Factor Analysis for Bicluster Acquisition (FABIA)

Sepp Hochreiter

August 31, 2009

Contents

1 Introduction

The *fabia* package is part of the BioConductor[1] project. The package allows to extract biclusters from data sets based on a generative model.

Especially, it has been designed for microarray data sets, but can be used for other data sets as well.

2 Getting Started: FABIA

First load the *fabia* package.

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

fabia is stand alone and does not require other packages.

---

[1] `http://www.bioconductor.org/`

2.1 Quick start

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

1. Create a working directory, e.g. `c:/fabia/data` in windows or `/home/myself/fabia/data` in Unix. Move the data file `datafile.cvs` to that directory, e.g. under Unix `cp datafile.cvs /home/myself/fabia/data/` or drag the file `datafile.cvs` into that directory under windows.

2. Start `R` and change to the working directory. Under windows

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

   and under Unix

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

   You can also start `R` in that directory under Unix.

3. Load the library.

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

4. Read the data file "datafile.cvs"

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

5. Normalize the data

   ```
   R> X <- X- rowMeans(X)
   R> XX <- (1/ncol(X))*tcrossprod(X)
   R> dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
   R> X <- dXX*X
   ```

6. Select the model based on the data: 4 biclusters; sparseness 0.8; 200 cycles

   ```
   R> res <- fabias(X,200,0.6,5)
   ```

7. Extract biclusters and plot results

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

8. Show bicluster 1

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

9. Show bicluster 2

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

10. Show bicluster 3

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

11. Show bicluster 4

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

The biclusters generated by `kmeans` for visualization pueposes are in list element "biclust". For example, the first bicluster can be obtained by

```
R> rr$biclust[1,] #$
```

The functions `fabia` and `fabias` were written in C for efficiency and numerical precision.

2.2 Test on Toy Data Set

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

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

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

2. store the generated data in variables

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

3. normalize the data

```
R> X <- X - rowMeans(X)
R> X <- X - rowMeans(X)
R> XX <- tcrossprod(X)
R> dXX <- 1/sqrt(diag(XX))
R> X <- dXX*X
```

4. perform `fabia` (sparseness by Laplace prior) to extract biclusters; 200 cycles, sparseness 0.1 (Laplace), 13 biclusters.

```
R> resToy1 <- fabia(X,200,0.3,1.0,1.0,13)
```

5. extract and plot results

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

6. perform `fabias` (sparseness by projection) to extract biclusters; 200 cycles, sparseness 0.8 (projection), 13 biclusters.

   ```
   R> resToy2 <- fabias(X,200,0.8,1.0,13)
   ```

7. extract and plot results

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

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

   ```
   R> resToy3 <- fabiap(X,200,0.3,1.0,1.0,13,0.7,0.7)
   ```

9. extract and plot results

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

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

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

11. extract and plot results

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

2.3 Demos

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

1. demo1: toy data.

   ```
   R> fabiaDemo()
   ```

   Choose “1” and you get above toy data demonstration.

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

   ```
   R> fabiaDemo()
   ```

   Choose “2” to extract subclasses in the data set of van’t Veer as biclusters.

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

   ```
   R> fabiaDemo()
   ```

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

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

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

   ```
   R> fabiaDemo()
   ```

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

3 Some Details

3.1 C implementation of FABIA

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

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

3.2 Extraction of Biclusters

There is no unique method to extract the biclusters. We used thresholds to determine which observations and which samples participate at a bicluster. However, these thresholds are quite arbitrary.

The sorting and extraction by kmeans is done for visualization but it may fail if observations and samples are in more than one bicluster. Clustering methods that assign a data point to only one cluster are not appropriate in such cases.

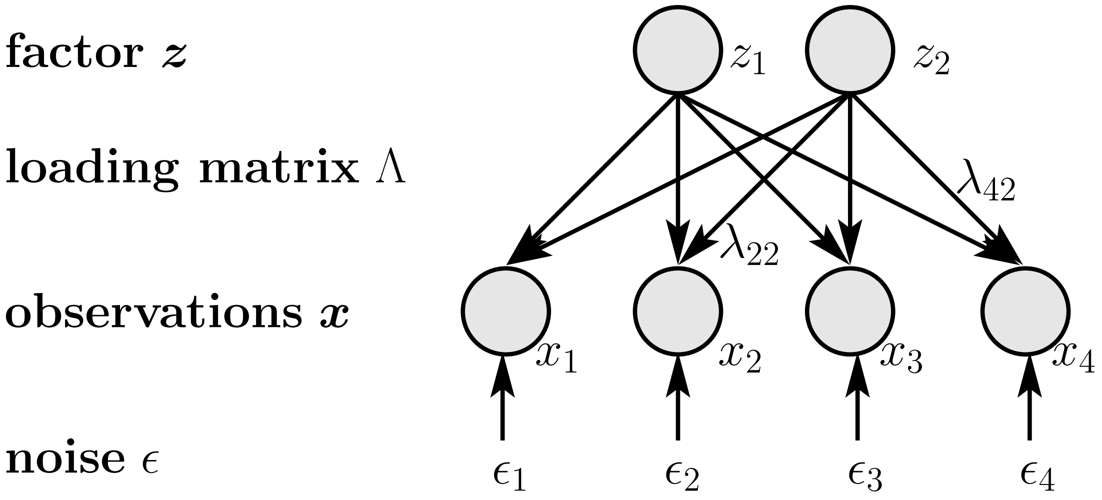

Figure 1: Factor analysis model with two factors and four observations.

4 The Underlying Methods

4.1 The Generative Model

We use a factor analysis model with p factors: In contrast to "principal component analysis" (PCA), factor analysis has the advantage that it is a *generative model* that accounts for the independent noise at the observations. Fig. 1 depicts the factor analysis model with two factors and four observations.

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

- $\boldsymbol{x}$: observations.
- $\boldsymbol{\Lambda}$: loading matrix.
- $\boldsymbol{\Lambda}_i$: loading vector for factor i.
- $\boldsymbol{z}$: vector of factors.
- z_i: value of factor i.
- $\boldsymbol{\epsilon}$: additive noise on observations.

In the standard factor analysis model we have

$$\boldsymbol{z} \; \sim \; \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right)$$

and

$$\boldsymbol{\epsilon} \ \sim \ \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Psi}\right) \ .$$

The observations $\boldsymbol{x}$ and $\boldsymbol{\epsilon}$ are from $\mathbb{R}^n$, the *factors* $\boldsymbol{z} \in \mathbb{R}^p$, the *factor loading matrix* $\boldsymbol{\Lambda} \in \mathbb{R}^{n \times p}$, and the noise covariance matrix $\boldsymbol{\Psi}$ is a diagonal matrix from $\mathbb{R}^{n \times n}$.

The diagonal form of $\boldsymbol{\Psi}$ comes from the assumption that the measurements are taken independently, thus the noise at the components is mutually independent.

Therefore, the observations components of $\boldsymbol{x}$ are mutually independent if the factors are known (only the noise is the random component).

We assume that $\boldsymbol{\Psi}$ and $\boldsymbol{z}$ are independent that means that the noise is independent of the signal strength.

The free parameters of the model are $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$. Exactly these free parameters explain the variance in the observations $\boldsymbol{x}$, where $\boldsymbol{\Lambda}$ explains the dependent part whereas $\boldsymbol{\Psi}$ explains the independent part of the variance.

The free parameters can be estimated by maximum likelihood Jöreskog (1967) based on the Expectation-Maximization (EM) optimization technique Dempster et al. (1977).

If a prior probability (a Gaussian) is used for the loading matrix then maximum a posteriori estimation of the parameters based on an Expectation-Maximization (EM) optimization is possible Hochreiter et al. (2006); Talloen et al. (2007); Clevert and Hochreiter (2007).

4.2 Matrix Factorization and Biclusters

Using all samples $\boldsymbol{x}_j, \ 1 \leq \ j \leq \ l$ we can write the model for the data in matrix form as:

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

This is *matrix factorization* with additive noise.

Here $\boldsymbol{X}$ and $\boldsymbol{\Upsilon}$ are from $\mathbb{R}^{n \times l}$, $\boldsymbol{\Lambda}$ from $\mathbb{R}^{n \times p}$, and $\boldsymbol{Z}$ from $\mathbb{R}^{p \times l}$.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

Here the Λ_i are from $\mathbb{R}^n$ and the $\boldsymbol{Z}_i$ from R^l, where $\boldsymbol{Z}_i$ is the ith column vector of matrix $\boldsymbol{Z}^T$ or, equivalently, $\left(\boldsymbol{Z}_i\right)^T$ is the ith row vector of matrix $\boldsymbol{Z}$.

More exactly would be to write $\left(\boldsymbol{Z}_i^T\right)^T$ instead of $\left(\boldsymbol{Z}_i\right)^T$.

Note, that we wrote $\boldsymbol{Z}_i$ instead of $\boldsymbol{z}_i$ for matrix factorization however for the generative model we use $\boldsymbol{z}_i$ and $\boldsymbol{z}$ to generate sample $\boldsymbol{x}_i$ or $\boldsymbol{x}$. If $\boldsymbol{\Lambda}_i$ is sparse (having many zero components) and if $\boldsymbol{Z}_i$ is sparse, then component i can be viewed as a *bicluster*. Observations that are nonzero and samples that are nonzero are members of bicluster i.

If the members of the bicluster are consecutively ordered in the observations and consecutively ordered in the samples (bicluster members are one after the other in the observations and are one after the other in the samples) then $\boldsymbol{\Lambda}_i \ (\boldsymbol{Z}_i)^T$ is a matrix with a block of nonzero entries and zero otherwise.

