

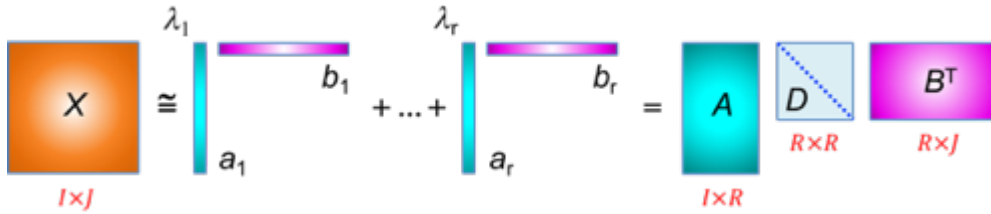
Function **nnmf_sca**: Nonnegative Matrix Factorization and Sparse Component Analysis.

The goal of Blind Source Separation (BSS) methods is to estimate the physical sources of a mixing system. Most BSS models can be expressed algebraically as a factorization of a data matrix into the factor matrices:

$$\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_R] \in \mathcal{R}^{I \times R} \quad \mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_R] \in \mathcal{R}^{J \times R}$$

$$\mathbf{X} = \mathbf{A} \mathbf{D} \mathbf{B}^T + \mathbf{E} = \sum_{r=1}^R \lambda_r \mathbf{a}_r \mathbf{b}_r^T + \mathbf{E} = \sum_{r=1}^R \lambda_r \mathbf{a}_r \circ \mathbf{b}_r + \mathbf{E}$$

where the symbol \circ denotes the *outer product*, $\mathbf{D} = \text{diag}[\lambda_1, \dots, \lambda_r]$ is a scaling matrix (possibly \mathbf{I}), the columns of \mathbf{B} are the unknown source signals (*factors or latent variables*), the columns of \mathbf{A} are the associated mixing vectors (or *factor loadings*), and \mathbf{E} is noise due to unmodelled part of the data or model error.

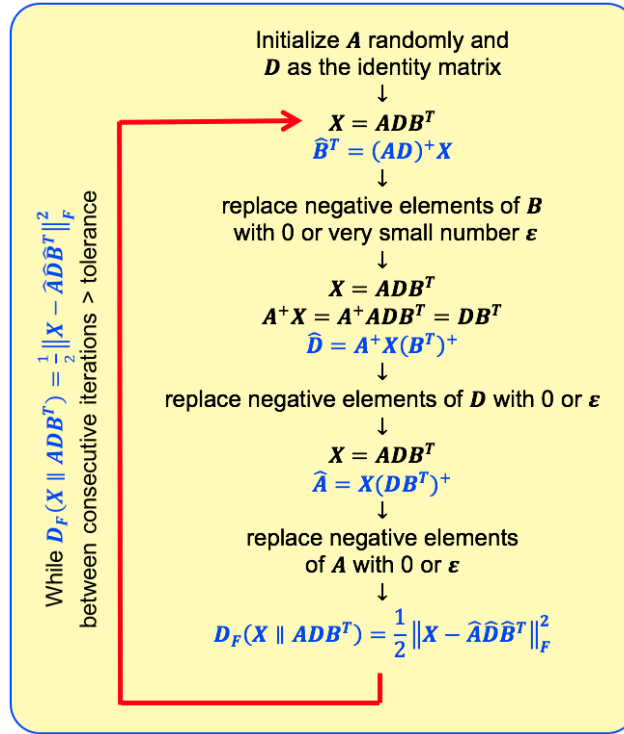


Without some *a priori* knowledge, or without specific constraints, it is not possible to estimate uniquely the original source signals. However, often \mathbf{X} can be nonnegative and the corresponding hidden components of \mathbf{X} may have a physical meaning only when nonnegative. In practice, both *nonnegative matrix factorization*, NMF, and *sparse component analysis*, SCA, of data can be necessary for the underlying latent components to have a physical interpretation.

