Package 'fabMix'

November 14, 2018

Type Package

Title Overfitting Bayesian Mixtures of Factor Analyzers with Parsimonious Covariance and Unknown Number of Components

Version 4.4

Date 2018-11-13

Maintainer Panagiotis Papastamoulis <papapast@yahoo.gr>

Description

Model-based clustering of multivariate continuous data using Bayesian mixtures of factor analyzers (Papastamoulis (2018) <DOI:10.1016/j.csda.2018.03.007>). The number of clusters is estimated using overfitting mixture models (Rousseau and Mengersen (2011) <DOI:10.1111/j.1467-9868.2011.00781.x>): suitable prior assumptions ensure that asymptotically the extra components will have zero posterior weight, therefore, the inference is based on the ``alive'' components. A Gibbs sampler is implemented in order to (approximately) sample from the posterior distribution of the overfitting mixture. A prior parallel tempering scheme is also available, which allows to run multiple parallel chains with different prior distributions on the mixture weights. These chains run in parallel and can swap states using a Metropolis-Hastings move. Eight different parameterizations give rise to parsimonious representations of the covariance per cluster (following Mc Nicholas and Murphy (2008) <DOI:10.1007/s11222-008-9056-0>). The model parameterization and number of factors is selected according to the Bayesian Information Criterion. Identifiability issues related to label switching are dealt by post-processing the simulated output with the Equivalence Classes Representatives algorithm (Papastamoulis and Iliopoulos (2010) <https://www.jstor.org/stable/25703571>, Papastamoulis (2016) <DOI:10.18637/jss.v069.c01>).

License GPL-2

URL https://github.com/mqbssppe/overfittingFABMix

Imports Rcpp (>= 0.12.17), MASS, doParallel, foreach, label.switching, mvtnorm, doRNG, RColorBrewer, corrplot, mclust, coda

LinkingTo Rcpp, RcppArmadillo

NeedsCompilation yes

Author Panagiotis Papastamoulis [aut, cre] (0000-0001-9468-7613)

R topics documented:

`fabMix-package` *Overfitting Bayesian Mixtures of Factor Analyzers with Parsimonious Covariance and Unknown Number of Components*

Description

Model-based clustering of multivariate continuous data using Bayesian mixtures of factor analyzers (Papastamoulis (2018) <DOI:10.1016/j.csda.2018.03.007>). The number of clusters is estimated using overfitting mixture models (Rousseau and Mengersen (2011) <DOI:10.1111/j.1467-9868.2011.00781.x>): suitable prior assumptions ensure that asymptotically the extra components will have zero posterior weight, therefore, the inference is based on the "alive" components. A Gibbs sampler is implemented in order to (approximately) sample from the posterior distribution of the overfitting mixture. A prior parallel tempering scheme is also available, which allows to run multiple parallel chains with different prior distributions on the mixture weights. These chains run in parallel and can swap states using a Metropolis-Hastings move. Eight different parameterizations give rise to parsimonious representations of the covariance per cluster (following Mc Nicholas and Murphy (2008) <DOI:10.1007/s11222-008-9056-0>). The model parameterization and number of factors is selected according to the Bayesian Information Criterion. Identifiability issues related to label switching are dealt by post-processing the simulated output with the Equivalence Classes Representatives algorithm (Papastamoulis and Iliopoulos (2010) <https://www.jstor.org/stable/25703571>, Papastamoulis (2016) <DOI:10.18637/jss.v069.c01>).

The main fuction of the package is `fabMix`.

Author(s)

Panagiotis Papastamoulis

Maintainer: Panagiotis Papastamoulis <papapast@yahoo.gr>

References

Fokoue, E. and Titterington, D.M. (2003). Mixtures of Factor Analysers: Bayesian Estimation and Inference by Stochastic Simulation. Machine Learing, 50(1): 73-94.

McNicholas, P.D. and Murphy, T.B. Statistics and Computing (2008) 18: 285. https://doi.org/10.1007/s11222-008-9056-0.