Model assumptions for biclustering:

- sparseness of the factors through a prior on the factors by an component-wise independent *Laplace* distribution:

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

- sparseness of the loadings

  FABIA: Loading prior is independent *Laplace*

$$p(\boldsymbol{\Lambda}_i) \ = \ \left(\frac{1}{\sqrt{2}}\right)^n \prod_{k=1}^{n} e^{- \ \sqrt{2} \ |\Lambda_{ik}|}$$

  OR

  FABIAS: Loading prior has finite support within a l_1-*norm constraint*

$$p(\boldsymbol{\Lambda}_i) \ = \ c \quad \text{for} \quad \|\boldsymbol{\Lambda}_i\|_1 \ \leq \ c_1$$

$$p(\boldsymbol{\Lambda}_i) \ = \ 0 \quad \text{for} \quad \|\boldsymbol{\Lambda}_i\|_1 \ > \ c_1$$

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

$$\text{sparseness}(\boldsymbol{\Lambda}_i) \ = \ \frac{\sqrt{n} \ - \ \sum_{k=1}^{n} |\Lambda_{ik}| \ / \ \sum_{k=1}^{n} \Lambda_{ik}^2}{\sqrt{n} \ - \ 1}$$

  The sparseness is constraint to a certain value for each i and $\boldsymbol{\Lambda}_i$.

- *Gaussian* independent noise

$$p(\boldsymbol{\epsilon}) \ = \ \left(\frac{1}{\sqrt{2 \ \pi}}\right)^n \left(\prod_{k=1}^{n} \frac{1}{\sigma_k}\right) \ e^{\sum_{k=1}^{l} \frac{\epsilon_k^2}{\sigma_k^2}} \ = \ \left(\frac{1}{\sqrt{2 \ \pi}}\right)^l \ \frac{1}{\sqrt{|\boldsymbol{\Psi}|}} \ e^{\boldsymbol{\epsilon}^T \ \boldsymbol{\Psi}^{-1} \ \boldsymbol{\epsilon}}$$

Preprocessing of the data:

- data should be normalized to *zero mean* because the model requires that

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

- we recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

  The model covariance is:

  $$\mathrm{E}(\boldsymbol{x}\ \boldsymbol{x}^T) \ = \ \boldsymbol{\Lambda}\mathrm{E}(\boldsymbol{z}\boldsymbol{z}^T)\boldsymbol{\Lambda}^T \ + \ \boldsymbol{\Lambda}\mathrm{E}(\boldsymbol{z})\mathrm{E}(\boldsymbol{\epsilon}^T) \ + \ \mathrm{E}(\boldsymbol{z})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \ + \ \mathrm{E}(\boldsymbol{\epsilon}\ \boldsymbol{\epsilon}^T) \ = \ \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \ + \ \mathrm{diag}(\sigma_k^2) \ = \ \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \ + \ \boldsymbol{\Psi}$$

  Normalizing the data to variance one for each component gives

  $$\boldsymbol{\Psi}_{kk} \ + \ \left(\boldsymbol{\Lambda}^k\right)^T \boldsymbol{\Lambda}^k \ = \ \sigma_k^2 \ + \ \left(\boldsymbol{\Lambda}^k\right)^T \boldsymbol{\Lambda}^k \ = \ 1$$

  Here the length of the row vector $\boldsymbol{\Lambda}^k$ is p in contrast to the column vectors $\boldsymbol{\Lambda}_i$ which length is n.

Estimated values:

- Estimated parameters through maximizing the posterior of the parameters: $\boldsymbol{\Lambda}$ and $\sigma_k^2 \ = \ \boldsymbol{\Psi}_{kk}$
- Estimated latent variables, the factors, through posterior on the latent variables: $\boldsymbol{Z}$
- Estimated noise free data through the parameter posterior and latent variable posterior: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
- Estimated biclusters through model assumption: $\boldsymbol{\Lambda}_i\ \left(\boldsymbol{Z}_i\right)^T$. Here the large values indicate the bicluster (in the ideal case the nonzero values would indicate the bicluster).

The independence of the hidden variables in the second moments can be achieved through variable transformation. Note, that this is no restriction because a factor analysis model cannot be uniquely identified with respect to orthogonal transformations, permutations, and scaling (including change of sign). For one example we have

$$\boldsymbol{\Lambda}\ \boldsymbol{z} \ = \ \boldsymbol{\Lambda}\ \boldsymbol{A}^{1/2}\ \boldsymbol{A}^{-1/2}\ \boldsymbol{z}\ ,$$

where $\boldsymbol{A}^{1/2}$ is an invertible matrix.

If

$$\mathrm{E}\left(\boldsymbol{z}\ \boldsymbol{z}^T\right) \ = \ \boldsymbol{A}$$

with a symmetric positive definite matrix $\boldsymbol{A}$ then

$$\boldsymbol{x} \ \sim \ \mathcal{N}\left(\boldsymbol{0}\ ,\ \boldsymbol{\Lambda}\ \boldsymbol{A}\ \boldsymbol{\Lambda}^T \ + \ \boldsymbol{\Psi}\right)\ .$$

Whitening leads now to

$$\hat{\boldsymbol{\Lambda}} \ = \ \boldsymbol{\Lambda}\ \boldsymbol{A}^{1/2}$$

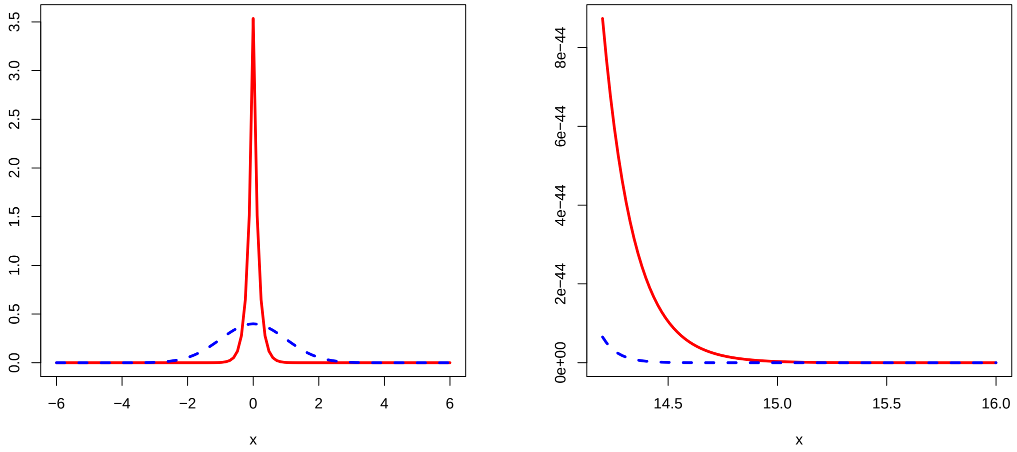

Figure 2: Left: the mode of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution. Right: The tails of a Laplace (red, solid) vs. a Gaussian (dashed, blue) distribution

and

$$\hat{\boldsymbol{z}} \ = \ \boldsymbol{A}^{-1/2} \ \boldsymbol{z}$$

which gives

$$\mathrm{E}\left(\hat{\boldsymbol{z}} \ \hat{\boldsymbol{z}}^T\right) \ = \ \boldsymbol{A}^{-1/2} \ \boldsymbol{A} \ \boldsymbol{A}^{-1/2} \ = \ \boldsymbol{I} \ ,$$

where

$$\hat{\boldsymbol{\Lambda}} \ \hat{\boldsymbol{z}} \ = \ \boldsymbol{\Lambda} \ \boldsymbol{A}^{1/2} \ \ \boldsymbol{A}^{-1/2} \boldsymbol{z} \ = \ \boldsymbol{\Lambda} \ \boldsymbol{z} \ .$$

4.3 Laplace Prior on the Hidden Factors

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.

A prominent example for sparse coding are the neurons in the human brain

In this application the model assumption is that only few sample show specific patterns in the observations.

Laplace prior is suited for modeling strong activation for few samples while being otherwise almost constant. Fig. 2 left shows the mode compared to a Gaussian which shows that the Laplace factors vary less about the mode and the likelihood of the mode is much larger than neighboring values. Fig. 2 right shows the tails where it can be seen that Laplace has higher likelihood for large values than Gaussian.

4.4 Model Selection

The likelihood

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

is analytically intractable for Laplacian prior $p\left(\boldsymbol{z}\right)$ (for Gaussian priors the integral can be analytically solved).

That means the likelihood, the objective, cannot be given explicitly and EM or gradient-based methods do not work.

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

The variational likelihood is equipped by and additional parameter $\boldsymbol{\xi}$, where the maximum with respect to this parameter is the true likelihood:

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

Using this lower bound on the likelihood we can derive another lower bound on the likelihood:

$$\begin{aligned}
&\log\; p(\boldsymbol{x}) \;\geq\; \log\; p(\boldsymbol{x}|\boldsymbol{\xi}) \;=\; \int Q(\boldsymbol{z})\; \log\; p(\boldsymbol{x}|\boldsymbol{\xi})\; d\boldsymbol{z} \;=\\
&\int Q(\boldsymbol{z})\; \log \frac{Q(\boldsymbol{z})}{p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\xi})}\; d\boldsymbol{z} - \int Q(\boldsymbol{z})\; \log \frac{Q(z)}{p(\boldsymbol{z}, \boldsymbol{x} \mid \boldsymbol{\xi})}\; d\boldsymbol{z} \;\geq\\
&\int Q(\boldsymbol{z})\; \log p(\boldsymbol{z}, \boldsymbol{x} \mid \boldsymbol{\xi})\; d\boldsymbol{z} \;,
\end{aligned}$$

where we used

$$p(\boldsymbol{x} \mid \boldsymbol{\xi}) \;=\; \frac{p(\boldsymbol{z}, \boldsymbol{x} \mid \boldsymbol{\xi})}{p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\xi})} \;.$$

We now set for each sample $\boldsymbol{x}_j$ like for the standard EM algorithm

$$Q_i(\boldsymbol{z}_j) \;=\; p\left(\boldsymbol{z}_j \mid \boldsymbol{x}_j, \boldsymbol{\Lambda}^{\mathrm{old}}, \boldsymbol{\Psi}^{\mathrm{old}}\right) \;.$$

Further we have

$$p(\boldsymbol{z}, \boldsymbol{x} \mid \boldsymbol{\xi}) \;=\; p(\boldsymbol{x} \mid \boldsymbol{z})\; p(\boldsymbol{z} \mid \boldsymbol{\xi}) \;.$$

