

PART 2

Algorithms for Non-Negative Matrix Factorization

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

$$\mathbf{Y} = \mathbf{AX} + \mathbf{V}, \quad \mathbf{A} \geq 0, \quad \mathbf{X} \geq 0 \quad (\text{component-wise}) \quad (1)$$

or equivalently $\mathbf{y}(k) = \mathbf{Ax}(k) + \mathbf{v}(k)$, $k = 1, 2, \dots, T$, or in a scalar form as $y_i(k) = \sum_{j=1}^r a_{ij}x_j(k) + v_i(k)$, $i = 1, \dots, m$, where $\mathbf{V} \in \mathbb{R}^{m \times T}$ represents a noise or error matrix, $\mathbf{y}(k) = [y_1(k), \dots, y_m(k)]^T$ is a vector of the observed components or signals for k -th sample or at the discrete time instants k , and $\mathbf{x}(k) = [x_1(k), \dots, x_r(k)]^T$ 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 $D(\mathbf{Y}||\mathbf{AX})$ 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 (\mathbf{A} and \mathbf{X}), we usually employ constrained alternating minimization, i.e., in one step \mathbf{A} is estimated and \mathbf{X} fixed, and in the other step we fix \mathbf{A} and estimate \mathbf{X} . Typically, the estimate of \mathbf{A} depends on both estimates \mathbf{A} and \mathbf{X} . Thus the iteration which updates \mathbf{A} may be performed several times before proceeding to the \mathbf{X} update. A similar choice is available when performing the \mathbf{X} update. Thus switching between computation of \mathbf{A} and \mathbf{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: EMLL, ISRA, Kompass algorithm, and projected pseudo-inverse. In their original incarnation, the EMLL and ISRA algorithms assumed that the basis (mixing) matrix \mathbf{A} was known and only the nonnegative matrix \mathbf{X} was unknown. However, EMLL and ISRA can be applied assuming the opposite is true. The EMLL and ISRA algorithms differ in how the updates are performed; EMLL 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 EMLL 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 \mathbf{A} and \mathbf{X} by applying the well-known technique of switching between updating \mathbf{A} with \mathbf{X} known and updating \mathbf{X} with \mathbf{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:

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

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

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

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

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 \mathbf{A} and \mathbf{X} . We can apply different kinds of algorithms for estimating \mathbf{A} and \mathbf{X} . For example, we could choose the EMLL algorithm to update estimation of \mathbf{A} while using the ISRA algorithm for the update of \mathbf{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:

$$a_{ij} \leftarrow a_{ij} \frac{[\mathbf{Y} \mathbf{X}^T]_{ij}}{[\mathbf{A} \mathbf{X} \mathbf{X}^T]_{ij}}, \quad (4)$$

$$x_{jk} \leftarrow x_{jk} \frac{[\mathbf{A}^T \mathbf{Y}]_{jk}}{[\mathbf{A}^T \mathbf{A} \mathbf{X}]_{jk}}, \quad (5)$$

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 \mathbf{X} iteratively with \mathbf{A} assumed to be known. See e.g.,

C. L. Byrne. Accelerating the EML 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:

$$\mathbf{A} \leftarrow \mathbf{A} \odot \mathbf{Y} \mathbf{X}^T \oslash \mathbf{A} \mathbf{X} \mathbf{X}^T, \quad (6)$$

$$\mathbf{X} \leftarrow \mathbf{X} \odot \mathbf{A}^T \mathbf{Y} \oslash \mathbf{A}^T \mathbf{A} \mathbf{X}, \quad (7)$$

where \odot and \oslash denote component-wise multiplication and division, respectively³.

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

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

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

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

C. L. Byrne. Accelerating the EML 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

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

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

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

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:

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

$$D_{KL}^{\alpha_A, \alpha_X}(\mathbf{Y} \parallel \mathbf{A}\mathbf{X}) = \sum_{ik} \left(y_{ik} \log \frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} + [\mathbf{A}\mathbf{X}]_{ik} - y_{ik} \right) + \alpha_A J_A(\mathbf{A}) + \alpha_X J_X(\mathbf{X}), \quad (13)$$

s. t. $\forall i, j, k : x_{jk} \geq 0, \quad a_{ij} \geq 0,$

where $\alpha_A \geq 0$ and $\alpha_X \geq 0$ are regularization parameters, and functions $J_X(\mathbf{X})$ and $J_A(\mathbf{A})$ 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:

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

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

$$a_{ij} = \frac{a_{ij}}{\sum_{j=1}^r a_{ij}}, \quad (16)$$

where the nonlinear operator is defined as

$$[x]_\varepsilon = \max\{\varepsilon, x\} \quad (17)$$

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

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

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

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

$$\nabla_A J_A(\mathbf{A}) = 1, \quad \nabla_X J_X(\mathbf{X}) = 1,$$

– **Squared L_2 norm** ($p = 2$) – *for smooth solution*:

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

$$\nabla_A J_A(\mathbf{A}) = a_{ij}, \quad \nabla_X J_X(\mathbf{X}) = x_{jk},$$

– **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

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

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 interest⁵. Thus

$$J_A(\mathbf{A}) = 0, \quad J_X(\mathbf{X}) = U(\mathbf{X}), \quad (19)$$

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

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

$$\psi(\xi, \delta) = \delta \log[\cosh(\xi/\delta)], \quad (21)$$

which leads to

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

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:

$$a_{ij} \leftarrow a_{ij} \frac{[\mathbf{Y} \mathbf{X}^T]_{ij} - \alpha_A}{[\mathbf{A} \mathbf{X} \mathbf{X}^T]_{ij}}_{\varepsilon}, \quad (23)$$

$$x_{jk} \leftarrow x_{jk} \frac{[\mathbf{A}^T \mathbf{Y}]_{jk} - \alpha_X}{[\mathbf{A}^T \mathbf{A} \mathbf{X}]_{jk}}_{\varepsilon}, \quad (24)$$

$$a_{ij} = \frac{a_{ij}}{\sum_{j=1}^r a_{ij}}, \quad (25)$$

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

Regularization of EMLL 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 EMLL 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 \mathbf{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 \mathbf{A} .

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

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

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

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. Neuroinformatics 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:

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

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:

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

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

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

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

$$\nabla_{\mathbf{X}} D_F(\mathbf{Y} || \mathbf{A} \mathbf{X}) = \mathbf{A}^T \mathbf{A} \mathbf{X} - \mathbf{A}^T \mathbf{Y}, \quad (32)$$

$$\nabla_{\mathbf{A}} D_F(\mathbf{Y} || \mathbf{A} \mathbf{X}) = \mathbf{A} \mathbf{X} \mathbf{X}^T - \mathbf{Y} \mathbf{X}^T. \quad (33)$$

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

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

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

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

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

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

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 parameters⁸. 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: $\mathbf{A}_{(0)}$, α_0 , τ ,
For $k = 1, 2, \dots$
 $\alpha_{(k)} = \alpha_0 \exp\left\{-\frac{k}{\tau}\right\}$,
 $\mathbf{X}_{(k)} = \max\left\{\varepsilon, \left(\mathbf{A}_{(k-1)}^T \mathbf{A}_{(k-1)} + \alpha_{(k)}\right)^+ \mathbf{A}_{(k-1)}^T \mathbf{Y}\right\}$,
 $\mathbf{A}_{(k)} = \max\left\{\varepsilon, \mathbf{Y} \mathbf{X}_{(k)}^T \left(\mathbf{X}_{(k)} \mathbf{X}_{(k)}^T + \alpha_{(k)}\right)^+\right\}$,
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 % EMLL
```

```

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 % EMLL

    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:

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

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

where $P_\Omega[\xi]$ is a projection of ξ onto feasible set Ω (i.e., in our case non-negativity of entries of \mathbf{X} and \mathbf{A}), \mathbf{P}_X and \mathbf{P}_A are descent directions for \mathbf{X} and \mathbf{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 $D_F^{\alpha_A, \alpha_X}(\mathbf{Y} || \mathbf{A}\mathbf{X})$ 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 $D_F(\mathbf{Y} || \mathbf{A}\mathbf{X}) = (1/2)\|\mathbf{Y} - \mathbf{A}\mathbf{X}\|_F^2$ 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:

$$\nabla_X^2 D_F(\mathbf{Y} || \mathbf{A}\mathbf{X}) \mathbf{P}_X = -\nabla_X D_F(\mathbf{Y} || \mathbf{A}\mathbf{X}), \quad (40)$$

$$\nabla_A^2 D_F(\mathbf{Y} || \mathbf{A}\mathbf{X}) \mathbf{P}_A = -\nabla_A D_F(\mathbf{Y} || \mathbf{A}\mathbf{X}), \quad (41)$$

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

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

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

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

where learning rate $\eta^{(k)}$ is adjusted with the Armijo rule, and Ω is a set of feasible solutions – in our case: $P_\Omega[\mathbf{X}] = [\mathbf{X}]_\epsilon$ (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 $\mathbf{A}, \mathbf{X}, \beta \in (0, 1), \mu \in (0, 1), \gamma_{GP} \in (0, 1),$ % Initialization
For $s = 0, 1, \dots,$ % 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 $n = 0, 1, \dots$, % Inner loop for \mathbf{X}

