Trennbarkeit von Instrumenten in monophonen Aufnahmen

Elisabeth Frauscher

Betreuung: Dr. Alois Sontacchi

29. Juni 2018, Graz

Inhaltsverzeichnis

1 Abstract

Trennbarkeit von Instrumenten in monophonen Aufnahmen

Im Rahmen dieses Toningenieur-Projektes wird die Non-negative-Matrix-Factorization (NMF) als Methode für die Quelltrennung bei Mono-Audiosignalen ausgewählt. Der Ansatz wird in Matlab implementiert und evaluiert. Die entstandene Toolbox bietet die Möglichkeit Audiosignale und Trainingsdaten zu laden, aus einem Plot die zu separierenden Komponenten auszuwählen und das Ergebnis akustisch wiederzugeben.

Für die Initialisierung werden verschiedene Methoden und Trainingsdaten inklusive eigener Bibliothek getestet. Die Faktorisierung wird für das Signal im Frequenzbereich mit Hilfe des multiplikativen Update Algorithmus und variablen Kostenfunktionen realisiert. Die NMF bezieht allerdings nur die Beträge als Input für das Erstellen der spektralen Masken mit ein, wodurch Artefakte bei der Separation auftreten. Deshalb werden zusätzlich Optimierungen, wie eine Phasenglättung in der Nachbearbeitung, implementiert. Mögliche Einsatzgebiete dieser Quelltrennung sind Crosstalk-Cancellation in Bühnensituationen oder Quelltrennung für Spatialisierungen für 3D Audio Anwendungen.

Separability of instruments in monophonic recordings

In this project thesis, the Non-negative-Matrix-Factorization (NMF) is chosen to be the method for source separation in monophonic audio signals. The approach is implemented and evaluated in Matlab. With the resulting toolbox it is possible to load audio signals and training data, choose the components which should be separated and analyse the result in a visual and aural way.

For the initialisation different methods and datasets are tested. The factorization of the signal in the frequency domain is realised with the multiplicative update algorithm and variable cost functions. NMF only uses the absolute values to gain the spectral masks and thus causes some artefacts during the separation process. Therefore, additional optimizations like phase smoothing in the post-processing are implemented. Possible applications of this source separation are crosstalk-cancellation on stages or source separation for spatialization in 3D audio.

2 Einleitung

Ziel des Toningenieur-Projektes ist es eine Toolbox zu implementieren und aufzubauen, die Audiosignale faktorisiert. Leitend soll die Idee einer Crosstalk-Cancellation beziehungsweise Source Separation in Bühnensituationen sein. Auch wenn für dieses Anwendungsgebiet der Einsatz im Live-Technik Bereich interessant wäre, wird rein die Entwicklung in Matlab inklusive Initialisierung, Implementierung der Algorithmen, Evaluierung und Optimierungen bzw. Post-Processing fokussiert.

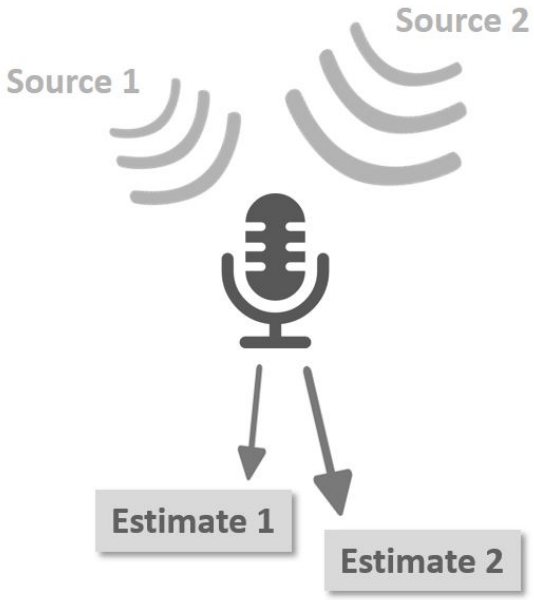

Abbildung 2.1: Schema zur Quelltrennung

Die Source Separation sollte prinzipiell per NMF (Non-negative Matrix Factorization), PCA (Principal Component Analysis) oder ICA (Independent Component Analysis) möglich sein, um die Ergebnisse miteinander vergleichen zu können und um überprüfen können welche dieser Faktorisierungen überhaupt funktionieren. Jedoch liegt der Hauptaugenmerk auf der NMF, für die hauptsächlich evaluiert und getestet werden soll.

3 Faktorisierung

Die Eingangsmatrix **X** [F x N], z.B. ein Audiosignal im Frequenzbereich mit F Frequenzbins und N Zeitsamples, wird in zwei Matrizen **W** [F x K] und **H** [K x N] faktorisiert. Je nach Anwendung oder User-Input wird in K Komponenten zerlegt. In Abbildung 3.1 wären es 3 Komponenten, wobei eine dieser Komponenten beispielsweise ein Ton oder das Frequenzmuster eines Instruments sein könnte. Es gilt $K \leq F, N$. [1, S. 2]

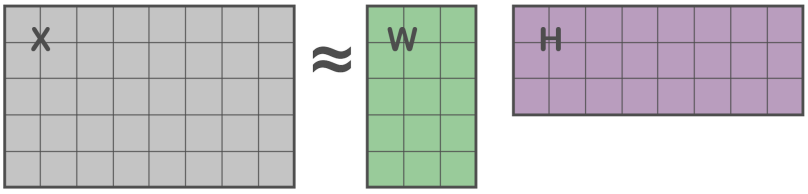

Abbildung 3.1: Veranschaulichung der Dimensionen einer Matrix Faktorisierung

Gemäß der Definition stellt die Matrix **W** die Basis und die Matrix **H** die Gewichtung dar. Umgelegt auf Audiosignale stellt die Basis **W** eine spektrale Gewichtung und **H** den zeitliche Amplitudenverlauf jeder Komponente dar. [2]

In vielen Papers wird die Faktorisierung anstatt von $\mathbf{X} = \mathbf{WH}$ mit $\mathbf{X} = \mathbf{As}$ beschrieben, wobei **A** die Mixing-Matrix und **s** die zu rekonstruierenden Quellen darstellt.

Generell stellt die Matrix Faktorisierung eine Lineare Transformation dar, die unterschiedlicher Ordnung sein kann. Hierbei fällt die PCA in eine Lineare Transformation 2. Ordnung und die NMF und die ICA in eine Transformation höherer Ordnung. Deshalb wird sich in der Implementierung zwischen den beiden Gruppen ein Unterschied ergeben, da die 2.Ordnung leichter erreicht werden kann.

Eine Übersicht über die Arbeitsschritte einer Matrix Faktorisierung bietet die Darstellung in Abbildung 3.2.

In dieser Arbeit wird von einem in den Frequenzbereich transformierten Quellen-Mix ausgegangen. Zusätzlich sind Trainingsdaten, die ebenfalls faktorisiert werden, als Input sinnvoll. Beispielsweise können bei einer Quelltrennung eines Klaviers und eines Schlagzeugs (wobei das Schlagzeug aus der Audiospur entfernt werden soll) eine bis zwei Komponente mit einem Klavier-File, und die anderen Komponenten mit jeweils einem Bass-Drum-, Hi-Hat- und Snare-File gefüllt werden und in dieser Form als Initialisierungsmatrix fungieren.
Oder das Spektrum eines Instruments wird pro Halbton in einer Bibliothek gespeichert und als Maske verwendet, wobei nur die Frequenz-Bins innerhalb der Maske im Algorithmus upgedatet werden und die Bereiche außerhalb Null gesetzt werden (wodurch bei Multiplikation nichts verändert wird).

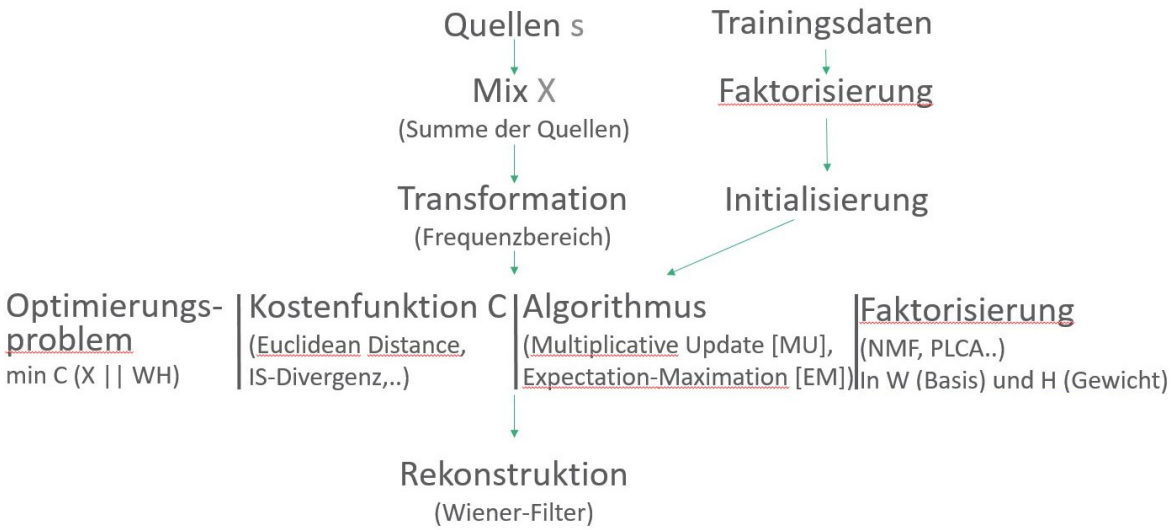

Abbildung 3.2: Signalfluss einer Matrix Faktorisierung

Das Optimierungsproblem ergibt sich durch die Forderung, dass $\mathbf{W} \cdot \mathbf{H}$ gegen $\mathbf{X}$ konvergieren soll. Diese Konvergenz soll über eine Kostenfunktion C erreicht werden. Die beiden Matrizen W und H werden iterativ solange über eine Update-Rule aktualisiert, bis die Funktion ausreichend konvergiert. (Wenn ein Algorithmus wie beispielsweise der Multiplicative-Update Alogorithmus ein konvexes Problem darstellt, ist das Finden eines globalen Optimums garantiert.)
Dabei gilt es zwischen verschiedenen Kostenfunktionen und Algorithmen auszuwählen. Prominent unter den Kostenfunktionen sind die sogenannten β-Divergenzen, zu denen die Euklidische Distanz, die Kullback-Leibler Divergenz und die Itakuro-Saito Divergenz zählen [3, S. 797]. Letztere wird üblicherweise für Audiosignale verwendet, da sie für hohe Dynamikunterschiede in Signalen gut geeignet ist. Bei den Algorithmen wären mögliche Alternativen zum verwendeten Multiplicative-Update Algorithmus (MU) [1, S. 3] noch der Estimation-Maximation (EM) Algorithmus, die Maximum-Likelihood Methode und der Gradient-Descent Algorithmus, ein Algorithmus basierend auf Non-Negative Least Squares. In Abbildung 3.3 [3] wird die Laufzeit der ersten beiden Algorithmen für verschiedene Werte für K (Anzahl der Komponenten) verglichen. (Folgende Abkürzungen werden verwendet: EUC = Euklidische Distanz, KL = Kullback-Leibler, IS = Itakuro-Saito.)

K	1	2	3	4	5	10
EUC-NMF	17	18	20	24	27	37
KL-NMF	90	90	92	100	107	117
IS-NMF/MU	127	127	129	135	138	149
IS-NMF/EM	81	110	142	171	204	376

Abbildung 3.3: Laufzeiten in Sekunden für 1000 Iterationen für Piano Input Daten mit einer Dauer von 16 s

Ersichtlich ist beim Vergleich des MU- und des EM-Algorithmus, dass ab einer Komponenten Anzahl $K = 3$ der MU-Algorithmus schneller konvergiert und somit geeigneter ist, da in den meisten Fällen in mehr als drei Komponenten zerlegt wird.

3.1 NMF

Generell basiert die NMF auf der Dekomposition des Audio Spektrums in die elementaren spektralen Patterns und zeit-abhängigen Gains. Die wichtigste Eigenschaft der Non-negative Matrix Factorization ist die Nicht-Negativität der Elemente (im Unterschied zu den meisten anderen Verfahren). Diese Eigenschaft trifft auch auf Audiosignale im Frequenzbereich zu, da üblicherweise nur das positive Spektrum verwendet wird, und spricht demnach für die Verwendung der NMF für die Faktorisierung von Audiosignalen.

Für die Implementierung in Matlab sollen verschiedene Kostenfunktionen ausgewählt werden können. Ziel ist es u.a. folgende β-Divergenzen (wobei β=0: IS-Divergenz, β=1: KL-Divergenz, β=2: Euklidische Distanz) zu implementieren.

$$d_\beta(x|y) = \begin{cases} \frac{1}{\beta(\beta-1)}(x^\beta + (\beta-1)y^\beta - \beta x y^{\beta-1}) & \beta \in \mathbb{R} \setminus \{0,1\} \\ x \log \frac{x}{y} + (y-x) & \beta = 1 \\ \frac{x}{y} - \log \frac{x}{y} - 1 & \beta = 0 \end{cases} \tag{3.1}$$

Mit Hilfe der folgenden iterativen Update-Rules aus [3, S. 798] ist es möglich die obengenannten Divergenzen und jeden anderen Wert völlig fexibel über β zu steuern.

$$\begin{aligned} \mathbf{H} &\leftarrow \mathbf{H}.\frac{\mathbf{W}^T((\mathbf{WH})^{.[\beta-2]}.\mathbf{X})}{\mathbf{W}^T(\mathbf{WH})^{.[\beta-1]}} \\ \mathbf{W} &\leftarrow \mathbf{W}.\frac{((\mathbf{WH})^{.[\beta-2]}.\mathbf{X})\mathbf{H}^T}{(\mathbf{WH})^{.[\beta-1]}\mathbf{H}^T} \end{aligned} \tag{3.2}$$

Diese sind in dieser Form auch im Code implementiert:

```
for ii = 1:param.iter
    H = H.*(W.'*((WH).^(param.beta-2).*X)./(W.'*(WH).^(param.beta-1)));

    WH = W*H; % update actual esstimator

    W = W.*(((WH.^(param.beta-2).*X)*H.')./((WH.^(param.beta-1)*H.')));

    [W, H] = normNmfMU(W, H, param.K);  % normalize

    WH = W*H; % update with normalized versions of W and H
end
```

Wichtig ist, dass zwischen jedem Schritt der Estimator upgedatet wird und am Ende jeder Iteration normalisiert wird [3, S. 805], um Skalierungs Unbestimmtheiten zu verhindern und damit in weiterer Folge die Kostenfunktion nicht zu verändern.

Dieser NMF-Algorithmus enthält allerdings keine Constraints beispielsweise bezüglich der Phase oder der Kompaktheit, welche eine zielgerechtere Separation für die jeweiligen zu trennenden Signaltypen ermöglichen wurde. Im Rahmen dieses Projektes wurden NMF-Algorithmen enthalten jedoch außer Acht gelassen, die Implementierung dieser würde sich als Weiterführung dieser Arbeit anbieten.