The variational approach gives

$$\begin{aligned}
&p(\boldsymbol{z}) \;=\; \left(\frac{1}{\sqrt{2}}\right)^p \prod_{i=1}^{p} e^{-\;\sqrt{2}\;|z_i|} \;=\\
&\arg\max_{\boldsymbol{\xi}} p(\boldsymbol{z} \;\mid\; \boldsymbol{\xi}) \;=\\
&\arg\max_{\boldsymbol{\xi}} \;\left(\frac{1}{\sqrt{2}}\right)^p \prod_{i=1}^{p} \phi(\xi_i)\; \mathcal{N}\left(z_i \;,\; |\xi_i|\right) \;,
\end{aligned}$$

where

$$\phi(\xi) \ = \ \exp \left(- \ \frac{1}{2} \ |\xi| \right) \ \sqrt{2 \ \pi \ |\xi|} \ .$$

Maximizing the lower bound on the likelihood with respect to $\boldsymbol{\xi}$ gives

$$\xi_{ij} \ = \ \sqrt{\int p(\boldsymbol{z}_i \ \mid \ \boldsymbol{x}_i) \ z_{ij}^2 \ d\boldsymbol{z}_i} \ .$$

For computing ξ_{ij}^2 we can use

$$p(\boldsymbol{z} \ \mid \ \boldsymbol{x}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}) \ \geq \ p(\boldsymbol{z} \ \mid \ \boldsymbol{x}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}, \boldsymbol{\xi}^{\text{old}}) \ = \\ \frac{p(\boldsymbol{x} \ \mid \ \boldsymbol{z}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}) \ p(\boldsymbol{z} \ \mid \ \boldsymbol{\xi}^{\text{old}})}{\int p(\boldsymbol{x} \ \mid \ \boldsymbol{z}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}) \ p(\boldsymbol{z} \ \mid \ \boldsymbol{\xi}^{\text{old}}) \ d\boldsymbol{z}}$$

where we know both the variational $p(\boldsymbol{z} \mid \boldsymbol{\xi}^{\text{old}})$ and the Gaussian

$$p(\boldsymbol{x} \ \mid \ \boldsymbol{z}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}) \ = \ \mathcal{N}\left(\boldsymbol{\Lambda} \ \boldsymbol{z} \ , \ \boldsymbol{\Psi}\right) \ .$$

Using

$$\boldsymbol{\Xi} \ = \ \text{diag}(\boldsymbol{\xi})$$

the variational prior is the multimodal Gaussian:

$$\left(\frac{1}{\sqrt{2}} \right)^p \ \phi(\Xi) \ \mathcal{N}\left(\boldsymbol{z} \ , \ \Xi\right) \ .$$

The posterior of $\boldsymbol{z}$ is now basically a product of Gaussians for which we can compute the conditional expectations analytically. The conditional mean is

$$\text{E}\left(\boldsymbol{z}_j \ \mid \ \boldsymbol{x}_j\right) \ = \ \left(\left(\boldsymbol{\Lambda}^{\text{old}}\right)^T \ \left(\boldsymbol{\Psi}^{\text{old}}\right)^{-1} \ \boldsymbol{\Lambda}^{\text{old}} \ + \ \left(\Xi_j^{\text{old}}\right)^{-1} \right)^{-1} \ \left(\boldsymbol{\Lambda}^{\text{old}}\right)^T \ \left(\boldsymbol{\Psi}^{\text{old}}\right)^{-1} \ \boldsymbol{x}_j$$

and the conditional co-variance is

$$\text{E}\left(\boldsymbol{z}_j \ \boldsymbol{z}_j^T \ \mid \ \boldsymbol{x}_j\right) \ = \ \left(\left(\boldsymbol{\Lambda}^{\text{old}}\right)^T \ \left(\boldsymbol{\Psi}^{\text{old}}\right)^{-1} \ \boldsymbol{\Lambda}^{\text{old}} \ + \ \left(\Xi_j^{\text{old}}\right)^{-1} \right)^{-1} + \text{E}\left(\boldsymbol{z}_j \ \mid \ \boldsymbol{x}_j\right) \ \text{E}\left(\boldsymbol{z}_j \ \mid \ \boldsymbol{x}_j\right)^T \ .$$

The update for $\boldsymbol{\xi}$ is now

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

$\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ updates:

1. *Rectified Gauss prior on loadings.* The updates of $\boldsymbol{\Lambda}$ and $\boldsymbol{\Psi}$ are as in the Expectation-Maximization (EM) optimization in Hochreiter et al. (2006); Talloen et al. (2007); Clevert and Hochreiter (2007) with above $\mathrm{E}\left(\boldsymbol{z}_j \mid \boldsymbol{x}_j\right)$ and $\mathrm{E}\left(\boldsymbol{z}_j \boldsymbol{z}_j^T \mid \boldsymbol{x}_j\right)$.

$$
\begin{aligned}
\boldsymbol{\Lambda}^{\text{Gauss}} &= \left(\frac{1}{l} \sum_{j=1}^{l} \boldsymbol{x}_j \, \mathrm{E}\left(\boldsymbol{z}_j \mid \boldsymbol{x}_j\right)^T \;+\; \mathrm{diag}\left(\boldsymbol{\Psi}^{\text{new}}\right) \, \mathrm{ones}(p)^T \, \frac{\mu_\Lambda}{\sigma_\Lambda^2 \, l} \right) \\
&\quad \left(\frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \, \boldsymbol{z}_j^T \mid \boldsymbol{x}_j\right) \;+\; \mathrm{diag}(\mathrm{ones}(p)) \, \frac{\|\Psi_{ii}^{\text{new}}\|_F}{\sigma_\Lambda^2 \, l} \right)^{-1} , \\
\boldsymbol{\Lambda}^{\text{new}} &= \begin{cases} \boldsymbol{\Lambda}^{\text{Gauss}} & \text{for} \quad \boldsymbol{\Lambda}^{\text{Gauss}} > 0 \\ 0 & \text{for} \quad \boldsymbol{\Lambda}^{\text{Gauss}} \leq 0 \end{cases} , \\
\mathrm{diag}\left(\boldsymbol{\Psi}^{\text{new}}\right) &= \mathrm{diag}\left(\frac{1}{l} \sum_{j=1}^{l} \boldsymbol{x}_j \, \boldsymbol{x}_j^T \;-\; \boldsymbol{\Lambda}^{\text{new}} \, \frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \mid \boldsymbol{x}_j\right) \boldsymbol{x}_j^T \right) \;+ \\
&\quad \frac{1}{\sigma_\Lambda^2 \, l} \, \mathrm{diag}\left(\boldsymbol{\Lambda}^{\text{new}} \, \left(\mu_\Lambda \, \mathrm{diag}\left(\boldsymbol{\Psi}^{\text{new}}\right) \, \mathrm{ones}(p)^T \;-\; \|\Psi_{ii}^{\text{new}}\|_F \, \boldsymbol{\Lambda}^{\text{new}} \right)^T \right) .
\end{aligned}
$$

2. *Laplace prior on loadings.* For the Laplace prior on $\boldsymbol{\Lambda}$ we have

$$
\boldsymbol{\Lambda} \;=\; \frac{\frac{1}{l} \, \sum_{j=1}^{l} \boldsymbol{x}_j \, \mathrm{E}\left(\boldsymbol{z}_j \mid \boldsymbol{x}_j\right)^T \;-\; \frac{1}{l} \, \alpha \, \boldsymbol{\Psi}^{\text{old}} \, \mathrm{sign}\left(\boldsymbol{\Lambda}^{\text{old}}\right)}{\frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \, \boldsymbol{z}_j^T \mid \boldsymbol{x}_j\right)} .
$$

3. *Finite support prior on loadings.* If the prior on $\boldsymbol{\Lambda}$ has finite support the we perform a projection after each $\boldsymbol{\Lambda}$ update.

   The projection is done according to Hoyer (2004): given an l_1-norm and an l_2-norm minimize the Euclidean distance to the original vector (currently the l_2-norm is fixed to 1). The projection is a convex quadratic problem which is solved iteratively where at each iteration at least one component is set to zero.

   Instead of the l_1-norm following sparseness condition is used:

$$
\mathrm{sparseness}(\boldsymbol{v}) \;=\; \frac{\sqrt{n} \;-\; \sum_{k=1}^{n} |v_k| \;/\; \sum_{k=1}^{n} v_k^2}{\sqrt{n} \;-\; 1} .
$$

4.5 Extreme Sparse Priors

4.5.1 Extreme sparse prior on loadings

For an extreme sparse prior on the loadings we use the

1. *generalized Gaussians*
$$p(z) \ \propto \ \exp\left(- \ |z|^{\beta}\right) \ ,$$
where $0 \ < \ \beta \ \leq \ 1$.
2. *Jeffrey's prior*
$$p(z) \ \propto \ \exp\left(- \ \ln|z|\right) \ \propto \ \frac{1}{|z|}$$
3. *improper prior*
$$p(z) \ \propto \ \exp\left(|z|^{-\beta}\right) \ ,$$
where $0 \ < \ \beta$. This distribution may only exist on the interval $[\epsilon, a]$ with $0 \ < \ \epsilon \ < \ a$. We assume that ϵ is smaller than the smallest absolute $\mathbf{\Lambda}$-values observed in our algorithm except for zero.

The derivatives of the negative "log"-densities are

1. *generalized Gaussians*
$$\frac{\partial\left(- \ln p(z)\right)}{\partial z} \ \propto \ \beta |z|^{\beta \ - \ 1} \ ,$$
where $0 \ < \ \beta \ \leq \ 1$.
2. *Jeffrey's prior*
$$\frac{\partial\left(- \ln p(z)\right)}{\partial z} \ \propto \ \frac{1}{|z|}$$
3. *improper prior*
$$\frac{\partial\left(- \ln p(z)\right)}{\partial z} \ \propto \ \beta |z|^{-\beta \ - \ 1}$$
where $0 \ < \ \beta$.

