Richtwirkungsaufnahme von Instrumenten und (Echtzeit-)Inversion ihrer Schallleistungsspektrogramme mit und ohne Phasenvorgabe

Seminararbeit für Algorithmen in Akustik und Computermusik 2 SE

Rüdiger Fasching, Korbinian Wegler, Johanna Kerber

Betreuung: Franz Zotter, Mathias Frank

Graz, 4. August 2018

kunst uni graz

institut für elektronische musik und akustik

Zusammenfassung

Für das Seminar Algorithmen in Akustik und Computermusik 2 entstanden Aufnahmen von Richtwirkungen einiger Instrumente. In diesem Zusammenhang wurden die Algorithmen von Griffin und Lim [GL84] und der Phase Gradient Heap Integration [PBS17] zur interferenzfreien Signalzusammenführung in Matlab implementiert und getestet. Besonderes Augenmerk wurde dabei auf die verschiedenen Möglichkeiten gelegt, eine Initialphase für den GLA zu finden.

Inhaltsverzeichnis

1 Themenstellung

Thema dieser Seminararbeit ist die Richtwirkungsaufnahme von Instrumenten und (Echtzeit-) Inversion ihrer Schallleistungsspektrogramme - mit und ohne Phasenvorgabe. Um die Richtwirkung eines Instruments zu erfassen ist eine Messanordnung mit vielen Kanälen erforderlich. Am IEM gibt es z.B. eine Messanordnung mit 64 Mikrofonen auf einer Kugeloberfläche (Abbildung 1)[1]. Bei der Messung werden große Datenmengen generiert, was zu erheblichem Aufwand beim Weiterverarbeiten und Speichern führt. Zur Lösung dieses Problems wird in [Süs11] vorgeschlagen, ein Ursignal zu erstellen, das die aufgezeichneten Kanäle interferenzfrei zusammenfasst. Das spart Speicherplatz, da man nun nur noch einen Kanal und zusätzlich die Richtungsinformationen der anderen Kanäle (Impulsantworten) abspeichern muss. Dieses Signal wird durch Mittelung der Betragsspektren nach einer Kurzzeit-Fouriertransformation erzeugt wobei allerdings für die Interferenzfreiheit die Phaseninformation der Kanäle verworfen werden muss. Das gemittelte Betragsspektrum kann ohne Phase nun nicht rücktransformiert werden, da hauptsächlich starke Störungen und wenig vom Signal reproduziert würden. Zur erfolgreichen Spektrogramminversion muss eine passende Phase gefunden werden.

Nun haben wir die Gelegenheit genutzt, eine zu anderen Zwecken am IEM aufgebaute Messanordnung (Abb. 5) zur Aufnahme von Richtwirkungen von Blockflöte, Cello, und Mundharmonika zu verwenden. Zur Erzeugung von Ursignalen daraus, wurde aufbauend auf [Süs11] und [BG16] zunächst der Algorithmus von Griffin und Lim [GL84] implementiert und getestet, der in der Literatur häufig Verwendung findet. Außerdem fand sich die Idee, die noch recht neue Phase Gradient Heap Integration nach [PBS17] zu verwenden um damit eine Anfangsbedingung für den Griffin und Lim Algorithmus zu erstellen und die nötige Anzahl an Iterationen zu reduzieren.

Abbildung 1: 64-Kanal Mikrofonkugel

[1] Bild: H.Pomberger, Quelle: https://futurezone.at/science/mikrofone-und-audioformat-mit-3d-ton/24.595.600

2 Untersuchte Algorithmen

2.1 Algorithmus von Griffin und Lim (GLA)

In [GL84] wird von Daniel W. Griffin und Jae S. Lim ein Algorithmus vorgestellt um ein Signal aus seinem Betragsspektrum zu rekonstruieren, wobei auch eine iterative Form angegeben wird, die hier von Interesse ist. Kontext in dem Paper ist die Sprachsignalverarbeitung; bei verschiedenen Verfahren wird die Kurzzeit-Fouriertransformation (STFT) oder oft nur das Betragsspektrum eines Signals verändert. Aus diesen modifizierten Spektren will man dann wieder Sprachsignale gewinnen. Anders gesagt wird ein Signal gesucht, dessen STFT ein bekanntes Betragsspektrum ergibt. Um das Signal zu erzeugen muss eine gültige Phase gefunden werden.

Dies wird erreicht indem ausgehend vom bekannten STFT-Betragsspektrum wiederholt ein Zeitsignal erzeugt wird, welches danach wieder einer STFT zugeführt wird. Die Erzeugung des Zeitsignals geschieht durch inverse Fouriertransformation und eine Addition der vier- oder achtfach überlappenden Blöcke im Zeitbereich. Durch eben diese starke Überlappung ergibt sich bei jeder Addition ein Verlust eines Anteils der Signalenergie. Dieser Verlust ist zwar im hoch überbestimmten Spektrogramm zulässig, kann aber nicht der Analyse eines Zeitsignals entspringen. Andererseits korrigiert dieser Schritt die Phasenfortschreitung, die nach jedem Iterationsschritt mit dem vorgegebenen Betragsspektrogramm kombiniert wird. Abbildung 2 stellt graphisch dar, wie dabei bei jeder Rücktransformation zwar die neu gewonnene Phaseninformation, aber immer das schon bekannte, richtige Betragsspektrum verwendet wird. So wird mit jeder Iteration das Zeitsignal weiter angepasst, damit dessen Betragsspektrum möglichst gut mit dem gewünschten übereinstimmt. Mathematisch ausgedrückt wird in jedem Schritt der quadratische Fehler zwischen dem gewünschten Betragsspektrum und dem des aktuell gefundenen Zeitsignals verringert. Somit ist eine Konvergenz des Algorithmus sichergestellt.

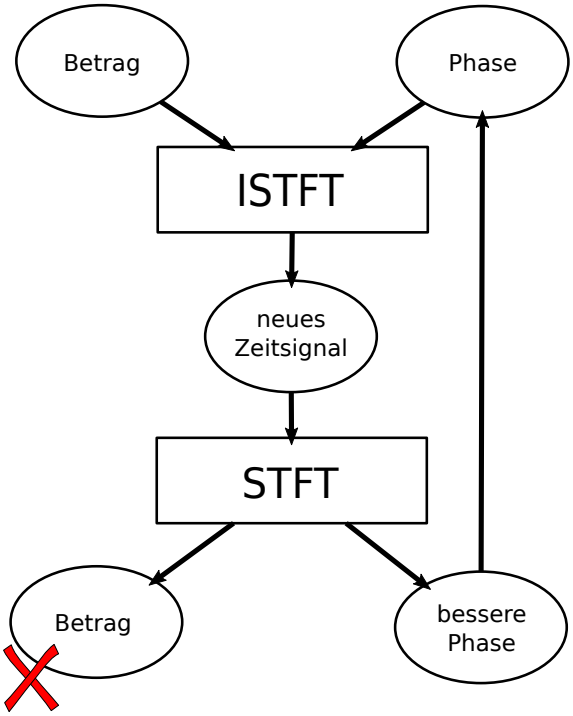

Abbildung 2: Blockschaltbild zum GLA

2.2 Phase Gradient Heap Integration (PGH)

2.2.1 Der Algorithmus

Die Phase-Gradient-Heap-Integration (PGH) ist ein nicht-iterativer Algorithmus zur Rekonstruktion der Phase eines Spektrogramms aus dessen Gabortransformationsbeträgen. Dieser wurde von Prusa (et al.) im Mai 2017 im Rahmen eines IEEE-Papers [PBS17] vorgestellt. Die Phase wird dabei, bei demjenigen Zeit-Frequenz-Bin mit der größten Amplitude zu Null gesetzt und dann in Richtung der geringsten Amplitudenveränderung kumulativ weiterentwickelt. Dem zu Grunde liegt, dass jeder Betragsänderung in eine Richtung des Spektrogramms (Block oder Frequenz) eine Phasenänderung in die jeweils andere Richtung zugeordnet ist.

2.2.2 Die mathematischen Grundlagen

Das Gradiententheorem beschreibt den Zusammenhang zwischen einem Nachbar-Phasenwert und dem nächsten daraus zu berechnenden. Folgende mathematische Beschreibung des Gradiententheorems, wie Prusa sie in seinem Paper [PBS17] zur Veranschaulichung seiner Erklärung des PGH voranstellt, zeigt diesen Zusammenhang der von ihm bezeichneten originalen Phase $\Phi^f_\varphi(\omega, t)$ und der Anfangsphase $\Phi^f_\varphi(\omega_0, t_0)$.

$$\Phi^f_\varphi(\omega, t) - \Phi^f_\varphi(\omega_0, t_0) = \int_0^1 \nabla\Phi^f_\varphi(r(\tau)) \cdot \frac{dr}{d\tau}(\tau) d\tau.$$

Dabei ist $r(\tau)$ der Integrationsweg von $\Phi^f_\varphi(\omega_0, t_0)$ nach $\Phi^f_\varphi(\omega, t)$, wobei vorausgesetzt wird, dass der Anfangspunkt $\Phi^f_\varphi(\omega_0, t_0)$ bekannt ist.

Das nach Zeit und Frequenz abgeleitete logarithmierte Spektrogramm wird jeweils mit finiten Differenzen geschätzt:

$\frac{\partial}{\partial t} s_{\log(m,n)} \approx [s_{\log}(m, n+1) - s_{\log}(m, n-1)]/(2/fs)$
$\frac{\partial}{\partial \omega} s_{\log}(m, n) \approx [s_{\log}(m+1, n) - s_{\log}(m-1, n)]/(2pi/M)$

Mittels Blockverarbeitung (Kern der Gabortransformation) ergibt sich abhängig von Zeit- und Frequenzindizes (m, n) eine Matrix, wobei m den Zeitindex darstellt. Für den Gradienten der Phase ($\nabla\Phi^{SC}$) gilt folgender mathematischer Zusammenhang bestehend aus einem Vorfaktor, dem Logarithmus des Signals (s_{log}) und den Gradientenmatrizen (D_t, D_ω),

$$\nabla\Phi^{SC}(m, n) = \left[-\frac{\lambda L}{aM}(s_{log} D_t)(m, n), \frac{aM}{\lambda L}(D_\omega s_{log})(m, n) + 2\pi am/M \right].$$

Mit:
M ... DFT-Länge (mit optionalem zero-padding)
L ... Fensterlänge (vor optionalem zero-padding)
λ ... Zeit zu Frequenzverhältnis der Blockaufteilung
α ... Sprunggröße (hop size)

Der Algorithmus arbeitet sich entlang der geringsten Phasenveränderung durch die Matrix aus STFT-Block-Bins und berechnet deren Phasenwerte aus den Benachbarten.

