Software Manual

Institute of Bioinformatics, Johannes Kepler University Linz

FABIA: Factor Analysis for Bicluster Acquisition
— Manual for the R package —

Sepp Hochreiter

Institute of Bioinformatics, Johannes Kepler University Linz
Altenberger Str. 69, 4040 Linz, Austria
hochreit@bioinf.jku.at

Version 0.1.2, December 23, 2009

Contents

1 Introduction

The fabia package is part of the Bioconductor (http://www.bioconductor.org) project. The package allows to extract biclusters from data sets based on a generative model. It has been designed especially for microarray data sets, but can be used for other kinds of data sets as well.

2 Getting Started: FABIA

First load the fabia package:

```
R> library(fabia) ##load the fabia package
```

The fabia package is stand-alone and does not require other packages.

2.1 Quick start

Assume your data is in the file datafile.csv in a matrix like format then you can try out the following steps to extract biclusters.

1. Create a working directory, e.g. c:/fabia/data in Windows or /home/myself/fabia/data in Unix. Move the data file datafile.csv to that directory, e.g. under Unix

   ```
   cp datafile.csv /home/myself/fabia/data/
   ```

   or drag the file datafile.csv into that directory under Windows.

2. Start R and change to the working directory. Under Windows

   ```
   R> setwd("c:/fabia/data")
   ```

   and under Unix

   ```
   R> setwd("/home/myself/fabia/data")
   ```

   You can also start R in that directory under Unix.

3. Load the library:

   ```
   R> library(fabia) ##load the fabia package
   ```

4. Read the data file “datafile.csv”:

   ```
   R> X <- read.table("datafile.csv",header = TRUE, sep = "\t")
   ```

5. Select the model based on the data: 5 biclusters; sparseness 0.1; 400 cycles

   ```
   R> res <- fabia(X,400,0.1,5)
   ```

6. Plot some results:

```
R> rr <- extract_plot(X,res$L,res$z,ti="FABIA")
```

7. Extract biclusters:

```
R> rb <- extract_bic(res$L,res$Z,lapla=res$lapla,Psi=res$Psi)
```

8. Show information content of the biclusters:

```
R> rb$avini #$
```

9. List bicluster 1:

```
R> rb$bic[1,] #$
```

10. List bicluster 2:

```
R> rb$bic[2,] #$
```

11. Show bicluster 3:

```
R> rb$bic[3,] #$
```

12. List bicluster 4:

```
R> rb$bic[4,] #$
```

13. List bicluster 5:

```
R> rb$bic[5,] #$
```

14. Plot bicluster 1:

```
R> PlotBicluster(X,unlist(rb$bic[1,5]),unlist(rb$bic[1,3]))
```

15. Plot bicluster 2:

```
R> PlotBicluster(X,unlist(rb$bic[2,5]),unlist(rb$bic[2,3]))
```

16. Plot bicluster 3:

```
R> PlotBicluster(X,unlist(rb$bic[3,5]),unlist(rb$bic[3,3]))
```

17. Plot bicluster 4:

```
R> PlotBicluster(X,unlist(rb$bic[4,5]),unlist(rb$bic[4,3]))
```

18. Plot bicluster 5:

```
R> PlotBicluster(X,unlist(rb$bic[5,5]),unlist(rb$bic[5,3]))
```

19. List opposite bicluster 1:

```
R> rb$bicopp[1,] #$
```

20. Plot opposite bicluster 1:

```
R> PlotBicluster(X,unlist(rb$bicopp[1,5]),unlist(rb$bicopp[1,3]))
```

2.2 Test on Toy Data Set

In the following, we describe how you can test the package `fabia` on a toy data set that is generated on-line.

1. generate bicluster data, where biclusters are in block format in order to obtain a better visualization of the results. 1000 observations, 100 samples, 10 biclusters:

```
R> dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,
    f2 = 5,of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,
    mean_z = 2.0,sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)
```

2. store the generated data in variables:

```
R> X <- dat[[1]]
R> Y <- dat[[2]]
```

3. perform `fabia` (sparseness by Laplace prior) to extract biclusters; 400 cycles, sparseness 0.1 (Laplace), 13 biclusters:

```
R> resToy1 <- fabia(X,400,0.1,1.0,1.0,13)
```

4. Plot some results:

```
R> rToy1 <- extract_plot(X,resToy1$L,resToy1$Z,"FABIA",Y=Y)
```

5. perform `fabias` (sparseness by projection) to extract biclusters; 200 cycles, sparseness 0.5 (projection), 13 biclusters:

```
R> resToy2 <- fabias(X,200,0.5,1.0,13)
```

6. Plot some results:

```
R> rToy2 <- extract_plot(X,resToy2$L,resToy2$Z,"FABIAS",Y=Y)
```

7. perform `fabiap` (Laplace prior then projection) to extract biclusters; 200 cycles, sparseness 0.1 (Laplace), 13 biclusters, 0.7 sparseness loading (projection), 0.7 sparseness factors (projection):

```
R> resToy3 <- fabiap(X,200,0.1,1.0,1.0,13,0.7,0.7)
```

8. Plot some results:

```
R> rToy3 <- extract_plot(X,resToy3$L,resToy3$Z,"FABIAP",Y=Y)
```

9. perform `mfsc` (sparse matrix factorization), 13 biclusters, 0.7 sparseness loading (projection), 0.7 sparseness factors (projection):

```
R> resToy4 <- mfsc(X,13,sL=0.7,sZ=0.7)
```

10. Plot some results:

```
R> rToy4 <- extract_plot(X,resToy4$L,resToy4$Z,"MFSC",Y=Y)
```

2.3 Demos

The package `fabia` has some demos which can be demonstrated by `fabiaDemo`.

1. demo1: toy data.

   ```
   R> fabiaDemo()
   ```

   Choose "1" and you get above toy data demonstration.

2. demo2: Microarray data set of van't Veer et al. (2002) on breast cancer.

   ```
   R> fabiaDemo()
   ```

   Choose "2" to extract subclasses in the data set of van't Veer as biclusters.

3. demo3: Microarray data set of Su et al. (2002) on different mammalian.

   ```
   R> fabiaDemo()
   ```

   Choose "3" to check whether the different mouse and human tissue types can be extracted.

4. demo4: Microarray data set of Rosenwald et al. (2002) diffuse large-B-cell lymphoma. Hoshida et al. (2007) divided the data set into three classes

   - OxPhos: oxidative phosphorylation
   - BCR: B-cell response
   - HR: host response

   ```
   R> fabiaDemo()
   ```

   Choose "4" to check whether the different classes can be extracted.

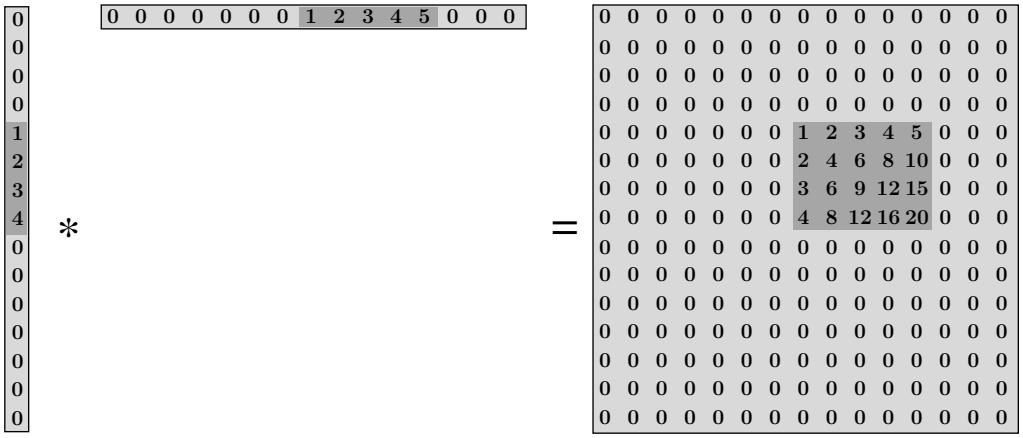

Figure 1: The outer product $\boldsymbol{\lambda}\, \boldsymbol{z}^T$ of two sparse vectors results in a matrix with a bicluster. Note, that the non-zero entries in the vectors are adjacent to each other for visualization purposes only.

3 The FABIA Model

3.1 Model Assumptions

We define a *bicluster* as a pair of a row (gene) set and a column (sample) set for which the rows are similar to each other on the columns and vice versa. In a multiplicative model, two vectors are similar if one is a multiple of the other, that is the angle between them is zero or as realization of random variables their correlation coefficient is one. It is clear that such a linear dependency on subsets of rows and columns can be represented as an outer product $\boldsymbol{\lambda}\, \boldsymbol{z}^T$ of two vectors $\boldsymbol{\lambda}$ and $\boldsymbol{z}$. The vector $\boldsymbol{\lambda}$ corresponds to a *prototype column vector* that contains zeros for genes not participating in the bicluster, whereas $\boldsymbol{z}$ is a vector of *factors* with which the prototype column vector is scaled for each sample; clearly $\boldsymbol{z}$ contains zeros for samples not participating in the bicluster. Vectors containing many zeros or values close to zero are called *sparse vectors*. Fig. 1 visualizes this representation by sparse vectors schematically.

The overall model for p biclusters and additive noise is

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i\, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;=\; \boldsymbol{\Lambda}\, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon}\,, \tag{1}$$

where $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$ is additive noise and $\boldsymbol{\lambda}_i \in \mathbb{R}^n$ and $\boldsymbol{z}_i \in \mathbb{R}^l$ are the sparse prototype vector and the sparse vector of factors of the i-th bicluster, respectively. The second formulation above holds if $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$ is the sparse prototype matrix containing the prototype vectors $\boldsymbol{\lambda}_i$ as columns and $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$ is the sparse factor matrix containing the transposed factors $\boldsymbol{z}_i^T$ as rows. Note that Eq. (1) formulates biclustering as sparse matrix factorization.

According to Eq. (1), the j-th sample $\boldsymbol{x}_j$, i.e., the j-th column of $\boldsymbol{X}$, is

$$\boldsymbol{x}_j \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i\, z_{ij} \;+\; \boldsymbol{\epsilon}_j \;=\; \boldsymbol{\Lambda}\, \tilde{\boldsymbol{z}}_j \;+\; \boldsymbol{\epsilon}_j\,, \tag{2}$$

where $\boldsymbol{\epsilon}_j$ is the j-th column of the noise matrix $\boldsymbol{\Upsilon}$ and $\tilde{\boldsymbol{z}}_j = (z_{1j}, \ldots, z_{pj})^T$ denotes the j-th column of the matrix $\boldsymbol{Z}$. Recall that $\boldsymbol{z}_i^T = (z_{i1}, \ldots, z_{il})$ is the vector of values that constitutes the i-th bicluster (one value per sample), while $\tilde{\boldsymbol{z}}_j$ is the vector of values that contribute to the j-th sample (one value per bicluster).

The formulation in Eq. (2) facilitates a generative interpretation by a factor analysis model with p factors

$$\boldsymbol{x} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \tilde{z}_i \;+\; \boldsymbol{\epsilon} \;=\; \boldsymbol{\Lambda} \, \tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon} \,, \tag{3}$$

where $\boldsymbol{x}$ are the observations, $\boldsymbol{\Lambda}$ is the loading matrix, $\tilde{z}_i$ is the value of the i-th factor, $\tilde{\boldsymbol{z}} = (\tilde{z}_1, \ldots, \tilde{z}_p)^T$ is the vector of factors, and $\boldsymbol{\epsilon} \in \mathbb{R}^n$ is the additive noise. Standard factor analysis assumes that the noise is independent of $\tilde{\boldsymbol{z}}$, that $\tilde{\boldsymbol{z}}$ is $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$-distributed, and that $\boldsymbol{\epsilon}$ is $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Psi})$-distributed, where the covariance matrix $\boldsymbol{\Psi} \in \mathbb{R}^{n \times n}$ is a diagonal matrix expressing independent Gaussian noise. The parameter $\boldsymbol{\Lambda}$ explains the dependent (common) and $\boldsymbol{\Psi}$ the independent variance in the observations $\boldsymbol{x}$. Normality of the additive noise in gene expression is justified by the findings in (Hochreiter et al., 2006).

