NMF TOOLBOX: MUSIC PROCESSING APPLICATIONS OF NONNEGATIVE MATRIX FACTORIZATION

Patricio López-Serrano[1], *Christian Dittmar*[2], *Yiğitcan Özer*[2], *Meinard Müller*[1]

[1]International Audio Laboratories Erlangen* , Erlangen, Germany
[2]Fraunhofer IIS, Erlangen, Germany
{patricio.lopez.serrano, meinard.mueller}@audiolabs-erlangen.de
{christian.dittmar, yigitcan.oezer}@iis.fraunhofer.de

ABSTRACT

Nonnegative matrix factorization (NMF) is a family of methods widely used for information retrieval across domains including text, images, and audio. Within music processing, NMF has been used for tasks such as transcription, source separation, and structure analysis. Prior work has shown that initialization and constrained update rules can drastically improve the chances of NMF converging to a musically meaningful solution. Along these lines we present the NMF toolbox, containing MATLAB and Python implementations of conceptually distinct NMF variants—in particular, this paper gives an overview for two algorithms. The first variant, called nonnegative matrix factor deconvolution (NMFD), extends the original NMF algorithm to the convolutive case, enforcing the temporal order of spectral templates. The second variant, called diagonal NMF, supports the development of sparse diagonal structures in the activation matrix. Our toolbox contains several demo applications and code examples to illustrate its potential and functionality. By providing MATLAB and Python code on a documentation website under a GNU-GPL license, as well as including illustrative examples, our aim is to foster research and education in the field of music processing.

1. INTRODUCTION

The general goal of NMF is to factorize a matrix V with nonnegative entries into two other nonnegative matrices W and H which typically are required to have much lower rank than V. Due to the fact that NMF can learn a semantically meaningful parts-based data representation [1], it has been used extensively in music signal processing and information retrieval, for tasks such as music transcription [2], automatic drum transcription (ADT) [3], drum source separation (DSS) [4], harmonic-percussive source separation (HPSS) [5, 6, 7], in combination with kernel additive modeling (KAM) [8, 9], redrumming [10, 11], sampling detection [12, 13], structure analysis [14, 15, 16], score-informed source separation [17], and key estimation [18], to name a few.

In the case of music tasks, we would like to learn two non-negative matrices W and H that capture *what* happened (i.e., which particular sounds were produced, which drum kit parts were struck, which piano notes were sounded), and *when* each one of these sound events was active in time. Intuitively, we can say that the template (or spectral basis) matrix W contains the "what"—whereas H, called the gain (or activation) matrix, contains the "when". We show this intuition in Figure 1, which illustrates an NMFD model of a short drum signal. The templates are shown on the left as color-coded component spectrograms representing kick drum (KD), snare drum (SD), and hi-hat (HH). The activations are shown as colored curves at the top of the figure. Throughout this paper we present a number of didactic examples, aiming to contribute to music processing education. In contrast to other NMF-related toolboxes [19], we provide an illustrated compendium of musically-motivated applications found throughout the literature. In the spirit of the DAFx book [20], the code examples can be used as a reference implementation in further research, whereas the figures (especially through the color-coding) provide concise illustrations of the principles behind NMF(D), and can be used as learning material.

The remainder of this paper is structured as follows. In Section 2 we introduce the basic theoretical framework and notation for NMF and the variants that we use in this paper. In Section 3 we give an overview of our toolbox code[1], discussing the main functions, parameters, and dependencies. In Sections 4, 5, and 6 we present three application scenarios, illustrated by going through their source code and examining the graphic output from visualization functions. Although our code examples are given as MATLAB listings, we ensured that the naming conventions and usage of our Python implementation are basically the same.

2. NMF AND VARIANTS

In this section we give a brief formal overview of the NMF variants that are provided as code in the toolbox.

2.1. NMF

Here we introduce NMF, closely following [21, Section 8.3] and [1]. NMF is based on iteratively computing a low-rank approximation $\mathrm{U} \in \mathbb{R}_{\geq 0}^{K \times M}$ of the nonnegative matrix (typically a magnitude spectrogram) $\mathrm{V} \in \mathbb{R}_{\geq 0}^{K \times M}$, where $K \in \mathbb{N}$ is the feature dimensionality and $M \in \mathbb{N}$ represents the number of elements or frames along the time axis. Specifically, U is defined as the linear combination of the templates $\mathrm{W} \in \mathbb{R}_{\geq 0}^{K \times R}$ and activations $\mathrm{H} \in \mathbb{R}_{\geq 0}^{R \times M}$ such that $\mathrm{V} \approx \mathrm{U} := \mathrm{W} \cdot \mathrm{H}$. The rank $R \in \mathbb{N}$ of the approximation (i. e., number of components) is an important parameter that needs to be specified beforehand.

NMF typically starts with a suitable initialization of the matrices W and H. For example, both matrices could be populated with non-negative random numbers—however, depending on the task and availability of prior information, it might make more sense to

* The International Audio Laboratories Erlangen is a joint institution of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Fraunhofer-Institut für Integrierte Schaltungen IIS.