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

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

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

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

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

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

If $D_F(\mathbf{X}^{(n-1)}) - D_F(\mathbf{X}^{(n)}) \leq \gamma_{GP} \max\{D_F(\mathbf{X}^{(l-1)}) - D_F(\mathbf{X}^{(l)}) | l = 1, \dots, n-1\}$,
Break

End If

End % Inner loop for \mathbf{X}

Algorithm 3 (A-GPCG)

For $n = 0, 1, \dots$, % Inner loop for \mathbf{A}

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

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

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

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

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

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

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

End If

End % Inner loop for \mathbf{A}

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

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

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

where α is a regularization term. In the field **L-seminorm**, \mathbf{L} can be specified by an identity matrix ($\mathbf{L} = \mathbf{I}$), 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:

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

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

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

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

where

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

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 \mathbf{X} , η_X is chosen to satisfy

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

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

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

A similar rule is applied for computing \mathbf{A} . We extend Lin's algorithm to other divergences $D(\mathbf{Y} || \mathbf{A}\mathbf{X})$ 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 $\beta = (\text{Beta}_{\text{Armijo}})^{-1}$.

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 $\mathbf{P}^{(k)}$ for \mathbf{X} is determined by a negative scaled gradient, i.e.

$$\mathbf{P}^{(k)} = -\mathbf{D}^{(k)} \odot \nabla_{\mathbf{X}} D_F(\mathbf{Y} || \mathbf{A}^{(k)} \mathbf{X}^{(k)}), \quad (48)$$

with the scaling vector

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

In the NMFLAB, the cost function $D_F(\mathbf{Y} || \mathbf{A} \mathbf{X})$ 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 $D_F(\mathbf{X}^{(k)} + \eta \mathbf{P}^{(k)} | \mathbf{Y}, \mathbf{A}^{(k)})$ or $D_F(\mathbf{A}^{(k)} + \eta \mathbf{P}^{(k)} | \mathbf{Y}, \mathbf{X}^{(k)})$, 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 $\mathbf{A}, \mathbf{X}, \tau \in (0, 1),$ % Initialization
For $s = 0, 1, \dots,$ % 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 $n = 0, 1, \dots,$ % Inner loop for \mathbf{A}
 $\nabla_{\mathbf{A}} D(\mathbf{Y} || \mathbf{A} \mathbf{X}) \leftarrow (\mathbf{A} \mathbf{X} - \mathbf{Y}) \mathbf{X}^T,$ % Gradient
 $\mathbf{D} \leftarrow \mathbf{A} \oslash (\mathbf{A} \mathbf{X} \mathbf{X}^T),$ % Scaling vector
 $\mathbf{P} \leftarrow -\mathbf{D} \odot \nabla_{\mathbf{A}} D(\mathbf{Y} || \mathbf{A} \mathbf{X}),$ % Descent direction
 $\eta^* = -(\text{vec}(\mathbf{P})^T \text{vec}(\nabla_{\mathbf{A}} D(\mathbf{Y} || \mathbf{A} \mathbf{X}))) / (\text{vec}(\mathbf{P} \mathbf{X})^T \text{vec}(\mathbf{P} \mathbf{X})),$ % Exact minimizer
 $\hat{\eta} = \max\{\eta : \mathbf{A} + \eta \mathbf{P} \geq 0\},$ % Step length towards boundary
 Set: $\hat{\tau} \in [\tau, 1), \eta = \min(\hat{\tau} \hat{\eta}, \eta^*),$ % Current step length
 $\mathbf{A} \leftarrow \mathbf{A} + \eta \mathbf{P},$ % Update
End % Alternating

Algorithm 6 (X-IPG)

For $n = 0, 1, \dots$, % Inner loop for \mathbf{X}
 $\nabla_X D(\mathbf{Y} \|\mathbf{A}\mathbf{X}) \leftarrow \mathbf{A}^T(\mathbf{A}\mathbf{X} - \mathbf{Y})$, % Gradient
 $\mathbf{D} \leftarrow \mathbf{X} \odot (\mathbf{A}^T \mathbf{A}\mathbf{X})$, % Scaling vector
 $\mathbf{P} \leftarrow -\mathbf{D} \odot \nabla_X D(\mathbf{Y} \|\mathbf{A}\mathbf{X})$, % Descent direction
 $\eta^* = -(\text{vec}(\mathbf{P})^T \text{vec}(\nabla_X D(\mathbf{Y} \|\mathbf{A}\mathbf{X}))) / (\text{vec}(\mathbf{A}\mathbf{P})^T \text{vec}(\mathbf{A}\mathbf{P}))$, % Exact minimizer
 $\hat{\eta} = \max\{\eta : \mathbf{X} + \eta\mathbf{P} \geq 0\}$, % Step length towards boundary
Set: $\hat{\tau} \in [\tau, 1)$, $\eta = \min(\hat{\tau}\hat{\eta}, \eta^*)$, % Current step length
 $\mathbf{X} \leftarrow \mathbf{X} + \eta\mathbf{P}$, % 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 \mathbf{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 \mathbf{A} is known)