3.2 PCA

Die Hauptkomponenten Analyse versucht die Daten als Linear Kombinationen der Basisvektoren auszudrücken. Mit den originalen Daten $\mathbf{X}$ und einer Transformationsmatrix $\mathbf{P}$ erhält man $\mathbf{Y}$,eine neue Repräsentation der Daten.

$$\mathbf{PX} = \mathbf{Y} \tag{3.3}$$

Die PCA stellt ebenso wie die Diskrete Fourier Transformation (DFT) und die Diskrete Cosinus Transformation (DCT) eine orthogonale Transformation dar.

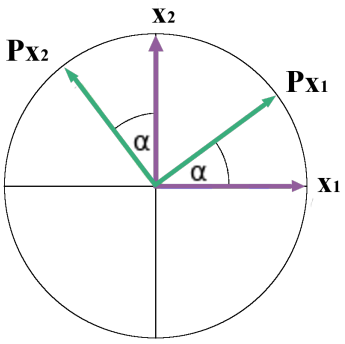

Abbildung 3.4: Orthogonale Transformation von $\mathbf{x}$ durch $\mathbf{P}$

Eigenschaften dieser Transformation sind die Tendenz zu Dekorrelieren und die Neuverteilung der Signalenergie, welche in einer Anhäufung der Signalenergie in einer geringeren Anzahl von Komponenten resultiert. Im Frequenzbereich konzentriert sich die Energie dabei hauptsächlich im Gleichanteil und den'Fundamental Frequencies'.

Geometrisch interpretiert ist $\mathbf{P}$ eine Rotations- bzw. Dehnungs-Matrix, die $\mathbf{X}$ in $\mathbf{Y}$ transformiert. Die Hauptkomponenten entsprechen dann den Reihen der Matrix $\mathbf{P}$ [5]. Die Hauptkomponenten können als Richtungen der größten Varianz interpretiert werden, wobei die Anzahl der Hauptkomponenten der Größe der Daten-Dimension entspricht. Außerdem müssen die Komponenten orthogonal zueinander sein und jeweils eine gauss'sche Verteilung aufweisen (siehe Abbildung 3.5). Letzteres muss gelten, da

die PCA annimmt, dass die Verteilung der Daten entlang der Achsen über die Suffiziente Statistik beschrieben wird, gemäß der die beiden Schätzer Mittelwert und Varianz die gesamte Wahrscheinlichkeitsverteilung beschreiben. Da die Gauss'sche Verteilung die einzige mittelwertfreie und vollständig über die Varianz beschriebene Verteilung ist, müssen die Komponenten also eine gauss'sche Verteilung aufweisen.

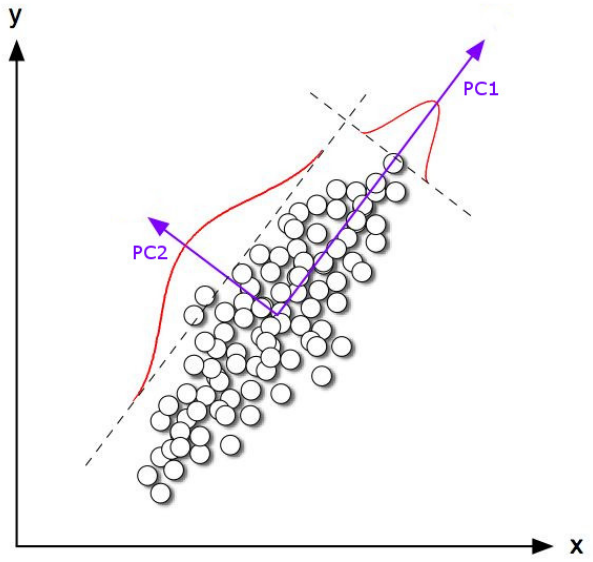

Abbildung 3.5: Richtungen der ersten beiden Hauptkomponenten[1]

Durch die Analyse wird das Koordinatensystem transformiert, als würde man aus einem anderen Winkel auf die Daten schauen, wobei die Daten auf die Hauptkomponenten Achsen projiziert werden (siehe Abbildung 3.6).

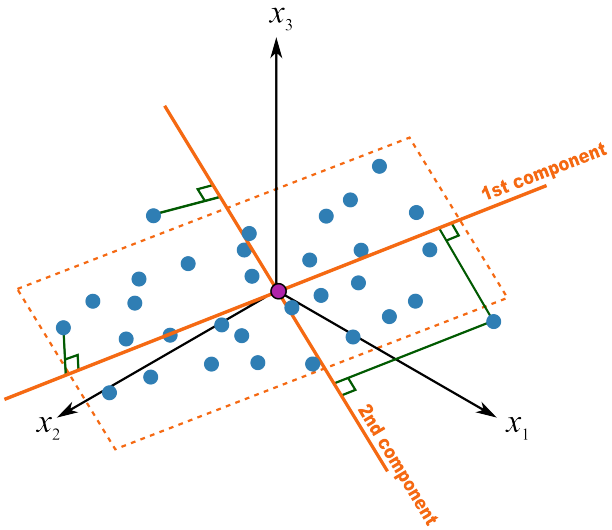

Abbildung 3.6: Projektion der Datenpunkte auf die Hauptkomponenten Achsen[2]

[1] Abbildung 3.5: https://lazyprogrammer.me/tutorial-principal-components-analysis-pca/

Die Hauptkomponenten können aber auch aus den Eigenvektoren der Kovarianzmatrix $\mathbf{S}$ von $\mathbf{X}$ berechnet werden [5, S. 6], wobei n der Spaltenanzahl entspricht.

$$\mathbf{S}(Y) = \frac{1}{n-1}\mathbf{Y}^T\mathbf{Y} \tag{3.4}$$

Der zugehörige Eigenwert gibt die Stärke der Varianz bzw. der kleinsten Korrelation und somit die Wertigkeit der Hauptkomponente an.

Ein wichtiger Begriff ist die bereits erwähnte Kovarianzmatrix, deren Elemente ein Maß der Korrelation zwischen den verschiedenen Datenwerten darstellen, wobei entlang der Diagonale die Varianz aufgetragen ist.
Anzumerken ist, dass die Korrelation immer zwischen zwei Dimensionen gemessen wird. Untersucht man beispielsweise den Zusammenhang zwischen der Nutzung des Smartphones, der Nutzung des Internets und dem Alter, könnte beispielsweise ein Eigenwert niedrig sein, da die Nutzung des Internets nicht vom Alter abhängt.

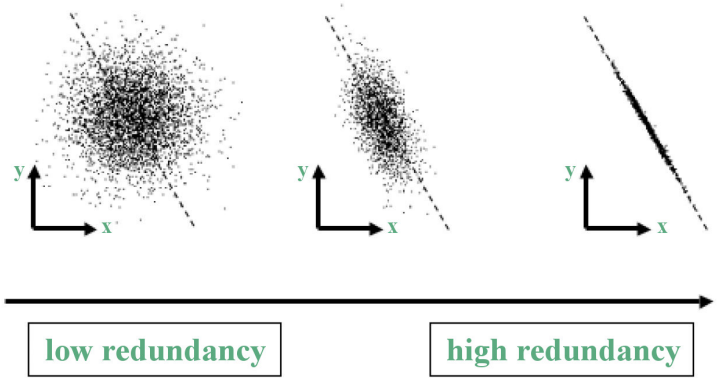

Abbildung 3.7: Mögliche Redundanzen einer 2D Daten-Matrix

In vielen Papers wird auf die Wichtigkeit der Mittelwertbefreiung vor der Berechnung hingewiesen, wobei dieser Schritt für die Bearbeitung von Audio Signalen im Frequenzbereich nicht sinnvoll ist, da dadurch auch negative Werte im Spektrum auftauchen können.
In Abbildung 3.8 soll dargestellt werden, dass für den nicht mittelwertbefreiten Fall die Richtungen der Hauptkomponenten nicht mit den Achsen übereinstimmen.

[2] Abbildung 3.6: http://learnche.org/pid/latent-variable-modelling/principal-component-analysis/interpreting-score-plots-and-loading-plots

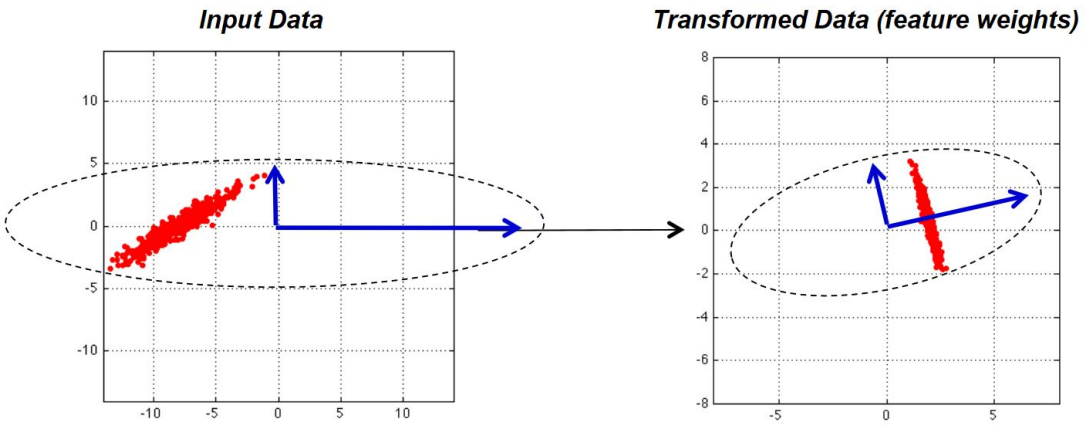

Abbildung 3.8: Transformation des Inputs für nicht mittelwertbefreite Daten

Die Hauptanwendung der PCA ist die Datenreduktion [6, S. 109, S. 119]. Hierbei werden die Eigenvektoren nach Größe des zugehörigen Eigenwertes sortiert und anschließend nur eine festgelegte Anzahl oder jene Eigenvektoren, deren Eigenwerte über einem bestimmten Threshold liegen, behalten. Dieses Prinzip kann auch für die Source Separation verwendet werden, indem nur jene Eigenvektoren behalten werden, die für die Separation relevant sind.

Zusammengefasst besteht die übliche Implementierung der PCA in der Berechnung der Eigenwerte und Eigenvektoren der Kovarianzmatrix, wobei die Eigenwerte die Richtung größter Varianz anzeigen und die Eigenvektoren als Basisvektoren den transformierten Raum aufspannen.

Eine andere Möglichkeit der Durchführung der Hauptkomponenten-Analyse besteht darin, eine Singulär-Wert-Zerlegung durchzuführen, wie im Fall der Factorization Toolbox. Hierbei wird die Mixing-Matrix in

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}'$$

aufgespalten. Mit dieser linearen Operation lässt sich jede Matrix durch eine Rotation $\mathbf{U}$, eine Dehnung auf einer Achse (Diagonalmatrix $\mathbf{S}$) und noch einer Rotation V beschreiben, wobei $\mathbf{U}$ und $\mathbf{V}$ orthogonal sind (quadratisch und reell, Zeilen- und Spaltenvektoren orthonormal bezüglich des Standardskalarprodukts).

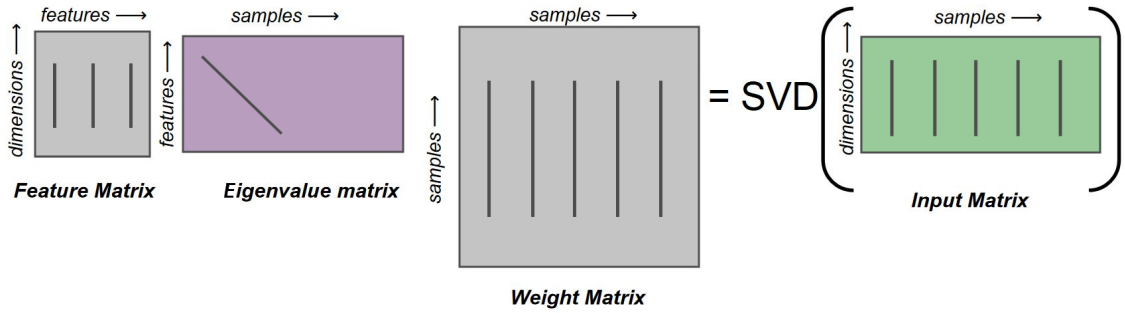

Abbildung 3.9: Veranschaulichung der zusammengesetzten Matrizen bei der SVD

Die Zusammensetzung dieser 3 Matrix Operationen ist der Operation durch **A** gleichgesetzt.

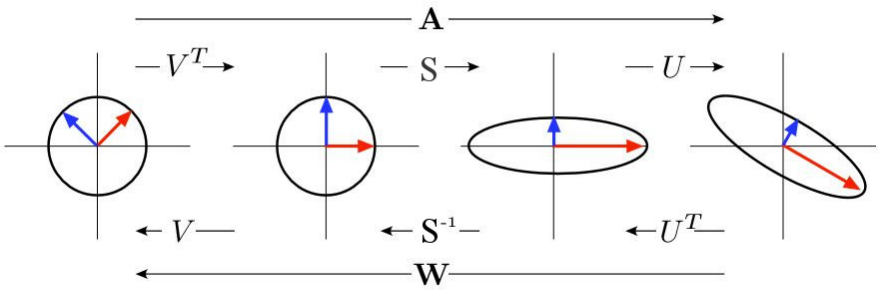

Abbildung 3.10: Grafische Beschreibung der Singulär-Wert-Zerlegung [7, S. 5]

Die SVD (Singular-Value-Decomposition) wurde implementiert, um in die gewohnten Matrizen **W** und **H** faktorisieren zu können.

```
% compute with SVD
[U, S, V] = svd(X,0);

% sort by eigenvalues in decreasing order and choose only eigenvalues
% over threshold which is defined in param.K
idx = find(diag(S)<param.K,1,'first');
S(idx:end,idx:end) = 0; % only PCs stay in matrix

% compute W and H
fact_out.W = U * S;
fact_out.H = V';
```

Ein wichtiges Paper vor allem für den Vergleich und das Verständnis des Zusammenhangs von NMF und PCA ist [4].

3.3 ICA

Die Theorie dieses Kapitels entstammt hauptsächlich aus [7], [6] und [8]. Das der ICA zugrundeliegende Modell der ICA ist

$$\mathbf{X} = \mathbf{A} * \mathbf{s}, \tag{3.5}$$

wobei $\mathbf{X}$ der Daten-Matrix, $\mathbf{A}$ der Mixing-Matrix und $\mathbf{s}$ den Quellen entspricht. Das Ziel dieser Faktorisierung ist somit das Finden einer Unmixing-Matrix $\mathbf{W} = \mathbf{A}^{-1}$. Diese sollte quadratisch sein und einen vollen Rang besitzen.
Üblicherweise werden auch hier die Daten mittelwertbefreit. Der nächste Schritt ist das sogenannte „whiten“ der Daten, bei dem eine PCA durchgeführt wird um Korrelationen zu entfernen.

