dazu wurde eine Strategie zur Vorgehensweise entwickelt, die die iterative Umstellung des Programms ermöglicht. Zu der in diesem Rahmen notwendigen Evaluation der Werte in den Registern wurden Programme geschrieben, die einen einfachen Zugriff auf relevante Daten ermöglichen. Die während der Arbeit auftretenden Schwierigkeiten wurden aufgeführt und jeweilige Lösungsansätze besprochen.

Die Portierung des Hauptprogrammes wurde vollständig abgeschlossen und noch verwendete Gleitkommaoperationen in Unterfunktionen gekapselt, um deren Austausch durch andere Programmbibliotheken zu ermöglichen. Die mit dem portierten Verfahren berechneten Merkmalsvektoren wurden sowohl grafisch, als auch durch Phonemerkennungsexperimente auf ihre Qualität untersucht. Dabei stellten sich jedoch Defizite hinsichtlich der Erkennungsgenauigkeit heraus, die teilweise behoben werden konnten. Dennoch bleibt noch einiges an Potential für weitere Optimierungen und Verbesserungen.

Zum einen ist bereits eine Verbesserung des Eingabesignals durch bestimmte Vorverarbeitungsschritte hilfreich. Besonders sinnvoll ist eine Vorfilterung durch Hochpassfilter. Dadurch können Störgeräusche mit tiefen Frequenzen unterdrückt werden. Diese besitzen meist einen Großteil der Signalenergie und führen somit zu noch dynamischeren Signalen, ohne dabei Informationen des Sprachsignals zu tragen. Gerade für die Festkommaimplementierung ist dies ein entscheidender Nachteil. Des Weiteren ist eine individuelle Anpassung aller Skalierungen an Eingangssignale notwendig, die unter anderen Randbedingungen entstanden sind und somit einen anderen Frequenzgang und abweichende Amplitudenwerte aufweisen. Besonders kritisch sind hierbei die Eingangswerte der nichtlinearen Funktionen, die im Regelfall mit den Werten der Gleitkommaimplementierung abgeglichen werden müssen, um vergleichbare Ausgangswerte zu erhalten.

Abschließend kann festgehalten werden, dass die Festkommaportierung eines Programmes mit einer solchen Komplexität mit vielen Schwierigkeiten und einem hohen Zeitaufwand verbunden ist. Da das Programm oft auf anderen Plattformen laufen soll, ist weiterhin eine individuelle Anpassung an die jeweilige Architektur notwendig, um optimale Ergebnisse zu erhalten. Aus diesen Gründen existieren bereits einige Ansätze, um dieses Vorgehen zu automatisieren und durch Simulationsmodelle gleichzeitig an bestimmte Zielplattformen anzupassen [CSH04].

Literaturverzeichnis

[CSH04] Keith B. Cullen, Guenole C.M. Silvestre und Neil J. Hurley: „Simulation Tools for Fixed Point DSP Algorithms and Architectures“. In: *Proc. International Conference on Signal Processing*. 2004.

[Gol91] David Goldberg: *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. [Online; Stand 15. Juli 2015]. 1991. URL: http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html.

[HW14] Rüdiger Hoffmann und Matthias Wolff: *Intelligente Signalverarbeitung 1: Signalanalyse*. Springer-Verlag, 2014.

[PK08] Beat Pfister und Tobias Kaufmann: *Sprachverarbeitung: grundlagen und methoden der sprachsynthese und spracherkennung*. Springer-Verlag, 2008.

[Str15] Guntram Strecha: *Skalierbare akustische Synthese für konkatenative Sprachsynthesesysteme*. Studientexte zur Sprachkommunikation 77. Dresden: TUDpress, 2015.

[SVN37] Stanley Smith Stevens, John Volkmann und Edwin B. Newman: „A scale for the measurement of the psychological magnitude pitch“. In: *The Journal of the Acoustical Society of America* 8.3 (1937), S. 185–190.

[Tok+94] Keiichi Tokuda u. a.: „Mel-generalized cepstral analysis - a unified approach to speech spectral estimation“. In: *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*. Yokohama, 1994.

[Yat13] Randy Yates: *Fixed-Point Arithmetic: An Introduction*. [Online; Stand 3. Juli 2015]. 2013. URL: www.digitalsignallabs.com/fp.pdf.

Abkürzungen

CSV Comma Separated Value.

DSP Digitaler Signalprozessor.

FFT Fast Fourier Transform.

FPGA Field Programmable Gate Array.

FPU Floating Point Unit.

GC Generalized Cepstrum.

LPC Linear Predictive Coding.

MGC Mel-Generalized Cepstrum.