The unity matrix as covariance matrix for $\tilde{\boldsymbol{z}}$ may be violated by overlapping biclusters, however we want to avoid to divide a real bicluster into two factors. Thus, we prefer uncorrelated factors over more sparseness. The factors can be decorrelated by setting $\hat{\boldsymbol{z}} := \boldsymbol{A}^{-1} \, \tilde{\boldsymbol{z}}$ and $\hat{\boldsymbol{\Lambda}} := \boldsymbol{\Lambda} \, \boldsymbol{A}$ with the symmetric invertible matrix $\boldsymbol{A}^2 = \mathrm{E}\big(\tilde{\boldsymbol{z}} \, \tilde{\boldsymbol{z}}^T\big)$:

$$\begin{aligned} &\boldsymbol{\Lambda} \, \boldsymbol{z} \;=\; \boldsymbol{\Lambda} \, \boldsymbol{A} \, \boldsymbol{A}^{-1} \, \boldsymbol{z} \;=\; \hat{\boldsymbol{\Lambda}} \, \hat{\boldsymbol{z}} \quad \text{and} \\ &\mathrm{E}\big(\hat{\boldsymbol{z}} \, \hat{\boldsymbol{z}}^T\big) \;=\; \boldsymbol{A}^{-1} \, \mathrm{E}\big(\tilde{\boldsymbol{z}} \, \tilde{\boldsymbol{z}}^T\big) \, \boldsymbol{A}^{-1} \;=\; \boldsymbol{A}^{-1} \boldsymbol{A}^2 \boldsymbol{A}^{-1} \;=\; \boldsymbol{I} \,. \end{aligned}$$

Standard factor analysis does not consider sparse factors and sparse loadings which are essential in our formulation to represent biclusters. Sparseness is obtained by a component-wise independent *Laplace* distribution (Hyvärinen and Oja, 1999), which is now used as a prior on the factors $\tilde{\boldsymbol{z}}$ instead of the Gaussian:

$$p(\tilde{\boldsymbol{z}}) \;=\; \left(\tfrac{1}{\sqrt{2}}\right)^p \prod_{i=1}^{p} e^{-\sqrt{2}\,|\tilde{z}_i|}$$

Sparse loadings $\boldsymbol{\lambda}_i$ and, therefore sparse $\boldsymbol{\Lambda}$, are achieved by two alternative strategies. In the first model, called FABIA, we assume a component-wise independent *Laplace* prior for the loadings (like for the factors):

$$p(\boldsymbol{\lambda}_i) \;=\; \left(\tfrac{1}{\sqrt{2}}\right)^n \prod_{k=1}^{n} e^{-\sqrt{2}\,|\lambda_{ki}|} \tag{4}$$

The FABIA model contains the product of Laplacian variables which is distributed proportionally to the 0-th order modified Bessel function of the second kind (Bithas et al., 2007). For large values, this Bessel function is a negative exponential function of the square root of the random variable. Therefore, the tails of the distribution are heavier than those of the Laplace distribution. The Gaussian noise, however, reduces the heaviness of the tails such that the heaviness is between Gaussian and Bessel function tails — about as heavy as the tails of the Laplacian distribution. These *heavy tails* are exactly the desired model characteristics.

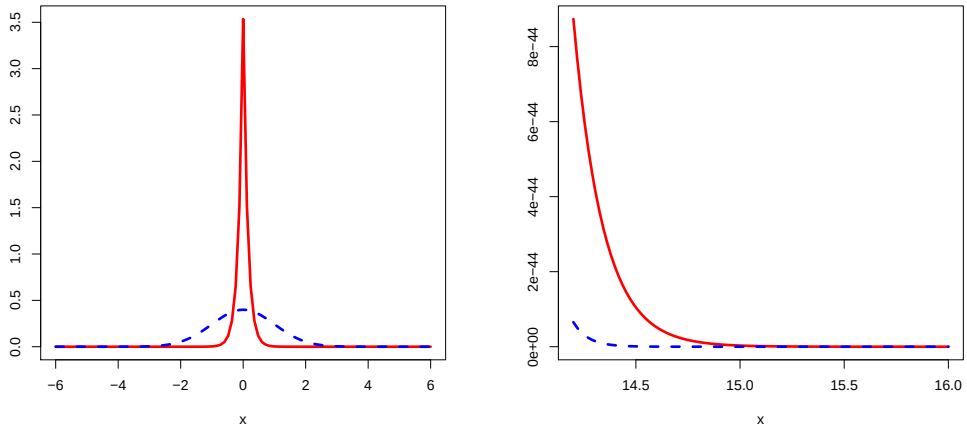

Figure 2: Left: The mode of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution. Right: The tails of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution

The second model, called FABIAS, applies a prior with parameter spL on the loadings that has only support at sparse regions. Following (Hoyer, 2004), we define sparseness as

$$\mathrm{sp}(\boldsymbol{\lambda}_i) \;=\; \frac{\sqrt{n} \;-\; \sum_{k=1}^{n} |\lambda_{ki}| \;/\; \sum_{k=1}^{n} \lambda_{ki}^2}{\sqrt{n} \;-\; 1}$$

leading to the prior

$$p(\boldsymbol{\lambda}_i) \;=\; \begin{cases} c & \text{for} \quad \mathrm{sp}(\boldsymbol{\lambda}_i) \;\leq\; \mathrm{spL} \\ 0 & \text{for} \quad \mathrm{sp}(\boldsymbol{\lambda}_i) \;>\; \mathrm{spL} \end{cases} . \tag{5}$$

3.2 Sparse Coding and Laplace Prior

Laplace prior enforces sparse codes on the factors. *Sparse coding* is the representation of items by the *strong activation* of a relatively *small set* of hidden factors while the factors are *almost constant* if not activated.

Laplace prior is suited for modeling strong activation for few samples while being otherwise almost constant. Fig. 2 left shows the Laplacian mode compared to a Gaussian mode. The Laplacian mode is higher and narrower. Fig. 2 right shows the tails of Gaussian and Laplace distribution, where the latter has higher values. This means for the Laplace distribution large values are more likely than for a Gaussian.

3.3 Model Selection

The free parameters $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ can be estimated by Expectation-Maximization (EM; Dempster et al., 1977). With a prior probability on the loadings, the a posteriori of the parameters is maximized like in (Hochreiter et al., 2006; Talloen et al., 2007).

3.3.1 Variational Approach for Sparse Factors

Model selection is not straightforward because the likelihood

$$p(\boldsymbol{x} \mid \boldsymbol{\Lambda}, \boldsymbol{\Psi}) \;=\; \int p(\boldsymbol{x} \mid \tilde{\boldsymbol{z}}, \boldsymbol{\Lambda}, \boldsymbol{\Psi})\; p(\tilde{\boldsymbol{z}})\; d\tilde{\boldsymbol{z}}$$

cannot be computed analytically for a Laplacian prior $p(\tilde{\boldsymbol{z}})$. We employ a variational approach according to Girolami (2001) and Palmer et al. (2006) for model selection. They introduce a model family that is parametrized by $\boldsymbol{\xi}$, where the maximum over models in this family is the true likelihood:

$$\arg\max_{\boldsymbol{\xi}} p(\boldsymbol{x}|\boldsymbol{\xi}) \;=\; \log p(\boldsymbol{x})\;.$$

Using an EM algorithm, not only the likelihood with respect to the parameters $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ is maximized, but also with respect to $\boldsymbol{\xi}$.

In the following, $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ denote the actual parameter estimates. According to Girolami (2001) and Palmer et al. (2006), we obtain

$$\begin{aligned}
\mathrm{E}\big(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j\big) \;&=\; \big(\boldsymbol{\Lambda}^T\, \boldsymbol{\Psi}^{-1}\, \boldsymbol{\Lambda} \;+\; \boldsymbol{\Xi}_j^{-1}\big)^{-1}\, \boldsymbol{\Lambda}^T\, \boldsymbol{\Psi}^{-1}\, \boldsymbol{x}_j \quad \text{and} \\
\mathrm{E}\big(\tilde{\boldsymbol{z}}_j\, \tilde{\boldsymbol{z}}_j^T \mid \boldsymbol{x}_j\big) \;&=\; \big(\boldsymbol{\Lambda}^T\, \boldsymbol{\Psi}^{-1}\, \boldsymbol{\Lambda} \;+\; \boldsymbol{\Xi}_j^{-1}\big)^{-1} \;+ \\
&\qquad \mathrm{E}\big(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j\big)\, \mathrm{E}(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j)^T\;,
\end{aligned}$$

where $\boldsymbol{\Xi}_j$ stands for $\mathrm{diag}\,(\boldsymbol{\xi}_j)$. The update for $\boldsymbol{\xi}_j$ is

$$\boldsymbol{\xi}_j \;=\; \mathrm{diag}\left(\sqrt{\mathrm{E}(\tilde{\boldsymbol{z}}_j\, \tilde{\boldsymbol{z}}_j^T \mid \boldsymbol{x}_j)}\right)\;.$$

3.3.2 New Update Rules for Sparse Loadings

The update rules for FABIA (Laplace prior on loadings) are

$$\begin{aligned}
\boldsymbol{\Lambda}^{\mathrm{new}} \;&=\; \frac{\frac{1}{l}\, \sum_{j=1}^{l} \boldsymbol{x}_j\, \mathrm{E}(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j)^T \;-\; \frac{\alpha}{l}\, \boldsymbol{\Psi}\, \mathrm{sign}(\boldsymbol{\Lambda})}{\frac{1}{l} \sum_{j=1}^{l} \mathrm{E}(\tilde{\boldsymbol{z}}_j\, \tilde{\boldsymbol{z}}_j^T \mid \boldsymbol{x}_j)} \\
\mathrm{diag}\,(\boldsymbol{\Psi}^{\mathrm{new}}) \;&=\; \boldsymbol{\Psi}^{\mathrm{EM}} \;+\; \mathrm{diag}\Big(\frac{\alpha}{l}\, \boldsymbol{\Psi}\, \mathrm{sign}(\boldsymbol{\Lambda})(\boldsymbol{\Lambda}^{\mathrm{new}})^T\Big)
\end{aligned} \tag{6}$$

where

$$\boldsymbol{\Psi}^{\mathrm{EM}} \;=\; \mathrm{diag}\Big(\frac{1}{l} \sum_{j=1}^{l} \boldsymbol{x}_j \boldsymbol{x}_j^T \;-\; \boldsymbol{\Lambda}^{\mathrm{new}} \frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j\right)\, \boldsymbol{x}_j^T\Big).$$

The update rules for FABIAS must take into account that each $\boldsymbol{\lambda}_i$ from $\boldsymbol{\Lambda}$ has a prior with restricted support. Therefore the sparseness constraints $\mathrm{sp}(\boldsymbol{\lambda}_i) \leq \mathrm{spL}$ from Eq. (5) hold. These constraints are ensured by a projection of $\boldsymbol{\lambda}_i$ after each $\boldsymbol{\Lambda}$ update according to Hoyer (2004). The projection is a convex quadratic problem which minimizes the Euclidean distance to the original

vector subject to the constraints. The projection problem can be solved fast by an iterative procedure where the l_2-norm of the vectors is fixed to 1. We update $\mathrm{diag}(\mathbf{\Psi}^{\mathrm{new}}) = \mathbf{\Psi}^{\mathrm{EM}}$ and project each updated prototype vector to a sparse vector with sparseness spL giving the overall projection:

$$\mathbf{\Lambda}^{\mathrm{new}} \;=\; \mathrm{proj}\left(\frac{\frac{1}{l}\,\sum_{j=1}^{l} \boldsymbol{x}_j\,\mathrm{E}\left(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j\right)^T}{\frac{1}{l}\sum_{j=1}^{l}\mathrm{E}(\tilde{\boldsymbol{z}}_j\,\tilde{\boldsymbol{z}}_j^T \mid \boldsymbol{x}_j)}, \mathrm{spL}\right)$$

3.3.3 Extremely Sparse Priors

Some gene expression data sets are sparser than Laplacian. For example, during estimating DNA copy numbers with Affymetrix SNP 6 arrays, we observed a kurtosis larger than 30 (FABIA results shown at http://www.bioinf.jku.at/software/fabia/fabia.html). We want to adapt our model class to deal with such sparse data sets. Toward this end, we define extremely sparse priors both on the factors and the loadings utilizing the following (pseudo) distributions:

Generalized Gaussians:	$p(z) \;\propto\; \exp\left(-\,\lvert z\rvert^{\beta}\right)$
Jeffrey's prior:	$p(z) \;\propto\; \exp\left(-\,\ln\lvert z\rvert\right) \;=\; 1/\lvert z\rvert$
Improper prior:	$p(z) \;\propto\; \exp\left(\lvert z\rvert^{-\beta}\right)$

For the first distribution, we assume $0 < \beta \leq 1$ and for the third $0 < \beta$. Note, the third distribution may only exist on the interval $[\epsilon, a]$ with $0 < \epsilon < a$. We assume that ϵ is sufficiently small.