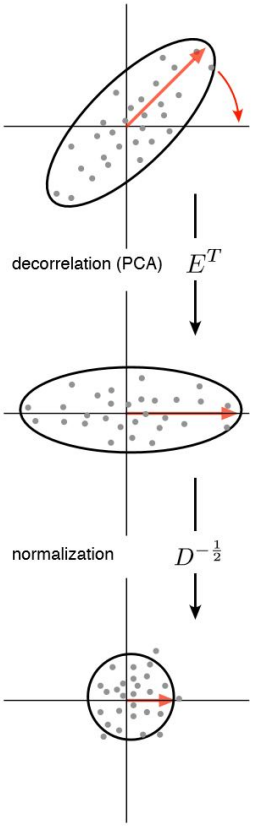

Abbildung 3.11: Grafische Beschreibung des „Whitenings“ [7, S. 6]

Die Komponenten müssen statistisch unabhängig sein und dürfen bis auf eine Komponente keine gauss'sche Verteilung besitzen [8]. Wären die Komponenten alle gauss' verteilt mit symmetrischen Verteilungsfunktionen, wäre keine Information über die Richtung der Spalten der Matrix A enthalten. Anzumerken ist, dass die Unabhängigkeit die zweite Ordnung und die stärkere Variante der Dekorrelation darstellt. Sind Werte unabhängig, sind sie automatisch dekorreliert, aber nicht umgekehrt.

Im Unterschied zur PCA werden nicht die Richtungen der maximalen Varianz als Basis für die Quelltrennung verwendet. Außerdem müssen die einzelnen Komponenten nicht orthogonal zueinander sein.

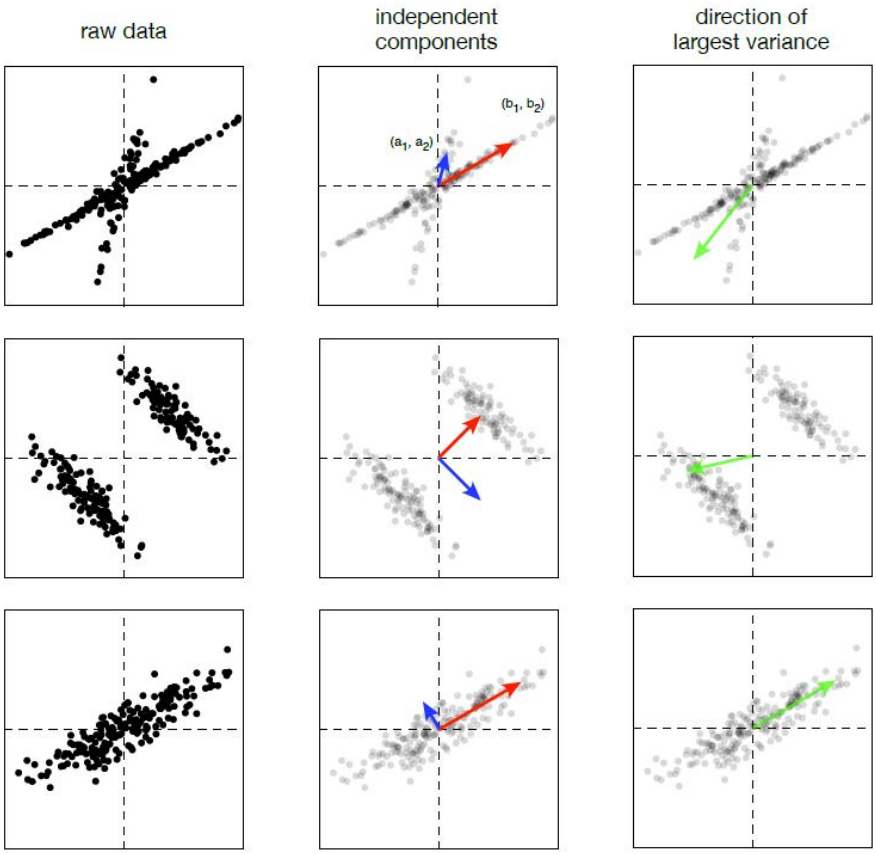

Abbildung 3.12: Analyse von linear gemixten Signalen [7, S. 3]

Methoden der ICA:

- Maximierung der Nicht-Gaussianität - ein entsprechendes Maß kann als Kurtosis (Gewölbtheit der Verteilungsfunktion) oder Negentropie (negative Entropie) ausgedrückt werden
- Gemeinsame Informationen Minimieren ('Minimizing Mutual Information')
- Maximum Likelihood Estimation (Vergleich Infomax Prinzip)

Die Implementierung ist angelehnt an [8] und somit den FastICA Algorithmus, in dem die Nicht-Gaussianität als Approximation der Negentropie gemessen wird. Für den Erwartungswert wird folgende nicht-quadratische Funktion g gewählt:

$$g(u) = tanh(a_1 u), \tag{3.6}$$

wobei meistens $a_1 = 1$ gesetzt wird.

Anmerkung: Der Unterschied zwischen Mittelwert und Erwartungswert liegt darin, dass der Mittelwert abhängig vom durchgeführten Zufallsexperiment ist und der Erwartungswert den Schwerpunkt der Verteilung darstellt. Dies soll anhand eines Würfel Beispiels gezeigt werden: Gewürfelt wird 1, 1, 2, 3, 6. Der Mittelwert ist demnach 2.6, der Erwartungswert liegt trotzdem bei 3.5.

Pre-Processing in der Implementierung:

```
% Centering
[X_m, mean_val] = remmean(X);

% Whitening
[D, E] = eig(cov(X_m'));  % compute eigenvector and eigenvalue
whiteningMatrix = inv(sqrt (D)) * E';
```

ICA Update-Algorithmus:

```
% ICA loop
for round = 1:param.iter
    % Symmetric orthogonalization
    B = B * real(inv(B' * B)^(1/2));

    Y = X_w' * B;
    hypTan = tanh(a1 * Y);  % g(u)
    Beta = sum(Y .* hypTan);
    D = diag(1 ./ (Beta - a1 * sum(1 - hypTan .^ 2)));  % D = diag(alpha)
    B = B + B * (Y' * hypTan - diag(Beta)) * D;

    % Calculate ICA filters
    W = real(B' * whiteningMatrix);    % W = A^(-1)
end

% Recover W and H
fact_out.H = W * X  + (W * X_m);
fact_out.W = pinv(W);
```

4 Toolbox Aufbau

Die Toolbox wird über die Funktion *facToolbox*() aufgerufen und mit Eingangsparametern aus vordefinierten Parameter-Sets gespeist.

4.1 Parameter Set

Das Parameter Set sollte folgende Definitionen enthalten:

```
% Parameters for read in audio files
read.max_dur = 20;              % for GUI Input; maximal duration in s
read.path = 'SampleLibrary/';  % path of audio files

% Optional for fixed paths
%read.fix.path_fix = 'SampleLibrary/Test_Piano2_Beat/';
%read.fix.name_fix = 'mix_piano2_beat';
%read.fix.path_init_fix = 'SampleLibrary/Test_Piano2_Beat/';
%read.fix.name_init_fix = 'mix_piano2_beat';

% Parameters for Signal Transformtation and Editing
sig.square = 1;  % [0,1] if signal in Frequency domain sould be squared
sig.F = 512;     % FFT size from 0Hz to fs/2
sig.nfft = (sig.F-1)*2;      %window size
sig.hop = sig.nfft/2;
sig.win = 'sinebell';

% Parameters for NMF
param_nmf.K = 5;                    % # of patterns
param_nmf.beta = 0;                 % type of Beta-Divergenz (0,1,2)
param_nmf.iter = 100;               % # of iterations
%param_nmf.rand_var = 123456;       % fixed random variable
param_fact.nmf = param_nmf;

% Parameters for PCA
param_pca.K = 0.001;
param_pca.iter = 10;
param_fact.pca = param_pca;

% Parameters for ICA
param_ica.iter = 10;
param_ica.K = 5;
param_fact.ica = param_ica;

% Flags
flag.fact_type = 'NMF';       % e.g. 'PCA', 'NMF', 'ICA'
flag.init_type = 2; % '1'->spectrogram; '2'->each component extra
flag.init_channel2 = 0; % just in case init_type = 2: '0' -> normal; '1'
   -> H matrix is initialized by 2nd channel which should be separated
   from the original file
```

```
flag.compare = 0;
flag.smooth = 1;
flag.merge = 1;
flag.eval_peass = 0;
flag.eval_transform = 0;
flag.converg_dev_check = 1; % render convergence development in loop
```

Vorgeschlagene Ordnerstruktur (im Matlab Projektordner):

Um einen erfolgreichen Programm Durchlauf zu garantieren sollten die beiden Ordner → evaluation (in dem unter → Soundfiles die Evaluationsergebnisse gespeichert werden) und → SampleLibrary angelegt werden. In Zweiterem sollen Test-Case Ordner wie z.B. → Test_Piano2_Beat erstellt werden. Diese sollen jeweils ein zu separierendes Audiofile und ein Initialisierungs Audiofile bzw. für den Fall einer Komponenten Initialisierung (siehe Abschnitt 4.3.2) stattdessen einen → Component_init Ordner mit Initialisierungs Audiofiles enthalten.

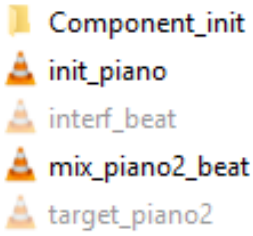

Abbildung 4.1: Ordnerstruktur im SampleLibrary Ordner, wobei die nur für die Evaluierung mit PEASS notwendigen Dateien ausgegraut dargestellt sind

Ordnerpfade und Dateinamen:

- Fix im Parameter-Set Dateinamen und Ordnerpfade vorgeben oder
- Auswahl des Ordner Pfads und der Dateien per GUI, hierbei wird der ausgewählte Pfad mit gespeichert

Audiofile:

- Maximale Länge der eingelesenen Audiofiles kann im Parameter-Set festgelegt werden (momentan 30 Sekunden)

Signal Transformation:

- Spektrogramm quadriert (`sig.square`: '0' nicht quadriert; '1' quadriert)
- Window Size (`sig.F`, `sig.nfft`)
- Fensterfunktion (`sig.win`)
- Hop-size (`sig.hop`)

NMF Parameter:

- Anzahl an Komponenten (`param.K`)

- Art der Distanz/Divergenz Messung (`param.beta`)
- Anzahl an Iterationen (`param.iter`)
- Fixierte Zufallszahlen (`param.rand_var`)

Flags:

- Faktorisierungstyp (`flag.fact_type`: 'NMF'; 'PCA'; 'ICA')
- Initialisierungstyp (`flag.init_type`: '1' -> Initialisierung per Spektrogramm des Init Inputs; '2'-> Initialisierung über gemittelte Frequenzvektoren über K viele Komponenten)
- Initialisierung über den Gain eines zweiten Kanals, nur bei `flag.init_type`=2 (`flag.init_channel2`: '0' -> Aus; '1' -> Ein)
- Amplituden Glättung als Nachbearbeitungs Schritt
- Mergen von verschiedenen separierten Komponenten (`flag.merge`: '0' -> Aus; '1' -> Ein)
- Evaluierung nach PEASS (`flag.eval_peass`: '0' -> Aus; '1' -> Ein)
- Evaluierung der Transformation (`flag.eval_transform`: '0' -> Aus; '1' -> Ein)
- Überprüfung der Konvergenz Entwicklung (`flag.converg_dev_check`: '1' -> Konvergenz Entwicklung soll in der NMF Berechnungs-Schleife mit gerendert werden; '2' -> Konvergenz wird nur am Ende der Berechnungen bestimmt)

Um eine Veränderung des Ergebnisses der Faktorisierung zu erreichen kann an den Parametern der Signal Transformation, an K und β der NMF (bzw. PCA, ICA) und an der Auswahl des Initialisierungs Typs (bei Komponenten Initialisierung zusätzlich Entscheidung über Initialisierung per zweitem Kanal) geschraubt werden. Einen großen Einfluss hat die Auswahl der Initialisierungs Audiofiles.
Die Wahl der Fenstergröße und Form für die STFT spielt eine überraschend große Rolle.

4.2 Strukturierung

Die Toolbox ist in folgende Teile gegliedert:

- Einlesen der Eingangsdaten
- Auswahl der Trainingsdaten
- Transformation in den Frequenzbereich
- Faktorisierung
- Plots zur Analyse für den User
- Separation and Rekonstruktion

- Post-Processing: Amplituden Glättung
- Evaluierung

Im den ersten drei Abschnitten werden das Eingangssignal und die Trainingsdaten per $readAudiofile()$ eingelesen sowie in $audioTransform()$ per STFT in den Frequenzbereich transformiert und optional quadriert. Außerdem werden in $audioTransform()$ die Absolutbeträge gebildet und Zeit- bzw. Frequenzvektoren für zukünftige Plots generiert.

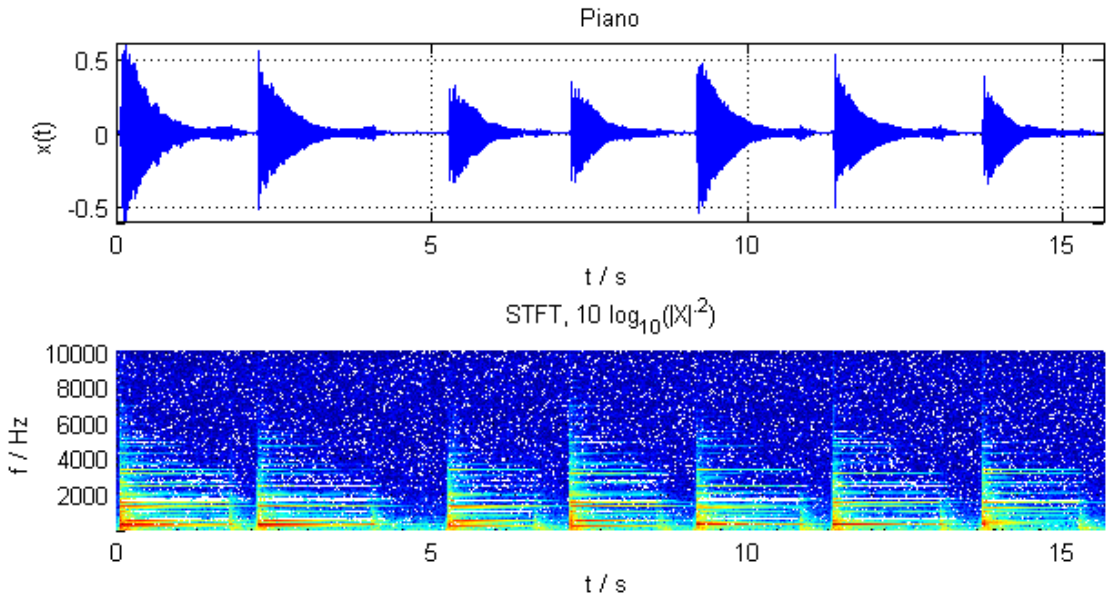

Abbildung 4.2: Eingangssignal im Zeit- und Frequenzbereich

Der folgende Bereich stellt den Kern der Toolbox, die Faktorisierung, dar. Diese werden abhängig von der definierten Faktorisierungs Art per $calcNmf()$, $calcPca()$ oder $calcIca()$ berechnet. Auf die Initialisierung, die für den Update-Algorithmus im Fall der NMF verwendet wird, wird später genauer eingegangen. [1]

Im Anschluss an die Faktorisierung werden die Patterns bestehend aus den Zeilen bzw. Spalten der Matrizen W und H geplottet (siehe Abbildung 4.3). In weiterer Folge wird der User der Toolbox gebeten jenes Pattern auszuwählen, welches das zu eliminierende Siganl am besten beschreibt und jenes das das zu separierende Signal darstellt, auszuwählen.