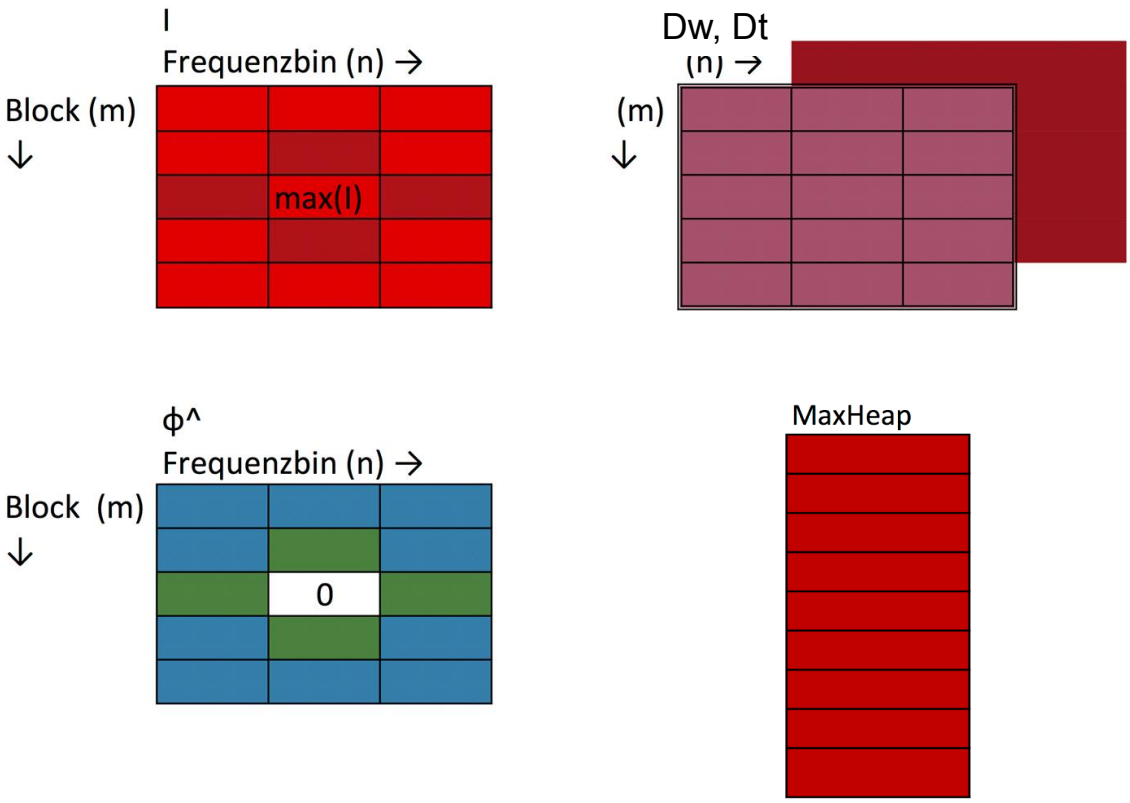

Abbildung 3: Verbildlichung der Teilstrukturen des PGHI-Algorithmus

2.2.3 Die Teilstrukturen

Der PGH verwendet genauer betrachtet zwei Matrizen, wobei in der einen die zeitliche Amplitudenableitungen zum Spektrogramm, in der anderen die frequenzbezogenen Ableitungen enthalten sind (oben rechts in Abbildung 3) und eine Eingangsmatrix (I, oben links). Außerdem ist eine self-sorting Heap-Struktur (sortierter Stapel, rechts unten) notwendig, in der alle Nachbarn des aktuell bearbeiteten Samples eingearbeitet und sortiert werden, um den richtigen Kandidaten für die nächste Berechnung auszuwählen (unten rechts in Abbildung 3). In einer Outputmatrix ($\widehat{\Phi}$, unten links) werden die integrierten Phasenwerte dann abgelegt.

Um den Algorithmus zu beschleunigen, wird vor der Ausführung der Phasenentwicklung allen Phasentermen, deren Amplitudenwerte kleiner als eine gewisse Grenze und damit nicht relevant sind ($i \leq 10^{-tol}$), ein Zufallswert (zwischen $-\pi$ und π) zugewiesen.

2.2.4 Die Umsetzung

Die Umsetzung des PGH-Algorithmus erfolgte in Matlab. Die Übergabeparameter der Funktion sind:

- stftMagnitude ... Matrix mit Blöcken des Betragsspektrums
- tolerance ... Grenze für die Zufallswerte

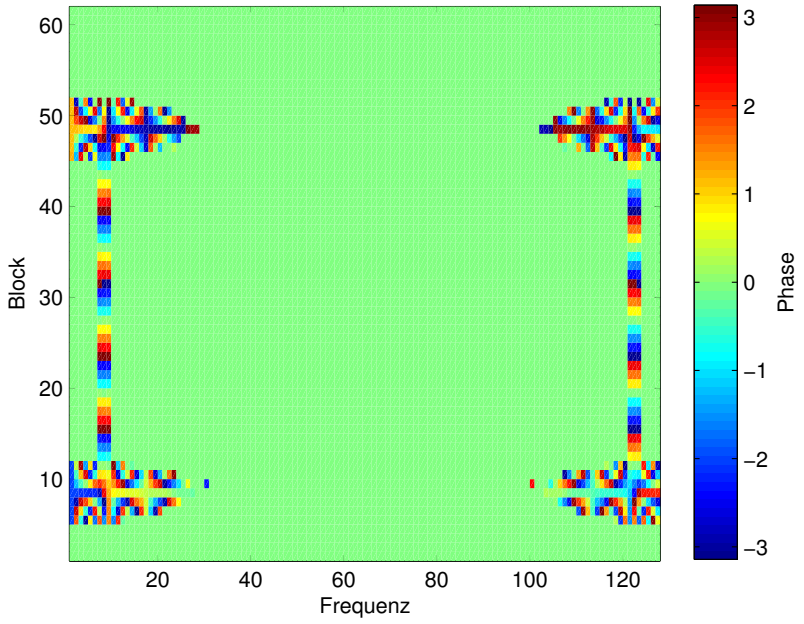

Abbildung 4: Beispiel für die errechete Phase eines Sinussignals

- Nblk . . . Blockgröße in Samples
- hop . . . Versatz der Blöcke zueinander
- Nfft . . . FFT-Blockgröße
- win_type . . . Fenstertyp 'HANN' oder 'GAUSS'

Daraus berechnet die PGH-Funktion dann die STFT-Phasenwerte, beispielhaft in Abbildung 4 für ein Sinussignal. Die Phase des Sinussignals steigt mit der Blocknummer und wurde zur besseren Erkennbarkeit auf das Intervall $\{-\pi, \pi\}$ beschränkt. Bei niedrigen, sowie hohen Blockzahlen zeigen sich Artefakte über alle Frequenzbins, ähnlich der Fouriertransformation eines Impulses. Diese entstehen durch das abrupte Einsetzen und Abbrechen der Sinusschwingung in diesen Blöcken.

3 Richtwirkungsmessung von Instrumenten

3.1 Messaufbau

In Abbildung 5 ist der Aufbau für die Richtwirkungsmessung zu sehen. Die 62 NTI Messmikrofone befinden sich auf zwei konzentrischen Kreisen. Durch die Überschneidung der Kreise ergeben sich dadurch zwei ringförmige Anordnungen mit jeweils 32 Mikrofonen. In deren Mitte ist Platz für eine/n Musiker/-in nebst Instrument. Mittels PureData können die Mikrofonkanäle einzeln im Pegel kalibriert werden. Die dafür verwendeten Patches sind in Unterkapitel 3.2 näher beschrieben.

Abbildung 5: Aufbau zur Vermessung eines Violoncellos

3.2 PD-Patch

Zur Aufnahme wurden zwei PD-Dateien verwendet. Einerseits **App_calibration_patch.pd** zur Kalibration der 62 Mikrofonkanäle, andererseits **rec_singer_v5.pd** zur Aufnahme der 64-kanaligen WAV-Dateien.

App_calibration_patch.pd ermöglicht es, alle Kanäle seriell und automatisch zu kalibrieren. Sobald der Patch gestartet wird, muss nur noch ein Messkalibrator auf jedes Mikrofon für einige Sekunden aufgesetzt werden. Die Kalibrationsdaten werden dann in einer Marixdatei gespeichert um die Aufnahme zu kompensieren.

rec_singer_v5.pd ist ursprünglich ein Patch zur Aufnahme einer/-s Sängerin/-ers. Neben den 62 Mikrofonkanälen werden daher ein Videokanal und ein Referenzmikrofon in der Kreismitte aufgenommen, die für unsere Zwecke nicht von Belang sind.

3.3 Durchführung

Nach erfolgter Kalibration und Platzierung des Instrumentes wurden kurze Musikstücke und einige längere Haltetöne aufgenommen. So entstanden Aufnahmen eines Violoncellos, einer Blockflöte und einer chromatischen Mundharmonika die im Weiteren zur Ursignalerzeugung und Richtwirkungsmessung verwendet werden können.

4 Anwendung der Algorithmen

4.1 Anwendung des GLA

Eine Implementierung der iterativen Form des GLA in Matlab ist in Listing 1 zu sehen. Aufgerufen wird diese Funktion von dem in Listing 3 abgebildeten Matlab-Skript.

Kern des Algorithmus ist wie schon beschrieben die Kurzzeit-Fouriertransformation, die pro Iteration in beide Richtungen durchgeführt werden muss. Somit ergibt sich erheblicher Rechenaufwand, und es liegt auf der Hand, dass die Parameter der STFT entscheidenden Einfluss auf den Algorithmus haben. Mehr dazu in Kapitel 4.1.1.

Erwähnenswert bleibt noch, dass in der vorliegenden Implementierung die Fenstermitte der verwendeten Blöcke/Fenster per zirkulärer Verschiebung immer auf Nullphase gedreht wird, um komplexe Spektren zu vermeiden und die Berechnung zu beschleunigen.

Außerdem wird dem Algorithmus auch eine Anfangsphase für die erste Iteration übergeben (Abbildung 2). Wenn keine zusätzlichen Informationen zur Verfügung stehen, liegt eine Initialisierung mit Nullen nahe. Die Anzahl der notwendigen Iterationen, also die Berechnungsdauer, bis sich ein brauchbares Ergebnis einstellt lässt sich jedoch durch Finden einer geeigneten Anfangsphase erheblich verringern.