[1]https://www.audiolabs-erlangen.de/resources/MIR/NMFtoolbox/

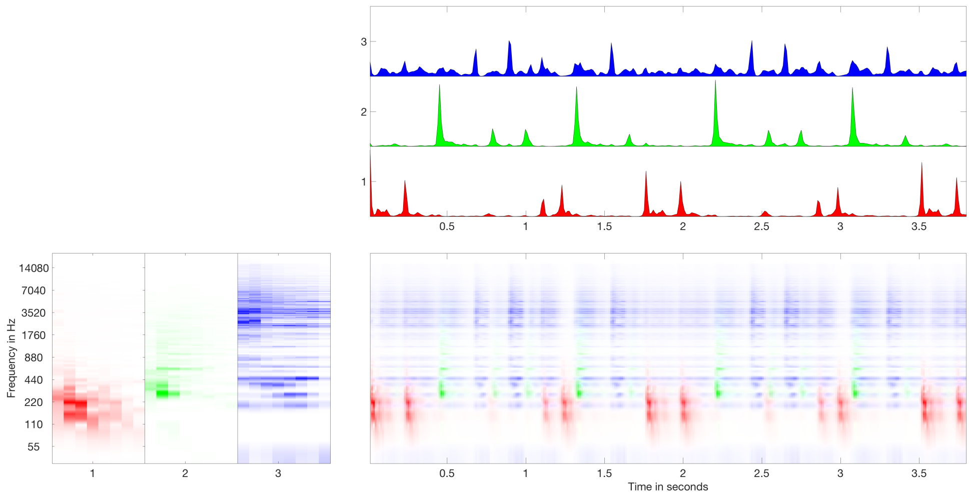

Figure 1: Output visualization generated by code in Listing L.1. NMFD model for drum source separation (KD in red, SD in green, HH in blue). Top: activation matrix H. Bottom left: slices of template tensor P. Bottom right: color-coded approximated target spectrogram U.

use other initialization strategies. After initialization, both W and H are iteratively updated to approximate V with respect to a cost function $\mathcal{L}$. A standard choice is the generalized Kullback-Leibler Divergence (KLD) [1], given as

$$\mathcal{L} = \mathcal{D}_{\mathrm{KL}}(\mathrm{V} \mid \mathrm{U}) = \sum \left(\mathrm{V} \odot \log\left(\frac{\mathrm{V}}{\mathrm{U}}\right) - \mathrm{V} + \mathrm{U} \right). \quad (1)$$

The symbol $\odot$ denotes element-wise multiplication; the logarithm and division are to be performed element-wise as well. The sum is to be computed over all $K \cdot M$ elements of V. To minimize this cost, an alternating scheme with multiplicative updates is used [1]. The respective update rules are given as

$$\mathrm{W} \leftarrow \mathrm{W} \odot \frac{\frac{\mathrm{V}}{\mathrm{U}} \cdot \mathrm{H}^\top}{\mathrm{J} \cdot \mathrm{H}^\top}, \quad (2)$$

$$\mathrm{H} \leftarrow \mathrm{H} \odot \frac{\mathrm{W}^\top \cdot \frac{\mathrm{V}}{\mathrm{U}}}{\mathrm{W}^\top \cdot \mathrm{J}}, \quad (3)$$

with $\mathrm{U} := \mathrm{W} \cdot \mathrm{H}$, where the symbol $\cdot$ denotes the matrix product. Furthermore, $\mathrm{J} \in \mathbb{R}^{K \times M}$ denotes a matrix of ones. Since this is an alternating update scheme, it should be noted that Eq. (2) uses the latest update of H from the previous iteration. In the same vein, (3) uses the latest update of W. These update rules are typically applied for a limited number of iterations $L \in \mathbb{N}$.

2.2. NMFD

In this section we introduce NMFD, originally proposed in [22]. We follow the notation and formulation found in [23].
NMFD extends NMF by using two-dimensional templates (or *patterns*) so that each of the R templates can be interpreted as a magnitude spectrogram snippet consisting of $T \ll M$ spectral frames. We assume that the magnitude spectrogram V can be modeled using a mixture of R patterns $\mathrm{P}^r \in \mathbb{R}^{K \times T^r}$, $r \in [0 : R-1] := \{0, \ldots, R-1\}$. The parameter $T^r \in \mathbb{N}$ is the number of feature frames or observations for pattern P^r. Although the patterns can have different lengths, without loss of generality, we define their lengths to be the same $T := T^0 = \ldots = T^{R-1}$, which could be achieved by adequately zero-padding shorter patterns until they reach the length of the longest. Based on this assumption, the patterns can be grouped into a pattern tensor $\mathrm{P} \in \mathbb{R}^{K \times R \times T}$. The subdimension (or *slice*) of the tensor which refers to a specific pattern with index r is $\mathrm{P}^r := \mathrm{P}(\cdot,\ r,\ \cdot)$, whereas $\mathrm{P}_t := \mathrm{P}(\cdot,\ \cdot,\ t)$ refers to frame index t simultaneously in all patterns. Thus, the magnitude spectrogram can be modeled as

$$\mathrm{U} := \sum_{t=0}^{T-1} \mathrm{P}_t \cdot \overset{t\rightarrow}{\mathrm{H}}, \quad (4)$$

where $\overset{t\rightarrow}{(\cdot)}$ denotes a frame shift operator [22]. Smaragdis [22] defined the update rules that extend Eqs. 2 and 3 to the convolutive case as follows:

$$\mathrm{P}_t \leftarrow \mathrm{P}_t \odot \frac{\frac{\mathrm{V}}{\mathrm{U}} \cdot \left(\overset{t\rightarrow}{\mathrm{H}}\right)^\top}{\mathrm{J} \cdot \left(\overset{t\rightarrow}{\mathrm{H}}\right)^\top}, \quad (5)$$

$$\mathrm{H} \leftarrow \mathrm{H} \odot \frac{\mathrm{P}_t^\top \cdot \overset{\leftarrow t}{\left[\frac{\mathrm{V}}{\mathrm{U}}\right]}}{\mathrm{P}_t^\top \cdot \mathrm{J}}. \quad (6)$$

As a side note, NMFD can be made to function like a regular NMF (Section 2.1) by using a pattern tensor with dimensions $K \times R \times 1$ (i. e., R patterns with a single frame).

2.3. Diagonal NMF

Diagonal NMF is a variant originally proposed by Driedger et al. in [24] for the task of audio mosaicing. In audio mosaicing, a (timbral) *source* is used to recreate or synthesize the sounds in a *target*. In the original publication, Driedger et al. present an example consisting of buzzing bees as a source and “Let It Be” by the Beatles as a target. Their objective is to synthesize a signal which sounds as if the bees buzzed at different pitches to the tune of the Beatles’ song. In order to maximize the recognizability of the source (buzzing bees), the authors propose using a fixed template matrix as well as an extended set of update rules that support the development of sparse diagonal structures in the activation matrix. The crucial observation is that diagonals activate sequences of frames, helping preserve temporality and recognizability. We present an example of diagonal NMF for mosaicing in Section 6.