Let us denote the negative exponent of $|z|$ in the derivatives by s. All $s \ \geq \ 0$ are possible, where $s = 0$ corresponds to the Laplace prior and larger s represent sparser priors.

For the M-step of the EM-algorithm on the posterior on the parameters we have to solve

$$\mathbf{\Lambda} \ \frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \ \boldsymbol{z}_j^T \ \mid \ \boldsymbol{x}_j\right) \ - \ \frac{1}{l} \ \sum_{j=1}^{l} \boldsymbol{x}_j \ \mathrm{E}\left(\boldsymbol{z}_j \ \mid \ \boldsymbol{x}_j\right)^T \ + \ \frac{1}{l} \ \alpha \ \mathbf{\Psi}^{\mathrm{old}} \ |\mathbf{\Lambda}|^{-s} \ = \ \mathbf{0} \ .$$

Without the prior, the solution would be

$$\mathbf{\Lambda}^{\mathrm{tmp}} \ = \ \frac{\frac{1}{l} \ \sum_{j=1}^{l} \boldsymbol{x}_j \ \mathrm{E}\left(\boldsymbol{z}_j \ \mid \ \boldsymbol{x}_j\right)^T}{\frac{1}{l} \sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \ \boldsymbol{z}_j^T \ \mid \ \boldsymbol{x}_j\right)}$$

which we use to obtain the update

$$\boldsymbol{\Lambda} \; = \; \boldsymbol{\Lambda}^{\mathrm{tmp}} \; - \; \frac{\mathrm{sign}\left(\boldsymbol{\Lambda}^{\mathrm{tmp}}\right) \; \min\left(\mathrm{abs}\left(\boldsymbol{\Lambda}^{\mathrm{tmp}}\right) \; , \; \frac{1}{l} \; \alpha \; \boldsymbol{\Psi}^{\mathrm{old}} \; \left|\boldsymbol{\Lambda}^{\mathrm{tmp}}\right|^{-s}\right)}{\frac{1}{l}\sum_{j=1}^{l} \mathrm{E}\left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j\right)} \; ,$$

where we set an overshoot over zero to zero.

In the algorithms s is denoted by `spl`.

4.5.2 Extreme sparse prior on factors

For an extreme sparse prior on the factors we use the

1. *generalized Gaussians*
$$p(z) \; \propto \; \exp\left(- \; |z|^{\beta}\right) \; ,$$
where $0 \; < \; \beta \; \leq \; 1$.
2. *Jeffrey's prior*
$$p(z) \; \propto \; \exp\left(- \; \ln|z|\right) \; \propto \; \frac{1}{|z|}$$
3. *improper prior*
$$p(z) \; \propto \; \exp\left(|z|^{-\beta}\right) \; ,$$
where $0 \; < \; \beta$. This distribution may only exist on the interval $[\epsilon, a]$ with $0 \; < \; \epsilon \; < \; a$. We assume that ϵ is smaller than the smallest absolute $\boldsymbol{\Lambda}$-values observed in our algorithm except for zero.

We want to represent the priors through a convex variational form. According to Palmer et al. (2006) we have to show that

$$g(x) \; = \; - \; \ln p(\sqrt{z})$$

is increasing and concave for $z > 0$.

The first and second order derivatives of g are

1. *generalized Gaussians* with $g \; = \; z^{\beta/2}$
$$\frac{\partial g(z)}{\partial z} \; = \; \frac{\beta}{2}|z|^{\beta/2 \; - \; 1} \; > 0 \; ,$$
$$\frac{\partial^2 g(z)}{\partial z^2} \; = \; - \; \frac{\beta}{2} \; (1 \; - \; \beta/2) \; |z|^{\beta/2 \; - \; 2} \; < 0$$
where $0 \; < \; \beta \; \leq \; 1$.

2. *Jeffrey's prior* $g \; = \; \frac{1}{2} \; \ln |z|$

$$\frac{\partial g(z)}{\partial z} \; = \; \frac{1}{2} \, \frac{1}{|z|} \; > 0$$

$$\frac{\partial^2 g(z)}{\partial z^2} \; = \; - \; \frac{1}{2} \, \frac{1}{|z|^2} \; < 0$$

3. *improper prior* with $g \; = \; - \; z^{- \; \beta/2}$

$$\frac{\partial g(z)}{\partial z} \; = \; \frac{\beta}{2} \; |z|^{-\beta/2 \; - \; 1} \; > 0$$

$$\frac{\partial^2 g(z)}{\partial z^2} \; = \; - \; \frac{\beta}{2} \; (1 \; + \; \beta/2) \; |z|^{- \; \beta/2 \; - \; 2} \; < 0$$

where $0 \; < \; \beta$.

This shows that for these priors the g is increasing and concave for $z \; > \; 0$ which allows us to to represent the priors through a convex variational form.

According to Palmer et al. (2006) the update for the variational parameter $\boldsymbol{\xi}$ is

$$\boldsymbol{\xi}_j \; = \; 2 \, \frac{\partial g}{\partial z} \left(\text{diagE} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right) \right) \; .$$

This gives following updates

1. *generalized Gaussians* with $g \; = \; z^{\beta/2}$

$$\boldsymbol{\xi}_j \; = \; \beta \; \text{diag} \left(\text{E} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right) \right)^{\beta/2 \; - \; 1} \; ,$$

where $0 \; < \; \beta \; \leq \; 1$.

2. *Jeffrey's prior* $g \; = \; \frac{1}{2} \; \ln |z|$

$$\boldsymbol{\xi}_j \; = \; \text{diag} \left(\text{E} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right) \right)^{-1}$$

3. *improper prior* with $g \; = \; - \; z^{- \; \beta/2}$

$$\boldsymbol{\xi}_j \; = \; \beta \; \text{diag} \left(\text{E} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right) \right)^{-\beta/2 \; - \; 1}$$

where $0 \; < \; \beta$.

If the negative exponent of $\text{E} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right)$ in the update is denoted by s then we cover all $s \; \geq \; 1/2$. The smallest $s \; = \; 1/2$ $(\beta = 1)$ represents the Laplace prior and $s \; > \; 1/2$ leads to sparser priors.

For the update of the variational variable we have

$$\boldsymbol{\xi}_j \; \propto \; \left(\text{diagE} \left(\boldsymbol{z}_j \; \boldsymbol{z}_j^T \; \mid \; \boldsymbol{x}_j \right)^{-s} \right) \; .$$

In the algorithms s is denoted by `spz`.

4.6 Informative Biclusters

4.6.1 Information of the Latent Variables on Observations

We will measure the information in biclusters through the mutual information between $\boldsymbol{z}$ and $\boldsymbol{x}$, that is how much information about $\boldsymbol{x}$ is contained in $\boldsymbol{z}$.

This idea is the basis of the I/NI calls in Talloen et al. (2007).

The entropy of a multivariate Gaussian

$$\boldsymbol{x} \ \sim \ \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \tag{1}$$

with density

$$p(\boldsymbol{x}) \ = \ \frac{1}{(2 \ \pi)^{n/2} \ |\boldsymbol{\Sigma}|^{1/2}} \exp\left(- \ \frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^T \ \boldsymbol{\Sigma}^{-1} \ (\boldsymbol{x} - \boldsymbol{\mu}) \right) \tag{2}$$

is

$$\mathrm{H}\left(\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)\right) \ = \ \ln \left(\sqrt{(2 \ \pi \ e)^n \ |\boldsymbol{\Sigma}|} \right) \ . \tag{3}$$

The mutual information is defined as

$$\mathrm{I}(\boldsymbol{x}; \boldsymbol{z}) \ = \ \mathrm{H}(\boldsymbol{x}) \ - \ \mathrm{H}(\boldsymbol{x} \mid \boldsymbol{z}) \ . \tag{4}$$

In our model we have

$$\boldsymbol{x}_j \ \sim \ \mathcal{N}\left(\boldsymbol{0} \ , \ \boldsymbol{\Psi} \ + \ \boldsymbol{\Lambda} \ \ \Xi_j \ \boldsymbol{\Lambda}^T\right) \tag{5}$$

and

$$\boldsymbol{x}_j \mid \boldsymbol{z}_j \ \sim \ \mathcal{N}\left(\boldsymbol{\Lambda} \ \boldsymbol{z}_j \ , \ \boldsymbol{\Psi}\right) \ . \tag{6}$$

Thus, the mutual information is

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j; \boldsymbol{z}_j) \ = \ \mathrm{H}(\boldsymbol{x}_j) \ - \ \mathrm{H}(\boldsymbol{x}_j \mid \boldsymbol{z}_j) \ = \\
&\ln \left(\sqrt{(2 \ \pi \ e)^n \ |\boldsymbol{\Psi} \ + \ \boldsymbol{\Lambda} \ \ \Xi_j \ \boldsymbol{\Lambda}^T|} \right) \ - \ \ln \left(\sqrt{(2 \ \pi \ e)^n \ |\boldsymbol{\Psi}|} \right) \ = \\
&\frac{1}{2} \ \ln \left| \left(\boldsymbol{\Psi} \ + \ \boldsymbol{\Lambda} \ \ \Xi_j \ \boldsymbol{\Lambda}^T \right) \ \boldsymbol{\Psi}^{-1} \right| \ = \\
&\frac{1}{2} \ \ln \left| \boldsymbol{I}_n \ + \ \boldsymbol{\Lambda} \ \ \Xi_j \ \boldsymbol{\Lambda}^T \ \boldsymbol{\Psi}^{-1} \right| \ = \\
&\frac{1}{2} \ \ln \left| \boldsymbol{I}_p \ + \ \boldsymbol{\Lambda}^T \ \boldsymbol{\Psi}^{-1} \ \boldsymbol{\Lambda} \ \Xi_j \right| \ = \\
&\frac{1}{2} \ \ln \left| \boldsymbol{I}_p \ + \ \Xi_j \ \boldsymbol{\Lambda}^T \ \boldsymbol{\Psi}^{-1} \ \boldsymbol{\Lambda} \right| \ ,
\end{aligned} \tag{7}$$

where we applied the "Sylvester's theorem for determinants" which is special case of the generalization of the "matrix determinant lemma" (`http://en.wikipedia.org/wiki/Matrix_determinant_lemma`):

$$\left|\boldsymbol{A} \; + \; \boldsymbol{U} \; \boldsymbol{V}^T\right| \; = \; \left|\boldsymbol{I} \; + \; \boldsymbol{V}^T \; \boldsymbol{A}^{-1} \; \boldsymbol{U}\right| \; |\boldsymbol{A}| \; . \tag{8}$$