Will man wie eingangs diskutiert beispielsweise ein Ursignal erzeugen, stehen zusätzlich zum gemittelten Betragsspektrum auch die einzelnen Mikrofonkanäle samt intakter Phaseninformation zur Verfügung. Initialisiert man den GLA mit Nullphase, verwirft man also Information, die mit dem gewünschten Ergebnis mit Sicherheit zu tun hat. Man kann also den Algorithmus mit der Phase eines beliebigen Mikrofonkanals initialisieren, um die Konvergenz zu beschleunigen. Überlegt man sich jedoch, dass der lauteste Kanal bei jeder Art der Mittelung den größten Anteil am Ergebnis hat, erscheint es sinnvoller die Phase eben dieses Kanals als Anfangsbedingung anzusetzen.

In Kapitel 2.2 wird mit phase gradient heap integration ein Algorithmus vorgestellt, der ebenfalls zur Erzeugung einer Initialphase für den GLA gut geeignet ist. Ein Vergleich dieser Ansätze und deren Ergebnisse finden sich in Kapitel 4.3.

4.1.1 Parametrierung

Die einzustellenden Parameter des Algorithmus beziehen sich vor Allem auf die verwendete STFT: Blocklänge, Überschneidung der Blöcke (hopsize), DFT-Länge und Fensterfunktion; außerdem eine maximale Anzahl an Iterationen bzw. ein Konvergenzwert in Dezibel, bei dem abgebrochen werden soll. Ein möglicher Unterschied zwischen DFT-Länge und Blocklänge wird durch auffüllen mit Nullen (zeropadding) ausgeglichen. Je nach Eingangssignal muss bei der Fouriertransformation ein guter Kompromiss zwischen Zeit und Frequenzauflösung gefunden werden. In Tabelle 1 sind beispielsweise Parameter zu sehen, die für ein Sinus-Testsignal zu guten Ergebnissen führten (Abbildungen 6 und 7). Die benötigte Anzahl an Iterationen um ein brauchbares Ergebnis zu erreichen, hängt dabei stark von der verwendeten Anfangsphase ab (Abbildung 18, oder 19).

Tabelle 1: Verwendete Parameter beim GLA für ein 900 Hz Sinussignal

Parameter	**Wert**	**in Blocklängen**
Blocklänge	2048	N
Hopsize	128	N/16
DFT-Länge	2048	1 N
Länge der Circ. Verschiebung	1024	N/2
Fensterfunktion	Gauss	
Iterationen	100	

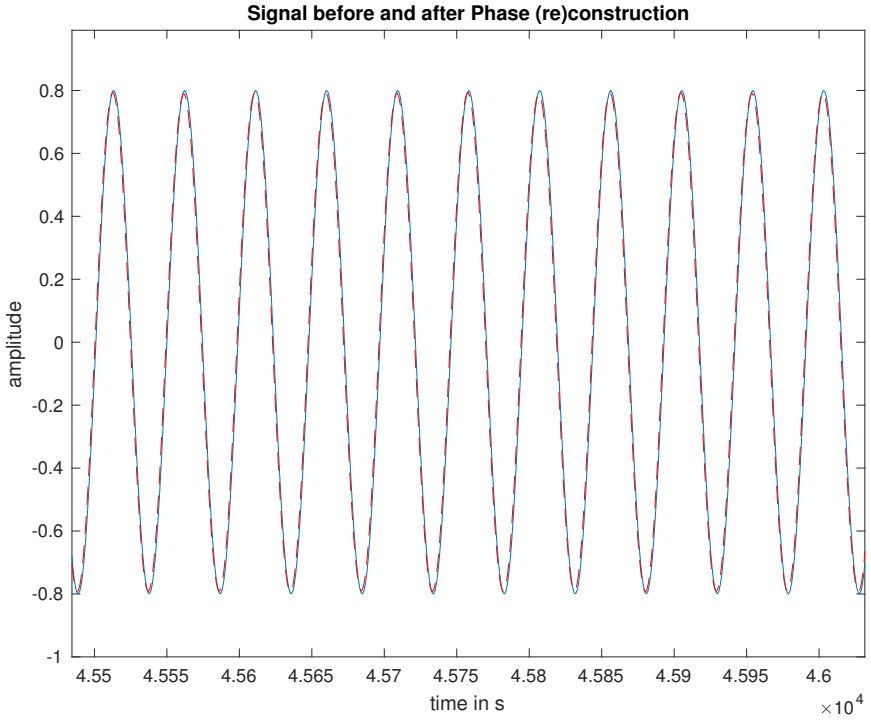

Abbildung 6: Vergleich der Signale mit Originalphase und reproduzierter Phase.

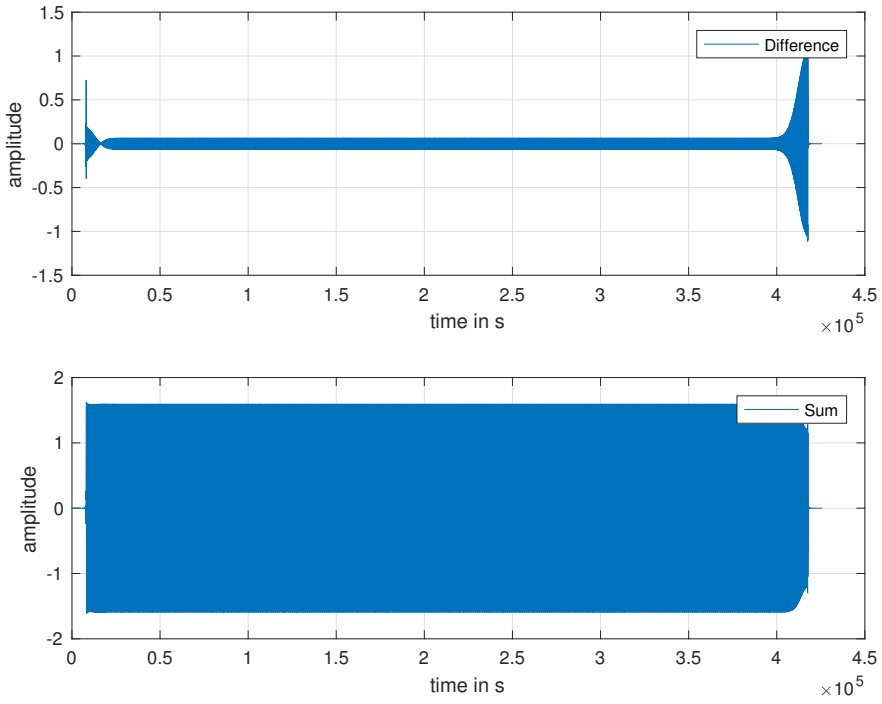

Abbildung 7: Differenz und Summe von Original und reproduziertem Signal.

4.1.2 Erzeugung von Ursignalen

Will man aus Richtwirkungsaufnahmen von Instrumenten - wie sie z.B. durch eine Messanordnung wie in Kapitel 3 entstehen - ein leichter handhabbares Ursignal erzeugen, kann man also den GLA verwenden um die fehlende Phaseninformation anzunähern. Zunächst müssen die Signale der Mikrofonkanäle jedoch in Blöcke unterteilt, gefenstert (Gauß oder Hann Fenster), zirkulär verschoben, sodass die Fenstermitte nullphasig zu liegen kommt, und fouriertransformiert werden (Matlab-Skript in Listing 3). Danach kann man die Betragsspektren über alle Kanäle blockweise mitteln und damit den GLA anwerfen. Im Matlab Skript (Listing 6) sind verschiedene Mittelungsformen implementiert. Nach der Formel

$$\bar{x} = \sqrt[\alpha]{\frac{1}{N}\sum_{i=1}^{N}(x_i)^{\alpha}} \tag{1}$$

lässt sich über den Faktor α eine arithmetische quadratische, kubische, etc. Mittelung erzielen. Im Extremfall, wenn α gegen unendlich geht, entspricht die Mittelung dem Maximalwert. Außerdem wurde noch geometrische Mittelung nach

$$\bar{x} = \sqrt[N]{\prod_{i=1}^{N} x_i} \tag{2}$$

implementiert. Siehe dazu auch [Süs11]. Nun kann mittels GLA eine Phase berechnet, und das Signal in den Zeitbereich zurücktransformiert werden.

4.2 Anwendung des PGH

Für einen Vergleich der tatsächlichen STFT-Phase eines Impulses und der durch den PGH berechneten Phase sollen folgende Testfälle untersucht werden: Ein Impuls zentriert in einem Block (Abbildung 8 und 9) und ein um 8 Stellen verschobener Impuls (Abbildung 10 und 11). In Block 25 ist die Phase für einen zentral gelegenen Impuls über alle Frequenzen konstant (Abbildung 8 und 9). Das entspricht der Erwartung, dass ein Impuls ein Schallereignis ist, das aus allen Frequenzen zum selben Zeitpunkt (gleiche Phase) besteht. Die Originalphase (Phi0) steigt in den Nachbarblöcken durch den Zeitversatz periodisch, während die PGH-Phase durch deren Entwicklung monoton mit der Frequenz ansteigt.
Bei einem um 8 Stellen (halber Blockversatz) verschobenen Impuls ist die STFT-Phase frequenzperiodisch in jedem Block (Abbildung 10). Die PGH-Phase setzt ihren Anfangswert zu Null und erkennt keine Phasenverschiebung um einen halben Block. Es entstehen zwei Blöcke (Blocknr. 25 und 26), deren Phase gleichbleibend Null ist. Die Phase der benachbarten Blöcke, bleibt aber monoton (Abbildung 11).

Ein Vergleich der auf $-\pi$ bis π begrenzten PGH- und STFT-Phase eines Sinussignals (Binfrequenz$=7$) bei dem dazu nächstgelegenen FFT-Bin ergibt Abbildung 12. Die Phasenbeträge wachsen parallel und sind um einen festen Wert (unterschiedliche Laufzeit) zueinander versetzt.

Bei einer anderen Signalfrequenz (Binfrequenz$=7,33$), die nicht exakt auf dem angezeigten Frequenzkanal liegt, zeigt sich ein nicht paralleles Verhalten (siehe Abbildung 13). Insgesamt wachsen die Phasenbeträge auch langsamer an als bei der Binfrequenz 7. Die Frequenzdifferenz von $0,33$ Schritten zum existierenden Bin ergibt eine Restfrequenz und deshalb einen Phasenzuwachs zwischen den Blöcken.