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

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

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

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

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

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

$$D_F^{(\alpha_X)}(\mathbf{Y} \|\mathbf{A}\mathbf{X}) = \frac{1}{2} \|\mathbf{Y} - \mathbf{A}\mathbf{X}\|_F^2 + \alpha_X J_X(\mathbf{X}), \quad (51)$$

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

Algorithm 7 (*MRNSD-NMF*)

Set $\mathbf{A}, \mathbf{X}, \mathbf{1}_m = [1, \dots, 1]^T \in \mathbb{R}^m$,
 $\mathbf{1}_r = [1, \dots, 1]^T \in \mathbb{R}^r$, $\mathbf{1}_T = [1, \dots, 1]^T \in \mathbb{R}^T$, % Initialization
For $s = 0, 1, \dots$, % 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*)

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

Algorithm 9 (*X-MRNSD*)

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

$$\min_{\mathbf{X} \geq 0} \{D_{KL}(\mathbf{Y}||\mathbf{A}\mathbf{X}) + D_H(\mathbf{Y}||\mathbf{A}\mathbf{X})\}, \quad (52)$$

where $D_{KL}(\mathbf{Y}||\mathbf{A}\mathbf{X})$ is Kullback-Leibler divergence (3), and

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

is the Hellinger distance.

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

Algorithm 10 (*Relaxed Hellinger*)

```

Set       $\mathbf{A}, \mathbf{X},$       % Initialization
For       $s = 0, 1, \dots,$  % Alternating
    For     $n = 0, 1, \dots, n_{inner},$  % Inner loop for  $\mathbf{A}$ 
Step 1:     $\mathbf{A} \leftarrow \mathbf{A} \odot \left[ \frac{1}{4} + \frac{1}{4} \sqrt{1 + 8[\mathbf{Y} \oslash \mathbf{A}\mathbf{X}]\mathbf{X}^T} \right]$ 
    End    % Inner loop for  $\mathbf{A}$ 
    For     $n = 0, 1, \dots, n_{inner},$  % Inner loop for  $\mathbf{X}$ 
Step 2:     $\mathbf{X} \leftarrow \left( \mathbf{X} \odot \left[ \frac{1}{4} + \frac{1}{4} \sqrt{1 + 8\mathbf{A}^T[\mathbf{Y} \oslash \mathbf{A}\mathbf{X}]} \right] \right)^{\alpha_{Sx}}$ 
    End    % Inner loop for  $\mathbf{X}$ 
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. Tusnady. 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 \mathbf{X} as in (27).

Algorithm 11 (*Csiszár based algorithm*)

Set \mathbf{A}, \mathbf{X} % Initialization
For $s = 0, 1, \dots,$ % Alternating
 For $n = 0, 1, \dots, n_{inner},$ % Inner loop for \mathbf{A}
 Step 1: $i = 1, \dots, m, j = 1, \dots, r:$

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

 End % Inner loop for \mathbf{A}
 For $n = 0, 1, \dots, n_{inner},$ % Inner loop for \mathbf{X}
 Step 2: $j = 1, \dots, r, k = 1, \dots, T:$

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

 End % Inner loop for \mathbf{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

$$D_C(\mathbf{z}||\mathbf{y}) = \sum_{k=1}^N z_k \varphi\left(\frac{y_k}{z_k}\right) \quad (53)$$

where $y_k \geq 0, z_k \geq 0$ and $\varphi : [0, \infty) \rightarrow (-\infty, \infty)$ 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:

$$D_A^{(\alpha)}(\mathbf{A}\mathbf{X}||\mathbf{Y}) = \sum_{ik} y_{ik} \frac{(y_{ik}/z_{ik})^{\alpha-1} - 1}{\alpha(\alpha - 1)} + \frac{z_{ik} - y_{ik}}{\alpha}, \quad z_{ik} = [\mathbf{A}\mathbf{X}]_{ik}. \quad (54)$$

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