[1] An dieser Stelle soll festgehalten werden, dass im Prozess der Entstehung der Toolbox beschlossen wurde den Fokus auf die NMF zu legen. Grund waren die bis dato schlechten Ergebnisse für PCA bzw. die ICA. Außerdem wurde in keinem Paper diskutiert wie die PCA und ICA Spektren als Eingangsdaten verwenden.

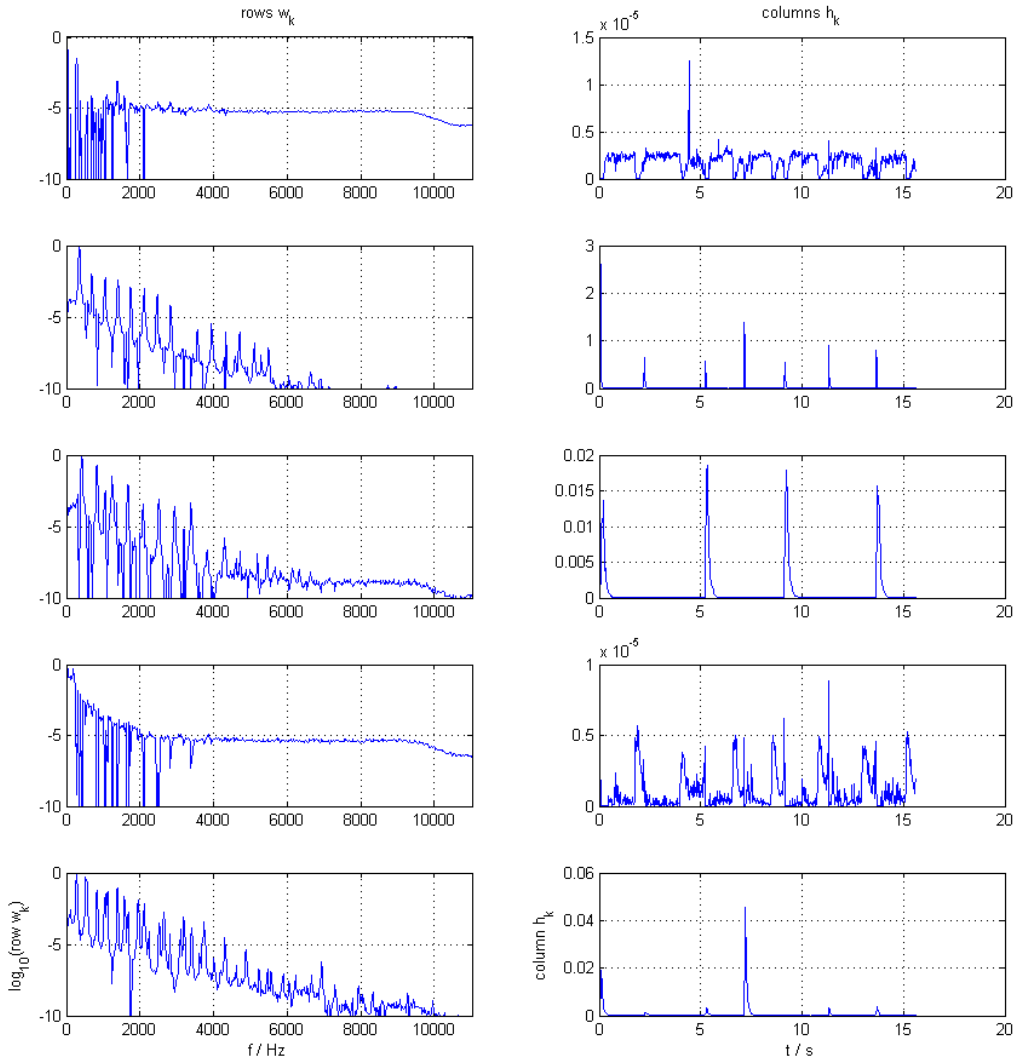

Abbildung 4.3: Plot der die faktorisierten Signale zeigt, wobei die linke Spalte den Spalten von **W** und die rechte Spalte den Zeilen von **H** entspricht (Vergleiche mit [3])

Zusätzlich wird der Konvergenzverlauf, der Unterschied zwischen **X** und der geschätzen Matrix **WH**, über die Iterationen geplottet. Ersichtlich ist, dass für dieses Beispiel schon nach 25 Iterationen die Konvergenz erreicht wird.
Für die Verwendung dieser Toolbox wird eine Wahl von 100 Iterationen empfohlen, um eine sichere Konvergenz zu gewährleisten.

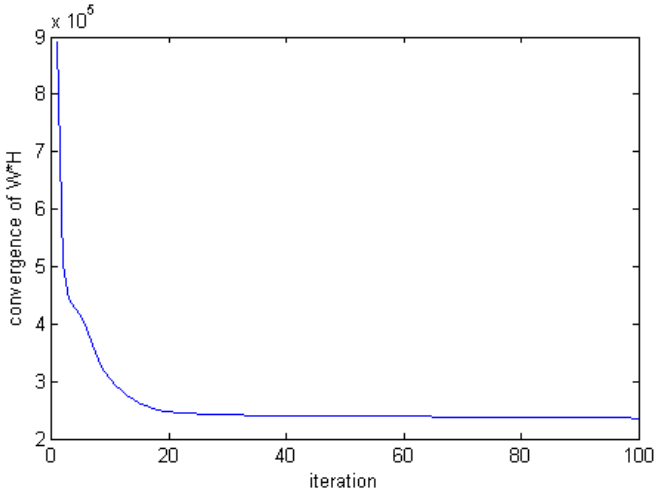

Abbildung 4.4: Konvergenzverlauf für die NMF

Im Fall der Faktorisierung durch PCA bzw. ICA werden die Datenpunkte über die ersten beiden Komponenten aufgetragen und geplottet, um die Stärke und Varianz ersichtlich zu machen.

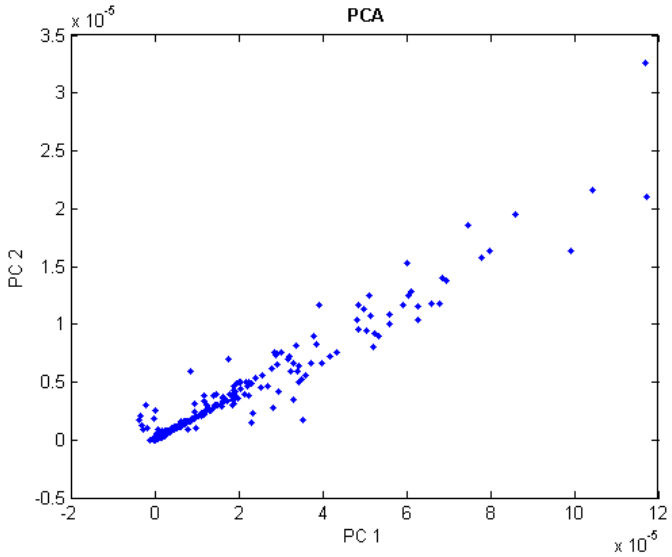

Abbildung 4.5: Plot der die Datenpunkte (Komponenten) für PCA zeigt

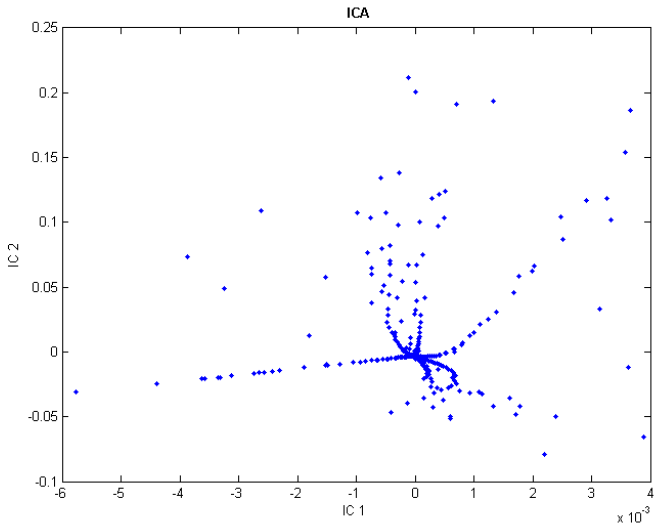

Abbildung 4.6: Plot der die Datenpunkte für ICA zeigt

Im vierten Teil werden die faktorisierten Matrizen zuerst per *cut*() in die einzelnen separierten Signale unterteilt und anschließend in *reconstruct*() per Wiener-Gain und ISTFT wieder zusammengebaut und zurück transformiert. In *analyseSep*() werden dann je nach User-Input (der abhängig von den geplotteten Faktorisierungsergebnissen erfolgen soll) die separierten Zeitsignale geplottet und abgespielt. Diese werden jeweils im Test Case Ordner unter dem Namen 'estimate.wav' abgespeichert.

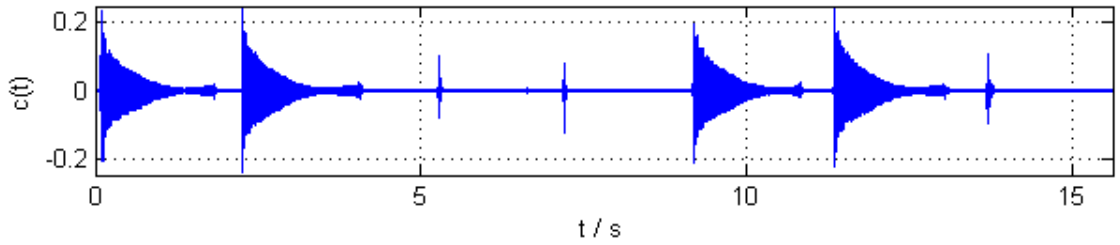

Abbildung 4.7: Separiertes ausgewähltes Signal im Zeitbereich und als Spektrogramm

Zusätzlich gibt es die Möglichkeit die separierten Signale in verschiedenen Konstellationen zusammensetzen zu lassen und sich dieses anzuhören. Hierfür muss das flag $merge = 1$ gesetzt werden (per Default 0). In diesem Fall wird nur das zusammengesetzte Audiofile unter dem Namen 'estimate_merged.wav' gespeichert.

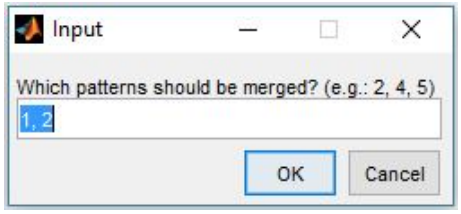

Abbildung 4.8: Abfrage welche Files zusammengesetzt werden sollen, wenn das *merge*-Flag gesetzt ist

Als Post-Processing Schritt kann bei gesetztem Flag $flag.smooth = 1$ die Amplitude geglättet werden (siehe Abschnitt 4.4).

Abschließend kann bei gesetztem Flag eine der beiden implementierten Evaluierungs Varianten ausgewählt werden, die im Kapitel 5 beschrieben werden.

4.3 Initialisierung

4.3.1 Initialisierung per Zufallszahlen

Für die Initialisierung der Initialisierung per $randn()$ wurden verschiedene Varianzen und Mittelwerte getestet, die in der Toolbox in $initNmf.m$ enthaltene Version brachte schließlich die besten Ergebnisse:

```
W_init = abs(randn(F,param.K)) + ones(F,param.K);
H_init = abs(randn(param.K,N)) + ones(param.K,N);
```

4.3.2 Komponentenweise Initialisierung

Bis jetzt wurde die Initialisierung über eine Matrix (Spektrogramm) mit einem Initialisierungs File, das dem gewünschten zu separierenden Signal möglichst ähnlich ist, durchgeführt. Getestet wurde nun eine Initialisierung mit je einem Vektor für eine Komponente der W (und/oder eventuell auch der H Matrix), um zielorientiertere Pattern-Aufteilungen zu erhalten. Dafür muss das Flag flag.init_type = 2 gesetzt werden und es sollte im selben Testcase Ordner, in dem auch die zu faktorisierende Datei liegt, ein Ordner mit dem Namen “Component_init“ erstellt werden. In diesem sollen so viele Input Files wie es Komponenten geben soll liegen, da die ersten K Dateien verwendet werden. Für das Piano_Beat Beispiel könnten das beispielsweise ein Bass Drum Sample, ein Hi Hat Sample, ein Shaker Sample eine einstimmige Melodie eines Pianos und Akkorde eines Pianos sein. Von jedem Sample wird das Spektrogramm berechnet und über die Zeit gemittelt. Aus diesem Grund ist auch die Zeitdauer des Signals egal, es empfiehlt sich aber ein eher kürzerer Ausschnitt, da spezifischere Frequenz Information der jeweiligen Instrumente mehr Effekt hat. (Eine Implementierung mit fixen Dateinamen ist nicht gegeben.)

Zuerst werden nur die einzelnen Zeilen der Matrix W mit den gemittelten Frequenzspektren von K Initialisierungsfiles gefüllt. Es wirkt jedoch so, als würden bei dieser Initialisierungsart die Ergebnisse zu 80 % auch durch Subtraktion im Frequenzbereich erreicht werden können. Es konnte noch nicht verifiziert werden, wie groß die Verbesserung gegenüber der Initialisierung mit einem File ist.

Für die Vorstellung, dass es auf einer Bühne 2 Quellen und 2 Kanäle gibt, erhält man sowohl für Kanal A als auch für Kanal B ein zu separierendes und ein interferierendes Signal. Betrachte ich eine Separation von Kanal A kann also der zu separierende Anteil in Kanal B als Initialisierung für die interferierenden Komponenten in Kanal A verwendet werden. Dadruch kann ausgenutzt werden, dass ich die Gain Funktion des interferierenden Anteils schon zur Gänze kenne.

Deshalb wurde anschließend auch eine dem entsprechende Initialisierung der H Matrix implementiert. Dieser Fall kann per flag.init_channel2 = 1 getestet werden (dafür muss flag.init_type = 2 gesetzt sein).
Hierbei wird im Code manuell eine Datei ausgewählt (die den übersprechenden Kanal simulieren soll, eventuell kann auch auf diesen Kanal eine NMF zur Separation der jeweiligen Hauptkomponente vorgenommen werden). Von dieser wird das Spektrogramm berechnet und über die Frequenzpins summiert, woraus ein Zeilenvektor resultiert. Je nachdem in wie viele interferierende Komponenten der Kanal A faktorisiert werden soll, soll dieser Vektor in dementsprechend viele Zeilen der Initialisierungsmatrix H_init geschrieben werden.
Mit einem Threshold werden dann die niedrigen Werte Null gesetzt, da aufgrund des multiplikativen Algorithmus' die H Matrix wie eine Schablone wirkt.

