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 2.20.0, October 17, 2016

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 according to the FABIA method (**?**). It has been designed especially for microarray data sets, but can be used for other kinds of data sets as well.

Please visit for additional information the FABIA homepage http://www.bioinf.jku.at/software/fabia/fabia.html.

2 Getting Started: FABIA

First load the `fabia` package:

```
R> library(fabia)
```

The `fabia` package imports the package `methods` and the function `plot` from the package `graphics`.

2.1 Quick start : without outputs and plots

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.

Examples with outputs and plot can be found at http://www.bioinf.jku.at/software/fabia/fabia.html.

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)
   ```

4. Read the data file “datafile.csv”:

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

5. Select the model based on the data: 5 biclusters; sparseness 0.01; 500 cycles

```
R> res <- fabia(X,5,0.01,500)
```

6. Give summary:

```
R> summary(res)
```

7. Plot some statistics:

```
R> show(res)
```

8. Plot the factorization results:

```
R> extractPlot(res,ti="FABIA")
```

9. Plot the result as a biplot:

```
R> plot(res)
```

10. Extract biclusters:

```
R> rb <- extractBic(res)
```

11. Show information content of the biclusters:

```
R> res@avini
```

12. List bicluster 1:

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

13. List bicluster 2:

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

14. Show bicluster 3:

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

15. List bicluster 4:

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

16. List bicluster 5:

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

17. Plot bicluster 1:

```
R> plotBicluster(rb,1)
```

18. Plot bicluster 2:

```
R> plotBicluster(rb,2)
```

19. Plot bicluster 3:

```
R> plotBicluster(rb,3)
```

20. Plot bicluster 4:

```
R> plotBicluster(rb,4)
```

21. Plot bicluster 5:

```
R> plotBicluster(rb,5)
```

22. List opposite bicluster 1:

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

23. Plot opposite bicluster 1:

```
R> plotBicluster(rb,1,opp=TRUE)
```

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> n <- 1000
R> l <- 100
R> p <- 10
R> dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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 and annotate the data according to which biclusters the samples and genes belong:

```
R> X <- dat[[1]]
R> Y <- dat[[2]]
R> ZC <- dat[[3]]
R> LC <- dat[[4]]
R> gclab <- rep.int(0,l)
R> gllab <- rep.int(0,n)
R> clab <- as.character(1:l)
R> llab <- as.character(1:n)
R> for (i in 1:p){
+  for (j in ZC[i]){
+      clab[j] <- paste(as.character(i),"_",clab[j],sep="")
```

```
+  }
+  for (j in LC[i]){
+      llab[j] <- paste(as.character(i),"_",llab[j],sep="")
+  }
+  gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
+  gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
+ }
groups <- gclab
```

3. **FABIA:** perform `fabia` (sparseness by Laplace prior) to extract biclusters; 13 biclusters, sparseness 0.01 (Laplace), 400 cycles:

```
R> resToy1 <- fabia(X,13,0.01,400)
```

4. Plot some results:

```
R> extractPlot(resToy1,ti="FABIA",Y=Y)
```

5. Extract the biclusters:

```
R> raToy1 <- extractBic(resToy1)
```

6. Plot bicluster 1:

```
R> if ((raToy1$bic[[1]][1]>1) && (raToy1$bic[[1]][2])>1) {
+      plotBicluster(raToy1,1)
+  }
```

7. Plot bicluster 2:

```
R> if ((raToy1$bic[[2]][1]>1) && (raToy1$bic[[2]][2])>1) {
+ plotBicluster(raToy1,2)
+ }
```

8. Plot bicluster 3:

```
R> if ((raToy1$bic[[3]][1]>1) && (raToy1$bic[[3]][2])>1) {
+ plotBicluster(raToy1,3)
+ }
```

9. Plot bicluster 4:

```
R> if ((raToy1$bic[[4]][1]>1) && (raToy1$bic[[4]][2])>1) {
+ plotBicluster(raToy1,4)
+ }
```

10. Prepare biplots:

```
R> colnames(resToy1@X) <- clab
R> rownames(resToy1@X) <- llab
```

11. Biplot of bicluster 1 and 2:

```
R> plot(resToy1,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
```

12. Biplot of bicluster 1 and 3:

```
R> plot(resToy1,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
```

13. Biplot of bicluster 2 and 3:

```
R> plot(resToy1,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)
```

14. **FABIAS:** perform `fabias` (sparseness by projection) to extract biclusters; 13 biclusters, sparseness 0.6 (projection), 400 cycles:

```
R> resToy2 <- fabias(X,13,0.6,400)
```

15. Plot some results:

```
R> extractPlot(resToy2,ti="FABIAS",Y=Y)
```

16. Extract the biclusters:

```
R> raToy2 <- extractBic(resToy2)
```

17. Plot bicluster 1:

```
R> if ((raToy2$bic[[1]][1]>1) && (raToy2$bic[[1]][2])>1) {
+      plotBicluster(raToy2,1)
+  }
```

18. Plot bicluster 2:

```
R> if ((raToy2$bic[[2]][1]>1) && (raToy2$bic[[2]][2])>1) {
+ plotBicluster(raToy2,2)
+ }
```

19. Plot bicluster 3:

```
R> if ((raToy2$bic[[3]][1]>1) && (raToy2$bic[[3]][2])>1) {
+ plotBicluster(raToy2,3)
+ }
```

20. Plot bicluster 4:

```
R> if ((raToy2$bic[[4]][1]>1) && (raToy2$bic[[4]][2])>1) {
+ plotBicluster(raToy2,4)
+ }
```

21. Prepare biplots:

```
R> colnames(resToy2@X) <- clab
R> rownames(resToy2@X) <- llab
```

22. Biplot of bicluster 1 and 2:

```
R> plot(resToy2,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
```

23. Biplot of bicluster 1 and 3:

```
R> plot(resToy2,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
```

24. Biplot of bicluster 2 and 3:

```
R> plot(resToy2,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)
```

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 (**?**) 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 (**?**) 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 (**?**) diffuse large-B-cell lymphoma. (**?**) 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.

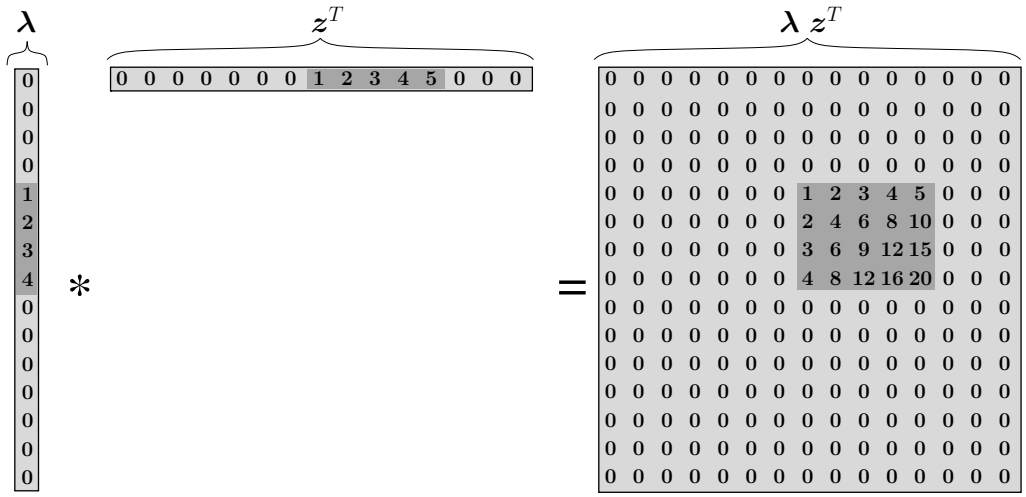

Figure 1: The outer product $\boldsymbol{\lambda}\, \boldsymbol{z}^T$ of two sparse vectors results in a matrix with a bicluster. Note, that the non-zero entries in the vectors are adjacent to each other for visualization purposes only.

3 The FABIA Model

For a detailed model description see 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. **??** 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. (**??**) formulates biclustering as sparse matrix factorization.

According to Eq. (**??**), 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. (**??**) 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 (**?**).

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 (**?**), 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 (**?**). For large values, this Bessel

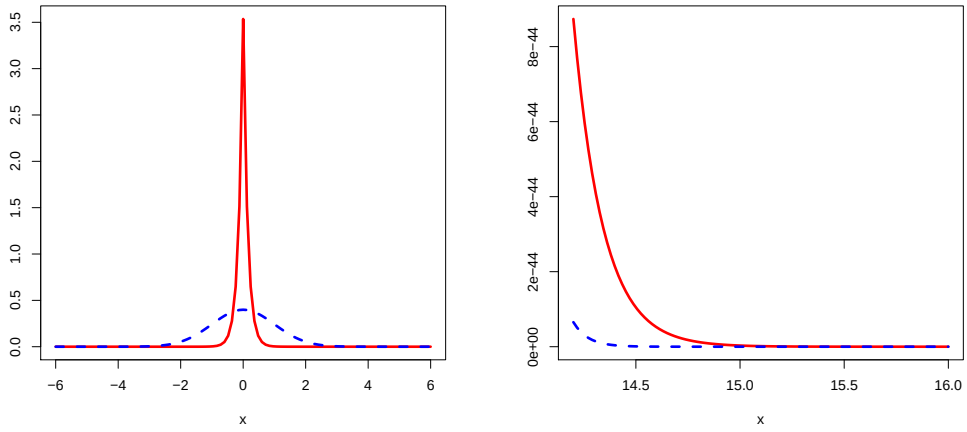

Figure 2: Left: The mode of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution. Right: The tails of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution

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.

The second model, called FABIAS, applies a prior with parameter spL on the loadings that has only support at sparse regions. Following (**?**), 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. **??** left shows the Laplacian mode compared to a Gaussian mode. The Laplacian mode is higher and narrower. Fig. **??** 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 $\mathbf{\Lambda}$ and $\mathbf{\Psi}$ can be estimated by Expectation-Maximization (EM; **?**). With a prior probability on the loadings, the a posteriori of the parameters is maximized like in (**??**).

3.3.1 Variational Approach for Sparse Factors

Model selection is not straightforward because the likelihood

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

cannot be computed analytically for a Laplacian prior $p(\tilde{\boldsymbol{z}})$. We employ a variational approach according to **?** and **?** 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 $\mathbf{\Lambda}$ and $\mathbf{\Psi}$ is maximized, but also with respect to $\boldsymbol{\xi}$.

In the following, $\mathbf{\Lambda}$ and $\mathbf{\Psi}$ denote the actual parameter estimates. According to **?** and **?**, we obtain

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

where $\mathbf{\Xi}_j$ 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

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

where

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

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. (**??**) hold. These constraints are ensured by a projection of $\boldsymbol{\lambda}_i$ after each $\boldsymbol{\Lambda}$ update according to **?**. 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 $\mathrm{diag}(\boldsymbol{\Psi}^{\mathrm{new}}) = \boldsymbol{\Psi}^{\mathrm{EM}}$ and project each updated prototype vector to a sparse vector with sparseness spL giving the overall projection:

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

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

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. (**??**), where $\mathrm{sign}(\boldsymbol{\Lambda})$ is replaced by $|\boldsymbol{\Lambda}|^{-\mathrm{spl}}\,\mathrm{sign}(\boldsymbol{\Lambda})$ with element-wise operations (absolute value, sign, exponentiation, multiplication).

For the *factors*, we represent the priors through a convex variational form according to **?**. 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 **?**, the update for $\boldsymbol{\xi}_j$ is

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

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 should be centered to zero mean, zero median, or zero mode (see supplementary). If the correlation of weak signals is of interest too, we recommend to normalize the data.

The iterative model selection procedure requires initialization of the parameters $\boldsymbol{\Lambda}$, $\boldsymbol{\Psi}$, and $\boldsymbol{\xi}_j$. We initialize the variational parameter vectors $\boldsymbol{\xi}_j$ by ones, $\boldsymbol{\Lambda}$ randomly, and $\boldsymbol{\Psi} = \mathrm{diag}(\max(\delta, \mathrm{covar}(\boldsymbol{x}) - \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T))$.

Alternatively $\mathbf{\Lambda}$ can be initialized by the result of a singular value decomposition (SVD).

The user may supply the result of independent component analysis (ICA) as a sparse initialization for $\mathbf{\Lambda}$. ICA can also determine first $\boldsymbol{Z}$ from which $\mathbf{\Lambda}$ can be obtained as the least square solution to $\boldsymbol{X} = \mathbf{\Lambda}\, \boldsymbol{Z}$.

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 $\boldsymbol{x}_j$:

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

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

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

For the FARMS summarization algorithm ($p = 1$ and $\mathbf{\Xi}_j = 1$), this information is the negative logarithm of the I/NI call (**?**).

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:

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

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

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

Again independence across j gives

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

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 **??** and ranking of bicluster in Section **??**, 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`, `fabias`, and `spfabia` 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.

A Class Factorization

`Factorization` is the class structure for results of matrix factorization. Especially it is designed for factor analysis used for biclustering.

The `summary` method shows information about biclusters. The `show` method plots information about biclusters. The `showSelected` method plots selected information about biclusters. The `plot` method produces biplots of biclusters.

Objects can be created by `fabia`, `fabias`, `fabiap`, `fabiasp`, `mfsc`, `nmfsc`, `nmfdiv`, and `nmfeu`.

Objects of class `Factorization` have the following slots:

1. `parameters`: Saves parameters of the factorization method in a list:
   "method",
   "number of cycles",
   "sparseness weight",
   "sparseness prior for loadings",
   "sparseness prior for factors",
   "number biclusters",
   "projection sparseness loadings",
   "projection sparseness factors",
   "initialization range",
   "are loadings rescaled after each iterations",
   "normalization = scaling of rows",
   "centering method of rows",
   "parameter for method".
2. `n`: number of rows, left dimension.
3. `p1`: right dimension of left matrix.
4. `p2`: left dimension of right matrix.
5. `l`: number of columns, right dimension.
6. `center`: vector of the centers.
7. `scaleData`: vector of the scaling factors.
8. `X`: centered and scaled data matrix $n \times l$.
9. `L`: left matrix $n \times p1$.
10. `Z`: right matrix $p2 \times l$.
11. `M`: middle matrix $p1 \times p2$.
12. `LZ`: matrix $\mathbf{\Lambda M Z}$.
13. `U`: noise matrix.
14. `avini`: information of each bicluster, vector of length p2.
15. `xavini`: information extracted from each sample, vector of length l.