Papastamoulis P. and Iliopoulos G. (2010). An artificial allocations based solution to the label switching problem in Bayesian analysis of mixtures of distributions. Journal of Computational and Graphical Statistics, 19: 313-331.

Rousseau, J. and Mengersen, K. (2011). Asymptotic behaviour of the posterior distribution in overfitted mixture models. Journal of the Royal Statistical Society, Series B (methodological), 73(5): 689-710.

van Havre, Z., White, N., Rousseau, J. and Mengersen, K. (2015). Overfitting Bayesian Mixture Models with an Unknown Number of Components. PLOS ONE, 10(7): 1-27.

Papastamoulis, P. (2016). `label.switching`: An R Package for Dealing with the Label Switching Problem in MCMC Outputs. Journal of Statistical Software, 69(1), 1-24.

Papastamoulis, P. (2018). Overfitting Bayesian mixtures of factor analyzers with an unknown number of components. Computational Statistics and Data Analysis, 124: 220-234. DOI: 10.1016/j.csda.2018.03.007.

See Also

`fabMix`, `plot.fabMix.object`

Examples

```
# TOY EXAMPLE (very small numbers... only for CRAN check purposes)

#####################################################################
# (a) using 2 cores in parallel, each one running 2 heated chains.
#####################################################################
library('fabMix')

n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2      # true number of clusters

sINV_diag = 1/((1:p))  # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)

# Run `fabMix` for a _small_ number of iterations for the
# `UUU` (maximal model) and `CCC` (minimal model) parameterizations,
#  using the default prior parallel heating parameters `dirPriorAlphas`.
# NOTE: `dirPriorAlphas` may require some tuning in general.


qRange <- 2 # values for the number of factors (only the true number
```

```
#                                                    is considered here)
Kmax <- 4 # number of components for the overfitted mixture model
nChains <- 2 # number of parallel heated chains

set.seed(1)
fm <- fabMix( model = c("UUU", "CCC"), nChains = nChains,
rawData = syntheticDataset$data, outDir = "toyExample",
        Kmax = Kmax, mCycles = 4, burnCycles = 1, q = qRange,
        g = 0.5, h = 0.5, alpha_sigma = 0.5, beta_sigma = 0.5,
        warm_up_overfitting = 2, warm_up = 5)

# WARNING: the following parameters:
#  Kmax, nChains, mCycles, burnCycles, warm_up_overfitting, warm_up
#  should take (much) _larger_ values. E.g. a typical implementation consists of:
#        Kmax = 20, nChains >= 3, mCycles = 1100, burnCycles = 100,
#        warm_up_overfitting = 500, warm_up = 5000.

# Now print a run summary and produce some plots.
print(fm)
plot(fm, what = "BIC")

######################################################################
# (b) using 12 cores_________________________________________________
#_______4 models with 3 heated chains running in parallel________
#_______considering all 8 model parameterizations_________________
######################################################################
## Not run:
library('fabMix')
set.seed(99)
n = 100                # sample size
p = 30                # number of variables
q = 2                # number of factors
K = 5      # number of clusters
sINV_diag = rep(1/100,p)  # diagonal of inverse variance of errors
syntheticDataset <- simData(sameLambda=FALSE,K.true = K, n = n, q = q, p = p,
sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
qRange <- 1:3 # range of values for the number of factors
Kmax <- 20 # number of components for the overfitted mixture model
nChains <- 3 # number of parallel heated chains

# the next command takes ~ 1 hour in a Linux workstation with 12 threads.
fm <- fabMix( parallelModels = 4,
nChains = nChains,
model = c("UUU","CUU","UCU","CCU","UCC","UUC","CUC","CCC"),
rawData = syntheticDataset$data, outDir = "toyExample_b",
        Kmax = Kmax, mCycles = 600, burnCycles = 100, q = qRange,
        g = 0.5, h = 0.5, alpha_sigma = 0.5, beta_sigma = 0.5,
        warm_up_overfitting = 500, warm_up = 5000)
print(fm)
plot(fm, what = "BIC")
plot(fm, what = "classification_pairs")
```

```
## End(Not run)
```

`complete.log.likelihood` *Complete log-likelihood function for xCx models.*

Description

Complete log-likelihood function for models with same error variance per component (xCx).