3. TOOLBOX

In this section we discuss the overall structure and functionality of our toolbox, summarized in Table 1. The first four rows contain the core functions, implementing variants of NMF. We begin with `NMFD`, a core function used for the examples in Sections 4 and 5 (we briefly discussed the theory in Section 2.2).
The variant used as an application example in Section 6 is `NMFdiag`, described in Section 2.3. It requires more input parameters as fields of the structure `parameter.continuity` and specific information can be found in the respective source code headers, as well as in the original publication [24].
Next in the table we have `convModel` and `shiftOperator`. `convModel` is called from within `NMFD` and `NMFconv`—it performs the core convolution operation between P and H at every iteration. On the other hand, `shiftOperator` is a helper function to perform the frame shifting used in Eqs. 4, 5, and 6, adding boundary checks and zero-padding as necessary. The functions `initActivations` and `initTemplates` are helper functions that help us initialize the basis matrices/tensors and the activations matrices. The user may call these functions explicitly to create and assign parameters, but these functions are also called internally with default settings from within the core functions if the user doesn't pass variables in (see Sections 4 and 5 for more information on the types of initialization that are used for specific circumstances.)
The function `NEMA` is used to introduce exponential decay when initializing certain types of templates and activations with both `init` functions. We also provide utility functions to transform between frequency in Hz and MIDI pitches, which are mainly used to generate log-frequency spectrograms for visualization and to generate harmonic templates. Finally, we use `forwardSTFT` to compute the spectrograms we use with our NMF examples. Some tasks also require resynthesizing time-domain signals (such as DSS in Section 4 and mosaicing in Section 6)—for these cases we include `LSEE_MSTFTM_GriffinLim` [28], `inverseSTFT`, and `alphaWienerFilter` (see [29, 30]).

4. APPLICATION: DRUM SOURCE SEPARATION

We begin with an example from the task of DSS, taken from [26]. Given a recording of mixed drum kit components into one signal, the goal of DSS is to produce individual component signals for each instrument or piece in the drum kit as if it had been recorded in isolation. This technique can be used within recording studio or remixing settings, where it is often desirable to treat individual drum kit components separately.

```matlab
1   inpPath = 'data/';
2   outPath = 'output/';
3   filename = 'Winstons_AmenBreak.wav';
4
5    % 1. load the audio signal
6   [x,fs] = audioread([inpPath filename]);
7   x = mean(x,2);
8
9    % 2. compute STFT
10  % spectral parameters
11  paramSTFT.blockSize = 2048;
12  paramSTFT.hopSize = 512;
13  paramSTFT.winFunc = hann(paramSTFT.blockSize);
14  paramSTFT.reconstMirror = true;
15  paramSTFT.appendFrame = true;
16  paramSTFT.numSamples = length(x);
17
18   % STFT computation
19  [X,A,P] = forwardSTFT(x,paramSTFT);
20
21   % get dimensions and time and freq resolutions
22  [numBins,numFrames] = size(X);
23  deltaT = paramSTFT.hopSize / fs;
24  deltaF = fs / paramSTFT.blockSize;
25
26   % 3. apply NMF variants to STFT magnitude
27  % set common parameters
28  numComp = 3;
29  numIter = 30;
30  numTemplateFrames = 8;
31
32   % generate initial guess for templates
33  paramTemplates.deltaF = deltaF;
34  paramTemplates.numComp = numComp;
35  paramTemplates.numBins = numBins;
36  paramTemplates.numTemplateFrames = numTemplateFrames;
37  initW = initTemplates(paramTemplates,'drums');
38
39   % generate initial activations
40  paramActivations.numComp = numComp;
41  paramActivations.numFrames = numFrames;
42  initH = initActivations(paramActivations,'uniform');
43
44   % NMFD parameters
45  paramNMFD.numComp = numComp;
46  paramNMFD.numFrames = numFrames;
47  paramNMFD.numIter = numIter;
48  paramNMFD.numTemplateFrames = numTemplateFrames;
49  paramNMFD.initW = initW;
50  paramNMFD.initH = initH;
51
52   % NMFD core method
53  [nmfdW, nmfdH, nmfdV, divKL] = NMFD(A, paramNMFD);
54
55   % alpha-Wiener filtering
56  nmfdA = alphaWienerFilter(A,nmfdV,1);
57
58   % visualize
59  paramVis.deltaT = deltaT;
60  paramVis.deltaF = deltaF;
61  paramVis.endeSec = 3.8;
62  paramVis.fontSize = 24;
63  visualizeComponentsNMF(A, nmfdW, nmfdH, nmfdA, paramVis);
64
65  % resynthesize
66  for k = 1:numComp
67    Y = nmfdA{k} .* exp(j * P);
68
69    % re-synthesize, omitting the Griffin Lim iterations
70    y = inverseSTFT(Y, paramSTFT);
71
72    % save result
73    audiowrite([outPath,'Winstons_AmenBreak_NMFD_component_',
          ...
74      num2str(k)],y,fs);
75  end
```

Listing L.1: MATLAB code for drum source separation using NMFD, following [26].