16. `ini`: information of each bicluster in each sample, matrix $p2 \times l$.
17. `Psi`: noise variance per row, vector of length n.
18. `lapla`: prior information for each sample, vector of length l.

This class contains the result of different matrix factorization methods. The methods may be generative or not.

Methods my be "singular value decomposition" ($\mathbf{M}$ contains singular values as well as avini, $\mathbf{\Lambda}$ and $\boldsymbol{Z}$ are orthonormal matrices), "independent component analysis" ($\boldsymbol{Z}$ contains the projection/sources, $\mathbf{\Lambda}$ is the mixing matrix, $\mathbf{M}$ is unity), "factor analysis" ($\boldsymbol{Z}$ contains factors, $\mathbf{\Lambda}$ the loadings, $\mathbf{M}$ is unity, $\mathbf{U}$ the noise, $\mathbf{\Psi}$ the noise covariance, lapla is a variational parameter for non-Gaussian factors, avini and ini are the information the factors convey about the observations).

B Biclustering and Matrix Factorization Methods

B.1 fabi

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

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

1. Usage: `fabi(X,p=5,alpha=0.1,cyc=500,spl=0,spz=0.5,center=2,norm=1)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - alpha: sparseness loadings (0 - 1.0); default = 0.1.
   - cyc: number of iterations; default = 500.
   - spl: sparseness prior loadings (0 - 2.0): default = 0 (Laplace).
   - spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
   - norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
3. Return Values:
   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\mathbf{\Lambda}\boldsymbol{Z}$), `L` (loadings $\mathbf{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \mathbf{\Lambda}\boldsymbol{Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$), `Psi` (noise variance $\mathbf{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n\times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p\times l}$.

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 $\boldsymbol{\Psi}$ (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 (**?**) and (**?**).

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 <- makeFabiaDataBlocks(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,3,0.01,20)



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

dat <- makeFabiaDataBlocks(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,13,0.01,200)

extractPlot(resToy,ti="FABI",Y=Y)

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

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)


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

extractPlot(resBreast,ti="FABI Breast cancer(Veer)")

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


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


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {
```

```
data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabi(X,5,0.01,200)

extractPlot(resMulti,ti="FABI Multiple tissues(Su)")

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


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


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(DLBCL_B)

X <- as.matrix(XDLBCL)


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

extractPlot(resDLBCL,ti="FABI Lymphoma(Rosenwald)")

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

B.2 fabia

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

C implementation of `fabia`.

1. Usage: `fabia(X,p=5,alpha=0.1,cyc=500,spl=0,spz=0.5,random=1.0,`
   `center=2,norm=1,scale=0.0,lap=1.0,nL=0,lL=0,bL=0)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - alpha: sparseness loadings (0 - 1.0); default = 0.1.
   - cyc: number of iterations; default = 500.
   - spl: sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
   - spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
   - random <=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
   - norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
   - scale: loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
   - lap: minimal value of the variational parameter, default = 1.
   - nL: maximal number of biclusters at which a row element can participate; default = 0 (no limit).
   - lL: maximal number of row elements per bicluster; default = 0 (no limit).
   - bL: cycle at which the nL or lL maximum starts; default = 0 (start at the beginning).
3. Return values:
   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$), `Psi` (noise variance $\boldsymbol{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 $\boldsymbol{\Psi}$ (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 (**?**) and (**?**).

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.

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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,3,0.01,50)


#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
```

```
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab



#### FABIA

resToy1 <- fabia(X,13,0.01,400)

extractPlot(resToy1,ti="FABIA",Y=Y)

raToy1 <- extractBic(resToy1)

if ((raToy1$bic[[1]][1]>1) && (raToy1$bic[[1]][2])>1) {
    plotBicluster(raToy1,1)
}
if ((raToy1$bic[[2]][1]>1) && (raToy1$bic[[2]][2])>1) {
plotBicluster(raToy1,2)
}
if ((raToy1$bic[[3]][1]>1) && (raToy1$bic[[3]][2])>1) {
plotBicluster(raToy1,3)
}
if ((raToy1$bic[[4]][1]>1) && (raToy1$bic[[4]][2])>1) {
plotBicluster(raToy1,4)
}



colnames(resToy1@X) <- clab

rownames(resToy1@X) <- llab
```

```
plot(resToy1,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy1,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy1,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)



#----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)


resBreast1 <- fabia(X,5,0.1,400)

extractPlot(resBreast1,ti="FABIA Breast cancer(Veer)")


raBreast1 <- extractBic(resBreast1)

if ((raBreast1$bic[[1]][1]>1) && (raBreast1$bic[[1]][2])>1) {
    plotBicluster(raBreast1,1)
}
if ((raBreast1$bic[[2]][1]>1) && (raBreast1$bic[[2]][2])>1) {
    plotBicluster(raBreast1,2)
}
if ((raBreast1$bic[[3]][1]>1) && (raBreast1$bic[[3]][2])>1) {
    plotBicluster(raBreast1,3)
}
if ((raBreast1$bic[[4]][1]>1) && (raBreast1$bic[[4]][2])>1) {
    plotBicluster(raBreast1,4)
}
```

```
plot(resBreast1,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast1,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(Multi_A)

X <- as.matrix(XMulti)

resMulti1 <- fabia(X,5,0.06,300,norm=2)

extractPlot(resMulti1,ti="FABIA Multiple tissues(Su)")

raMulti1 <- extractBic(resMulti1)

if ((raMulti1$bic[[1]][1]>1) && (raMulti1$bic[[1]][2])>1) {
    plotBicluster(raMulti1,1)
}
```

```
if ((raMulti1$bic[[2]][1]>1) && (raMulti1$bic[[2]][2])>1) {
    plotBicluster(raMulti1,2)
}
if ((raMulti1$bic[[3]][1]>1) && (raMulti1$bic[[3]][2])>1) {
    plotBicluster(raMulti1,3)
}
if ((raMulti1$bic[[4]][1]>1) && (raMulti1$bic[[4]][2])>1) {
    plotBicluster(raMulti1,4)
}

plot(resMulti1,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti1,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)

resDLBCL1 <- fabia(X,5,0.1,400,norm=2)
```

```
extractPlot(resDLBCL1,ti="FABIA Lymphoma(Rosenwald)")

raDLBCL1 <- extractBic(resDLBCL1)

if ((raDLBCL1$bic[[1]][1]>1) && (raDLBCL1$bic[[1]][2])>1) {
    plotBicluster(raDLBCL1,1)
}
if ((raDLBCL1$bic[[2]][1]>1) && (raDLBCL1$bic[[2]][2])>1) {
    plotBicluster(raDLBCL1,2)
}
if ((raDLBCL1$bic[[3]][1]>1) && (raDLBCL1$bic[[3]][2])>1) {
    plotBicluster(raDLBCL1,3)
}
if ((raDLBCL1$bic[[4]][1]>1) && (raDLBCL1$bic[[4]][2])>1) {
    plotBicluster(raDLBCL1,4)
}

plot(resDLBCL1,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL1,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}
```

