NMFLAB for Signal Processing

Toolbox for
NMF (Non-negative Matrix Factorization)
and
BSS (Blind Source Separation)

Andrzej CICHOCKI and Rafal ZDUNEK

May 25, 2006

The NMFLAB Package for Signal Processing has been developed, designed, and implemented in MATLAB by Andrzej **CICHOCKI** and Rafal **ZDUNEK**.

The toolbox consists of various NMF algorithms and some generalizations written in MATLAB®. They have been extensively tested by Rafal **Zdunek** and Andrzej **Cichocki** in collaboration with Shun-ichi **Amari**, Raul **Kompass**, Seungjin **Choi** , Sergio **Cruces-Alavarez**, Scott **Rickard,** Ruairi **de Frein,** Zhaoshui **He,** Zhe **Chen**, Gen **Hori,** Yoshikazu **Washizawa,** Arao **Funase,** and other members of the Laboratory for Advanced Brain Signal Processing.

The graphic design, data visualization and user interface, is partially adopted from the ICALAB ver. 1.5 developed by Andrzej **Cichocki**, Shun-ichi **Amari**, Krzysztof **Siwek,** Toshihisa **Tanaka**, Sergio **Cruces-Alvarez**, et al.

The current version 1 of the NMFLAB is as of May 25, 2006.

A similar NMFLAB package has been developed for **Image Processing,** which will be released soon.

The references for both of these Toolboxes are given below:

[1] A. Cichocki and R. Zdunek, NMFLAB – MATLAB Toolbox for Non-Negative Matrix Factorization,
[2] A. Cichocki, R. Zdunek, and S. Amari, "Csiszar's Divergences for Non-Negative Matrix Factorization: Family of New Algorithms", 6th International Conference on Independent Component Analysis and Blind Signal Separation, Charleston SC, USA, March 5-8, 2006 Springer LNCS 3889, pp. 32-39. [.pdf]
[3] A. Cichocki, S. Amari, R. Zdunek, R. Kompass , G. Hori and Z. He: "Extended SMART Algorithms for Non-Negative Matrix Factorization", 8th International Conference on Artificial Intelligence and Soft Computing, ICAISC, Zakopane, Poland, 25-29 June, 2006. [.pdf]

[4] R. Zdunek, and A. Cichocki, "Non-Negative Matrix Factorization with Quasi-Newton Optimization", 8th International Conference on Artificial Intelligence and Soft Computing, ICAISC, Zakopane, Poland, 25-29 June, 2006. [.pdf]
[5] A. Cichocki, R. Zdunek, and S. Amari, "New Algorithms for Non-Negative Matrix Factorization in Applications to Blind Source Separation", 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP2006, May 14-19, 2006, Toulouse, France. [.pdf]
[6] R. Zdunek and A. Cichocki, "Conjugate Gradient (GPCG) and Quasi Newton Hybrid Methods for NMF (submitted for publication).
[7] A. Cichocki and R. Zdunek, "Multilayer Nonnegative Matrix Factorization Using Projected Gradient Approaches", (ICONIP-2006).
[8] A. Cichocki, R. Zdunek, S. Amari, G. Hori and K. Umeno, "Blind Signal Separation Method and System Using Modular and Hierarchical-Multilayer Processing for Blind Multidimensional Decomposition, Identification, Separation or Extraction", Patent pending, Riken, Japan, March 2006.

Please cite the relevant contribution above if you make use NMFLAB in your research.

DISCLAIMER

The general concept of NMFLAB

The important and unique features of our NMFLAB toolbox are as follows:

1. Implementation of a variety of NMF algorithms: Multiplicative Algorithms (MA), Exponentiated Gradient (EG), Projected Gradient (PG), Conjugate Gradient (CG), and Quasi-Newton (QN), some of which are new. The user can easily compare various algorithms and also different combination of these algorithms.

2. Possibility of imposing some additional constraints such as sparseness and/or smoothness.

3. Possibility of comparing the performance of various NMF and ICA/BSS algorithms for difficult benchmarks including benchmarks based on Monte Carlo Analysis.

4. Introduction of the new concept of multilayer or cascade NMF that may considerably improve reliability and performance of some NMF algorithms especially for ill-conditioned problems.

5. Preprocessing and post-processing of data. (Currently the optional **PREPROCESSING** tools are not implemented in the version 1, except for Principal Component Analysis (**PCA**) preprocessing that estimates the number of hidden components. **POSTPROCESSING** tools currently include: Deflation and Reconstruction ("cleaning") of the data by removing undesirable components, noise or artifacts.)

6. The NMFLAB Toolbox has a flexible and extendable structure which can be used to add user defined NMF algorithms and compare them with the developed and implemented algorithms.

A Brief Introduction to NMF

Basic NMF algorithms perform matrix factorization of the form:

$$\mathbf{Y}=\mathbf{A}\,\mathbf{X}+\mathbf{V} \text{ subject to non-negativity constraints for } \mathbf{A} \text{ and } \mathbf{X},$$

where $\mathbf{A}$ is an m-by-r non-negative mixing matrix or a matrix of basis vectors, $\mathbf{Y}$ is an m-by-T matrix of the observed data, $\mathbf{X}=\mathbf{S}$ is an r-by-T matrix of original non-negative sources, and $\mathbf{V}$ is an m-by-T and represents the matrix of additive noise.

Successful and efficient use of NMFLAB depends on an *a priori* knowledge, common sense and appropriate use of the advanced parameters (see the list of the publications for details) used in the various algorithms implemented in the toolbox.

Possible Applications of NMFLAB

NMFLAB could be useful in the following tasks:

1. Decomposition of multi-variable signals or patterns into non-negative (and usually sparse components),
2. Dimensionality reduction,
3. Extraction of features and patterns,
4. Extraction and removal of undesirable artifacts and interference by applying deflation,
5. Removal of noise or "cleaning" the raw observed (sensor) data,
6. Comparison of the performance for various NMF algorithms and Blind Source Separation (BSS),
7. Data analysis and data mining,
8. NMF analysis of acoustic signals and text data, and
9. Classification, clustering and segmentation of patterns, e.g., face/image recognition, language modeling, speech processing and gene analysis.

Several benchmarks signals are included which enable the user to compare and illustrate the performance of the various algorithms on synthetic and real-world signals (see Benchmarks).

Limitations of the version 1:

The version of the NMFLAB is limited to **200** channels and some combinations of algorithms may not work properly (this is an implementation constraint not a mathematical one). A higher number of channels may be useful for high-density array processing in biomedical signal processing and in multivariate time series analysis and is available on request.

Note: this package runs on MATLAB 7.0 or higher. **NMFLAB for Signal Processing** was developed and tested under MATLAB version 7.1. **Previous versions (i.e., 6.5) may not work properly due to some unsupported graphics functions.**

PART 1

NMFLAB User Guide

Starting NMFLAB

To start **NMFLAB for Signal Processing** type:

```
nmflab
```

in the **MATLAB** Command Window.

Loading the data

To load new signals or data for further processing:

1. Click on `File` in the menu bar and then select the first option on the drop-down menu `Open`. Both the `HELP` and `EXIT` buttons in the main window will become active after loading the data file. The user can load real data (the matrix of observed data – matrix **Y**) or synthetic data (representing original sources - matrix **X**) to test and compare various NMF algorithms.

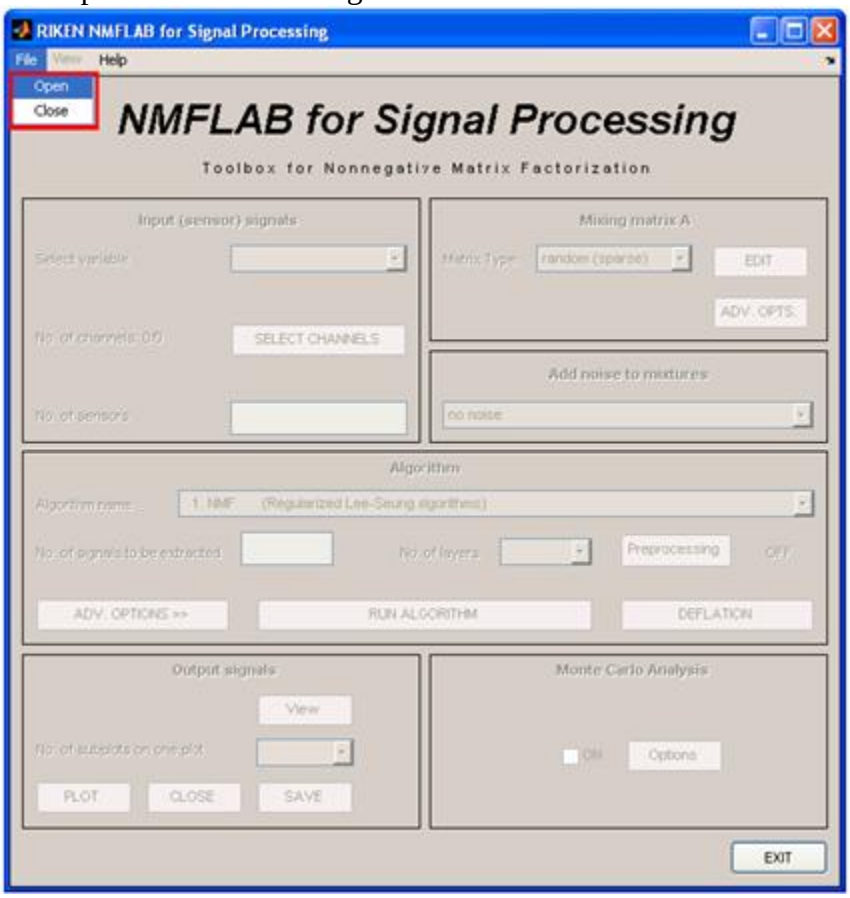


Fig. 1: The initial window that appears after running "nmflab". Click `File` and `Open` to load your data. The data should be in MATLAB format (*.mat), ASCI format (*.txt and *.dat) or Excel formats (*.xls and *.csv).

2. The loaded data (both the supplied benchmarks and the user's test data) should be stored in the MATLAB (*.mat files), ASCII (*.txt and *.dat files) or Excel (*.xls, *.csv) format.

   The data must be stored in a 2-dimensional matrix: the number of rows corresponds to the number of sensors (the number of observations) and the number of columns corresponds to the length of the signals, i.e., the number of samples.

   The loaded Matlab files can contain one or more data matrices. The user can then choose an appropriate data matrix. The data loaded in ASCII or Excel format should contain only one data file. There is no limit to the length of the signals (number of samples), but in the version 1 available on the web, only a maximum of **200 signals** (time series) can be loaded and processed.

   Usually, the convergence speed of the algorithms in this package is heavily dependent on the dimension of signals (i.e., the number of observations and the number of samples) and the computer resources, e.g., the processor speed and the available memory. To process large files, it is recommended that the swap space on a hard disk is increased and/or the data is split into smaller dimension – sub-windows.

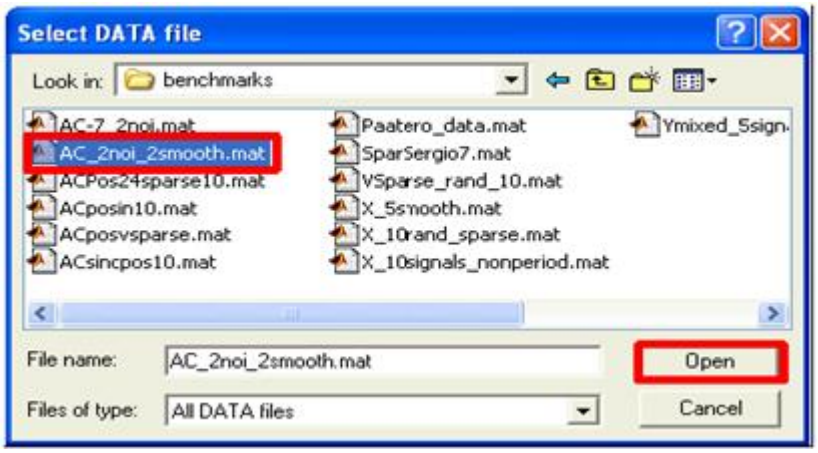


Fig. 2: This window illustrates how to load the benchmark or the user's data. Click the `Open` button when the desired data file has been selected.

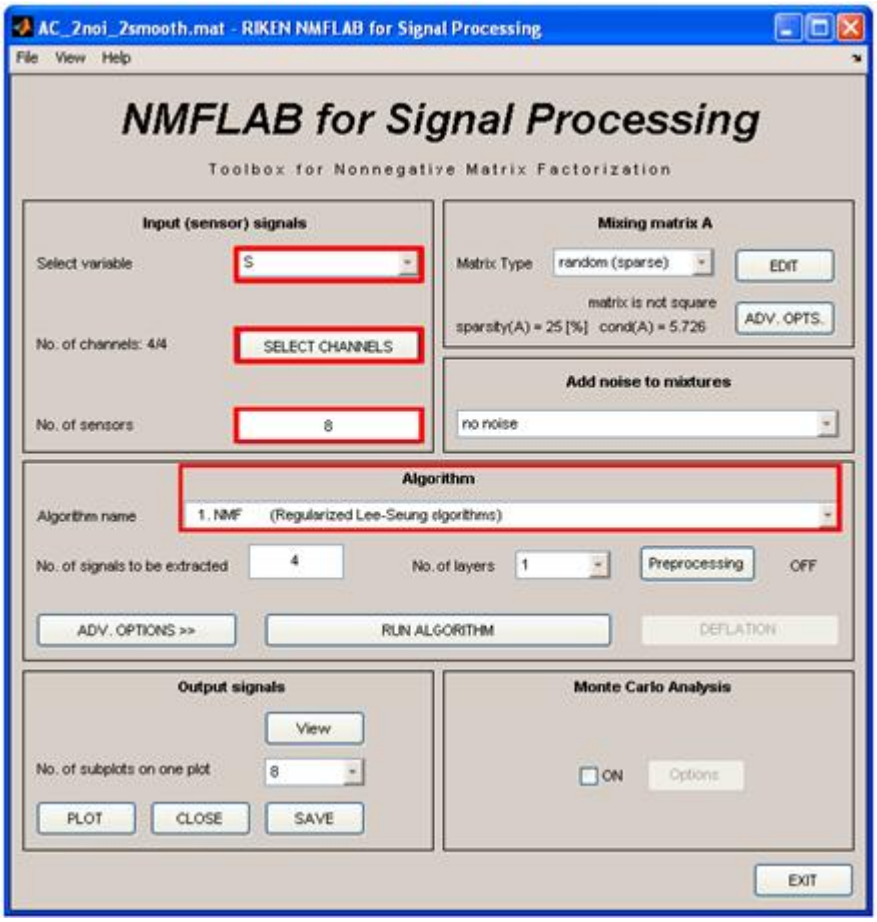


Fig. 3: An example of the main window after the data has been loaded. If the user's MATLAB data consist of several matrices, click the `Select variable` popup menu to choose the desired data. The user can also click `SELECT CHANNELS` in order to choose the time window (the number of samples) and the desired signals or `Algorithm` to select one of the algorithms (The basic Regularized `Lee-Seung` algorithm (EMML) is the default algorithm).

3. The user can discard some signals and select an arbitrary (time) sub-window by clicking the *Select channels* button. The *Select channels* window allows the user to mark the signals (or data) desired for further processing. The user can also choose a specific time window for the input signals (number of samples) by entering the first and last sample numbers into the fields marked as `start` and `end` at the bottom of the window. The numbers in the respective fields specify the current starting and ending positions for the processing. Pressing the `OK` button (In the window below, this appears in the highlighted area) loads the selected signals. Unselected channels (in our example, channels No. 2 and 4) are ignored (and removed from original data). Pressing `SAVE` stores the selected signals as a *.mat file.

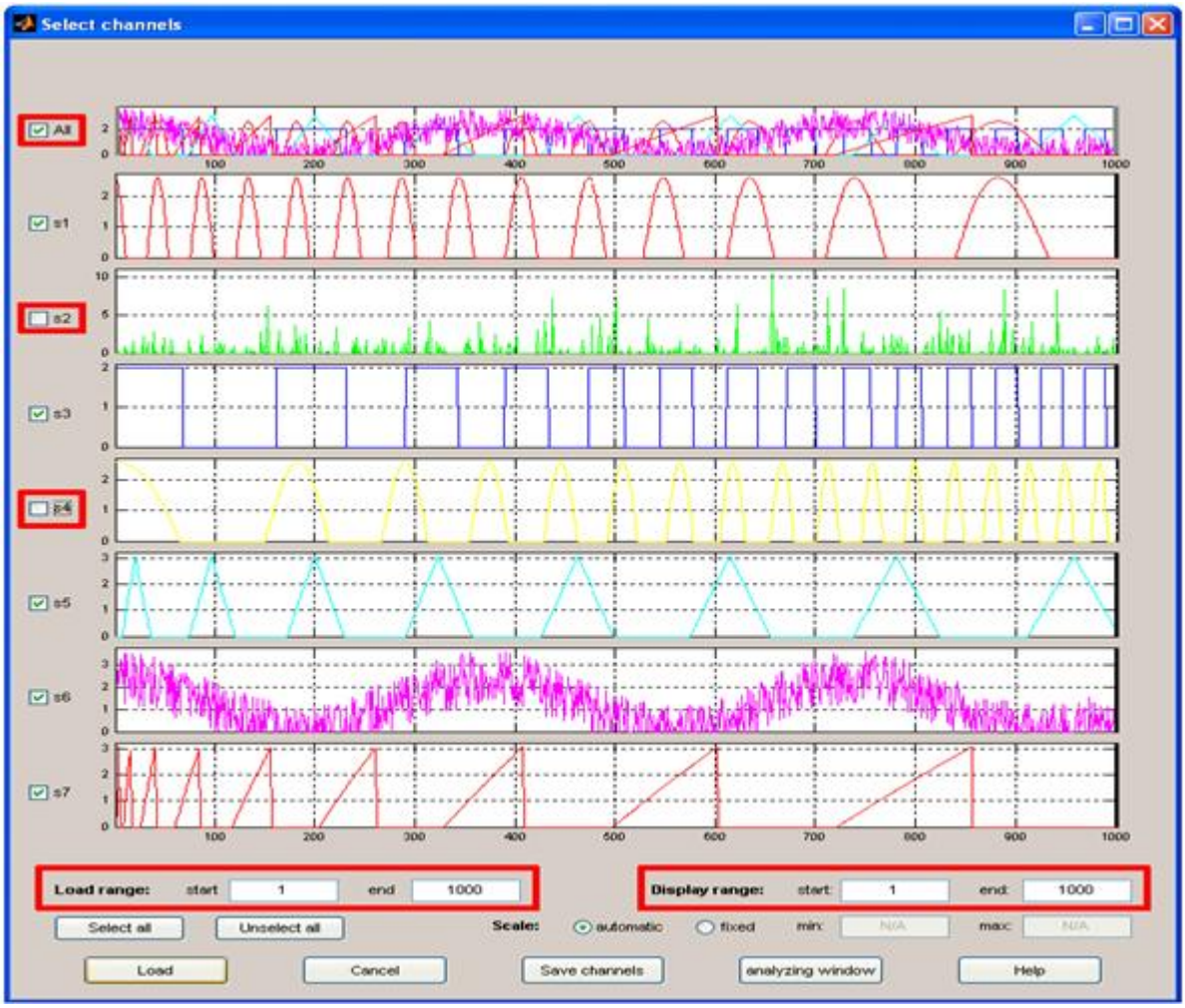


Fig. 4: This window illustrates the selection of channels and the time window. In this case channels s2 and s4 are deleted.

Mixing the signals

The signals can be mixed synthetically (using the mixing model: **Y**=**AX**+**V**) if they are not already in mixed inputted file.

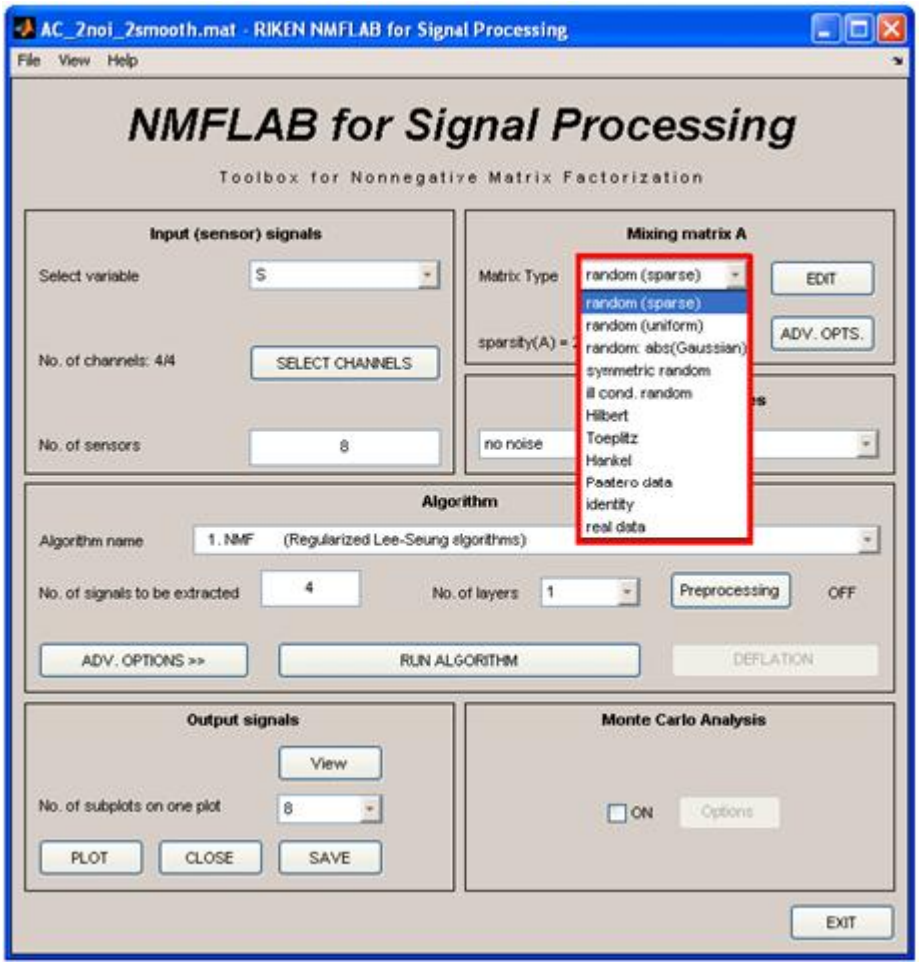


Fig. 5: This window illustrates how to choose mixing matrix A. If the data is real (i.e. user inputted source signals and not benchmark data), choose the `real data` option.

The mixing matrix **A** can be square or rectangular with the number of columns and rows specified by the user. Several alternative ways to generate such mixing matrices are available:

- Randomly generated full rank sparse matrix with the sparseness level adjusted by the user,
- Randomly generated full rank uniform distributed matrix: **A**=rand(m,n),
- Randomly generated full rank exponentially distributed matrix: **A**=abs(randn(m,n)),
- Randomly generated symmetric nonsingular square matrix,
- Randomly generated ill-conditioned uniform distributed matrix,
- Hilbert matrix (very ill conditioned),
- Toeplitz matrix,
- Hankel matrix,
- Paatero example,
- Identity (unit matrix).

Fig. 6: Pressing the `ADV. OPTS.` button causes the following window to appear. The user can set the sparseness level using this window.

Editing the mixing matrix can be achieved by clicking the `EDIT` button, which generates the following window:

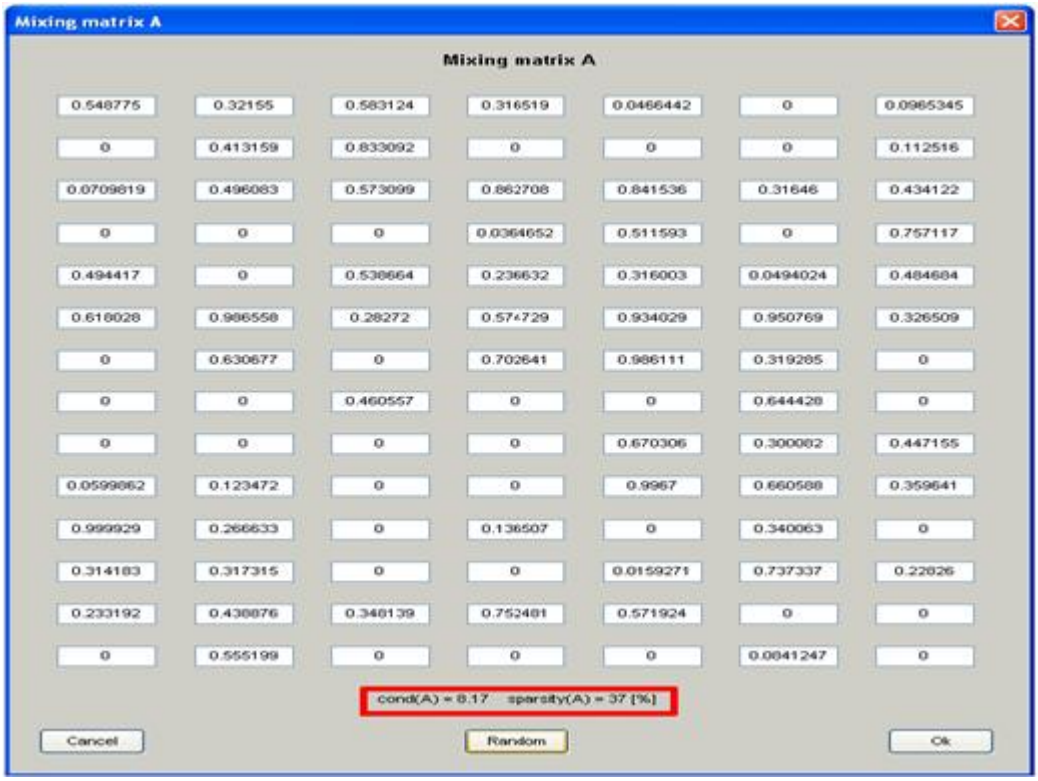


Fig. 7: This window illustrates how to edit the mixing matrix A. The editable mixing matrix A cannot be larger than a 20-by-14.

The user can to edit every element of the mixing matrix using the window in figure 7. The condition number of the mixing matrix is updated automatically. The maximum mixing matrix allowed is a *20*-by-*14*.

Adding noise to the signals

Noise can be added to each sensor signal before performing NMF or BSS by choosing one of the following options:

- **No noise** - the sensor signals are not changed.
- **Gaussian noise** with a SNR in the range 20dB to 0dB can be added to the signals.
- **Uniform noise** with a SNR in the range 20dB to 0dB can be added to the signals.

Remark: The negative entries of the matrix **Y** are ignored, that is, they are replaced with zero-entries. Noise can be added to test the robustness of the algorithms using the various benchmarks.

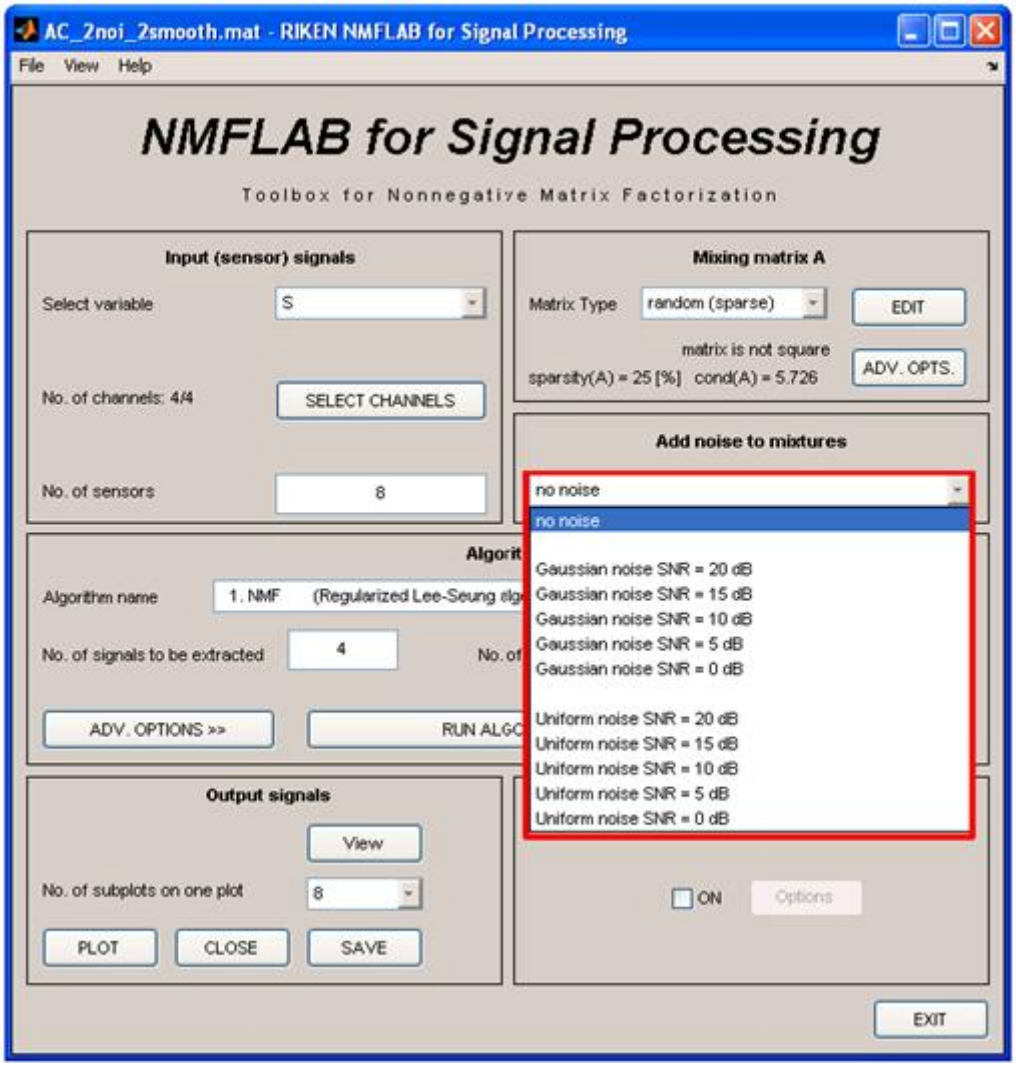


Fig. 8: This window illustrates the procedure of adding noise synthetically to the signal. This option should be ignored for real data.