Above formula can also be obtained from

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j) \; = \; \mathrm{H}(\boldsymbol{z}_j) \; - \; \mathrm{H}(\boldsymbol{z}_j \mid \boldsymbol{x}_j) \; = \\
&\ln\left(\sqrt{(2\;\pi\;e)^n \; |\boldsymbol{\Xi}_j|}\right) \; - \; \ln\left(\sqrt{(2\;\pi\;e)^n \; \left|\left(\boldsymbol{\Lambda}^T \; \boldsymbol{\Psi}^{-1} \; \boldsymbol{\Lambda} \; + \; \boldsymbol{\Xi}_j^{-1}\right)^{-1}\right|}\right) \; = \\
&\frac{1}{2} \; \ln\left|\boldsymbol{\Xi}_j \; \left(\boldsymbol{\Lambda}^T \; \boldsymbol{\Psi}^{-1} \; \boldsymbol{\Lambda} \; + \; \boldsymbol{\Xi}_j^{-1}\right)\right| \; = \\
&\frac{1}{2} \; \ln\left|\boldsymbol{I}_p \; + \; \boldsymbol{\Xi}_j \; \boldsymbol{\Lambda}^T \; \boldsymbol{\Psi}^{-1} \; \boldsymbol{\Lambda}\right| \; .
\end{aligned} \tag{9}$$

Independence of the factors (up to the second moment) results in a diagonal matrix of the co-variance of $\boldsymbol{z}$. Diagonal $\boldsymbol{\Xi}_j$ allows following expansion

$$\boldsymbol{\Lambda} \; \boldsymbol{\Xi}_j \; \boldsymbol{\Lambda}^T \; = \; \sum_{i=1}^{p} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T \; . \tag{10}$$

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j) \; = \; \frac{1}{2} \; \ln\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Lambda} \;\; \boldsymbol{\Xi}_j \; \boldsymbol{\Lambda}^T \; \boldsymbol{\Psi}^{-1}\right| \; = \\
&\frac{1}{2} \; \ln\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \boldsymbol{\Lambda} \;\; \boldsymbol{\Xi}_j \; \boldsymbol{\Lambda}^T\right| \; = \\
&\frac{1}{2} \; \ln\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \sum_{i=1}^{p} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right| \; = \\
&\frac{1}{2} \; \ln\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \sum_{i=1}^{p-1} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T \; + \; \boldsymbol{\Psi}^{-1} \; \Xi_{jp} \; \boldsymbol{\Lambda}_p \; \boldsymbol{\Lambda}_p^T\right| \; = \\
&\frac{1}{2} \; \ln\Bigg(\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \sum_{i=1}^{p-1} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right| \\
&\left(1 \; + \; \Xi_{jp} \; \boldsymbol{\Lambda}_p^T \; \boldsymbol{\Psi}^{-1} \left(\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \sum_{i=1}^{p-1} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right)^{-1} \; \boldsymbol{\Lambda}_p\right)\Bigg) \; = \\
&\frac{1}{2} \; \ln\Bigg(\left|\boldsymbol{I}_n \; + \; \boldsymbol{\Psi}^{-1} \; \sum_{i=1}^{p-1} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right| \\
&\left(1 \; + \; \Xi_{jp} \; \boldsymbol{\Lambda}_p^T \; \left(\boldsymbol{\Psi} \; + \; \sum_{i=1}^{p-1} \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right)^{-1} \; \boldsymbol{\Lambda}_p\right)\Bigg) \; ,
\end{aligned} \tag{11}$$

where we again used the the generalization of the “matrix determinant lemma” with

$$\boldsymbol{A} \;=\; \boldsymbol{I}_n \;+\; \sum_{i=1}^{p-1} \Xi_{ji}\; \boldsymbol{\Lambda}_i\; \boldsymbol{\Lambda}_i^T\; \boldsymbol{\Psi}^{-1}\;. \tag{12}$$

The last formula can be interpreted from information theory as

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j) \;=\; \mathrm{H}(\boldsymbol{x}_j) \;-\; \mathrm{H}(\boldsymbol{x}_j \mid \boldsymbol{z}_j) \;= \\
&\mathrm{H}(\boldsymbol{x}_j) \;-\; \mathrm{H}(\boldsymbol{x}_j \mid z_{ji}) \;+\; \mathrm{H}(\boldsymbol{x}_j \mid z_{ji}) \;-\; \mathrm{H}(\boldsymbol{x}_j \mid \boldsymbol{z}_j) \;= \\
&\mathrm{I}(\boldsymbol{x}_j; z_{ji}) \;+\; \mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j \mid z_{ji})
\end{aligned} \tag{13}$$

with

$$\mathrm{I}(\boldsymbol{x}_j; z_{ji}) \;=\; \frac{1}{2}\;\ln\left(1 \;+\; \Xi_{ji}\; \boldsymbol{\Lambda}_i^T\; \left(\boldsymbol{\Psi} \;+\; \sum_{t=1:t\neq i}^{p} \Xi_{jt}\; \boldsymbol{\Lambda}_t\; \boldsymbol{\Lambda}_t^T\right)^{-1}\; \boldsymbol{\Lambda}_i\right) \tag{14}$$

and

$$\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j \mid z_{ji}) \;=\; \frac{1}{2}\;\ln\left|\boldsymbol{I}_n \;+\; \boldsymbol{\Psi}^{-1}\; \sum_{t=1:t\neq i}^{p} \Xi_{jt}\; \boldsymbol{\Lambda}_t\; \boldsymbol{\Lambda}_t^T\right|\;. \tag{15}$$

Now inductively we obtain

$$\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j) \;=\; \frac{1}{2}\sum_{i=1}^{p}\ln\left(1 \;+\; \Xi_{ji}\; \boldsymbol{\Lambda}_i^T\; \left(\boldsymbol{\Psi} \;+\; \sum_{t=1}^{i-1} \Xi_{jt}\; \boldsymbol{\Lambda}_t\; \boldsymbol{\Lambda}_t^T\right)^{-1}\; \boldsymbol{\Lambda}_i\right)\;. \tag{16}$$

The last formula can be interpreted from information theory as

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j) \;=\; \mathrm{I}(\boldsymbol{x}_j; z_{ji_1}) \;+\; \mathrm{I}(\boldsymbol{x}_j;\boldsymbol{z}_j \mid z_{ji_1}) \;= \\
&\mathrm{I}(\boldsymbol{x}_j; z_{ji_1}) \;+\; \mathrm{I}(\boldsymbol{x}_j; z_{ji_2} \mid z_{ji_1}) \;+\; \mathrm{I}(\boldsymbol{x}_j; z_{ji_3} \mid z_{ji_1}, z_{ji_2}) \;+\; \ldots \;+ \\
&\mathrm{I}(\boldsymbol{x}_j; z_{ji_p} \mid z_{ji_1}, z_{ji_2}, \ldots, z_{ji_{p-1}})\;.
\end{aligned} \tag{17}$$

The mutual information between $\boldsymbol{X}$ and $\boldsymbol{Z}$ is the sum of the mutual informations between $\boldsymbol{x}_j$ and $\boldsymbol{z}_j$. This follows from the independence of $\boldsymbol{x}_j$ and the independence of $\boldsymbol{z}_j$ across j and the fact that the entropy is additive for independent variables:

$$\mathrm{I}(\boldsymbol{X};\boldsymbol{Z}) \;=\; \frac{1}{2}\;\sum_{j=1}^{l}\ln\left|\boldsymbol{I}_p \;+\; \boldsymbol{\Lambda}^T\; \boldsymbol{\Psi}^{-1}\; \boldsymbol{\Lambda}\; \boldsymbol{\Xi}_j\right|\;. \tag{18}$$

4.6.2 Information of One Latent Variable on Observations

Now we can consider the case if one factor z_i is removed from the final model. This corresponds to setting

$$\Xi_{ji} \; = \; 0 \tag{19}$$

and therefore the explained covariance

$$\Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T \tag{20}$$

is lost.

We have

$$\boldsymbol{x}_j \mid (\boldsymbol{z}_j \setminus z_{ji}) \;\sim\; \mathcal{N}\left(\boldsymbol{\Lambda} \; \boldsymbol{z}_j|_{z_{ji}=0} \;,\; \boldsymbol{\Psi} \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right) \;. \tag{21}$$

The information of this factor in the context of the other factors can now be accessed by

$$\begin{aligned}
&\mathrm{I}(\boldsymbol{x}_j; z_{ji} \mid (\boldsymbol{z}_j \setminus z_{ji})) \;=\; \mathrm{H}(\boldsymbol{x}_j \mid (\boldsymbol{z}_j \setminus z_{ji})) \;-\; \mathrm{H}(\boldsymbol{x}_j \mid \boldsymbol{z}_j) \;= \\
&\ln\left(\sqrt{(2\;\pi\;e)^n \;\; |\boldsymbol{\Psi} \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T|}\right) \;-\; \ln\left(\sqrt{(2\;\pi\;e)^n \;\; |\boldsymbol{\Psi}|}\right) \;= \\
&\frac{1}{2} \; \ln\left|\left(\boldsymbol{\Psi} \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T\right) \; \boldsymbol{\Psi}^{-1}\right| \;= \\
&\frac{1}{2} \; \ln\left|\boldsymbol{I}_n \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i \; \boldsymbol{\Lambda}_i^T \boldsymbol{\Psi}^{-1}\right| \;= \\
&\frac{1}{2} \; \ln\left(1 \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i^T \boldsymbol{\Psi}^{-1} \boldsymbol{\Lambda}_i\right) \;.
\end{aligned} \tag{22}$$

The mutual information between $\boldsymbol{X}$ and $\boldsymbol{z}^i$ is the sum of the mutual informations between $\boldsymbol{x}_j$ and z_{ji}. This follows from the independence of $\boldsymbol{x}_j$ and the independence of z_{ji} and the fact the entropy is additive for independent variables:

$$\mathrm{I}(\boldsymbol{X}; \boldsymbol{z}^i \mid (\boldsymbol{Z} \setminus \boldsymbol{z}^i)) \;=\; \frac{1}{2} \sum_{j=1}^{l} \ln\left(1 \;+\; \Xi_{ji} \; \boldsymbol{\Lambda}_i^T \boldsymbol{\Psi}^{-1} \boldsymbol{\Lambda}_i\right) \;. \tag{23}$$

4.7 Extracting Biclusters

After the model is selected the biclusters should be extracted. A bicluster is given by

$$\boldsymbol{\Lambda}_i \; (\boldsymbol{Z}_i)^T \;\;,$$

but not all Λ_{ik} with k not beloning to the bicluster are zero. The same for z_{ij} with j not beloning to the bicluster.