B.3 fabiap

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

1. Usage: `fabiap(X,p=5,alpha=0.1,cyc=500,spl=0,spz=0.5,sL=0.6,sZ=0.6,`
   `random=1.0,center=2,norm=1,scale=0.0,lap=1.0,nL=0,lL=0,bL=0)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.

- alpha: sparseness loadings (0 - 1.0); default = 0.1.
- cyc: number of iterations; default = 500.
- spl: sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
- spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
- sL: final sparseness loadings; default = 0.6.
- sZ: final sparseness factors; default = 0.6.
- random <=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
- center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
- norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
- scale: loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
- lap: minimal value of the variational parameter, default = 1.
- nL: maximal number of biclusters at which a row element can participate; default = 0 (no limit).
- lL: maximal number of row elements per bicluster; default = 0 (no limit).
- bL: cycle at which the nL or lL maximum starts; default = 0 (start at the beginning).

3. Return Values:

   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$), `Psi` (noise variance $\boldsymbol{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 $\boldsymbol{\Psi}$ (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 (**?**) and (**?**).

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 and t he projection in R .

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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,3,0.01,50)



#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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]]
ZC <- dat[[3]]
LC <- dat[[4]]
```

```
gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

#### FABIAP

resToy3 <- fabiap(X,13,0.01,400)

extractPlot(resToy3,ti="FABIAP",Y=Y)

raToy3 <- extractBic(resToy3)

if ((raToy3$bic[[1]][1]>1) && (raToy3$bic[[1]][2])>1) {
    plotBicluster(raToy3,1)
}
if ((raToy3$bic[[2]][1]>1) && (raToy3$bic[[2]][2])>1) {
    plotBicluster(raToy3,2)
}
if ((raToy3$bic[[3]][1]>1) && (raToy3$bic[[3]][2])>1) {
    plotBicluster(raToy3,3)
}
if ((raToy3$bic[[4]][1]>1) && (raToy3$bic[[4]][2])>1) {
    plotBicluster(raToy3,4)
}

colnames(resToy3@X) <- clab

rownames(resToy3@X) <- llab


plot(resToy3,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
```

```
plot(resToy3,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy3,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#-----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#-----------------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)


resBreast3 <- fabiap(X,5,0.1,400)

extractPlot(resBreast3,ti="FABIAP Breast cancer(Veer)")

raBreast3 <- extractBic(resBreast3)

if ((raBreast3$bic[[1]][1]>1) && (raBreast3$bic[[1]][2])>1) {
    plotBicluster(raBreast3,1)
}
if ((raBreast3$bic[[2]][1]>1) && (raBreast3$bic[[2]][2])>1) {
    plotBicluster(raBreast3,2)
}
if ((raBreast3$bic[[3]][1]>1) && (raBreast3$bic[[3]][2])>1) {
    plotBicluster(raBreast3,3)
}
if ((raBreast3$bic[[4]][1]>1) && (raBreast3$bic[[4]][2])>1) {
    plotBicluster(raBreast3,4)
}

plot(resBreast3,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
```

```
plot(resBreast3,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast3,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti3 <- fabiap(X,5,0.01,300)

extractPlot(resMulti3,ti="FABIAP Multiple tissues(Su)")


raMulti3 <- extractBic(resMulti3)

if ((raMulti3$bic[[1]][1]>1) && (raMulti3$bic[[1]][2])>1) {
    plotBicluster(raMulti3,1)
}
if ((raMulti3$bic[[2]][1]>1) && (raMulti3$bic[[2]][2])>1) {
    plotBicluster(raMulti3,2)
}
if ((raMulti3$bic[[3]][1]>1) && (raMulti3$bic[[3]][2])>1) {
```

```
    plotBicluster(raMulti3,3)
}
if ((raMulti3$bic[[4]][1]>1) && (raMulti3$bic[[4]][2])>1) {
    plotBicluster(raMulti3,4)
}

plot(resMulti3,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti3,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL3 <- fabiap(X,5,0.1,400)

extractPlot(resDLBCL3,ti="FABIAP Lymphoma(Rosenwald)")
raDLBCL3 <- extractBic(resDLBCL3)
```

```
if ((raDLBCL3$bic[[1]][1]>1) && (raDLBCL3$bic[[1]][2])>1) {
    plotBicluster(raDLBCL3,1)
}
if ((raDLBCL3$bic[[2]][1]>1) && (raDLBCL3$bic[[2]][2])>1) {
    plotBicluster(raDLBCL3,2)
}
if ((raDLBCL3$bic[[3]][1]>1) && (raDLBCL3$bic[[3]][2])>1) {
    plotBicluster(raDLBCL3,3)
}
if ((raDLBCL3$bic[[4]][1]>1) && (raDLBCL3$bic[[4]][2])>1) {
    plotBicluster(raDLBCL3,4)
}

plot(resDLBCL3,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL3,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)


}
```

B.4 fabias

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

C implementation of `fabias`.

1. Usage: `fabias(X,p=5,alpha=0.6,cyc=500,spz=0.5,random=1.0,`
   `center=2,norm=1,lap=1.0,nL=0,lL=0,bL=0)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - alpha: sparseness loadings (0 - 1.0); default = 0.1.
   - cyc: number of iterations; default = 500.
   - spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).

- random <=0: by SVD, >0: random initialization of loadings in [-random,random]; default = 1.0.
- center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
- norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
- lap: minimal value of the variational parameter, default = 1.
- nL: maximal number of biclusters at which a row element can participate; default = 0 (no limit).
- lL: maximal number of row elements per bicluster; default = 0 (no limit).
- bL: cycle at which the nL or lL maximum starts; default = 0 (start at the beginning).

3. Return Values:

   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$), `Psi` (noise variance $\boldsymbol{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 (**?**) and (**?**).

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 (**?**): 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 C.

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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 <- fabias(X,3,0.6,50)


#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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]]
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
```

```
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

#### FABIAS

resToy2 <- fabias(X,13,0.6,400)

extractPlot(resToy2,ti="FABIAS",Y=Y)


raToy2 <- extractBic(resToy2)

if ((raToy2$bic[[1]][1]>1) && (raToy2$bic[[1]][2])>1) {
    plotBicluster(raToy2,1)
}
if ((raToy2$bic[[2]][1]>1) && (raToy2$bic[[2]][2])>1) {
    plotBicluster(raToy2,2)
}
if ((raToy2$bic[[3]][1]>1) && (raToy2$bic[[3]][2])>1) {
    plotBicluster(raToy2,3)
}
if ((raToy2$bic[[4]][1]>1) && (raToy2$bic[[4]][2])>1) {
    plotBicluster(raToy2,4)
}

colnames(resToy2@X) <- clab

rownames(resToy2@X) <- llab


plot(resToy2,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy2,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy2,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#----------------------------------------

avail <- require(fabiaData)
```

```
if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)

resBreast2 <- fabias(X,5,0.6,300)

extractPlot(resBreast2,ti="FABIAS Breast cancer(Veer)")


raBreast2 <- extractBic(resBreast2)

if ((raBreast2$bic[[1]][1]>1) && (raBreast2$bic[[1]][2])>1) {
    plotBicluster(raBreast2,1)
}
if ((raBreast2$bic[[2]][1]>1) && (raBreast2$bic[[2]][2])>1) {
    plotBicluster(raBreast2,2)
}
if ((raBreast2$bic[[3]][1]>1) && (raBreast2$bic[[3]][2])>1) {
    plotBicluster(raBreast2,3)
}
if ((raBreast2$bic[[4]][1]>1) && (raBreast2$bic[[4]][2])>1) {
    plotBicluster(raBreast2,4)
}

plot(resBreast2,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast2,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}
```

```
#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti2 <- fabias(X,5,0.6,300)

extractPlot(resMulti2,ti="FABIAS Multiple tissues(Su)")

raMulti2 <- extractBic(resMulti2)

if ((raMulti2$bic[[1]][1]>1) && (raMulti2$bic[[1]][2])>1) {
    plotBicluster(raMulti2,1)
}
if ((raMulti2$bic[[2]][1]>1) && (raMulti2$bic[[2]][2])>1) {
    plotBicluster(raMulti2,2)
}
if ((raMulti2$bic[[3]][1]>1) && (raMulti2$bic[[3]][2])>1) {
    plotBicluster(raMulti2,3)
}
if ((raMulti2$bic[[4]][1]>1) && (raMulti2$bic[[4]][2])>1) {
    plotBicluster(raMulti2,4)
}

plot(resMulti2,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
```

```
plot(resMulti2,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti2,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}


#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(DLBCL_B)

X <- as.matrix(XDLBCL)

resDLBCL2 <- fabias(X,5,0.6,300)

extractPlot(resDLBCL2,ti="FABIAS Lymphoma(Rosenwald)")

raDLBCL2 <- extractBic(resDLBCL2)

if ((raDLBCL2$bic[[1]][1]>1) && (raDLBCL2$bic[[1]][2])>1) {
    plotBicluster(raDLBCL2,1)
}
if ((raDLBCL2$bic[[2]][1]>1) && (raDLBCL2$bic[[2]][2])>1) {
    plotBicluster(raDLBCL2,2)
}
if ((raDLBCL2$bic[[3]][1]>1) && (raDLBCL2$bic[[3]][2])>1) {
    plotBicluster(raDLBCL2,3)
}
if ((raDLBCL2$bic[[4]][1]>1) && (raDLBCL2$bic[[4]][2])>1) {
    plotBicluster(raDLBCL2,4)
}

plot(resDLBCL2,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
```

```
plot(resDLBCL2,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL2,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}
```

B.5 fabiasp

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

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

1. Usage: `fabiasp(X,p=5,alpha=0.6,cyc=500,spz=0.5,center=2,norm=1)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - alpha: sparseness loadings (0 - 1.0); default = 0.1.
   - cyc: number of iterations; default = 500.
   - spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
   - norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
3. Return Values:
   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$), `Psi` (noise variance $\boldsymbol{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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:

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

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:

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

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 $\boldsymbol{\Psi}$ (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 (**?**) and (**?**).