Choosing the algorithm

Once the data (sources) and the mixing matrix (optional) have been chosen, the user can select one of the available algorithms to run on the data. Part 2 of the NMLAB manual gives more details on each of them. The algorithms have been arranged into groups. Currently the following groups of algorithms are implemented:

1. Regularized Lee-Seung NMF algorithms. The algorithms in this group minimize the regularized Frobenius norm, Kullback-Leibler divergence or beta divergence (which integrates both loss functions). This group contains the following algorithms: EMML, ISRA, Kompass algorithm, and projected pseudo-inverse.

2. Projected gradient algorithms: GPCG (Gradient Projected Conjugate Gradient), PG (Projected Gradient), IPG (Interior Projected Gradient), Regularized MRNSD (Regularized Minimal Residual Norm Steepest Descent), Relaxed Hellinger, and Projected pseudo-inverse.

3. NMF-ALPHA -Multiplicative algorithms based on the Amari's alpha divergence.

4. SMART algorithms based on the Exponentiated Gradient (EG) and various divergences.

4. Second-order algorithms (mixing matrix is estimated using algorithms based on the quasi-Newton method).

6. Cascade algorithms that estimate the missing matrix as a chain of matrices.

In order to compare performance of NMF algorithms with BSS/ICA we added three standard BSS algorithms:

7. AMUSE algorithm (ICA),

8. SOBI algorithm (ICA),

9. ThinICA,

10. User-defined algorithms (User algorithm 10-15).

The user-defined algorithms can be specified arbitrarily by the user. Refer to the example m-files: user_alg*k*.m to see how the NMFLAB calls the user-defined algorithms. The user-defined algorithm should return only the estimated mixing matrix **A** and estimated sources **X.**

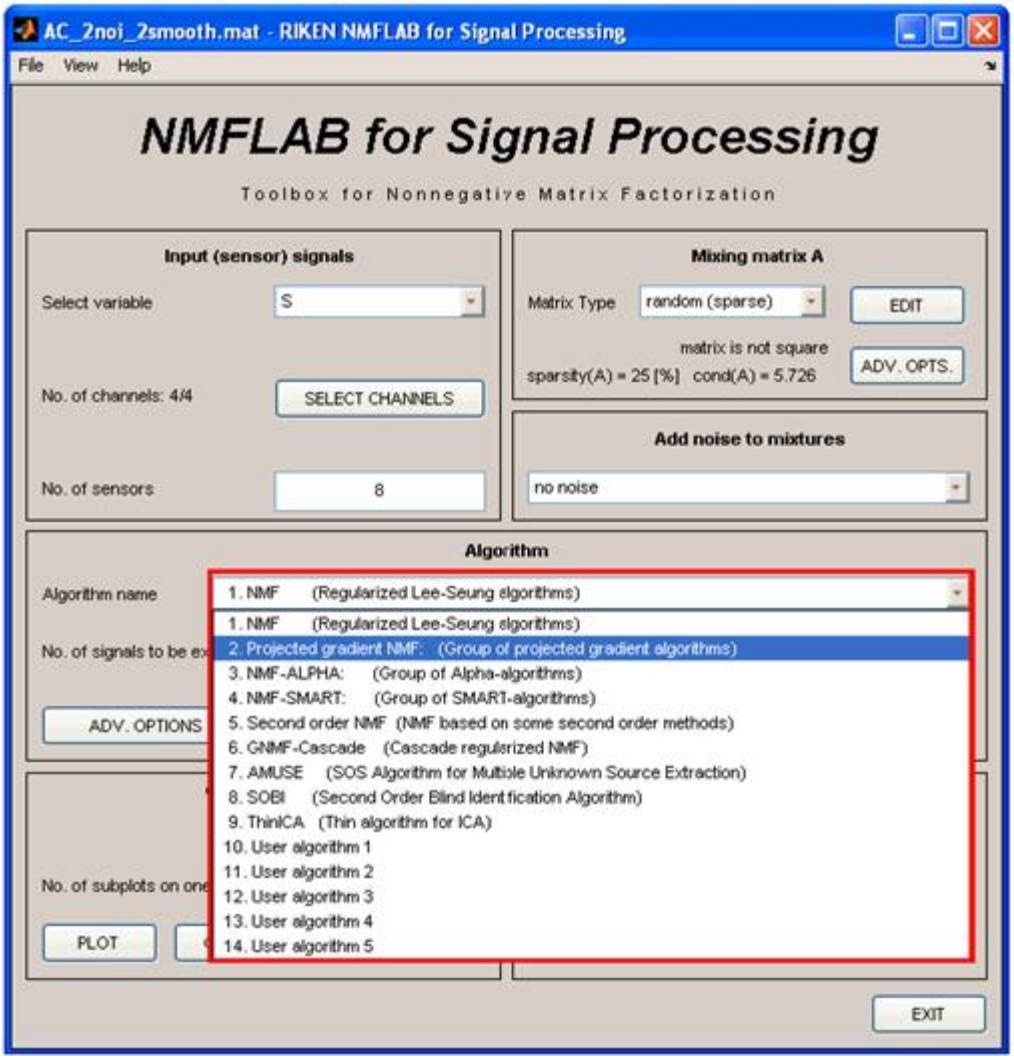
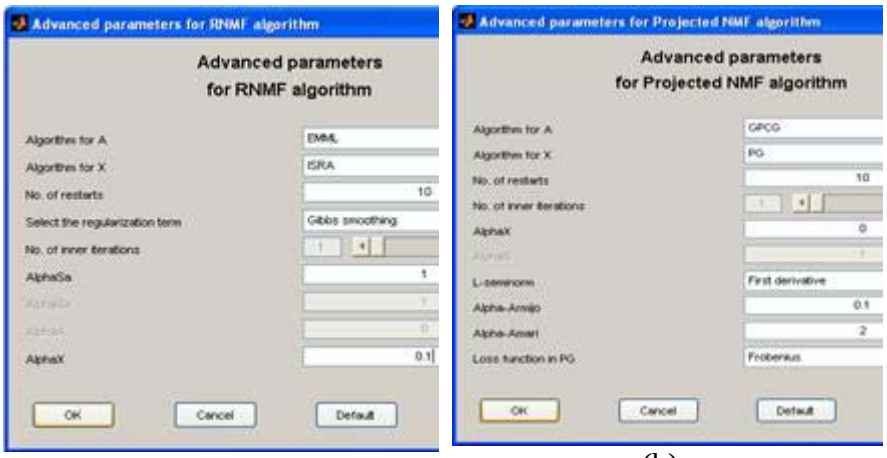


Fig. 9: This window illustrates how to select the algorithm from the list of available NMF algorithms.



(a) (b)

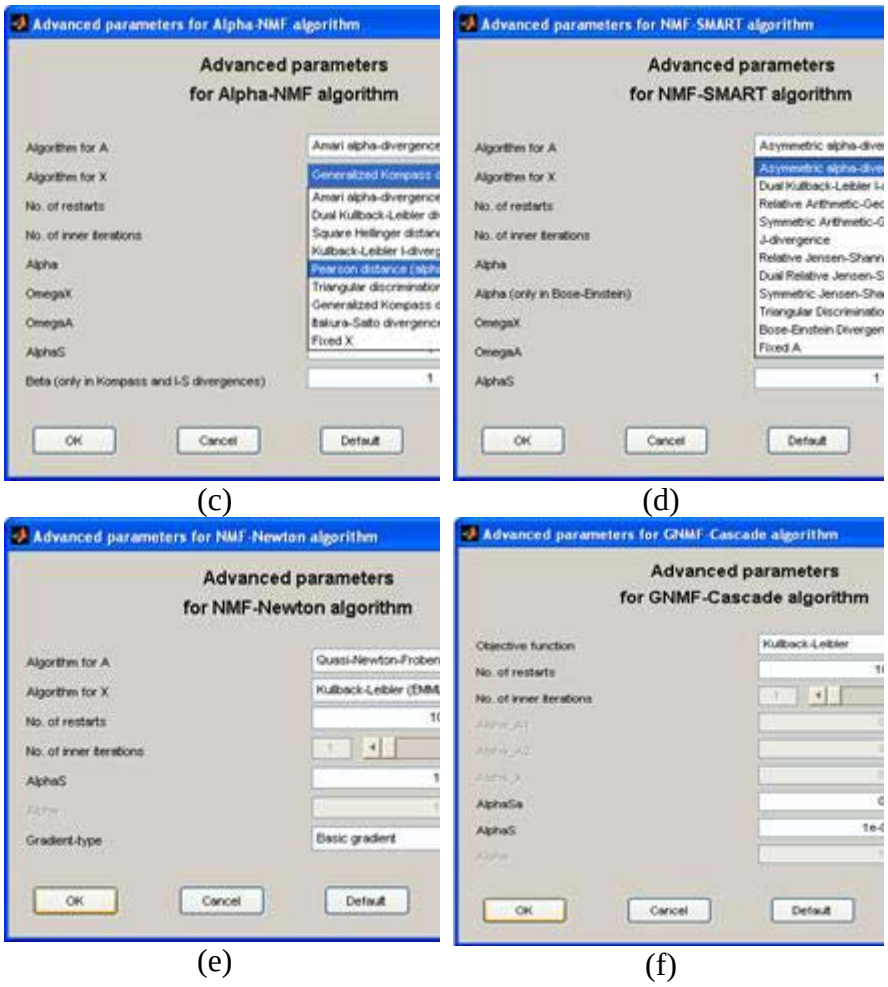


Fig. 10: This window illustrates how to adjust the parameters using `Advanced Options`.

In the package, most of the algorithms are given with default parameters; hence the user can start testing the algorithms without the needing to set the parameters. The default parameters are tuned for near optimum values for typical data. It is necessary to tune some parameters to achieve optimal performance in the case of some of the algorithms. This can be done by clicking on the `ADV. OPTIONS` button. When the algorithm has been selected and its free parameters adjusted, click the `RUN ALGORITHM` button. The learning procedure will be started and algorithm specific messages will appear in the main MATLAB command window. During the computation, an additional window will display the algorithm name. You can stop the learning process by clicking on the `INTERRUPT` button.

In the version 1 of NMFLAB, the interrupting feature is disabled for almost all the algorithms.

Fig. 11: While the algorithm is running the following window should appear. The process can be interrupted by the user by clicking on the INTERRUPT button.

Multi-layer NMF

In order to improve the performance of NMF, especially for ill-conditioned and badly scaled data and also to reduce the risk of getting stuck in a local minima of the cost function we have developed a simple hierarchical and multi-stage procedure in which we perform a sequential decomposition (factorization) of nonnegative matrices.

In the first step, we perform the basic decomposition $\mathbf{Y} = \mathbf{A}_1\mathbf{X}_1$ using any available NMF algorithm. In the second stage, the results obtained from the first stage are used to perform a similar decomposition: $\mathbf{X}_1 = \mathbf{A}_2\mathbf{X}_2$ using the same or different update rules. This multi-layer decomposition is continued using only the components generated in the last step. The process can be repeated arbitrary many times until some stopping criteria is satisfied. In each step, we usually obtain gradual improvements of the performance. The model has the form: $\mathbf{Y} = \mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_L\mathbf{X}_L$, with the basis matrix defined as $\mathbf{A} = \mathbf{A}_1\mathbf{A}_2 \cdots \mathbf{A}_L$. Physically, this means that we build up a system that has many layers or a cascade connection of L mixing sub-systems. The key point in our novel approach is that the learning (update) process to find parameters of sub-matrices $\mathbf{A}_l$ and $\mathbf{X}_l$ is performed sequentially, i.e., layer by layer.
Thus, our approach can be described by the following algorithm:

Set: $\mathbf{Y}_{(0)} = \mathbf{Y}$, Initialize randomly basis matrix $\mathbf{A}_{(0)}^{(0)}$ and/or $\mathbf{X}_{(0)}^{(0)}$:
For $l = 1, 2, \ldots, L$ do:

For $t = 0, 1, \ldots, T_{max}$ do:

$$\mathbf{X}_{(l)}^{(t+1)} = \arg\min_{\mathbf{X} \geq 0} D\left(\mathbf{Y}_{(l)} \,\|\, \mathbf{A}_{(l)}^{(t)}\mathbf{X}\right)\Big|_{\mathbf{X}=\mathbf{X}_{(l)}^{(t)}},$$

$$\mathbf{A}_{(l)}^{(t+1)} = \arg\min_{\mathbf{A} \geq 0} \tilde{D}\left(\mathbf{Y}_{(l)} \,\|\, \mathbf{A}\mathbf{X}_{(l)}^{(t+1)}\right)\Big|_{\mathbf{A}=\mathbf{A}_{(l)}^{(t)}}, \qquad \mathbf{A}_{(l)}^{(t+1)} \leftarrow \left[a_{ij} / \sum_{i=1}^{m} a_{ij}\right]_{(l)}^{(t+1)},$$

End (for t)

$$\mathbf{Y}_{(l+1)} = \mathbf{X}_{(l)}^{(T_{max}+1)}$$

End (for *l*)

In the above algorithm, the cost functions $D(\mathbf{Y}\|\mathbf{AX})$ and $D(\mathbf{Y}\|\mathbf{AX})$ can take various forms, e.g.: the Amari alpha divergence, Bregman divergence, Csiszar divergence, beta divergence, Euclidean distance.

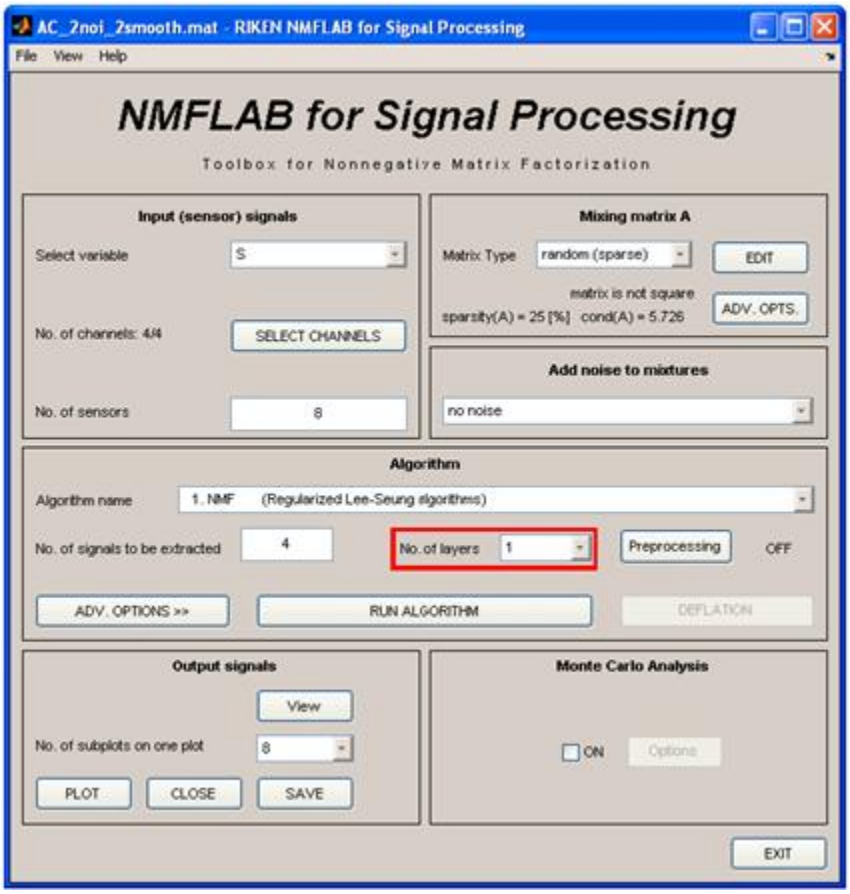


Fig. 12: Selection of the number of layers in Multilayer NMF (maximum 25 layers can be used).

Visualizing the results

When the program finishes computation, we have three sets of parameters:

1. The matrix $\mathbf{A}_{\mathbf{est}}$ which is an estimate of the original mixing matrix **A**. In MATLAB, this matrix has the name **AH**.
2. The matrix $\mathbf{X}_{\mathbf{est}}$ which is an estimate of the original sources **X.** In MATLAB, this matrix has the name **XH**.
    - In practice the data is stored in the matrix form, i.e., $\mathbf{Y} = \mathbf{A}\,\mathbf{X}$, where **A** is a mixing matrix, $\mathbf{Y} = [\mathbf{y}(1), \mathbf{y}(2), ..., \mathbf{y}(T)]$ is m-by-T matrix of observations and $\mathbf{X} = [\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(T)]$ - is the matrix of primary source signals.
3. The vectors SIR_A and SIR_X represent the Signal to Interferer Ratios (**SIR)** of individual columns of the matrix **A** and individual rows of the matrix **X**.

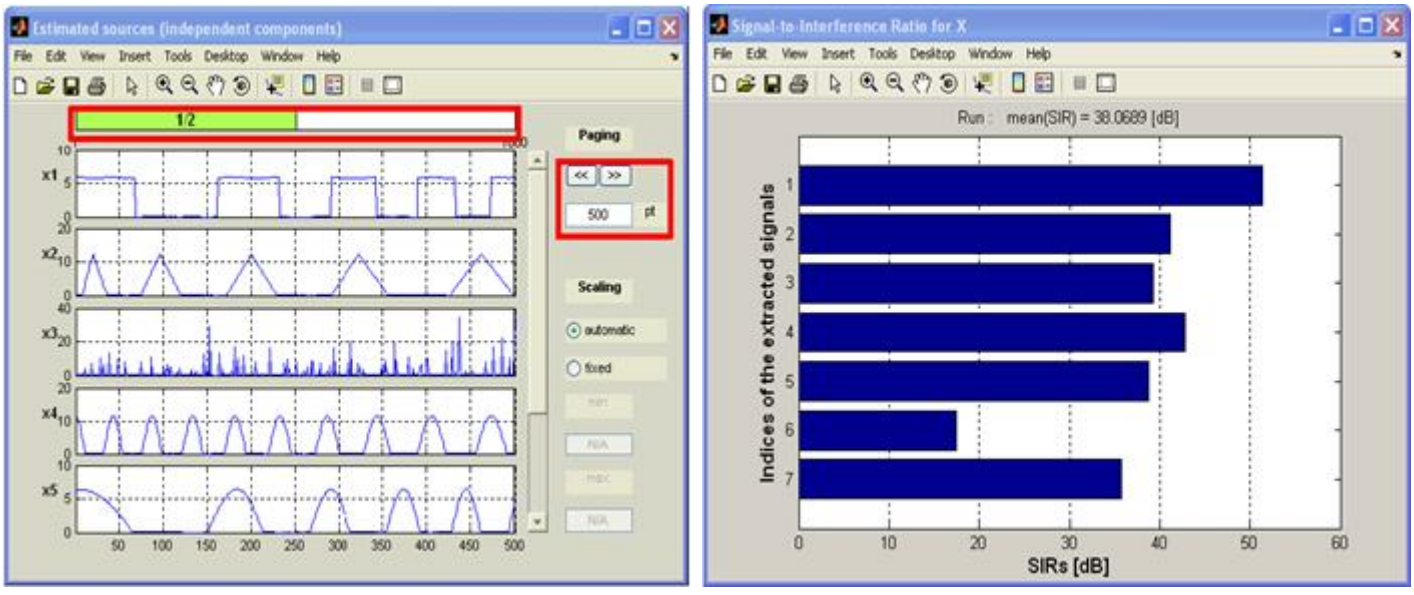
NMFLAB computes SIRs (Signal to Interference Ratio) for individual sources (the rows of the matrix **X** and the columns of the matrix **A).**

After convergence, the results can be visualized by clicking the `PLOT` button. The number of signals *p* displayed simultaneously in every window can be selected by changing the number of subplots. The user can scroll through the signals by dragging the vertical scroll-bar. The number of signals displayed simultaneously is defined by the user in the main NMFLAB window. The user can also scroll through the signals by clicking on the arrow. The estimated source signals are shown in the *Independent components* window. In addition, the loaded signals can be seen in the *Sources* window, which displays the original signals.

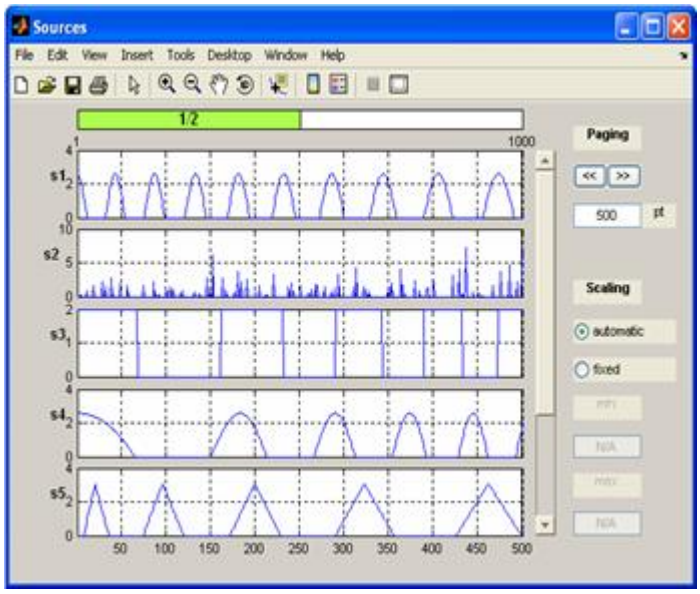
If the user mixed the signals using an arbitrary mixing matrix (**A** *not equal* **I**) it is recommended that the *Sensors (mixtures)* window is viewed. There is no need to mix the signals if the original signals were originally mixed of they consist of real (measured or observed) superimposed data prepared for decomposition into independent (or spatio-temporal de-correlated) components and further processing. The performance of the selected algorithm can be viewed using the *Global Matrix (Performance Index - PI)* window for the artificially mixed sources. Additionally, for each chosen plot window, it is possible to obtain additional subwindow by checking the appropriate `Subwindow` option.

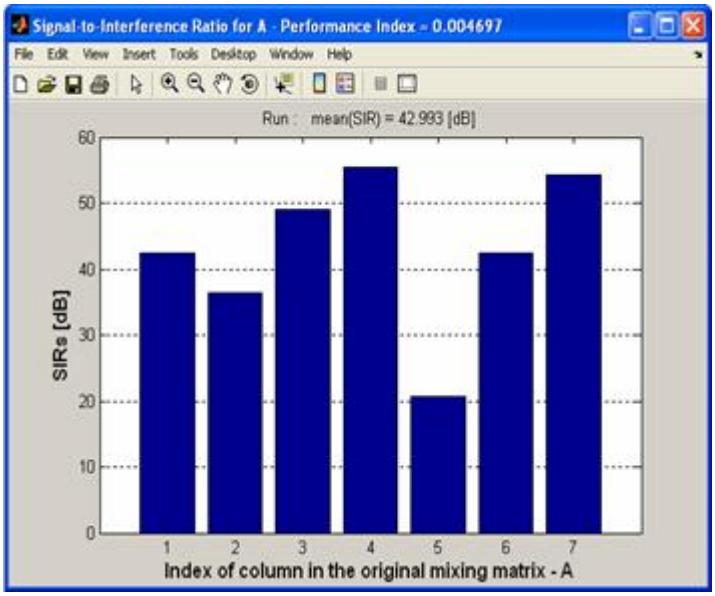
Paging and zooming

The signals that are greater than 1000 samples long are divided automatically into pages (see Fig.12). Each page contains approximately one half of the data samples by default. The user can change the number of samples displayed in each page by replacing the number in the field by an arbitrary integer number. This allows the user to employ the zoom facility on the visualized data. The pages can be changed by clicking on the << and >> arrows or on the bar displayed above the plot (see Figure 12).



(a) **(b)**





(c) **(d)**

Fig. 13: Four windows illustrating the output results: (a) Plot of the estimated non-negative components or separated/extracted (estimated) sources, depending on the algorithm used. (b) Plot of the SIR_A for the estimated columns of matrix A, This plot only makes sense for benchmarks or for known mixing matrix. (c) Plot of the primary sources (if available). If the sources are not available, this window displays the sensor signals. (d) Plot of the SIR_X representing the performance achieved in estimating the individual sources.

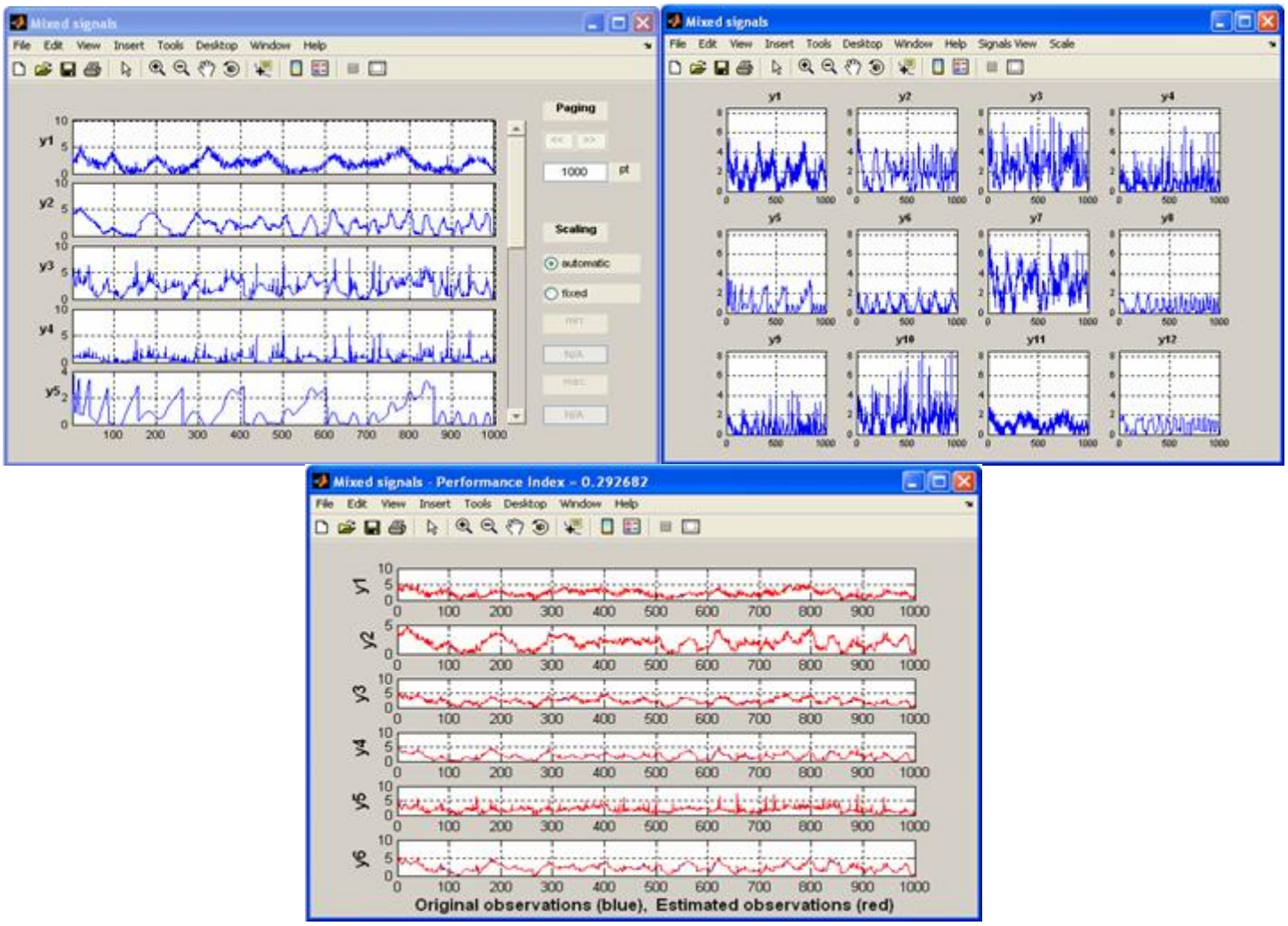


Fig.14 Windows illustrating various visualization of the observed signals

Sub-windows

The sub-windows option offers an additional possible visualization of the estimated sources, sources and mixed signals. Two menu items are available:

- **`Signals view`**

1. Time domain.
2. FFT - Frequency domain.
3. Spectrogram (Caution! This calculation may take a long time).
4. PSD - power spectrum.
5. Scatter plot.