Usage

```
complete.log.likelihood(x_data, w, mu, Lambda, SigmaINV, z)
```

Arguments

`x_data`	$n \times p$ matrix containing the data
`w`	a vector of length K containing the mixture weights
`mu`	$K \times p$ matrix containing the marginal means per component
`Lambda`	$K \times p \times q$ array of factor loadings per component
`SigmaINV`	$p \times p$ precision matrix (inverse covariance)
`z`	A vector of length n containing the allocations of the n datapoints to the K mixture components

Value

complete log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
```

```
Lambda <- array( runif(K*p*q), dim = c(K,p,q) )
SigmaINV <- array(1, dim = c(p,p))
# compute the complete.log.likelihood
complete.log.likelihood(x_data = x_data, w = w, mu = mu,
Lambda = Lambda, SigmaINV = SigmaINV, z = z)
```

`complete.log.likelihood_q0` *Complete log-likelihood function for xUx models and* $q = 0$

Description

Complete log-likelihood function for models with different error variance per component (xUx) and diagonal covariance structure per component ($q = 0$.

Usage

```
complete.log.likelihood_q0(x_data, w, mu, SigmaINV, z)
```

Arguments

`x_data`	$n \times p$ matrix containing the data
`w`	a vector of length K containing the mixture weights
`mu`	$K \times p$ matrix containing the marginal means per component
`SigmaINV`	$K \times p \times p$ precision matrix (inverse covariance) per component
`z`	A vector of length n containing the allocations of the n datapoints to the K mixture components

Value

complete log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(waveDataset1500[ 1:20, -1]) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
w <- rep(1/K, K)
```

```
mu <- array( -0.1 + 0.2*runif(K * p), dim = c(K,p) )
SigmaINV <- array( 1, dim = c(K,p,p))
# compute the complete.log.likelihood ( -inf )
complete.log.likelihood_q0(x_data = x_data, w = w, mu = mu,
SigmaINV = SigmaINV, z = z)
```

`complete.log.likelihood_q0_sameSigma`

Complete log-likelihood function for xCx models and $q = 0$

Description

Complete log-likelihood function for models with same error variance per component (xCx) and diagonal covariance structure per component ($q = 0$.

Usage

```
complete.log.likelihood_q0_sameSigma(x_data, w, mu, SigmaINV, z)
```

Arguments

`x_data`	$n \times p$ matrix containing the data
`w`	a vector of length K containing the mixture weights
`mu`	$K \times p$ matrix containing the marginal means per component
`SigmaINV`	$p \times p$ precision matrix (inverse covariance)
`z`	A vector of length n containing the allocations of the n datapoints to the K mixture components

Value

complete log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(waveDataset1500[ 1:20, -1]) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
w <- rep(1/K, K)
```

```
mu <- array( -0.1 + 0.2*runif(K * p), dim = c(K,p) )
SigmaINV <- array( 1, dim = c(p,p))
# compute the complete.log.likelihood ( -inf )
complete.log.likelihood_q0_sameSigma(x_data = x_data, w = w, mu = mu,
SigmaINV = SigmaINV, z = z)
```

`complete.log.likelihood_Sj` *Complete log-likelihood function for xUx models.*

Description

Complete log-likelihood function for models with different error variance per component (xUx).

Usage