Die vom PGH gefundene Phase dreht schneller als die STFT-Phase, wodurch nach 50 Blöcken bereits eine halbe Periode Offset entsteht. Dieser und unten gezeigte Drifts legen nahe, dass für ausgedehnte Signale eine Verbesserung des Ergebnisses durch einen GLA wünschenswert ist.

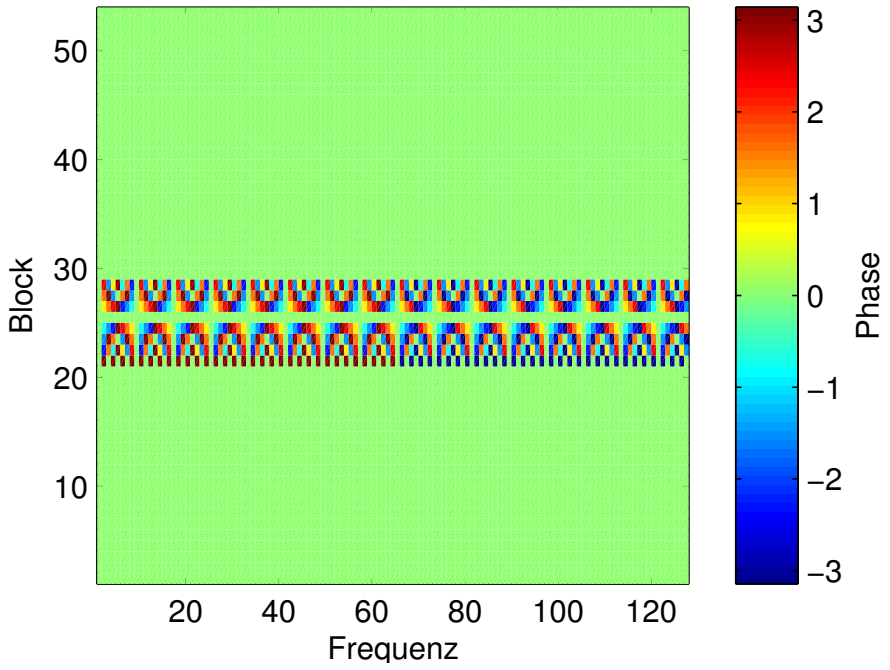

Abbildung 8: STFT-Phase eines Impulses, mittig im Block.

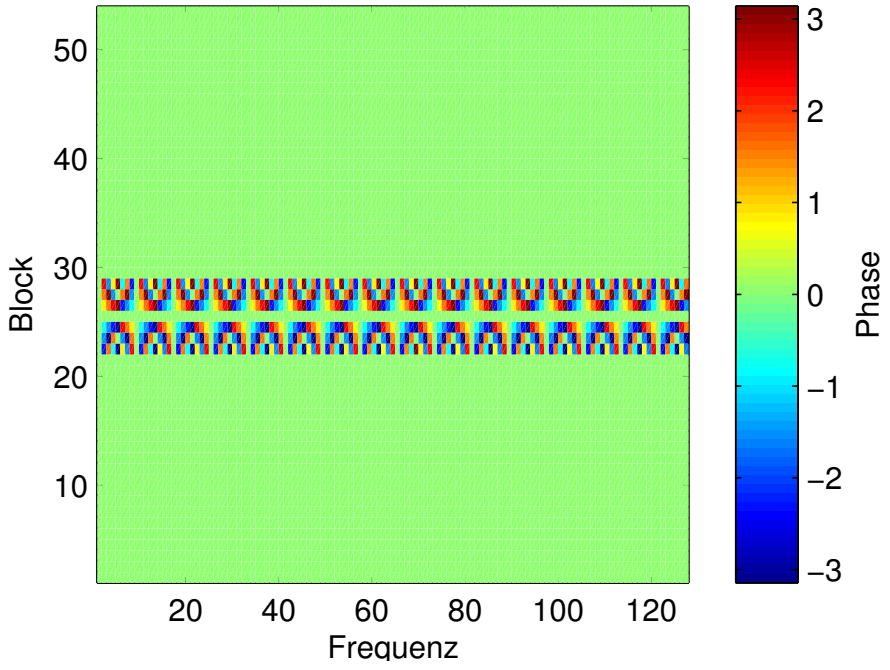

Abbildung 9: PGH-Phase eines Impulses, mittig im Block

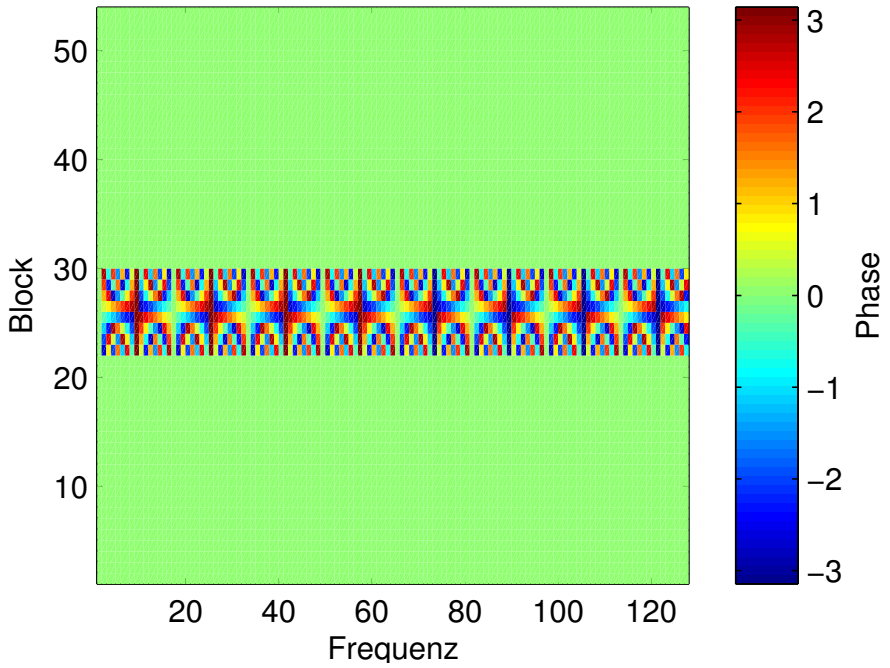

Abbildung 10: STFT-Phase eines Impulses, um 8 Samples aussermittig.

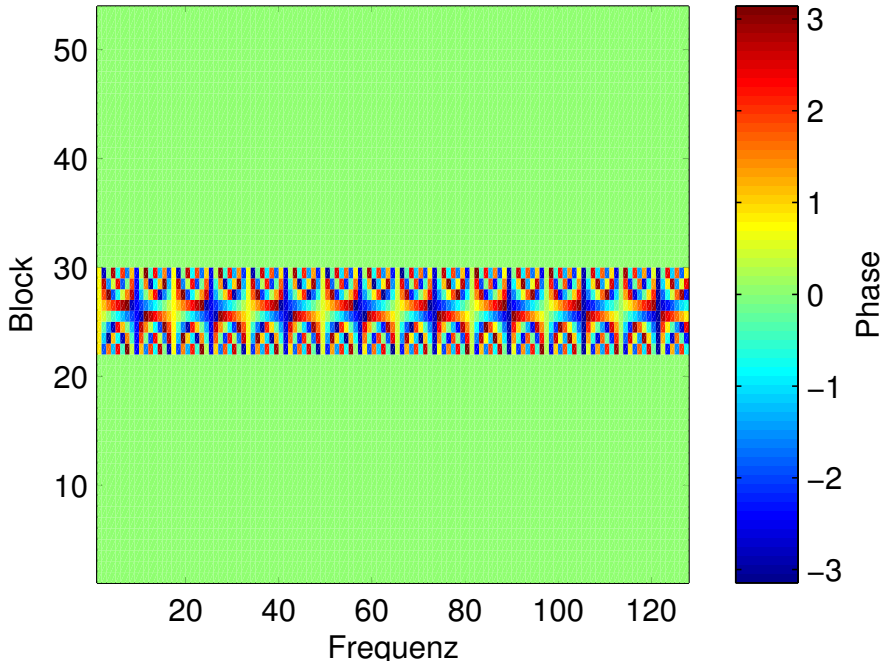

Abbildung 11: PGH-Phase eines Impulses, um 8 Samples aussermittig.

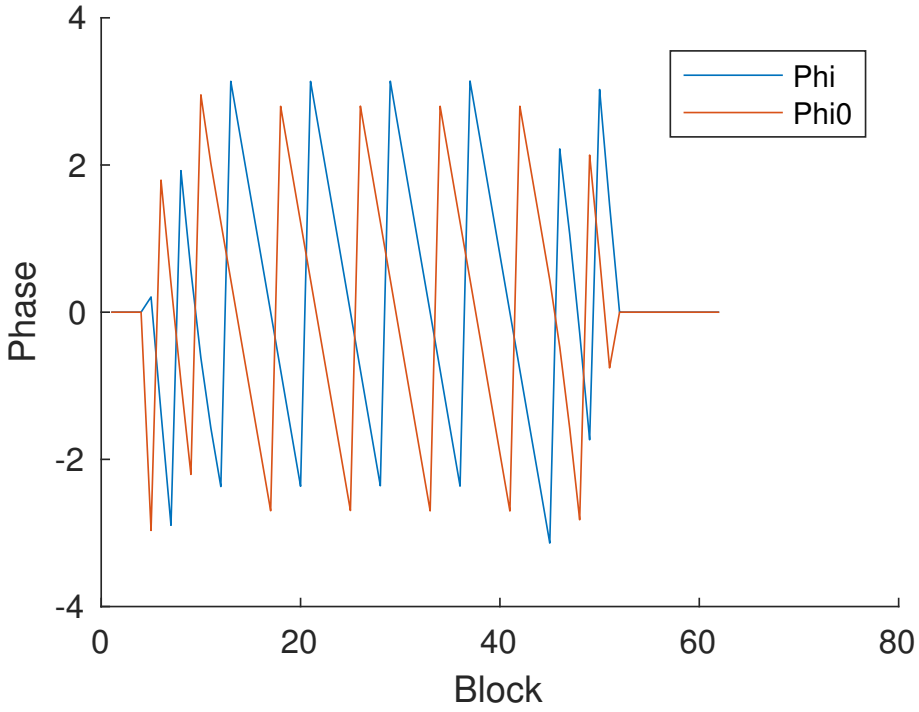

Abbildung 12: Blockweise Veränderung der Phase bei der Frequenz $f = 7$.