- **`Scale`**

1. Auto / Fixed.
2. Linear / Logarithmic.
3. X / Y Limits.
4. Scatter plot options - see Fig. 17.

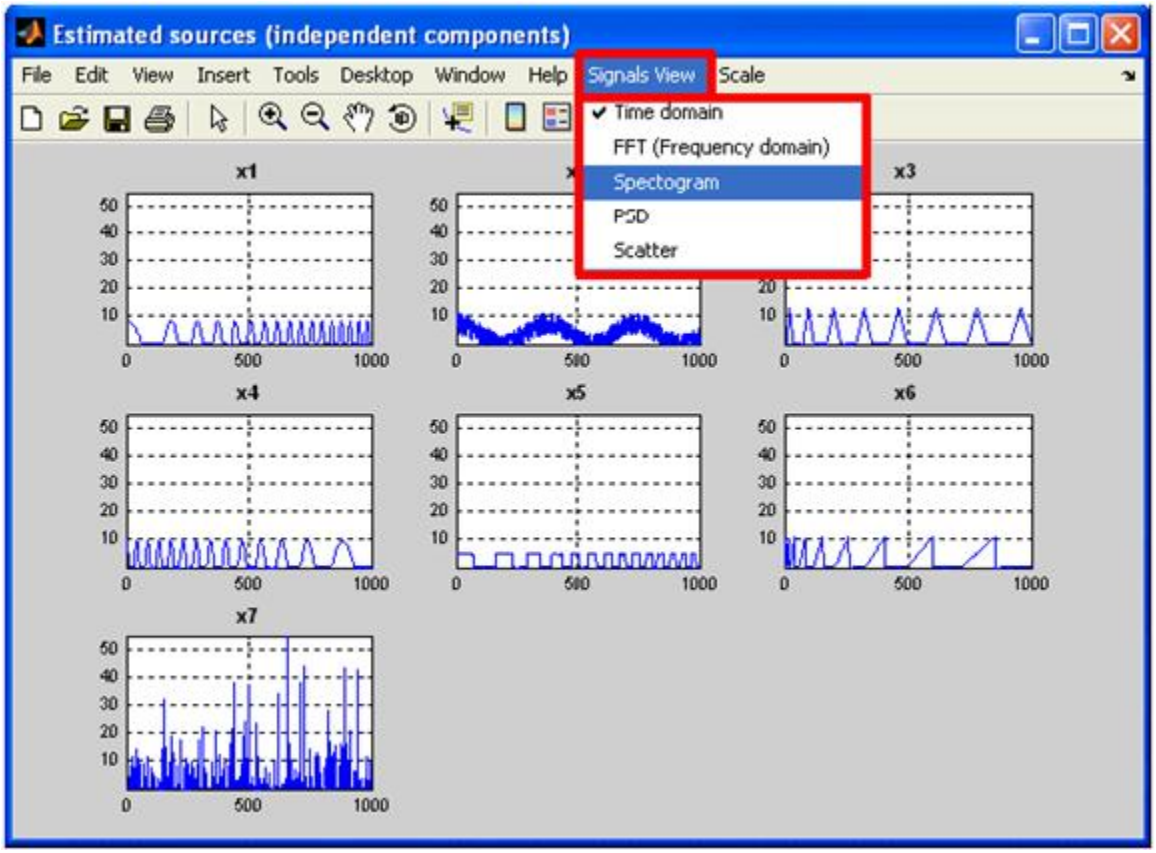


Fig. 15: Sub-windows: visualization of signals in time, frequency and time-frequency domains. Adjustment of scale and type of axes is possible.

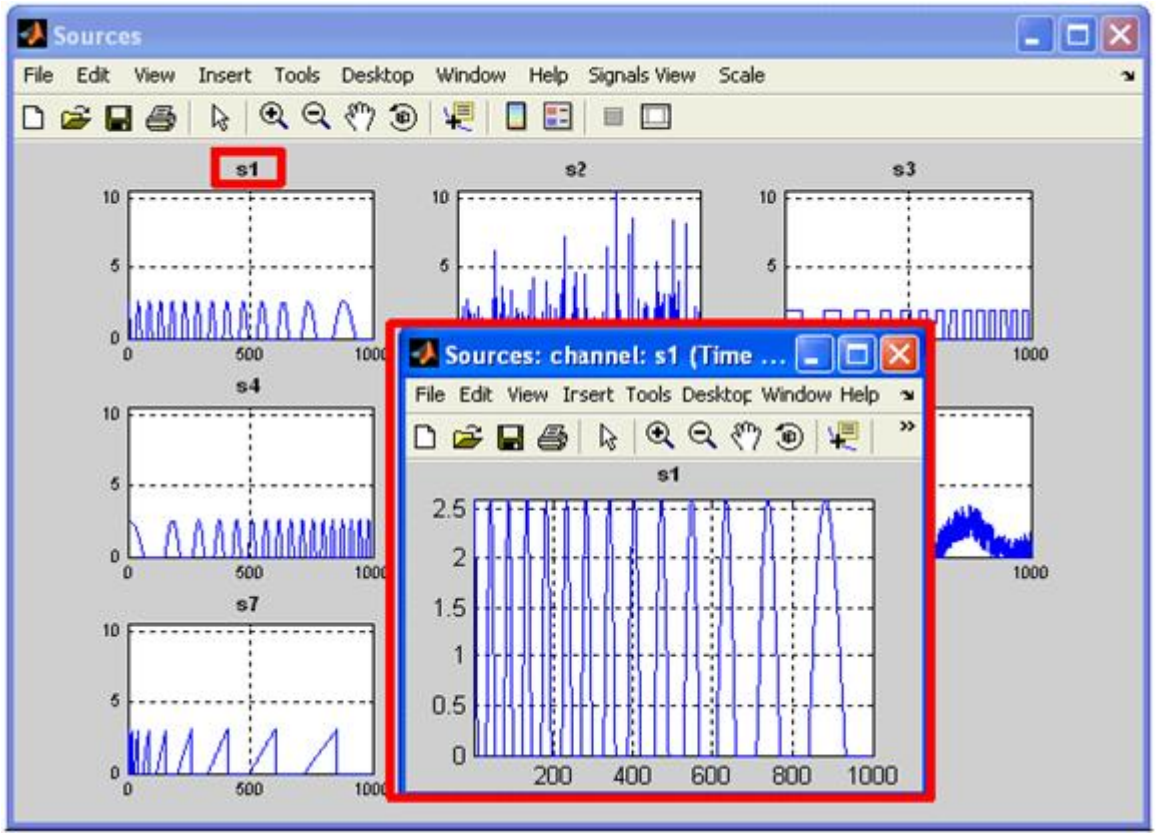


Fig. 16: By clicking on the label of the signal in the main sub-window, it is possible to display an individual signal in a separate, large window.

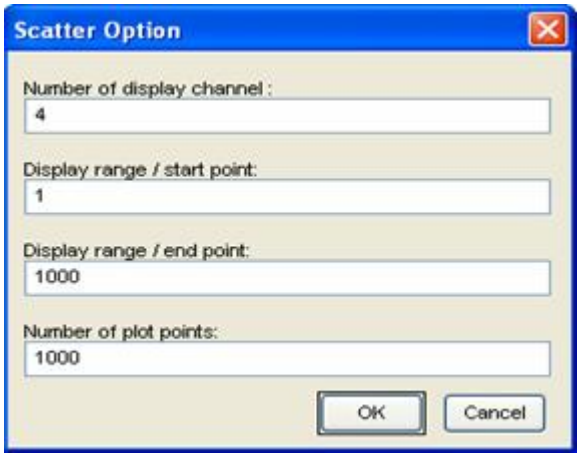


Fig. 17: Scatter plot options.

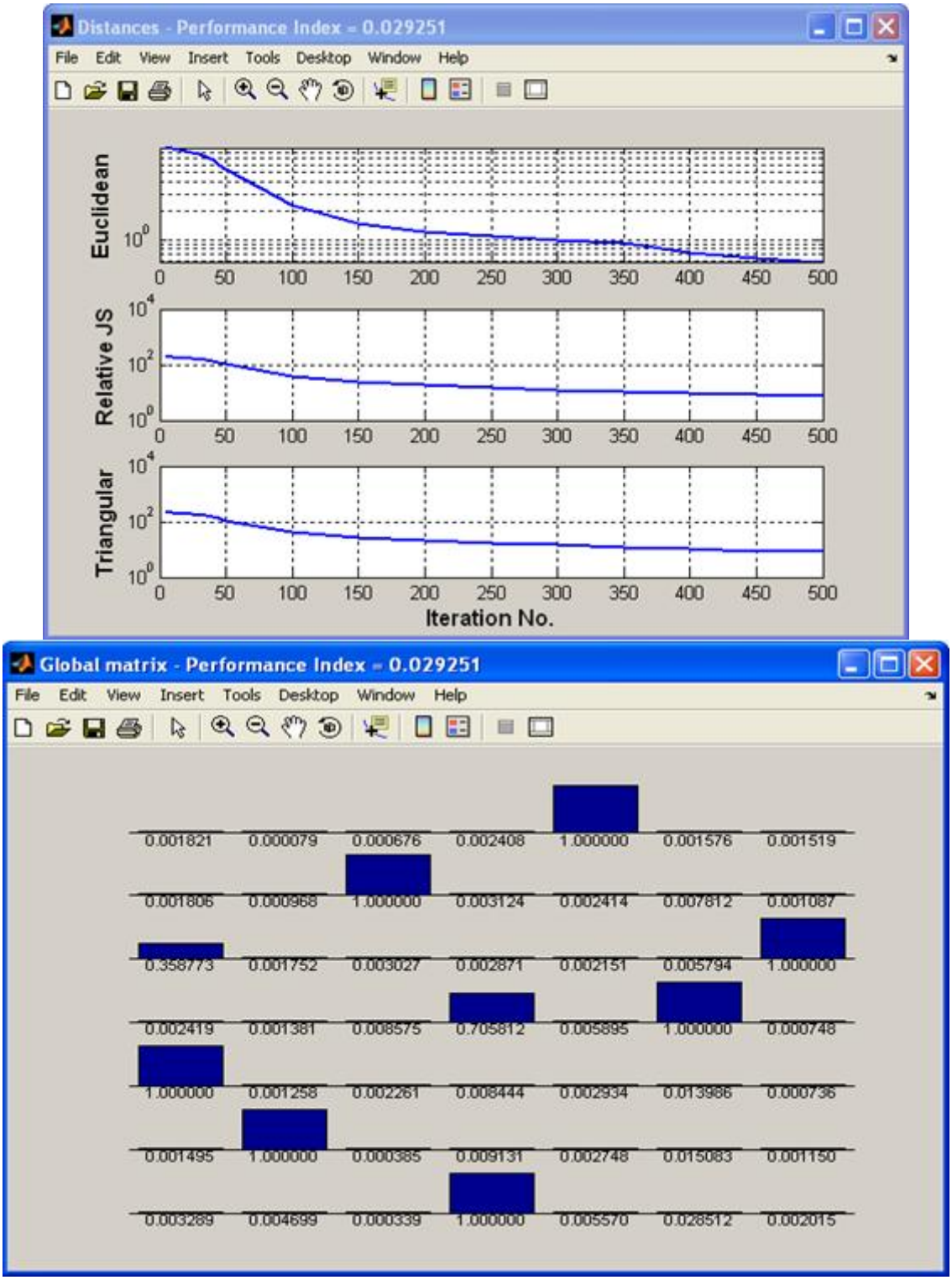


Fig. 18: Windows for visualizing various loss function versus alternating steps; and also the global matrix related to the Amari index.

The user can also choose to see a 3D plot of the global matrix $\mathbf{G} = \mathbf{W}\mathbf{A}_{est}$, where $\mathbf{W}$=pinv($\mathbf{A}_{est}$).

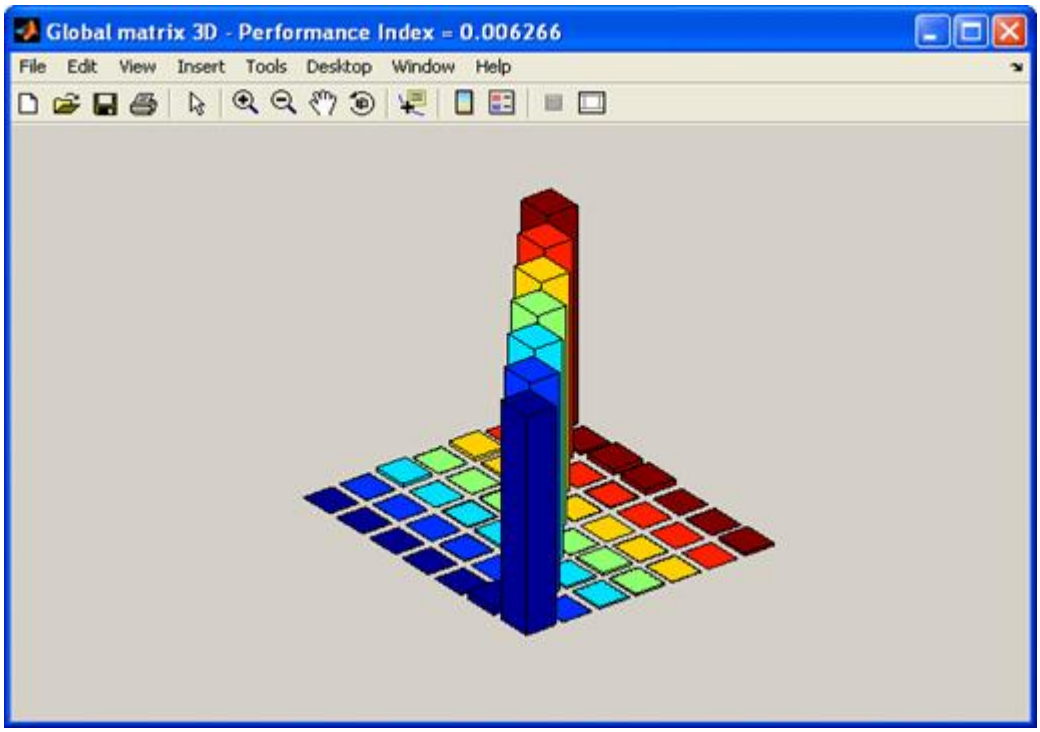


Fig. 19: An exemplary of a 3-D plot of the global performance matrix ($G = WA_{est}$).

Note: Rendering of 3D plots from a large number of signals can be very time-consuming.

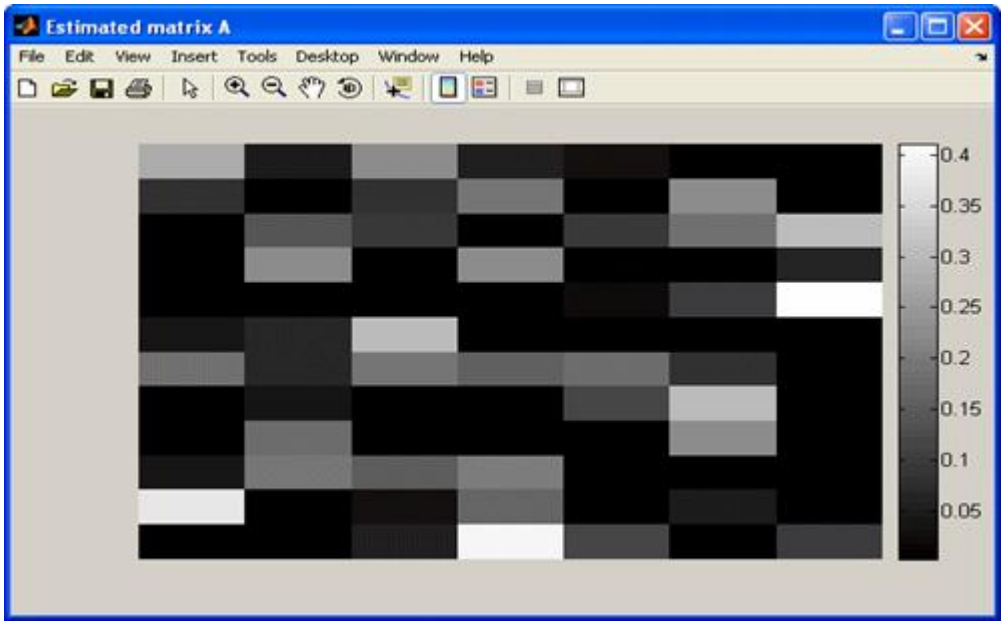


Fig. 20: An exemplary plot of the estimated matrix A_{est} = inv(W).

The window *Estimated mixing matrix* $\mathbf{A_{est}}$**=** inv(**W**) shows the distribution of the entries of the estimated mixing matrix.

The user can specify which plots are to be drawn. At the bottom of the main NMFLAB window, clicking `View` will allow the user to choose from a multitude of different plots. Instead of closing each of the windows separately, the user can click the `CLOSE` button to close all the plots. The plot windows are closed automatically if you re-run any algorithm. All the signals (sources, mixtures, independent components) are displayed in a separate window.

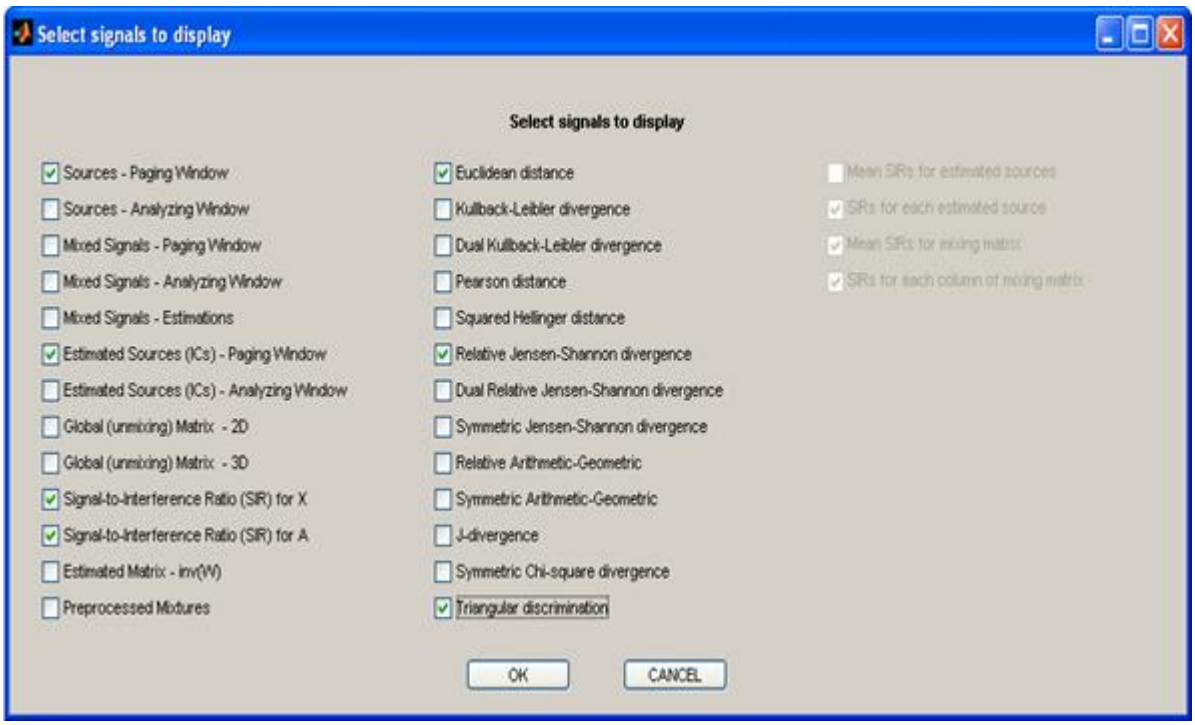


Fig. 21: Window illustrating how to choose various plots for visualizing the obtained results.

Saving the results

The NMFLAB package generally assumes the following generative mixing model

$$\mathbf{y}(k) = \mathbf{A}\,\mathbf{x}(k) + \mathbf{v}(k), \qquad k = 1, 2, ..., T$$

where $\mathbf{v}(k)$ is a vector of additive noise,

or in batch mode

$$\mathbf{Y} = \mathbf{A}\,\mathbf{X} + \mathbf{V}$$

where:

$$\mathbf{Y} = [\,\mathbf{y}(1), \mathbf{y}(2), ..., \mathbf{y}(N)\,],$$
$$\mathbf{X} = [\,\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(N)\,],$$
$$\mathbf{V} = [\,\mathbf{v}(1), \mathbf{v}(2), ..., \mathbf{v}(N)\,];$$

Table 1. Basic variables used in NMFLAB

Variable	Description	Dimension of matrix
X	Original sources or components	$r \times T$
Y	Observations (sensor signals)	$m \times T$ (usually $m >> r$)
XH(=X_{est})	Estimated sources or independent components	$r \times T$
AH(=A_{est})	Estimated mixing matrix	$m \times r$ or $m \times m$
W	Demixing matrix (if it exists, $m >= r$)	$m \times r$ or $r \times r$
G	Global (mixing-demixing) matrix	$m \times r$ or $r \times r$

Remark: Most algorithms automatically detect the number of sources and setting them to *r* using SVD.

The results can be saved for further processing by clicking on the `SAVE` button. Note that all the signals will be saved into a single *.mat (MATLAB) file (with the specified name) i.e., **X** - original, primary sources, **Y** - sensors (mixture), **AH** - estimated components, **XH** - estimated sources, **A** - mixing matrix, **W** - demixing matrix, and **G** - global (mixing-demixing) matrix. The data can also be saved in the following forms: *.txt (ASCII), or *.xls (Excel). The data can only be saved in the Excel format `*.xls` if Microsoft Excel program is installed on your computer. On `UNIX` operating systems the data can be saved in `csv` format. Saving in this format can take a long time.

Monte Carlo Analysis

The algorithms included in this version of NMFLAB do not guarantee convergence to the global minimum of the objective function, even though some of them reach stationary points quite quickly. Therefore, random initializations of each algorithm may give different solutions, since the algorithm may converge to different local minima (if it is initialized differently). Monte Carlo (MC) analysis is applied to evaluate the performance of the algorithm objectively. The algorithm is run several times with different initializations for each run. A more accurate picture of the performance can be obtained from the statistics of the SIR samples, both for the mixing matrix and the estimated sources. To run the MC analysis, mark the checkbox in the field *Monte Carlo Analysis* (as shown below).

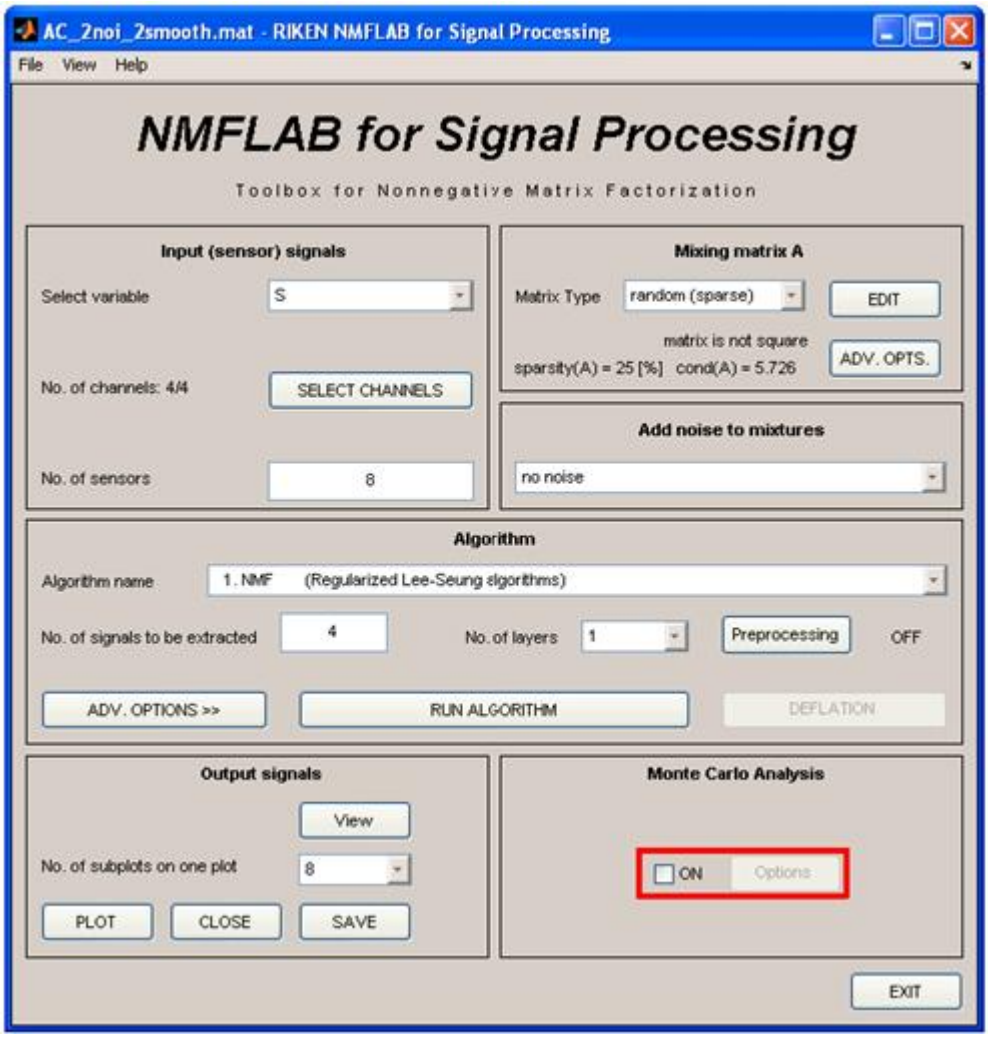


Fig. 22: To enable Monte Carlo analysis, activate the check box as show in the figure above.

Clicking the **`Options`** button opens a window that allows the user to set the parameters for MC analysis. The **`Setup`** field allows the user to set the number of times the algorithm is run i.e. using **`Runs.`** The default is 100 runs. This number should be large yet the larger the number the longer the computation time. Monte Carlo analysis provides credible results, provided that a chain of samples has ergodic properties. To check the ergodicity: run two independent MC analysis experiments, and then to compare the means of the samples. The means of sufficient long sub-chains of an ergodic chain should be the same. If the means of the samples from independent MC experiments are consistently close, this would be an indicator of ergodicity.

For a single simulation the algorithms terminate if they satisfy the following stop criterion: if the changes in the entries of the estimated mixing matrix between two subsequent alternating steps are sufficiently small, or if the total number of alternating steps exceeds 2000. Selecting **`Dynamically Adjusted`** allows the use of the default stop criterion. Selecting **`Fixed`** in the **`Alternatings`** field allows the user to force the algorithm to run a fixed number of steps in each run.

When a single simulation is run, it restarts several times with different initializations and the factors that generate the lowest cost are chosen as the estimates for the sources and mixing matrix after a fixed number of alternating steps. This is redundant in the case of Monte Carlo analysis and can be stopped by un-ticking the checkbox **Restarts**.

The fields **Histograms** and **SIRs for the selected samples** enable the user to visualize the statistics of the samples. Histograms can be plotted for SIR samples of the mixing matrix and/or the estimated sources. The means and variances of the histograms are also displayed at the top of the plots. The histograms of SIR samples for each column of the mixing matrix or for each estimated signal can also be seen.

Alternatively, the histograms of the means of the SIRs over all the columns of the mixing matrix or over all the estimated sources can be plotted if the checkboxes **Mean-Values** in the fields **Mixing Matrix** or **Estimated Sources** respectively, are checked.

There are four possible ways to view the estimated SIRs for the selected sample. For the mixing matrix and the sources both the **Worst case** and **Best case** SIRs for the selected sample can be viewed. For example choosing **Mixing Matrix** and **Worst Case** the sample with the worst SIR of the mixing matrix is used, alternatively the option Best Case displays the runs with the best SIR.