See also Ali-Silvey, 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 $\alpha \rightarrow 1$ and $\alpha \rightarrow 0$, respectively. When these limits are evaluated one gets for $\alpha \rightarrow 1$ the generalized Kullback-Leibler divergence (called I-divergence) defined by equations (3) and for $\alpha \rightarrow 0$ the dual generalized KL divergence:

$$D_{KL2}(\mathbf{A}\mathbf{X}||\mathbf{Y}) = \sum_{ik} \left([\mathbf{A}\mathbf{X}]_{ik} \log \frac{[\mathbf{A}\mathbf{X}]_{ik}}{y_{ik}} - [\mathbf{A}\mathbf{X}]_{ik} + y_{ik} \right) \quad (55)$$

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

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

with normalization of columns of \mathbf{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:

$$\begin{aligned} \mathbf{X} &\leftarrow \mathbf{X} \odot \left(\mathbf{A}^T ((\mathbf{Y} + \varepsilon) \oslash (\mathbf{A}\mathbf{X} + \varepsilon))^\alpha \right)^{1/\alpha} \\ \mathbf{A} &\leftarrow \mathbf{A} \odot \left(((\mathbf{Y} + \varepsilon) \oslash (\mathbf{A}\mathbf{X} + \varepsilon))^\alpha \mathbf{X}^T \right)^{1/\alpha}, \quad \mathbf{A} \leftarrow \mathbf{A} \text{diag}\{1 \oslash \mathbf{A}^T \mathbf{1}_m\}, \end{aligned} \quad (56)$$

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:

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

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

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

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

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

Table 2. Amari Alpha-NMF algorithms (continued)

<i>Kullback-Leibler divergence: ($\alpha \rightarrow 1$):</i>	
$\lim_{\alpha \rightarrow 1} D_A^{(\alpha)}(y_{ik} z_{ik}) = \sum_{ik} y_{ik} \log \frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} - y_{ik} + [\mathbf{A}\mathbf{X}]_{ik},$	
Algorithm:	$x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^m a_{ij} \frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} \right)^{\omega_X} \right)^{1+\alpha_{sX}}$ $a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^N x_{jk} \frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} \right)^{\omega_A} \right)^{1+\alpha_{sA}}$ $a_{ij} \leftarrow a_{ij} / \sum_p a_{pj}$ $0 < \omega_X < 2, \quad 0 < \omega_A < 2$
<i>Dual Kullback-Leibler divergence: ($\alpha \rightarrow 0$):</i>	
$\lim_{\alpha \rightarrow 0} D_A^{(\alpha)}(y_{ik} z_{ik}) = \sum_{ik} [\mathbf{A}\mathbf{X}]_{ik} \log \frac{[\mathbf{A}\mathbf{X}]_{ik}}{y_{ik}} + y_{ik} - [\mathbf{A}\mathbf{X}]_{ik}$	
Algorithm:	$x_{jk} \leftarrow \left(x_{jk} \prod_{i=1}^m \left(\frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} \right)^{\omega_X a_{ij}} \right)^{1+\alpha_{sX}}$ $a_{ij} \leftarrow \left(a_{ij} \prod_{k=1}^N \left(\frac{y_{ik}}{[\mathbf{A}\mathbf{X}]_{ik}} \right)^{\tilde{\eta}_j x_{jk}} \right)^{1+\alpha_{sA}}$ $a_{ij} \leftarrow a_{ij} / \sum_p a_{pj}$ $0 < \omega_X < 2, \quad 0 < \omega_A < 2$

Table 3. Other generalized NMF algorithms