In lines 11–16 we define the parameters for STFT computation: a block size $N = 2048$ samples, a hop size $H = 512$ samples, a Hann window, a flag to discard the mirror spectrum (`reconstMirror = true`), and a flag to indicate that we want to zero-pad the entire signal with half-block lengths at the beginning and end. Now we will discuss the most important part of this example, which is the particular setup of the NMF-related variables and parameters. In line 28 we initialize the number of components R to 3, since we know a priori that the drum recording contains the instruments kick

Filename	Description and main parameters
`NMFD.m`	Nonnegative Matrix Factor Deconvolution with KLD and fixable components [22]. `V, numComp, numIter, numTemplateFrames, initW, initH, paramConstr, fixH`
`NMF.m`	Nonnegative matrix factorization with KLD as default cost function [21, Section 8.3], [1]. `V, costFunc, numIter, numComp.`
`NMFdiag.m`	Nonnegative matrix factorization with enhanced diagonal continuity constraints [24]. `V, W0, H0, distmeas, numOfIter, fixW, continuity.length, continuity.grid, continuity.sparsen, continuity.polyphony`
`NMFconv.m`	Convolutive NMF with beta-divergence [25, Chapter 3.7]. `V, numComp, numIter, numTemplateFrames, initW, initH, beta, sparsityWeight, uncorrWeight`
`convModel.m`	Convolutive NMF model implementing Eq. (4) from [26]. Note that it can also be used to compute the standard NMF model in case the number of time frames of the templates equals one. `W, H`
`shiftOperator.m`	Shift operator as described in Eq. (5) from [26]. It shifts the columns of a matrix to the left or the right and fills undefined elements with zeros. `A, shiftAmount`
`initActivations.m`	Initialization strategies for NMF activations, including `random` and `uniform`. The `pitched` strategy places gate-like activations at the frames where certain notes are active in the ground truth [27]. The strategy `drums` uses decaying impulses at these positions [26]. `numComp, numFrames, deltaT, pitches, onsets, durations, drums, decay, onsetOffsetTol, tolerance, strategy`
`initTemplates.m`	NMF template initialization strategies, including `random` and `uniform`. The strategy `pitched` uses comb-filter templates [27]. The `drums` strategy uses pre-extracted averaged spectra of typical drum types. `numComp, numBins, numTemplateFrames, pitches, drumTypes, strategy`
`NEMA.m`	Row-wise nonlinear exponential moving average, introducing decaying slopes according to Eq. (3) from [9]. `lambda`
`midi2freq.m, freq2midi.m, logFreqLogMag.m`	Helper functions to convert between MIDI pitches and frequencies in Hz, as well as log-frequency and log-magnitude representations for visualization. `midi, freq, A, deltaF, binsPerOctave, upperFreq, lowerFreq`
`LSEE_MSTFTM_GriffinLim, forwardSTFT.m, inverseSTFT.m`	Reconstruct the time-domain signal by means of the frame-wise inverse FFT and overlap-add method described as least squares error estimation from the modified STFT magnitude (LSEE-MSTFT) in [28]. `blockSize, hopSize, anaWinFunc, synWinFunc, reconstMirror, appendFrame, analyticSig, numSamples`
`alphaWienerFilter.m`	Alpha-related soft masks for extracting sources from mixture. Details in [29] and experiments in [30]. `alpha, binarize`

Table 1: Overview of MATLAB functions, descriptions, and main parameters. The Python version of the toolbox follows the same naming convention as far as possible.

drum (KD), snare drum (SD), and hi-hat (HH). We will run NMFD for a total of 30 iterations (of the update rules), $L = 30$, line 29, and each spectral slice P^{τ} will have a length of $T = 8$ time frames, line 30. A unique thing that sets apart this example from the rest is the fact that we initialize the templates with a particular strategy that has proven to be very effective in DSS [26]. In line 37 we use the string parameter `drums` to specify that we want to initialize the templates by seeding them with a single-frame, data-driven statistical mean of many drum sounds of that type, and then applying a short exponential decay to that single frame in order to expand the single frame into multiple frames across the template pattern matrix. This initializes the template tensor to have three patterns, each containing prototypical spectral properties of KD, SD, and HH, respectively. For this particular application, the initial activation matrix `initH` can be initialized using a constant value of `1` throughout the entire matrix, indicated by the parameter `uniform` in line 42. In lines 45–50 all the previously created parameters are assigned to the parameter structure `paramNMFD` which will be passed to `NMFD()`, the main function which we call in line 53. In line 56 we use `alphaWienerFilter()` to compute spectrogram estimates for the components via Wiener filtering (see [29, 30]). We set visualization parameters to `paramVis` in lines 59–62 and call `visualizeComponentsNMF()` in line 63, which produces the output seen in Figure 1. It is important to mention that the visualization function assigns default color schemes depending on the NMF model's rank. For instance, as seen in Figure 1, a model with $R = 3$ components will be colored with with red, green, and blue, respectively. A model with $R = 4$ components (Figure 2) is colored using `hsv(4)`, and models with $R > 4$ (Figure 3) are colored with a grayscale colormap. Finally, we loop over the component spectrograms, resynthesizing each one using `inverseSTFT()` (line 70) and saving the result `y` as a WAV file.

5. APPLICATION: ELECTRONIC MUSIC STRUCTURE

Taken from [23], we show an example of the following task: Given a downmix of an electronic music (EM) track produced with certain loops, together with individual instances of the loops that were used to produce the track, can we use NMFD to learn an activation matrix which tells us when each loop was used in the track? In this example, we want to highlight the fact that after initializing the template tensor pages, each pattern to one instance of one of the loops used, we subsequently disallow updating/learning the templates—also called fixing the templates. Thus, we are only interested in allowing updates to the activation matrix, which will end up telling us when each loop type was used.

```
% initialization
inpPath = 'data/';
outPath = 'output/';
filename = 'LSDDM_EM_track.wav';
filenameEffects = 'LSDDM_EM_Effects.wav';
filenameBass = 'LSDDM_EM_bass.wav';
filenameMelody = 'LSDDM_EM_melody.wav';
filenameDrums = 'LSDDM_EM_drums.wav';

% 1. load the audio signal
[xTr,fs] = audioread([inpPath filename]);

[xEffects, fsEffects] = audioread([inpPath filenameEffects]);

[xBass, fsBass] = audioread([inpPath filenameBass]);
[xMelody, fsMelody] = audioread([inpPath filenameMelody]);
[xDrums, fsDrums] = audioread([inpPath filenameDrums]);
% make monaural if necessary
xTr = mean(xTr, 2);
xEffects = mean(xEffects, 2);
xBass = mean(xBass, 2);
```