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 (**?**): 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 R , therefore it is *slow*.

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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,3,0.6,50)
```

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

dat <- makeFabiaDataBlocks(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,13,0.6,200)

extractPlot(resToy,ti="FABIASP",Y=Y)

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


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

resBreast <- fabiasp(X,5,0.6,200)

extractPlot(resBreast,ti="FABIASP Breast cancer(Veer)")

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


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

```
# DEMO3
#---------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti <- fabiasp(X,5,0.6,200)

extractPlot(resMulti,"ti=FABIASP Multiple tissues(Su)")

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

}


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


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(DLBCL_B)

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

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

extractPlot(resDLBCL,ti="FABIASP Lymphoma(Rosenwald)")

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

}
```

B.6 spfabia

Factor Analysis for Bicluster Acquisition: SPARSE FABIA (**?**).

C implementation of `spfabia`.

1. Usage:
```
spfabia(X,p=5,alpha=0.1,cyc=500,spl=0,spz=0.5,non_negative=0,random=1.0,
        write_file=1,norm=1,scale=0.0,lap=1.0,nL=0,lL=0,bL=0,samples=0,initL=0,
        iter=1,quant=0.001,dorescale=FALSE,doini=FALSE,eps=1e-3,eps1=1e-10)
```

2. Arguments:

   - X: the file name of the sparse matrix in sparse format.
   - p: number of hidden factors = number of biclusters; default = 5.
   - alpha: sparseness loadings (0 - 1.0); default = 0.1.
   - cyc: number of iterations; default = 500.
   - spl: sparseness prior loadings (0 - 2.0); default = 0 (Laplace).
   - spz: sparseness factors (0.5 - 2.0); default = 0.5 (Laplace).
   - non_negative: Non-negative factors and loadings if non_negative > 0; default = 0.
   - random: >0: random initialization of loadings in [0,random], <0: random initialization of loadings in [-random,random]; default = 1.0.
   - write_file: >0: results are written to files (L in sparse format), default = 1.
   - norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
   - scale: loading vectors are scaled in each iteration to the given variance. 0.0 indicates non scaling; default = 0.0.
   - lap: minimal value of the variational parameter, default = 1.
   - nL: maximal number of biclusters at which a row element can participate; default = 0 (no limit).
   - lL: maximal number of row elements per bicluster; default = 0 (no limit).
   - bL: cycle at which the nL or lL maximum starts; default = 0 (start at the beginning).

- samples: vector of samples which should be included into the analysis; default = 0 (all samples).
- initL: vector of indices of the selected samples which are used to initialize L; default = 0 (random initialization).
- iter: number of iterations; default = 1.
- quant: qunatile of largest L values to remove in each iteration; default = 0.001.
- lowerB: lower bound for filtering the inputs columns, the minimal column sum; default = 0.0.
- upperB: upper bound for filtering the inputs columns, the maximal column sum; default = 1000.0.
- dorescale: rescale factors Z to variance 1 and consequently also L; logical; default: FALSE.
- doini: compute the information content of the biclusters and sort the biclusters according to their information content; logical, default: FALSE.
- eps: lower bound for variational parameter lapla; default: 1e-3.
- eps1: lower bound for divisions to avoid division by zero; default: 1e-10.

3. Return values:

   - object of the class `Factorization`. Containing `L` (loadings $\mathbf{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `Psi` (noise variance $\mathbf{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

Version of fabia for a sparse data matrix. The data matrix is directly scanned by the C-code and must be in sparse matrix format.

Sparse matrix format:

- first line: numer of rows (the samples).
- second line: number of columns (the features).
- following lines: for each sample (row) three lines with

  1. number of nonzero row elements
  2. indices of the nonzero row elements (ATTENTION: starts with 0!!)
  3. values of the nonzero row elements (ATTENTION: floats with decimal point like 1.0 !!)

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 $\boldsymbol{\Psi}$ (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 (**?**) and (**?**).

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.

EXAMPLE:

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

samples <- 20
features <- 200
sparseness <- 0.9
write(samples, file = "sparseFarmsTest.txt",ncolumns = features,append = FALSE, sep = " ")
write(features, file = "sparseFarmsTest.txt",ncolumns = features,append = TRUE, sep = " ")
for (i in 1:samples)
{
  ind <- which(runif(features)>sparseness)-1
  num <- length(ind)
  val <- abs(rnorm(num))
  write(num, file = "sparseFarmsTest.txt",ncolumns = features,append = TRUE, sep = " ")
  write(ind, file = "sparseFarmsTest.txt",ncolumns = features,append = TRUE, sep = " ")
  write(val, file = "sparseFarmsTest.txt",ncolumns = features,append = TRUE, sep = " ")
}

res <- spfabia("sparseFarmsTest",p=3,alpha=0.03,cyc=50,non_negative=1,write_file=0,norm=0)

unlink("sparseFarmsTest.txt")

plot(res,dim=c(1,2))
```

```
plot(res,dim=c(1,3))
plot(res,dim=c(2,3))
```

B.7 readSamplesSpfabia

Factor Analysis for Bicluster Acquisition: Read Samples of SpFabia (**?**).

C implementation of `readSamplesSpfabia`.

1. Usage: `readSamplesSpfabia(X,samples=0,lowerB=0.0,upperB=1000.0)`
2. Arguments:
   - X: the file name of the sparse matrix in sparse format.
   - samples: vector of samples which should be read; default = 0 (all samples)
   - lowerB: lower bound for filtering the inputs columns, the minimal column sum; default = 0.0.
   - upperB: upper bound for filtering the inputs columns, the maximal column sum; default = 1000.0.
3. Return values:
   - `X` (data of the given samples)

Read the samples to analyze results of spfabia.

The code is implemented in C.

EXAMPLE:

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

# no test
```

B.8 readSpfabiaResult

Factor Analysis for Bicluster Acquisition: Read Results of SpFabia (**?**).

C implementation of `readSpfabiaResult`.

1. Usage: `readSpfabiaResult(X)`
2. Arguments:
   - X: the file prefix name of the result files of `spfabia`.
3. Return values:
   - object of the class `Factorization`. Containing `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `Psi` (noise variance $\boldsymbol{\Psi}$), `lapla` (variational parameter), `avini` (the information which the factor z_{ij} contains about $\boldsymbol{x}_j$ averaged over j) `xavini` (the information which the factor $\tilde{\boldsymbol{z}}_j$ contains about $\boldsymbol{x}_j$ averaged over j) `ini` (for each j the information which the factor z_{ij} contains about $\boldsymbol{x}_j$).

Read the results of spfabia.

The code is implemented in C.

EXAMPLE:

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

# no test
```

B.9 mfsc

Sparse Matrix Factorization for bicluster analysis (MFSC) (**?**).

1. Usage: `mfsc(X,p=5,cyc=100,sL=0.6,sZ=0.6,center=2,norm=1)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - cyc: maximal number of iterations; default = 100.
   - sL: final sparseness loadings; default = 0.6.
   - sZ: final sparseness factors; default = 0.6.
   - center: data centering: 1 (mean), 2 (median), > 2 (mode), 0 (no); default = 2.
   - norm: data normalization: 1 (0.75-0.25 quantile), >1 (var=1), 0 (no); default = 1.
3. Return Values:

- object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `center` (centering vector), `scaleData` (scaling vector), `X` (centered and scaled data $\boldsymbol{X}$).