<p><i>Beta generalized divergence:</i></p> $D_K^{(\beta)}(y_{ik} z_{ik}) = \sum_{ik} y_{ik} \frac{y_{ik}^{\beta-1} - [\mathbf{A}\mathbf{X}]_{ik}^{\beta-1}}{\beta(\beta-1)} + [\mathbf{A}\mathbf{X}]_{ik}^{\beta-1} \frac{[\mathbf{A}\mathbf{X}]_{ik} - y_{ik}}{\beta}$ <p>Kompass algorithm:</p> $x_{jk} \leftarrow x_{jk} \frac{\sum_{i=1}^m a_{ij} (y_{ik}/[\mathbf{A}\mathbf{X}]_{ik}^{2-\beta})}{\sum_{i=1}^m a_{ij} [\mathbf{A}\mathbf{X}]_{ik}^{\beta-1} + \varepsilon}$ $a_{ij} \leftarrow \left(a_{ij} \frac{\sum_{k=1}^N x_{jk} (y_{ik}/[\mathbf{A}\mathbf{X}]_{ik}^{2-\beta})}{\sum_{k=1}^N x_{jk} [\mathbf{A}\mathbf{X}]_{ik}^{\beta-1} + \varepsilon} \right)^{1+\alpha_{sA}}$ $a_{ij} \leftarrow a_{ij} / \sum_p a_{pj},$
<p><i>Triangular discrimination:</i></p> $D_T^{(\beta)}(y_{ik} z_{ik}) = \sum_{ik} \frac{y_{ik}^\beta z_{ik}^{1-\beta} - \beta y_{ik} + (\beta-1)z_{ik}}{\beta(\beta-1)}$ <p>Algorithm:</p> $x_{jk} \leftarrow \left(x_{jk} \left(\sum_{i=1}^m a_{ij} \left(\frac{2y_{ik}}{y_{ik} + [\mathbf{A}\mathbf{X}]_{ik}} \right)^2 \right)^{\omega_X} \right)^{1+\alpha_{sX}}$ $a_{ij} \leftarrow \left(a_{ij} \left(\sum_{k=1}^N x_{jk} \left(\frac{2y_{ik}}{y_{ik} + [\mathbf{A}\mathbf{X}]_{ik}} \right)^2 \right)^{\omega_A} \right)^{1+\alpha_{sA}}$ $a_{ij} \leftarrow a_{ij} / \sum_p a_{pj}, \quad 0 < \omega_X < 2, \quad 0 < \omega_A < 2$
<p><i>Itakura-Saito distance:</i></p> $D_{IS}(y_{ik} z_{ik}) = \sum_{ik} \frac{y_{ik}}{z_{ik}} - \log \left(\frac{y_{ik}}{z_{ik}} \right) - 1$ <p>Algorithm:</p> $\mathbf{X} \leftarrow \mathbf{X} \odot [(\mathbf{A}^T \mathbf{P}) \oslash (\mathbf{A}^T \mathbf{Q} + \varepsilon)].^\beta$ $\mathbf{A} \leftarrow \mathbf{A} \odot [(\mathbf{P}\mathbf{X}^T) \oslash (\mathbf{Q}\mathbf{X}^T + \varepsilon)].^\beta$ $a_{ij} \leftarrow a_{ij} / \sum_p a_{pj}, \quad \beta = [0.5, 1]$ $\mathbf{P} = \mathbf{Y} \oslash (\mathbf{A}\mathbf{X} + \varepsilon)^2, \quad \mathbf{Q} = \mathbf{1} \oslash (\mathbf{A}\mathbf{X} + \varepsilon)$

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

C. L. Byrne. Accelerating the EML 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:

$$\mathbf{X} \leftarrow \mathbf{X} \odot \exp \left(\boldsymbol{\eta}_A \mathbf{A}^T \odot \ln(\mathbf{Y} \oslash (\mathbf{A}\mathbf{X} + \epsilon)) \right) \quad (60)$$

$$\mathbf{A} \leftarrow \mathbf{A} \odot \exp \left(\ln(\mathbf{Y} \oslash (\mathbf{A}\mathbf{X} + \epsilon)) \mathbf{X}^T \boldsymbol{\eta}_X \right), \quad (61)$$

$$\mathbf{A} \leftarrow \mathbf{A} \text{diag}\{1 \oslash \mathbf{A}^T \mathbf{1}_m\}, \quad (62)$$

where in practice a small constant $\epsilon = 10^{-16}$ 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 $\boldsymbol{\eta}_A = \omega_A \text{diag}\{1 \oslash \mathbf{A}^T \mathbf{1}_m\}$ and $\boldsymbol{\eta}_X = \omega_X \text{diag}\{1 \oslash \mathbf{X} \mathbf{1}_T\}$.

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

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

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

where the error functions defined as

$$\rho(y_{ik}, z_{ik}) = -\frac{\partial D(\mathbf{Y} \parallel \mathbf{A}\mathbf{X})}{\partial z_{ik}} \quad (65)$$

can take different forms depending on the chosen or designed loss (cost) function $D(\mathbf{Y} \parallel \mathbf{A}\mathbf{X})$ (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

$$\mathbf{X} \leftarrow \left[\mathbf{X} - [\mathbf{H}_{D_A}^{(X)}]^{-1} \nabla_X D_A \right]_{\epsilon}, \quad \mathbf{A} \leftarrow \left[\mathbf{A} - [\mathbf{H}_{D_A}^{(A)}]^{-1} \nabla_A D_A \right]_{\epsilon}, \quad (66)$$

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