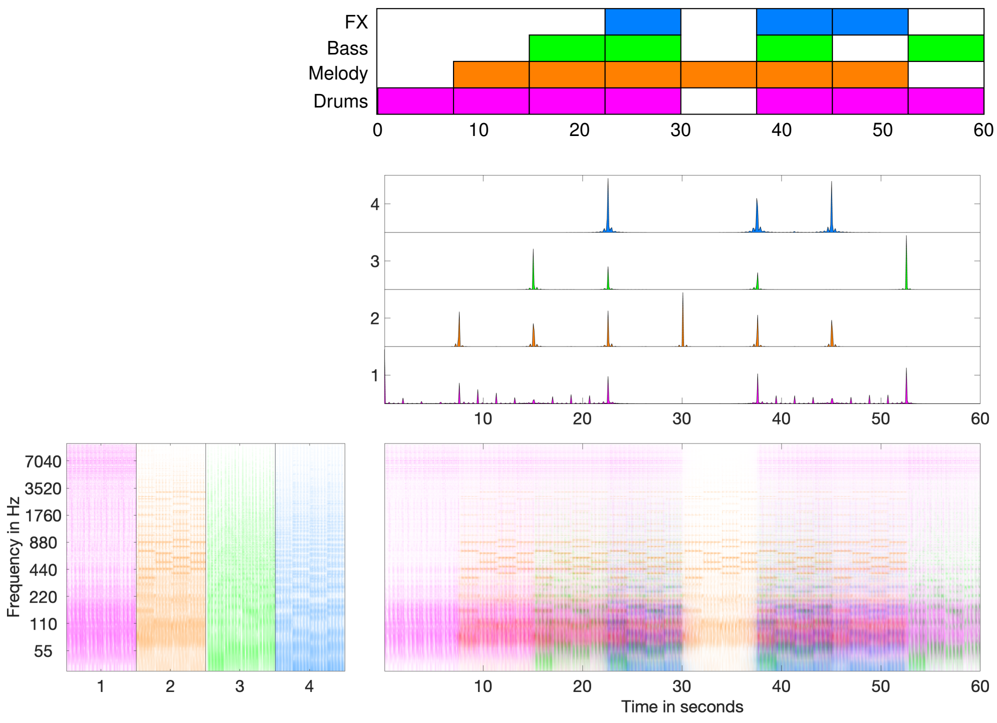

Figure 2: Top: Ground truth loop activations for the track used in Listing L.2. Middle: Activation matrix H with one row per loop/component. Bottom left: Template tensor P with component spectrogram slices for drums (1), melody (2), bass (3), and FX (4). Bottom right: component-colored spectrogram U.

```
21  xMelody = mean(xMelody, 2);
22  xDrums = mean(xDrums, 2);
23
24  % 2. compute STFT
25  paramSTFT.blockSize = 4096;
26  paramSTFT.hopSize = 2048;
27  paramSTFT.winFunc = hann(paramSTFT.blockSize);
28  paramSTFT.reconstMirror = true;
29  paramSTFT.appendFrame = true;
30
31  % STFT computation
32  [XTr, ATr, PTr] = forwardSTFT(xTr, paramSTFT);
33
34  % get dimensions and time and freq resolutions
35  [numBinsTr, numFramesTr] = size(XTr);
36  deltaT = paramSTFT.hopSize / fs;
37  deltaF = fs / paramSTFT.blockSize;
38
39  % STFT computation
40  [XEffects, AEffects, PEffects] = forwardSTFT(xEffects,
        paramSTFT);
41  [XBass, ABass, PBass] = forwardSTFT(xBass, paramSTFT);
42  [XMel, AMel, PMel] = forwardSTFT(xMelody, paramSTFT);
43  [XDrums, ADrums, PDrums] = forwardSTFT(xDrums, paramSTFT);
44  [numBinsBass, numFramesBass] = size(XBass);
45
46  % 3. apply NMF variants to STFT magnitude
47  numComp = 4;
48  numIter = 30;
49
50  initW = [];
51  initW{1} = ABass;
52  initW{2} = AMel;
53  initW{3} = ADrums;
54  initW{4} = AEffects;
55  paramNMFD.initW = initW;
56  numTemplateFrames = numFramesBass;
57
58  % generate initial activations
59  paramActivations.numComp = numComp;
60  paramActivations.numFrames = numFramesTr;
61  initH = initActivations(paramActivations, 'uniform');
62  paramNMFD.initH = initH;
63
64  % NMFD parameters
65  paramNMFD.numComp = numComp;
66  paramNMFD.numFrames = numFramesTr;
67  paramNMFD.numIter = numIter;
68  paramNMFD.numTemplateFrames = numTemplateFrames;
69  paramNMFD.numBins = numBinsTr;
70  paramNMFD.fixW = 1;
71
72  % NMFD core method
73  [nmfdW, nmfdH, nmfdV, divKL] = NMFD(ATr, paramNMFD);
74
75  %% visualize
76  paramVis.deltaT = deltaT;
77  paramVis.deltaF = deltaF;
78  paramVis.logComp = 1e5;
79  fh1 = visualizeComponentsNMF(ATr, nmfdW, nmfdH, nmfdV,
        paramVis);
80
81  %% save result
82  saveas(fh1,[outPath,'LSDDM_EM.png']);
```

Listing L.2: Example code for learning loop activation points using NMFD, following [23].