In standard NMF we only assume nonnegativity of the factor matrices \mathbf{A} and \mathbf{B} , and unlike ICA, we do not assume that the sources are *independent*. In order to estimate factor matrices \mathbf{A} and \mathbf{B} we need to quantify a *cost function*, the distance between the data matrix and the NMF model $\hat{\mathbf{X}} = \mathbf{A} \mathbf{D} \mathbf{B}^T$. The simplest *distance* measure is based on the *Frobenius norm*:

$$D_F(\mathbf{X} \parallel \mathbf{A} \mathbf{D} \mathbf{B}^T) = \frac{1}{2} \|\mathbf{X} - \mathbf{A} \mathbf{D} \mathbf{B}^T\|_F^2$$

also referred to as the *squared Euclidean distance*. Alternating minimization of such a cost leads to the *Alternating Least Squares* (ALS) algorithm: in this method, after an initial random initialization of \mathbf{A} , a least squares solution for \mathbf{A} with \mathbf{B} fixed and for \mathbf{B} with \mathbf{A} fixed is carried out iteratively until the cost function reaches a minimum, or the difference in cost function between consecutive iterations becomes smaller than a given tolerance value, or a maximum number of iterations is reached. In each iteration, the negative elements of \mathbf{A} and \mathbf{B} and the off-diagonal elements of \mathbf{D} are replaced with 0 or with a very small number ϵ :



where a + superscript indicate the Moore-Penrose pseudoinverse. A simple modification of this algorithm allows also the imposition of a *sparseness* constraint (with or without nonnegativity) on the **A** matrix. In this case at each iteration we set to 0 a given fraction of the smallest elements of **A**. This way, *Nonnegative matrix factorization* (NMF) turns into *Sparse component analysis* (SCA). The algorithm is implemented in the function `nnmf_sca`.

`nnmf_sca` can be conveniently used to obtain a *lower rank approximation* of the original matrix **X** as ADB^T , with a diagonal matrix **D**, or constraining $D = I$. The following are some examples of possible uses of `nnmf_sca`:

```

% function [ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
%     nnmf_sca(X,k,dchoice,achoice,aspase,schoice,maxiter)
% Nonnegative matrix factorization and/or Sparse component analysis.
%
% Usage:
% [ A,D,Bt,X_hat,niter,sse,sse_diff,sparseness,tot_sparseness,sparseness ] = ...
%     nnmf_sca(X,k,dchoice,balance,achoice,aspase,schoice,maxiter)
%
% Inputs:
% X:  m x n data matrix.
% k:  number of factors requested.
% Two choices for the D matrix:
% dchoice:  diag|ident
% Three choices for the A matrix:
% achoice:  nneg|sparse|both
% aspase:   level of sparseness
% Two choices for sparseness, random or along the long dimension of X:
% schoice:  random|bylong
% maxiter:  maximum number of iterations allowed.

```

```
%
% Outputs:
% A,D,Bt
% X_hat: A*D*Bt
% niter: total number of iterations used.
% sse: sum of squared errors
% sse_diff: difference in sse between final iterations
% tot_sparseness: total sparseness
% sparseness: sparseness along the long dimension
```

```
X = randi(20,10,50)
```

```
x = 10×50
    17    12    14    14     1     8     5    13     8    16     8    20    18 ...
    16     9     9     4     8     9     9     8    11     6    10     4    10
    12    13    13     9     4    17     3    16     1    16     4     3     8
    18    16    20     3     6     3    16     1    20     6     8    17    12
     3    17     9     2     3    15    10     2    13    19    11     7     5
    18    17     6    10     6    16     9     6     8     3     9     5     1
    19     1     9     2     8    19    12    20    10    18    17    18     8
     5     5    17    13    11    19    12    15    12    15     5    19     2
    20     4    11     1     8    14     2     1    15    16    12    11    16
     2     7    19     7     9    14    20    10    14     2    12    11    17
```

```
% X = randi(20,50,10)
openvar A, openvar D, openvar Bt, openvar X, openvar X_hat
```

```
% NMF
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness ] = nnmf_sca(X,5);
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness ] = nnmf_sca(X,5,'diag');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness ] = nnmf_sca(X,10,'diag');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness ] = nnmf_sca(X,10,'ident');
```

```
% SCA
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness ] = nnmf_sca(X,5,'ident','sparse');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,5,'ident','sparse',0.3);
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,10,'diag','sparse',0.3,'bylong');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,5,'ident','sparse',0.3,'random');
```

```
% NMF + SCA
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,5,'ident','both',0.3,'bylong');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,5,'diag','both',0.3,'bylong');
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...
    nnmf_sca(X,5,'diag','both',0.3,'bylong');
```

```
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...  
    nnmf_sca(X,5,'diag','both',0.3,'random');  
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...  
    nnmf_sca(X,10,'diag','both',0.3,'random');  
[ A,D,Bt,X_hat,niter,sse,sse_diff,tot_sparseness,sparseness ] = ...  
    nnmf_sca(X,10,'ident','both',0.3,'random');
```