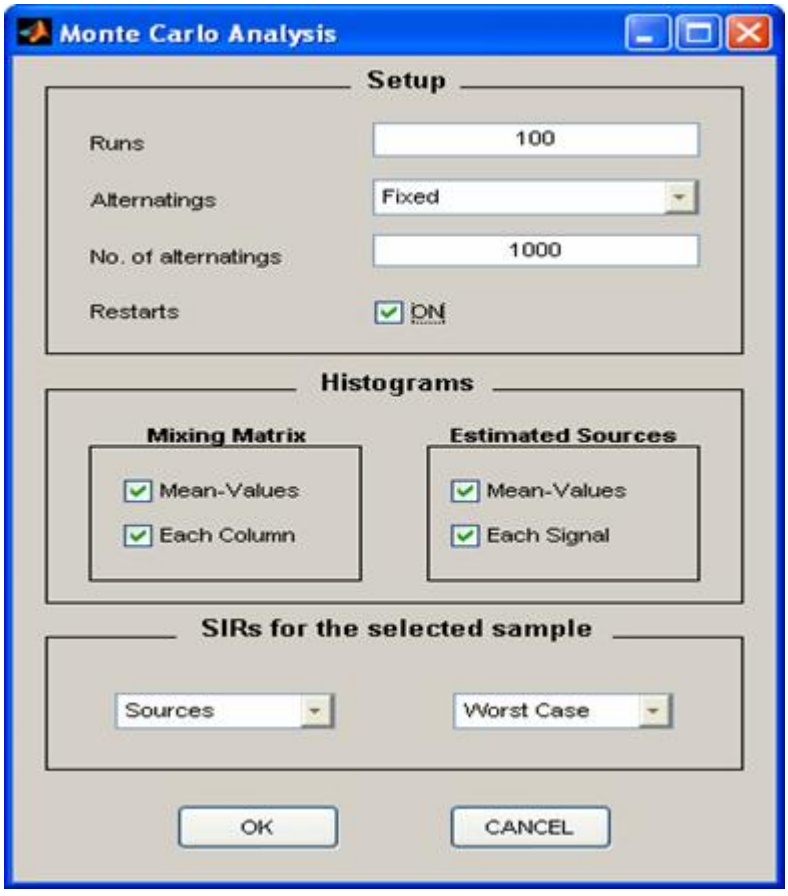
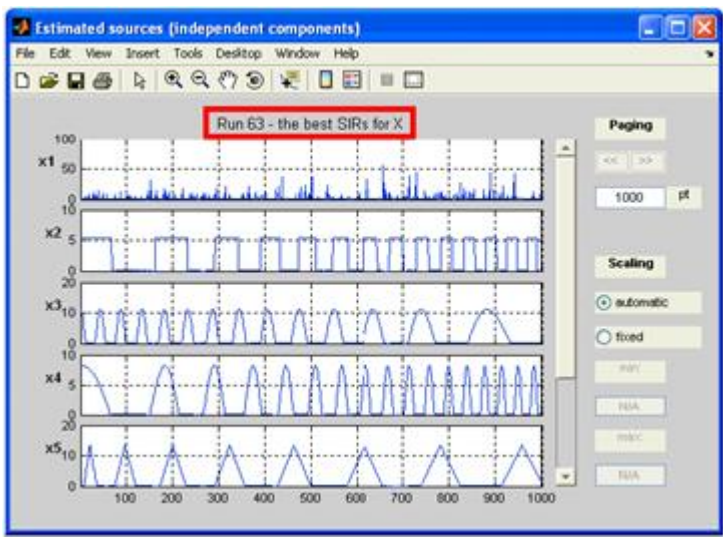
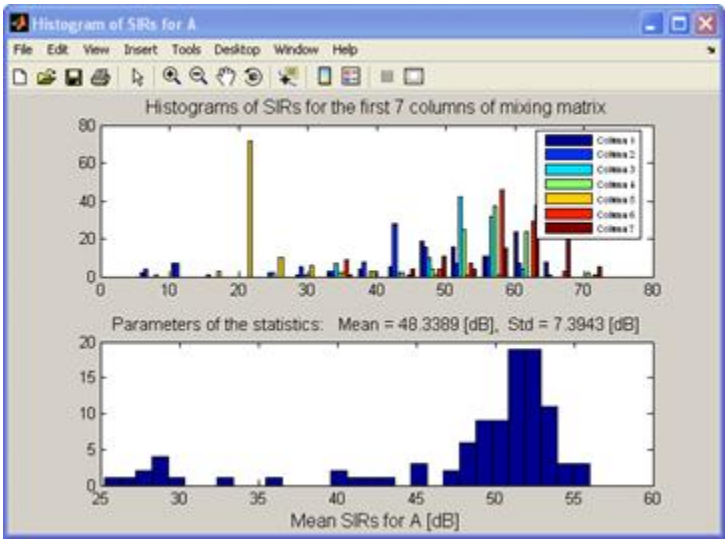


Fig. 23: Parameters that can be set for the Monte Carlo analysis.

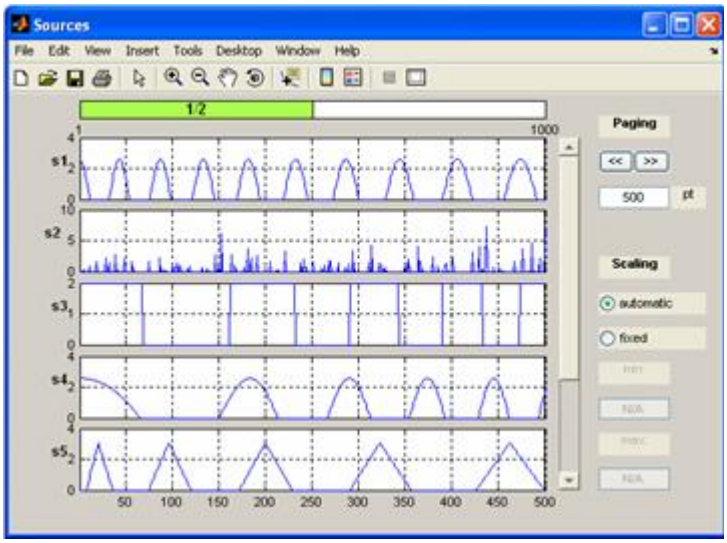
When the MC analysis is finished, pressing **PLOT** will display the results. Exemplary MC results are shown in Fig. 24.



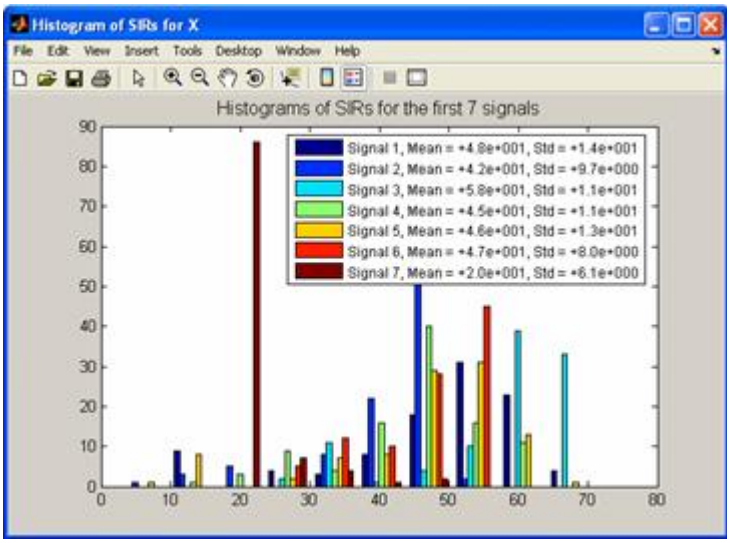
(a)



(b)



(c)



(d)

Fig. 24: The four windows above illustrate the output from the MC analysis: (a) Plot of the estimated sources in the selected sample (No. of the run is shown on the top), (b) Plots of the SIRs for the estimated mixing matrix – the parameters of the statistics are also displayed. (c) Plot of the primary sources (if available). If the sources are not available this window displays the sensor signals. (d) Plot of the SIRs for the estimated sources.

Deflation

When the independent components have been extracted or blind separation of signals has been performed (from the mixture), the effects of discarding some of components by reconstructing the sensor signals from the remaining components can be explored. This procedure is called deflation or reconstruction, and allows the user to remove unnecessary (or undesirable) components that are hidden in the mixture (superimposed or overlapped data). In other words, the deflation procedure allows the user to extract and remove one or more independent component (or uncorrelated sources) from the mixture $\mathbf{x}(k)$. This can be achieved by clicking the `DEFLATION` button in the main **NMFLAB** window which opens the *Reconstruction procedure* window. The `DEFLATION` button becomes activated after the estimation of the demixing matrix $\mathbf{W}$ is completed using any of the built-in or user-defined algorithms.

The deflation procedure is carried out in two steps.

1. In the first step, the selected algorithm estimates the mixing matrix $\mathbf{A}_{\mathbf{est}}$ and then performs the decomposition of observations into independent components: $\mathbf{y}(k) = \mathbf{A}_{\mathbf{est}}\, \mathbf{x}(k)$.
2. In the second step, the deflation algorithm eliminates one or more of the components from the vector $\mathbf{y}(k)$ and then performs the back-propagation $\mathbf{x}_{\mathbf{r}} = \mathbf{W}^{+}\, \mathbf{y}_{\mathbf{r}}(k)$, where $\mathbf{x}_{\mathbf{r}}$ is a vector of reconstructed sensor signals, $\mathbf{W}^{+} = \mathbf{A}_{\mathbf{est}}$ is a generalized pseudo inverse matrix of the estimated demixing matrix $\mathbf{W}$, and $\mathbf{x}_{\mathbf{r}}(k)$ is the vector obtained from the vector $\mathbf{x}(k)$ after removal of all the undesirable components (i.e., by replacing them with zeros). In the special case, where the number of sources is equal to the number of sensors and the number of outputs, we can use the inverse matrix $\mathbf{W}^{-1}$ instead of $\mathbf{W}^{+}$.

   In batch format, the reconstruction procedure can be written as

$$\mathbf{Y}_{\mathbf{r}} = \mathbf{A}_{\mathbf{est}}\mathbf{X}_{\mathbf{r}} = \mathbf{W}^{+}\, \mathbf{X}_{\mathbf{r}}$$

   where

   $\mathbf{Y}_{\mathbf{r}} = [\, \mathbf{y}_{\mathbf{r}}(1), \mathbf{y}_{\mathbf{r}}(2), ..., \mathbf{y}_{\mathbf{r}}(N)\,]$ - reconstructed sensor signals,

   $\mathbf{X}_{\mathbf{r}} = [\, \mathbf{x}_{\mathbf{r}}(1), \mathbf{x}_{\mathbf{r}}(2), ..., \mathbf{x}_{\mathbf{r}}(N)\,]$ - reduced (selected or filtered) independent components.

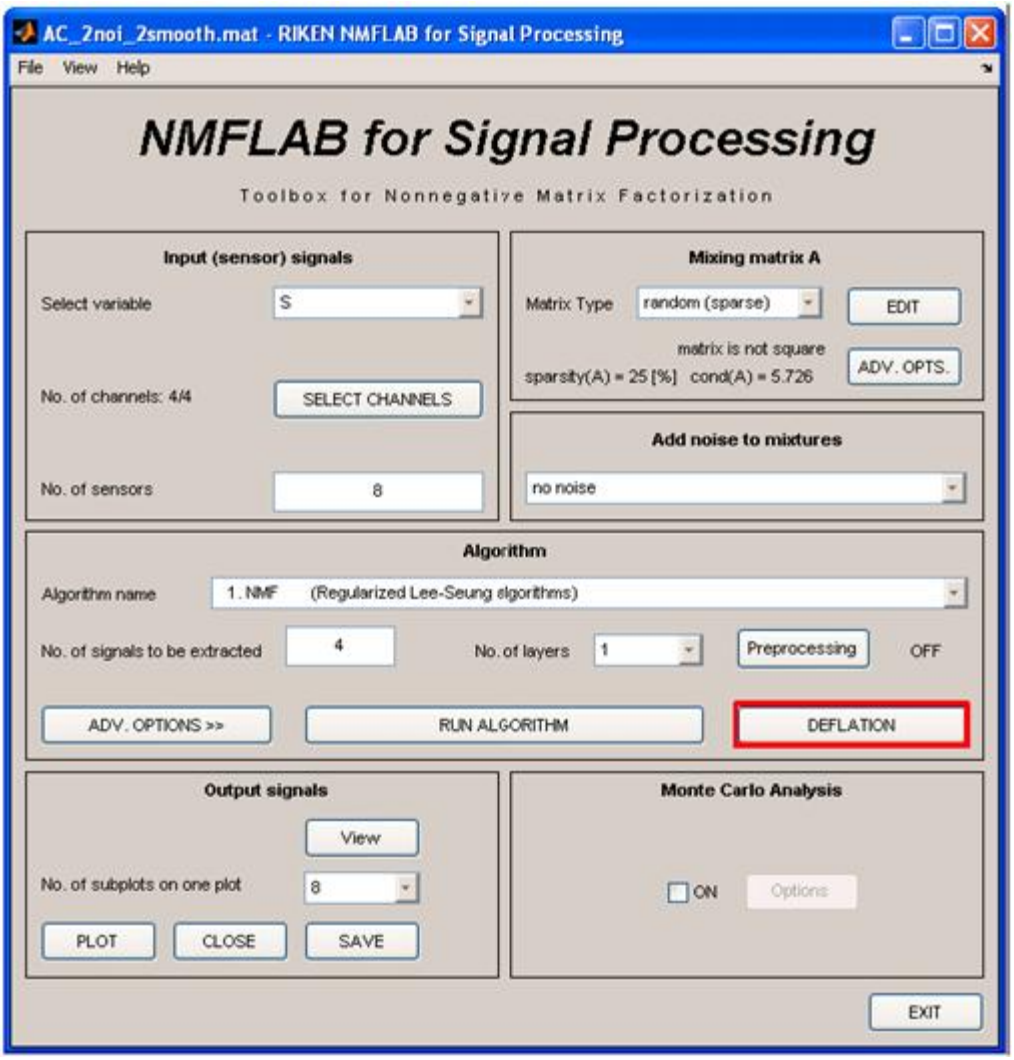


Fig. 25: This window illustrates the selection of the deflation procedure (post-processing) from the main window. Click the DEFLATION button to reconstruct "clean" data or remove some components.

In the deflation procedure, undesirable components, such as noise, artifacts or undesirable interference, can be eliminated by un-ticking the check box at the left of each component (signal) as seen in figure 26. Deflation is performed by pressing the `Do deflation` button. Simultaneously, a second window appears, similar to the subwindow in Fig. 15, which allows the detailed analysis in time and frequency domains of each reconstructed sensor signal (Figs. 15, 16).

The deflation (reconstruction) procedure is illustrated in Fig. 26. In this figure, almost all of the components x_i are reset to zero except for the components x_2 and x_7. These are projected back to sensor level (as $\mathbf{y_r}(k)$ = pinv($\mathbf{W}$) $\mathbf{x_r}$ (k) = $\mathbf{W}^{+}\mathbf{x_r}(k)$, where $\mathbf{W}^{+}$ (10x10) means pseudo-inverse of demixing matrix $\mathbf{W}$ and $\mathbf{x_r}(k)$ = [0, $x_2(k)$, 0, 0, 0, 0, $x_7(k)$, 0, 0, 0]^T). The results are shown on right-side plot in Fig. 27.

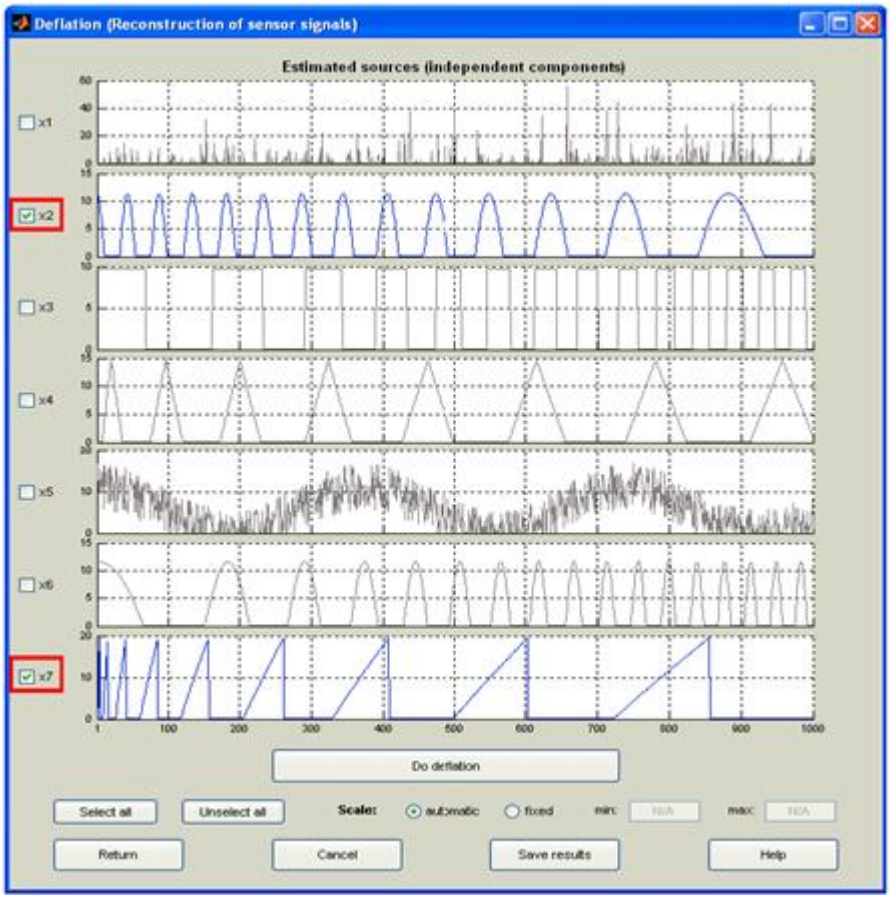
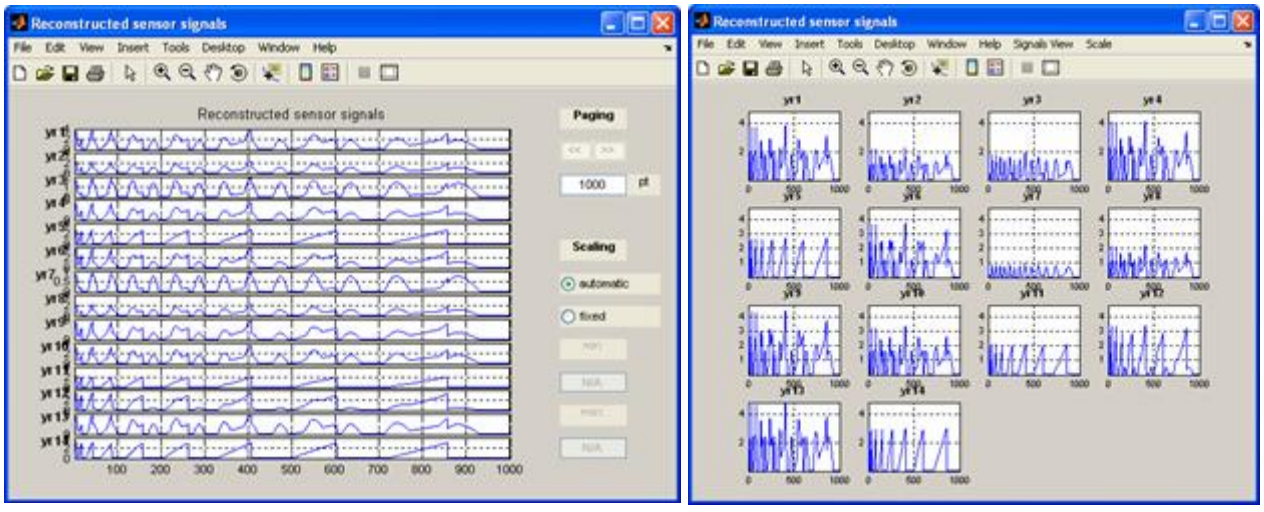


Fig. 26: This window illustrates the deflation procedure. The plot shows the estimated components or separated signals before deflation. In this example, only signals *x2* and *x7* are back-projected, while the other components are neglected.



(a) **(b)**

Fig. 27: This window illustrates the plots after deflation has been done. The left plot (a) is a paging window, and the right plot (b) is an analyzing window. Both windows show the reconstructed sensor signals after eliminating the signals x1, x3, x4, x5, x6.

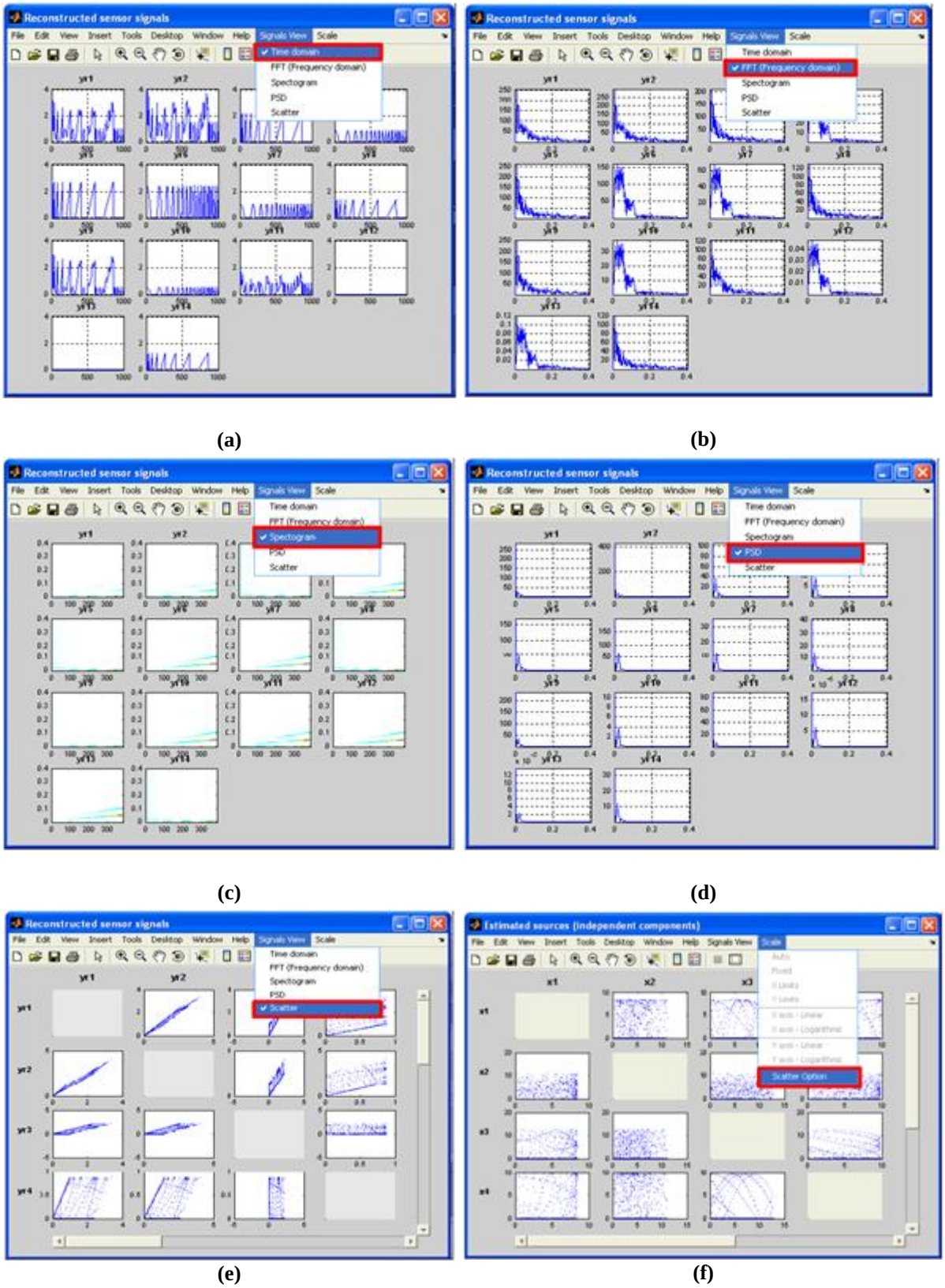


Fig. 28: This window illustrates different representations of signals

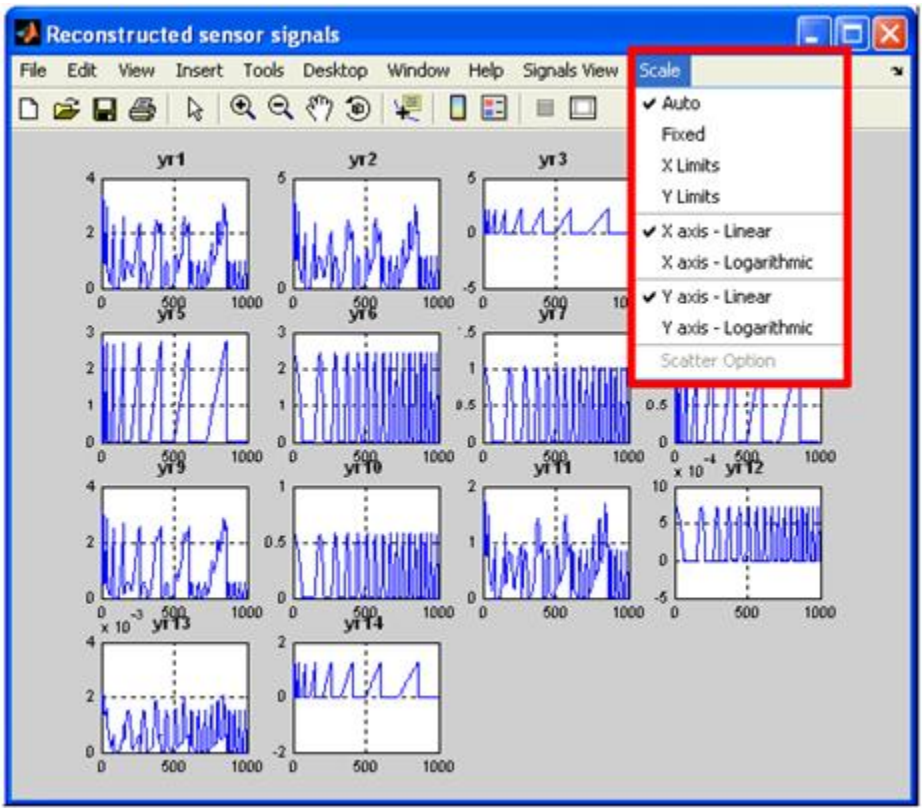


Fig. 29: This figure shows sub-windows for the detailed analysis in time and frequency domains of each of the reconstructed sensor signal. Such signal analysis can be also performed for source and observed signals.

When the deflation procedure has been finished, the reconstructed sensor signals can be saved for further processing by clicking on the `Save results` button. This allows the user to process the reconstructed sensor signals by applying the same or different NMF/ICA algorithms. For example, in the first stage you can apply a second-order statistics BSS algorithm to recover sources with temporal structures. In the second stage, you may use any higher-order statistics ICA algorithm to recover independent components.

In the same way, the reconstructed data can be saved using the deflation procedure. The reconstructed data can be saved as follows

- `name_XR.mat`
- `name_XR.txt`
- `name_XR.xls`
- `name_XR.csv`

The `CANCEL` button allows the user to leave this window (i.e. deflation procedure) and go back to the main program without saving or making any changes. Pressing the `RETURN` button, enables the user to save the reconstructed signals $\mathbf{X_r}$ as new sensor signals $\mathbf{X}$ which can be further processed by any built-in algorithm. This allows facilitates further processing and extraction of the signals as part of a multistage procedure (multilayer system).

Exit from NMFLAB

It is recommended NMFLAB is closed by clicking the `EXIT` button in the main program window as shown below:

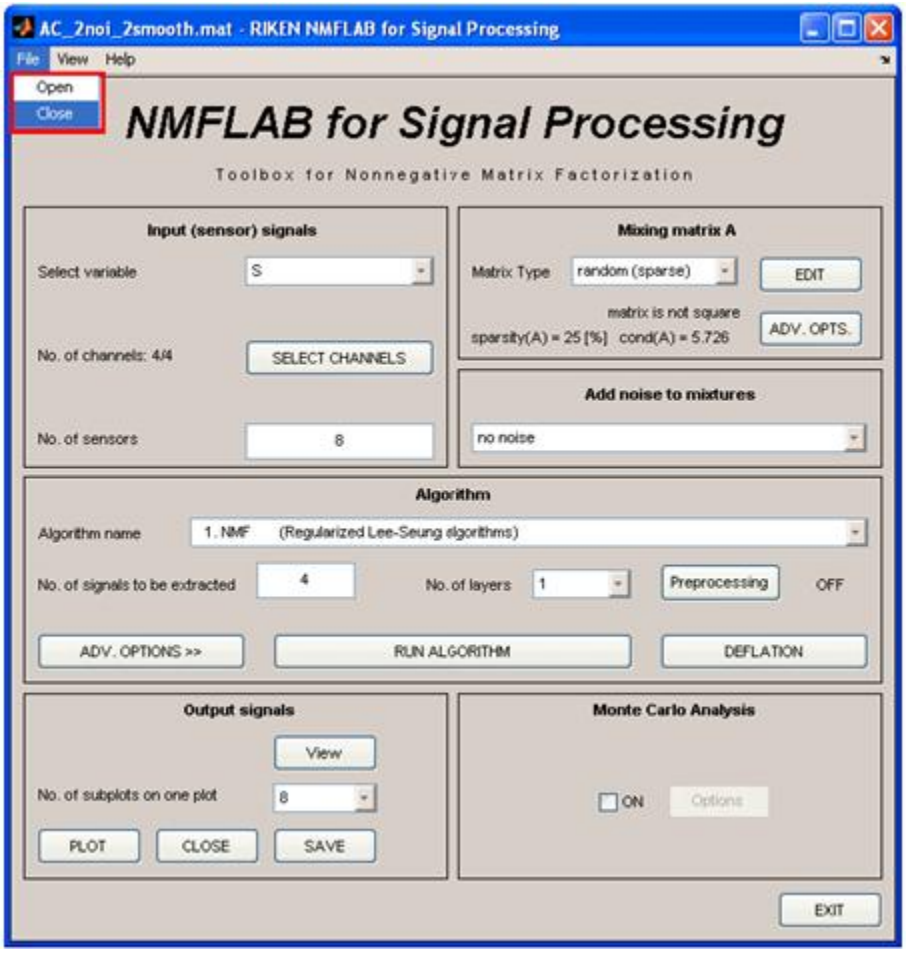


Fig. 30: Window illustrating how to exit from the program. Click the EXIT button to leave the program or choose Close in the menu bar.