```
complete.log.likelihood_Sj(x_data, w, mu, Lambda, SigmaINV, z)
```

Arguments

`x_data`	$n \times p$ matrix containing the data
`w`	a vector of length K containing the mixture weights
`mu`	$K \times p$ matrix containing the marginal means per component
`Lambda`	$K \times p \times q$ array of factor loadings per component (maybe restricted to be the same)
`SigmaINV`	$K \times p \times p$ precision matrix (inverse covariance) per component
`z`	

Value

complete log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
```

```
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
Lambda <- array( runif(K*p*q), dim = c(K,p,q) )
SigmaINV <- array( c(0.5, 0.75, 1), dim = c(K,p,p))
# compute the complete.log.likelihood
complete.log.likelihood_Sj(x_data = x_data, w = w, mu = mu,
Lambda = Lambda, SigmaINV = SigmaINV, z = z)
```

`compute_A_B_G_D_and_simulate_mu_Lambda`
Computation and simulations

Description

This function simulates μ and Λ.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda(SigmaINV,
suff_statistics, OmegaINV, K, priorConst1, T_INV, v_r)
```

Arguments

`SigmaINV`	Precision matrix Σ^{-1}
`suff_statistics`	Sufficient statistics
`OmegaINV`	Prior parameter: Ω^{-1}
`K`	Number of overfitting mixture components
`priorConst1`	Prior constant: $T^{-1}\xi$
`T_INV`	Prior parameter: $T^{-1}\xi$
`v_r`	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ and Λ:

`Lambdas`	$K \times p \times q$ array (factor loadings per component)
`mu`	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(as.matrix(waveDataset1500[ 1:20, -1])) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
n <- dim(x_data)[1]
q <- 2
K <- length(table(z))           # 3 classes

T_INV <- array(data = 0, dim = c(p,p))
diag(T_INV) <- diag(var(x_data))
diag(T_INV) <- 1/diag(T_INV)
ksi <- colMeans(x_data)
priorConst1 <- array(diag(T_INV)*ksi, dim =c(p,1))
# give some arbitrary values to the parameters:
set.seed(1)
mu <- array( runif(K * p), dim = c(K,p) )
y <- array(rnorm(n = q*n), dim = c(n,q))
SigmaINV <- array(data = 0, dim = c(p,p) )
diag(SigmaINV) <- 0.5 + 0.5*runif(p)
OmegaINV <- diag(q)
# compute sufficient stats
suf_stat <- compute_sufficient_statistics(y = y,
 z = z, K = K, x_data = x_data)

v_r <- numeric(p) #indicates the non-zero values of Lambdas
for( r in 1:p ){
v_r[r] <- min(r,q)
}
# now simulate mu and Lambda
f2 <- compute_A_B_G_D_and_simulate_mu_Lambda(SigmaINV = SigmaINV,
                suff_statistics = suf_stat, OmegaINV = OmegaINV,
                K = K, priorConst1 = priorConst1, T_INV = T_INV, v_r = v_r)
# f2$mu contains the simulated means
# f2$Lambdas contains the simulated factor loadings
```

---

compute_A_B_G_D_and_simulate_mu_Lambda_CCU

Computation and simulations for CCU

Description

This function simulates μ and Λ for the CCU model.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda_CCU(SigmaINV,
suff_statistics, OmegaINV, K, priorConst1, T_INV, v_r)
```

Arguments

SigmaINV	Precision matrix Σ^{-1}
suff_statistics	Sufficient statistics
OmegaINV	Prior parameter: Ω^{-1}
K	Number of overfitting mixture components
priorConst1	Prior constant: $T^{-1}\xi$
T_INV	Prior parameter: $T^{-1}\xi$
v_r	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ and Λ:

Lambdas	$K \times p \times q$ array (factor loadings per component)
mu	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(as.matrix(waveDataset1500[ 1:20, -1])) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
n <- dim(x_data)[1]
q <- 2
K <- length(table(z))           # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
mu <- array( runif(K * p), dim = c(K,p) )
y <- array(rnorm(n = q*n), dim = c(n,q))
SigmaINV <- array(data = 0, dim = c(p,p) )
diag(SigmaINV) = 0.5 + 0.5*runif(p)
OmegaINV <- diag(q)
# compute sufficient stats
suf_stat <- compute_sufficient_statistics_given_mu(y = y,
 z = z, K = K, x_data = x_data, mu = mu)

v_r <- numeric(p) #indicates the non-zero values of Lambdas
for( r in 1:p ){
v_r[r] <- min(r,q)
}
T_INV <- array(data = 0, dim = c(p,p))
diag(T_INV) <- diag(var(x_data))
```

```
diag(T_INV) <- 1/diag(T_INV)
ksi <- colMeans(x_data)
priorConst1 <- array(diag(T_INV)*ksi, dim =c(p,1))
# now simulate mu and Lambda
f2 <- compute_A_B_G_D_and_simulate_mu_Lambda_CCU(SigmaINV = SigmaINV,
                suff_statistics = suf_stat, OmegaINV = OmegaINV,
                K = K, priorConst1 = priorConst1, T_INV = T_INV, v_r = v_r)
# f2$mu contains the simulated means
# f2$Lambdas contains the simulated factor loadings
```