In lines 2–22 we prepare path names and load files for five audio signals. These include the mixed track, as well as the four loops that were used to produce the track: effects, bass, melody, and drums. At the end of this block we convert all signals to mono.
In the next block (lines 25–29), we set the STFT parameters. In lines 32–43 we use the parameter structure `paramSTFT` together with the respective signals to compute all the necessary spectrograms by calling `forwardSTFT()`.
In the next lines we will be preparing the parameters that are re-

quired to run NMFD with the specific conditions for this example. First, in line 47 we set `numComp`, the number of components or patterns R, to 4 (since we know a priori that the track was composed with four loops or patterns). In line 48 we specify 30 as the desired number of iterations for the algorithm. In lines 50–55 we prepare a cell array `initW` containing the magnitude spectrograms for the loops, and set the initial template tensor to the parameter structure `paramNMFD.initW`. As the number of template frames T we use the number of frames of the bass loop, but we could have used any loop's length—we have constructed the example such that all loops have the same number of frames. In lines 59–62 we initialize the activation matrix using the `uniform` strategy (i. e., with a constant value of one for all rows) and append it to the parameter structure. In lines 65–69 we set some basic values to the parameter structure `paramNMFD`. Most importantly, in line 70, we indicate with the flag `fixW = 1` that we want to fix the template tensor. Since we initialized the pattern tensor with the loops we know to be contained in the track, we disallow modifications and only wish to learn the activation matrix. In line 73 we call `NMFD()` with the track magnitude spectrogram and the previously set parameter structure, obtaining four return variables: `nmfdW`, the learned templates (which are not modified during the learning); `nmfdH`, the learned activation matrix, `nmfdV`, a cell array containing the learned component spectrograms; and `divKL`, an array with the KLD at each iteration (to visualize learning error throughout the iterations). In lines 76–78 we set basic visualization parameters to the structure `paramVis`, then visualize the learned NMFD model with `visualizeComponentsNMF()` (line 79), and save the result to disk (line 82).

6. APPLICATION: AUDIO MOSAICING

In this example we will be loading two sounds: a recording of bees buzzing, which will act as the *timbral source* for the mosaicing and a fragment of "Let It Be" by the Beatles, which is the *target* to be synthesized with the sounds of bees buzzing. The main idea is to produce a signal sounding as if buzzing bees were "playing" "Let It Be" by the Beatles by buzzing at different pitches or frequencies.

```
1  % initialization
2  inpPath = 'data/';
3  outPath = 'output/';
4
5  filenameSource = 'Bees_Buzzing.wav';
6  filenameTarget = 'Beatles_LetItBe.wav';
7
8  % 1. load the source and target signal
9  % read signals
10 [xs,fs] = audioread([inpPath filenameSource]);
11 [xt,fs] = audioread([inpPath filenameTarget]);
12 % make monaural if necessary
13 xs = mean(xs,2);
14 xt = mean(xt,2);
15
16 % 2. compute STFT of both signals
17 % spectral parameters
18 paramSTFT.blockSize = 2048;
19 paramSTFT.hopSize = 1024;
20 paramSTFT.winFunc = hann(paramSTFT.blockSize);
21 paramSTFT.reconstMirror = true;
22 paramSTFT.appendFrame = true;
23 paramSTFT.numSamples = length(xt);
24
25 % STFT computation
26 [Xs,As,Ps] = forwardSTFT(xs,paramSTFT);
27 [Xt,At,Pt] = forwardSTFT(xt,paramSTFT);
28
29 % get dimensions and time and freq resolutions
30 [numBins,numTargetFrames] = size(Xt);
31 [numBins,numSourceFrames] = size(Xs);
32 deltaT = paramSTFT.hopSize / fs;
33 deltaF = fs / paramSTFT.blockSize;
34
35 % 3. apply continuity NMF variants to mosaicing pair
36 % initialize activations randomly
37 H0 = rand(numSourceFrames,numTargetFrames);
38
39 % init templates by source frames
40 W0 = bsxfun(@times,As,1./(eps+sum(As)));
41 Xs = bsxfun(@times,Xs,1./(eps+sum(As)));
42
43 % parameters taken from Jonathan Driedger's toolbox
44 paramNMFdiag.fixW = 1;
45 paramNMFdiag.numOfIter = 20;
46 paramNMFdiag.continuity.polyphony = 10;
47 paramNMFdiag.continuity.length = 7;
48 paramNMFdiag.continuity.grid = 5;
49 paramNMFdiag.continuity.sparsen = [1 7];
50
51 % reference implementation by Jonathan Driedger
52 [nmfdiagW, nmfdiagH] = NMFdiag(At, W0, H0, paramNMFdiag);
53
54 % create mosaic, replace magnitude by complex frames
55 contY = Xs*nmfdiagH;
56
57 % visualize
58 paramVis = [];
59 paramVis.deltaF = deltaF;
60 paramVis.deltaT = deltaT;
61 fh1 = visualizeComponentsNMF(At, nmfdiagW, nmfdiagH, [],
        paramVis);
62
63 % save result
64 saveas(fh1,[outPath,'LetItBee_NMFdiag.png']);
65
66 % resynthesize using Griffin-Lim, 50 iterations by default
67 [Xout, Pout, res] = LSEE_MSTFTM_GriffinLim(contY, paramSTFT);

68
69 % save result
70 audiowrite([outPath,'LetItBee_NMFdiag_with_target_',
        filenameTarget],res,fs);
```

Listing L.3: Example of audio mosaicing using NMF model with diagonality-enhanced activation matrix and fixed templates, following [24].

We will now go through the code in Listing L.3. In lines 2–14 we load the audio files for both source and target signals, making them mono for further processing. We then compute the source spectrograms `Xs` (complex-valued), `As` (magnitude), and `Ps` (phase) by calling `forwardSTFT()` in line 26, and the same for the target signal (yielding `Xt`, `At`, and `Pt`, in line 27). In lines 30–33 we obtain the spectrogram dimensions, as well as the time and frequency resolutions under the current settings. Since we want to learn activations that will tell us which source frames to use to synthesize the target, we initialize the activation matrix `H0` with random values in line 37. We then initialize the template matrix `W0` with the source magnitude spectrogram `As`, normalized so that each column has unit sum (line 40). We also normalize the complex-valued spectrogram `Xs` on line 41. The following code block, in lines 44–49, is the most important for this example, since we set crucial parameters by using the structure `paramNMFdiag`. In line 44 we set `fixW = 1`, indicating that the template matrix `W0` should not be updated during the NMF learning (i. e., we do not want to modify the source's timbral characteristics). In line 45 we set the number of iterations to 20. In line 46 we set the degree of polyphony to 10—this means that during learning, for every column in the activation matrix, the 10 highest entries will keep their original magnitude, and the rest will be scaled down. In line 47 we set `continuity.length` to 7, which controls the filter kernel length that will be used to smooth the diagonal lines that we wish to enhance in the activation matrix. This choice is empirical since