Therefore biclusters are selected by choosing absolute values Λ_{ik} above a threshold `thresL` and abolute values z_{ij} above a threshold `thresZ`.

4.7.1 One Bicluster for Each Factor

For each factor $\boldsymbol{Z}_i$ only one bicluster is extracted. The bicluster has a specific observation pattern (e.g. each sample as a gene expression pattern) so that the negative pattern does not belong to the bicluster. To decide whether the positive or the negative pattern is extracted we compare the sum of absolute values of the factor above the threshold to the sum of absolute values of the factor below the negative threshold:

$$\sum_{j=1:\ z_{ij} > \text{thresZ}}^{l} z_{ij}$$

and

$$- \sum_{j=1:\ z_{ij} < -\text{thresZ}}^{l} z_{ij} \ .$$

The j contributing to the larger value are considered to belong to the bicluster.

Of course, the opposite bicluster (the negative observation pattern) can also be extracted. This makes sense if bimodal conditions are present where the positive observation pattern is indicative for one condition and the negative observation pattern is indicative for the other condition.

4.7.2 Normalizing the Factors

Because

$$\boldsymbol{\Lambda}\ \boldsymbol{Z} \ = \ \boldsymbol{\Lambda}\ \sqrt{\operatorname{diag}\left(\boldsymbol{Z}\ \boldsymbol{Z}^T\right)}\ \left(\sqrt{\operatorname{diag}\left(\boldsymbol{Z}\ \boldsymbol{Z}^T\right)}\right)^{-1} \boldsymbol{Z}$$

we can define

$$\hat{\boldsymbol{\Lambda}} \ = \ \boldsymbol{\Lambda}\ \sqrt{\operatorname{diag}\left(\boldsymbol{Z}\ \boldsymbol{Z}^T\right)}$$

and

$$\hat{\boldsymbol{Z}} \ = \ \left(\sqrt{\operatorname{diag}\left(\boldsymbol{Z}\ \boldsymbol{Z}^T\right)}\right)^{-1} \ \boldsymbol{Z} \ .$$

This scaling normalizes the moments of the factors to 1.

Now the threshold `thresZ` can be determined more appropriate. We choose $\text{thresZ} \ = \ 0.5$ in our experiments which would mean that 30% of the samples can belong to a bicluster if we assume a Gaussian distribution $(0.5\ (1 + \operatorname{erf}\left(\frac{0.5}{\sqrt{2}}\right))$. However the Gaussian assumption is not true because $\boldsymbol{Z}$ is assumed to be Laplace or even sparser.

For the Laplace distribution with variance 1, we obtain $1 \ - \ \frac{1}{2}\exp(-\sqrt{2}/2) \ \approx \ 0.75$. The value $\text{thresZ} \ = \ 0.5$ corresponds to 25% of the samples can belong to a bicluster.

For the improper distribution $\text{spz} \ = \ 1.0$ that we used in the the experiments, the percentage of samples which belong to a bicluster are smaller for $\text{thresZ} \ = \ 0.5$.

4.7.3 Estimating the Threshold `thresL`

We cannot normalize $\boldsymbol{\Lambda}$ for determining `thresL` because it would interfer with the normalization of $\boldsymbol{Z}$.

Even if the components of the observations $\boldsymbol{x}$ are normalized we cannot estimate `thresL` because biclusters may overlap. Overlapping biclusters might have small individual contributions which sum up to explain the observations.

The straightforward approach would be to extract in bicluster i those $\hat{z}_{ij}$ and $\hat{\Lambda}_{ik}$ which are above their standard deviation in bicluster i:

$$\hat{z}_{ij} \ geq \ \sigma_{\hat{\boldsymbol{z}}_i} \ = \ 1 \ = \ \mathrm{thresZ}$$

and

$$\hat{\Lambda}_{ik} \ \geq \ \sigma_{\hat{\boldsymbol{\Lambda}}_i} \ = \ \mathrm{thresL}_i \ .$$

This assumes that `thresZ` is one, but if `thresZ` is given and not equal 1 then we can correct that by

$$\hat{\Lambda}_{ik} \ \geq \ \frac{\sigma_{\hat{\boldsymbol{\Lambda}}_i}}{\mathrm{thresZ}} \ = \ \frac{\mathrm{thresL}_i}{\mathrm{thresZ}} \ .$$

However we would not regard the contribution of a bicluster compared to other biclusters. Therefore we set `thresL` to the standard deviation $\sigma_{\hat{\boldsymbol{\Lambda}}}$ divided by `thresZ`:

$$\mathrm{thresL} \ = \ \frac{\sigma_{\hat{\boldsymbol{\Lambda}}}}{\mathrm{thresZ}} \ = \ \frac{\sqrt{\frac{1}{p \ n} \sum_{(i,k)=(1,1)}^{(p,n)} \left(\hat{\Lambda}_{ik}\right)^2}}{\mathrm{thresZ}} \ . \tag{24}$$

To estimate the average contribution of biclusters, we compute the second moment vlz of a the bicluster contribution $\hat{\Lambda}_{ik} \ \hat{z}_{ij} \ = \ \Lambda_{ik} \ \boldsymbol{z}_{ij}$:

$$\begin{aligned} \mathrm{vlz} \ &= \ \mathrm{E}\left(\left(\hat{\Lambda}_{ik} \ \hat{z}_{ij}\right)^2\right) \ = \qquad (25) \\ &\frac{1}{p \ l \ n} \sum_{(i,j,k)=(1,1,1)}^{(p,l,n)} \left(\hat{\Lambda}_{ik} \ \hat{z}_{ij}\right)^2 \ = \\ &\frac{1}{p \ n} \sum_{(i,k)=(1,1)}^{(p,n)} \left(\hat{\Lambda}_{ik}\right)^2 \ \frac{1}{l} \sum_{j=1}^{l} \left(\hat{z}_{ij}\right)^2 \ = \\ &\frac{1}{p \ n} \sum_{(i,k)=(1,1)}^{(p,n)} \left(\hat{\Lambda}_{ik}\right)^2 \ . \end{aligned}$$

Therefore we have

$$\mathrm{thresL} \ = \ \frac{\sqrt{\mathrm{vlz}}}{\mathrm{thresZ}} \ .$$

Note, the average contribution $\sqrt{\text{vlz}}$ includes elements close to zero that do not belong to any bicluster. Therefore $\sqrt{\text{vlz}}$ underestimates the contribution of a bicluster because the non-bicluster elements should not count.

On the other hand both Λ_{ik} and z_{ij} are assumed to stem from sparse distributions which favour large values which might dominate the second moment. In this case $\sqrt{\text{vlz}}$ overestimates the contribution of a bicluster because large values dominate.

In summary, above estimate for `thresL` is a trade-off between underestimate due to sparseness and overestimate due to large values.

A Methods and Functions

A.1 extract_plot

Extraction of Biclusters and Plotting of the Results.

1. Usage: `extract_plot(X,L,Z,thresZ=0.5,ti,thresL=NULL,Y=NULL,x11b=TRUE)`
2. Arguments:
    - X: original data matrix.
    - L: loading, left matrix.
    - Z: factor, right matrix.
    - thresZ: threshold for sample belonging to bicluster (default 0.5).
    - thresL: threshold for loading belonging to bicluster (estimated if not given).
    - ti: plot title.
    - Y: alternative: noise free data matrix.
    - x11b: screen output or not.
3. Return Values:
    - bic: extracted biclusters.
    - numn: indexes for the extracted biclusters.
    - biclust: clusters of kmeans clustering.
    - pmZ: permutation matrix of z from kmeans clustering.
    - pmL: permutation matrix of L from kmeans clustering.
    - nL: normalized loadings (left matrix).
    - nZ normalized factors (right matrix).
    - Xord: sorted original matrix according to kmeans on Z and kmeans on L.

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

Essentially the model is the sum of outer products of vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

The hidden dimension p is used for kmeans clustering of $\boldsymbol{\Lambda}_i$ and $\boldsymbol{Z}_i$.

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.

The method produces a couple of plots given below.

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

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

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

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

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

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

Implementation in `R` .

EXAMPLE:

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X



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

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

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

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

A.2 extract_bic

Extraction of Biclusters.

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

Essentially the model is the sum of outer products of vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ (\boldsymbol{Z}_i)^T \ + \ \boldsymbol{\Upsilon}$$

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

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

Using these values we can computer for each j the variance $\boldsymbol{Z}_i$ given $\boldsymbol{x}_j$. Here

$$\boldsymbol{x}_j \ = \ \boldsymbol{\Lambda} \ \boldsymbol{z}_j \ + \ \boldsymbol{\epsilon}_j$$

This variance can be used to determine the information content of a bicluster.

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

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

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

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

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

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

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

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

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

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

Implementation in `R` .