Achtung: Bei Komponentenweiser Initialisierung darf K nicht größer als die Anzahl der vorhandenen Dateien im Component_init Ordner sein.

```
% get direction and names of input files in Component_init\ folder
dirname = sprintf('%sComponent_init',flag.path);
dirfiles = dir(dirname);
ci_name = {dirfiles.name};
fix.path_fix = 'SampleLibrary/Test_Piano2_Beat/Component_init/';

% choose first 5 files in folder, computes spectrogram, sums up over
    time
% result: K frequency pattern vectors for initialization
for k = 1:param.K
    fix.name_fix = ci_name{k+2};     % +2 because dir() sets dots in
        the first two cells
    [audioWH.x, audioWH.fs, audioWH.bit, audioWH.name, read.path] =
        readAudiofile(read.max_dur,0, dirname, fix);
    [audioWH, ~] = audioTransform(audioWH.x(:,1), audioWH, sig);
    W_init(:,k) = sum(audioWH.X,2);%./size(audio.X,2);
end

if flag.init_channel2 == 1
    fix.path_fix = 'SampleLibrary/Test_Piano2_Beat/';
    fix.name_fix = 'interf_beat.wav';
    [init_H.x, init_H.fs, init_H.bit, init_H.name, read.path] =
        readAudiofile(read.max_dur,0, dirname, fix);
```

```
        [init_H, ~] = audioTransform(init_H.x(:,1), init_H, sig);
        H_init(1,:) = sum(init_H.X,1);  % depending on components and use
            case
        H_init(2,:) = sum(init_H.X,1);
        H_init(3,:) = sum(init_H.X,1);
        H_init(H_init < 0.02) = 0;      % Threshold
    end

    % initialize with audio file
    fact_out = calcNmf(H_init, W_init, X, param);
    W_init = fact_out.W;
    H_init = fact_out.H;
```

4.3.3 Initialisierungsfiles

Folgende Initialierungsfiles sind aktuell in der angelegten Source Separation Sample Library (`Sample Collection Init/`) zu finden, wobei die freien Samples entweder im Internet oder in der MSD100 Sammlung gefunden wurden.

- Piano Akkorde
- Piano Melodie mit Hall (Ursprünglich war die Idee auch Files mit viel Hall zu verwenden für die Komponentenweise Initialisierung zu verwenden, um Hallanteile extra als Pattern zu erhalten. Dies resultierte daraus, dass ohne Komponenten-Init. das Klavier immer in einen klaren und einen verschwommenen Anteil zerlegt wurde. Da aber Nachhall den gesamten Frequenzbereich beeinflusst, kann aus der separierten Komponente nicht viel Information gezogen werden.)
- Piano Tonleitern (Das Ergebnis ist eine Separation des Pianos in eine Tiefen betonte und ein Höhen betonte Komponente)
- Bass
- Verzerrte E-Gitarre
- Geige gestrichen
- Vocals männlich

Zusammengefasst sind Initialisierungsfiles mit eingeschränkterem Frequenzbereich sinnvoll.

4.4 Post-Processing

Für das Nachbearbeiten der Ergebnisses wurde eine Glättung der Phase sowie der Amplitude getestet. Diese sollte das Ergebnis vor allem für eine Trennung von perkussiven und harmonischen bzw. ausklingenden Signalen verbessern.
Die Notwendigkeit dieses Schrittes würde durch Einfügen von Constraints in den Algorithmus zum Teil wegfallen.

Da die Phase durch die Faktorisierung über das Betragsspektrum unbeachtet bleibt, bestand der erste Ansatz im linearen Interpolieren der Phase im Bereich der "Störungen".
Allerdings konnte keine relevante Korrelation von Unregelmäßigkeiten in der Phase und den Störbereichen gefunden werden und somit brachte die Implementierung keine wesentlichen Veränderungen mit sich.

Effektiver erwies sich die ebenfalls lineare Interpolation der Amplitude während der Störungen. Diese wurde, um einen zu artifiziellen Klang zu verhindern, nur für jene Frequenzbins angewendet, welche über das kumulieren der Amplitude als Harmonische ausgewählt wurden.

Als noch zielführender für die getesteten Beispiele erwies sich schlussendlich jedoch das Median-Filtern des gesamten Betragsspektrums über 15 Punkte.

Beispielhaft ist im Folgenden das grafische Ergebnis der Median-Filterung für einen Ausschnitt aus einem Audiofile mit separiertem Klavier und Restbeständen von Schlagzeug Samples zu sehen:

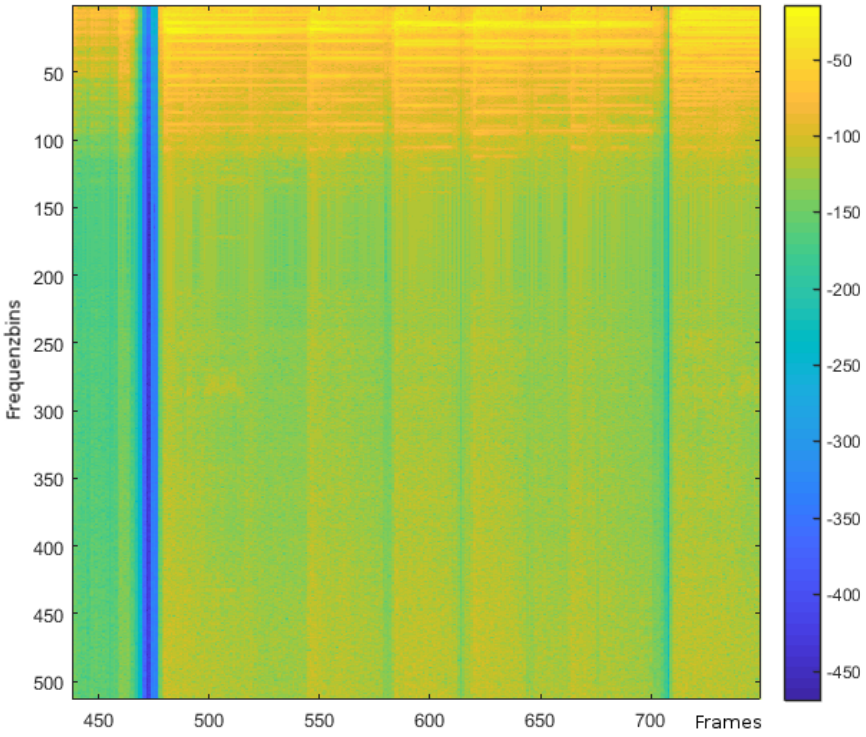

Abbildung 4.9: Ausschnitt aus einem nicht geglätteten Betragsspektrum

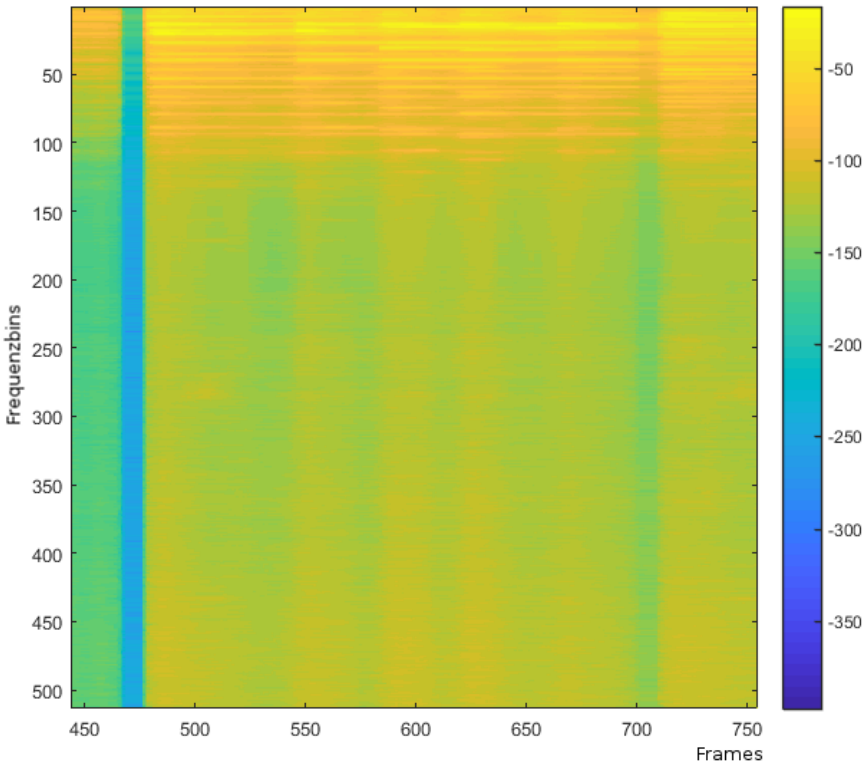

Abbildung 4.10: Selber Ausschnitt mit geglättetem Betragsspektrum

Vor allem die Artefakte der Faktorisierung beim 470. und 700. Frame (Abbildung 4.9) fallen störend in der Wiedergabe auf.

Nach der Glättung (Abbildung 4.10) ist sichtbar, dass die beschriebenen Bereiche wesentlich weniger stark ins Gewicht fallen. In Kauf genommen muss dafür eine Weichzeichnung der Anschläge im Klavier.

5 Evaluierung

Für die Evaluierung stehen zwei verschiedene Möglichkeiten zur Auswahl. Auf der einen Seite die Evaluation per PEASS-Toolkit (Perceptual Evaluation methods for Audio Source Separation), das eine Adaption des BSS-Eval-Toolkits (Blind Source Separation Evaluation) für Audio Signale ist. Eine Adaption ist notwendig, da die der BSS-Eval Berechnungen zugrunde liegenden Energie-Verhältnisse mit den subjektiven Ergebnissen nur bedingt übereinstimmen. Auf der anderen Seite kann eine Evaluation der Ergebnisse nach einem Vorschlag in [9, S. 74] durchgeführt werden, wobei hier die Wirkungsweisen der aneinandergereihten Transformationen analysiert werden.

5.1 PEASS-Toolkit

Für die Evaluierung per PEASS-Toolkit, das unter den Bedinungen der „GNU Public License version 3“ erhältlich ist[1], steht die Funktion *evaluation_peass*() zur Verfügung. Diese ist nach folgendem Schema aufgebaut:

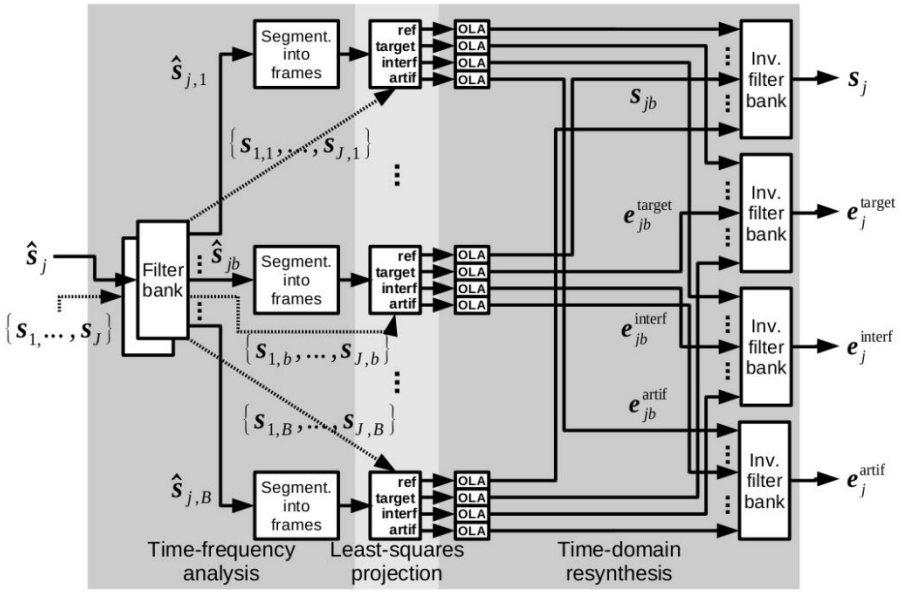

Abbildung 5.1: Block Diagramm des PEASS Toolkits für die Dekomposition der separierten Quelle in die Summe aus originaler Referenzquelle und den 3 Verzerrungskomponenten [10, S. 5]

Die state-of-the-art Herangehensweise, um die Unterschiede zwischen der separierten Quelle und der Referenz zu untersuchen, wäre das Herausstechen der gesamten Verzer-

[1]PEASS Software: http://bass-db.gforge.inria.fr/peass/PEASS-Software.html

rung beziehungsweise der Verzerrung, der Interferenz und der Artefakte der separierten Quelle zu messen. Das Ergebnis wären SDR (Signal to Distortion Ratio), ISR (Source Image to Spatial distortion Ratio), SIR (Signal to Interference Ratio) und SAR (Signal to Artifacts Ratio) [10, S. 7]. Da aber diese Energy-Ratios die tatsächliche auditive Perzeption nur mäßig wiedergeben, weil beispielsweise tiefe Frequenzen zu stark bemessen werden oder die Maskierung durch lautere Komponenten nicht mit einberechnet wird, werden andere Maße vorgeschlagen.

Mittels der auf auditive Wahrnehmung zugeschnittene PEMO-Q Metrik (Methode zur objektiven Bewertung von Audio Qualität unter Verwendung eines Modells für die auditive Wahrnehmung) werden e_j^{target}, e_j^{interf} und e_j^{artif} in q_j^{target}, q_j^{interf} und q_j^{artif} transformiert.

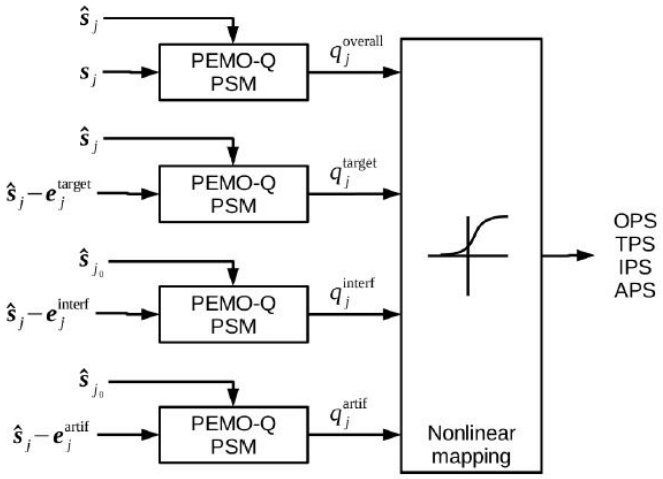

Abbildung 5.2: Block Diagramm des PEASS Toolkits für die Bestimmung der OPS, TPS, IPS und APS Kriterien [10, S. 8]

Anschließend werden die transformierten q-Komponenten nichtlinear kombiniert, um folgende Kriterien zu erhalten:

- Overall Perceptual Score (OPS)
- Target-related Perceptual Score (TPS)
- Interference-related Perceptual Score (IPS)
- Artifacts-related Perceptual Score (APS)

Für die Implementation der *evaluation_peass*() Funktion wurde das originale example.m File dahingehend verändert, dass die Audiofiles aus den eingangs ausgewählten Ordnern ausgelesen werden und auf korrekte Länge überprüft werden.