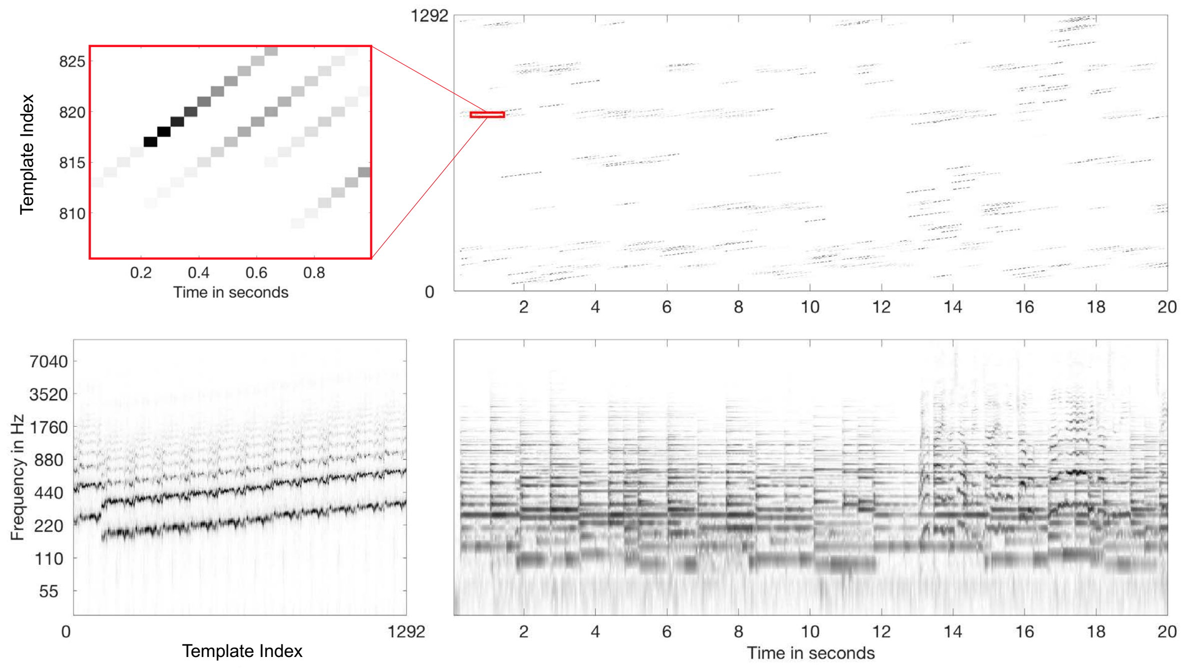

Figure 3: Visual output generated by code in Listing L.3: diagonally enhanced NMF for audio mosaicing. Top right: Diagonally enhanced activation matrix H. Top left: Detail of diagonal structures in subregion of activation matrix (subplot was added manually). Bottom right: Approximated magnitude spectrogram resulting from the diagonal NMF model $\mathrm{U} = \mathrm{W} \cdot \mathrm{H}$. Bottom left: Magnitude spectrogram used as timbral source for mosaicing, used directly as template matrix W. The timbral source consists of bees buzzing sounds, pitched up stepwise throughout an entire octave.

it responds to perceived sound quality and can be varied according to combinations of source and target signals. In line 48 we set `continuity.grid` to 5, indicating that we want to filter the activation matrix at every fifth iteration. If the user wishes to intervene less in the NMF algorithm updates, a higher number can be set (i. e., filtering will be performed every `continuity.grid` iterations).
We use the field `parameter.sparsen` (line 49) to sparsen the activation matrix (i. e., increase the distance between neighboring diagonal structures). We now call the main function, `NMFdiag()` (line 52), which returns the templates `nmfdiagW` and diagonally enhanced activations `nmfdiagH`. It only remains to create the mosaic by multiplying the learned activations `nmfdiagH` with the previously normalized source complex-valued spectrogram, in line 55. In lines 58–60 we set the visualization parameters as fields of `paramVis`, and call the visualization function in line 61, producing the output seen in Figure 3, and saving it as a `png` file in line 64. To make the mosaicing result audible, we apply `LSEE_MSTFTM_GriffinLim()` to the complex mosaic spectrogram `contY` (line 67) and write the audio file to disk (line 70).

7. SUMMARY

We have presented the NMF Toolbox, an easy-to-use collection of illustrated code examples intended for research and learning of the principles behind NMF, through real-world music processing applications. In particular, the toolbox provides baseline implementations and a small dataset for tasks such as drum source separation, structure analysis of electronic music, and audio mosaicing. Although the toolbox is not optimized for high execution speed and is not comprehensive (considering the large number of existing NMF variants), we hope that it serves an educational purpose and as a starting point for integrating these techniques into existing projects.

8. ACKNOWLEDGMENTS

This work has been supported by the German Research Foundation (DFG MU 2686/10-1).

9. REFERENCES

[1] D. D. Lee and H. S. Seung, “Algorithms for non-negative matrix factorization,” in *Proc. Neural Information Processing Systems (NIPS)*, Denver, Colorado, USA, November 2000, pp. 556–562.

[2] N. Bertin, R. Badeau, and E. Vincent, “Enforcing harmonicity and smoothness in bayesian non-negative matrix factorization applied to polyphonic music transcription,” *IEEE Trans. Audio, Speech, and Language Processing*, vol. 18, no. 3, pp. 538–549, 2010.

[3] C.-W. Wu and A. Lerch, “Drum transcription using partially fixed non-negative matrix factorization with template adaptation,” in *Proc. Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Málaga, Spain, October 2015, pp. 257–263.