For the *loadings*, we need the derivatives of the negative log-distributions for optimizing the log-posterior. These derivatives are proportional to $|z|^{-\mathrm{spl}}$, where $\mathrm{spl} = 0$ corresponds to the Laplace prior and $\mathrm{spl} > 0$ to sparser priors. The update rule is as in Eq. (6), where $\mathrm{sign}(\mathbf{\Lambda})$ is replaced by $|\mathbf{\Lambda}|^{-\mathrm{spl}}\,\mathrm{sign}(\mathbf{\Lambda})$ with element-wise operations (absolute value, sign, exponentiation, multiplication).

For the *factors*, we represent the priors through a convex variational form according to Palmer et al. (2006). That is possible because $g(z) = -\ln p(\sqrt{z})$ is increasing and concave for $z > 0$ (first order derivatives are larger and second order smaller than zero). According to Palmer et al. (2006), the update for $\boldsymbol{\xi}_j$ is

$$\boldsymbol{\xi}_j \;\propto\; \mathrm{diag}\Big(\mathrm{E}(\tilde{\boldsymbol{z}}_j\,\tilde{\boldsymbol{z}}_j^T \mid \boldsymbol{x}_j)^{\mathrm{spz}}\Big)$$

for all $\mathrm{spz} \geq 1/2$, where $\mathrm{spz} = 1/2$ ($\beta = 1$) represents the Laplace prior and $\mathrm{spz} > 1/2$ leads to sparser priors.

3.4 Data Preprocessing and Initialization

The data $\boldsymbol{x}$ may be centered either to zero mean or to zero median which we prefer to obtain sparser raw data. Then the data should be scaled to unit second moments to allow initialization of the parameters in the same range. See the supplementary for justification of these preprocessing steps.

The iterative model selection procedure requires initialization of the parameters $\mathbf{\Lambda}$, $\mathbf{\Psi}$, and $\boldsymbol{\xi}_j$. The simplest strategy is to initialize $\mathbf{\Lambda}$ randomly while ensuring that

$$\mathbf{\Psi} \;=\; \mathrm{diag}\Big(\mathrm{covar}(\boldsymbol{x}) \;-\; \mathbf{\Lambda}\,\mathbf{\Lambda}^T\Big) \;\geq\; \delta \;>\; 0.$$

The variational parameter vectors $\boldsymbol{\xi}_j$ are initialized as vectors of ones. An alternative initialization strategy can be based on ICA. The ICA solution supplies factors $\boldsymbol{Z}_{\mathrm{ICA}}$ that are sparse and decorrelated.

3.5 Information Content of Biclusters

A highly desired property for biclustering algorithms is the ability to rank the extracted biclusters analogously to principal components which are ranked according to the data variance they explain. We rank biclusters according to the information they contain about the data. The information content of $\tilde{\boldsymbol{z}}_j$ for the j-th observation $\boldsymbol{x}_j$ is the mutual information between $\tilde{\boldsymbol{z}}_j$ and $\boldsymbol{x}_j$:

$$\mathrm{I}(\boldsymbol{x}_j; \tilde{\boldsymbol{z}}_j) \;=\; \mathrm{H}(\tilde{\boldsymbol{z}}_j) \;-\; \mathrm{H}(\tilde{\boldsymbol{z}}_j \mid \boldsymbol{x}_j) \;=\frac{1}{2}\; \ln \left| \boldsymbol{I}_p \;+\; \boldsymbol{\Xi}_j\, \boldsymbol{\Lambda}^T\, \boldsymbol{\Psi}^{-1}\, \boldsymbol{\Lambda} \right|$$

The independence of $\boldsymbol{x}_j$ and $\tilde{\boldsymbol{z}}_j$ across j gives

$$\mathrm{I}(\boldsymbol{X}; \boldsymbol{Z}) \;=\; \frac{1}{2} \sum_{j=1}^{l} \ln \left| \boldsymbol{I}_p \;+\; \boldsymbol{\Xi}_j\, \boldsymbol{\Lambda}^T\, \boldsymbol{\Psi}^{-1}\, \boldsymbol{\Lambda} \right| .$$

For the FARMS summarization algorithm ($p = 1$ and $\boldsymbol{\Xi}_j = 1$), this information is the negative logarithm of the I/NI call (Talloen et al., 2007).

To assess the information content of one factor, we consider the case that factor $\tilde{z}_i$ is removed from the final model. This corresponds to setting $\xi_{ij} = 0$ (by ξ_{ij}, we denote the i-th entry in $\boldsymbol{\xi}_j$) and therefore the explained covariance $\xi_{ji}\, \boldsymbol{\lambda}_i\, \boldsymbol{\lambda}_i^T$ is removed:

$$\boldsymbol{x}_j \mid (\tilde{\boldsymbol{z}}_j \setminus z_{ij}) \;\sim\; \mathcal{N}\big(\boldsymbol{\Lambda}\; \tilde{\boldsymbol{z}}_j|_{z_{ij}=0}\;,\; \boldsymbol{\Psi} \;+\; \xi_{ij}\, \boldsymbol{\lambda}_i\, \boldsymbol{\lambda}_i^T\big)$$

The information of z_{ij} given the other factors is

$$\begin{aligned} \mathrm{I}(\boldsymbol{x}_j; z_{ij} \mid (\tilde{\boldsymbol{z}}_j \setminus z_{ij})) \;&=\; \mathrm{H}(z_{ij} \mid (\tilde{\boldsymbol{z}}_j \setminus z_{ij})) - \mathrm{H}(z_{ij} \mid (\tilde{\boldsymbol{z}}_j \setminus z_{ij}), \boldsymbol{x}_j) \\ &=\; \frac{1}{2}\; \ln \big(1 \;+\; \xi_{ij}\, \boldsymbol{\lambda}_i^T \boldsymbol{\Psi}^{-1} \boldsymbol{\lambda}_i\big)\; . \end{aligned}$$

Again independence across j gives

$$\mathrm{I}(\boldsymbol{X}; \boldsymbol{z}_i^T \mid (\boldsymbol{Z} \setminus \boldsymbol{z}_i^T)) \;=\; \frac{1}{2} \sum_{j=1}^{l} \ln \big(1 \;+\; \xi_{ij}\, \boldsymbol{\lambda}_i^T \boldsymbol{\Psi}^{-1} \boldsymbol{\lambda}_i\big)\; .$$

This information content gives that part of information in $\boldsymbol{x}$ that $\boldsymbol{z}_i^T$ conveys across all examples. Note that also the number of nonzero $\boldsymbol{\lambda}_i$'s (size of the bicluster) enters the information content.

3.6 Extracting Members of Biclusters

After model selection in Section 3.3 and ranking of bicluster in Section 3.5, the i-th bicluster has soft gene memberships given by the absolute values of $\boldsymbol{\lambda}_i$ and soft sample memberships given by the absolute values of $\boldsymbol{z}_i^T$.

However, applications may need hard memberships. We determine the members of bicluster i by selecting absolute values λ_{ki} and z_{ij} above thresholds thresL and thresZ, respectively.

First, the second moment of each factor is normalized to 1 resulting in a factor matrix $\hat{\boldsymbol{Z}}$ (in accordance with $\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T) = \boldsymbol{I}$). Consequently, $\boldsymbol{\Lambda}$ is rescaled to $\hat{\boldsymbol{\Lambda}}$ such that $\boldsymbol{\Lambda}\boldsymbol{Z} = \hat{\boldsymbol{\Lambda}}\hat{\boldsymbol{Z}}$. Now the threshold thresZ can be chosen to determine which percentage of samples will on average belong to a bicluster. For a Laplace prior, this percentage can be computed by $\frac{1}{2}\exp(-\sqrt{2}/\mathrm{thresZ})$.

In the default setting, for each factor $\hat{\boldsymbol{z}}_i$, only one bicluster is extracted. In gene expression, an expression pattern is either absent or present but not negatively present. Therefore, the i-th bicluster is either determined by the positive or negative values of $\hat{z}_{ij}$. Which one of these two possibilities is chosen is decided by whether the sum over $\left|\hat{z}_{ij}\right| > \mathrm{thresZ}$ is larger for the positive or negative $\hat{z}_{ij}$.

The threshold thresL for the loadings is more difficult to determine, because normalization would lead to a rescaling of the already normalized factors. Since biclusters may overlap, the contribution of $\lambda_{ki}z_{ij}$ that are relevant must be estimated. Therefore, we first estimate the standard deviation of $\boldsymbol{\Lambda}\boldsymbol{Z}$ by

$$\mathrm{sdLZ} \;=\; \sqrt{\frac{1}{p\;l\;n}\sum_{(i,j,k)=(1,1,1)}^{(p,l,n)}\left(\hat{\lambda}_{ki}\;\hat{z}_{ij}\right)^2}\,.$$

We set this standard deviation to the product of both thresholds which is solved for thresL: $\mathrm{thresL} = \mathrm{sdLZ} \;/\; \mathrm{thresZ}$. However, an optimal thresL depends on the sparseness parameters and on the characteristics of the biclustering problem.

3.7 C implementation of FABIA

The functions `fabia` and `fabias` are implemented in C. It turned out that these implementations are not only faster, but also more precise. Especially we use an efficient Cholesky decomposition to compute the inverse of positive definite matrices. Some R functions for computing the inverse like `solve` were inferior to that implementation.

The interface between R and C is realized by the package `Rcpp` Samperi (2006).

A Methods and Functions

A.1 extract_plot

Extraction of Biclusters and Plotting of the Results.

1. Usage: `extract_plot(X,L,Z,thresZ=0.5,ti,thresL=NULL,Y=NULL,`
   `x11b=TRUE,norm=1,center=2)`
2. Arguments:
   - X: original data matrix.

- L: loading, left matrix.
- Z: factor, right matrix.
- thresZ: threshold for sample belonging to bicluster (default 0.5).
- thresL: threshold for loading belonging to bicluster (estimated if not given).
- ti: plot title.
- Y: alternative: noise free data matrix.
- x11b: screen output or not.
- norm: data normalization: default = 1 (yes), 0 (no).
- center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.