EXAMPLE:

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

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

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

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


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

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

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


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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

rToy$avini  #$

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

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

```
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

rBreast$avini    #$

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

A.3 fabi

Factor Analysis for Bicluster Acquisition: Laplace Prior (FABI).
`R` implementation of `fabia`, therefore it is *slow*.

1. Usage: `fabi(X,cyc,alpha,spl,spz,p)`
2. Arguments:
    - X: the data matrix.
    - cyc: number of cycles to run.
    - alpha: sparseness loadings (0.1 - 1.0).
    - spl: sparseness prior loadings (0.5 - 4.0).
    - spz: sparseness factors (0.5 - 4.0).
    - p: number of hidden factor = number of biclusters.
3. Return Values:
    - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
    - L: Loadings: $\boldsymbol{\Lambda}$

- Z: Factors: $\boldsymbol{Z}$
- Psi: Noise variance: $\boldsymbol{\Psi}$
- lapla: Variational parameter

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \; = \; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \; (\boldsymbol{Z}_i)^T \; + \; \boldsymbol{\Upsilon}$$

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

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

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

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 is Independent Laplace:

$$p(\boldsymbol{\Lambda}_i) \; = \; \left(\frac{1}{\sqrt{2}} \right)^n \prod_{j=1}^{n} e^{- \; \sqrt{2} \; |\Lambda_{ij}|}$$

Noise: Gaussian independent

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

Data Mean:

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

Therefore the data should be normalized to *zero mean.*

Data Covariance:

$\mathrm{E}(\boldsymbol{x}\,\boldsymbol{x}^T) \;=\; \boldsymbol{\Lambda}\mathrm{E}(\boldsymbol{z}\boldsymbol{z}^T)\boldsymbol{\Lambda}^T \;+\; \boldsymbol{\Lambda}\mathrm{E}(\boldsymbol{z})\mathrm{E}(\boldsymbol{\epsilon}^T) \;+\; \mathrm{E}(\boldsymbol{z})\mathrm{E}(\boldsymbol{\epsilon})\boldsymbol{\Lambda}^T \;+\; \mathrm{E}(\boldsymbol{\epsilon}\,\boldsymbol{\epsilon}^T) \;=\; \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \;+\; \mathrm{diag}(\sigma_k^2)$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\Lambda^k\right)^T \Lambda^k \;=\; 1$$

Here the length of Λ^k is p. We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

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

Estimated Latent Variables: $\boldsymbol{Z}$

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

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

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

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

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

EXAMPLE:

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(Multi_A)

X <- as.matrix(XMulti)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(DLBCL_B)

X <- as.matrix(XDLBCL)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

A.4 fabia

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

C implementation of `fabia`.

1. Usage: `fabia(X,cyc,alpha,spl,spz,p)`
2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings (0.1 - 1.0).
   - spl: sparseness prior loadings (0.5 - 4.0).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
3. Return values:
   - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
   - L: Loadings: $\boldsymbol{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\boldsymbol{\Psi}$
   - lapla: Variational parameter

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

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

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

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

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 is Independent Laplace:

$$p(\boldsymbol{\Lambda}_i) \;=\; \left(\frac{1}{\sqrt{2}}\right)^{n} \prod_{j=1}^{n} e^{-\;\sqrt{2}\;|\Lambda_{ij}|}$$

Noise: Gaussian independent

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

Data Mean:

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

Therefore the data should be normalized to *zero mean*.
Data Covariance:

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

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\Lambda^k\right)^T \Lambda^k \;=\; 1$$

Here the length of Λ^k is p. We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

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

Estimated Latent Variables: $\boldsymbol{Z}$

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

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

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

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

The code is implemented in C using the Rcpp package.

EXAMPLE:

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X



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


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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

```
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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

data(Multi_A)

X <- as.matrix(XMulti)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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

data(DLBCL_B)
```

```
X <- as.matrix(XDLBCL)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

A.5 fabiaVersion

Display version info for package and for FABIA.

1. Usage: `fabiaVersion()`

EXAMPLE:

```
fabiaVersion()
```

A.6 fabiap

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

1. Usage: `fabiap(X,cyc,alpha,spl,spz,p,sL,sZ)`
2. Arguments:
    - X: the data matrix.
    - cyc: number of cycles to run.
    - alpha: sparseness loadings (0.1 - 1.0).
    - spl: sparseness prior loadings (0.5 - 4.0).
    - spz: sparseness factors (0.5 - 4.0).
    - p: number of hidden factor = number of biclusters.
    - sL: final sparseness loadings.

   - sZ: final sparseness factors.

3. Return Values:

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

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

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

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

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

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 is Independent Laplace:

$$p(\boldsymbol{\Lambda}_i) \ = \ \left(\frac{1}{\sqrt{2}}\right)^n \prod_{j=1}^{n} e^{-\ \sqrt{2}\ |\Lambda_{ij}|}$$

Noise: Gaussian independent

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

Data Mean:

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

Therefore the data should be normalized to *zero mean.*

Data Covariance:

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

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \;+\; \left(\Lambda^k\right)^T \Lambda^k \;=\; 1$$

Here the length of Λ^k is 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: $\boldsymbol{\Lambda}_i\;\left(\boldsymbol{Z}_i\right)^T$ Larges values give the bicluster (ideal the nonzero values).

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

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

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

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

EXAMPLE:

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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


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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

data(Breast_A)

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

```
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(Multi_A)

X <- as.matrix(XMulti)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(DLBCL_B)

X <- as.matrix(XDLBCL)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
```

```
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

A.7 fabias

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

C implementation of `fabias`.

1. Usage: `fabias(X,cyc,alpha,spz,p)`
2. Arguments:
   - X: the data matrix.
   - cyc: number of cycles to run.
   - alpha: sparseness loadings via projection (0.1 - 0.9).
   - spz: sparseness factors (0.5 - 4.0).
   - p: number of hidden factor = number of biclusters.
3. Return Values:
   - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda} \, \boldsymbol{Z}$
   - L: Loadings: $\boldsymbol{\Lambda}$
   - Z: Factors: $\boldsymbol{Z}$
   - Psi: Noise variance: $\boldsymbol{\Psi}$
   - lapla: Variational parameter

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

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

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

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

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

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

Data Mean:

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

Therefore the data should be normalized to *zero mean.*

Data Covariance:

$$\mathrm{E}(\boldsymbol{x} \ \boldsymbol{x}^T) \ = \ \boldsymbol{\Lambda} \mathrm{E}(\boldsymbol{z}\boldsymbol{z}^T) \boldsymbol{\Lambda}^T \ + \ \boldsymbol{\Lambda} \mathrm{E}(\boldsymbol{z}) \mathrm{E}(\boldsymbol{\epsilon}^T) \ + \ \mathrm{E}(\boldsymbol{z}) \mathrm{E}(\boldsymbol{\epsilon}) \boldsymbol{\Lambda}^T \ + \ \mathrm{E}(\boldsymbol{\epsilon} \ \boldsymbol{\epsilon}^T) \ = \ \boldsymbol{\Lambda}\boldsymbol{\Lambda}^T \ + \ \mathrm{diag}(\sigma_k^2)$$

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \ + \ \left(\Lambda^k\right)^T \Lambda^k \ = \ 1$$

Here the length of Λ^k is p. We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

Estimated Parameters: $\boldsymbol{\Lambda}$ and σ_k
Estimated Latent Variables: $\boldsymbol{Z}$
Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
Estimated Biclusters: $\boldsymbol{\Lambda}_i\ \ (\boldsymbol{Z}_i)^T$ Larges values give the bicluster (ideal the nonzero values).

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

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

$$\text{sparseness}(\boldsymbol{\Lambda}_i) \ = \ \frac{\sqrt{n} \ - \ \sum_{j=1}^{n} |\Lambda_{ij}| \ / \ \sum_{j=1}^{n} \Lambda_{ij}^2}{\sqrt{n} \ - \ 1}$$

The code is implemented in C using the Rcpp package.

EXAMPLE:

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

```
#---------------
# DEMO2
#---------------


data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(Multi_A)

X <- as.matrix(XMulti)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

#---------------
# DEMO4
```

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


data(DLBCL_B)

X <- as.matrix(XDLBCL)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

A.8 fabiasp

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

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

1. Usage: `fabiasp(X,cyc,alpha,spz,p)`
2. Arguments:
    - X: the data matrix.
    - cyc: number of cycles to run.
    - alpha: sparseness loadings via projection (0.1 - 0.9).
    - spz: sparseness factors (0.5 - 4.0).
    - p: number of hidden factor = number of biclusters.
3. Return Values:
    - LZ: Estimated Noise Free Data: $\boldsymbol{\Lambda}\ \boldsymbol{Z}$
    - L: Loadings: $\boldsymbol{\Lambda}$
    - Z: Factors: $\boldsymbol{Z}$
    - Psi: Noise variance: $\boldsymbol{\Psi}$

- lapla: Variational parameter

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \; = \; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \; \left(\boldsymbol{Z}_i\right)^T \; + \; \boldsymbol{\Upsilon}$$

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

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

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

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

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

Data Mean:

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

Therefore the data should be normalized to *zero mean.*

Data Covariance:

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

Normalizing the data to variance one for each component gives

$$\sigma_k^2 \; + \; \left(\Lambda^k\right)^T \Lambda^k \; = \; 1$$

Here the length of Λ^k is p. We recommend to *normalize the components to variance one* in order to make the signal and noise comparable across components.

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

Estimated Latent Variables: $\boldsymbol{Z}$

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

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

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

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

$$\text{sparseness}(\boldsymbol{\Lambda}_i) \;\; = \;\; \frac{\sqrt{n} \; - \; \sum_{j=1}^{n} |\Lambda_{ij}| \; / \; \sum_{j=1}^{n} \Lambda_{ij}^2}{\sqrt{n} \; - \; 1}$$

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

EXAMPLE:

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


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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X
```

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


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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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


data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(Multi_A)

X <- as.matrix(XMulti)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

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


data(DLBCL_B)

X <- as.matrix(XDLBCL)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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

A.9 make_fabi_data

Generation of bicluster data.