[4] M. Kim, J. Yoo, K. Kang, and S. Choi, "Nonnegative matrix partial co-factorization for spectral and temporal drum source separation," *IEEE J. Selected Topics Signal Processing*, vol. 5, no. 6, pp. 1192–1204, 2011.

[5] F. J. Cañadas-Quesada, D. FitzGerald, P. Vera-Candeas, and N. Ruiz-Reyes, "Harmonic-percussive sound separation using rhythmic information from non-negative matrix factorization in single-channel music recordings," in *Proc. Intl. Conf. Digital Audio Effects (DAFx)*, Edinburgh, UK, September 2017, pp. 276–282.

[6] C. Laroche, H. Papadopoulos, M. Kowalski, and G. Richard, "Drum extraction in single channel audio signals using multi-layer non negative matrix factor deconvolution," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, New Orleans, Louisiana, USA, March 2017, pp. 46–50.

[7] C. Laroche, M. Kowalski, H. Papadopoulos, and G. Richard, "Hybrid projective nonnegative matrix factorization with drum dictionaries for harmonic/percussive source separation," *IEEE/ACM Trans. Audio, Speech, and Language Processing*, vol. 26, no. 9, pp. 1499–1511, 2018.

[8] D. F. Yela, S. Ewert, D. FitzGerald, and M. B. Sandler, "Interference reduction in music recordings combining kernel additive modelling and non-negative matrix factorization," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, New Orleans, Louisiana, USA, March 2017, pp. 51–55.

[9] C. Dittmar, P. López-Serrano, and M. Müller, "Unifying local and global methods for harmonic-percussive source separation," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, Calgary, Canada, April 2018, pp. 176–180.

[10] P. López-Serrano, M. E. P. Davies, J. Hockman, C. Dittmar, and M. Müller, "Break-informed audio decomposition for interactive redrumming," in *Late Breaking and Demo Session of the Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Paris, France, September 2018.

[11] T. Nakamura, H. Kameoka, K. Yoshii, and M. Goto, "Timbre replacement of harmonic and drum components for music audio signals," in *Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, May 2014, pp. 7520–7524.

[12] C. Dittmar, K. F. Hildebrand, D. Gärtner, M. Winges, F. Müller, and P. Aichroth, "Audio forensics meets music information retrieval – a toolbox for inspection of music plagiarism," in *Proc. European Signal Processing Conf. (EUSIPCO)*, Bucharest, Romania, August 2012, pp. 1249–1253.

[13] S. Gururani and A. Lerch, "Automatic sample detection in polyphonic music," in *Proc. Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Suzhou, China, October 2017, pp. 264–271.

[14] R. J. Weiss and J. P. Bello, "Unsupervised discovery of temporal structure in music," *IEEE J. Selected Topics in Signal Processing*, vol. 5, pp. 1240–1251, 2011.

[15] F. Kaiser and T. Sikora, "Music structure discovery in popular music using non-negative matrix factorization," in *Proc. Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Utrecht, The Netherlands, August 2010, pp. 429–434.

[16] P. Seetharaman and B. Pardo, "Simultaneous separation and segmentation in layered music," in *Proc. Intl. Conf. Music Information Retrieval (ISMIR)*, New York City, USA, August 2016, pp. 495–501.

[17] S. Ewert and M. Müller, "Using score-informed constraints for NMF-based source separation," in *Proc. IEEE Intl. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, Kyoto, Japan, March 2012, pp. 129–132.

[18] Ö. Izmirli, "Localized key finding from audio using nonnegative matrix factorization for segmentation," in *Proc. Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Vienna, Austria, September 2007, pp. 195–200.

[19] A. Ozerov, E. Vincent, and F. Bimbot, "A general flexible framework for the handling of prior information in audio source separation," *IEEE Trans. Audio, Speech, and Language Processing*, vol. 20, no. 4, pp. 1118–1133, May 2012.

[20] U. Zölzer, *DAFX: Digital Audio Effects*, Wiley Publishing, 2nd edition, 2011.

[21] M. Müller, *Fundamentals of Music Processing*, Springer Verlag, 2015.

[22] P. Smaragdis, "Non-negative matrix factor deconvolution; extraction of multiple sound sources from monophonic inputs," in *Proc. Intl. Conf. Independent Component Analysis and Blind Signal Separation (ICA)*, Granada, Spain, September 2004, pp. 494–499.

[23] P. López-Serrano, C. Dittmar, J. Driedger, and M. Müller, "Towards modeling and decomposing loop-based electronic music," in *Proc. Intl. Conf. Music Information Retrieval (ISMIR)*, New York City, USA, August 2016, pp. 502–508.

[24] J. Driedger, T. Prätzlich, and M. Müller, "Let It Bee – Towards NMF-inspired audio mosaicing," in *Proc. Intl. Soc. for Music Information Retrieval Conf. (ISMIR)*, Málaga, Spain, October 2015, pp. 350–356.

[25] A. Cichocki, R. Zdunek, A. H. Phan, and S. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*, John Wiley and Sons, 2009.

[26] C. Dittmar and M. Müller, "Reverse engineering the Amen break – score-informed separation and restoration applied to drum recordings," *IEEE/ACM Trans. Audio, Speech, and Language Processing*, vol. 24, no. 9, pp. 1531–1543, 2016.

[27] J. Driedger, H. Grohganz, T. Prätzlich, S. Ewert, and M. Müller, "Score-informed audio decomposition and applications," in *Proc. ACM Intl. Conf. Multimedia (ACM-MM)*, Barcelona, Spain, October 2013, pp. 541–544.

[28] D. W. Griffin and J. S. Lim, "Signal estimation from modified short-time Fourier transform," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[29] A. Liutkus and R. Badeau, "Generalized Wiener filtering with fractional power spectrograms," in *Proc. IEEE Intl. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, Australia, April 2015, pp. 266–270.

[30] C. Dittmar, J. Driedger, M. Müller, and J. Paulus, "An experimental approach to generalized Wiener filtering in music source separation," in *Proc. European Signal Processing Conf. (EUSIPCO)*, Budapest, Hungary, August 2016, pp. 1743–1747.