3. Return Values:

- bic: extracted biclusters.
- numn: indexes for the extracted biclusters.
- biclust: clusters of kmeans clustering.
- pmZ: permutation matrix of z from kmeans clustering.
- pmL: permutation matrix of L from kmeans clustering.
- nL: normalized loadings (left matrix).
- nZ normalized factors (right matrix).
- Xord: sorted original matrix according to kmeans on Z and kmeans on L.

4. Produced Plots:

- Y: noise free data (if available)
- X: data
- LZ: reconstructed data
- LZ-X: error
- abs(Z): absolute factors
- abs(L): absolute loadings
- abs(nL): absolute loadings normalized
- abs(nZ): absolute factors normalized
- nZ*pmZ: factors sorted
- pmL*nL: loadings sorted
- pmL*L*Z*pmZ: reconstructed matrix sorted
- pmL*X*pmZ: original matrix sorted

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \; \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda}\,\boldsymbol{Z} \;+\; \boldsymbol{\Upsilon}\,.$$

The $\boldsymbol{\lambda}_i$ and $\boldsymbol{z}_i^T$ are used to extract the bicluster i, where a threshold determines which observations and which samples belong the the bicluster.

The method produces a couple of plots given below.

In the above plots, the matrix $\boldsymbol{\Lambda}$ and the matrix $\boldsymbol{Z}$ are sorted. For sorting, first `kmeans` on the p dimensional space is performed and then the vectors which belong to the same cluster are put together in the sorting. This sorting is made for visualization but in general it is not possible to visualize all biclusters as blocks if they overlap.

In `bic` the biclusters are extracted according to the largest absolute values of the component i, i.e. the largest values of $\boldsymbol{\lambda}_i$ and the largest values of $\boldsymbol{z}_i$. The factors $\boldsymbol{z}_i$ are normalized to variance 1.

The components of `bic` are `bixv`, `bixn`, `biypv`, `biypn`, `biynv`, and `biynn`. `bixv` gives the values of the observations that have absolute values above a threshold. They are sorted and `bixn` gives their names (e.g. gene names). `biypv` gives the values of the samples that have values above a threshold. They are sorted and `biypn` gives their names (e.g. sample names). `biynv` gives the values of the samples that have values below this threshold. They are sorted and `biynn` gives their names (e.g. sample names).

That means the samples are divided into two groups where one group shows large positive values and the other group has negative values with large absolute values. That means a observation pattern can be switched on or switched off relative to the average value.

`numn` gives the indexes of `bic` with components: `numn1 = bix` ,`numn2 = biyp`, and `numn3 = biyn`.

The kmeans clusters are given by `biclust` with components `biclustx` (the clustered observations) and `biclusty` (the clustered samples).

Implementation in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
resEx <- fabia(X,20,0.3,1.0,1.0,3)

rEx <- extract_plot(X,resEx$L,resEx$Z,ti="FABIA",Y=Y,x11b=FALSE)

rEx$bic[1,]
rEx$bic[2,]
rEx$bic[3,]
rEx$biclust[1,]
rEx$biclust[2,]
rEx$biclust[3,]


#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,200,0.4,1.0,1.0,13)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="FABIA",Y=Y)

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)


resBreast <- fabia(X,200,0.1,1.0,1.0,5)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,ti="FABIA Breast cancer(Veer)")

#sorting of predefined labels
CBreast%*%rBreast$pmZ
```

A.2 extract_bic

Extraction of Biclusters.

1. Usage: `extract_bic(L,Z,thresL=0.02,thresZ=1.0,lapla=NULL,Psi=NULL)`
2. Arguments:
   - L: loading, left matrix.
   - Z: factor, right matrix.
   - thresZ: threshold for sample belonging to bicluster (default 0.5).
   - thresL: threshold for loading belonging to bicluster (if not given it is estimated).
   - lapla: inverse variance of the variational approximation for each sample and each factor.
   - Psi: noise variance vector for observations where independent noise is assumed.
3. Return Values:
   - bic: extracted biclusters.
   - numn: indexes for the extracted biclusters.
   - bicopp: extracted opposite biclusters.
   - numnopp: indexes for the extracted opposite biclusters.
   - avini: average over j of the variance $\boldsymbol{z}_i$ given $\boldsymbol{x}_j$.
   - ini: for each j the variance $\boldsymbol{z}_i$ given $\boldsymbol{x}_j$.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda}\, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon} \;.$$

$\boldsymbol{\Upsilon}$ is the Gaussian noise with a diagonal covariance matrix which entries are given by `Psi`.

The $\boldsymbol{Z}$ is locally approximated by a Gaussian with inverse variance given by `lapla`.

The $\boldsymbol{\lambda}_i$ and $\boldsymbol{z}_i$ are used to extract the bicluster i, where a threshold determines which observations and which samples belong the the bicluster.

In `bic` the biclusters are extracted according to the largest absolute values of the component i, i.e. the largest values of $\boldsymbol{\lambda}_i$ and the largest values of $\boldsymbol{z}_i$. The factors $\boldsymbol{z}_i$ are normalized to variance 1.

The components of `bic` are `binp`, `bixv`, `bixn`, `biypv`, and `biypn`.

`binp` give the size of the bicluster: number observations and number samples. `bixv` gives the values of the extracted observations that have absolute values above a threshold. They are sorted.

bixn gives the extracted observation names (e.g. gene names). biypv gives the values of the extracted samples that have absolute values above a threshold. They are sorted. biypn gives the names of the extracted samples (e.g. sample names).

In bicopp the opposite cluster to the biclusters are give. Opposite means that the negative pattern is present.

The components of opposite clusters bicopp are binn, bixv, bixn, biypnv, and biynn.

binp give the size of the opposite bicluster: number observations and number samples. bixv gives the values of the extracted observations that have absolute values above a threshold. They are sorted. bixn gives the extracted observation names (e.g. gene names). biynv gives the values of the opposite extracted samples that have absolute values above a threshold. They are sorted. biynn gives the names of the opposite extracted samples (e.g. sample names).

That means the samples are divided into two groups where one group shows large positive values and the other group has negative values with large absolute values. That means a observation pattern can be switched on or switched off relative to the average value.

numn gives the indexes of bic with components: numng = bix and numnp = biypn.

numn gives the indexes of bicopp with components: numng = bix and numnn = biynn.

Implementation in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabia(X,20,0.1,1.0,1.0,3)

rEx <- extract_bic(resEx$L,resEx$Z,lapla=resEx$lapla,Psi=resEx$Psi)

rEx$bic[1,] #$
rEx$bic[2,] #$
rEx$bic[3,] #$


#---------------
```

```
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,200,0.4,1.0,1.0,13)

rToy <- extract_bic(resToy$L,resToy$Z,lapla=resToy$lapla,Psi=resToy$Psi)

rToy$avini  #$

rToy$bic[1,] #$
rToy$bic[2,] #$
rToy$bic[3,] #$

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,200,0.1,1.0,1.0,5)

rBreast <- extract_bic(resBreast$L,resBreast$Z,lapla=resBreast$lapla,Psi=resBreast$Psi)

rBreast$avini    #$

rBreast$bic[1,]  #$
rBreast$bic[2,]  #$
rBreast$bic[3,]  #$
```

A.3 fabi

Factor Analysis for Bicluster Acquisition: Laplace Prior (FABI).

R implementation of `fabia`, therefore it is *slow*.

1. Usage: `fabi(X,cyc,alpha,spl,spz,p,norm=1,center=2)`

2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings (0.1 - 1.0).
   - spl: sparseness prior loadings (0.5 - 4.0).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
   - norm: data normalization: default = 1 (yes), 0 (no).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
3. Return Values:
   - LZ: Estimated Noise Free Data: $\mathbf{\Lambda}\ \boldsymbol{Z}$
   - L: Loadings: $\mathbf{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\mathbf{\Psi}$
   - lapla: Variational parameter

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i \ \boldsymbol{z}_i^T \ + \ \mathbf{\Upsilon} \ ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \ = \ \mathbf{\Lambda}\ \boldsymbol{Z} \ + \ \mathbf{\Upsilon} \ .$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \ + \ \boldsymbol{\epsilon} \ = \ \mathbf{\Lambda}\tilde{\boldsymbol{z}} \ + \ \boldsymbol{\epsilon}$$

The model assumptions are:

Factor Prior is Independent Laplace:

$$p(\tilde{\boldsymbol{z}}) \ = \ \left(\frac{1}{\sqrt{2}}\right)^p \prod_{i=1}^{p} e^{-\ \sqrt{2}\ |z_i|}$$

Loading Prior is Independent Laplace:

$$p(\boldsymbol{\lambda}_i) \;=\; \left(\frac{1}{\sqrt{2}}\right)^n \prod_{k=1}^{n} e^{-\sqrt{2}\,|\lambda_{ki}|}$$

Noise: Gaussian independent

$$p(\boldsymbol{\epsilon}) \;=\; \left(\frac{1}{\sqrt{2\,\pi}}\right)^n \prod_{k=1}^{n} \frac{1}{\sigma_k} e^{\sum_{k=1}^{n} \frac{\epsilon_k^2}{\sigma_k^2}}$$

Data Mean:

$$\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\boldsymbol{\Lambda}\;\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}) \;=\; \boldsymbol{\Lambda}\;\mathrm{E}(\tilde{\boldsymbol{z}}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0}$$

Data Covariance:

$$\begin{aligned}\mathrm{E}(\boldsymbol{x}\;\boldsymbol{x}^T) \;&=\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\;\boldsymbol{\epsilon}^T) \\ &=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2)\end{aligned}$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\boldsymbol{\lambda}^k\right)^T \boldsymbol{\lambda}^k \;=\; 1$$

Here $\boldsymbol{\lambda}^k$ is the k-th row of $\boldsymbol{\Lambda}$ (which is a row vector of length p). We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k

Estimated Latent Variables: $\boldsymbol{Z}$

Estimated Noise Free Data: $\boldsymbol{\Lambda}\;\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i\;\boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a variational approach according to Girolami (2001) and Palmer et al. (2006).

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

The code is implemented in R , therefore it is *slow*.

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabi(X,10,0.3,1.0,1.0,3)



#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabi(X,200,0.4,1.0,1.0,13)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="FABI",Y=Y)

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)


resBreast <- fabi(X,200,0.1,1.0,1.0,5)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,ti="FABI Breast cancer(Veer)")
```

```
#sorting of predefined labels
CBreast%*%rBreast$pmZ

#---------------
# DEMO3
#---------------


data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabi(X,200,0.1,1.0,1.0,5)

rMulti <- extract_plot(X,resMulti$L,resMulti$Z,ti="FABI Multiple tissues(Su)")