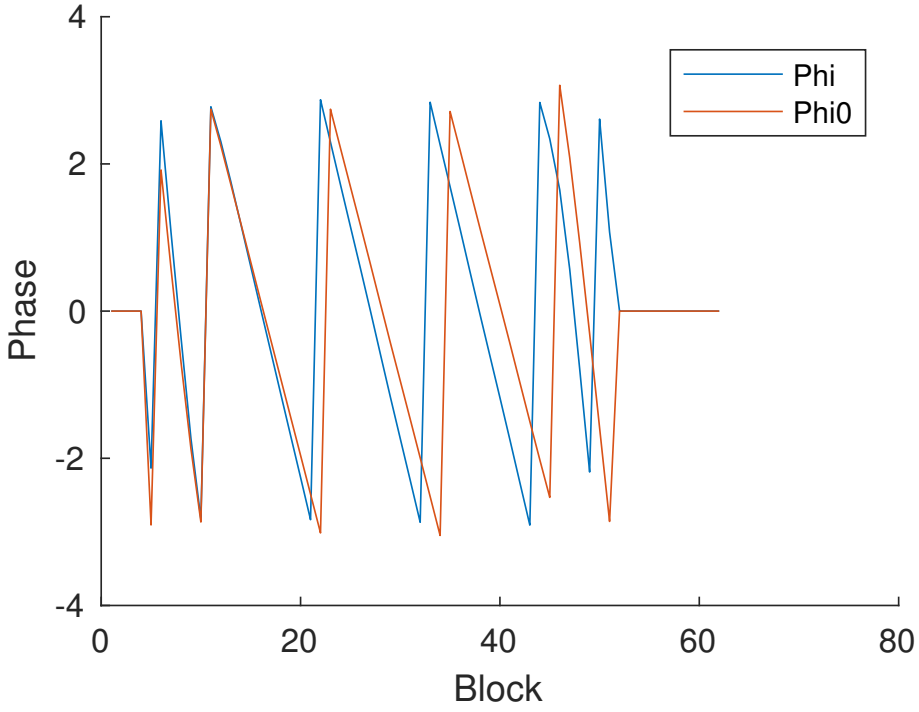

Abbildung 13: Blockweise Veränderung der Phase bei der Frequenz $f = 7.33$.

4.3 Kombination der Algorithmen

Bereits Prusa et al wiesen in [PBS17] auf die Möglichkeit hin, den PGH-Algorithmus zu verwenden um eine Phaseninitialisierung für GLA oder andere iterative Algorithmen zu erreichen. Das bietet nahezu die Geschwindigkeit eines nicht-iterativen Ansatzes und erhöht die Präzision gegenüber einem reinen Entwicklungsansatz. Um den Geschwindigkeitsvorteil und die Genauigkeit messen zu können, wird die Konvergenz beider Ansätze herangezogen.

4.3.1 Konvergenz

Die Konvergenz zu jedem Iterationsschritt ist die Summe über alle Fehlerbetragsquadrate normiert auf die Summe der Betragsquadrate des tatsächlichen Signals S,

$$Conv(it) = \frac{\sum_{n,m}[|S_i(n,m)| - |S(n,m)|]^2}{\sum_{n,m}|S(n,m)|^2.}$$

Die Plots in Abbildung 14 bis 17 zeigen jeweils den Konvergenzvergleich zwischen einem nullphasig initialisierten GLA-Algorithmus und einem PGH-initialisierten GLA. Für die vier Testsignale ergeben sich durch den PGH-Algorithmus Verbesserungen zwischen 10 dB, und 60 dB und je nach Anwendungsfall sind nur bis zu 50 Iterationen des GLA notwendig um weniger als -60 dB Fehler zu erreichen.

Offensichtlich können die Schätzfehler des PGHI bereits durch wenige GLA-Iterationen enorm verbessert werden, of reichen 4, und das Resultat ist dabei um 40dB besser als mit Nullphaseninitialisierung.

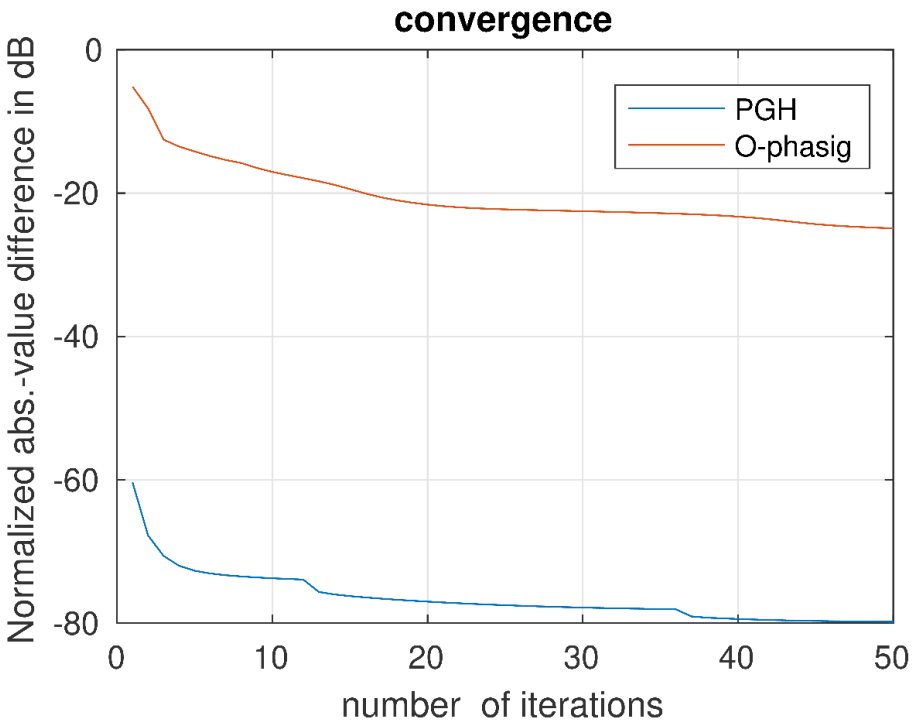

Abbildung 14: Konvergenz für ein Sinussignal mit $f = 7$.

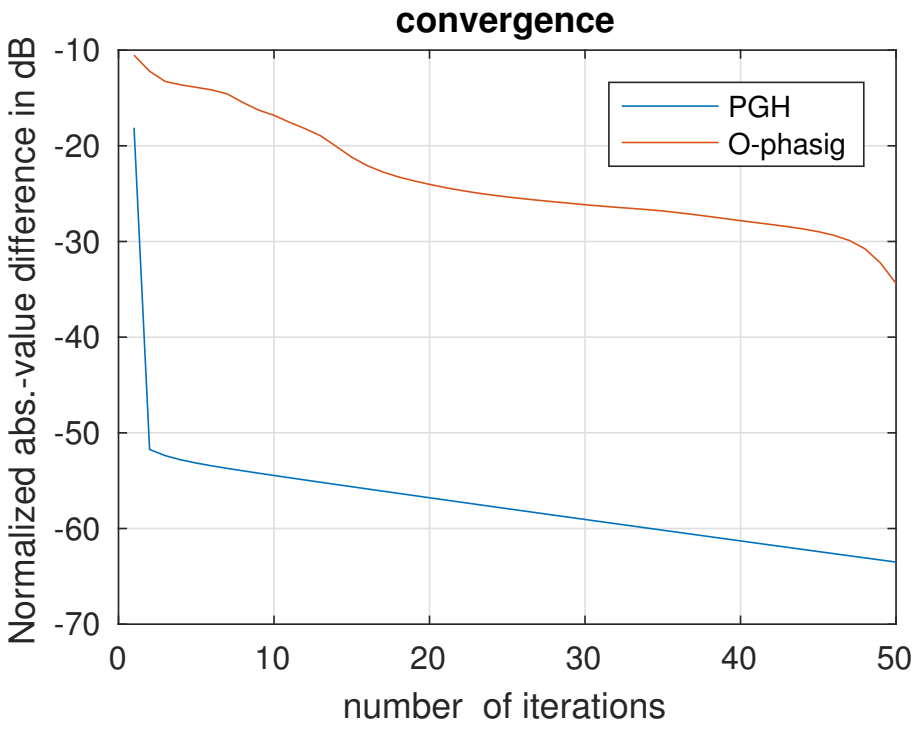

Abbildung 15: Konvergenz für ein Sinussignal mit $f = 7.33$.

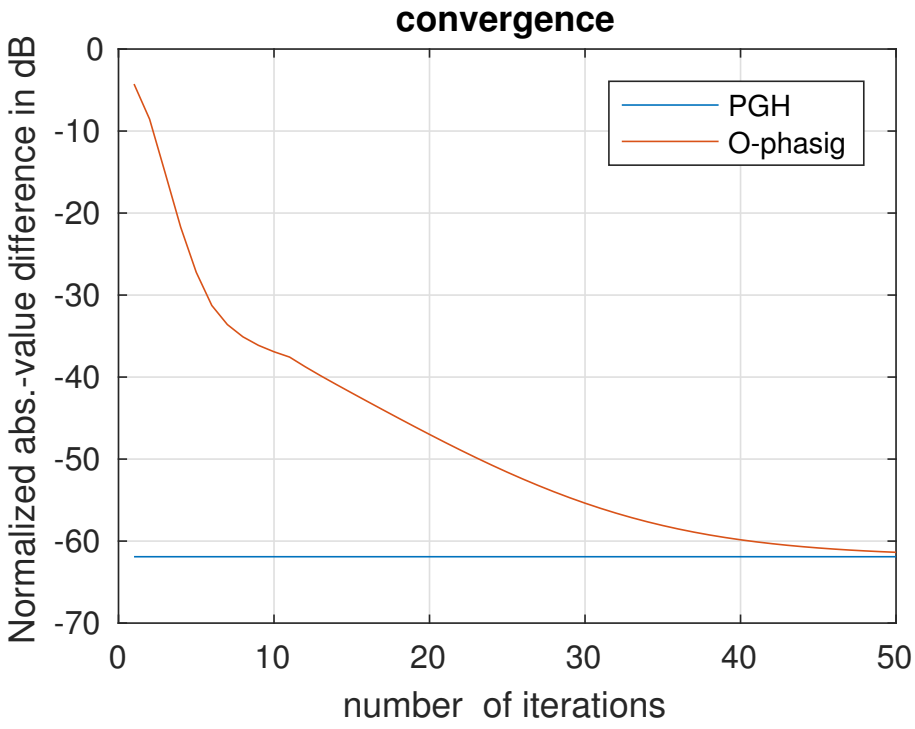

Abbildung 16: Konvergenz für einen blockzentrierten Impuls.