`compute_A_B_G_D_and_simulate_mu_Lambda_CUU`
Computation and simulations for CUU

Description

This function simulates μ and Λ for the CUU model.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda_CUU(SigmaINV,
suff_statistics, OmegaINV, K, priorConst1, T_INV, v_r)
```

Arguments

`SigmaINV`	Precision matrix Σ^{-1}
`suff_statistics`	Sufficient statistics
`OmegaINV`	Prior parameter: Ω^{-1}
`K`	Number of overfitting mixture components
`priorConst1`	Prior constant: $T^{-1}\xi$
`T_INV`	Prior parameter: $T^{-1}\xi$
`v_r`	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ and Λ:

`Lambdas`	$K \times p \times q$ array (factor loadings per component)
`mu`	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(as.matrix(waveDataset1500[ 1:20, -1])) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
n <- dim(x_data)[1]
q <- 2
K <- length(table(z))           # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
mu <- array( runif(K * p), dim = c(K,p) )
y <- array(rnorm(n = q*n), dim = c(n,q))
SigmaINV <- array(data = 0, dim = c(K,p,p) )
for(k in 1:K){
diag(SigmaINV[k,,]) <- 0.5 + 0.5*runif(p)
}
OmegaINV <- diag(q)
# compute sufficient stats
suf_stat <- compute_sufficient_statistics_given_mu(y = y,
 z = z, K = K, x_data = x_data, mu = mu)

v_r <- numeric(p) #indicates the non-zero values of Lambdas
for( r in 1:p ){
v_r[r] <- min(r,q)
}
T_INV <- array(data = 0, dim = c(p,p))
diag(T_INV) <- diag(var(x_data))
diag(T_INV) <- 1/diag(T_INV)
ksi <- colMeans(x_data)
priorConst1 <- array(diag(T_INV)*ksi, dim =c(p,1))
# now simulate mu and Lambda
f2 <- compute_A_B_G_D_and_simulate_mu_Lambda_CUU(SigmaINV = SigmaINV,
                suff_statistics = suf_stat, OmegaINV = OmegaINV,
                K = K, priorConst1 = priorConst1, T_INV = T_INV, v_r = v_r)
# f2$mu contains the simulated means
# f2$Lambdas contains the simulated factor loadings
```

`compute_A_B_G_D_and_simulate_mu_Lambda_q0`

Computation and simulations for $q = 0$.