PART 2
Algorithms for Non-Negative Matrix Factorization

Non-negative matrix factorization (NMF) decomposes the data matrix $\boldsymbol{Y} = [\boldsymbol{y}(1), \boldsymbol{y}(2), \ldots, \boldsymbol{y}(T)] \in \mathbb{R}^{m \times T}$ as a product of two matrices: the basis or mixing matrix $\boldsymbol{A} \in \mathbb{R}^{m \times r}$ and the source component matrix $\boldsymbol{X} = [\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(T)] \in \mathbb{R}^{r \times T}$, where both matrices have only non-negative elements. Although some decompositions or matrix factorizations provide an exact reconstruction of the data (i.e., $\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}$), we shall consider here decompositions which are approximative in nature, i.e.,

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{V}, \qquad \boldsymbol{A} \geq 0, \;\; \boldsymbol{X} \geq 0 \quad \text{(component-wise)} \tag{1}$$

or equivalently $\boldsymbol{y}(k) = \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{v}(k), \;\; k = 1, 2, \ldots, T$, or in a scalar form as $y_i(k) = \sum_{j=1}^{r} a_{ij} x_j(k) + \nu_i(k), \;\; i = 1, \ldots, m,$ where $\boldsymbol{V} \in \mathbb{R}^{m \times T}$ represents a noise or error matrix, $\boldsymbol{y}(k) = [y_1(k), \ldots, y_m(k)]^T$ is a vector of the observed components or signals for $k - th$ sample or at the discrete time instants k, and $\boldsymbol{x}(k) = [x_1(k), \ldots, x_r(k)]^T$ is a vector of unknown components or source signals for the same sample. For further information see, e.g.,

A. Cichocki and S. Amari. *Adaptive Blind Signal And Image Processing (New revised and improved edition).* John Wiley, New York, 2003.

We assume generally that $r \leq m < T$, and use the following notation: $x_{jk} = x_j(k)$, $y_{ik} = y_i(k)$, and $v_{ik} = v_i(k)$. The number of the true components r can be easily estimated automatically from the noise-free and well-posed data using SVD or PCA [1].

NMF algorithms use alternating minimization of a cost (or loss) function $D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})$ or set of loss functions subject to nonnegativity constraints ($a_{ij} \geq 0$, $x_{jk} \geq 0$) and in some cases, possibly other constraints to enforce sparsity and/or smoothness. Since any suitably designed cost function has two sets of parameters ($\boldsymbol{A}$ and $\boldsymbol{X}$), we usually employ constrained alternating minimization, i.e., in one step $\boldsymbol{A}$ is estimated and $\boldsymbol{X}$ fixed, and in the other step we fix $\boldsymbol{A}$ and estimate $\boldsymbol{X}$. Typically, the estimate of $\boldsymbol{A}$ depends on both estimates $\boldsymbol{A}$ and $\boldsymbol{X}$. Thus the iteration which updates $\boldsymbol{A}$ may be performed several times before proceeding to the $\boldsymbol{X}$ update. A similar choice is available when performing the $\boldsymbol{X}$ update. Thus switching between computation of $\boldsymbol{A}$ and $\boldsymbol{X}$ can be performed after each single update, or we can use so-called *inner iterations* for performing the 'one-step' estimation [2].

[1] In NMFLAB, we apply the SVD to estimate r automatically; however, the user can set the parameter in the field `No. of signals to be extracted`.

[2] In NMFLAB, the default setting for the number of inner iterations is one, thus switching proceeds every single iteration. Some algorithms demonstrate better con-

1 Regularized Lee-Seung Algorithms (Multiplicative Updates)

The Regularized Lee-Seung algorithms group contains the following algorithms: EMML, ISRA, Kompass algorithm, and projected pseudo-inverse. In their original incarnation, the EMML and ISRA algorithms assumed that the basis (mixing) matrix $\boldsymbol{A}$ was known and only the nonnegative matrix $\boldsymbol{X}$ was unknown. However, EMML and ISRA can be applied assuming the opposite is true. The EMML and ISRA algorithms differ in how the updates are performed; EMML minimizes the Kullback-Leibler cost function, and ISRA minimizes a squared Euclidean cost function (in our case squared Frobenius norm). Both algorithms perform alternating minimization of the specific cost function using gradient descent, i.e., they use alternating switching between set of parameters to generate the updates till convergence is achieved.

The Lee-Seung algorithms, which are derivatives of EMML and ISRA are considered to be the standard NMF algorithms and we will introduce them first. One contribution in Lee-Seung,

D. D. Lee and H. S. Seung. Learning of the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999,

was to update both $\boldsymbol{A}$ and $\boldsymbol{X}$ by applying the well-known technique of switching between updating $\boldsymbol{A}$ with $\boldsymbol{X}$ known and updating $\boldsymbol{X}$ with $\boldsymbol{A}$ known. Additionally, Lee-Seung derived a simple multiplicative update instead of the standard additive update and suggested that the multiplicative update was 'a good compromise between speed and ease of implementation.' These algorithms belong to a class of multiplicative algorithms and use the alternating technique for minimization of the cost function.

Lee-Seung consider two different cost (loss) functions: the squared Euclidean distance (squared Frobenius norm), used in ISRA:

$$D_F(\boldsymbol{Y}||\boldsymbol{AX}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{AX}\|_F^2, \tag{2}$$

and generalized Kullback-Leibler (KL) divergence, which is used in EMML:

$$D_{KL}(\boldsymbol{Y}||\boldsymbol{AX}) = \sum_{ik}\left(y_{ik}\log\frac{y_{ik}}{[\boldsymbol{AX}]_{ik}} + [\boldsymbol{AX}]_{ik} - y_{ik}\right), \tag{3}$$

$$\text{s. t.} \quad \forall i,j,k: \; x_{jk} \geq 0, \quad a_{ij} \geq 0, \;\; \|\boldsymbol{a}_j\|_1 = \sum_{i=1}^{m} a_{ij} = 1.$$

vergence if the number of inner iterations is in the range 5–10 (e.g., the GPCG algorithm). Also, in NMFLAB, we can use the same or different cost (loss) functions for estimating $\boldsymbol{A}$ and $\boldsymbol{X}$. We can apply different kinds of algorithms for estimating $\boldsymbol{A}$ and $\boldsymbol{X}$. For example, we could choose the EMML algorithm to update estimation of $\boldsymbol{A}$ while using the ISRA algorithm for the update of $\boldsymbol{X}$. Other combinations of algorithms are possible for algorithms which belong to the same group, e.g., the Regularized Lee-Seung Algorithms.

Applying the standard gradient descent technique to the cost function (2), we have the following algorithm:

$$a_{ij} \leftarrow a_{ij} \frac{[\boldsymbol{Y}\,\boldsymbol{X}^T]_{ij}}{[\boldsymbol{A}\,\boldsymbol{X}\,\boldsymbol{X}^T]_{ij}}, \tag{4}$$

$$x_{jk} \leftarrow x_{jk} \frac{[\boldsymbol{A}^T\,\boldsymbol{Y}]_{jk}}{[\boldsymbol{A}^T\,\boldsymbol{A}\,\boldsymbol{X}]_{jk}}, \tag{5}$$

which is the Lee-Seung algorithm for the squared Euclidean cost function and is an alternating version of the well-known ISRA (Image Space Reconstruction Algorithm). The original ISRA algorithm has the same multiplicative form but involved updating only $\boldsymbol{X}$ iteratively with $\boldsymbol{A}$ assumed to be known. See e.g.,

C. L. Byrne. Accelerating the EMML algorithm and related iterative algorithms by rescaled block-iterative (RBI) methods. *IEEE Transactions on Image Processing*, IP-7:100–109, 1998 .

In matrix notation, the multiplicative updates become:

$$\boldsymbol{A} \leftarrow \boldsymbol{A} \odot \boldsymbol{Y}\boldsymbol{X}^T \oslash \boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^T, \tag{6}$$

$$\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \boldsymbol{A}^T\boldsymbol{Y} \oslash \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}, \tag{7}$$

where $\odot$ and $\oslash$ denote component-wise multiplication and division, respectively[3].

Alternatively, the minimization of the cost function (3) leads to

$$a_{ij} \leftarrow a_{ij} \frac{\sum_{k=1}^{T} x_{jk}\,(y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})}{\sum_{p=1}^{T} x_{jp}}, \tag{8}$$

$$x_{jk} \leftarrow x_{jk} \frac{\sum_{i=1}^{m} a_{ij}\,(y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})}{\sum_{q=1}^{m} a_{qj}}, \tag{9}$$

which is the Lee-Seung algorithm for the generalized K-L divergence. This is an alternating version of the well-known EMML or Richardson-Lucy algorithm (RLA). Similarly, as in the case of ISRA algorithm, EMML involved updating $\boldsymbol{X}$ iteratively as $\boldsymbol{A}$ was assumed to be known and fixed. See e.g.,

C. L. Byrne. Accelerating the EMML algorithm and related iterative algorithms

[3] In order to avoid multiplication and division by zero in all practical implementations of NMF multiplicative algorithms, we added to numerator and denominator small additive positive constant ε, typically $\varepsilon = 10^{-16}$.

by rescaled block-iterative (RBI) methods. *IEEE Transactions on Image Processing*, IP-7:100–109, 1998.

In matrix notation, we have

$$\boldsymbol{A} \leftarrow \boldsymbol{A} \odot (\boldsymbol{Y} \oslash \boldsymbol{A}\boldsymbol{X})\, \boldsymbol{X}^T \oslash [\boldsymbol{X}\mathbf{1}_T, \ldots, \boldsymbol{X}\mathbf{1}_T]^T, \tag{10}$$

$$\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \boldsymbol{A}^T (\boldsymbol{Y} \oslash \boldsymbol{A}\boldsymbol{X}) \oslash [\boldsymbol{A}^T\mathbf{1}_m, \ldots, \boldsymbol{A}^T\mathbf{1}_m], \tag{11}$$

where $\mathbf{1}_T = [1, \ldots, 1]^T \in \mathbb{R}^T$, $\mathbf{1}_m = [1, \ldots, 1]^T \in \mathbb{R}^m$, $[\boldsymbol{X}\mathbf{1}_T, \ldots, \boldsymbol{X}\mathbf{1}_T] \in \mathbb{R}^{r \times m}$, and $[\boldsymbol{A}^T\mathbf{1}_m, \ldots, \boldsymbol{A}^T\mathbf{1}_m] \in \mathbb{R}^{r \times T}$.

1.1 Regularization of the Lee-Seung Algorithms

We proposed regularized versions of the above Lee-Seung algorithms in

A. Cichocki, R. Zdunek, and S. Amari. New algorithms for non-negative matrix factorization in applications to blind source separation. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP2006*, 2006.

Using the squared Euclidean and K-L cost functions we considered the following constrained alternating minimization problems:

$$D_F^{\alpha_A,\alpha_X}(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2 + \alpha_A J_A(\boldsymbol{A}) + \alpha_X J_X(\boldsymbol{X}), \tag{12}$$

$$\begin{aligned} D_{KL}^{\alpha_A,\alpha_X}(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = &\sum_{ik}\left(y_{ik}\log\frac{y_{ik}}{[\boldsymbol{A}\boldsymbol{X}]_{ik}} + [\boldsymbol{A}\boldsymbol{X}]_{ik} - y_{ik}\right) \\ &+ \alpha_A J_A(\boldsymbol{A}) + \alpha_X J_X(\boldsymbol{X}), \\ &\text{s. t.} \quad \forall i,j,k: \; x_{jk} \geq 0, \quad a_{ij} \geq 0, \end{aligned} \tag{13}$$

where $\alpha_A \geq 0$ and $\alpha_X \geq 0$ are regularization parameters, and functions $J_X(\boldsymbol{X})$ and $J_A(\boldsymbol{A})$ are used to enforce certain application-dependent characteristics of the solution.

Regularization of ISRA and Lee-Seung algorithm for Euclidean cost function: The ISRA or Lee-Seung algorithm for Euclidean cost function can be regularized using additive penalty terms. Using equation (12) we developed the following generalized learning rules:

$$a_{ij} \leftarrow a_{ij}\, \frac{\Big[[\boldsymbol{Y}\,\boldsymbol{X}^T]_{ij} - \alpha_A[\nabla_A J_A(\boldsymbol{A})]_{ij}\Big]_\varepsilon}{[\boldsymbol{A}\,\boldsymbol{X}\,\boldsymbol{X}^T]_{ij}}, \tag{14}$$

$$x_{jk} \leftarrow x_{jk}\, \frac{\Big[[\boldsymbol{A}^T\,\boldsymbol{Y}]_{jk} - \alpha_X[\nabla_X J_X(\boldsymbol{X})]_{jk}\Big]_\varepsilon}{[\boldsymbol{A}^T\boldsymbol{A}\,\boldsymbol{X}]_{jk}}, \tag{15}$$

$$a_{ij} = \frac{a_{ij}}{\sum_{j=1}^{r} a_{ij}}, \tag{16}$$

where the nonlinear operator is defined as

$$[x]_{\varepsilon} = \max\{\varepsilon, x\} \tag{17}$$

with small ε (*eps*) Typically, $\varepsilon = 10^{-16}$.

Regularization functions: [4] The regularization terms $J_A(\boldsymbol{A})$ and $J_X(\boldsymbol{X})$ can be defined in many ways. Let us assume the following definition for L_p-norm of a given matrix $\boldsymbol{C} = [c_{mn}] \in \mathbb{R}^{M \times N}$: $L_p(\boldsymbol{C}) \triangleq \left(\sum_{m=1}^{M} \sum_{n=1}^{N} |c_{mn}|^p\right)^{\frac{1}{p}}$, thus:

- L_1 **norm** ($p = 1$) – *for sparse solution:*

  $J_A(\boldsymbol{A}) = \sum_{i=1}^{m} \sum_{j=1}^{r} a_{ij}, \quad J_X(\boldsymbol{X}) = \sum_{j=1}^{r} \sum_{k=1}^{T} x_{jk},$

  $\nabla_A J_A(\boldsymbol{A}) = 1, \quad \nabla_X J_X(\boldsymbol{X}) = 1,$
- **Squared** L_2 **norm** ($p = 2$) – *for smooth solution:*

  $J_A(\boldsymbol{A}) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{r} a_{ij}^2, \quad J_X(\boldsymbol{X}) = \frac{1}{2} \sum_{j=1}^{r} \sum_{k=1}^{T} x_{jk}^2,$

  $\nabla_A J_A(\boldsymbol{A}) = a_{ij}, \quad \nabla_X J_X(\boldsymbol{X}) = x_{jk},$
- **Gibbs smoothing:**

  Gibbs smoothing is related to the Markov Random Field (MRF) model that is often used in image reconstruction to enforce local smoothing. This model is formulated with the Gibbs prior

  $$p(x) = \frac{1}{Z} \exp\{-\alpha U(x)\}, \tag{18}$$

  where Z is a partition function, α is a regularization parameter, and $U(x)$ is a total energy function that measures the total roughness in the object of interest[5]. Thus

  $$J_A(\boldsymbol{A}) = 0, \; J_X(\boldsymbol{X}) = U(\boldsymbol{X}), \tag{19}$$

  $$U(\boldsymbol{X}) = \sum_{j=1}^{r} \sum_{k=1}^{T} \sum_{l \in S_k} w_{kl} \psi\left(x_{jk} - x_{jl}, \delta\right), \tag{20}$$

[4] In NMFLAB (see `Advanced options`), we implemented the L_1 norm, the squared L_2 norm and Gibbs smoothing for the regularization function $J_X(\boldsymbol{X})$ as it has been found after extensive testing that they work well. These have not been implemented for $J_A(\boldsymbol{A})$ yet.

[5] In NMFLAB, we only apply the Gibbs smoothing to the estimated sources and not the mixing matrix.

where S_k is the nearest neighborhood of sample k, w_{kl} is a weighting factor, δ is a scaling factor, and $\psi(\xi, \delta)$ is some potential function of ξ, which can take different forms. We selected the function proposed by Green in

P. Green. Bayesian reconstructions from emission tomography data using a modified EM algorithm. *IEEE Transactions on Medical Imaging*, 9:84–93, 1990,

$$\psi(\xi, \delta) = \delta \log[\cosh(\xi/\delta)], \tag{21}$$

which leads to

$$[\nabla_X J_X(\boldsymbol{X})]_{jk} = \sum_{l \in S_k} w_{kl} \tanh\left(\frac{x_{jk} - x_{jl}}{\delta}\right). \tag{22}$$

We assumed $\delta = 0.1$, $w_{kl} = 1$ and $S_k = \{k-1, k+1\}$.

Sparse solutions: To enforce sparse representations the updates can be simplified as follows:

$$a_{ij} \leftarrow a_{ij} \frac{\left[[\boldsymbol{Y}\,\boldsymbol{X}^T]_{ij} - \alpha_A\right]_\varepsilon}{[\boldsymbol{A}\,\boldsymbol{X}\,\boldsymbol{X}^T]_{ij}}, \tag{23}$$

$$x_{jk} \leftarrow x_{jk} \frac{\left[[\boldsymbol{A}^T\,\boldsymbol{Y}]_{jk} - \alpha_X\right]_\varepsilon}{[\boldsymbol{A}^T\boldsymbol{A}\,\boldsymbol{X}]_{jk}}, \tag{24}$$

$$a_{ij} = \frac{a_{ij}}{\sum_{j=1}^{r} a_{ij}}, \tag{25}$$

where $[\xi]_\varepsilon = \max\{\varepsilon, \xi\}$. The positive parameters α_X and α_A control the level of sparsity of $\boldsymbol{X}$ and $\boldsymbol{A}$, respectively. The normalization of the columns of $\boldsymbol{A}$ is necessary to ensure a control of sparsity.[6]

Regularization of EMML and Lee-Seung algorithm for K-L cost function: Sparsity constraints can be also added to the Kullback-Leibler cost function – see (13). However, in order to enforce sparsity while using the Kullback-Leibler loss function we proposed a different approach. The EMML or Lee-Seung algorithm with Kullback-Leibler cost function are regularized using non-linear projections based on equations (8) and (9) and yield the following update rules:

[6] Also, normalization of $\boldsymbol{A}$ in some algorithms is a key factor as it prevents them from becoming unstable. It has been found that the performance of some algorithms can be improved by normalizing $\boldsymbol{A}$.

$$a_{ij} \leftarrow \left(a_{ij} \frac{\sum_{k=1}^{T} x_{jk} \, (y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})}{\sum_{p=1}^{T} x_{jp}} \right)^{1+\alpha_{Sa}}, \tag{26}$$

$$x_{jk} \leftarrow \left(x_{jk} \, \frac{\sum_{i=1}^{m} a_{ij} \, (y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})}{\sum_{q=1}^{m} a_{qj}} \right)^{1+\alpha_{Sx}}, \tag{27}$$

$$a_{ij} \leftarrow \frac{a_{ij}}{\sum_{i=1}^{m} a_{ij}}, \tag{28}$$

where α_{Sa} and α_{Sx} enforce sparse solutions. Typically: $\alpha_{Sa},\ \alpha_{Sx} \in [0.001, 0.005]$[7].

This is a rather intuitive or heuristic approach to sparsification. Raising the Lee-Seung learning rules to the power of $(1+\alpha_{Sa})$ or $(1+\alpha_{Sx})$, and having $\alpha > 0$ resulting in an exponent that is greater than one, implies that the small values in the non-negative matrix tend to zero as the number of iterations increase. The components of the non-negative matrix that are greater than one are forced to be larger. In practice these learning rules are stable and the algorithm works well.

In conclusion, the squared Euclidean distance is sparsified by the additive penalty terms, i.e., the algorithm given by (14), (15) and (16), whereas the K-L divergence is regularized using non-linear projections, i.e., the algorithm given by (26),(27) and(28).

Generalized β-divergence: The generalized divergence that unified the Euclidean distance and the Kullback-Leibler divergence was proposed by Kompass in

R. Kompass. A generalized divergence measure for nonnegative matrix factorization. Neuroinfomatics Workshop, Torun, Poland, September, 2005.

The same β-divergence was proposed earlier by Minami and Eguchi for application in BSS:

M. Minami and S. Eguchi. Robust blind source separation by beta-divergence. *Neural Computation*, 14:1859–1886, 2002.

The generalized β-divergence has the form:

$$D^{\beta}(\boldsymbol{Y}, \boldsymbol{AX}) = \sum_{i=1}^{m} \sum_{k=1}^{T} \begin{cases} y_{ik} \frac{y_{ik}^{\beta} - [\boldsymbol{AX}]_{ik}^{\beta}}{\beta(\beta+1)} + \frac{1}{\beta+1} [\boldsymbol{AX}]_{ik}^{\beta} [\boldsymbol{AX} - \boldsymbol{Y}]_{ik}, & \text{for } \beta \in [-1, 1], \\ y_{ik}(\log y_{ik} - \log[\boldsymbol{AX}]_{ik}) + [\boldsymbol{AX}]_{ik} - y_{ik}, & \text{for } \beta = 0. \end{cases}$$

[7] In NMFLAB, after selecting the `Regularized Lee-Seung algorithms`, press `ADV. OPTIONS` to set up α_{Sx} and α_{Sa}. As default, $\alpha_{Sx} = 0$ and $\alpha_{Sa} = 0$.

Hence, we can derive the Kompass algorithm as follows:

$$a_{ij} \leftarrow a_{ij} \frac{\sum_{k=1}^{T} x_{jk} \, [\boldsymbol{A}\,\boldsymbol{X}]_{ik}^{\beta-1}}{\sum_{k=1}^{T} x_{jk} [\boldsymbol{A}\boldsymbol{X}]_{ik}^{\beta}}, \tag{29}$$

$$x_{jk} \leftarrow x_{jk} \frac{\sum_{i=1}^{m} a_{ij} \, [\boldsymbol{A}\,\boldsymbol{X}]_{ik}^{\beta-1}}{\sum_{i=1}^{m} a_{ij} [\boldsymbol{A}\boldsymbol{X}]_{ik}^{\beta}}, \tag{30}$$

$$a_{ij} \leftarrow \frac{a_{ij}}{\sum_{i=1}^{m} a_{ij}}. \tag{31}$$

Regularized ALS: The gradients of the squared Euclidean distance (2) with respect to $\boldsymbol{A}$ and $\boldsymbol{X}$ are given by

$$\nabla_X D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X} - \boldsymbol{A}^T\boldsymbol{Y}, \tag{32}$$

$$\nabla_A D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = \boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^T - \boldsymbol{Y}\boldsymbol{X}^T. \tag{33}$$

Assuming $\nabla_X D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = 0$ and $\nabla_A D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = 0$ for $\boldsymbol{A} > 0$ and $\boldsymbol{X} > 0$, we have:

$$\boldsymbol{X} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{Y}, \tag{34}$$

$$\boldsymbol{A} = \boldsymbol{Y}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}. \tag{35}$$

The positive solutions are obtained by projecting (34) and (35) onto $\mathbb{R}_+$, i.e., $\boldsymbol{X} \leftarrow \max\{\varepsilon, \boldsymbol{X}\}$, $\boldsymbol{A} \leftarrow \max\{\varepsilon, \boldsymbol{A}\}$. This projection enforces sparsity in $\boldsymbol{A}$ and $\boldsymbol{X}$, and is therefore, suited for use with sparse representations. In case of the normal matrices $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{X}\boldsymbol{X}^T$ being very ill-conditioned, (35) and (34) can be implemented with the regularized pseudo-inversion, i.e.,

$$\boldsymbol{X} \leftarrow \max\{\varepsilon, (\boldsymbol{A}^T\boldsymbol{A} + \alpha_X)^+\boldsymbol{A}^T\boldsymbol{Y}\}, \tag{36}$$

$$\boldsymbol{A} \leftarrow \max\{\varepsilon, \boldsymbol{Y}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \alpha_A)^+\}. \tag{37}$$

To avoid getting stuck in local minima, we assume the regularization parameters α_X and α_A depend on the k-th iterative step, i.e.

$$\alpha_A^{(k)} = \alpha_X^{(k)} = \alpha_0 \exp\left\{-\frac{k}{\tau}\right\},$$

where α_0 and τ are parameters[8]. This approach is motivated by a temperature schedule in the simulated annealing technique. Thus, the improved form of the algorithm, which we called *Regularized Alternating Least Squares (RALS)*[9], is as follows:

[8] As default in NMFLAB we set $\alpha_0 = 20$ and $\tau = 10$.

[9] in `ADV. OPTIONS`

Set: $\boldsymbol{A}_{(0)}$, α_0, τ,
For $k = 1, 2, \ldots$
$\alpha_{(k)} = \alpha_0 \exp\left\{-\frac{k}{\tau}\right\}$,
$\boldsymbol{X}_{(k)} = \max\left\{\varepsilon, \left(\boldsymbol{A}_{(k-1)}^T \boldsymbol{A}_{(k-1)} + \alpha_{(k)}\right)^+ \boldsymbol{A}_{(k-1)}^T \boldsymbol{Y}\right\}$,
$\boldsymbol{A}_{(k)} = \max\left\{\varepsilon, \boldsymbol{Y}\boldsymbol{X}_{(k)}^T \left(\boldsymbol{X}_{(k)}\boldsymbol{X}_{(k)}^T + \alpha_{(k)}\right)^+\right\}$,
End

Initialization: The NMF algorithms are initialized by matrices with elements drawn independently from a uniform distribution. In order to reduce the risk of getting stuck in local minima we usually perform initialization several times with different randomly generated matrices. More precisely, we run the specific algorithm several times (typically 10 times) for different initial conditions for a limited number of iterations (typically 10 iterations) and select an initial condition which provides the smallest value of the loss function. If the number of restarts in the field: `No. of restarts` is equal to zero, the selected algorithm is initialized as follows (our code in Matlab)

```
m_sx = 1:m; r_sx = 1:r; T_sx = 1:T;
Ainit(m_sx',r_sx) = abs(repmat(.1*sin(2*pi*.1*m_sx'),1,r)
+ repmat(.1*cos(2*pi*.1*r_sx),m,1)
+ repmat(cos(2*pi*.471*m_sx'),1,r) + repmat(sin(2*pi*.471*r_sx),m,1));
Ainit = Ainit/max(max(Ainit));

Xinit(r_sx',T_sx) = abs(repmat(.1*sin(2*pi*.1*r_sx'),1,T)
+ repmat(.1*cos(2*pi*.1*T_sx),r,1)
+ repmat(cos(2*pi*.471*r_sx'),1,T) + repmat(sin(2*pi*.471*T_sx),r,1));
Xinit = Xinit/max(max(Xinit));
```

Otherwise, the algorithm is initialized as many times as given in `No. of restarts` from

```
Ainit=rand(m,r);
Xinit=rand(r,T);
```

The number of the inner iterations can be given in the field: `No. of inner iterations`.

The fields `AlphaSa` and `AlphaSx` correspond to α_{Sa} and α_{Sx} in (26) and (27), and the fields `AlphaA` and `AlphaX` denote α_A and α_X in (14) and (15), respectively.

In Matlab, we coded the algorithms as follows:

```
switch type_alg_X

        case 1  % EMML
```

```
            X = (X.*(A'*(Y./(A*X + eps)))).^alphaSx;

        case 2  % ISRA

            % Smoothing
            switch reg_fun

               case 1 % L1 norm

                   psi = 1;

               case 2 % L2 norm

                   psi = X;

               case 3 % Gibbs smoothing

                   Xp1 = [X(:,T) X(:,1:T-1)]; Xm1 = [X(:,2:T) X(:,1)];
                   psi = tanh((X - Xp1)/delta) + tanh((X - Xm1)/delta);

            end % switch reg_fun

            Yx = A'*Y - alphaX*psi;
            Yx(Yx <= 0) = 100*eps;
            X = X.*(Yx./((A'*A)*X + eps));

        case 3 % Kompass algorithm

            Yx = A*X + eps;
            X = X.*(A'*(Y.*Yx.^(alpha - 1) ))./(A'*Yx.^alpha);

        case 4 % L1-regularized pseudo-inverse

            X = max(1E6*eps,pinv(A'*A + alpha_reg)*A'*Y);

        case 5 % Fixed X

            X = s + eps; % X_true
            niter_selected = 1000; % number of iterations
                                   % in computation of X at A fixed

end % type_alg_X

switch type_alg_A
```

```
        case 1  % EMML

            Ap = A;
            A = (A.*((Y./(A*X + eps))*X')./repmat(sum(X,2)',m,1)).^alphaSa;
            A = A*diag(1./(sum(A,1) + eps));

        case 2  % ISRA

            Ap = A;
            Ya = X*Y' - alphaA;
            Ya(Ya <= 0) = 100*eps;
            A = A.*(Ya./((A*(X*X'))' + eps))';
            A = A*diag(1./(sum(A,1)+eps));

        case 3 % Kompass algorithm

            Ap = A;
            Yx = A*X + eps;
            A = A.*((Y.*Yx.^(alpha - 1))*X')./((Yx.^alpha)*X');
            A = A*diag(1./sum(A,1));

        case 4 % L1-regularized pseudo-inverse

            Ap = A;
            A = max(1E6*eps,Y*X'*pinv(X*X' + alpha_reg_A));
            A = A*diag(1./sum(A,1));

        case 5  % Fixed A

            niter_selected = 1000; % number of iterations
                                   % in computation of A at X fixed
            A = A_true + eps;
            A = A*diag(1./(sum(A,1) + eps));

end % type_alg_A
```

2 Projected gradient NMF (Group of projected gradient algorithms)

This group is composed of the following algorithms in the alternating mode: GPCG, PG, IPG, Regularized MRNSD, Relaxed Hellinger, and Projected pseudo-inverse.