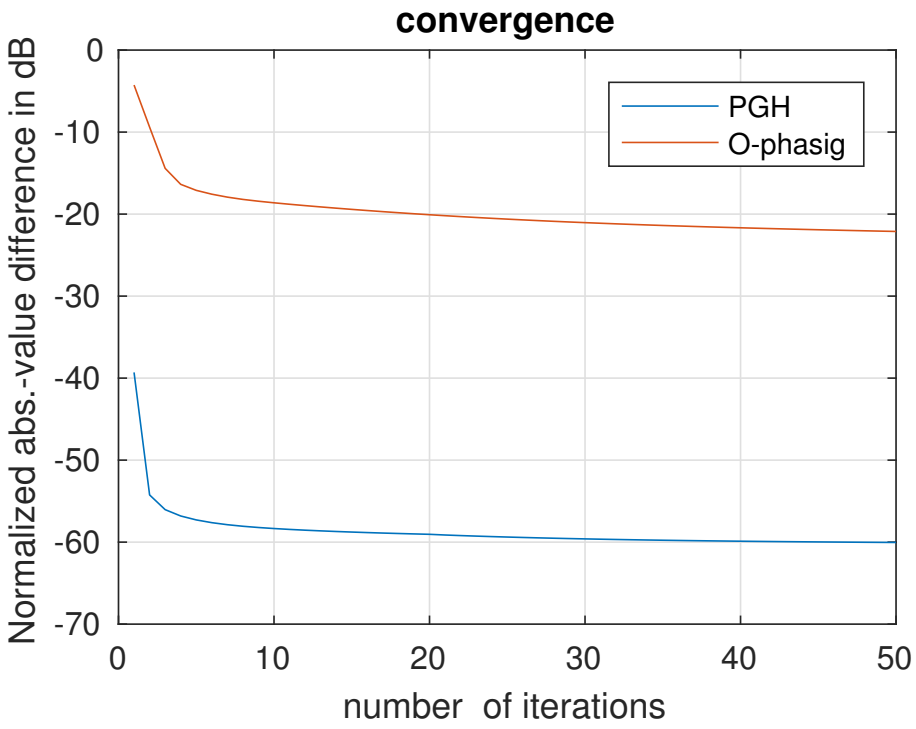

Abbildung 17: Konvergenz für einen aussermittig positionierten Impuls.

4.3.2 Geschwindigkeit

Für die vier bereits im letzten Kapitel verwendeten Testsignale ergeben sich die Ausführungszeiten in Tabelle 4.3.2. Verglichen werden der mit Nullphase initialisierte GLA (GLA 0-phasig) und der mittels PGH initialisierte GLA (GLA & PGH). Als Abbruchkriterium wurde dem GLA dabei eine Konvergenz von mindestens -50 dB mitgegeben. Die angegebene Anzahl an GLA-Iterationen waren also nötig um dieses Ziel durch verbessern der Initialphase zu erreichen. Der PGH kommt den -50 dB in den getesteten Fällen so nahe, dass maximal zwei GLA-Iterationen nötig sind. Mit Nullphase initialisiert erreicht der GLA dagegen den Wert teilweise überhaupt nicht. Erwähnenswert ist außerdem der Fall des mittig gelegenen Impulses: hier braucht der GLA alleine nur 13 Iterationen und ist damit in der Ausführungsdauer schneller als PGH plus eine Iteration GLA.

	Sinus 7		Sinus 7.33		Impuls mittig		Impuls aussermittig	
	Iterat.	Zeit	Iterat.	Zeit	Iterat.	Zeit	Iterat.	Zeit
GLA 0-phasig	52	0.1382	>500	1.1918 (500 It.)	13	0.0500	>500	1.2554 (500 It.)
GLA & PGH	1	0.0539	1	0.0659	1	0.0563	2	0.0595

Tabelle 2: Ausführungszeit der Algorithmen für obige Testsignale bis -50 dB Konvergenz erreicht sind.

4.3.3 Versuche mit Richtwirkungsaufnahmen

Für Richtwirkungsaufnahmen von Instrumenten gibt es demnach drei Testfälle für das Finden eines Ursignals (erstellen einer Phase): GLA mit Nullphase initialisiert, GLA mit der Phase des lautesten Kanals initialisiert und GLA mit dem Ergebnis des PGH initialisiert. Die beiden letzteren Anfangsphasen zeigen wesentlich besseres Konvergenzverhalten; je nach Signal schneidet die Verwendung des PGH zur Initialisierung am besten ab (siehe Abbildungen 18 und 19). Betrachtet man die dabei aufgetretenen Berechnungszeiten in Tabelle 4 erkennt man den zusätzlichen Berechnungsaufwand durch den PGH-Algorithmus. Will man allerdings einen bestimmten Konvergenzwert erreichen, spart eine Initialisierung mittels PGH Iterationen, wie im letzten Kapitel gezeigt wurde.

Tabelle 3: Verwendete Parameter zu den Abbildungen 18 und 19

Parameter	**Wert**	**in Blocklängen**
Blocklänge	2048	N
Hopsize	256	N/8
DFT-Länge	2048	N
Länge der Circ. Verschiebung	1024	N/2
Fensterfunktion	Gauss	
Iterationen	100	
Mittelungsart	quadratisch	
PGH Schwellwert	10^{-5}	

Tabelle 4: Ausführungszeiten zu den Abbildungen 18 und 19 für jeweils 100 Iterationen

Signal	**Modus**	**Zeit**	**Signallänge**
Cello	GLA - Nullphase	21.6 s	2 s
Cello	GLA - lautester Kanal	20.8 s	2 s
Cello	GLA - PGH	24. 07 s	2 s
Saxophon	GLA - Nullphase	22.19 s	2 s
Saxophon	GLA - lautester Kanal	20.01 s	2 s
Saxophon	GLA - PGH	22.6 s	2 s

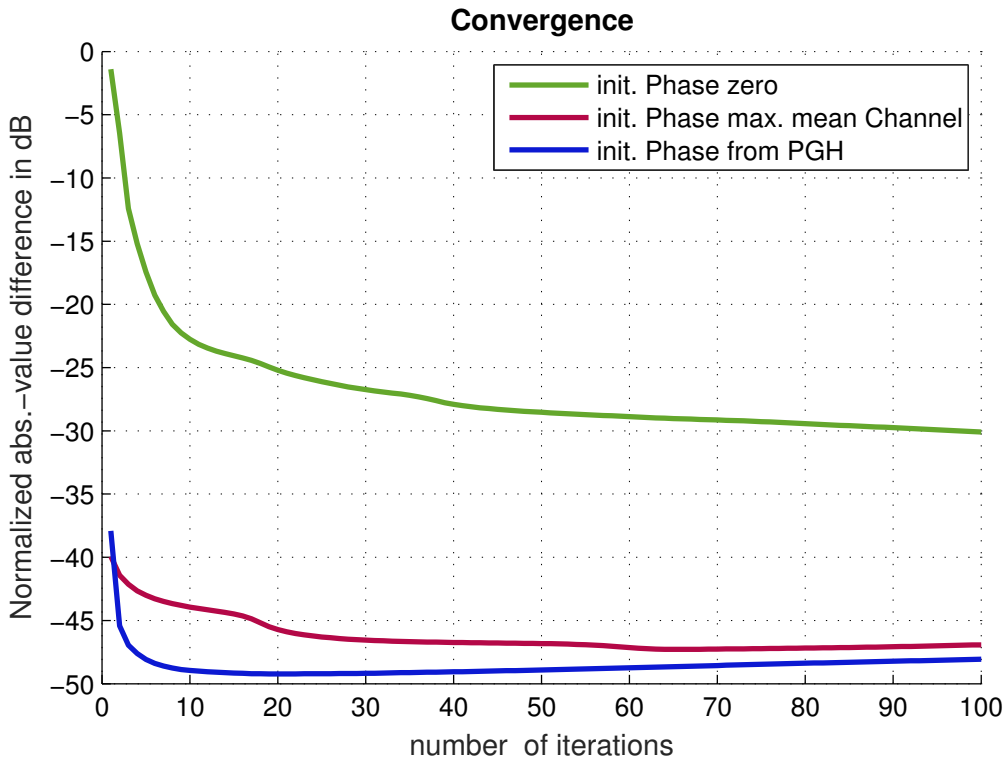

Abbildung 18: Konvergenz des GLA mit verschiedenen Anfangsphasen - Cello (vlg. Abb. 5)

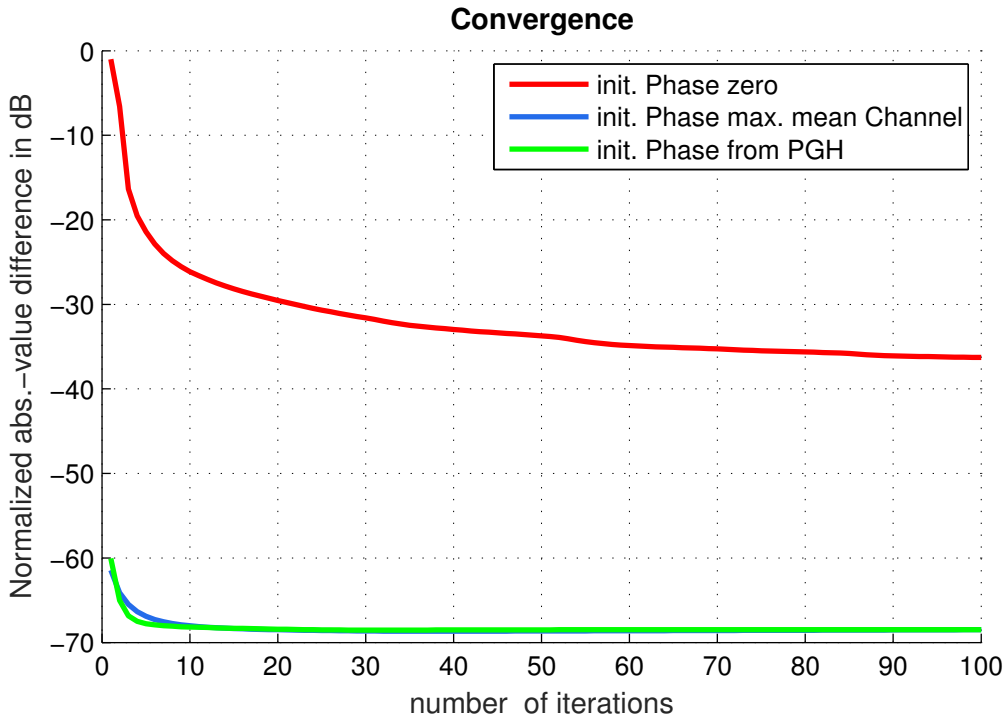

Abbildung 19: Konvergenz des GLA mit verschiedenen Anfangsphasen - Saxofon (vlg. Abb. 1)