Folgende Funktionen werden in *evaluation_peass*() aus dem PEASS-Toolkit verwendet:

```
PEASS_ObjectiveMeasure.m
  extractDistortionComponents.m  % Decompose the distortion
                                 % into specific components
  ISR_SIR_SAR_fromNewDecomposition.m  % Compute ISR, SIR, SAR, SDR
                                      % from the estimated components
  audioQualityFeatures.m  % Compute quality features using PEMO-Q
  map2SubjScale.m         % Non-linear mapping to subjective scale
```

Um die Korrektheit des Evaluierungs-Tools zu überprüfen wurde als Input die Referenz auch als Ergebnis der Source Separation gewählt, wobei folgende Ergebnisse erzielt werden konnten:

```
The ISR, SIR, SAR and SDR criteria are:
 - SDR = Inf dB
 - ISR = Inf dB
 - SIR = Inf dB
 - SAR = Inf dB
The audio quality (PEMO-Q) criteria are:
 - qGlobal = 1.000
 - qTarget = 1.000
 - qInterf = 1.000
 - qArtif = 1.000
************************
****  FINAL RESULTS  ****
************************
 - Overall Perceptual Score: OPS = 99/100
 - Target-related Perceptual Score: TPS = 93/100
 - Interference-related Perceptual Score: IPS = 97/100
 - Artifact-related Perceptual Score: APS = 87/100
```

Für OPS, TPS, IPS und APS werden jedoch nicht 100% erreicht.

Im Vergleich dazu wurde ebenfalls der Mix als Ergebnis der Separation ausgewählt, um eine zweite Referenz zu erhalten:

```
The ISR, SIR, SAR and SDR criteria are:
 - SDR = 20.4 dB
 - ISR = 66.9 dB
 - SIR = 20.4 dB
 - SAR = 63.6 dB
The audio quality (PEMO-Q) criteria are:
 - qGlobal = 0.786
 - qTarget = 1.000
 - qInterf = 0.789
 - qArtif = 1.000
************************
****  FINAL RESULTS  ****
************************
 - Overall Perceptual Score: OPS = 8/100
 - Target-related Perceptual Score: TPS = 70/100
 - Interference-related Perceptual Score: IPS = 10/100
 - Artifact-related Perceptual Score: APS = 87/100
```

Vergleicht man diese beiden Ergebnisse, kann interpretiert werden, dass der OPS der maßgebendste Wert ist. Der TPS scheint zu zeigen wie präsent das Targaet File im Ergebnis ist, da der Wert für den ersten Fall höher aber dennoch für beide sehr hoch ist.
Der IPS bewertet mit dem Referenzfile für die Interferenz den Gehalt dieser im Ergebnis. Die Artefakte im Ergebnis scheinen in beiden Fällen sinnvollerweise gleich bewertet zu werden.

5.2 Evaluierung nach Nagathil

Die zweite Evaluations-Variante über die $evaluation\_transform()$ Funktion geht mehr auf die Kombinationen der verschiedenen Transformationen ein. Zum Beispiel STFT-NMF, STFT-PCA, STFT-ICA oder CQT(const. Q Transformation)-NMF, CQT-PCA, CQT-ICA.

Kriterien sind „Spectral Sparsity“, „Data Compactness“ und „Temporal Continuity“.

Für die Berechnung des „Spectral Sparsity“ Maßes wird der Gini-Index verwendet, ein statistisches Maß zur Berechnung der Ungleichheitsverteilung.

$$\phi_{SS}(\kappa) = \frac{1}{T} \sum_{\tau=1}^{T} gini|Y(\kappa, \tau)| \tag{5.1}$$

Ein hohes Maß an 'spectral sparsity' gilt für $\phi_{SS} \to 1$. Für einen Gini-Index von 0 sind die Daten gleichförmig verteilt, für einen Wert von 1 ist die Daten Matrix maximal spärlich und nur ein Koeffizient ist nicht Null.

„Data Compactnes“:

$$\phi_{DC}(\kappa) = (N - \alpha\kappa T)/(N + \alpha\kappa T) \tag{5.2}$$

Für $\phi_{DC} = 0$ wurden die Daten nicht komprimiert.

„Temporal Continuity“:

$$\phi_{TC}(\kappa) = 1 - \frac{1}{\kappa} \sum_{\zeta=1}^{\kappa} \hat{Y}_{mod}(\kappa), \tag{5.3}$$

wobei $\hat{Y}_{mod}(\kappa)$ wie in [9, S. 76] beschrieben berechnet wird. Für $\phi_{TC}(\kappa) \to 1$ ist die 'temporal continuity' groß.

Die Variablen Namen sind zum Teil auch aus [11], wobei aber die Berechnungen teilweise fehlerhaft sind und in [9] geändert wurden.

„The quality of the melody reconstruction is evaluated by the source separation measures implemented in the MATLAB bss eval toolbox which compute the signal-to-distortion ratio (SDR), the signal-to-interference ratio (SIR) and the signal-to-artifacts ratio (SAR).“ [11]

„At the same time, the STFT-based transforms ensure the highest degree of leading voice preservation and accompaniment attenuation as reflected by source separation measures.“ [11, S. 93]

Das Resultat für das selbe Beispiel, das für die erste beschriebene Evaluierungsvariante verwendet wurde:

```
phi =
   TC:  0.5181
   SC:  0.7789
   DC:  0.9807
```

Verglichen mit den Ergebnissen die Nagathil [9] für den Fall „stftNMF“ zeigt (siehe Abbildung 5.3), stimmen die Resultate für die „Spectral Compactness“ und die „Data Compression“ gut überein. Das Ergebnis für die „Temporal Continuity“ jedoch weißt völlig andere Werte auf. Vergleicht man allerdings dieses Resultat mit [11], passt das Ergebnis.

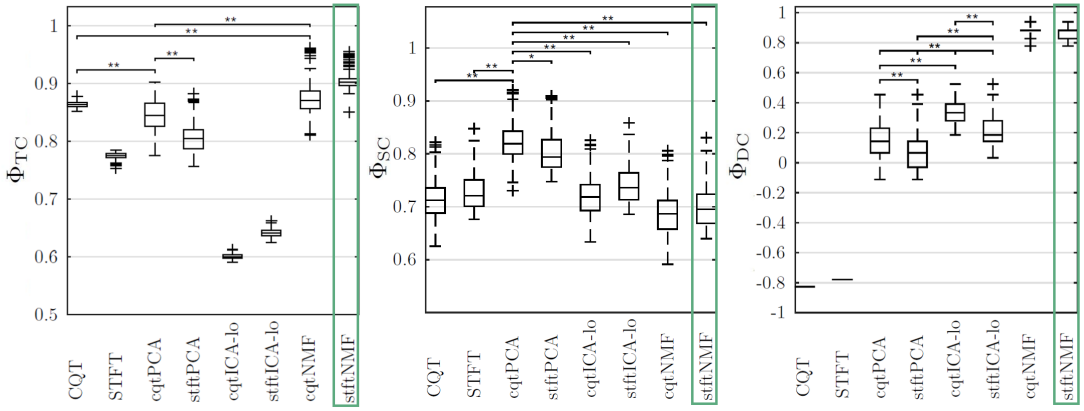

Abbildung 5.3: Boxplots der Evaluierung der drei Kriterien von Nagathil in „Evaluation of Spectral Transforms for Music Signal Analysis“ (für einen SNR von 12 dB) [11, S. 3]

6 Test Cases

Im folgenden werden fünf Test Cases auf ihre Separations-Eigenschaften durch die Toolbox untersucht werden.

Die Verwendung der Toolbox soll dabei anhand eines Mixes aus Klavier und Schlagzeug Samples gezeigt werden, welches im Verlauf des Projektes stets als Referenz herangezogen wurde, um den Fortschritt beurteilen zu können. Aus diesem Grund wird auch das Ergebnis für dieses Beispiel besonders detailliert beleuchtet.

6.1 Piano Beat - Beispiel Testcase

Die Beispiel Audiodateien sind im „Test_Piano2_beat“ Ordner in der Sample Library zu finden, dort kann sich unter anderem der Mix unter dem Namen 'mix_piano2_beat' angehört werden.

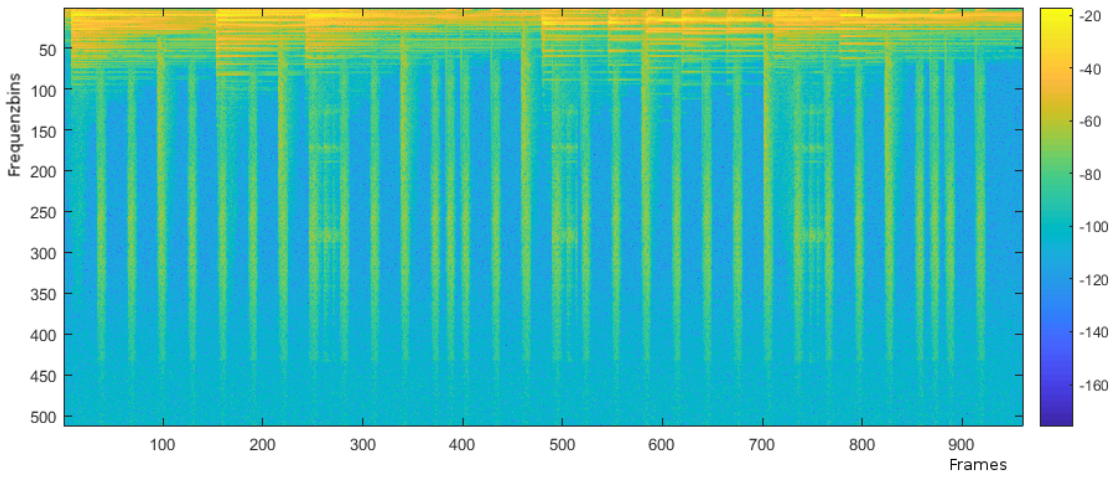

Abbildung 6.1: Spektrum des Mixes aus Klavier und Schlagzeug Samples

Bei einer Wahl von $K = 4$ für die Anzahl der Patterns kann der Hörer den in Abbildung 6.2 dargestellten Komponenten eindeutig bestimmte Signal Anteile zuordnen.

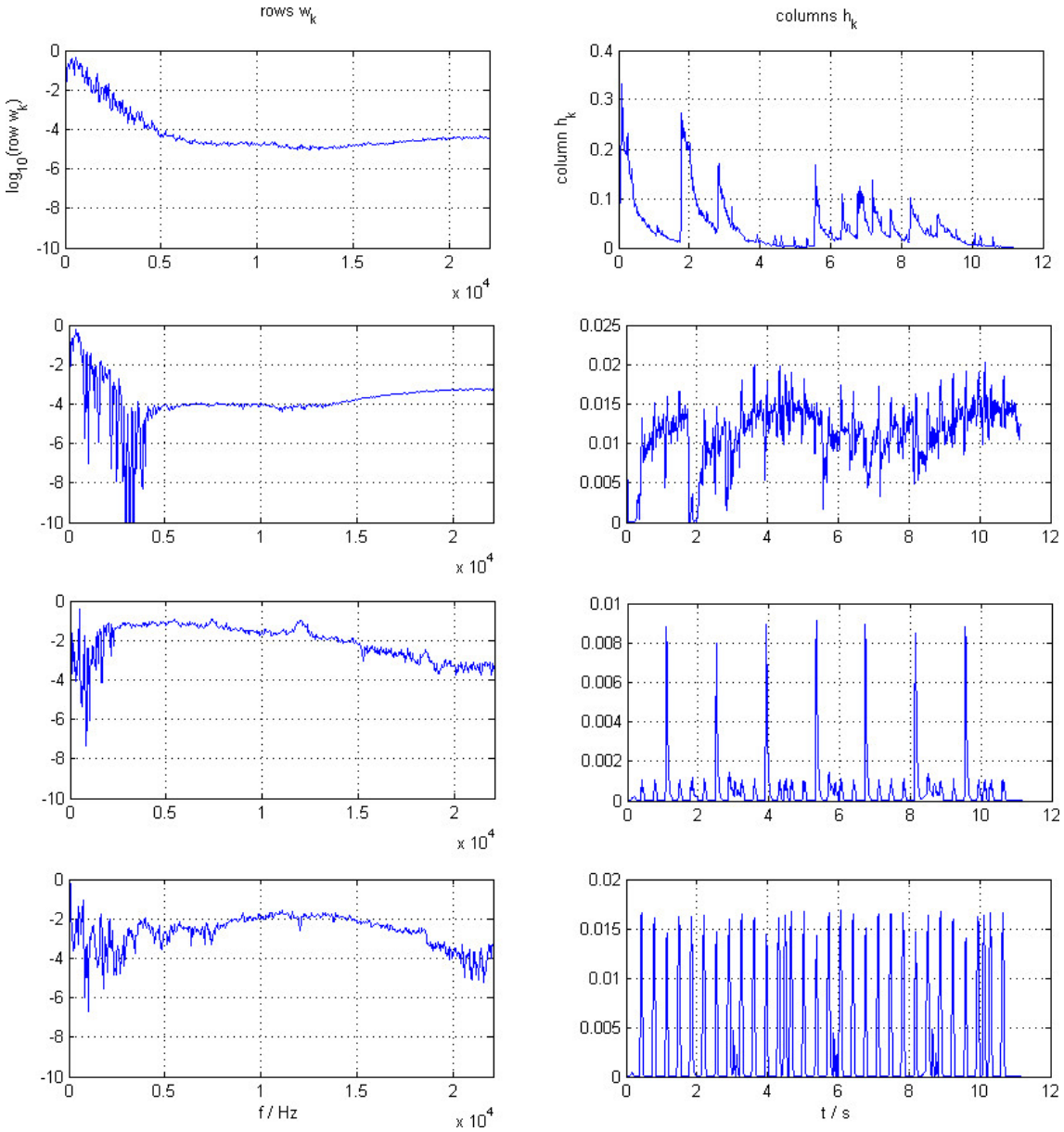

Abbildung 6.2: Ergebnis der Faktorisierung in der Pattern Darstellung

- Pattern 1 - Piano

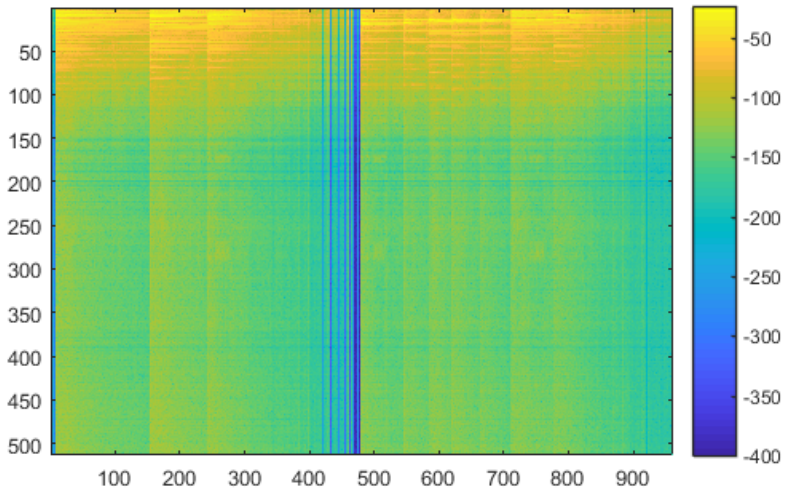

Abbildung 6.3: Spektrum der separierten Komponente 1

- Pattern 2 - Piano Ausklang

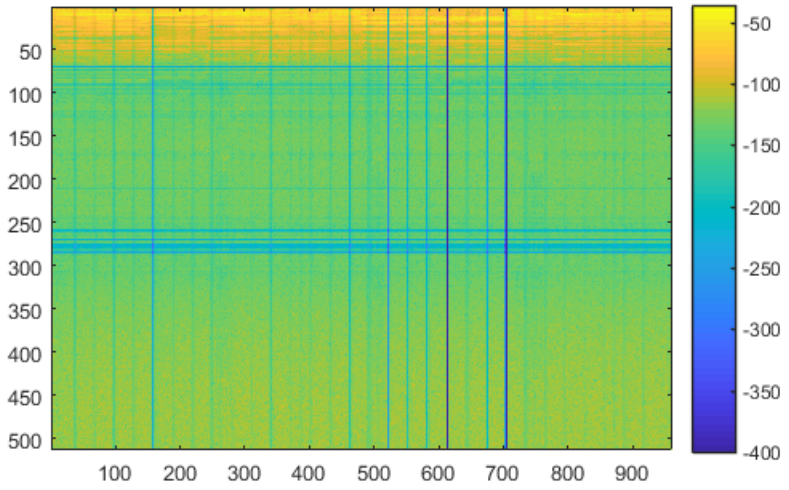

Abbildung 6.4: Spektrum der separierten Komponente 1

- Pattern 3 - Beat (hochfrequentere Anteile und hohler Klang)