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

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

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

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

Thus for \mathbf{X} we have

$$\mathbf{G}_{D_A}^{(X)} = \nabla_{\mathbf{X}} D_A = \frac{1}{\alpha} \mathbf{A}^T (\mathbf{1} - (\mathbf{Y} \odot (\mathbf{A}\mathbf{X}))^\alpha) \in \mathbb{R}^{R \times K}. \quad (67)$$

The Hessian has the form:

$$\mathbf{H}_{D_A}^{(X)} = \frac{1}{\alpha} \text{diag}\{[\mathbf{h}_k^{(X)}]_{k=1,\dots,K}\} \in \mathbb{R}^{RK \times RK} \quad (68)$$

where

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

For \mathbf{A} , we get

$$\mathbf{G}_{D_A}^{(A)} = \nabla_{\mathbf{A}} D_A = \frac{1}{\alpha} (\mathbf{1} - (\mathbf{Y} \odot (\mathbf{A}\mathbf{X}))^\alpha) \mathbf{X}^T \in \mathbb{R}^{M \times R}. \quad (69)$$

The Hessian has the form:

$$\mathbf{H}_{D_A}^{(A)} = \frac{1}{\alpha} \text{diag}\{[\mathbf{h}_m^{(A)}]_{m=1,\dots,M}\} \in \mathbb{R}^{MR \times MR} \quad (70)$$

where

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

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

$$\mathbf{X} \leftarrow [\mathbf{X} - \gamma \mathbf{R}_X \backslash \mathbf{W}_X]_\epsilon, \quad \mathbf{A} \leftarrow [\mathbf{A} - \gamma \mathbf{R}_A \backslash \mathbf{W}_A]_\epsilon, \quad (71)$$

$$\mathbf{W}_X = \mathbf{Q}_X^T \nabla_{\mathbf{X}} D_A, \quad \mathbf{Q}_X \mathbf{R}_X = \mathbf{H}_{D_A}^{(X)} + \lambda \mathbf{I}_X,$$

$$\mathbf{W}_A = \mathbf{Q}_A^T \nabla_{\mathbf{A}} D_A, \quad \mathbf{Q}_A \mathbf{R}_A = \mathbf{H}_{D_A}^{(A)} + \lambda \mathbf{I}_A,$$

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

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

$$\begin{aligned} D_{KL2}(\mathbf{Y} \parallel \mathbf{A}\mathbf{X}) &= \lim_{\alpha \rightarrow 0} D_A(\mathbf{A}\mathbf{X} \parallel \mathbf{Y}) \\ &= \sum_{mk} \left(z_{mk} \log \frac{z_{mk}}{y_{mk}} + y_{mk} - z_{mk} \right), \quad z_{mk} = [\mathbf{A}\mathbf{X}]_{mk}, \end{aligned} \quad (72)$$

and consequently the gradient and Hessian are as follows

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

and

$$\mathbf{H}_{D_{KL2}}^{(X)} = \text{diag}\{[\mathbf{h}_k^{(X)}]_{k=1,\dots,K}\} \in \mathbb{R}^{RK \times RK}, \quad (74)$$

where

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

For \mathbf{A} , we have

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

The Hessian has the form:

$$\mathbf{H}_{D_{KL2}}^{(A)} = \text{diag}\{[\mathbf{h}_m^{(A)}]_{m=1,\dots,M}\} \in \mathbb{R}^{MR \times MR}, \quad (76)$$

where

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

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 (\mathbf{X} and \mathbf{A}) does not give a satisfactory solution. In BSS application, \mathbf{A} has much smaller dimension than \mathbf{X} , i.e. $T \gg m \geq r$, and hence, the approximations of \mathbf{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, \mathbf{X} is calculated within a few iterations. Also, in each alternating step the columns of \mathbf{A} are normalized to a unity.

To enforce sparsity of estimated variables we applied the nonlinear projection as in (27) for computation of \mathbf{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
end