1. Usage: `make_fabi_data(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise, mean_z,sd_z,sd_l_noi`
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 beloning to ith bicluster.
   - LC: list where ith element gives observations beloning to ith bicluster.

Essentially the data generation model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

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

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \; (\boldsymbol{Z}_i)^T \;+\; \boldsymbol{\Upsilon}$$

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

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

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

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

Implementation in `R` .

EXAMPLE:

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

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

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

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

A.10 make_fabi_data_pos

Generation of bicluster data.

1. Usage: `make_fabi_data_pos(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise, mean_z,sd_z,sd_l`
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 beloning to ith bicluster.
   - LC: list where ith element gives observations beloning to ith bicluster.

Essentially the data generation model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

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

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \;\left(\boldsymbol{Z}_i\right)^T \;+\; \boldsymbol{\Upsilon}$$

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

Sequentially $\mathbf{\Lambda}_i$ 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.

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

Implementation in `R` .

EXAMPLE:

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

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

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

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

A.11 make_fabi_data_blocks

Generation of bicluster data with bicluster blocks.

1. Usage: `make_fabi_data_blocks(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise, mean_z,sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
   - n: number of observations.

- l: number of samples.
- p: number of biclusters.
- f1: $l/f1$ max. additional samples are active in a bicluster.
- f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
- of1: minimal active samples in a bicluster.
- of2: minimal observations that form a pattern in a bicluster.
- sd_noise: Gaussian zero mean noise std on data matrix.
- sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
- mean_z: Gaussian mean for activated factors.
- sd_z: Gaussian std for activated factors.
- sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
- mean_l: Gaussian mean for observation patterns.
- sd_l: Gaussian std for observation patterns.

3. Return Values:

   - X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.
   - Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
   - ZC: list where ith element gives samples beloning to ith bicluster.
   - LC: list where ith element gives observations beloning 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 data generation model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

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

$$\boldsymbol{X} \;=\; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \;\left(\boldsymbol{Z}_i\right)^T \;+\; \boldsymbol{\Upsilon}$$

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

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

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

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

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

Implementation in `R` .

EXAMPLE:

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

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

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

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

A.12 make_fabi_data_blocks_pos

Generation of bicluster data with bicluster blocks.

1. Usage: `make_fabi_data_blocks_pos(n,l,p,f1,f2,of1,of2,sd_noise,sd_z_noise, mean_z,sd_z,sd_l_noise,mean_l,sd_l)`
2. Arguments:
    - n: number of observations.

- l: number of samples.
- p: number of biclusters.
- f1: $l/f1$ max. additional samples are active in a bicluster.
- f2: $n/f2$ max. additional observations that form a pattern in a bicluster.
- of1: minimal active samples in a bicluster.
- of2: minimal observations that form a pattern in a bicluster.
- sd_noise: Gaussian zero mean noise std on data matrix.
- sd_z_noise: Gaussian zero mean noise std for deactivated hidden factors.
- mean_z: Gaussian mean for activated factors.
- sd_z: Gaussian std for activated factors.
- sd_l_noise: Gaussian zero mean noise std if no observation patterns are present.
- mean_l: Gaussian mean for observation patterns.
- sd_l: Gaussian std for observation patterns.

3. Return Values:
   - X: the noisy data $\boldsymbol{X}$ from $\mathbb{R}^{n \times l}$.
   - Y: the noise free data $\boldsymbol{Y}$ from $\mathbb{R}^{n \times l}$.
   - ZC: list where ith element gives samples beloning to ith bicluster.
   - LC: list where ith element gives observations beloning 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 data generation model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{Y} \ = \ \boldsymbol{\Lambda} \ \boldsymbol{Z}$$

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T \ + \ \boldsymbol{\Upsilon}$$

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

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

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

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

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

Implementation in `R` .

EXAMPLE:

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

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

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

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

A.13 mfsc

Sparse Matrix Factorization for bicluster analysis (MFSC).

1. Usage: `mfsc(X,p,sL,sZ,cyc=100)`
2. Arguments:
    - X: the data matrix.
    - p: number of hidden factor = number of biclusters.

- sL: sparseness loadings.
- sZ: sparseness factors.
- cyc: maximal number of iterations.

3. Return Values:

   - L: Left matrix: $\mathbf{\Lambda}$
   - Z: Right matrix: $\boldsymbol{Z}$

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

Essentially the model is the sum of outer products of sparse vectors. The number of summands p is the number of biclusters.

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \mathbf{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T$$

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

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

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

Estimated Biclusters: $\mathbf{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T$ Larges values give the bicluster (ideal the nonzero values).

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

$$\min_{\mathbf{\Lambda}, \boldsymbol{Z}} \left\| \boldsymbol{X} \ - \ \mathbf{\Lambda} \ \boldsymbol{Z} \right\|_F^2$$

$$\text{subject to} \ \ \left\| \mathbf{\Lambda} \right\|_F^2 \ = \ 1$$

$$\text{subject to} \ \ \left\| \mathbf{\Lambda} \right\|_1 \ = \ k_L$$

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

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

Model selection is done by gradient descent on the Euclidean objective and thereafter projection of single vectors of $\mathbf{\Lambda}$ 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:

$$\text{sparseness}(\mathbf{\Lambda}_i) \ = \ \frac{\sqrt{n} \ - \ \sum_{j=1}^{n} |\Lambda_{ij}| \ / \ \sum_{j=1}^{n} \Lambda_{ij}^2}{\sqrt{n} \ - \ 1}$$

The code is implemented in `R` .

EXAMPLE:

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

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

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


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


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

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

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

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


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

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

A.14 myImagePlot

Plotting of a matrix.

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

Plotting a table of numbers as an image using `R` .
The color scale is based on the highest and lowest values in the matrix.
Program has been obtained by `http://www.phaget4.org/R/myImagePlot.R`

EXAMPLE:

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

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

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

```
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

myImagePlot(X)
```

A.15 PlotBicluster

Plots a bicluster.

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

Plots a bicluster.

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

The bicluster is marked by a rectangle.

Plot2: Only the bicluster is presented.

Implementation in `R` .

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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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

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

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


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

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

X <- dat[[1]]
Y <- dat[[2]]
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X


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



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

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

```
#---------------
# DEMO2
#---------------

data(Breast_A)

X <- as.matrix(XBreast)
X <- X- rowMeans(X)
XX <- (1/ncol(X))*tcrossprod(X)
dXX <- 1/sqrt(diag(XX)+0.001*as.vector(rep(1,nrow(X))))
X <- dXX*X

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

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

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

A.16 nmfdiv

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

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

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

$$\boldsymbol{X} \ = \ \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \ \left(\boldsymbol{Z}_i\right)^T$$

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

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

objective:

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

update:

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

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

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

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


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


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

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

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

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


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

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

A.17 nmfeu

Non-negative Matrix Factorization with Euclidean distance as objective.

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

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

$$\boldsymbol{X} \; = \; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \; (\boldsymbol{Z}_i)^T$$

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

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

objective:

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

update:

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

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

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

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

The code is implemented in `R` .

EXAMPLE:

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

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

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


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


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

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

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

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


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

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

A.18 nmfsc

Non-negative Sparse Matrix Factorization with sparseness constraints.

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

Essentially the model is the sum of outer products of sparse vectors.

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

$$\boldsymbol{X} \; = \; \sum_{i=1}^{p} \boldsymbol{\Lambda}_i \; \left(\boldsymbol{Z}_i\right)^T$$

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: $\boldsymbol{\Lambda}_i \; \left(\boldsymbol{Z}_i\right)^T$ Larges values give the bicluster (ideal the nonzero values).

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

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

$$\begin{aligned}
\text{subject to} \;\; & \left\| \boldsymbol{\Lambda} \right\|_F^2 \; = \; 1 \\
\text{subject to} \;\; & \left\| \boldsymbol{\Lambda} \right\|_1 \; = \; k_L \\
\text{subject to} \;\; & \boldsymbol{\Lambda} \; \geq \; \boldsymbol{0} \\
\text{subject to} \;\; & \left\| \boldsymbol{Z} \right\|_F^2 \; = \; 1 \\
\text{subject to} \;\; & \left\| \boldsymbol{Z} \right\|_1 \; = \; k_Z \\
\text{subject to} \;\; & \boldsymbol{Z} \; \geq \; \boldsymbol{0}
\end{aligned}$$

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:

$$\text{sparseness}(\boldsymbol{\Lambda}_i) \; = \; \frac{\sqrt{n} \; - \; \sum_{j=1}^{n} \left|\Lambda_{ij}\right| \; / \; \sum_{j=1}^{n} \Lambda_{ij}^2}{\sqrt{n} \; - \; 1}$$

The code is implemented in `R` .

EXAMPLE:

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

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

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


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


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

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

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


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

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

A.19 nprojfunc

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

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

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

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

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

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

The code is implemented in `R` .

EXAMPLE:

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

size <- 30
sparseness <- 0.7

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

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

s
ss
```

A.20 projfunc

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

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

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

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

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

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
```

B Data Sets

B.1 Breast_A

Microarray data set of van't Veer breast cancer.

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

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

Here we want to find subclasses in the data set.

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

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

B.2 DLBCL_B

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

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

Goal was to predict the survival after chemotherapy

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

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

We want to identify these subclasses.

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

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

B.3 Multi_A

Microarray data set of Su on different mammalian tissue types.

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

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

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

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

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

References

D.-A. Clevert and S. Hochreiter. FARMS: a probabilistic latent variable model for summarizing Affymetrix array data at probe level. 15th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB), 2007.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society B*, 39(1):1–22, 1977.

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

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

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

P. O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

K. G. Jöreskog. Some contributions to maximum likelihood factor analysis. *Psychometrika*, 32:443–482, 1967.

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

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

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

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

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

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

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

W. Talloen, D.-A. Clevert, S. Hochreiter, D. Amaratunga, L. Bijnens, S. Kass, and H. W. H. Göhlmann. I/NI-calls for the exclusion of non-informative genes: a highly effective feature filtering tool for microarray data. *Bioinformatics*, 21(23):2897–2902, 2007. doi:10.1093/bioinformatics/btm478.

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