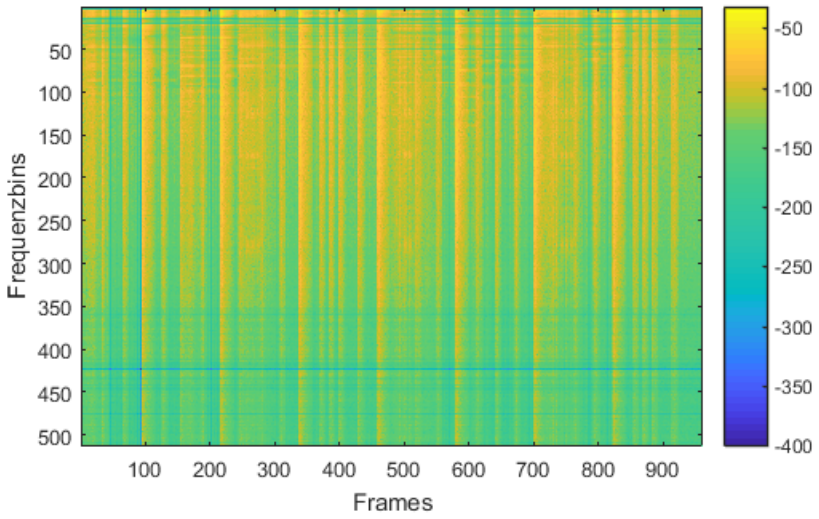

Abbildung 6.5: Spektrum der separierten Komponente 1

- Pattern 4 - Beat (tieffrequente Anteile und Shaker-Sound)

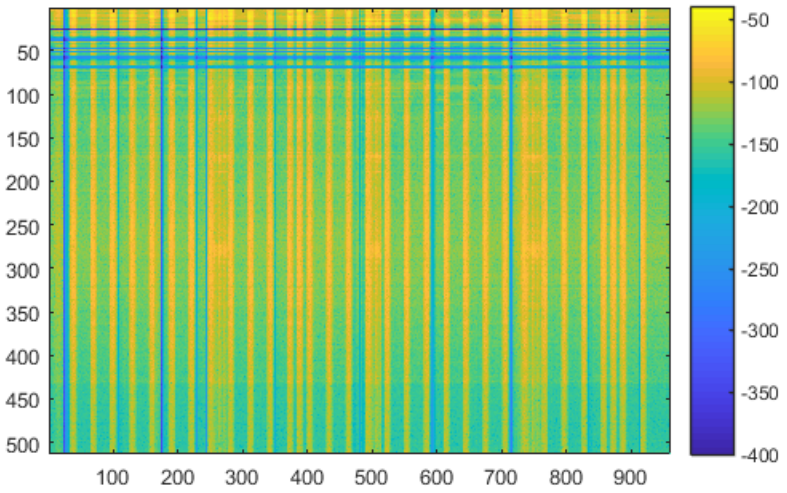

Abbildung 6.6: Spektrum der separierten Komponente 1

Output im Command Window nach erfolgreichem Ausführen in Matlab:

```
Elapsed time is 4.008005 seconds.

Play separated pattern with index 3
finished

Flag merge is set -> play merged Signal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Result of Evaluation (see Nagathil):

phi =

    DC: 0.9845
    SC: 0.7677
    TC: 0.5251

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Terminated
```

Ergebnis der Evaluierung (siehe Kapitel 5):

```
The ISR, SIR, SAR and SDR criteria computed with the new decomposition
    are:
 - SDR = 10.6 dB
 - ISR = 11.3 dB
 - SIR = 26.8 dB
 - SAR = 25.6 dB
The audio quality (PEMO-Q) criteria computed with the new decomposition
    are:
 - qGlobal = 0.971
 - qTarget = 0.986
 - qInterf = 0.992
 - qArtif = 0.992
************************
****  FINAL RESULTS  ****
************************
 - Overall Perceptual Score: OPS = 44/100
 - Target-related Perceptual Score: TPS = 97/100
 - Interference-related Perceptual Score: IPS = 70/100
 - Artifact-related Perceptual Score: APS = 83/100
```

Zum Vergleich für eine zweite Faktorisierung mit den selben Input Daten wurden folgende PEASS-Evaluierungs-Ergebnisse erzielt:

```
 - Overall Perceptual Score: OPS = 35/100
 - Target-related Perceptual Score: TPS = 98/100
 - Interference-related Perceptual Score: IPS = 63/100
 - Artifact-related Perceptual Score: APS = 79/100
```

Analyse der Ergebnisse

Es werden nun 9 Parameter-Sets mit den in Abschnitt 4.1 beschrieben Optionen untersucht werden. Die jeweils zugehörigen Wave-Files sind im Ordner "`Test Cases Demo/Piano_Beat`" zu finden.

- Test 1: "default" Parameter
- Test 2: F = 1024 (Piano wurde aufgeteilt auf 2 Patterns, höhere und tiefere Frequenzen, trotzdem ist noch viel Schlagzeug zu hören. Mit besserer Frequenzauflösung wird aber vor allem die Frequenzauflösung im unteren Frequenzbereich verbessert, dafür aber die Zeitauflösung verschlechtert. Sollen perkussive Signale separiert werden sollte die Fensterlänge also kürzer gewählt werden.)
- Test 3: sig.squared = 1 (Im Plot gibt es einen sichtbaren Unterschied da schwächere Anteile noch schwächer gemacht werden. Außerdem können jetzt auch Akkordteile von Melodieteilen grafisch unterschieden werden. Siehe Abbildung 6.7)

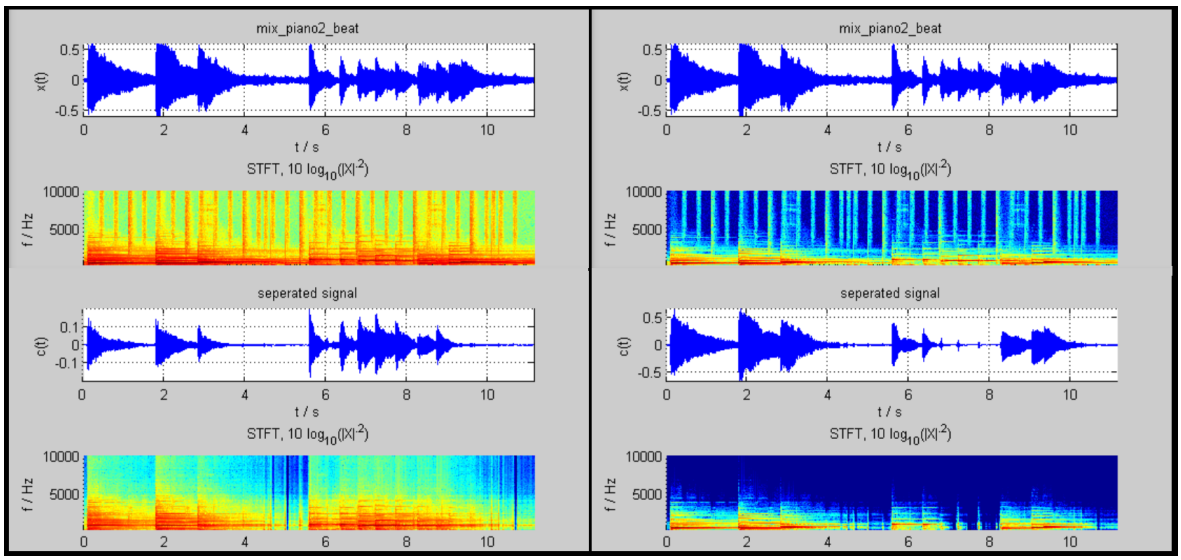

Abbildung 6.7: Vergleich der grafischen Ergebnisse von Test 1 und 3

- Test 4: sig.hop = nfft/4 und sig.squared = 1 (Percussion wird genauer getrennt)
- Test 5: param.beta = 1 bzw. 2 (Funktionert wesentlich schlechter - es wird auch nur mehr in eine Schlagzeug Komponente separiert. Außerdem stottert das separierte Audiofile. Wird wie in 4a noch zusätzlich sig.squared = 1 gesetzt wird in gar keine Schlagzeug Komponente mehr separiert.)
- Test 6: flag.init_type = 2 und sig.squared = 1 (Beim Ausklang gibt es erhebliche Probleme mit der Separation)
- Test 7: flag.init_type = 2 und sig.squared = 0 (Leicht besseres Ergebnis beim Ausklang)

- Test 8: flag.init_type = 2, sig.squared = 0 und flag.init_channel2 = 1 (Schlechtes Ergebnis wobei noch nicht geklärt werden konnte warum. Von der Theorie her sollte die Initialisierung mit der Gain Zusatzinformation viel besser funktionieren)
- Test 9: “default“ und sig.win = ’hann’ (Ergebnis ist klanglich bröselig)

6.2 Piano Violine

Auf dieser Aufnahme sind ein Klavier und eine interferiernde Geige zusammen gemixed, um im Kontrast zum vorigen Beispiel die Separation von ähnlicheren spektralen Mustern untersuchen zu können.

Es funktioniert relativ gut das Klavier zu separieren, umgekehrt jedoch schlechter. Vor allem in den Klavier Pausen kann die Geige nicht zur Gänze eliminiert werden.

6.3 Gitarre Bass Drums

Es funktioniert sehr gut das Schlagzeug weg zu rechnen. Die komponentenweise Initialisierung funktioniert für die Gitarre ähnlich gut wie die Initialisierung über nur ein Audiofile (Vergleich “estimate_guitar_1024“ und “estimate_guitar_ci“), allerdings wird nur eine ausgewählte Komponente initialisiert und nicht in drei gewünschte Komponenten gleichzeitig. Mit einer Fensterlänge von 2048 Samples funktionierte die Separation besser, allerdings ist das Ergebnis etwas bruchstückhafter.

Die Berechnung für 50 Iterationen dauert ca. 5 Sekunden.

6.4 Gitarre Bass Drums Vocals

Die Separation der Vocals funktioniert sehr schlecht. Ähnlich wie beim Schlagzeug funktioniert es aber relativ gut die Vocals weg zu rechnen.

6.5 Band Aufnahme

Bei der untersuchten Aufnahme einer Pop-Band sind die Ergebnisse wesentlich besser wenn das Signal im Frequenzbereich nicht quadriert wird (sig.square = 0). Der Gesang wurde am besten in der Komponente der verzerrten Gitarre separiert. Die Separation des Basses funktioniert am Besten, ebenfalls akzeptabel funktioniert sie mit dem Schlagzeug.

7 Optimierungen

7.1 Blockaufteilung

Da die spektralen Patterns der Matrix **W** mit zunehmender Länge des Signals unspezifischer werden, da mehrere Spektren zu einem zuammengefasst werden, muss überlegt auf welche Weise längere Signale in Blöcke zur Verarbeitung aufgeteilt werden.
Getestet wurden folgende Möglichkeiten für das *Piano*2_*Beeat* Beispiel:

1. Den Mix in Blöcke einer fixen Größe zerschneiden
2. Den Mix auf die Onsets zerschneiden (wurde für den Test manuell umgesetzt)

Die Beispiel beziehungsweise Ergebnis Files können im Ordner *Examples/* angehört werden:

Gesamt: estimate_piano2_beat.wav
3 Sekunden Schnitt: estimate_piano2_beat_schnitt_3s.wav
Onsets: estimate_piano2_beat_schnitt_onsets.wav

Der automatisierte Schnitt in 3 s lange Teile bringt schlechte Ergebnisse, da die Dynamiken nicht kontinuierlich verlaufen. Beim ersten Schnitt in Abbildung 7.1 sollte der zuvor angeschlagene Ton eigentlich ausklingen.

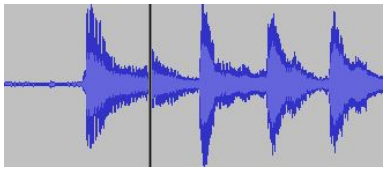

Abbildung 7.1: Ergebnis eines simulierten automatischen Schnittes nach Zusammensetzen

Auch der Schnitt auf die Onsets bringt schlechte Ergebnisse, da die Dynamik im Gesamtkontext nicht stabil bleibt, sondern nach Anschlägen ein starker Decay erfolgt, der in Abbildung 7.2 im Vergleich zum Ergebnis ohne Zerschneiden ersichtlich ist.

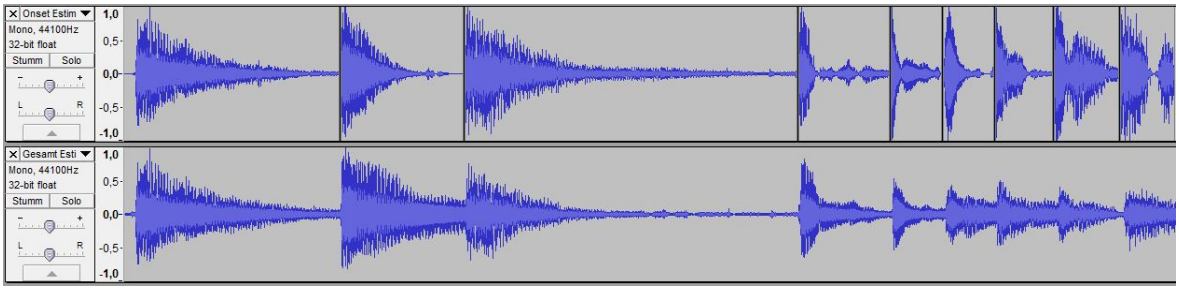

Abbildung 7.2: Vergleich der Ergebnisse der Separation eines simulierten Onset-Schnitts und des gesamten Files (1.Zeile: Onset, 2.Zeile: Gesamt)

Fazit: Das Zerschneiden in Frames in diese kurzen Abschnitte ist nicht sinnvoll, die Länge sollte, abhängig vom Musikstil, 10-15 Sekunden entsprechen.