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

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

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

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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

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

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 (**?**). 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 <- makeFabiaDataBlocks(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(X,3,30,0.6,0.6)

\dontrun{

#-----------------
# DEMO1: Toy Data
#-----------------

n = 1000
l= 100
p = 10

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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]]
```

```
ZC <- dat[[3]]
LC <- dat[[4]]

gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
     llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


groups <- gclab

#### MFSC

resToy4 <- mfsc(X,13,100,0.6,0.6)

extractPlot(resToy4,ti="MFSC",Y=Y)

raToy4 <- extractBic(resToy4,thresZ=0.01,thresL=0.05)

if ((raToy4$bic[[1]][1]>1) && (raToy4$bic[[1]][2])>1) {
    plotBicluster(raToy4,1)
}
if ((raToy4$bic[[2]][1]>1) && (raToy4$bic[[2]][2])>1) {
    plotBicluster(raToy4,2)
}
if ((raToy4$bic[[3]][1]>1) && (raToy4$bic[[3]][2])>1) {
    plotBicluster(raToy4,3)
}
if ((raToy4$bic[[4]][1]>1) && (raToy4$bic[[4]][2])>1) {
    plotBicluster(raToy4,4)
}

colnames(resToy4@X) <- clab

rownames(resToy4@X) <- llab
```

```
plot(resToy4,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy4,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resToy4,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)


#----------------------------------------
# DEMO2: Laura van't Veer's gene expression
#        data set for breast cancer
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)

resBreast4 <- mfsc(X,5,100,0.6,0.6)

extractPlot(resBreast4,ti="MFSC Breast cancer(Veer)")


raBreast4 <- extractBic(resBreast4,thresZ=0.01,thresL=0.05)

if ((raBreast4$bic[[1]][1]>1) && (raBreast4$bic[[1]][2])>1) {
    plotBicluster(raBreast4,1)
}
if ((raBreast4$bic[[2]][1]>1) && (raBreast4$bic[[2]][2])>1) {
    plotBicluster(raBreast4,2)
}
if ((raBreast4$bic[[3]][1]>1) && (raBreast4$bic[[3]][2])>1) {
    plotBicluster(raBreast4,3)
}
if ((raBreast4$bic[[4]][1]>1) && (raBreast4$bic[[4]][2])>1) {
    plotBicluster(raBreast4,4)
}
```

```
plot(resBreast4,dim=c(1,2),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(1,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(1,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(1,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,3),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(2,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(3,4),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(3,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)
plot(resBreast4,dim=c(4,5),label.tol=0.03,col.group=CBreast,lab.size=0.6)

}


#-----------------------------------
# DEMO3: Su's multiple tissue types
#        gene expression data set
#-----------------------------------


avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {

data(Multi_A)

X <- as.matrix(XMulti)

resMulti4 <- mfsc(X,5,100,0.6,0.6)

extractPlot(resMulti4,ti="MFSC Multiple tissues(Su)")


raMulti4 <- extractBic(resMulti4,thresZ=0.01,thresL=0.05)

if ((raMulti4$bic[[1]][1]>1) && (raMulti4$bic[[1]][2])>1) {
    plotBicluster(raMulti4,1)
}
if ((raMulti4$bic[[2]][1]>1) && (raMulti4$bic[[2]][2])>1) {
    plotBicluster(raMulti4,2)
```

```
}
if ((raMulti4$bic[[3]][1]>1) && (raMulti4$bic[[3]][2])>1) {
    plotBicluster(raMulti4,3)
}
if ((raMulti4$bic[[4]][1]>1) && (raMulti4$bic[[4]][2])>1) {
    plotBicluster(raMulti4,4)
}

plot(resMulti4,dim=c(1,2),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(1,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(1,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(1,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,3),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(2,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(3,4),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(3,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)
plot(resMulti4,dim=c(4,5),label.tol=0.01,col.group=CMulti,lab.size=0.6)

}



#----------------------------------------
# DEMO4: Rosenwald's diffuse large-B-cell
#        lymphoma gene expression data set
#----------------------------------------

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(DLBCL_B)

X <- as.matrix(XDLBCL)


resDLBCL4 <- mfsc(X,5,100,0.6,0.6)
```

```
extractPlot(resDLBCL4,ti="MFSC Lymphoma(Rosenwald)")

raDLBCL4 <- extractBic(resDLBCL4,thresZ=0.01,thresL=0.05)

if ((raDLBCL4$bic[[1]][1]>1) && (raDLBCL4$bic[[1]][2])>1) {
    plotBicluster(raDLBCL4,1)
}
if ((raDLBCL4$bic[[2]][1]>1) && (raDLBCL4$bic[[2]][2])>1) {
    plotBicluster(raDLBCL4,2)
}
if ((raDLBCL4$bic[[3]][1]>1) && (raDLBCL4$bic[[3]][2])>1) {
    plotBicluster(raDLBCL4,3)
}
if ((raDLBCL4$bic[[4]][1]>1) && (raDLBCL4$bic[[4]][2])>1) {
    plotBicluster(raDLBCL4,4)
}

plot(resDLBCL4,dim=c(1,2),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(1,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,3),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(2,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(3,4),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(3,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)
plot(resDLBCL4,dim=c(4,5),label.tol=0.03,col.group=CDLBCL,lab.size=0.6)

}
```

B.10 nmfdiv

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

1. Usage: `nmfdiv(X,p=5,cyc=100)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - cyc: maximal number of iterations; default = 100.
3. Return Values:

- object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `X` (scaled data $\boldsymbol{X}$).

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

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

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

The model selection is performed according to (**??**).

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:

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

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

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 <- makeFabiaDataBlocks(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)


resEx <- nmfdiv(X,3)



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

dat <- makeFabiaDataBlocks(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)


resToy <- nmfdiv(X,13)

extractPlot(resToy,ti="NMFDIV",Y=Y)
```

B.11 nmfeu

Non-negative Matrix Factorization with Euclidean distance as objective.

1. Usage: `nmfeu(X,p=5,cyc=100)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - cyc: maximal number of iterations; default = 100.
3. Return Values:
   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `X` (scaled data $\boldsymbol{X}$).

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

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

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

The model selection is performed according to (**??**).

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:

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

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

The code is implemented in R .

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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)


resEx <- nmfeu(X,3)


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

dat <- makeFabiaDataBlocks(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)


resToy <- nmfeu(X,13)

extractPlot(resToy,ti="NMFEU",Y=Y)
```

B.12 nmfsc

Non-negative Sparse Matrix Factorization with sparseness constraints.

1. Usage: `nmfsc(X,p=5,cyc=100,sL=0.6,sZ=0.6)`
2. Arguments:
   - X: the data matrix.
   - p: number of hidden factors = number of biclusters; default = 5.
   - cyc: maximal number of iterations; default = 100.
   - sL: sparseness loadings; default = 0.6.
   - sZ: sparseness factors; default = 0.6.
3. Return Values:
   - object of the class `Factorization`. Containing `LZ` (estimated noise free data $\boldsymbol{\Lambda Z}$), `L` (loadings $\boldsymbol{\Lambda}$), `Z` (factors $\boldsymbol{Z}$), `U` (noise $\boldsymbol{X} - \boldsymbol{\Lambda Z}$), `X` (data $\boldsymbol{X}$).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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

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

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 (**?**). 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 <- makeFabiaDataBlocks(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)

resEx <- nmfsc(X,3,30,0.6,0.6)


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

dat <- makeFabiaDataBlocks(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)


resToy <- nmfsc(X,13,100,0.6,0.6)

extractPlot(resToy,ti="NMFSC",Y=Y)
```

C Analyzing the Results of Biclustering / Matrix Factorization

C.1 plot

Plot of a Matrix Factorization.

Produces a (biplot) of a Matrix Factorization result.