#sorting of predefined labels
CMulti%*%rMulti$pmZ

#---------------
# DEMO4
#---------------


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabi(X,200,0.1,1.0,1.0,5)

rDLBCL <- extract_plot(X,resDLBCL$L,resDLBCL$Z,ti="FABI Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL%*%rDLBCL$pmZ
```

A.4 fabia

Factor Analysis for Bicluster Acquisition: Laplace Prior (FABIA).

C implementation of `fabia`.

1. Usage: `fabia(X,cyc,alpha,spl,spz,p,random=NULL,scale=0.0,norm=1,center=2,lap=1.0)`
2. Arguments:

- X: the data matrix.
- cyc: number of cycles to run.
- alpha: sparseness loadings (0.1 - 1.0).
- spl: sparseness prior loadings (0.5 - 4.0).
- spz: sparseness factors (0.5 - 4.0).
- p: number of hidden factor = number of biclusters.
- random: random initialization of loadings in [-random,random] (if not given: half of the square root of variance of component).
- scale: loading vectors are scaled in each iteration to the given variance. zero (default) indicates that non scaling.
- norm: data normalization: default = 1 (yes), 0 (no).
- center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
- lap: minimal value of the variational parameter, default = 1.

3. Return values:

- LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
- L: Loadings: $\boldsymbol{\Lambda}$
- Z: Factors: $\boldsymbol{Z}$
- Psi: Noise variance: $\boldsymbol{\Psi}$
- lapla: Variational parameter

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i \ \boldsymbol{z}_i^T \ + \ \boldsymbol{\Upsilon} \ ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \ = \ \boldsymbol{\Lambda}\ \boldsymbol{Z} \ + \ \boldsymbol{\Upsilon} \ .$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \ + \ \boldsymbol{\epsilon} \ = \ \boldsymbol{\Lambda}\tilde{\boldsymbol{z}} \ + \ \boldsymbol{\epsilon}$$

The model assumptions are:

Factor Prior is Independent Laplace:

$$p(\tilde{\boldsymbol{z}}) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{p} \prod_{i=1}^{p} e^{-\sqrt{2}\,|z_i|}$$

Loading Prior is Independent Laplace:

$$p(\boldsymbol{\lambda}_i) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{n} \prod_{k=1}^{n} e^{-\sqrt{2}\,|\lambda_{ki}|}$$

Noise: Gaussian independent

$$p(\boldsymbol{\epsilon}) \;=\; \left(\frac{1}{\sqrt{2\,\pi}}\right)^{n} \prod_{k=1}^{n} \frac{1}{\sigma_k} e^{\sum_{k=1}^{n} \frac{\epsilon_k^2}{\sigma_k^2}}$$

Data Mean:

$$\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\boldsymbol{\Lambda}\;\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}) \;=\; \boldsymbol{\Lambda}\;\mathrm{E}(\tilde{\boldsymbol{z}}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0}$$

Data Covariance:

$$\begin{aligned} \mathrm{E}(\boldsymbol{x}\;\boldsymbol{x}^T) \;&=\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\;\boldsymbol{\epsilon}^T) \\ &=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2) \end{aligned}$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\boldsymbol{\lambda}^k\right)^T \boldsymbol{\lambda}^k \;=\; 1$$

Here $\boldsymbol{\lambda}^k$ is the k-th row of $\boldsymbol{\Lambda}$ (which is a row vector of length p). We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k

Estimated Latent Variables: $\boldsymbol{Z}$

Estimated Noise Free Data: $\boldsymbol{\Lambda}\;\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i\;\boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a variational approach according to Girolami (2001) and Palmer et al. (2006).

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

The code is implemented in C using the Rcpp package.

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]



resEx <- fabia(X,50,0.3,1.0,1.0,3)


#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,200,0.4,1.0,1.0,13)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="FABIA",Y=Y)

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,200,0.1,1.0,1.0,5)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,"FABIA Breast cancer(Veer)")

#sorting of predefined labels
```

```
CBreast%*%rBreast$pmZ

#---------------
# DEMO3
#---------------

data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabia(X,200,0.1,1.0,1.0,5)

rMulti <- extract_plot(X,resMulti$L,resMulti$Z,ti="FABIA Multiple tissues(Su)")

#sorting of predefined labels
CMulti%*%rMulti$pmZ

#---------------
# DEMO4
#---------------

data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabia(X,200,0.1,1.0,1.0,5)

rDLBCL <- extract_plot(X,resDLBCL$L,resDLBCL$Z,ti="FABIA Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL%*%rDLBCL$pmZ
```

A.5 fabiaVersion

Display version info for package and for FABIA.

1. Usage: `fabiaVersion()`

EXAMPLE:

```
fabiaVersion()
```

A.6 fabiap

Factor Analysis for Bicluster Acquisition: Post-Projection (FABIAP).

1. Usage: `fabiap(X,cyc,alpha,spl,spz,p,sL,sZ,random=NULL,scale=0.0, norm=1,center=2,lap=1.0)`
2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings (0.1 - 1.0).
   - spl: sparseness prior loadings (0.5 - 4.0).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
   - sL: final sparseness loadings.
   - sZ: final sparseness factors.
   - random: random initialization of loadings in [-random,random] (if not given: half of the square root of variance of component).
   - scale: loading vectors are scaled in each iteration to the given variance. zero (default) indicates that non scaling.
   - norm: data normalization: default = 1 (yes), 0 (no).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
   - lap: minimal value of the variational parameter, default = 1.
3. Return Values:
   - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
   - L: Loadings: $\boldsymbol{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\boldsymbol{\Psi}$
   - lapla: Variational parameter

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse. Post-processing by projecting the final results to a given sparseness criterion.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i \ \boldsymbol{z}_i^T \ + \ \boldsymbol{\Upsilon} \ ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \ = \ \boldsymbol{\Lambda}\ \boldsymbol{Z} \ + \ \boldsymbol{\Upsilon} \ .$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \;+\; \boldsymbol{\epsilon} \;=\; \boldsymbol{\Lambda}\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}$$

The model assumptions are:

Factor Prior is Independent Laplace:

$$p(\tilde{\boldsymbol{z}}) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{p} \prod_{i=1}^{p} e^{-\sqrt{2}\,|z_i|}$$

Loading Prior is Independent Laplace:

$$p(\boldsymbol{\lambda}_i) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{n} \prod_{k=1}^{n} e^{-\sqrt{2}\,|\lambda_{ki}|}$$

Noise: Gaussian independent

$$p(\boldsymbol{\epsilon}) \;=\; \left(\frac{1}{\sqrt{2\,\pi}}\right)^{n} \prod_{k=1}^{n} \frac{1}{\sigma_k} e^{\sum_{k=1}^{n} \frac{\epsilon_k^2}{\sigma_k^2}}$$

Data Mean:

$$\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\boldsymbol{\Lambda}\;\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}) \;=\; \boldsymbol{\Lambda}\;\mathrm{E}(\tilde{\boldsymbol{z}}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0}$$

Data Covariance:

$$\begin{aligned} \mathrm{E}(\boldsymbol{x}\;\boldsymbol{x}^T) \;&=\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\;\boldsymbol{\epsilon}^T) \\ &=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2) \end{aligned}$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\boldsymbol{\lambda}^k\right)^T \boldsymbol{\lambda}^k \;=\; 1$$

Here $\boldsymbol{\lambda}^k$ is the k-th row of $\boldsymbol{\Lambda}$ (which is a row vector of length p). We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k

Estimated Latent Variables: $\boldsymbol{Z}$

Estimated Noise Free Data: $\boldsymbol{\Lambda}\;\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a variational approach according to Girolami (2001) and Palmer et al. (2006).

We included a prior on the parameters and minimize a lower bound on the posterior of the parameters given the data. The update of the loadings includes an additive term which pushes the loadings toward zero (Gaussian prior leads to an multiplicative factor).

Post-processing: The final results of the loadings and the factors are projected to a sparse vector according to Hoyer, 2004: given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

The code is implemented in C using the Rcpp package. The projection is implemented in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabiap(X,50,0.3,1.0,1.0,3,0.7,0.7)


#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

resToy <- fabiap(X,200,0.4,1.0,1.0,13,0.7,0.7)
```

```
rToy <- extract_plot(X,resToy$L,resToy$Z,ti="FABIAP",Y=Y)

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabiap(X,200,0.1,1.0,1.0,5,0.5,0.3)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,ti="FABIAP Breast cancer(Veer)")

#sorting of predefined labels
CBreast%*%rBreast$pmZ

#---------------
# DEMO3
#---------------


data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabiap(X,200,0.1,1.0,1.0,5,0.5,0.3)

rMulti <- extract_plot(X,resMulti$L,resMulti$Z,ti="FABIAP Multiple tissues(Su)")

#sorting of predefined labels
CMulti%*%rMulti$pmZ

#---------------
# DEMO4
#---------------


data(DLBCL_B)

X <- as.matrix(XDLBCL)

resDLBCL <- fabiap(X,200,0.1,1.0,1.0,5,0.5,0.3)

rDLBCL <- extract_plot(X,resDLBCL$L,resDLBCL$Z,ti="FABIAP Lymphoma(Rosenwald)")
```

```
#sorting of predefined labels
CDLBCL%*%rDLBCL$pmZ
```

A.7 fabias

Factor Analysis for Bicluster Acquisition: Sparseness Projection (FABIAS).

C implementation of `fabias`.

1. Usage: `fabias(X,cyc,alpha,spz,p,random=NULL,norm=1,center=2,lap=1.0)`
2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings via projection (0.1 - 0.9).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
   - random: random initialization of loadings in [-random,random] (if not given: half of the square root of variance of component).
   - norm: data normalization: default = 1 (yes), 0 (no).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
   - lap: minimal value of the variational parameter, default = 1.
3. Return Values:
   - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
   - L: Loadings: $\boldsymbol{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\boldsymbol{\Psi}$
   - lapla: Variational parameter

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i \ \boldsymbol{z}_i^T \ + \ \boldsymbol{\Upsilon} \ ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \ = \ \boldsymbol{\Lambda}\ \boldsymbol{Z} \ + \ \boldsymbol{\Upsilon} \ .$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \;+\; \boldsymbol{\epsilon} \;=\; \boldsymbol{\Lambda}\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}$$

The model assumptions are:

Factor Prior is Independent Laplace:

$$p(\tilde{\boldsymbol{z}}) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{p} \prod_{i=1}^{p} e^{-\sqrt{2}\,|z_i|}$$

Loading Prior has Finite Support:

$$p(\boldsymbol{\lambda}_i) \;=\; c \quad \text{for} \quad \|\boldsymbol{\lambda}_i\|_1 \;\leq\; k$$

$$p(\boldsymbol{\lambda}_i) \;=\; 0 \quad \text{for} \quad \|\boldsymbol{\lambda}_i\|_1 \;>\; k$$

Noise: Gaussian independent

$$p(\boldsymbol{\epsilon}) \;=\; \left(\frac{1}{\sqrt{2\,\pi}}\right)^{n} \prod_{k=1}^{n} \frac{1}{\sigma_k} e^{\sum_{k=1}^{n} \frac{\epsilon_k^2}{\sigma_k^2}}$$

Data Mean:

$$\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\boldsymbol{\Lambda}\;\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}) \;=\; \boldsymbol{\Lambda}\;\mathrm{E}(\tilde{\boldsymbol{z}}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0}$$

Data Covariance:

$$\begin{aligned} \mathrm{E}(\boldsymbol{x}\;\boldsymbol{x}^T) \;&=\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\;\boldsymbol{\epsilon}^T) \\ &=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2) \end{aligned}$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\boldsymbol{\lambda}^k\right)^T \boldsymbol{\lambda}^k \;=\; 1$$

Here $\boldsymbol{\lambda}^k$ is the k-th row of $\boldsymbol{\Lambda}$ (which is a row vector of length p). We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k

Estimated Latent Variables: $\boldsymbol{Z}$

Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i\ \boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a variational approach according to Girolami (2001) and Palmer et al. (2006).

The prior has finite support, therefore after each update of the loadings they are projected to the finite support. The projection is done according to Hoyer (2004): given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\text{sparseness}(\boldsymbol{\lambda}_i) \ = \ \frac{\sqrt{n} \ - \ \sum_{k=1}^{n} |\lambda_{ki}| \ / \ \sum_{k=1}^{n} \lambda_{ki}^2}{\sqrt{n} \ - \ 1}$$

The code is implemented in C using the Rcpp package.

EXAMPLE:

```
#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabias(X,200,0.8,1.0,13)

rToy <- extract_plot(X,resToy$L,resToy$Z,"FABIAS",Y=Y)

#---------------
# DEMO2
#---------------


data(Breast_A)

X <- as.matrix(XBreast)
```

```
resBreast <- fabias(X,300,0.6,1.0,3)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,"FABIAS Breast cancer(Veer)")

#sorting of predefined labels
CBreast%*%rBreast$pmZ

#---------------
# DEMO3
#---------------


data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabias(X,200,0.8,1.0,4)

rMulti <- extract_plot(X,resMulti$L,resMulti$Z,"FABIAS Multiple tissues(Su)")