In contrast to the multiplicative Lee-Seung algorithms, this class of algorithms has additive updates. The algorithms in this group mostly use the squared

Euclidean distance (2). The projected gradient method can be generally expressed by iterative updates:

$$\boldsymbol{X}^{(k+1)} = P_{\Omega}[\boldsymbol{X}^{(k)} + \eta_X^{(k)} \boldsymbol{P}_X], \tag{38}$$

$$\boldsymbol{A}^{(k+1)} = P_{\Omega}[\boldsymbol{A}^{(k)} + \eta_A^{(k)} \boldsymbol{P}_A], \tag{39}$$

where $P_{\Omega}[\xi]$ is a projection of ξ onto feasible set Ω (i.e., in our case non-negativity of entries of $\boldsymbol{X}$ and $\boldsymbol{A}$), $\boldsymbol{P}_X$ and $\boldsymbol{P}_A$ are descent directions for $\boldsymbol{X}$ and $\boldsymbol{A}$, respectively. There are many rules for choosing the learning rates $\eta_X^{(k)}$ and $\eta_A^{(k)}$. In many methods included in the NMFLAB, the learning rules are adjusted in this way to maintain nonnegativity constraints, which is equivalent to performing the projection $P_{\Omega}[\xi]$.

The NMFLAB contains the various modifications of the alternating projected algorithm. The fields: `No. of restarts` and `No. of inner iterations` have the same meaning as for the `Regularized Lee-Seung algorithms`.

2.1 Alternating Gradient Projected Conjugate Gradient (GPCG) algorithm

The Gradient Projected Conjugate Gradient (GPCG) algorithm that was originally invented by More and Toraldo in

J. J. More and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optimization*, 1(1):93–113, 1991.

is designed for solving large-scale, nonnegatively constrained optimization problems. In our approach, we extended the "reduced Newton" version of the GPCG which was proposed by Bardsley and Vogel in

J. M. Bardsley and C. R. Vogel. Nonnegatively constrained convex programming method for image reconstruction. *SIAM J. Sci. Comput.*, 4:1326–1343, 2004

to the problem of NMF. This algorithm is especially efficient for solving sparse and ill-conditioned problems, and this motivates us to extend and implement this approach to NMF. In the NMFLAB, the GPCG is applied to the cost function $D_F^{\alpha_A,\alpha_X}(\boldsymbol{Y}||\boldsymbol{AX})$ in (12). This is a second-order algorithm which is based on the Newton method, however, it is free from a direct inversion of the Hessian. This is possible because for the convex cost function $D_F(\boldsymbol{Y}||\boldsymbol{AX}) = (1/2)||\boldsymbol{Y} - \boldsymbol{AX}||_F^2$ the Hessian matrix is a symmetric positive definite matrix, and hence, the Conjugate Gradient (CG) method can be applied to the following systems of linear equations:

$$\nabla_X^2 D_F(\boldsymbol{Y}||\boldsymbol{AX})\boldsymbol{P}_X = -\nabla_X D_F(\boldsymbol{Y}||\boldsymbol{AX}), \tag{40}$$

$$\nabla_A^2 D_F(\boldsymbol{Y}||\boldsymbol{AX})\boldsymbol{P}_A = -\nabla_A D_F(\boldsymbol{Y}||\boldsymbol{AX}), \tag{41}$$

where $\boldsymbol{P}_X$ and $\boldsymbol{P}_A$ are the estimated descent directions. The terms $\nabla_X D_F(\boldsymbol{Y}||\boldsymbol{AX})$ and $\nabla_A D_F(\boldsymbol{Y}||\boldsymbol{AX})$ denote the gradient with respect to $\boldsymbol{X}$ and $\boldsymbol{A}$, respectively.

The expressions $\nabla^2_X D_F(\boldsymbol{Y}||\boldsymbol{AX})$ and $\nabla^2_A D_F(\boldsymbol{Y}||\boldsymbol{AX})$ are the corresponding Hessian matrices. Solving equations (40) and (41), we get the descent directions $\boldsymbol{P}_X$ and $\boldsymbol{P}_A$ that are then used for computing the projections with the Projected Gradient(PG) iterations:

$$\boldsymbol{X}^{(k+1)} = P_\Omega[\boldsymbol{X}^{(k)} + \eta^{(k)}\boldsymbol{P}_X],$$

$$\boldsymbol{A}^{(k+1)} = P_\Omega[\boldsymbol{A}^{(k)} + \eta^{(k)}\boldsymbol{P}_A],$$

where learning rate $\eta^{(k)}$ is adjusted with the Armijo rule, and Ω is a set of feasible solutions – in our case: $P_\Omega[\boldsymbol{X}] = [\boldsymbol{X}]_\epsilon$ (See (17)). Another advantage of the GPCG approach is the application of the CG iterations only to the non-boundary variables. These variables which are projected on the boundary (zero) are excluded from the gradient and Hessian. This is particularly useful in NMF with sparse data. More details on the "standard" GPCG in application to different problems can be found in

J. J. More and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM J. Optimization*, 1(1):93–113, 1991.
J. M. Bardsley and C. R. Vogel. Nonnegatively constrained convex programming method for image reconstruction. *SIAM J. Sci. Comput.*, 4:1326–1343, 2004.
J. M. Bardsley. A nonnegatively constrained trust region algorithm for the restoration of images with an unknown blur. *Electronic Transactions on Numerical Analysis*, 20, 2005.

The Algorithm 1 is the simplified version of the GPCG adapted for NMF.

Algorithm 1 *(GPCG-NMF)*
Set $\boldsymbol{A}$, $\boldsymbol{X}$, $\beta \in (0,1)$, $\mu \in (0,1)$, $\gamma_{GP} \in (0,1)$, *% Initialization*
For $s = 0, 1, \ldots,$ *% Alternating*
Step 1: Do **X-GPCG** iterations with Algorithm 2,
Step 2: Do **A-GPCG** iterations with Algorithm 3,
End *% Alternating*

Algorithm 2 *(***X-GPCG***)*

For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{X}$

Step 1: $\boldsymbol{P}^{(X)} \leftarrow -\nabla_X D_F(\boldsymbol{A}, \boldsymbol{X}) = \boldsymbol{A}^T(\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}) \in \mathbb{R}^{r \times T}$,
$\mathrm{vec}(\boldsymbol{P}^{(X)}) = [p_{11}^{(X)}, p_{21}^{(X)}, \ldots, p_{r1}^{(X)}, p_{12}^{(X)}, \ldots, p_{rT}^{(X)}]^T \in \mathbb{R}^{rT}$,
where $\boldsymbol{P}^{(X)} = [p_{jk}^{(X)}]$, *% Vectorization*

Step 2: $\boldsymbol{H}_X = \mathrm{Hess}_X D_F(\boldsymbol{A}, \boldsymbol{X}) = \mathrm{diag}\{[\boldsymbol{h}_k^{(X)}]_{k=1,\ldots,T}\} \in \mathbb{R}^{rT \times rT}$,
$\boldsymbol{h}_k^{(X)} = \boldsymbol{A}^T\boldsymbol{A} + \alpha_X \nabla^2 J_X(\boldsymbol{X})$,

Step 3: $\boldsymbol{X} \leftarrow \max\{\boldsymbol{X} + \eta^p \boldsymbol{P}^{(X)}, 0\}$, *% Projection*
where $\eta^p = \beta^p \eta_0$, $\eta_0 = \dfrac{\mathrm{vec}(\boldsymbol{P}^{(X)})^T \mathrm{vec}(\boldsymbol{P}^{(X)})}{\mathrm{vec}(\boldsymbol{P}^{(X)})^T \boldsymbol{H}_X \mathrm{vec}(\boldsymbol{P}^{(X)})}$,
and $p = 0, 1, \ldots$ is the first non-negative integer for which:
$D_F(\boldsymbol{X}(\eta^p)) - D_F(\boldsymbol{X}) \leq -\frac{\mu}{\eta^p} ||\boldsymbol{X} - \boldsymbol{X}(\eta^p)||_2^2$,
$\boldsymbol{Z}^{(X)} = \mathrm{diag}\{z_{jk}^{(X)}\}$, $z_{jk}^{(X)} = \begin{cases} 1 & \text{if} \quad x_{jk} > 0, \\ 0 & \text{otherwise,} \end{cases}$

Step 4: $\boldsymbol{P}_R^{(X)} = \boldsymbol{Z}^{(X)} \odot \boldsymbol{P}^{(X)}$, *% Reduced gradient*
$\boldsymbol{H}_R^{(X)} = \mathrm{diag}\{\mathrm{vec}(\boldsymbol{Z}^{(X)})\}\boldsymbol{H}_X \,\mathrm{diag}\{\mathrm{vec}(\boldsymbol{Z}^{(X)})\} + \mathrm{diag}\{1 - \mathrm{vec}(\boldsymbol{Z}^{(X)})\}$,

Step 5: Solve: $\boldsymbol{H}_R^{(X)} \boldsymbol{p}_X = -\mathrm{vec}(\boldsymbol{P}_R^{(X)})$ with `CG` algorithm
$\boldsymbol{P}^{(X)} \leftarrow \mathrm{Matrix}(\boldsymbol{p}_X) \in \mathbb{R}^{r \times T}$,

Step 6: $\boldsymbol{X} \leftarrow \max\{\boldsymbol{X} + \eta^p \boldsymbol{P}^{(X)}, 0\}$, *% Projection*
where $\eta^p = \beta^p \eta_0$, $\eta_0 = 1$,
and $p = 0, 1, \ldots$ is the first non-negative integer for which:
$D_F(\boldsymbol{X}(\eta^p)) < D_F(\boldsymbol{X})$,

If $D_F(\boldsymbol{X}^{(n-1)}) - D_F(\boldsymbol{X}^{(n)}) \leq \gamma_{GP} \max\{D_F(\boldsymbol{X}^{(l-1)}) - D_F(\boldsymbol{X}^{(l)}) | l = 1, \ldots, n-1\}$,
Break
End If
End *% Inner loop for* $\boldsymbol{X}$

Algorithm 3 *(***A-GPCG***)*

For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{A}$

Step 1: $\boldsymbol{P}^{(A)} \leftarrow -\nabla_A D_F(\boldsymbol{A}, \boldsymbol{X}) = (\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X})\boldsymbol{X}^T \in \mathbb{R}^{m \times r}$,
$\mathrm{vec}(\boldsymbol{P}^{(A)}) = [p_{11}^{(A)}, p_{12}^{(A)}, \ldots, p_{1r}^{(A)}, p_{21}^{(A)}, \ldots, p_{mr}^{(A)}]^T \in \mathbb{R}^{mr}$,
where $\boldsymbol{P}^{(A)} = [p_{ij}^{(A)}]$, *% Vectorization*

Step 2: $\boldsymbol{H}_A = \mathrm{Hess}_A D_F(\boldsymbol{A}, \boldsymbol{X}) = \mathrm{diag}\{[\boldsymbol{h}_i^{(A)}]_{i=1,\ldots,m}\} \in \mathbb{R}^{mr \times mr}$,
$\boldsymbol{h}_i^{(A)} = \boldsymbol{X}\boldsymbol{X}^T + \alpha_A \nabla^2 J_A(\boldsymbol{A})$,

Step 3: $\boldsymbol{A} \leftarrow \max\{\boldsymbol{A} + \eta^p \boldsymbol{P}^{(A)}, 0\}$, *% Projection*
where $\eta^p = \beta^p \eta_0$, $\eta_0 = \dfrac{\mathrm{vec}(\boldsymbol{P}^{(A)})^T \mathrm{vec}(\boldsymbol{P}^{(A)})}{\mathrm{vec}(\boldsymbol{P}^{(A)})^T \boldsymbol{H}_X \mathrm{vec}(\boldsymbol{P}^{(A)})}$,
and $p = 0, 1, \ldots$ is the first non-negative integer for which:
$D_F(\boldsymbol{A}(\eta^p)) - D_F(\boldsymbol{A}) \leq -\frac{\mu}{\eta^p} ||\boldsymbol{A} - \boldsymbol{A}(\eta^p)||_2^2$,
$\boldsymbol{Z}^{(A)} = \mathrm{diag}\{z_{ij}^{(A)}\}$, $z_{ij}^{(A)} = \begin{cases} 1 & \text{if} \quad a_{ij} > 0, \\ 0 & \text{otherwise}, \end{cases}$

Step 4: $\boldsymbol{P}_R^{(A)} = \boldsymbol{Z}^{(A)} \odot \boldsymbol{P}^{(A)}$, *% Reduced gradient*
$\boldsymbol{H}_R^{(A)} = \mathrm{diag}\{\mathrm{vec}(\boldsymbol{Z}^{(A)})\} \boldsymbol{H}_A \, \mathrm{diag}\{\mathrm{vec}(\boldsymbol{Z}^{(A)})\} + \mathrm{diag}\{1 - \mathrm{vec}(\boldsymbol{Z}^{(A)})\}$,

Step 5: Solve: $\boldsymbol{H}_R^{(A)} \boldsymbol{p}_A = -\mathrm{vec}(\boldsymbol{P}_R^{(A)})$ with `CG` algorithm
$\boldsymbol{P}^{(A)} \leftarrow \mathrm{Matrix}(\boldsymbol{p}_A) \in \mathbb{R}^{m \times r}$,

Step 6: $\boldsymbol{X} \leftarrow \max\{\boldsymbol{A} + \eta^p \boldsymbol{P}^{(A)}, 0\}$, *% Projection*
where $\eta^p = \beta^p \eta_0$, $\eta_0 = 1$,
and $p = 0, 1, \ldots$ is the first non-negative integer for which:
$D_F(\boldsymbol{A}(\eta^p)) < D_F(\boldsymbol{A})$,

If $D_F(\boldsymbol{A}^{(n-1)}) - D_F(\boldsymbol{A}^{(n)}) \leq \gamma_{GP} \max\{D_F(\boldsymbol{A}^{(l-1)}) - D_F(\boldsymbol{A}^{(l)}) | l = 1, \ldots, n-1\}$,
Break

End If

End *% Inner loop for* $\boldsymbol{A}$

In the GPCG algorithm given by Bardsley and Vogel, the regularization term can be specified by the semi-norm $||\boldsymbol{L}_X \boldsymbol{X}||_2$, or alternatively in our application by $||\boldsymbol{L}_A \boldsymbol{A}||_2$.

In the NMFLAB, we used only $\boldsymbol{L}_X$ to introduce a prior knowledge about the estimated sources. Thus the cost function to be minimized is given by

$$D_F^{(\alpha)}(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2 + \frac{\alpha}{2}||\boldsymbol{L}\boldsymbol{X}||_F^2, \tag{42}$$

where α is a regularization term. In the field `L-seminorm`, $\boldsymbol{L}$ can be specified by an identity matrix ($\boldsymbol{L} = \boldsymbol{I}$), or by the banded matrix that represents the first or second derivative of the estimation. The regularization parameter is denoted by `AlphaX`.

2.2 Alternating Projected Gradient (PG) algorithm

We formulated the NMF problem as the following optimization problem:

$$\min_{\mathbf{X}\in\mathbb{R}^{n\times N},\, \mathbf{A}\in\mathbb{R}^{m\times n}} D_F(\boldsymbol{Y}||\boldsymbol{AX}), \quad \text{s.t.} \quad a_{ij}, x_{jk} \geq 0, \tag{43}$$

which can be also solved with the following alternating projected iterative updates

$$\boldsymbol{X}^{(k+1)} = P_\Omega[\boldsymbol{X}^{(k)} - \eta_X^{(k)} \nabla_X D_F(\boldsymbol{Y}||\boldsymbol{A}^{(k)}\boldsymbol{X})|_{\boldsymbol{X}=\boldsymbol{X}^{(k)}}], \tag{44}$$

$$\boldsymbol{A}^{(k+1)} = P_\Omega[\boldsymbol{A}^{(k)} - \eta_A^{(k)} \nabla_A D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}^{(k+1)})|_{\boldsymbol{A}=\boldsymbol{A}^{(k)}}], \tag{45}$$

where

$$P_\Omega[\xi] = \begin{cases} \xi & \text{if} \quad \xi \geq 0, \\ 0 & \text{otherwise} \end{cases} . \tag{46}$$

Several choices are available for selecting the optimal values of $\eta_X^{(k)}$ and $\eta_A^{(k)}$ in each iteration k.

For example, Chih-Jen Lin in

Ch-J. Lin. Projected gradient methods for non-negative matrix factorization. Technical report, Department of Computer Science, National Taiwan University, 2005.

applied the Armijo rule to the squared Euclidean distance. For computation of $\boldsymbol{X}$, η_X is chosen to satisfy

$$\eta_X^{(k)} = \beta^{m_k}, \tag{47}$$

where m_k is the first non-negative integer m for which

$$D_F(\boldsymbol{Y}||\boldsymbol{AX}^{(k+1)}) - D_F(\boldsymbol{Y}||\boldsymbol{AX}^{(k)}) \leq \sigma \nabla_X D_F(\boldsymbol{Y}||\boldsymbol{AX}^{(k)})^T(\boldsymbol{X}^{(k+1)} - \boldsymbol{X}^{(k)}).$$

A similar rule is applied for computing $\boldsymbol{A}$. We extend Lin's algorithm to other divergences $D(\boldsymbol{Y}||\boldsymbol{AX})$ such as the Kullback-Leibler, dual Kullback-Leibler or α-divergences. The parameters β and σ control a convergence speed. In our algorithm we set $\sigma = 0.01$.

In the NMFLAB, β in (47) can be set from the inverse parameter `Beta-Armijo`, where $\beta = (Beta_{Armijo})^{-1}$.

The field `Alpha-Amari` used in `Advanced Options` refers to parameter α which is a basic parameter in the Amari α-divergence, and it is discussed in detail in Section 3.

2.3 Alternating Interior Point Gradient (IPG) algorithm

The Interior Point Gradient (IPG) algorithm was proposed by Merritt and Zhang in

M. Merritt and Y. Zhang. An interior-point gradient method for large-scale totally nonnegative least squares problems. Technical report, Department of Computational and Applied Mathematics, Rice University, Houston, Texas, USA, 2004

for a nonnegatively constrained least-squares problem. We extend this algorithm to NMF applications. This algorithm is based on a scaled gradient descent method, where the descent direction $\boldsymbol{P}^{(k)}$ for $\boldsymbol{X}$ is determined by a negative scaled gradient, i.e.

$$\boldsymbol{P}^{(k)} = -\boldsymbol{D}^{(k)} \odot \nabla_X D_F(\boldsymbol{Y}||\boldsymbol{A}^{(k)}\boldsymbol{X}^{(k)}), \tag{48}$$

with the scaling vector

$$\boldsymbol{D}^{(k)} = \boldsymbol{X}^{(k)} \oslash (\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}^{(k)}). \tag{49}$$

In the NMFLAB, the cost function $D_F(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})$ is an unregularized squared Euclidean distance (squared Frobenius norm).

In interior-point methods, the step length is adjusted in each iteration to keep the iterates positive. In the IPG algorithm, the step length η_k is chosen so as to be close to η_k^* which is the exact minimizer of $D_F(\boldsymbol{X}^{(k)} + \eta\boldsymbol{P}^{(k)}|\boldsymbol{Y}, \boldsymbol{A}^{(k)})$ or $D_F(\boldsymbol{A}^{(k)} + \eta\boldsymbol{P}^{(k)}|\boldsymbol{Y}, \boldsymbol{X}^{(k)})$, and on the other hand, to maintain some distance to the boundary of the nonnegative orthant.

In the NMFLAB the following implementation of the alternating IPG algorithm is used:

Algorithm 4 *(IPG-NMF)*

Set $\boldsymbol{A}$, $\boldsymbol{X}$, $\tau \in (0,1)$, *% Initialization*
For $s = 0, 1, \ldots,$ *% Alternating*
Step 1: Do **A-IPG** iterations with Algorithm 5,
Step 2: Do **X-IPG** iterations with Algorithm 6,
End *% Alternating*

Algorithm 5 *(***A-IPG***)*

For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{A}$
$\nabla_A D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}) \leftarrow (\boldsymbol{A}\boldsymbol{X} - \boldsymbol{Y})\boldsymbol{X}^T$, *% Gradient*
$\boldsymbol{D} \leftarrow \boldsymbol{A} \oslash (\boldsymbol{A}\boldsymbol{X}\boldsymbol{X}^T)$, *% Scaling vector*
$\boldsymbol{P} \leftarrow -\boldsymbol{D} \odot \nabla_A(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})$, *% Descent direction*
$\eta^* = -\big(\mathrm{vec}(\boldsymbol{P})^T \mathrm{vec}(\nabla_A D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X}))\big) \,/\, \big(\mathrm{vec}(\boldsymbol{P}\boldsymbol{X})^T \mathrm{vec}(\boldsymbol{P}\boldsymbol{X})\big)$, *% Exact minimizer*
$\hat{\eta} = \max\{\eta : \boldsymbol{A} + \eta\boldsymbol{P} \geq 0\}$, *% Step length towards boundary*
Set: $\hat{\tau} \in [\tau, 1)$, $\eta = \min(\hat{\tau}\hat{\eta}, \eta^*)$, *% Current step length*
$\boldsymbol{A} \leftarrow \boldsymbol{A} + \eta\boldsymbol{P}$, *% Update*
End *% Alternating*

Algorithm 6 *(**X-IPG**)*

For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{X}$

$\nabla_X D(\boldsymbol{Y}||\boldsymbol{AX}) \leftarrow \boldsymbol{A}^T(\boldsymbol{AX} - \boldsymbol{Y})$, *% Gradient*

$\boldsymbol{D} \leftarrow \boldsymbol{X} \oslash (\boldsymbol{A}^T\boldsymbol{AX})$, *% Scaling vector*

$\boldsymbol{P} \leftarrow -\boldsymbol{D} \odot \nabla_X(\boldsymbol{Y}||\boldsymbol{AX})$, *% Descent direction*

$\eta^* = -\left(\text{vec}(\boldsymbol{P})^T \text{vec}(\nabla_X D(\boldsymbol{Y}||\boldsymbol{AX}))\right) / \left(\text{vec}(\boldsymbol{AP})^T \text{vec}(\boldsymbol{AP})\right)$, *% Exact minimizer*

$\hat{\eta} = \max\{\eta : \boldsymbol{X} + \eta\boldsymbol{P} \geq 0\}$, *% Step length towards boundary*

Set: $\hat{\tau} \in [\tau, 1)$, $\eta = \min(\hat{\tau}\hat{\eta}, \eta^*)$, *% Current step length*

$\boldsymbol{X} \leftarrow \boldsymbol{X} + \eta\boldsymbol{P}$, *% Update*

End *% Alternating*

2.4 Alternating Regularized Minimal Residual Norm Steepest Descent (MRNSD) algorithm

The MRNSD algorithm, which has been proposed by Nagy and Strakos

J. G. Nagy and Z. Strakos. Enforcing Nonnegativity in Image Reconstruction Algorithms, Mathematical Modeling, Estimation, and Imaging. volume 4121, pages 182–190, 2000.

for image restoration problems (where $\boldsymbol{A}$ is known), has been found to be the same as the EMLS algorithm developed by Kaufman

L. Kaufman. Maximum likelihood, least squares, and penalized least squares for PET. *IEEE Transactions on Medical Imaging*, 12(2):200–214, 1993.

The original MRNSD algorithm solves the problem (assuming that the basis matrix $\boldsymbol{A}$ is known)

$$\Phi(\boldsymbol{x}(k)) = \frac{1}{2}||\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}(k)||_2^2, \quad \text{subject to } \boldsymbol{x}(k) \geq 0, \ \ k = 1, \ldots, T. \tag{50}$$

The nonnegativity is achieved by assuming the nonlinear transformation $\boldsymbol{x}(k) = \exp\{\boldsymbol{z}(k)\}$, and then

$$\begin{aligned} \nabla_{z(k)}\Phi(\boldsymbol{x}(k)) &= \text{diag}(\boldsymbol{x}(k))\nabla_{x(k)}\Phi(\boldsymbol{x}(k)) \\ &= \text{diag}(\boldsymbol{x}(k))\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}(k) - \boldsymbol{y}) = 0 \end{aligned}$$

satisfies the KKT conditions. The solution is updated by the following rules:

$$\boldsymbol{p} \leftarrow \text{diag}(\boldsymbol{x}(k))\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}(k) - \boldsymbol{y}), \quad \boldsymbol{x}(k) \leftarrow \boldsymbol{x}(k) + \eta\boldsymbol{p}.$$

In the NMFLAB, we applied the MRNSD approach to the regularized cost function:

$$D_F^{(\alpha_X)}(\boldsymbol{Y}||\boldsymbol{AX}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{AX}\|_F^2 + \alpha_X J_X(\boldsymbol{X}), \tag{51}$$

where the matrices $\boldsymbol{A}$ and $\boldsymbol{X}$ are unknown and the regularization term has the form $J_X(\boldsymbol{X}) = \sum_{j=1}^{r}\sum_{k=1}^{T} x_{jk}$. We have used the following implementation of the MRNSD algorithm:

Algorithm 7 *(MRNSD-NMF)*

Set $\boldsymbol{A}$, $\boldsymbol{X}$, $\mathbf{1}_m = [1,\ldots,1]^T \in \mathbb{R}^m$,
$\mathbf{1}_r = [1,\ldots,1]^T \in \mathbb{R}^r$, $\mathbf{1}_T = [1,\ldots,1]^T \in \mathbb{R}^T$, *% Initialization*
For $s = 0, 1, \ldots,$ *% Alternating*
Step 1: Do **A-MRNSD** iterations with Algorithm 8,
Step 2: Do **X-MRNSD** iterations with Algorithm 9,
End *% Alternating*

Algorithm 8 *(***A-MRNSD***)*

$\boldsymbol{G} = \nabla_A D(\boldsymbol{Y}||\boldsymbol{AX}) = (\boldsymbol{AX} - \boldsymbol{Y})\boldsymbol{X}^T$, *% Gradient*
$\gamma = \mathbf{1}_m^T[\boldsymbol{G} \odot \boldsymbol{A} \odot \boldsymbol{G}]\mathbf{1}_r^T$
For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{A}$
$\boldsymbol{P} = -\boldsymbol{A} \odot \boldsymbol{G}$, $\boldsymbol{U} = \boldsymbol{PX}$,
$\eta = \min\left\{\gamma\left(\mathbf{1}_m^T[\boldsymbol{U} \odot \boldsymbol{U}]\mathbf{1}_T\right)^{-1}, \min_{p_{jk}<0}(-\boldsymbol{A} \oslash \boldsymbol{P})\right\}$, *% Step length*
$\boldsymbol{A} \leftarrow \boldsymbol{A} + \eta\boldsymbol{P}$, $\boldsymbol{Z} = \boldsymbol{UX}^T$, $\boldsymbol{G} \leftarrow \boldsymbol{G} + \eta\boldsymbol{Z}$ *% Updates*
$\gamma = \mathbf{1}_m^T[\boldsymbol{G} \odot \boldsymbol{X} \odot \boldsymbol{G}]\mathbf{1}_r^T$
End *% Alternating*

Algorithm 9 *(***X-MRNSD***)*

$\boldsymbol{G} = \nabla_X D(\boldsymbol{Y}||\boldsymbol{AX}) = \boldsymbol{A}^T(\boldsymbol{AX} - \boldsymbol{Y}) + \eta_X$, *% Gradient*
$\gamma = \mathbf{1}_r^T[\boldsymbol{G} \odot \boldsymbol{X} \odot \boldsymbol{G}]\mathbf{1}_T^T$
For $n = 0, 1, \ldots,$ *% Inner loop for* $\boldsymbol{X}$
$\boldsymbol{P} = -\boldsymbol{X} \odot \boldsymbol{G}$, $\boldsymbol{U} = \boldsymbol{AP}$,
$\eta = \min\left\{\gamma\left(\mathbf{1}_m^T[\boldsymbol{U} \odot \boldsymbol{U}]\mathbf{1}_T\right)^{-1}, \min_{p_{jk}<0}(-\boldsymbol{X} \oslash \boldsymbol{P})\right\}$, *% Step length*
$\boldsymbol{X} \leftarrow \boldsymbol{X} + \eta\boldsymbol{P}$, $\boldsymbol{Z} = \boldsymbol{A}^T\boldsymbol{U}$, $\boldsymbol{G} \leftarrow \boldsymbol{G} + \eta\boldsymbol{Z}$ *% Updates*
$\gamma = \mathbf{1}_r^T[\boldsymbol{G} \odot \boldsymbol{X} \odot \boldsymbol{G}]\mathbf{1}_T^T$
End *% Alternating*

In the NMFLAB (see `Advanced Options`), the regularization parameter α_X is denoted by `AlphaX`.

2.5 Relaxed Hellinger algorithm

The relaxed Hellinger algorithm considered in

P. P. B. Eggermont and V. N. LaRiccia. On EM-like algorithms for minimum distance estimation. Technical report, University of Delaware, Newark, USA, 1998. http://www.udel.edu/FREC/eggermont/Preprints/emlike.pdf

solves the optimization problem:

$$\min_{X \geq 0} \{D_{KL}(\boldsymbol{Y}||\boldsymbol{AX}) + D_H(\boldsymbol{Y}||\boldsymbol{AX})\}, \tag{52}$$

where $D_{KL}(\boldsymbol{Y}||\boldsymbol{AX})$ is Kullback-Leibler divergence (3), and

$$D_H(\boldsymbol{Y}||\boldsymbol{AX}) = \sum_{i=1}^{m}\sum_{k=1}^{T} |\sqrt{([\boldsymbol{AX}]_{ik})} - \sqrt{(y_{ik})}|^2$$

is the Hellinger distance.