5 Zusammenfassung und Ausblick

Am Anwendungsfall der Erzeugung von Ursignalen bei der Aufnahme von Richtwirkungen von Instrumenten wurde der Algorithmus von Griffin und Lim getestet. Der Algorithmus ist wegen der vielen Fouriertransformationen aufwendig, was sich auch in der Berechnungsdauer niederschlägt. Ein Initialisieren des Algorithmus durch die Phase des betragsmäßig lautesten Kanals beschleunigt den Vorgang erheblich. Ob das so entstandene Signal allerdings ein geeignetes Ursignal darstellt, kann man dennoch in Frage stellen. Zur besseren Beurteilung wäre es sinnvoll die entstehenden Rekonstruktionsvorschriften für die Einzelsignale zu untersuchen. Die erzeugten Ursignale klingen realistisch; es sind keine Artefakte der Signalsummierung zu hören was für eine plausibel errechnete Phase spricht.
Eine Initialisierung durch den PGH Algorithmus verringert den Aufwand beim anschließenden GLA ebenfalls beträchtlich, wobei die zusätzliche Berechnungsdauer durch den PGH natürlich zu berücksichtigen ist. Von einer Echtzeitanwendung ist man auch hier weit entfernt - in dieser Hinsicht sind wohl eher Ansätze in Richtung RTISI wie z.B. in [ZBW06] geeigneter.
Die im Rahmen des Seminars entstandenen Multikanalaufnahmen von Blockflöte, Mundharmonika und Cello stehen für weitere Arbeiten zur Verfügung.

A Matlab Code

Listing 1: GLA.m

```
function [Phi, convergence] = GLA(S, hop, N, Ncirc, w, its, satConv, ws,log)
% Function implementing Griffin and Lim Algorithm to generate a Phase from
% an STFT magnitude.
%
% Based on:
%   [1]
%
% IN:
%   S ... stft magnitude in a matrix format. Rows contain samples
%                   in time, columns samples in frequency.
%   hop ... Hop size for block processing
%   N ... block length
%       Ncirc ... circular shift length
%       w ... window function to use
%       its ... maximum number of iterations
%   satConv ... dB value for satisfying Convergence (stop Iterations)
%       ws ... window correction term

% OUT:
%   Phi ... estimated phase values for the given magnitudes.
%       convergence ... for convergence plot
%
% Authors: Franz Zotter, Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

W = repmat(w,[1 size(S,2),size(S,3)]);
Nfft = size(S,1);
absS = abs(S);
Phi = angle(S);
convergence = zeros(its,1);

for it = 1:its
    S = absS.*exp(1i*Phi);          % construct complex exponential with (updated) phase
    S = real(ifft(S));              % inverse DFT (blocked)
    S = circshift(S,Ncirc);         % shift the center of the window function to zero
        phase
    s = signal_overlapp_add(S,hop); % reconstructed signal
    s  =  s./ws;
    S = signal_blocking(s,hop,N); % segment in blocks
    S = S(:,1:size(absS,2),:);
    S = W.*S;
    S = [S;zeros(Nfft-N,size(S,2),size(S,3))]; % Zeropadding to DFT length
    S = circshift(S,-Ncirc);
    S = fft(S,Nfft);
    Phi = angle(S);                 % new phase values
    absS_it = abs(S);
    convergence(it) = sum((absS_it(:)-absS(:)).^2)./sum(absS(:).^2);
    if abs(db(convergence(it))) > satConv
        break;
    end
end
fprintf(log,'Iterations: %d\n',it);
```

Listing 2: PGH.m

```
function phaseEstimate = PGH(stftMagnitude, tolerance, Nblk, hop, Nfft, win_type)
% Function implementing Phase Gradient Heap Integration to generate a Phase from
% an STFT magnitude.
%
% Based on:
%   [1] Prusa et al. "A Noniterative Method for Reconstruction of Phase from
%       STFT Magnitude",
%       IEEE/ACM Transactions on Audio, Speech, and Language Processing
%       Volume 25 Issue 5 Pages 1154-1164,
%       2017
%
% IN:
%   stftMagnitude ... stft magnitude in a matrix format. Rows contain samples
%                     in time, columns samples in frequency.
%   tolerance ... order of magnitude of the tolerance from the maximum value of
%                 the stftMagnitude. Bins with a magnitude below exp(tolerance)
%                 times the maximum value are assigned a random Phase.
%   Nblk ... blocksize in Samples
%       hop ... Samples, hopsize in time
%   Nfft ... DFT length in Samples
%       win_type ...window type, 'Hann' or 'Gauss'
%
% OUT:
%   phaseEstimate ... estimated phase values for the given magnitudes.
%
% Authors: Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

% initialize variables
  numberOfBlocks = size(stftMagnitude,2);  % | N
  freqVec = (0:min(Nfft-1,size(stftMagnitude,1)-1))'; % Vector of frequency steps | m

  lambda = hop*Nfft / Nblk; % time frequenzy ratio of window, as in [1], section IV, A
  switch win_type
      case 'Hann'
          lambda = 2.413*lambda;
      case 'Gauss'
          lambda = 2*lambda;
      otherwise
          error('unknown win_type');
  end

  % initialize phase estimates with random values from [0,2*pi]
  phaseEstimate = zeros(size(stftMagnitude))*2*pi;

  % initialize the heap empty
  heap = [];

  % calculate numerical differentiation of magnitude in time and frequency
  [timeDiff,freqDiff] = gradient(log(stftMagnitude+exp(-10)));

  % calculate the scaled stft phase gradient (approximation)
  % consisting of the numerical differentiations of the stft magnitude
  % in time and frequency. See (29) in [1]
  frequencyPhaseGradient =  -(lambda/hop * (Nblk/Nfft)).*timeDiff ;
  timePhaseGradient      = (hop/(lambda)*(Nfft/Nblk)).*freqDiff + ...
                           repmat(2*pi*hop*freqVec/Nfft, 1,numberOfBlocks) ;

  % ignore bins with a magnitude below exp(tolerance) times the maximum magnitude
  stftMagnitude(stftMagnitude < exp(tolerance)*max(max(stftMagnitude))) = 0;

  % the following implements Algorithm 1 from [1]
  while any(stftMagnitude(:) ~= 0 )
    if isempty(heap)
        [val,idx] = max(stftMagnitude(:));
        [m,n] = ind2sub(size(stftMagnitude),idx);
        heap = [val,m,n];
```

```
68
69            phaseEstimate(m,n)= 0;
70            stftMagnitude(m, n) = 0;
71          end
72
73          while (~isempty(heap))
74            [~,idxmax] = max(heap(:,1));
75            m = heap(idxmax,2);
76            n = heap(idxmax,3);
77            heap(idxmax,:) = [];
78
79            if (m+1 < size(stftMagnitude,1) && stftMagnitude(m+1,n) ~= 0)
80              phaseEstimate(m+1, n) = phaseEstimate(m,n) + ...
81                                      0.5*  (frequencyPhaseGradient(m,n) + ...
82                                        frequencyPhaseGradient(m+1, n));
83              heap = [heap; stftMagnitude(m+1,n),m+1,n];
84              stftMagnitude(m+1,n) = 0;
85            end
86
87            if ((m-1 >0)  && (stftMagnitude(m-1,n)~= 0 ))
88              phaseEstimate(m-1, n) = phaseEstimate(m,n) - ...
89                                      0.5*  (frequencyPhaseGradient(m,n) + ...
90                                        frequencyPhaseGradient(m-1, n));
91              heap = [heap; stftMagnitude(m-1,n),m-1,n];
92              stftMagnitude(m-1,n) = 0;
93            end
94
95            if ((n+1 < size(stftMagnitude,2))  && (stftMagnitude(m,n+1)~= 0) )
96              phaseEstimate(m, n+1) = phaseEstimate(m,n) + ...
97                                0.5*   (timePhaseGradient(m,n) + ...
98                                        timePhaseGradient(m, n+1));
99              heap = [heap; stftMagnitude(m,n+1),m,n+1];
100             stftMagnitude(m,n+1) = 0;
101           end
102
103           if ((n-1 >0)  && (stftMagnitude(m,n-1)~= 0) )
104             phaseEstimate(m, n-1) = phaseEstimate(m,n) - ...
105                               0.5*  (timePhaseGradient(m,n) + ...
106                                       timePhaseGradient(m, n-1));
107             heap = [heap; stftMagnitude(m,n-1),m,n-1];
108             stftMagnitude(m,n-1) = 0;
109           end
110         end
111       end
112     end
113
114     % EOF
```