```

```

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

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

$$\mathbf{Y} = \mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_{L+1} + \mathbf{V}, \quad \mathbf{A}_p \geq 0, \quad \forall p, p = 1, 2, \dots, L+1 \quad (77)$$

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

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

$$\mathbf{Z} = \mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_{L+1} \quad (78)$$

$$\overline{\mathbf{A}}_p = \mathbf{A}_1 \cdots \mathbf{A}_{p-1}, \quad \overline{\mathbf{A}}_1 = \mathbf{I}_{m \times m} \quad (79)$$

$$\overline{\mathbf{X}}_p = \mathbf{A}_{p+1} \cdots \mathbf{A}_{L+1}, \quad \overline{\mathbf{X}}_{L+1} = \mathbf{I}_{N \times N}, \quad p = 1, 2, \dots, L+1. \quad (80)$$

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 respect to several sets of the parameters \mathbf{A}_p , ($p = 1, 2, \dots, L+1$) 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:

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

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

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

$$\mathbf{A}_p \leftarrow \mathbf{A}_p \odot \left[\bar{\mathbf{A}}_p^T \mathbf{Y} \bar{\mathbf{X}}_p^T - \alpha_{A_p} \right]_{+\varepsilon} \oslash \left[\bar{\mathbf{A}}_p^T \mathbf{Z} \bar{\mathbf{X}}_p^T + \varepsilon \right]. \quad (82)$$

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:

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

$$\mathbf{A}_2 \leftarrow \mathbf{A}_2 \odot \left[\mathbf{A}_1^T \mathbf{Y} \mathbf{A}_3^T - \alpha_{A_2} \right]_{+\varepsilon} \oslash \left[\mathbf{A}_1^T \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3 \mathbf{A}_3^T + \varepsilon \right] \quad (84)$$

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

The above formula can be generalized as

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

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

$$\mathbf{A}_p \leftarrow \mathbf{A}_p \odot \left[\bar{\mathbf{A}}_p^T [\mathbf{Y} \oslash \mathbf{Z}] \bar{\mathbf{X}}_p^T - \alpha_{A_p} \right]_{+\varepsilon} \oslash \left[\bar{\mathbf{A}}_p^T [\mathbf{1}] \bar{\mathbf{X}}_p^T + \varepsilon \right], \quad (87)$$

Alternatively, we can use the following learning rule:

$$\mathbf{A}_p \leftarrow \mathbf{A}_p \odot \left[\bar{\mathbf{A}}_p^T [\mathbf{Y} \oslash \mathbf{Z}] \bar{\mathbf{X}}_p^T - \alpha_{A_p} \right]_{+\varepsilon} \quad (88)$$

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

The regularized algorithm for α divergence:

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

$$\mathbf{A}_p \leftarrow \mathbf{A}_p * \text{diag}\{1 \oslash \mathbf{A}_p^T \mathbf{1}_m\}.$$

The generalized SMART algorithm for GNMF can take the form:

$$\mathbf{A}_p \leftarrow \mathbf{A}_p \odot \exp \left(\eta_{\bar{\mathbf{A}}_p} * \bar{\mathbf{A}}_p^T * \ln(\mathbf{Y} \oslash [\mathbf{Z} + \varepsilon]) * \bar{\mathbf{X}}_p^T * \eta_{\bar{\mathbf{X}}_p} \right), \quad (91)$$

$$\mathbf{A}_p \leftarrow \mathbf{A}_p * \text{diag}\{1 \oslash \mathbf{A}_p^T \mathbf{1}_m\}, \quad (92)$$

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

$$\mathbf{A}_p \leftarrow \mathbf{A}_p^T \odot \exp \left(\eta_{\bar{\mathbf{A}}_p} * \bar{\mathbf{A}}_p * \rho(\mathbf{Y}, \mathbf{Z}) * \bar{\mathbf{X}}_p^T * \eta_{\bar{\mathbf{X}}_p} \right), \quad (93)$$

$$\mathbf{A}_p \leftarrow \mathbf{A}_p * \text{diag}\{1 \odot \mathbf{A}_p^T \mathbf{1}_m\}, \quad (94)$$

For example, for the symmetric Jensen-Shannon divergence:

$$\rho(\mathbf{Y}, \mathbf{Z}) = \ln ([\mathbf{Y} + \mathbf{Z}] \odot [2\mathbf{Z} + \varepsilon]) \quad (95)$$

and for α divergence (see Table 1):

$$\rho(\mathbf{Y}, \mathbf{Z}) = \frac{1}{\alpha} [([\mathbf{Y} + \varepsilon] \odot [\mathbf{Z} + \varepsilon])^\alpha - 1] \quad (96)$$

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;

$$\mathbf{Y} = \mathbf{A}\mathbf{S}\mathbf{X} \quad (97)$$

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

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

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 \mathbf{X} occurs. However, as Θ tends to 1 the vector $\tilde{\mathbf{x}}(k) = \mathbf{S}\mathbf{x}(k)$ tends to a constant vector with entries almost equal to the average of the elements of $\mathbf{x}(k)$. This is the smoothest possible vector and also non-sparse because all entries are equal to the same nonzero value. The matrix \mathbf{S} performs some smoothness (or non-sparseness) operation on the columns of the matrix \mathbf{X} . However, due to the multiplicative nature of this model, strong smoothing in \mathbf{S} will force strong sparseness in both the basis matrix and encoding vectors $\mathbf{X}(k)$ in order to maintain faithfulness of the model to the data \mathbf{Y} . Please see for details

A. Pascual-Montano, J. M. Carazo, K. Kochi, D. Lehman, and R. Pascual-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 \mathbf{X} by employing the special form of the GNMF as follows

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{M} \quad (99)$$

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

$$\mathbf{M} = \frac{1}{3} \begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & 1 & 1 & 1 & 0 & 0 & 0 & 0 \dots \\ \dots & 0 & 1 & 1 & 1 & 0 & 0 & 0 \dots \\ \dots & 0 & 0 & 1 & 1 & 1 & 0 & 0 \dots \\ \dots & 0 & 0 & 0 & 1 & 1 & 1 & 0 \dots \\ \dots & 0 & 0 & 0 & 0 & 1 & 1 & 1 \dots \\ & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \quad (100)$$

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 corrélees, in Proc. Grets, (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, vol. 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, vol. 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-Alvarez, 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 \mathbf{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.