7.2 Robustheit

Da die W und H Matrix bei der NMF vor der Initialisierung durch ein Audiofile mit Zufallszahlen initialisiert werden, können die Ergebnisse für eine mittlere Iterationszahl leicht unterschiedlich ausfallen. Für die Evaluierung nach Nagathil beispielsweise zeichnet sich für die „Spectral Sparsity“ eine Abweichungen von ca. 0.01 vom Durchschnittswert ab.
(Der dritte Evaluierungs Parameter nach Nagathil, die Datenkompression, bleibt immer gleich, da sich die Dimensionen der Input Daten nicht verändern.)

	Versuch 1	Versuch 2	Versuch 3	Versuch 4
Spectral Compactness	0.8127	0.8110	0.8214	0.88075
Time Continuity	0.5196	0.5198	0.5197	0.5186

Wird die Zufallszahl für die Initialisierung also nicht über die $param\_nmf.rand\_var$ Variable fixiert, ist das Ergebnis der Faktorisierung erwartungsgemäß variabel (siehe Abbildung 7.3).
Zudem verändert sich die Pattern-Reihenfolge mit jedem Durchlauf.

Einen Unterschied im Bezug auf die Robustheit macht auch die gewählte Anzahl der Patterns K. Wird allerdings die Iterationsanzahl auf 100 erhöht, wird die Aufteilung auf die einzelnen Patterns für jeder Durchführung sehr ähnlich (getestet wurde mit $K = 4$).

Ein Kompromiss für die Anzahl der Iterationen bezüglich Robustheit und Berechnungsdauer könnte bei $iter = 70$ liegen (die Berechnung dauert ca. halb so lang wie das Audio File). Soll die Faktorisierung weniger Zeit in Anspruch nehmen kann $iter$ auf 40 reduziert werden.

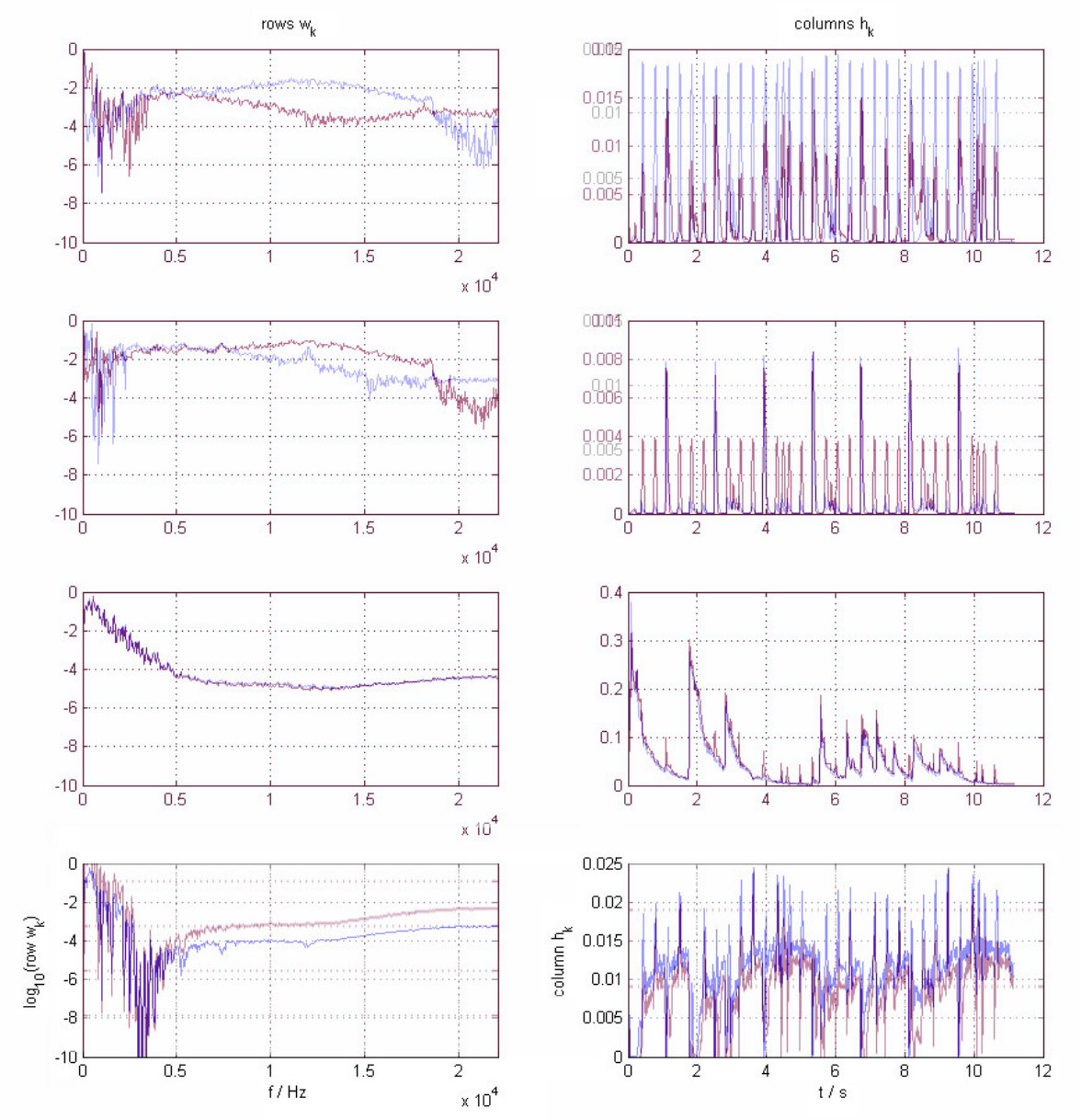

Abbildung 7.3: Die größten Unterschiede mehrerer Durchläufe ohne fixierte Zufallszahlen. (Die Pattern-Reihenfolge der beiden Durchläufe war originalerweise nicht die selbe.)

Fazit: Auffallend ist, dass das gewünschte Haupt-Pattern trotzdem sehr robust und unabhängig von einzelnen Durchläufen und der gewählten Patternanzahl ist. Grundsätzlich verändert sich hauptsächlich die Reihenfolge der Patterns und die Aufteilung der zu ungewollten Anteile.

8 Erkenntnisse

8.1 Allgemeine Erkenntnisse

- Einen überraschend großen Einfluss auf die Faktorisierung haben die Parameter der STFT wie FFT-Fenstergröße, Fensterfunktion und Hop Size. Die Ergebnisse unterscheiden sich stark für verschiedene Werte.
- Der Parameter K kann für die getesteten Beispiele ähnliche Werte annehmen. Für die NMF liegt der Wert zwischen 5 und 9, für die PCA etwas höher bei 10 bis 15. In [9, S. 81] wird darauf eingegangen für welche Transformationen verschiedene K Werte optimal sind.
- NMF,PCA und ICA bringen ohne Transformation in den Frequenzbereich keine brauchbaren Ergebnisse für die Single-Channel Anwendung.
- Das Quadrieren des Betragsspektrums verändert das Ergebnis der Faktorisierung nicht wesentlich.
- Die Separation bringt bessere Ergebnisse, wenn die Instrumente mit mehreren verschiedenen Frequenzbereichen initialisiert werden.
- Perkussive Komponenten zu eliminieren funktioniert besser als erwartet. Umgekehrt lässt es sich bei der Separation eines Schlagzeugs nicht vermeiden dass tonale Komponenten erhalten bleiben.
- Die Amplituden Glättung in der Nachbearbeitung verbessert das Ergebnis der Quelltrennung für den Hörer wesentlich. Um zu verhindern, dass Anschläge zu stark weich gezeichnet werden, könnte die Glättung nur auf Bereiche die Störungen enthalten angewendet werden.

8.1.1 NMF

- Der wichtigste Punkt für das Endergebnis ist die Initialisierung der **W** und **H** Matrizen. Hier könnte der User im Fall eines Plug-Ins auswählen welche Instrumente vorkommen und diesen Schlagwörter (wie Gitarre, clean/verzerrt, viel Hall/trocken, Melodie/Akkorde) zuordnen, nach denen ein passendes Initialisierungs-File aus einer Bibliothek ausgewählt wird.
- Für eine wesentliche Verbesserung könnte das Einbeziehen von Constraints in den NMF Algorithmus sorgen.
- Problematisch bei der NMF ist die Separation vor allem bei Ausklängen. Ein gutes Beispiel dafür ist das Klavier-Beat Beispiel. Dieses Problem wurde durch Amplituden Glättung als entsprechenden Nachbearbeitungs Schritt zu einem Teil gelöst.

- Werden die komplexen Werte der Eingangsmatrix behalten, funktionert die Faktorisierung für die NMF schlechter (wurde für die PCA und ICA noch nicht getestet)
- Die richtige Größe für K ist sehr wichtig. Die Faktorisierung funktioniert nicht wenn K zu klein ist, für $K = 3$ ist zwar schon eine Residuums Komponente zusätzlich zu den Komponenten für die beiden Instrumente verfügbar, jedoch wird eher zeitlich in erste Hälfte und zweite Hälfte getrennt. Wenn K zu groß wird, sind die Ergebnisse eher schlecht verwertbar. Üblicherweise funktioniert die Source Separation für $K = 4 - 9$ am Besten.
- Die Faktorisierung durch die NMF nimmt für die Berechnung mehr Zeit im Vergleich zur ICA und PCA in Anspruch.
- Spielen Instrumente in einem Abschnitt Tonhöhen aus unterschiedlichsten Frequenzbereichen oder stark unterschiedlich Rhythmen, ist es zielführend die Dauer der zu separierenden Abschnitte kleiner, beispielsweise $10s$, zu wählen.

8.1.2 PCA

- Die PCA funktioniert in der Form der aktuellen Implementierung nicht richtig. Nach der Separation kann in den erhaltenen Komponenten kein sinnvolles musikalisches Signal mehr erkannt werden.
- Der Plot der Komponenten zeigt bei jeder separierten Komponente einen sehr ähnlichen Frequenzgang an.

8.1.3 ICA

- Die Ergebnisse wurden mit Ergebnissen der FastICA Toolbox verglichen und stimmen überein - bei beiden ist das akustische Ergebnis aber gleich wenig zu gebrauchen.

8.2 Ausblick und Fazit

PCA und ICA funktionieren schneller, bringen aber zum aktuellen Stand keine sinnvollen Ergebnisse für die Verwendung zur Quelltrennung. Hierbei konnte auch nicht geklärt werden, ob ein Spektrum ein sinnvoller Input für eine PCA ist. Die ICA aus der FacToolbox bringt dabei ähnliche Ergebnisse wie die FastICA Toolbox von Hyvärinen, die als PlugIn für Matlab erhältlich ist.

Die NMF funktioniert je nach Eingangsdaten und Parameter unterschiedlich gut, bringt jedoch für einige Test Cases gute Ergebnisse. Weitere Schritte, wie die verbesserte Initialisierung per Sound Libraries, bestimmte Pre-Processing Bearbeitungen oder die Überprüfung neuer Testcases, könnten Verbesserungen bringen.

Weitere Möglichkeiten in der Implementierung wären Stereo Files zu verwenden und das Panning mit einzubeziehen oder die Matrix H abhängig vom Signaltyp zu initialisieren. Außerdem könnte mehr Input des Users mit einbezogen werden. Beispielsweise könnte der User Teile markieren in denen ein Instrument alleine spielt.

Offene Fragen sind, ob es überhaupt zweckmäßig ist die STFT der Daten als Matrix $\mathbf{X}$ zu verwenden, oder ob die constant-Q Transformation (wie in [11] gezeigt) nicht besser geeignet wäre. Weiter überlegt werden kann, ob im Frequenzbereich mit komplexen oder reellen Werten gearbeitet werden soll beziehungsweise wie wichtig die Phasen Information für die Feature Extraction ist.

9 Anhang

9.1 Funktionen

Liste und Strukturierung der nicht Matlab internen Funktionen:

```
readAudiofile.m
audioTransform.m
  stft.m (Sonible-Engine)
fact.m
  initNmf.m
  calcNmf.m
      updateRuleNmf.m
          normNmfMU.m
  calcPca.m
  calcIca.m
thirdSmooth.m
checkConverg.m
plotSignal.m
plotNmf.m
plotConverg.m
plotPca.m
plotIca.m
cut.m
reconstr.m
  wienerFilt.m
  istft.m (Sonible-Engine)
analyseSep.m
  plotSep.m
  playSeparatedSignal.m
  smoothing.m
mergeSep.m
evaluationPeass.m (PEASS-Eval Toolbox)
evaluationTransform.m
  gini.m (Oleg Komarov)
```

9.2 Existierende Source Separation Software

Eine kommerzielle Source Separation Software wird von Audionamix verkauft „ADX TRAX“, wobei der Fokus eher auf der Trennung von Vocals liegt.

Im Folgenden sind Youtube-Links zu Show-Cases zu finden:

Gitarren-Separation:
$https://www.youtube.com/watch?v=qCpL5gpZGYs$

Musik-Separation:
$https://www.youtube.com/watch?v=8tCOSZII74Q$

Vocals-Separation:
$https://www.youtube.com/watch?v=1xkyjfwSG8w$

ADX TRAX Pro 3 Tutorial:
$https://www.youtube.com/watch?v=JHu_cM4XtZI$

Literaturverzeichnis

[1] Lee, D.D.; Seung, H.S. (2001): "Algorithms for non-negative matrix factorization." In: *Advances in Neural Information Processing Systems 13: Proceedings of the 2000 Conference. MIT Press. pp. 556–562.*

[2] Smaragdis, P.; Brown, J.C. (2003): "Non-negative matrix and factorization for polyphonic and music transcription." In: *IEEE Workshop on Applications of Signal Proccssing to Audio and Acoustics*, 177–180, New Pallz, NY.

[3] Fevotte, C.; Bertin, N.; Durrieu, J.L. (2009): "Nonnegative matrix and factorization with the itakura-saito and divergence: With and application to music and analysis." In: *Neural Computation*, vol. 21, 793–830, MIT Press Cambridge.

[4] Lee; Seung (1999): "Learning the parts of objects by nmf." In: *Nature*, **401**:788–791.

[5] Shlens, J. (2003), "A tutorial on principal component analysis - derivation, discussion and singular value decomposition."

[6] Köhler, B.U. (2005): "Konzepte der statistischen Signalverarbeitung, Springer Berlin Heidelberg."

[7] Shlens, J. (2014): "A tutorial on independent component analysis." In: *arXiv.*

[8] Hyvärinen, A.; Oja, E. (2000): "Independent component analysis: Algorithms and applications." In: *Neural Networks*, **13**:411–430.

[9] Nagathil, A. (2016): "Music signal processing for the reduction of auditory distortions in hearing-impaired listeners, Ph.D. thesis, Ruhr-Universität Bochum."

[10] Emiya, V.; Vincent, E.; Harlander, N.; Hohmann, V. (2011): "Subjective and objective quality assessment of audio source separation." In: *IEEE Transactions on Audio, Speech and Language Processing.*

[11] Nagathil, A.; Martin, R. (2013): "Evaluation of spectral transforms for music and signal analysis."