Listing 3: ALGOmain.m

```
clear all; close all;
% Script generating Phase values from an STFT Magnitude.
% Depending on the operating mode, either the Griffin and Lim Algorithm or
% Phase Gradient Heap Integration or both are applied.
%
% PARAMETERS:
%
%       operatingMode: 1 ... GLA initialized with zero
%                              2 ... GLA initialized with the phase from the channel with max.
%                                    stft-magnitude - multichannel input signals only
%                              3 ... GLA performed after PGH
%                              4 ... PGH only
%                              5 ... run 1) through 3) for nice convergence comparison
%
%       saveOutput ... save reconstructed signal after finding a phase
%       Nblk ... blocksize in Samples
%       Nfft ... DFT length in Samples
%       overlap ... overlap factor of neighbouring blocks
%       win_type ... window type, 'Hann' or 'Gauss'
%       enfoceHermitian ... enforce hermitian symmetry during PGH
%       tol ... order of magnitude of the tolerance from the maximum value of
%           the stftMagnitude. Bins with a magnitude below exp(tolerance)
%           times the maximum value are assigned a random Phase during PGH.
%
%       alpha ... STFT magnitude averaging mode, see average_magnitudes.m
%
% DEPENDENCIES:
%
%       GLA.m
%       PGH.m
%       average_magnitudes.m
%       signal_blocking.m
%       signal_overlapp_add.m
%
% Authors: Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

%------------------------%
%--- Parameters ---%
%------------------------%
operatingMode = 5; % 1) GLA initialized with zero
                                % 2) GLA initialized with the phase from the channel with max.
                                %    stft-magnitude - useful for multichannel input signals only
                                % 3) GLA performed after PGH
                                % 4) PGH only
                                % 5) run 1) through 3) for nice convergence comparison
saveOutput = 0;

Nblk = 4096; % Samples | M
Nfft = 2*Nblk; % Samples
overlap = 4;
win_type = 'Gauss';

maxIts = 100; % maximum number of GLA Iterations
satConv = inf; % dB, satisfying Convergence for GLA

enforceHermitian = 0;
tol = -5; % exponent (power of ten)

alpha = 2; % STFT magnitude averaging mode

%------------------------%
%--- Read Signal ---%
```

```
64  %-----------------------%
65  [filename, filepath] = uigetfile('.wav','select .wav File');
66  file = strcat(filepath,filename); % Filename und Pfad verbinden
67
68  [s, fs] = audioread(file);
69
70  % Remove CH 63 & 64 from our recordings (Trackingdata and Reference mic)
71  % s = s(:,1:end-2);
72
73  hop = Nblk*1/overlap; % Samples, hopsize in time | a
74  % fill up with zeros where window function fades in and out
75  s = [zeros(Nblk+3*hop,size(s,2)); s; zeros(Nblk+3*hop,size(s,2))];
76
77  %-----------------------%
78  %--- Processing ---%
79  %-----------------------%
80  %-- Signal Blocking & Preprocessing --%
81  S = signal_blocking(s,hop,Nblk);
82
83  switch win_type
84      case 'Hann'
85          w = cos(pi*(-Nblk/2+1:Nblk/2)'/Nblk).^2;
86      case 'Gauss'
87          w = gausswin(Nblk+1);
88          w = w(2:end);
89  end
90
91
92  W = repmat(w,[1 size(S,2),size(S,3)]);
93
94  % sum of windows for correction term
95  % ws=signal_overlapp_add(W,hop)/2;
96  ws = overlap/2; % use simpler method b.c. of line 75
97
98  Sw = W.*S;
99
100 %- Zero-Padding -%
101 Sw = [Sw;zeros(Nfft-Nblk,size(Sw,2),size(Sw,3))];
102 Sw0 = Sw;
103
104 %- shift the center of the window to zero phase -%
105 Sw = circshift(Sw0,-Nfft/2+1);
106
107 %-- STFT --%
108 Swf = fft(Sw,Nfft);
109 Phi0 = angle(Swf);
110
111 %-- Averaging STFT Magnitude for multichannel signals --%
112 if size(s,2) >= 2
113     SwfAllCH = Swf;
114         [Swf, indexMaxMean] = average_magnitudes(abs(Swf), alpha);
115         Phi0 = 0;
116 else
117     indexMaxMean = 1;
118 end
119
120 switch operatingMode
121     case 1
122         numGLAs = 1;
123     case 2
124         numGLAs = 2;
125         iterations = 100;
126     case 3
127         numGLAs = 3;
128         case 4
129                 numGLAs = 0;
130         disp('PGH:')
131         tic
```

```
132                     Phi = PGH(abs(Swf), tol, Nblk, hop, Nfft, win_type);
133             toc
134             case 5
135                     numGLAs = 1:3;
136     end
137
138     % --- GLA - algorithm --- %
139     % Initialize convergence plot
140     figure
141     hold on
142     grid on
143     title('Convergence')
144     ylabel('Normalized abs.-value difference in dB')
145     xlabel('number of iterations')
146
147     for k = numGLAs
148         tic
149         switch k
150                     case 1
151                     Phi = 0;
152                     legendString = 'init. Phase zero';
153                 case 2
154                     Phi = angle(SwfAllCH(:,:,indexMaxMean));
155                      legendString = 'init. Phase max. mean Channel';
156                     case 3
157                             Phi = PGH(abs(Swf), tol, Nblk, hop, Nfft, win_type);
158                             legendString = 'init. Phase from PGH';
159
160         end
161         disp(legendString);
162
163         [Phi, convergence] = GLA( ...
164                                 abs(Swf).*exp(1i*Phi), ... % Signal to use GLA on (STFT
                                        magnitude and starting Phase)
165                                 hop, ...                                % Hop size for
                                        block processing
166                                 Nblk, ...                               % block length
167                                 Nblk/2, ...                         % circular shift length
168                                 w, ...                                  % window
                                        function to use
169                                 maxIts, ...                     % max number of iterations
170                                 satConv, ...                % break on dB conversion
171                                 ws ...                                  % window
                                        correction term
172                             );
173             toc;
174
175             % Plot Convergence
176         plot(db(convergence),'color',rand(1,3),'DisplayName',legendString)
177     end
178     % enable legend on convergence plot
179     legend('show')
180
181     % --- Resynthesis --- %
182     % enforce Hermitian symmetry:
183     if enforceHermitian == 1
184         Phi = [Phi; zeros(Nfft-size(Phi,1),size(Phi,2))];
185         Phi(Nfft/2+2:Nfft,:)=-flipud(Phi(2:Nfft/2,:));
186     end
187
188     Swf = abs(Swf).*exp(1i*Phi);
189     Sw = real(ifft(Swf));
190
191     % shift back from zero phase to original
192     Sw = circshift(Sw,Nfft/2-1);
193
194     % remove zero padding
195     Sw = Sw(1:Nblk,:,:);
```

```
196
197 % Overlapp and Add and account for the gain of the sum of the windows
198 sr = signal_overlapp_add(Sw,hop)./ws;
199
200 if saveOutput == 1
201         [filename, filepath] = uiputfile('.wav','select .wav File');
202         file = strcat(filepath,filename); % Filename und Pfad verbinden
203         audiowrite(file,0.8*sr,fs);
204 end
205
206 %------------------%
207 %--- Plots ---%
208 %------------------%
209
210 if size(s,2) <= 2
211     figure
212     plot(s)
213     hold on
214     plot(sr,'r--')
215     title('Signal before and after Phase (re)construction')
216
217     figure
218     subplot(211)
219     plot(s-sr(1:length(s)))
220     legend('Difference')
221     grid on
222     subplot(212)
223     plot(s+sr(1:length(s)))
224     legend('Sum')
225     grid on
226 end
```

Listing 4: signal_blocking.m

```
function S = signal_blocking(s, hop, N)
% Function segmenting a possibly multichannel signal for block processing.
%
% IN:
%   s ... signal to segment
%
%   hop ... hopsize in Samples
%
%   N ... blocksize in Samples
%
% OUT:
%   S ... Matrix of segmented signal
%
% Authors: Franz Zotter, Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

L = size(s,1);
Hops = ceil(L/hop);
Ch = size(s,2);
S = zeros(N,Hops,Ch);
idx = 1:N;
maxhops1 = Hops-N/hop;

for h = 1:maxhops1
    S(:,h,:) = reshape(s((h-1)*hop+idx,:),[N 1 Ch]);
end

for h = maxhops1+1:Hops
    lastidx = (h-1)*hop+idx;
    lastidx = lastidx(lastidx<=L);
    S(1:length(lastidx),h,:)=reshape(s(lastidx,:),[length(lastidx) 1 Ch]);
end
```

Listing 5: signal_overlapp_add.m

```
function s = signal_overlapp_add(S, hop)
% Function performing an overlapp and and operation for reconstruction of a
% signal after blockprocessing (via signal_blocking.m)
%
% IN:
%         S ... Matrix of segmented signal
%
%       hop ... hopsize in Samples
%
% OUT:
%   s ... reconstructed signal.
%
% Authors: Franz Zotter, Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

N = size(S,1);
Ch = size(S,3);
Hops = size(S,2);
L = (Hops-1)*hop+N;
s = zeros(L,Ch);
idx = 1:N;

for h = 1:Hops
    s(idx+(h-1)*hop,:) = s(idx+(h-1)*hop,:) +...
        squeeze(S(:,h,:));
end
```

Listing 6: average_magnitudes.m

```
function [magAVG, indexMaxMean] = average_magnitudes( ...
                                                                                magSTFT
                                                                                    ,
                                                                                    ...
                                                                                alpha
                                                                                    ...
                                                                                )

% Function to average over STFT magnitudes.
%
% IN:
%   magSTFT ... Matrix of STFT magnitude vectors to be averaged.
%
%   alpha ... power of used averaging formula (1 ... arithmetic, inf ... max)
%
% OUT:
%   magAVG ... averaged STFT magnitude
%
%   indexMaxMean ... index of the loudest channel (maximum mean value)
%
% Authors: Ruediger Fasching, Johanna Kerber, Korbinian Wegler
% Date: 2018

numChannels = size(magSTFT, 3);

% find channel with maximum mean value over all bins and blocks
[valueMaxMean, indexMaxMean] = max(mean(mean(abs(magSTFT))));

if alpha >= 1 && alpha < inf
        magAVG = (sum(magSTFT.^alpha, 3)/numChannels).^(1/alpha);
elseif alpha == inf % use maximum value for "averaging"
     magAVG = magSTFT(:,:,indexMaxMean);
elseif alpha == -1
  magAVG = (prod(magSTFT, 3)).^(1/numChannels); % geom
  fprintf('Geom. mean\n');
end
end

% EOF
```

Literatur

[BG16] C. Baumgartner and S. Grill, "Resynthese von spektrogrammen mit nicht vorhandener oder unzureichender phaseninformation, seminararbeit algorithmen 2 se," 2016.

[GL84] D. W. Griffin and J. S. Lim, "Signal estimation from modified short-time fourier transform," in *IEEE Transactions on Acoustics, Speech Signal Processing, Vol. ASSP-32, No. 2*, 1984.

[PBS17] Z. Prusa, P. Balazs, and P. L. Sondergaard, "A noniterative method for reconstruction of phase from stft magnitude," in *IEEE/ACM Transactions on Audio, Speech and Language Processing, Vol. 25, No. 5*, 2017.

[Süs11] S. Süss, "Auffinden von ursignalen aus aufnahmen umgebender kugelförmiger mikrofonanordnungen," Audio Engineering Project Paper, Institut für Elektronische Musik und Akustik, Kunstuni Graz, Technical University Graz, Graz, Austria, 2011.

[ZBW06] X. Zhu, G. T. Beauregard, and L. Wyse, "Real-time iterative spectrum inversion with look-ahead," in *2006 IEEE International Conference on Multimedia and Expo*, July 2006, pp. 229–232.