#sorting of predefined labels
CMulti%*%rMulti$pmZ

#---------------
# DEMO4
#---------------


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabias(X,200,0.8,1.0,3)

rDLBCL <- extract_plot(X,resDLBCL$L,resDLBCL$Z,"FABIAS Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL%*%rDLBCL$pmZ
```

A.8 fabiasp

Factor Analysis for Bicluster Acquisition: Sparseness Projection (FABIASP).

R implementation of `fabias`, therefore it is *slow*.

1. Usage: `fabiasp(X,cyc,alpha,spz,p,norm=1,center=2)`
2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings via projection (0.1 - 0.9).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
   - norm: data normalization: default = 1 (yes), 0 (no).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
3. Return Values:
   - LZ: Estimated Noise Free Data: $\mathbf{\Lambda}\ \boldsymbol{Z}$
   - L: Loadings: $\mathbf{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\mathbf{\Psi}$
   - lapla: Variational parameter

Biclusters are found by sparse factor analysis where *both* the factors and the loadings are sparse.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i \ \boldsymbol{z}_i^T \ + \ \mathbf{\Upsilon} \ ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \ = \ \mathbf{\Lambda}\ \boldsymbol{Z} \ + \ \mathbf{\Upsilon} \ .$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \ = \ \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \ + \ \boldsymbol{\epsilon} \ = \ \mathbf{\Lambda}\tilde{\boldsymbol{z}} \ + \ \boldsymbol{\epsilon}$$

The model assumptions are:

Factor Prior is Independent Laplace:

$$p(\boldsymbol{z}) \ = \ \left(\frac{1}{\sqrt{2}}\right)^p \prod_{i=1}^{p} e^{-\sqrt{2}\,|z_i|}$$

Loading Prior has Finite Support:

$$p(\boldsymbol{\lambda}_i) \;=\; c \quad \text{for} \quad \|\boldsymbol{\lambda}_i\|_1 \;\leq\; k$$

$$p(\boldsymbol{\lambda}_i) \;=\; 0 \quad \text{for} \quad \|\boldsymbol{\lambda}_i\|_1 \;>\; k$$

Noise: Gaussian independent

$$p(\boldsymbol{\epsilon}) \;=\; \left(\frac{1}{\sqrt{2\,\pi}}\right)^n \prod_{k=1}^{n} \frac{1}{\sigma_k} e^{\sum_{k=1}^{n} \frac{\epsilon_k^2}{\sigma_k^2}}$$

Data Mean:

$$\mathrm{E}(\boldsymbol{x}) \;=\; \mathrm{E}(\boldsymbol{\Lambda}\;\tilde{\boldsymbol{z}} \;+\; \boldsymbol{\epsilon}) \;=\; \boldsymbol{\Lambda}\;\mathrm{E}(\tilde{\boldsymbol{z}}) \;+\; \mathrm{E}(\boldsymbol{\epsilon}) \;=\; \mathbf{0}$$

Data Covariance:

$$\begin{aligned}\mathrm{E}(\boldsymbol{x}\;\boldsymbol{x}^T) \;&=\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}}\tilde{\boldsymbol{z}}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\tilde{\boldsymbol{z}})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\;\boldsymbol{\epsilon}^T) \\ &=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2)\end{aligned}$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\boldsymbol{\lambda}^k\right)^T \boldsymbol{\lambda}^k \;=\; 1$$

Here $\boldsymbol{\lambda}^k$ is the k-th row of $\boldsymbol{\Lambda}$ (which is a row vector of length p). We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k

Estimated Latent Variables: $\boldsymbol{Z}$

Estimated Noise Free Data: $\boldsymbol{\Lambda}\;\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i\;\boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a variational approach according to Girolami (2001) and Palmer et al. (2006).

The prior has finite support, therefore after each update of the loadings they are projected to the finite support. The projection is done according to Hoyer (2004): given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\mathrm{sparseness}(\boldsymbol{\lambda}_i) \;=\; \frac{\sqrt{n} \;-\; \sum_{k=1}^{n} |\lambda_{ki}| \;/\; \sum_{k=1}^{n} \lambda_{ki}^2}{\sqrt{n} \;-\; 1}$$

The code is implemented in R , therefore it is *slow*.

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]




resEx <- fabiasp(X,50,0.8,1.0,3)


\dontrun{
#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabiasp(X,200,0.6,1.0,13)

rToy <- extract_plot(X,resToy$L,resToy$Z,"ti=FABIASP",Y=Y)

#---------------
# DEMO2
#---------------


data(Breast_A)
```

```
X <- as.matrix(XBreast)

resBreast <- fabiasp(X,200,0.4,1.0,5)

rBreast <- extract_plot(X,resBreast$L,resBreast$Z,ti="FABIASP Breast cancer(Veer)")

#sorting of predefined labels
CBreast%*%rBreast$pmZ

#---------------
# DEMO3
#---------------


data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabiasp(X,200,0.4,1.0,5)

rMulti <- extract_plot(X,resMulti$L,resMulti$Z,"ti=FABIASP Multiple tissues(Su)")

#sorting of predefined labels
CMulti%*%rMulti$pmZ

#---------------
# DEMO4
#---------------


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL <- fabiasp(X,200,0.6,1.0,5)

rDLBCL <- extract_plot(X,resDLBCL$L,resDLBCL$Z,ti="FABIASP Lymphoma(Rosenwald)")

#sorting of predefined labels
CDLBCL%*%rDLBCL$pmZ
```

A.9 make_fabi_data

Generation of bicluster data.

1. Usage: `make_fabi_data(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,mean_z, sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
   - n. number of observations.
   - l: number of samples.
   - p: number of biclusters.
   - f1: $l/f1$ max. additional samples are active in a bicluster.
   - f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
   - of1: minimal active samples in a bicluster.
   - of2: menial observations that form a pattern in a bicluster.
   - sd_noise: Gaussian zero mean noise std on data matrix.
   - sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
   - mean_z: Gaussian mean for activated factors.
   - sd_z: Gaussian std for activated factors.
   - sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
   - mean_l: Gaussian mean for observation patterns.
   - sd_l: Gaussian std for observation patterns.
3. Return values:
   - X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.
   - Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
   - ZC: list where ith element gives samples belonging to ith bicluster.
   - LC: list where ith element gives observations belonging to ith bicluster.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \,,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda} \, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon} \,.$$

Here the $\boldsymbol{\lambda}_i$ are from $\mathbb{R}^n$, the $\boldsymbol{z}_i$ from R^l, and both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are from $\mathbb{R}^{n \times l}$.

Sequentially $\boldsymbol{\lambda}_i$ are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are

added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. The sign of the mean is randomly chosen for each component.

Sequentially z_i are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

Υ is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

dat <- make_fabi_data(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

myImagePlot(Y)
x11()
myImagePlot(X)
```

A.10 make_fabi_data_pos

Generation of bicluster data.

1. Usage: `make_fabi_data_pos(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise,`
   `mean_z,sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
   - n. number of observations.
   - l: number of samples.
   - p: number of biclusters.
   - f1: $l/f1$ max. additional samples are active in a bicluster.

- f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
- of1: minimal active samples in a bicluster.
- of2: menial observations that form a pattern in a bicluster.
- sd_noise: Gaussian zero mean noise std on data matrix.
- sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
- mean_z: Gaussian mean for activated factors.
- sd_z: Gaussian std for activated factors.
- sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
- mean_l: Gaussian mean for observation patterns.
- sd_l: Gaussian std for observation patterns.

3. Return values:

- X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.
- Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
- ZC: list where ith element gives samples belonging to ith bicluster.
- LC: list where ith element gives observations belonging to ith bicluster.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda} \, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon} \;.$$

Here the $\boldsymbol{\lambda}_i$ are from $\mathbb{R}^n$, the $\boldsymbol{z}_i$ from R^l, and both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are from $\mathbb{R}^{n \times l}$.

Sequentially $\boldsymbol{\lambda}_i$ are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. "POS": The sign of the mean is fixed.

Sequentially $\boldsymbol{z}_i$ are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

$\boldsymbol{\Upsilon}$ is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

dat <- make_fabi_data_pos(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

myImagePlot(Y)
x11()
myImagePlot(X)
```

A.11 make_fabi_data_blocks

Generation of bicluster data with bicluster blocks.

1. Usage: `make_fabi_data_blocks(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise, mean_z,sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
   - n: number of observations.
   - l: number of samples.
   - p: number of biclusters.
   - f1: $l/f1$ max. additional samples are active in a bicluster.
   - f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
   - of1: minimal active samples in a bicluster.
   - of2: minimal observations that form a pattern in a bicluster.
   - sd_noise: Gaussian zero mean noise std on data matrix.
   - sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
   - mean_z: Gaussian mean for activated factors.
   - sd_z: Gaussian std for activated factors.
   - sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
   - mean_l: Gaussian mean for observation patterns.
   - sd_l: Gaussian std for observation patterns.
3. Return Values:
   - X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.

- Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
- ZC: list where ith element gives samples belonging to ith bicluster.
- LC: list where ith element gives observations belonging to ith bicluster.

Bicluster data is generated for visualization because the biclusters are now in block format. That means observations and samples that belong to a bicluster are consecutive. This allows visual inspection because the use can identify blocks and whether they have been found or reconstructed.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \; = \; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \; + \; \boldsymbol{\Upsilon} \; ,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \; = \; \boldsymbol{\Lambda} \, \boldsymbol{Z} \; + \; \boldsymbol{\Upsilon} \; .$$

Here the $\boldsymbol{\lambda}_i$ are from $\mathbb{R}^n$, the $\boldsymbol{z}_i$ from R^l, and both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are from $\mathbb{R}^{n \times l}$.

Sequentially $\boldsymbol{\lambda}_i$ are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. The sign of the mean is randomly chosen for each component.

Sequentially $\boldsymbol{z}_i$ are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

$\boldsymbol{\Upsilon}$ is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

Y <- dat[[1]]
```

```
X <- dat[[2]]

myImagePlot(Y)
x11()
myImagePlot(X)
```

A.12 make_fabi_data_blocks_pos

Generation of bicluster data with bicluster blocks.

1. Usage: `make_fabi_data_blocks_pos(n,l,p,f1,f2,of1,of2,sd_noise, sd_z_noise,mean_z,sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
   - n: number of observations.
   - l: number of samples.
   - p: number of biclusters.
   - f1: $l/f1$ max. additional samples are active in a bicluster.
   - f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
   - of1: minimal active samples in a bicluster.
   - of2: minimal observations that form a pattern in a bicluster.
   - sd_noise: Gaussian zero mean noise std on data matrix.
   - sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
   - mean_z: Gaussian mean for activated factors.
   - sd_z: Gaussian std for activated factors.
   - sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
   - mean_l: Gaussian mean for observation patterns.
   - sd_l: Gaussian std for observation patterns.
3. Return Values:
   - X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.
   - Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
   - ZC: list where ith element gives samples belonging to ith bicluster.
   - LC: list where ith element gives observations belonging to ith bicluster.

Bicluster data is generated for visualization because the biclusters are now in block format. That means observations and samples that belong to a bicluster are consecutive. This allows visual inspection because the use can identify blocks and whether they have been found or reconstructed.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda} \, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon} \;.$$

Here the $\boldsymbol{\lambda}_i$ are from $\mathbb{R}^n$, the $\boldsymbol{z}_i$ from R^l, and both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are from $\mathbb{R}^{n \times l}$.

Sequentially $\boldsymbol{\lambda}_i$ are generated using `n`, `f2`, `of2`, `sd_l_noise`, `mean_l`, `sd_l`. `of2` gives the minimal observations participating in a bicluster to which between 0 and $n/f2$ observations are added, where the number is uniformly chosen. `sd_l_noise` gives the noise of observations not participating in the bicluster. `mean_l` and `sd_l` determines the Gaussian from which the values are drawn for the observations that participate in the bicluster. "POS": The sign of the mean is fixed.

Sequentially $\boldsymbol{z}_i$ are generated using `l`, `f1`, `of1`, `sd_z_noise`, `mean_z`, `sd_z`. `of1` gives the minimal samples participating in a bicluster to which between 0 and $l/f1$ samples are added, where the number is uniformly chosen. `sd_z_noise` gives the noise of samples not participating in the bicluster. `mean_z` and `sd_z` determines the Gaussian from which the values are drawn for the samples that participate in the bicluster.

$\boldsymbol{\Upsilon}$ is the overall Gaussian zero mean noise generated by `sd_noise`.

Implementation in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks_pos(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

Y <- dat[[1]]
X <- dat[[2]]

myImagePlot(Y)
x11()
myImagePlot(X)
```

A.13 mfsc

Sparse Matrix Factorization for bicluster analysis (MFSC).