Using the alternating projection technique, we have developed the following NMF implementation

Algorithm 10 *(Relaxed Hellinger)*

Set $\boldsymbol{A}$, $\boldsymbol{X}$, *% Initialization*

For $s = 0, 1, \ldots,$ *% Alternating*

For $n = 0, 1, \ldots, n_{inner},$ *% Inner loop for* $\boldsymbol{A}$

Step 1: $\boldsymbol{A} \leftarrow \boldsymbol{A} \odot \left[\frac{1}{4} + \frac{1}{4}\sqrt{1 + 8[\boldsymbol{Y} \oslash \boldsymbol{AX}]\boldsymbol{X}^T}\right]$

End *% Inner loop for* $\boldsymbol{A}$

For $n = 0, 1, \ldots, n_{inner},$ *% Inner loop for* $\boldsymbol{X}$

Step 2: $\boldsymbol{X} \leftarrow \left(\boldsymbol{X} \odot \left[\frac{1}{4} + \frac{1}{4}\sqrt{1 + 8\boldsymbol{A}^T[\boldsymbol{Y} \oslash \boldsymbol{AX}]}\right]\right)^{\alpha_{Sx}}$

End *% Inner loop for* $\boldsymbol{X}$

End *% Alternating*

The nonlinear projection parameter α_{Sx} has the same meaning as in (27), and in the NMFLAB it is denoted by `AlphaS`. The number of inner iterations n_{inner} can be entered in the field `No. of inner iterations`.

2.6 Csiszár algorithm

This is a non-negatively constrained least-squares algorithm that comes from the theory of Csiszár and Tusnady

I. Csiszár and G. Tusnády. Information geometry and alternating minimization procedures. *Statistics and Decisions Supp.*, 1:205 – 237, 1984

for alternating projections. In the NMFLAB, we implemented this algorithm with nonlinear projections for $\boldsymbol{X}$ as in (27).

Algorithm 11 *(Csiszár based algorithm)*

Set $\boldsymbol{A}$, $\boldsymbol{X}$ *% Initialization*

For $s = 0, 1, \ldots,$ *% Alternating*

For $n = 0, 1, \ldots, n_{inner},$ *% Inner loop for* $\boldsymbol{A}$

Step 1: $i = 1, \ldots, m,\ j = 1, \ldots, r$:

$$a_{ij} \leftarrow \max\left\{\epsilon, a_{ij} + \frac{\sum_{k=1}^{T} x_{jk}(y_{jk} - \sum_{j=1}^{r} a_{ij}x_{jk})}{m\sum_{k=1}^{T} x_{jk}^2}\right\},$$

End *% Inner loop for* $\boldsymbol{A}$

For $n = 0, 1, \ldots, n_{inner},$ *% Inner loop for* $\boldsymbol{X}$

Step 2: $j = 1, \ldots, r,\ k = 1, \ldots, T$:

$$x_{jk} \leftarrow \max\left\{\epsilon, \left(x_{jk} + \frac{\sum_{i=1}^{m} a_{ij}(y_{jk} - \sum_{j=1}^{r} a_{ij}x_{jk})}{r\sum_{i=1}^{m} a_{ij}^2}\right)^{\alpha_{Sx}}\right\},$$

End *% Inner loop for* $\boldsymbol{X}$

End *% Alternating*

In NMFLAB, α_{Sx} is denoted by `AlphaS`. The number of inner iterations n_{inner} can be entered in the field `No. of inner iterations`.

2.7 Projected pseudo-inverse

This is a non-regularized version of the ALS algorithm presented in Section 1.1. The assumption $\alpha_X = \alpha_A = 0$ leads to the equations (34) and (35), however, in the *Projected pseudo-inverse* algorithm we still compute the inversions with the Moore-Penrose pseudo-inverse. This protects the algorithm against instabilities for very ill-conditioned matrices.

3 NMF-ALPHA (Group of Alpha-algorithms)

There are three large classes of generalized divergences which can be potentially useful for developing new flexible algorithms for NMF: the Bregman divergences, alpha divergence and Csiszár's φ-divergences:

S. Amari. *Differential-Geometrical Methods in Statistics*. Springer Verlag, 1985.

I. Csiszár. Information measures: A critical survey. In *Prague Conference on Information Theory, Academia Prague*, volume A, pages 73–86, 1974.

A. Cichocki, R. Zdunek, and S. Amari. Csiszar's divergences for non-negative matrix factorization: Family of new algorithms. *LNCS*, 3889:32–39, 2006.

I. Dhillon and S. Sra. Generalized nonnegative matrix approximations with Bregman divergences. In *Neural Information Proc. Systems, Vancouver Canada*, December 2005.

Csiszár's φ-divergence is defined as

$$D_C(\boldsymbol{z}||\boldsymbol{y}) = \sum_{k=1}^{N} z_k \varphi(\frac{y_k}{z_k}) \tag{53}$$

where $y_k \geq 0, z_k \geq 0$ and $\varphi:\ [0,\infty) \to (-\infty,\infty)$ is a function which is convex on $(0,\infty)$ and continuous at zero. Depending on the application, we can impose different restrictions on φ. In order to use Csiszár's divergence as a distance measure, we assume that $\varphi(1) = 0$ and that it is strictly convex at 1.
Several basic examples include ($u_{ik} = y_{ik}/z_{ik}$):
1. If $\varphi(u) = (\sqrt{u}-1)^2$, then $D_{C-H} = \sum_{ik}(\sqrt{y}_{ik} - \sqrt{z}_{ik})^2$ -Hellinger distance;
2. If $\varphi(u) = (u-1)^2$, then $D_{C-P} = \sum_{ik}(y_{ik} - z_{ik})^2/z_{ik}$ -Pearson's distance;

For $\varphi(u) = u(u^{\alpha-1}-1)/(\alpha^2-\alpha) + (1-u)/\alpha$ we have a family of alpha divergences:

$$D_A^{(\alpha)}(\boldsymbol{AX}||\boldsymbol{Y}) = \sum_{ik} y_{ik} \frac{(y_{ik}/z_{ik})^{\alpha-1}-1}{\alpha(\alpha-1)} + \frac{z_{ik}-y_{ik}}{\alpha}, \qquad z_{ik} = [\boldsymbol{AX}]_{ik}. \tag{54}$$

S. Amari. *Differential-Geometrical Methods in Statistics.* Springer Verlag, 1985.

See also Ali-Sllvey, Liese & Vajda, Cressie-Read disparity

N. A. Cressie and T.C.R. Read. *Goodness-of-Fit Statistics for Discrete Multivariate Data.* Springer, New York, 1988.

It is interesting to note that in the special cases for $\alpha = 2, 0.5, -1$, we obtain Pearson's, Hellinger's and Neyman's chi-square distances, respectively, For $\alpha = 1$ and $\alpha = 0$, the divergences have to be defined as limit points at $\alpha \to 1$ and $\alpha \to 0$, respectively. When these limits are evaluated one gets for $\alpha \to 1$ the generalized Kullback-Leibler divergence (called I-divergence) defined by equations (3) and for $\alpha \to 0$ the dual generalized KL divergence:

$$D_{KL2}(\boldsymbol{AX}||\boldsymbol{Y}) = \sum_{ik}\left([\boldsymbol{AX}]_{ik} \log \frac{[\boldsymbol{AX}]_{ik}}{y_{ik}} - [\boldsymbol{AX}]_{ik} + y_{ik}\right) \tag{55}$$

For $\alpha \neq 0$ in (54) we have developed the following new algorithm

$$x_{jk} \leftarrow x_{jk}\left(\sum_{i=1}^{m} a_{ij}\ (y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})^{\alpha}\right)^{1/\alpha}, \quad a_{ij} \leftarrow a_{ij}\left(\sum_{k=1}^{N} (y_{ik}/[\boldsymbol{A}\,\boldsymbol{X}]_{ik})^{\alpha}\, x_{jk}\right)^{1/\alpha}$$

with normalization of columns of $\boldsymbol{A}$ in each iteration to unit length: $a_{ij} \leftarrow a_{ij}/\sum_p a_{pj}$. The algorithm can be written in a compact matrix form:

$$\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \left(\boldsymbol{A}^T\left((\boldsymbol{Y}+\varepsilon) \oslash (\boldsymbol{A}\boldsymbol{X}+\varepsilon)\right)^{\alpha}\right)^{1/\alpha} \tag{56}$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} \odot \left(\left((\boldsymbol{Y}+\varepsilon) \oslash (\boldsymbol{A}\boldsymbol{X}+\varepsilon)\right)^{\alpha}\boldsymbol{X}^T\right)^{1/\alpha}, \quad \boldsymbol{A} \leftarrow \boldsymbol{A}\,\mathrm{diag}\{1 \oslash \boldsymbol{A}^T\mathbf{1}_m\},$$

where the powers mean component-wise powers. A small constant $\varepsilon = 10^{-16}$ is introduced in order to ensure non-negativity constraints and avoid possible division by zero.

The algorithms in the group of alpha-algorithms in the NMFLAB are listed in Tables 1 and 2.

In the NMFLAB, parameter α in (54) can be set in the field `Alpha` in `ADV. OPTIONS`. The relaxation parameters $\omega_X, \omega_A \in (0, 2)$ are denoted by `OmegaX` and `OmegaA`, respectively.

For completeness and in order to compare different algorithms, we have included in this group also algorithms derived from the beta divergence (Kompass algorithm). If `Generalized beta-divergence` is chosen, parameter β can be entered in the field `Beta` in `ADV. OPTIONS`. `Itakuro-Saito divergence` is a special case of the beta divergence for $\beta = -1$.

4 NMF-SMART (Group of SMART-algorithms)

The generalized SMART algorithms for NMF are proposed in

A. Cichocki, S. Amari, R. Zdunek, Z. He, and R. Kompass. Extended SMART algorithms for non-negative matrix factorization. *LNAI*, 4029, 2006.

Applying multiplicative exponentiated gradient (EG) updates:

$$x_{jk} \leftarrow x_{jk}\exp\left(-\eta_j \frac{\partial D_{KL}}{\partial x_{jk}}\right), \quad a_{ij} \leftarrow a_{ij}\exp\left(-\tilde{\eta}_j \frac{\partial D_{KL}}{\partial a_{ij}}\right), \tag{57}$$

to the cost function (55), we obtain simple multiplicative learning rules for NMF

$$x_{jk} \leftarrow x_{jk}\exp\left(\sum_{i=1}^{m}\eta_j a_{ij}\log(\frac{y_{ik}}{[\boldsymbol{A}\boldsymbol{X}]_{ik}})\right) = x_{jk}\prod_{i=1}^{m}\left(\frac{y_{ik}}{[\boldsymbol{A}\boldsymbol{X}]_{ik}}\right)^{\eta_j a_{ij}}, \tag{58}$$

$$a_{ij} \leftarrow a_{ij}\exp\left(\sum_{k=1}^{N}\tilde{\eta}_j x_{jk}\log(\frac{y_{ik}}{[\boldsymbol{A}\boldsymbol{X}]_{ik}})\right) = a_{ij}\prod_{k=1}^{N}\left(\frac{y_{ik}}{[\boldsymbol{A}\boldsymbol{X}]_{ik}}\right)^{\tilde{\eta}_j x_{jk}}, \tag{59}$$

The nonnegative learning rates $\eta_j, \tilde{\eta}_j$ can take different forms. Typically, in order to guarantee stability of the algorithm we assume that $\eta_j = \omega_A\,(\sum_{i=1}^{m} a_{ij})^{-1}$, $\tilde{\eta}_j = \omega_X\,(\sum_{k=1}^{N} x_{jk})^{-1}$, where $\omega_A, \omega_X \in (0, 2)$ is an over-relaxation parameter. The above algorithm can be considered as an alternating minimization/projection

Table 1. Amari Alpha-NMF algorithms

Amari alpha divergence: $D_A^{(\alpha)}(y_{ik}\|\|z_{ik}) = \sum_{ik} \frac{y_{ik}^{\alpha} z_{ik}^{1-\alpha} - \alpha y_{ik} + (\alpha-1) z_{ik}}{\alpha(\alpha-1)}$	
Algorithm:	$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^{m} a_{ij} \, (\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}})^{\alpha} \right)^{\frac{\omega_X}{\alpha}} \right)^{1+\alpha_{sX}}$
	$a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^{N} x_{jk} \, (\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}})^{\alpha} \right)^{\frac{\omega_A}{\alpha}} \right)^{1+\alpha_{sA}}$
	$a_{ij} \leftarrow a_{ij} / \sum_p a_{pj},$
	$0 < \omega_X < 2, \quad 0 < \omega_A < 2$
Pearson distance: $(\alpha = 2)$:	$D_A^{(\alpha=2)}(y_{ik}\|\|z_{ik}) = \sum_{ik} \frac{(y_{ik} - [\boldsymbol{A}\boldsymbol{X}]_{ik})^2}{[\boldsymbol{A}\boldsymbol{X}]_{ik}},$
Algorithm:	$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^{m} a_{ij} \, (\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}})^{2} \right)^{\frac{\omega_X}{2}} \right)^{1+\alpha_{sX}}$
	$a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^{N} x_{jk} \, (\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}})^{2} \right)^{\frac{\omega_A}{2}} \right)^{1+\alpha_{sA}}$
	$a_{ij} \leftarrow a_{ij} / \sum_p a_{pj},$
	$0 < \omega_X < 2, \quad 0 < \omega_A < 2$
Hellinger distance: $(\alpha = \frac{1}{2})$:	$D_A^{(\alpha=0.5)}(y_{ik}\|\|z_{ik}) = \sum_{ik} \frac{(y_{ik} - [\boldsymbol{A}\boldsymbol{X}]_{ik})^2}{[\boldsymbol{A}\boldsymbol{X}]_{ik}},$
Algorithm:	$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^{m} a_{ij} \sqrt{\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}}} \right)^{2\omega_X} \right)^{1+\alpha_{sX}}$
	$a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^{N} x_{jk} \sqrt{\frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}}} \right)^{2\omega_A} \right)^{1+\alpha_{sA}}$
	$a_{ij} \leftarrow a_{ij} / \sum_p a_{pj},$
	$0 < \omega_X < 2, \quad 0 < \omega_A < 2$

Table 2. Amari Alpha-NMF algorithms (continued)

Kullback-Leibler divergence: $(\alpha \to 1)$:

$$\lim_{\alpha\to 1} D_A^{(\alpha)}(y_{ik}||z_{ik}) = \sum_{ik} y_{ik} \log \frac{y_{ik}}{[\boldsymbol{AX}]_{ik}} - y_{ik} + [\boldsymbol{AX}]_{ik},$$

Algorithm:

$$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^{m} a_{ij} \frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}} \right)^{\omega_X} \right)^{1+\alpha_{sX}}$$

$$a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^{N} x_{jk} \frac{y_{ik}}{[\boldsymbol{A}\,\boldsymbol{X}]_{ik}} \right)^{\omega_A} \right)^{1+\alpha_{sA}}$$

$$a_{ij} \leftarrow a_{ij} / \textstyle\sum_p a_{pj}$$

$$0 < \omega_X < 2, \quad 0 < \omega_A < 2$$

Dual Kullback-Leibler divergence: $(\alpha \to 0)$:

$$\lim_{\alpha\to 0} D_A^{(\alpha)}(y_{ik}||z_{ik}) = \sum_{ik} [\boldsymbol{AX}]_{ik} \log \frac{[\boldsymbol{AX}]_{ik}}{y_{ik}} + y_{ik} - [\boldsymbol{AX}]_{ik}$$

Algorithm:

$$x_{jk} \leftarrow \left(x_{jk} \prod_{i=1}^{m} \left(\frac{y_{ik}}{[\boldsymbol{AX}]_{ik}} \right)^{\omega_X\, a_{ij}} \right)^{1+\alpha_{sX}}$$

$$a_{ij} \leftarrow \left(a_{ij} \prod_{k=1}^{N} \left(\frac{y_{ik}}{[\boldsymbol{AX}]_{ik}} \right)^{\tilde{\eta}_j x_{jk}} \right)^{1+\alpha_{sA}}$$

$$a_{ij} \leftarrow a_{ij} / \textstyle\sum_p a_{pj}$$

$$0 < \omega_X < 2, \quad 0 < \omega_A < 2$$

Table 3. Other generalized NMF algorithms

Beta generalized divergence:

$$D_K^{(\beta)}(y_{ik}||z_{ik}) = \sum_{ik} y_{ik} \frac{y_{ik}^{\beta-1} - [\boldsymbol{AX}]_{ik}^{\beta-1}}{\beta(\beta-1)} + [\boldsymbol{AX}]_{ik}^{\beta-1} \frac{[\boldsymbol{AX}]_{ik} - y_{ik}}{\beta}$$

Kompass algorithm:

$$x_{jk} \leftarrow x_{jk} \frac{\sum_{i=1}^{m} a_{ij}\ (y_{ik}/[\boldsymbol{AX}]_{ik}^{2-\beta})}{\sum_{i=1}^{m} a_{ij}\ [\boldsymbol{AX}]_{ik}^{\beta-1} + \varepsilon}$$

$$a_{ij} \leftarrow \left(a_{ij} \frac{\sum_{k=1}^{N} x_{jk}\ (y_{ik}/[\boldsymbol{AX}]_{ik}^{2-\beta})}{\sum_{k=1}^{N} x_{jk}\ [\boldsymbol{AX}]_{ik}^{\beta-1} + \varepsilon} \right)^{1+\alpha_{sA}}$$

$$a_{ij} \leftarrow a_{ij} / \textstyle\sum_p a_{pj},$$

Triangular discrimination:

$$D_T^{(\beta)}(y_{ik}||z_{ik}) = \sum_{ik} \frac{y_{ik}^{\beta} z_{ik}^{1-\beta} - \beta y_{ik} + (\beta-1) z_{ik}}{\beta(\beta-1)}$$

Algorithm:

$$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^{m} a_{ij} \left(\frac{2y_{ik}}{y_{ik} + [\boldsymbol{A}\,\boldsymbol{X}]_{ik}} \right)^2 \right)^{\omega_X} \right)^{1+\alpha_{sX}}$$

$$a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^{N} x_{jk} \left(\frac{2y_{ik}}{y_{ik} + [\boldsymbol{A}\,\boldsymbol{X}]_{ik}} \right)^2 \right)^{\omega_A} \right)^{1+\alpha_{sA}}$$

$$a_{ij} \leftarrow a_{ij} / \textstyle\sum_p a_{pj}, \quad 0 < \omega_X < 2, \quad 0 < \omega_A < 2$$

Itakura-Saito distance:

$$D_{IS}(y_{ik}||z_{ik}) = \sum_{ik} \frac{y_{ik}}{z_{ik}} - \log\left(\frac{y_{ik}}{z_{ik}}\right) - 1$$

Algorithm:

$$\boldsymbol{X} \leftarrow \boldsymbol{X} \odot [(\boldsymbol{A}^T \boldsymbol{P}) \oslash (\boldsymbol{A}^T \boldsymbol{Q} + \varepsilon)].^{\beta}$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} \odot [(\boldsymbol{P}\boldsymbol{X}^T) \oslash (\boldsymbol{Q}\boldsymbol{X}^T + \varepsilon)].^{\beta}$$

$$a_{ij} \leftarrow a_{ij} / \textstyle\sum_p a_{pj}, \ \ \beta = [0.5, 1]$$

$$\boldsymbol{P} = \boldsymbol{Y} \oslash (\boldsymbol{AX} + \varepsilon).^2, \ \ \boldsymbol{Q} = 1 \oslash (\boldsymbol{AX} + \varepsilon)$$

extension of the well known SMART (Simultaneous Multiplicative Algebraic Reconstruction Technique).

C. L. Byrne. Accelerating the EMML algorithm and related iterative algorithms by rescaled block-iterative (RBI) methods. *IEEE Transactions on Image Processing*, IP-7:100–109, 1998.

The above learning rules can be written (implemented in Matlab) in the compact matrix form:

$$\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \exp\Big(\boldsymbol{\eta_A}\boldsymbol{A}^T \odot \ln(\boldsymbol{Y} \oslash (\boldsymbol{A}\boldsymbol{X} + \epsilon))\Big) \tag{60}$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} \odot \exp\Big(\ln(\boldsymbol{Y} \oslash (\boldsymbol{A}\boldsymbol{X} + \epsilon))\boldsymbol{X}^T\boldsymbol{\eta}_X)\Big), \tag{61}$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A}\,\mathrm{diag}\{1 \oslash \boldsymbol{A}^T\mathbf{1}_m\}, \tag{62}$$

where in practice a small constant $\varepsilon = 10^{-16}$ is introduced in order to ensure positivity constraints and/or to avoid possible division by zero and $\boldsymbol{\eta}_A$ and $\boldsymbol{\eta}_X$ are non-negative scaling matrices representing learning rates. Typically we choose $\boldsymbol{\eta}_A = \omega_A\,\mathrm{diag}\{1 \oslash \boldsymbol{A}^T\mathbf{1}_m\}$ and $\boldsymbol{\eta}_X = \omega_X\,\mathrm{diag}\{1 \oslash \boldsymbol{X}\mathbf{1}_T\}$.

The learning algorithm (58) and (59) can be generalized to the following flexible algorithm:

$$x_{jk} \leftarrow x_{jk} \exp\left[\sum_{i=1}^{m} \eta_j\, a_{ij}\, \rho(y_{ik}, z_{ik})\right], \tag{63}$$

$$a_{ij} \leftarrow a_{ij} \exp\left[\sum_{k=1}^{N} \tilde{\eta}_j\, x_{jk}\, \rho(y_{ik}, z_{ik})\right] \tag{64}$$

where the error functions defined as

$$\rho(y_{ik}, z_{ik}) = -\frac{\partial D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})}{\partial z_{ik}} \tag{65}$$

can take different forms depending on the chosen or designed loss (cost) function $D(\boldsymbol{Y}||\boldsymbol{A}\boldsymbol{X})$ (see Table 4 and 5).

5 Second order NMF (NMF based on some second order methods)

The second order NMF algorithms are presented in

R. Zdunek and A. Cichocki. Non-negative matrix factorization with quasi-Newton optimization. *LNAI*, 4029, 2006.

Applying the quasi-Newton method to (54), we have

$$\boldsymbol{X} \leftarrow \Big[\boldsymbol{X} - [\boldsymbol{H}^{(X)}_{D_A}]^{-1}\nabla_X D_A\Big]_\epsilon, \quad \boldsymbol{A} \leftarrow \Big[\boldsymbol{A} - [\boldsymbol{H}^{(A)}_{D_A}]^{-1}\nabla_A D_A\Big]_\epsilon, \tag{66}$$

Table 4. Generalized SMART NMF adaptive algorithms and corresponding loss functions - part I.

Generalized SMART algorithms	
$a_{ij} \leftarrow a_{ij} \exp\left(\sum_{k=1}^{N} \tilde{\eta}_j \, x_{jk} \, \rho(y_{ik}, z_{ik})\right), \quad x_{jk} \leftarrow x_{jk} \exp\left(\sum_{i=1}^{m} \eta_j \, a_{ij} \, \rho(y_{ik}, z_{ik})\right),$ $a_j = \sum_{i=1}^{m} a_{ij} = 1, \ \forall j, \quad a_{ij} \geq 0, \quad y_{ik} > 0, \ z_{ik} = [\boldsymbol{A}\boldsymbol{X}]_{ik} > 0, \ x_{jk} \geq 0$	
Divergence: $D(\boldsymbol{Y}\|\|\boldsymbol{A}\,\boldsymbol{X})$	Error function: $\rho(y_{ik}, z_{ik})$
Dual Kullback-Leibler I-divergence: $D_{KL2}(\boldsymbol{A}\boldsymbol{X}\|\|\boldsymbol{Y})$	
$\sum_{ik} \left(z_{ik} \ln \frac{z_{ik}}{y_{ik}} + y_{ik} - z_{ik} \right),$	$\rho(y_{ik}, z_{ik}) = \ln\left(\frac{y_{ik}}{z_{ik}}\right),$
Relative Arithmetic-Geometric divergence: $D_{RAG}(\boldsymbol{Y}\|\|\boldsymbol{A}\boldsymbol{X})$	
$\sum_{ik} \left((y_{ik} + z_{ik}) \ln\left(\frac{y_{ik} + z_{ik}}{2y_{ik}}\right) + y_{ik} - z_{ik} \right),$	$\rho(y_{ik}, z_{ik}) = \ln\left(\frac{2y_{ik}}{y_{ik} + z_{ik}}\right),$
Symmetric Arithmetic-Geometric divergence: $D_{SAG}(\boldsymbol{Y}\|\|\boldsymbol{A}\boldsymbol{X})$	
$2\sum_{ik} \left(\frac{y_{ik} + z_{ik}}{2} \ln\left(\frac{y_{ik} + z_{ik}}{2\sqrt{y_{ik} z_{ik}}}\right) \right),$	$\rho(y_{ik}, z_{ik}) = \frac{y_{ik} - z_{ik}}{2z_{ik}} + \ln\left(\frac{2\sqrt{y_{ik} z_{ik}}}{y_{ik} + z_{ik}}\right),$
J-divergence: $D_J(\boldsymbol{Y}\|\|\boldsymbol{A}\boldsymbol{X})$	
$\sum_{ik} \left(\frac{y_{ik} - z_{ik}}{2} \ln\left(\frac{y_{ik}}{z_{ik}}\right) \right),$	$\rho(y_{ik}, z_{ik}) = \frac{1}{2} \ln\left(\frac{y_{ik}}{z_{ik}}\right) + \frac{y_{ik} - z_{ik}}{2z_{ik}},$

Table 5. Generalized SMART NMF adaptive algorithms and corresponding loss functions - part II.

Relative Jensen-Shannon divergence: $D_{RJS}(\boldsymbol{Y}\|\|\boldsymbol{AX})$	
$\sum_{ik}\left(2y_{ik}\ln\left(\frac{2y_{ik}}{y_{ik}+z_{ik}}\right)+z_{ik}-y_{ik}\right),$	$\rho(y_{ik},z_{ik})=\frac{y_{ik}-z_{ik}}{2z_{ik}}+\ln\left(\frac{2\sqrt{y_{ik}z_{ik}}}{y_{ik}+z_{ik}}\right),$
Dual Jensen-Shannon divergence: $D_{DJS}(\boldsymbol{Y}\|\|\boldsymbol{AX})$	
$\sum_{ik} y_{ik}\ln\left(\frac{2z_{ik}}{z_{ik}+y_{ik}}\right)+y_{ik}\ln\left(\frac{2y_{ik}}{z_{ik}+y_{ik}}\right),$	$\rho(y_{ik},z_{ik})=\ln\left(\frac{z_{ik}+y_{ik}}{2y_{ik}}\right),$
Symmetric Jensen-Shannon divergence: $D_{SJS}(\boldsymbol{Y}\|\|\boldsymbol{AX})$	
$\sum_{ik} y_{ik}\ln\left(\frac{2y_{ik}}{y_{ik}+z_{ik}}\right)+z_{ik}\ln\left(\frac{2z_{ik}}{y_{ik}+z_{ik}}\right),$	$\rho(y_{ik},z_{ik})=\ln\left(\frac{y_{ik}+z_{ik}}{2z_{ik}}\right),$
Triangular discrimination: $D_T(\boldsymbol{Y}\|\|\boldsymbol{AX})$	
$\sum_{ik}\left\{\frac{(y_{ik}-z_{ik})^2}{y_{ik}+z_{ik}}\right\},$	$\rho(y_{ik},z_{ik})=\left(\frac{2y_{ik}}{y_{ik}+z_{ik}}\right)^2-1,$
Bose-Einstein divergence: $D_{BE}(\boldsymbol{Y}\|\|\boldsymbol{AX})$	
$\sum_{ik} y_{ik}\ln\left(\frac{(1+\alpha)y_{ik}}{y_{ik}+\alpha z_{ik}}\right)+\alpha z_{ik}\ln\left(\frac{(1+\alpha)z_{ik}}{y_{ik}+\alpha z_{ik}}\right),$	$\rho(y_{ik},z_{ik})=\alpha\ln\left(\frac{y_{ik}+\alpha z_{ik}}{(1+\alpha)z_{ik}}\right),$

where $\boldsymbol{H}_{D_A}^{(X)}$ and $\boldsymbol{H}_{D_A}^{(A)}$ are Hessian matrices for (54) with respect to $\boldsymbol{X}$ and $\boldsymbol{A}$, respectively. The nonlinear operator $[\cdot]_\epsilon = \max\{\cdot, \epsilon\}$ enforces nonnegativity.