1. Usage: `plot(x,y, ...)`
2. Arguments:
   - x: object of the class `Factorization`.
   - y: missing, not used.
   - ...: with following parameters
     - Rm: row weighting vector.
     - Cm: column weighting vector.
     - dim: optional principal factors that are plotted along the horizontal and vertical axis. Defaults to `c(1,2)`.
     - zoom: optional zoom factor for row and column items. Defaults to `c(1,1)`.
     - col.group: optional vector (character or numeric) indicating the different groupings of the columns. Defaults to 1.
     - colors: vector specifying the colors for the annotation of the plot; the first two elements concern the rows; the third till the last element concern the columns; the first element will be used to color the unlabeled rows; the second element for the labeled rows and the remaining elements to give different colors to different groups of columns.
     - col.areas: logical value indicating whether columns should be plotted as squares with areas proportional to their marginal mean and colors representing the different groups (`TRUE`), or with symbols representing the groupings and identical size (`FALSE`). Defaults to `TRUE`.
     - col.symbols: vector of symbols when `col.areas=FALSE` corresponds to the `pch` argument of the function `plot`.
     - sampleNames: Either a logical vector of length one or a character vector of length equal to the number of samples in the dataset. If a logical is provided, sample names will be displayed on the plot (`TRUE`; default) or not (`FALSE`); if a character vector is provided, the names provided will be used to label the samples instead of the default column names.
     - rot: rotation of plot. Defaults to `c(-1,-1)`.
     - labels: character vector to be used for labeling points on the graph; if `NULL`, the row names of `x` are used instead.
     - label.tol: numerical value specifying either the percentile (`label.tol<=1`)of rows or the number of rows (`label.tol>1`) most distant from the plot-center `(0,0)` that are labeled and are plotted as circles with area proportional to the marginal means of the original data.

- lab.size: size of identifying labels for row- and column-items as `cex` parameter of the `text` function.
- col.size: size in mm of the column symbols.
- row.size: size in mm of the row symbols.
- do.smoothScatter: use smoothScatter or not instead of plotting individual points.
- do.plot: produce a plot or not.
- ...: further arguments to `eqscaleplotLoc` which draws the canvasfor the plot; useful for adding a `main` or a custom `sub`.

and with the default values

- Rm=rep(1,nrow(X)),
- Cm=rep(1,ncol(X)),
- dim = c(1, 2),
- zoom = rep(1, 2),
- col.group = rep(1, ncol(X)),
- colors = c("orange1", "red", rainbow(length(unique(col.group)), start=2/6, end=4/6)),
- col.areas = TRUE,
- col.symbols = c(1, rep(2, length(unique(col.group)))),
- sampleNames = TRUE,
- rot = rep(-1, length(dim)),
- labels = NULL,
- label.tol = 1,
- lab.size = 0.725,
- col.size = 10,
- row.size = 10,
- do.smoothScatter = FALSE,
- do.plot = TRUE,

3. Return Values:

   - Rows: a list with the X and Y coordinates of the rows and an indication `Select` of whether the row was selected according to `label.tol`.
   - Columns: a list with the X and Y coordinates of the columns.

The function `plot.fabia` is based on the function `plot.mpm` in the R package `mpm` (Version: 1.0-16, Date: 2009-08-26, Title: Multivariate Projection Methods, Maintainer: Tobias Verbeke <tobias.verbeke@openanalytics.be>, Author: Luc Wouters <wouters_luc@telenet.be>).

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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

For noise free projection like independent component analysis we set the noise term to zero: $\boldsymbol{\Upsilon} = \mathbf{0}$.

The argument `label.tol` can be used to select the most informative rows, i.e. rows that are most distant from the center of the plot (smaller 1: percentage of rows, larger 1: number of rows).

Only these row-items are then labeled and represented as circles with their areas proportional to the row weighting.

If the column-items are grouped these groups can be visualized by colors given by `col.group`.

Implementation in R .

EXAMPLE:

```
n=200
l=100
p=4

dat <- makeFabiaDataBlocks(n = n,l= l,p = p,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]]
ZC <- dat[[3]]
LC <- dat[[4]]


resEx <- fabia(X,p,0.01,400)


gclab <- rep.int(0,l)
gllab <- rep.int(0,n)
clab <- as.character(1:l)
llab <- as.character(1:n)
for (i in 1:p){
 for (j in ZC[i]){
     clab[j] <- paste(as.character(i),"_",clab[j],sep="")
 }
 for (j in LC[i]){
```

```
    llab[j] <- paste(as.character(i),"_",llab[j],sep="")
 }
 gclab[unlist(ZC[i])] <- gclab[unlist(ZC[i])] + p^i
 gllab[unlist(LC[i])] <- gllab[unlist(LC[i])] + p^i
}


labels <- paste(as.character(clab) ,"_",as.character(1:l),sep="")

groups <- gclab

colnames(resEx@X) <- clab

rlabels <- paste(as.character(llab) ,"_",as.character(1:l),sep="")

rownames(resEx@X) <- llab


plot(resEx,dim=c(1,2),label.tol=0.1,col.group = groups,lab.size=0.6)

plot(resEx,dim=c(1,3),label.tol=0.1,col.group = groups,lab.size=0.6)
plot(resEx,dim=c(2,3),label.tol=0.1,col.group = groups,lab.size=0.6)
```

C.2 show

Plots Statistics of a Matrix Factorization.

This function plots statistics on a matrix factorization which is stored as an instance of `Factorization-class`.

1. Usage: `show(object)`
2. Arguments:
   - object: An instance of `Factorization-class`.
3. Return Values:
   - no value.

Following is plotted:

1) the information content of biclusters.

2) the information content of samples.

3) the loadings per bicluster.

4) the factors per bicluster.

Implementation in R .

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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]]


resEx <- fabia(X,3,0.01,100)

show(resEx)
```

C.3 showSelected

Plots selected Statistics of a Matrix Factorization.

This function plots selected statistics on a matrix factorization which is stored as an instance of `Factorization-class`.

1. Usage: `showSelected(object, which=c(1,2,3,4))`
2. Arguments:
   - object: An instance of `Factorization-class`.
   - which: A list of which plots should be generated: 1=the information content of biclusters, 2=the information content of samples, 3=the loadings per bicluster, 4=the factors per bicluster, default c(1,2,3,4).
3. Return Values:
   - no value.

Following is plotted depending on the plot selection variable "which":

1) the information content of biclusters.

2) the information content of samples.

3) the loadings per bicluster.

4) the factors per bicluster.

Implementation in R .

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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]]


resEx <- fabia(X,3,0.01,100)

showSelected(resEx,which=1)
showSelected(resEx,which=2)
```

C.4 summary

Summary of Matrix Factorization.

This function gives information on a matrix factorization which is stored as an instance of `Factorization-class`.

1. Usage: `summary(object, ...)`
2. Arguments:
   - object: An instance of `Factorization-class`.
   - ...: further arguments.
3. Return Values:
   - no value.

First, it gives the number or rows and columns of the original matrix.

Then the number of clusters for rows and columns is given.

Then for the row cluster the information content is given.

Then for each column its information is given.

Then for each column cluster a summary is given.

Then for each row cluster a summary is given.

Implementation in R .

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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]]


resEx <- fabia(X,3,0.01,100)