1. Usage: `mfsc(X,p,sL,sZ,cyc=100,norm=1,center=2)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factor = number of biclusters.
   - sL: sparseness loadings.
   - sZ: sparseness factors.
   - cyc: maximal number of iterations.
   - norm: data normalization: default = 1 (yes), 0 (no).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default 2.
3. Return Values:
   - L: Left matrix: $\mathbf{\Lambda}$
   - Z: Right matrix: $\boldsymbol{Z}$

Biclusters are found by sparse matrix factorization where *both* factors are sparse.

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;+\; \boldsymbol{\Upsilon} \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \mathbf{\Lambda}\, \boldsymbol{Z} \;+\; \boldsymbol{\Upsilon} \;.$$

No noise assumption: In contrast to factor analysis there is no noise assumption.

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \;=\; \mathbf{\Lambda}\tilde{\boldsymbol{z}}$$

Estimated Parameters: $\mathbf{\Lambda}$ and $\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i \; \boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a constraint optimization according to Hoyer (2004). The Euclidean distance (the Frobenius norm) is minimized subject to sparseness constraints:

$$\min_{\boldsymbol{\Lambda},\boldsymbol{Z}} \|\boldsymbol{X} \; - \; \boldsymbol{\Lambda} \; \boldsymbol{Z}\|_F^2$$

$$\text{subject to} \;\; \|\boldsymbol{\Lambda}\|_F^2 \; = \; 1$$

$$\text{subject to} \;\; \|\boldsymbol{\Lambda}\|_1 \; = \; k_L$$

$$\text{subject to} \;\; \|\boldsymbol{Z}\|_F^2 \; = \; 1$$

$$\text{subject to} \;\; \|\boldsymbol{Z}\|_1 \; = \; k_Z$$

Model selection is done by gradient descent on the Euclidean objective and thereafter projection of single vectors of $\boldsymbol{\Lambda}$ and single vectors of $\boldsymbol{Z}$ to fulfill the sparseness constraints.

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\text{sparseness}(\boldsymbol{\lambda}_i) \; = \; \frac{\sqrt{n} \; - \; \sum_{k=1}^{n} |\lambda_{ki}| \; / \; \sum_{k=1}^{n} \lambda_{ki}^2}{\sqrt{n} \; - \; 1}$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
resEx <- mfsc(as.matrix(abs(X)),3,0.7,0.7)


#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- mfsc(as.matrix(abs(X)),8,0.7,0.7)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="NMFSC",Y=Y)
```

A.14 myImagePlot

Plotting of a matrix.

1. Usage: `myImagePlot(x,xLabels=NULL, yLabels=NULL, zlim=NULL, title=NULL)`
2. Arguments:
   - x: the matrix.
   - xLabels: vector of strings to label the rows (default "rownames(x)").
   - yLabels: vector of strings to label the columns (default "colnames(x)").
   - zlim: vector containing a low and high value to use for the color scale.
   - title: title of the plot.

   Plotting a table of numbers as an image using R .

   The color scale is based on the highest and lowest values in the matrix.

   Program has been obtained by `http://www.phaget4.org/R/myImagePlot.R`

EXAMPLE:

```
#---------------
# DEMO
```

```
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]

myImagePlot(X)
```

A.15 PlotBicluster

Plots a bicluster.

1. Usage: `PlotBicluster(x,samples,observations,xLabels=NULL, yLabels=NULL, zlim=NULL, title=NULL,x11b=TRUE)`
2. Arguments:
   - x: data matrix with columns as samples and rows as observations.
   - samples: samples belonging to the bicluster.
   - observations: observations belonging to the bicluster.
   - xLabels: vector of strings to label the columns where "samples" are a subset (default "colnames(x)").
   - yLabels: vector of strings to label the rows where "observations" are a subset (default "rownames(x)").
   - zlim: vector containing a low and high value to use for the color scale.
   - title: title of the plot.
   - x11b: screen output or not.

   Plots a bicluster.

   Plot1: The data matrix is sorted such that the bicluster appear at the upper left corner.

   The bicluster is marked by a rectangle.

   Plot2: Only the bicluster is presented.

   Implementation in R .

```
#---------------
# TEST
#---------------
```

```
dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resEx <- fabia(X,20,0.1,1.0,1.0,3)

rEx <- extract_bic(resEx$L,resEx$Z,lapla=resEx$lapla,Psi=resEx$Psi)

PlotBicluster(X,unlist(rEx$bic[1,5]),unlist(rEx$bic[1,3]),x11b=FALSE)


#---------------
# DEMO1
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]


resToy <- fabia(X,200,0.4,1.0,1.0,13)



rToy <- extract_bic(resToy$L,resToy$Z,lapla=resToy$lapla,Psi=resToy$Psi)

PlotBicluster(X,unlist(rToy$bic[1,5]),unlist(rToy$bic[1,3]))

#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabia(X,200,0.1,1.0,1.0,5)
```

```
rBreast <- extract_bic(resBreast$L,resBreast$Z,lapla=resBreast$lapla,Psi=resBreast$Psi)

PlotBicluster(X,unlist(rBreast$bic[1,5]),unlist(rBreast$bic[1,3]))
```

A.16 nmfdiv

Non-negative Matrix Factorization with Kullaback-Leibler divergence as objective.

1. Usage: `nmfdiv(X,p,cyc=100)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factor.
   - cyc: maximal number of iterations.
3. Return Values:
   - L: Left matrix: $\mathbf{\Lambda}$
   - Z: Right matrix: $\boldsymbol{Z}$

$$\boldsymbol{X} \;=\; \mathbf{\Lambda}\, \boldsymbol{Z}$$

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i\, \boldsymbol{z}_i^T$$

Estimated Parameters: $\mathbf{\Lambda}$ and $\boldsymbol{Z}$

The model selection is performed according to Lee and Seung (1999, 2001).

objective:

$$\mathrm{D}(\boldsymbol{A} \parallel \boldsymbol{B}) \;=\; \sum_{ij} \left(A_{ij}\, \log \frac{A_{ij}}{B_{ij}} \;+\; A_{ij} \;-\; B_{ij} \right)$$

update:

$$L_{ik} \;=\; L_{ik} \frac{\sum_{j=1}^{n} Z_{ji}\, V_{jk} \;/\; (\mathbf{\Lambda}\, \boldsymbol{Z})_{jk}}{\sum_{j=1}^{n} Z_{ji}}$$

$$Z_{ji} \;=\; Z_{ji} \frac{\sum_{k=1}^{l} L_{ik}\, V_{jk} \;/\; (\boldsymbol{\Lambda}\, \boldsymbol{Z})_{jk}}{\sum_{k=1}^{l} L_{ik}}$$

or in matrix notation with "$*$" and "$/$" as element-wise operators:

$$\boldsymbol{\Lambda} \;=\; \boldsymbol{\Lambda} * ((\boldsymbol{X} \;/\; (\boldsymbol{\Lambda}\, \boldsymbol{Z}))\, t(\boldsymbol{Z})) \;/\; \mathrm{rowSums}(\boldsymbol{Z})$$

$$\boldsymbol{Z} \;=\; \boldsymbol{Z} * (t(\boldsymbol{\Lambda})\, (\boldsymbol{X} \;/\; (\boldsymbol{\Lambda}\, \boldsymbol{Z}))) \;/\; \mathrm{colSums}(\boldsymbol{\Lambda})$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resEx <- nmfdiv(as.matrix(abs(X)),3)


#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
```

```
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resToy <- nmfdiv(as.matrix(abs(X)),8)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="NMFDIV",Y=Y)
```

A.17 nmfeu

Non-negative Matrix Factorization with Euclidean distance as objective.

1. Usage: `nmfeu(X,p,cyc=100)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factor.
   - cyc: maximal number of iterations.
3. Return Values:
   - L: Left matrix: $\boldsymbol{\Lambda}$
   - Z: Right matrix: $\boldsymbol{Z}$

$$\boldsymbol{X} \;=\; \boldsymbol{\Lambda}\, \boldsymbol{Z}$$

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i\, \boldsymbol{z}_i^T$$

Estimated Parameters: $\boldsymbol{\Lambda}$ and $\boldsymbol{Z}$

The model selection is performed according to Lee and Seung (2001); Paatero and Tapper (1997).

objective:

$$\|\boldsymbol{A} \;-\; \boldsymbol{B}\|_F^2 \;=\; \sum_{ij} (A_{ij} \;-\; B_{ij})^2$$

update:

$$L_{ik} \;=\; L_{ik} \frac{\left(\boldsymbol{\Lambda}^T\, \boldsymbol{X}\right)_{ik}}{\left(\boldsymbol{\Lambda}^T\, \boldsymbol{\Lambda}\, \boldsymbol{Z}\right)_{ik}}$$

$$Z_{ji} \;=\; Z_{ji} \frac{\left(\boldsymbol{X}\, \boldsymbol{Z}^T\right)_{ji}}{\left(\boldsymbol{\Lambda}\, \boldsymbol{Z}\, \boldsymbol{Z}^T\right)_{ji}}$$

or in matrix notation with "$*$" and "$/$" as element-wise operators:

$$\boldsymbol{Z} \;=\; \boldsymbol{Z} * \left(t(\boldsymbol{\Lambda})\, \boldsymbol{X}\right) \,/\, \left(t(\boldsymbol{\Lambda})\, \boldsymbol{\Lambda}\, \boldsymbol{Z}\right)$$

$$\boldsymbol{\Lambda} \;=\; \boldsymbol{\Lambda} * \left(\boldsymbol{X}\, t(\boldsymbol{Z})\right) \,/\, \left(\boldsymbol{\Lambda}\, \boldsymbol{Z}\, t(\boldsymbol{Z})\right)$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resEx <- nmfeu(as.matrix(abs(X)),3)