Thus for $\boldsymbol{X}$ we have

$$\boldsymbol{G}_{D_A}^{(X)} = \nabla_X D_A = \frac{1}{\alpha}\boldsymbol{A}^T \left(\boldsymbol{1} - (\boldsymbol{Y} \oslash (\boldsymbol{AX}))^\alpha\right) \in \mathbb{R}^{R\times K}. \tag{67}$$

The Hessian has the form:

$$\boldsymbol{H}_{D_A}^{(X)} = \frac{1}{\alpha}\,\mathrm{diag}\{[\boldsymbol{h}_k^{(X)}]_{k=1,\ldots,K}\} \in \mathbb{R}^{RK\times RK} \tag{68}$$

where

$$\boldsymbol{h}_k^{(X)} = \boldsymbol{A}^T\,\mathrm{diag}\{[\boldsymbol{Y}^\alpha \oslash (\boldsymbol{AX})^{\alpha+1}]_{*,k}\}\boldsymbol{A} \in \mathbb{R}^{R\times R}$$

For $\boldsymbol{A}$, we get

$$\boldsymbol{G}_{D_A}^{(A)} = \nabla_A D_A = \frac{1}{\alpha}\left(\boldsymbol{1} - (\boldsymbol{Y} \oslash (\boldsymbol{AX}))^\alpha\right)\boldsymbol{X}^T \in \mathbb{R}^{M\times R}. \tag{69}$$

The Hessian has the form:

$$\boldsymbol{H}_{D_A}^{(A)} = \frac{1}{\alpha}\,\mathrm{diag}\{[\boldsymbol{h}_m^{(A)}]_{m=1,\ldots,M}\} \in \mathbb{R}^{MR\times MR} \tag{70}$$

where

$$\boldsymbol{h}_m^{(A)} = \boldsymbol{X}\,\mathrm{diag}\{[\boldsymbol{Y}^\alpha \oslash (\boldsymbol{AX})^{\alpha+1}]_{m,*}\}\boldsymbol{X}^T \in \mathbb{R}^{R\times R}$$

Since the Hessian is usually ill-conditioned, especially if we have sparse representations of the image to be estimated, some regularization of the Hessian is essential, which leads to quasi-Newton iterations. We applied the Levenberg-Marquardt approach with a small regularization parameter $\lambda = 10^{-12}$. Additionally, we control the convergence by a slight relaxation of the iterative updates. To reduce substantially a computational cost, the inversion of the Hessian is replaced with the Q-less QR factorization computed with LAPACK. Thus the final form of the algorithm is

$$\boldsymbol{X} \leftarrow [\boldsymbol{X} - \gamma \boldsymbol{R}_X \backslash \boldsymbol{W}_X]_\epsilon\,, \qquad \boldsymbol{A} \leftarrow [\boldsymbol{A} - \gamma \boldsymbol{R}_A \backslash \boldsymbol{W}_A]_\epsilon\,, \tag{71}$$

$$\boldsymbol{W}_X = \boldsymbol{Q}_X^T \nabla_X D_A, \qquad \boldsymbol{Q}_X \boldsymbol{R}_X = \boldsymbol{H}_{D_A}^{(X)} + \lambda \boldsymbol{I}_X,$$

$$\boldsymbol{W}_A = \boldsymbol{Q}_A^T \nabla_A D_A, \qquad \boldsymbol{Q}_A \boldsymbol{R}_A = \boldsymbol{H}_{D_A}^{(A)} + \lambda \boldsymbol{I}_A,$$

where $\boldsymbol{I}_X \in \mathbb{R}^{RK\times RK}$, $\boldsymbol{I}_A \in \mathbb{R}^{MR\times MR}$ are identity matrices, $\boldsymbol{R}_X$ and $\boldsymbol{R}_A$ are upper triangular matrices, and γ controls the relaxation. We set $\gamma = 0.9$. The $\backslash$ in (71) means Gaussian elimination.

For $\alpha \to 0$, we get the dual KL, i.e.

$$\begin{aligned} D_{KL2}(\boldsymbol{Y}||\boldsymbol{AX}) &= \lim_{\alpha\to 0} D_A(\boldsymbol{AX}||\boldsymbol{Y}) \\ &= \sum_{mk}\left(z_{mk}\log\frac{z_{mk}}{y_{mk}} + y_{mk} - z_{mk}\right), \quad z_{mk} = [\boldsymbol{AX}]_{mk}, \end{aligned} \tag{72}$$

and consequently the gradient and Hessian are as follows

$$\boldsymbol{G}_{D_{KL2}}^{(X)} = \nabla_X D_{KL2} = \boldsymbol{A}^T \log\left((\boldsymbol{A}\boldsymbol{X}) \oslash \boldsymbol{Y}\right) \in \mathbb{R}^{R\times K}, \tag{73}$$

and

$$\boldsymbol{H}_{D_{KL2}}^{(X)} = \mathrm{diag}\{[\boldsymbol{h}_k^{(X)}]_{k=1,\ldots,K}\} \in \mathbb{R}^{RK\times RK}, \tag{74}$$

where

$$\boldsymbol{h}_k^{(X)} = \boldsymbol{A}^T \mathrm{diag}\{[1 \oslash (\boldsymbol{A}\boldsymbol{X})]_{*,k}\}\boldsymbol{A} \in \mathbb{R}^{R\times R}$$

For $\boldsymbol{A}$, we have

$$\boldsymbol{G}_{D_{KL2}}^{(A)} = \nabla_A D_{KL2} = \log\left((\boldsymbol{A}\boldsymbol{X}) \oslash \boldsymbol{Y}\right) \boldsymbol{X}^T \in \mathbb{R}^{M\times R}. \tag{75}$$

The Hessian has the form:

$$\boldsymbol{H}_{D_{KL2}}^{(A)} = \mathrm{diag}\{[\boldsymbol{h}_m^{(A)}]_{m=1,\ldots,M}\} \in \mathbb{R}^{MR\times MR}, \tag{76}$$

where

$$\boldsymbol{h}_m^{(A)} = \boldsymbol{X}\, \mathrm{diag}\{[1./(\boldsymbol{A}\boldsymbol{X})]_{m,*}\}\boldsymbol{X}^T \in \mathbb{R}^{R\times R}.$$

Since the Newton method does not ensure nonnegative approximations, the application of Newton-like optimization with such a nonlinear operation as in (66) to both sets of the arguments ($\boldsymbol{X}$ and $\boldsymbol{A}$) does not give a satisfactory solution. In BSS application, $\boldsymbol{A}$ has much smaller dimension than $\boldsymbol{X}$, i.e. $T >> m \geq r$, and hence, the approximations of $\boldsymbol{X}$ can be calculated, e.g., with (15). Because a convergence rate for the Newton method is much higher than for the ISRA, in each alternating step, $\boldsymbol{X}$ is calculated within a few iterations. Also, in each alternating step the columns of $\boldsymbol{A}$ are normalized to a unity.

To enforce sparsity of estimated variables we applied the nonlinear projection as in (27) for computation of $\boldsymbol{X}$. The degree of the sparsity can be controlled by α_{Sx}. In the NMFLAB, this parameter is denoted as `AlphaS`, but parameter α in (54) can be given in the field `Alpha`.

The Matlab code for our implementation of the quasi-Newton methods in the NMFLAB is as followed:

```
switch type_alg_X

      case 1 % Kullback-Leibler (EMML)

          for t = 1:no_iter
              X = (X.*(A'*(Y./(A*X + eps)))).^alphaS;
          end
          X = X + 100*eps;

      case 2 % Frobenius (ISRA)
```

```
     for t = 1:no_iter
         X = X.*((A'*Y)./(A'*A*X + eps));
     end
     X = X + 100*eps;

case 3 % Newton applied to Frobenius

    hX = A'*A;
    GX = A'*Y - hX*X; % Gradient
    for t = 1:T
        HX(((t-1)*r+1):t*r,((t-1)*r+1):t*r) = -hX; % Hessian
    end
    hk = 0;
    alpha_h = alpha_h_init;
    while condest(HX) > 1E7
          hk = hk + 1;
          alpha_h = 10*alpha_h;
          HX = HX + alpha_h*speye(r*T);
    end
    [QX,RX] = qr(HX,GX(:));
    X(:) = X(:) - .9*RX\QX;
    X(X <= 0) = 100*eps;

case 4 % Newton applied to KL

    Zx = A*X+1E5*eps;
    GX = A'*(E - Y./Zx);  % Gradient
    Zxx = (Y+100*eps)./Zx.^2;
    for t = 1:T
        HX(((t-1)*r+1):t*r,((t-1)*r+1):t*r) =...
        A'*diag(Zxx(:,t))*A;  % Hessian
    end
    hk = 0;
    alpha_h = alpha_h_init;
    while condest(HX) > 1E7
          hk = hk + 1;
          alpha_h = 10*alpha_h;
          HX = HX + alpha_h*speye(r*T);
    end
    [QX,RX] = qr(HX,GX(:));
    X(:) = X(:) + .9*RX\QX;
    X(X <= 0) = 100*eps;

case 5 % Newton applied to dual KL (KL2)
```

```
        Zx = A*X+10*eps;
        Zxx = 1./Zx + eps;
        GX = A'*log(Zx./(Y + 10*eps)); % Gradient
        for t = 1:T
            HX(((t-1)*r+1):t*r,((t-1)*r+1):t*r) = ...
            A'*diag(Zxx(:,t))*A; % Hessian
        end
        hk = 0;
        alpha_h = alpha_h_init;
        while condest(HX) > 1E7
              hk = hk + 1;
              alpha_h = 10*alpha_h;
              HX = HX + alpha_h*speye(r*T);
        end
        [QX,RX] = qr(HX,GX(:));
        X(:) = X(:) - .9*RX\QX;
        X(X <= 0) = 100*eps;

      case 6 % Fixed X

        X = s + eps;
        niter_selected = 1000;     % maximum number of iterations
                                   % for the selected sample (can be adjusted)

end % switch for X

switch type_alg_A


    case 1 % Newton applied to Frobenius

        Ap = A;
        hA = X*X';
        GA = Y*X' - A*hA;  % Gradient
        for i = 1:m
            HA(((i-1)*r+1):i*r,((i-1)*r+1):i*r) = -hA; % Hessian
        end
        GA = GA';
        hk = 0;
        alpha_h = alpha_h_init;
        while condest(HA) > 1E7
              hk = hk + 1;
              alpha_h = 10*alpha_h;
              HA = HA + alpha_h*speye(r*m);
```

```
    end
    [QA,R] = qr(HA,GA(:));
    A = A';
    A(:) = A(:) - .9*R\QA;
    A(A <= 0) = 100*eps;
    A = A';
    A = A*diag(1./sum(A,1));

case 2 % Newton applied to KL

    Ap = A;
    Zx = A*X+100*eps;
    Zxx = (Y+eps)./Zx.^2;
    for i = 1:m
        HA(((i-1)*r+1):i*r,((i-1)*r+1):i*r) = ...
        (X.*repmat(Zxx(i,:),r,1))*X'; % Hessian
    end

    GA = X*(E - (Y+eps)./Zx)'; % Gradient

    hk = 0;
    alpha_h = alpha_h_init;
    while condest(HA) > 1E7
          hk = hk + 1;
          alpha_h = 10*alpha_h;
          HA = HA + alpha_h*speye(r*m);
    end
    A = A';
    [QA,R] = qr(HA,GA(:));
    A(:) = A(:) - .9*R\QA;
    A(A < 0) = 100*eps;
    A = A';
    A = A*diag(1./sum(A,1));

case 3 % Newton applied to dual KL (KL2)

    Zx = A*X+10*eps;
    Zxx = 1./Zx;

    GA = X*(log(Zx./(Y + eps)))'; % Gradient

    for i = 1:m
        HA(((i-1)*r+1):i*r,((i-1)*r+1):i*r) = ...
        (X.*repmat(Zxx(i,:),r,1))*X';  % Hessian
    end
```

```
    hk = 0;
    alpha_h = alpha_h_init;
    while condest(HA) > 1E7
          hk = hk + 1;
          alpha_h = 10*alpha_h;
          HA = HA + alpha_h*speye(r*m);
    end

    A = A';
    [QA,R] = qr(HA,GA(:));
    A(:) = A(:) - .9*R\QA;
    A(A < 0) = 100*eps;
    A = A';
    A = A*diag(1./sum(A,1));

case 4 % Newton applied to Amari divergence

    Ap = A;
    Zx = A*X+100*eps;
    Zxx = ((Y+eps).^beta)./(Zx.^(beta + 1));
    for i = 1:m
        HA(((i-1)*r+1):i*r,((i-1)*r+1):i*r) = ...
        (X.*repmat(Zxx(i,:),r,1))*X'; % Hessian
    end

    GA = (1/beta)*X*(E - ((Y+eps)./Zx).^beta)'; % Gradient

    hk = 0;
    alpha_h = alpha_h_init;
    while condest(HA) > 1E7
          hk = hk + 1;
          alpha_h = 10*alpha_h;
          HA = HA + alpha_h*speye(r*m);
    end
    A = A';
    [QA,R] = qr(HA,GA(:));
    A(:) = A(:) - .9*R\QA;
    A(A < 0) = 100*eps;
    A = A';
    A = A*diag(1./sum(A,1));

case 5 % Fixed A

    niter_selected = 1000;     % maximum number of iterations
```

```
                                        % for the selected sample (can be adjusted)
        A = A_true + eps;
        A = A*diag(1./(sum(A,1) + eps));

end % switch for A
```

6 GNMF-Cascade (Cascade regularized NMF)

In the generalized NMF (cascade or multi-layer NMF), we approximate a given (observed or sensor) data array $\boldsymbol{Y}$ with positive entries by a product of several (generally more than two) non-negative matrices, i.e., $\boldsymbol{Y} \approx \boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_{L+1}$, where the number of columns of basis (mixing) matrices $\boldsymbol{A}_p$ is typically much smaller than that of $\boldsymbol{Y}$ and rows of matrix $\boldsymbol{A}_{L+1} = \boldsymbol{X}$ represent usually latent (hidden) components or sources (depending on applications).

In other words, the generalized non-negative matrix factorization (GNMF) decomposes the data matrix $\boldsymbol{Y} = [\boldsymbol{y}(1), \boldsymbol{y}(2), \ldots, \boldsymbol{y}(N)] \in \mathbb{R}^{m \times N}$ as a product of several matrices $\boldsymbol{A}_p$. i.e.,

$$\boldsymbol{Y} = \boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_{L+1} + \boldsymbol{V}, \qquad \boldsymbol{A}_p \geq 0, \ \ \forall p, \ p = 1, 2, \ldots, L+1 \tag{77}$$

where $\boldsymbol{V} \in \mathbb{R}^{m \times N}$ represents a noise or error matrix (depending on applications), $\boldsymbol{A}_p \in \mathbb{R}^{m \times m}$, for $p = 1, 2, \ldots, L-1$, $\boldsymbol{A}_L \in \mathbb{R}^{m \times n}$ and $\boldsymbol{A}_{L+1} = \boldsymbol{X} \in \mathbb{R}^{n \times N}$ usually represents source signals and matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_L$ denote the basis or sparse mixing matrices connected in cascade. The objective is to estimate all mixing (basis) matrices $\boldsymbol{A}_p$ for $(p = 1, 2, \ldots, L)$ or global mixing matrix $\boldsymbol{A} = \boldsymbol{A}_1 \cdots \boldsymbol{A}_L$ and sources $\boldsymbol{A}_{L+1} = \boldsymbol{X}$ subject to nonnegativity constraints for all entries.

In the NMFLAB, we used the following notations: $x_j(k) = x_{jk} = [\boldsymbol{X}]_{jk}$, $y_i(k) = y_{ik} = [\boldsymbol{Y}]_{ik}$ and $z_{ik} = [\boldsymbol{Z}]_{ik}$ means ik-element of the matrix $\boldsymbol{Z} = \boldsymbol{A}_1 \cdots \boldsymbol{A}_{L+1}$. In order to simplify update formulas for GNMF, we have introduced the additional matrix notations:

$$\boldsymbol{Z} = \boldsymbol{A}_1 \boldsymbol{A}_2 \cdots \boldsymbol{A}_{L+1} \tag{78}$$

$$\overline{\boldsymbol{A}}_p = \boldsymbol{A}_1 \cdots \boldsymbol{A}_{p-1}, \qquad \overline{\boldsymbol{A}}_1 = \boldsymbol{I}_{m \times m} \tag{79}$$

$$\overline{\boldsymbol{X}}_p = \boldsymbol{A}_{p+1} \cdots \boldsymbol{A}_{L+1}, \qquad \overline{\boldsymbol{X}}_{L+1} = \boldsymbol{I}_{N \times N}, \quad p = 1, 2, \ldots, L+1. \tag{80}$$

.

The basic approach to GNMF is similar to standard NMF by applying alternating minimization or alternating projection: once specified by the user a loss function is alternately minimized with respective to several sets of the parameters $\boldsymbol{A}_p$, $(p = 1, 2, \ldots L+1)$ each time optimizing one set of arguments while keeping the other one fixed.

In the NMFLAB we implemented several loss functions. The basic approach is as follows. Let us consider the following constrained optimization problem:

Minimize:

$$D_{F\alpha}(\boldsymbol{A}_p) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}_1 \cdots \boldsymbol{A}_{L+1}\|_F^2 + \sum_p \alpha_{A_p} \boldsymbol{A}_p \tag{81}$$

subject to non-negativity constraints applied to all matrices $\boldsymbol{A}_p$, where $\alpha_{A_p} \geq 0$ are nonnegative regularization parameters. The terms $\alpha_{A_p} \boldsymbol{A}_p$ are used to enforce or control a sparse solution.

Using a gradient descent approach, we have derived the following basic learning rules for GNMF:

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot \left[\overline{\boldsymbol{A}}_p^T \boldsymbol{Y}\, \overline{\boldsymbol{X}}_p^T - \alpha_{A_p}\right]_{+\varepsilon} \oslash \left[\overline{\boldsymbol{A}}_p^T \boldsymbol{Z}\, \overline{\boldsymbol{X}}_p^T + \varepsilon\right]. \tag{82}$$

In the special case, for $L = 1$ the above algorithm simplifies to the standard ISRA or Lee-Seung algorithm. For $L = 2$ it takes the form:

$$\boldsymbol{A}_1 \leftarrow \boldsymbol{A}_1 \odot \left[\boldsymbol{Y}\,(\boldsymbol{A}_2\boldsymbol{A}_3)^T - \alpha_{A_1}\right]_{+\varepsilon} \oslash \left[\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3(\boldsymbol{A}_2\boldsymbol{A}_3)^T + \varepsilon\right] \tag{83}$$

$$\boldsymbol{A}_2 \leftarrow \boldsymbol{A}_2 \odot \left[\boldsymbol{A}_1^T \boldsymbol{Y}\, \boldsymbol{A}_3^T - \alpha_{A_2}\right]_{+\varepsilon} \oslash \left[\boldsymbol{A}_1^T\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3\boldsymbol{A}_3^T + \varepsilon\right] \tag{84}$$

$$\boldsymbol{X} = \boldsymbol{A}_3 \leftarrow \boldsymbol{A}_3 \odot \left[(\boldsymbol{A}_1\boldsymbol{A}_2)^T \boldsymbol{Y} - \alpha_{A_3}\right]_{+\varepsilon} \oslash \left[(\boldsymbol{A}_1\boldsymbol{A}_2)^T\boldsymbol{A}_1\boldsymbol{A}_2\boldsymbol{A}_3 + \varepsilon\right]. \tag{85}$$

The above formula can be generalized as

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot \frac{\left[\overline{\boldsymbol{A}}_p^T [\boldsymbol{Y} \odot F(\boldsymbol{Z})]\, \overline{\boldsymbol{X}}_p^T - \alpha_{A_p}\right]_{+\varepsilon}}{\left[\overline{\boldsymbol{A}}_p^T [\boldsymbol{Z} \odot F(\boldsymbol{Z})]\, \overline{\boldsymbol{X}}_p^T + \varepsilon\right]}, \tag{86}$$

where the nonlinear function $F(\boldsymbol{Z})$ can take different forms depending on the loss function. For example, for the Kullback Leibler divergence we have $F(Z) = 1/Z$, and it simplifies to

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot \left[\overline{\boldsymbol{A}}_p^T [\boldsymbol{Y} \oslash \boldsymbol{Z}]\, \overline{\boldsymbol{X}}_p^T - \alpha_{A_p}\right]_{+\varepsilon} \oslash \left[\overline{\boldsymbol{A}}_p^T [\boldsymbol{1}]\, \overline{\boldsymbol{X}}_p^T + \varepsilon\right], \tag{87}$$

Alternatively, we can use the following learning rule:

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot \left[\overline{\boldsymbol{A}}_p^T [\boldsymbol{Y} \oslash \boldsymbol{Z}]\, \overline{\boldsymbol{X}}_p^T - \alpha_{A_p}\right]_{+\varepsilon} \tag{88}$$

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot (1 \oslash \boldsymbol{A}_p^T \boldsymbol{1}_m), \tag{89}$$

The regularized algorithm for α divergence:

$$\boldsymbol{A}_p \leftarrow \left(\boldsymbol{A}_p \odot \left(\overline{\boldsymbol{A}}_p^T \left([\boldsymbol{Y} + \varepsilon] \oslash [\boldsymbol{Z} + \varepsilon]\right).^{\alpha} * \overline{\boldsymbol{X}}_p^T \oslash [\boldsymbol{A}_p^T \boldsymbol{1}_n \boldsymbol{1}_N^T]\right).^{1/\alpha}\right)^{1+\alpha_{sA}} \tag{90}$$

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p * \mathrm{diag}\{1 \oslash \boldsymbol{A}_p^T \boldsymbol{1}_m\}.$$

The generalized SMART algorithm for GNMF can take the form:

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p \odot \exp\left(\eta_{\overline{A}P} * \overline{\boldsymbol{A}}_p^T * \ln(\boldsymbol{Y} \oslash [\boldsymbol{Z} + \epsilon]) * \overline{\boldsymbol{X}}_p^T * \boldsymbol{\eta}_{\overline{X}_p})\right), \tag{91}$$

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p * \mathrm{diag}\{1 \oslash \boldsymbol{A}_p^T \boldsymbol{1}_m\}, \tag{92}$$

which can be further generalized to the form (see Table 4 and 5):

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p^T \odot \exp\left(\eta_{\overline{A}_P} * \overline{\boldsymbol{A}}_p * \rho(\boldsymbol{Y}, \boldsymbol{Z}) * \overline{\boldsymbol{X}}_p^T * \boldsymbol{\eta}_{\overline{X}_p})\right), \tag{93}$$

$$\boldsymbol{A}_p \leftarrow \boldsymbol{A}_p * \mathrm{diag}\{1 \oslash \boldsymbol{A}_p^T \mathbf{1}_m\}, \tag{94}$$

For example, for the symmetric Jensen-Shannon divergence:

$$\rho(\boldsymbol{Y}, \boldsymbol{Z}) = \ln\left([\boldsymbol{Y} + \boldsymbol{Z}] \oslash [2\boldsymbol{Z} + \varepsilon]\right) \tag{95}$$

and for α divergence (see Table 1):

$$\rho(\boldsymbol{Y}, \boldsymbol{Z}) = \frac{1}{\alpha}\left[([\boldsymbol{Y} + \varepsilon] \oslash [\boldsymbol{Z} + \varepsilon])^{\alpha} - 1\right] \tag{96}$$

The main motivation for implementation of the GNMF is to impose some additional constraints on factorizing matrices such as sparseness and smoothness or to find localized part-based representations of nonnegative multivariate data.

For example, if we apply the following model;

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{S}\boldsymbol{X} \tag{97}$$

where the smoothing matrix $\boldsymbol{S}$ is defined as:

$$\boldsymbol{S} = (1 - \Theta)\boldsymbol{I} + \frac{\Theta}{n}\mathbf{1}\mathbf{1}^T \tag{98}$$

with Θ a parameter between zero and 1, we have so called non-smooth NMF. If $\Theta = 0$ then we have the standard NMF model and no smoothing on the matrix $\boldsymbol{X}$ occurs. However, as Θ tends to 1 the vector $\widetilde{\boldsymbol{x}}(k) = \boldsymbol{S}\boldsymbol{x}(k)$ tends to a constant vector with entries almost equal to the average of the elements of $\boldsymbol{x}(k)$. This is the smoothest possible vector and also non-sparse because all entries are equal to the same nonzero value. The matrix $\boldsymbol{S}$ performs some smoothness (or non-sparseness) operation on the columns of the matrix $\boldsymbol{X}$. However, due to the multiplicative nature of this model, strong smoothing in $\boldsymbol{S}$ will force strong sparseness in both the basis matrix and encoding vectors $\boldsymbol{X}(k)$ in order to maintain faithfulness of the model to the data $\boldsymbol{Y}$. Please see for details

A. Pascual-Montano, J. M. Carazo, K. Kochi, D. Lehmean, and R. Pacual-Marqui. Nonsmooth nonnegative matrix factorization (nsNMF). *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(3):403–415, 2006

Similar local smoothing operations can be performed for rows of the matrix $\boldsymbol{X}$ by employing the special form of the GNMF as follows

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X}\boldsymbol{M} \tag{99}$$

where the smoothing matrix $\boldsymbol{M}$ is defined as follows

$$\boldsymbol{M} = \frac{1}{3}\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ \dots & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \dots \\ \dots & 0 & 1 & 1 & 1 & 0 & 0 & 0 & \dots \\ \dots & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \dots \\ \dots & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \dots \\ \dots & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \dots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \tag{100}$$

In order to compare the NMF algorithms with the conventional ICA approach we added to NMFLAB standard BSS algorithms: AMUSE, SOBI and ThinICA.

7 AMUSE - Algorithm for Multiple Unknown Source Extraction based on EVD

AMUSE - Algorithm for Multiple Unknown Source Extraction is based on the EVD of a single time-delayed covariance matrix for prewhitened data. This algorithm was originally proposed by

L. Tong, V. Soon, Y. F. Huang, and R. Liu, Indeterminacy and identifiability of blind identification, IEEE Trans. CAS, vol. 38, pp. 499-509, March 1991.

L. Tong, Inouye, and R. Liu, Waveform-preserving blind estimation of multiple independent sources, IEEE Trans. on Signal Processing, 41 (7), pp. 2461-2470, July 1993.

It was implemented in modified form in:

R. Szupiluk, A. Cichocki, Blind signal separation using second order statistics, Proc. of SPETO 2001, pp. 485-488.

This algorithm uses the sub-optimal time delay p = 1. It is relatively fast but sensitive to noise. The algorithm belongs to the class of second order statistics (SOS) methods since it uses only the time-delayed covariance matrix. It performs blind source separation of colored sources (i.e., the signals with temporal structure) and uses the standard prewhitening technique. In this implementation, it has been found that the algorithm is very sensitive to additive noise, when the number of sources is equal to the number of sensors. In order to reduce dimension and to improve the performance for noisy data you may wish to modify the algorithm by using a different time delay or by adopting a robust prewhitening method.

8 SOBI - Second Order Blind Identification

SOBI (Second Order Blind Identification) was originally developed by Adel Belouchrani et al. A similar algorithm has been proposed independently by Molgedey and Schuster. In `ADV. OPTIONS`, users can select the number of time-delayed covariance matrices to be jointly diagonalized (default number of the matrices is K = 4) and the time-delays (default is p = 1, 2, 3, 4). The SOBI algorithm has been presented and analyzed in the following papers:

A. Belouchrani, K. Abed-Meraim, J.F. Cardoso, and E. Moulines, Second-order blind separation of temporally correlated sources, in Proc. Int. Conf. on Digital Sig. Proc., (Cyprus), pp. 346-351, 1993.

A. Belouchrani and K. Abed-Meraim, Separation aveugle au second ordre de sources correlees, in Proc. Gretsi, (Juan-les-pins), pp. 309-312, 1993. (in French)

A. Belouchrani, K. Abed-Meraim, J.F. Cardoso and E. Moulines, A blind source separation technique using second order statistics, IEEE Trans. on Signal Processing, vole. 45, No. 2, pp. 434-444, February 1997.

L. Molgedey and G. Schuster, Separation of a mixture of independent signals using time delayed correlations, Physical Review Letters, vole. 72, No. 23, pp. 3634-3637, 1994.

A. Ziehe, K.-R. Mueller, TDSEP - an efficient algorithm for blind separation using time structure, ICANN'98, 675-680, Skovde 1998.

A. Belouchrani, and A. Cichocki, Robust whitening procedure in blind source separation context, Electronics Letters, vol. 36, No. 24, 2000, pp. 2050-2053.

S. Choi, A. Cichocki and A. Belouchrani, Blind separation of nonstationary sources in noisy mixtures, Journal of VLSI Signal Processing 2002.

A. Cichocki and A. Belouchrani, Sources separation of temporally correlated sources from noisy data using a bank of band-pass filters, in Proc. of Third International Conference on Independent Component Analysis and Signal Separation (ICA-2001), pp. 173-178, San Diego, USA, Dec. 9-13, 2001.

A. Cichocki, T Rutkowski and K Siwek, Blind extraction of signals with specified frequency bands, Neural Networks for Signal Processing (NNSP-2002), Martigny, Switzerland, September 2002.

R.R. Gharieb and A. Cichocki, Second order statistics based blind signal separation using a bank of subband filters, Journal of Digital Signal Processing, 2003.

9 ThinICA Algorithm for Independent Component Analysis

(ThinICA) has been developed by Sergio Cruces Alvarez and Andrzej Cichocki and in modified form by Cruces-Alavarez, Cichocki and De Lathauwer (see ICALAB for more details). The ThinICA algorithm is able to extract simultaneously arbitrary number of components specified by the user. The algorithm is based on criteria that jointly perform maximization of several cumulants of the outputs and/or second order time delay covariance matrices. This employed contrast function combines the robustness of the joint approximate diagonalization techniques with the flexibility of the methods for blind signal extraction. Its maximization leads to hierarchical and simultaneous ICA extraction algorithms which are respectively based on the thin QR and thin SVD factorizations. The ThinICA algorithm can be regarded as hierarchical/simultaneous extensions of the fast fixed point algorithms. The implemented version of ThinICA called ThinICAP forces that all entries of estimated matrix $\boldsymbol{A}$ are non-negative. The main references are:

S. A. Cruces-Alvarez and A. Cichocki, Combining blind source extraction with joint approximate diagonalization: Thin Algorithms for ICA, Proc. of the Fourth Symposium on Independent Component Analysis and Blind Signal Separation, Japan, pp. 463-469, 2003.

S. A. Cruces-Alvarez, A. Cichocki, and L. De Lathauwer, Thin QR and SVD factorizations for simultaneous blind signal extraction, in Proceedings of the European Signal Processing Conference (EUSIPCO), (Vienna, Austria), pp. 217-220, 2004.