summary(resEx)
```

C.5 extractBic

Extraction of Biclusters.

1. Usage: `extractBic(fact,thresZ=0.5,thresL=NULL)`
2. Arguments:
   - fact: object of class `Factorization`.
   - thresZ: threshold for sample belonging to bicluster; default 0.5.
   - thresL: threshold for loading belonging to bicluster (if not given it is estimated).
3. Return Values:
   - bic: extracted biclusters.
   - numn: indexes for the extracted biclusters.

- bicopp: extracted opposite biclusters.
- numnopp: indexes for the extracted opposite biclusters.
- X: scaled and centered data matrix.
- np: number of biclusters.

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 $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

$\boldsymbol{\Upsilon}$ 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 <- makeFabiaDataBlocks(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,3,0.01,20)

rEx <- extractBic(resEx)

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


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

dat <- makeFabiaDataBlocks(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,13,0.01,200)

rToy <- extractBic(resToy)

resToy@avini

rToy$bic[1,]
rToy$bic[2,]
```

```
rToy$bic[3,]

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

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#########################################################")
} else {


data(Breast_A)

X <- as.matrix(XBreast)

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

rBreast <- extractBic(resBreast)

resBreast@avini

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

C.6 extractPlot

Plotting of Biclustering Results.

1. Usage: `extractPlot(fact,thresZ=0.5,ti="",thresL=NULL,Y=NULL,which=c(1,2,3,4,5,6,7,8))`
2. Arguments:
   - fact: object of class `Factorization`.
   - thresZ: threshold for sample belonging to bicluster; default 0.5.
   - thresL: threshold for loading belonging to bicluster (estimated if not given).
   - ti: plot title; default "".

- Y: noise free data matrix.
- which: which plot is shown: 1=noise free data (if available), 2=data, 3=reconstructed data, 4=error, 5=absolute factors, 6=absolute loadings, 7=reconstructed matrix sorted,8=original matrix sorted; default c(1,2,3,4,5,6,7,8).

3. The method produces following plots depending what plots are chosen by the "which" variable:
   - $\boldsymbol{Y}$: noise free data (if available),
   - $\boldsymbol{X}$: data,
   - $\boldsymbol{\Lambda Z}$: reconstructed data,
   - $\boldsymbol{\Lambda Z} - \boldsymbol{X}$: error,
   - $\mathrm{abs}(\boldsymbol{Z})$: absolute factors,
   - $\mathrm{abs}()$: absolute loadings,
   - $\mathbf{pmL}\boldsymbol{Z}\mathbf{pmZ}$: reconstructed matrix sorted,
   - $\mathbf{pmL}\boldsymbol{X}\mathbf{pmZ}$: 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

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

Here $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 the plots given above at “Plots”.

For sorting first `kmeans` is performed on the p dimensional space and then the vectors which belong to the same cluster are put together.

This sorting is in general not able to visualize all biclusters as blocks if they overlap.

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

Implementation in R .

EXAMPLE:

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

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

dat <- makeFabiaDataBlocks(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,3,0.01,20)

extractPlot(resEx,ti="FABIA",Y=Y)



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

dat <- makeFabiaDataBlocks(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,13,0.01,200)

extractPlot(resToy,ti="FABIA",Y=Y)

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

avail <- require(fabiaData)

if (!avail) {
    message("")
    message("")
    message("#######################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("#######################################################")
} else {
```

```
data(Breast_A)

X <- as.matrix(XBreast)


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

extractPlot(resBreast,ti="FABIA Breast cancer(Veer)")

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

C.7 plotBicluster

Plots a bicluster.

1. Usage: `plotBicluster(r,p,opp=FALSE,zlim=NULL,title=NULL,which=c(1, 2))`
2. Arguments:
   - r: the result of extractBic.
   - p: the bicluster to plot.
   - opp: plot opposite bicluster, default = FALSE.
   - zlim: vector containing a low and high value to use for the color scale.
   - title: title of the plot.
   - which: which plots are shown: 1=data matrix with bicluster on upper left, 2=plot of the bicluster; default c(1, 2).

One bicluster is visualized by two plots. The variable "which" indicates which plots should be shown.

Plot1 (which=1): The data matrix is sorted such that the bicluster appear at the upper left corner.

The bicluster is marked by a rectangle.

Plot2 (which=2): Only the bicluster is presented.

Implementation in R .

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

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

dat <- makeFabiaDataBlocks(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,3,0.01,20)

rEx <- extractBic(resEx)

plotBicluster(rEx,p=1)


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

dat <- makeFabiaDataBlocks(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,13,0.01,200)



rToy <- extractBic(resToy)

plotBicluster(rToy,p=1)

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

avail <- require(fabiaData)

if (!avail) {
    message("")
```

```
    message("")
    message("########################################################")
    message("Package 'fabiaData' is not available: please install.")
    message("########################################################")
} else {

data(Breast_A)

X <- as.matrix(XBreast)

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

rBreast <- extractBic(resBreast)

plotBicluster(rBreast,p=1)

}
```

D Data Generation Methods

D.1 makeFabiaData

Generation of bicluster data.

1. Usage: `makeFabiaData(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 $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 <- makeFabiaData(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]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)
```

D.2 makeFabiaDataPos

Generation of bicluster data.

1. Usage: `makeFabiaDataPos(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 $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 <- makeFabiaDataPos(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]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)
```

D.3 makeFabiaDataBlocks

Generation of bicluster data with bicluster blocks.

1. Usage: `makeFabiaDataBlocks(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 $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n\times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n\times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p\times l}$.

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 <- makeFabiaDataBlocks(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]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)
```

D.4 makeFabiaDataBlocksPos

Generation of bicluster data with bicluster blocks.

1. Usage: `makeFabiaDataBlocksPos(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 $\boldsymbol{X} \in \mathbb{R}^{n \times l}$ and $\boldsymbol{\Upsilon} \in \mathbb{R}^{n \times l}$, $\boldsymbol{\lambda}_i \in \mathbb{R}^n$, $\boldsymbol{z}_i \in \mathbb{R}^l$, $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, $\boldsymbol{Z} \in \mathbb{R}^{p \times l}$.

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 <- makeFabiaDataBlocksPos(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]]

matrixImagePlot(Y)
dev.new()
matrixImagePlot(X)
```

E Utilities

E.1 fabiaVersion

Display version info for package and for FABIA.

1. Usage: `fabiaVersion()`

EXAMPLE:

```
fabiaVersion()
```

E.2 matrixImagePlot

Plotting of a matrix.

1. Usage: `matrixImagePlot(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 at http://www.phaget4.org/R/myImagePlot.R

EXAMPLE:

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

dat <- makeFabiaDataBlocks(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]]

matrixImagePlot(X)
```

E.3 projFuncPos

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

1. Usage: `projFuncPos(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 <- projFuncPos(s,k1=sp,k2=1)

s
ss
```

E.4 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 (**?**): 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
```

E.5 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: $\boldsymbol{X} - \boldsymbol{u}\mathbf{1}^T$ 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 <- makeFabiaDataBlocksPos(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)
```

E.6 eqscplotLoc

Local Version of `eqscplot` from the package MASS.

Plots with Geometrically Equal Scales: Version of a scatterplot with scales chosen to be equal on both axes, that is 1cm represents the same units on each.

1. Usage: `eqscplotLoc(x, y, ratio = 1, tol = 0.04, uin, ...)`
2. Arguments:
   - x: vector of x values, or a 2-column matrix, or a list with components “x” and “y”.
   - y: vector of y values.
   - ratio: desired ratio of units on the axes. Units on the y axis are drawn at “ratio” times the size of units on the x axis. Ignored if “uin” is specified and of length 2.
   - tol: proportion of white space at the margins of plot.
   - uin: desired values for the units-per-inch parameter. If of length 1, the desired units per inch on the x axis.
   - ...: further arguments for “plot” and graphical parameters. Note that “par(xaxs="i", yaxs="i")” is enforced, and “xlim” and “ylim” will be adjusted accordingly.

3. Return Values:

    - invisibly, the values of "uin" used for the plot.

Plots with Geometrically Equal Scales eqscplot from the package MASS (Version: 7.3-3, 2009-10-15 Maintainer: Brian Ripley <ripley@stats.ox.ac.uk>).

To make the package stand alone, this function is included as source.

Limits for the x and y axes are chosen so that they include the data. One of the sets of limits is then stretched from the midpoint to make the units in the ratio given by "ratio". Finally both are stretched by "1 + tol" to move points away from the axes, and the points plotted.

Implementation in R .

REFERENCE:

- Venables, W. N. and Ripley, B. D., (2002), "Modern Applied Statistics with S", Fourth edition, Springer.

F Demo Data Sets from `fabiaData`

In above examples gene expression data sets from the R package `fabiaData` have been used. In this section these data sets are described.

F.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 (**?**) (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.

(**?**) 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 (**?**).

F.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 (**?**) (http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi)

Goal was to predict the survival after chemotherapy

(**?**) 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 (**?**).

F.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 (**?**) (http://www.brxsoadinstitute.org/cgi-bin/cancer/datasets.cgi) s

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.