#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
```

```
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resToy <- nmfeu(as.matrix(abs(X)),8)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="NMFEU",Y=Y)
```

A.18 nmfsc

Non-negative Sparse Matrix Factorization with sparseness constraints.

1. Usage: `nmfsc(X,p,sL,sZ,cyc=100)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factor = number of biclusters.
   - sL: sparseness loadings.
   - sZ: sparseness factors.
   - cyc: maximal number of iterations.
3. Return Values:
   - L: Left matrix: $\mathbf{\Lambda}$
   - Z: Right matrix: $\boldsymbol{Z}$

Essentially the model is the sum of outer products of vectors:

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i \, \boldsymbol{z}_i^T \;,$$

where the number of summands p is the number of biclusters. The matrix factorization is

$$\boldsymbol{X} \;=\; \mathbf{\Lambda} \, \boldsymbol{Z} \;.$$

If the nonzero components of the sparse vectors are grouped together then the outer product results in a matrix with a nonzero block and zeros elsewhere.

For a single data vector $\boldsymbol{x}$ that is

$$\boldsymbol{x} \;=\; \sum_{i=1}^{p} \boldsymbol{\lambda}_i z_i \;=\; \boldsymbol{\Lambda z}$$

Estimated Parameters: $\boldsymbol{\Lambda}$ and $\boldsymbol{Z}$

Estimated Biclusters: $\boldsymbol{\lambda}_i\ \boldsymbol{z}_i^T$ Larges values give the bicluster (ideal the nonzero values).

The model selection is performed by a constraint optimization according to Hoyer (2004). The Euclidean distance (the Frobenius norm) is minimized subject to sparseness and non-negativity constraints:

$$\min_{\boldsymbol{\Lambda},\boldsymbol{Z}} \|\boldsymbol{x} \;-\; \boldsymbol{\Lambda}\ \boldsymbol{Z}\|_F^2$$

$$\text{subject to}\ \ \|\boldsymbol{\Lambda}\|_F^2 \;=\; 1$$

$$\text{subject to}\ \ \|\boldsymbol{\Lambda}\|_1 \;=\; k_L$$

$$\text{subject to}\ \ \boldsymbol{\Lambda} \;\geq\; \boldsymbol{0}$$

$$\text{subject to}\ \ \|\boldsymbol{Z}\|_F^2 \;=\; 1$$

$$\text{subject to}\ \ \|\boldsymbol{Z}\|_1 \;=\; k_Z$$

$$\text{subject to}\ \ \boldsymbol{Z} \;\geq\; \boldsymbol{0}$$

Model selection is done by gradient descent on the Euclidean objective and thereafter projection of single vectors of $\boldsymbol{\Lambda}$ and single vectors of $\boldsymbol{Z}$ to fulfill the sparseness and non-negativity constraints.

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm and enforcing non-negativity.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero. Instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\text{sparseness}(\boldsymbol{\lambda}_i) \;=\; \frac{\sqrt{n} \;-\; \sum_{k=1}^{n} |\lambda_{ki}| \;/\; \sum_{k=1}^{n} \lambda_{ki}^2}{\sqrt{n} \;-\; 1}$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# TEST
#---------------

dat <- make_fabi_data_blocks(n = 100,l= 50,p = 3,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resEx <- nmfsc(as.matrix(abs(X)),3,0.7,0.7)


#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks(n = 1000,l= 100,p = 10,f1 = 5,f2 = 5,
  of1 = 5,of2 = 10,sd_noise = 3.0,sd_z_noise = 0.2,mean_z = 2.0,
  sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

X <- dat[[1]]
Y <- dat[[2]]
X <- abs(X)
XX <- tcrossprod(X)
dXX <- 1/sqrt(diag(XX))
X <- dXX*X


resToy <- nmfsc(as.matrix(abs(X)),8,0.7,0.7)

rToy <- extract_plot(X,resToy$L,resToy$Z,ti="NMFSC",Y=Y)
```

A.19 nprojfunc

Projection of a vector to a sparse non-negative vector with given sparseness and given l_2-norm.

1. Usage: `nprojfunc(s, k1, k2)`

2. Arguments:
   - s: data vector.
   - k1: sparseness, l_1 norm constraint.
   - k2: l_2 norm constraint.
3. Return Values:
   - v: non-negative sparse projected vector.

The projection minimize the Euclidean distance to the original vector given an l_1-norm and an l_2-norm and enforcing non-negativity.

The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero.

In the applications, instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\text{sparseness}(\boldsymbol{v}) \;=\; \frac{\sqrt{l} \;-\; \sum_{j=1}^{l} |v_j| \;/\; \sum_{j=1}^{l} v_j^2}{\sqrt{l} \;-\; 1}$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

size <- 30
sparseness <- 0.7

s <- as.vector(rnorm(size))
sp <- sqrt(1.0*size)-(sqrt(1.0*size)-1.0)*sparseness

ss <- nprojfunc(s,k1=sp,k2=1)

s
ss
```

A.20 projfunc

Projection of a vector to a sparse vector with given sparseness and given l_2-norm.

1. Usage: `projfunc(s, k1, k2)`
2. Arguments:
   - s: data vector.
   - k1: sparseness, l_1 norm constraint.
   - k2: l_2 norm constraint.
3. Return Values:
   - v: sparse projected vector.

The projection is done according to Hoyer (2004): given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector. The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero.

In the applications, instead of the l_1-norm a sparseness measurement is used which relates the l_1-norm to the l_2-norm:

$$\text{sparseness}(\boldsymbol{v}) \;=\; \frac{\sqrt{l} \;-\; \sum_{j=1}^{l} |v_j| \;/\; \sum_{j=1}^{l} v_j^2}{\sqrt{l} \;-\; 1}$$

The code is implemented in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

size <- 30
sparseness <- 0.7

s <- as.vector(rnorm(size))
sp <- sqrt(1.0*size)-(sqrt(1.0*size)-1.0)*sparseness

ss <- projfunc(s,k1=sp,k2=1)

s
ss
```

A.21 estimateMode

Estimation of the modes of the rows of a matrix.

1. Usage: `estimateMode(X,maxiter=50,tol=0.001,alpha=0.1,a1=4.0,G1=FALSE)`
2. Arguments:
   - X: matrix of which the modes of the rows are estimated.
   - maxiter: maximal number of iterations, default 50.
   - tol: tolerance for stopping, default 0.001.
   - alpha: learning rate, default 0.1.
   - a1: parameter of the width of the given distribution, default 4.
   - G1: kind of distribution, TRUE: $\frac{1}{a_1}\ln(\cosh(a1x))$, FALSE: $-\frac{1}{a_1}\exp(-\frac{a_1}{2}x^2)$, default FALSE.
3. Return Values:
   - u: the vector of estimated modes.
   - xu: $X - u$ the mode centered data.

The mode is estimated by contrast functions G1 $\frac{1}{a_1}\ln(\cosh(a1x))$ or G2 $-\frac{1}{a_1}\exp(-\frac{a_1}{2}x^2)$. The estimation is performed by gradient descent initialized by the median.

Implementation in R .

EXAMPLE:

```
#---------------
# DEMO
#---------------

dat <- make_fabi_data_blocks_pos(n = 100,l= 50,p = 10,f1 = 5,f2 = 5,
       of1 = 5,of2 = 10,sd_noise = 2.0,sd_z_noise = 0.2,mean_z = 2.0,
       sd_z = 1.0,sd_l_noise = 0.2,mean_l = 3.0,sd_l = 1.0)

     X <- dat[[1]]

modes <- estimateMode(X)

modes$u - apply(X, 1, median)
%$
```

B Data Sets

B.1 Breast_A

Microarray data set of van't Veer breast cancer.

Microarray data from Broad Institute "Cancer Program Data Sets" which was produced by van't Veer et al. (2002) (`http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`) Array S54 was removed because it is an outlier.

Goal was to find a gene signature to predict the outcome of a cancer therapy that is to predict whether metastasis will occur. A 70 gene signature has been discovered.

Here we want to find subclasses in the data set.

Hoshida et al. (2007) found 3 subclasses and verified that 50/61 cases from class 1 and 2 were ER positive and only in 3/36 from class 3.

`XBreast` is the data set with 97 samples and 1213 genes, `CBreast` give the three subclasses from Hoshida et al. (2007).

B.2 DLBCL_B

Microarray data set of Rosenwald diffuse large-B-cell lymphoma.

Microarray data from Broad Institute "Cancer Program Data Sets" which was produced by Rosenwald et al. (2002) (`http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`)

Goal was to predict the survival after chemotherapy

Hoshida et al. (2007) divided the data set into three classes:

- OxPhos: oxidative phosphorylation
- BCR: B-cell response
- HR: host response

We want to identify these subclasses.

The data has 180 samples and 661 probe sets (genes).

`XDLBCL` is the data set with 180 samples and 661 genes, `CDLBCL` give the three subclasses from Hoshida et al. (2007).

B.3 Multi_A

Microarray data set of Su on different mammalian tissue types.

Microarray data from Broad Institute "Cancer Program Data Sets" which was produced by Su et al. (2002) (`http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`)

Gene expression from human and mouse samples across a diverse array of tissues, organs, and cell lines have been profiled. The goal was to have a reference for the normal mammalian transcriptome.

Here we want to identify the subclasses which correspond to the tissue types.

The data has 102 samples and 5565 probe sets (genes).

`XMulti` is the data set with 102 samples and 5565 genes, `CMulti` give the four subclasses corresponding to the tissue types.

References

P. S. Bithas, N. C. Sagias, T. A. Tsiftsis, and G. K. Karagiannidis. Distributions involving correlated generalized gamma variables. In *Proc. Int. Conf. on Applied Stochastic Models and Data Analysis*, volume 12, Chania, 2007.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *J. R. Stat. Soc. B Met.*, 39(1):1–22, 1977.

M. Girolami. A variational method for learning sparse and overcomplete representations. *Neural Comput.*, 13(11):2517–2532, 2001.

S. Hochreiter, D.-A. Clevert, and K. Obermayer. A new summarization method for Affymetrix probe level data. *Bioinformatics*, 22(8):943–949, 2006.

Y. Hoshida, J.-P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov. Subclass mapping: Identifying common subtypes in independent disease data sets. *PLoS ONE*, 2(11):e1195, 2007.

P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *J. Mach. Learn. Res.*, 5:1457–1469, 2004.

A. Hyvärinen and E. Oja. A fast fixed-point algorithm for independent component analysis. *Neural Comput.*, 9(7):1483–1492, 1999.

D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13*, pages 556–562, 2001.

D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

P. Paatero and U. Tapper. Least squares formulation of robust non-negative factor analysis. *Chemometr. Intell. Lab.*, 37:23–35, 1997.

J. Palmer, D. Wipf, K. Kreutz-Delgado, and B. Rao. Variational EM algorithms for non-Gaussian latent variable models. In *Advances in Neural Information Processing Systems 18*, pages 1059–1066, 2006.

A. Rosenwald, G. Wright, W. C. Chan, J. M. Connors, E. Campo, R. I. Fisher, R. D. Gascoyne, H. K. Muller-Hermelink, E. B. Smeland, J. M. Giltnane, E. M. Hurt, H. Zhao, L. Averett, L. Yang, W. H. Wilson, E. S. Jaffe, R. Simon, R. D. Klausner, J. Powell, P. L. Duffey, D. L. Longo, T. C. Greiner, D. D. Weisenburger, W. G. Sanger, B. J. Dave, J. C. Lynch, J. Vose, J. O. Armitage, E. Montserrat, A. L'opez-Guillermo, T. M. Grogan, T. P. Miller, M. LeBlanc, G. Ott, S. Kvaloy, J. Delabie, H. Holte, P. Krajci, T. Stokke, and L. M. Staudt. The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphoma. *New Engl. J. Med.*, 346:1937–1947, 2002.

D. Samperi. Building R packages that call C++ functions
Rcpp: R/C++ interface classes version 5.0. Vignette of a CRAN package, 2006. http://cran.r-project.org/web/packages/Rcpp/vignettes/RcppAPI.pdf.

A. I. Su, M. P. Cooke, K. A.Ching, Y. Hakak, J. R. Walker, T. Wiltshire, A. P. Orth, R. G. Vega, L. M. Sapinoso, A. Moqrich, A. Patapoutian, G. M. Hampton, P. G. Schultz, and J. B. Hogenesch. Large-scale analysis of the human and mouse transcriptomes. *P. Natl. Acad. Sci. USA*, 99(7):4465–4470, 2002.

W. Talloen, D.-A. Clevert, S. Hochreiter, D. Amaratunga, L. Bijnens, S. Kass, and H. W. H. Göhlmann. I/NI-calls for the exclusion of non-informative genes: a highly effective feature filtering tool for microarray data. *Bioinformatics*, 23(21):2897–2902, 2007.

L. J. van't Veer, H. Dai, M. J. van de Vijver, Y. D. He, A. A. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, 2002.