Description

This function simulates μ.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda_q0(SigmaINV,
suff_statistics, K, priorConst1, T_INV, v_r)
```

Arguments

SigmaINV	Precision matrix Σ^{-1}
suff_statistics	Sufficient statistics
K	Number of overfitting mixture components
priorConst1	Prior constant: $T^{-1}\xi$
T_INV	Prior parameter: $T^{-1}\xi$
v_r	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ:

mu	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

---

compute_A_B_G_D_and_simulate_mu_Lambda_q0_sameSigma

Computation and simulations for $q = 0$.

Description

This function simulates μ.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda_q0_sameSigma(SigmaINV,
suff_statistics, K, priorConst1, T_INV, v_r)
```

Arguments

SigmaINV	Precision matrix Σ^{-1}
suff_statistics	Sufficient statistics
K	Number of overfitting mixture components
priorConst1	Prior constant: $T^{-1}\xi$
T_INV	Prior parameter: $T^{-1}\xi$
v_r	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ:

`mu`	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

---

compute_A_B_G_D_and_simulate_mu_Lambda_Sj

Computation and simulations

Description

This function simulates μ and Λ.

Usage

```
compute_A_B_G_D_and_simulate_mu_Lambda_Sj(SigmaINV,
suff_statistics, OmegaINV, K, priorConst1, T_INV, v_r)
```

Arguments

`SigmaINV`	Precision matrix Σ^{-1} per component
`suff_statistics`	Sufficient statistics
`OmegaINV`	Prior parameter: Ω^{-1}
`K`	Number of overfitting mixture components
`priorConst1`	Prior constant: $T^{-1}\xi$
`T_INV`	Prior parameter: $T^{-1}\xi$
`v_r`	This vector is used to set to zero the upper right $(q-1)\times(q-1)$ diagonal block of factor loadings for identifiability purposes.

Value

A list containing a draw from the conditional distributions of μ and Λ:

`Lambdas`	$K \times p \times q$ array (factor loadings per component)
`mu`	$K \times p$ array (marginal mean per component)

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- scale(as.matrix(waveDataset1500[ 1:20, -1])) # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
n <- dim(x_data)[1]
q <- 2
K <- length(table(z))           # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
mu <- array( runif(K * p), dim = c(K,p) )
y <- array(rnorm(n = q*n), dim = c(n,q))
SigmaINV <- array(data = 0, dim = c(K,p,p) )
for(k in 1:K){
diag(SigmaINV[k,,]) <- 0.5 + 0.5*runif(p)
}
OmegaINV <- diag(q)
# compute sufficient stats
suf_stat <- compute_sufficient_statistics(y = y,
 z = z, K = K, x_data = x_data)

v_r <- numeric(p) #indicates the non-zero values of Lambdas
for( r in 1:p ){
v_r[r] <- min(r,q)
}
T_INV <- array(data = 0, dim = c(p,p))
diag(T_INV) <- diag(var(x_data))
diag(T_INV) <- 1/diag(T_INV)
ksi <- colMeans(x_data)
priorConst1 <- array(diag(T_INV)*ksi, dim =c(p,1))
# now simulate mu and Lambda
f2 <- compute_A_B_G_D_and_simulate_mu_Lambda_Sj(SigmaINV = SigmaINV,
                suff_statistics = suf_stat, OmegaINV = OmegaINV,
                K = K, priorConst1 = priorConst1, T_INV = T_INV, v_r = v_r)
# f2$mu contains the simulated means
# f2$Lambdas contains the simulated factor loadings
```

---

compute_sufficient_statistics

Compute sufficient statistics

Description

Compute sufficient statistics given y and z.

Usage

```
compute_sufficient_statistics(y, z, K, x_data)
```

Arguments

y	$n \times q$ matrix of factors
z	Allocation vector
K	Number of components
x_data	$n \times p$ matrix with observed data

Value

A list with six entries of sufficient statistics.

cluster_size	Integer vector of length K
sx	$K \times p$ array
sy	$K \times q$ array
sxx	Not used
syy	$K \times q \times q$ array
sxy	$K \times p \times q$ array

Author(s)

Panagiotis Papastamoulis

Examples

```
        data(waveDataset1500)
        x_data <- as.matrix(waveDataset1500[ 1:20, -1]) # data
        z <-  waveDataset1500[ 1:20, 1] # class
        p <- dim(x_data)[2]
        n <- dim(x_data)[1]
        q <- 2
        K <- length(table(z))           # 3 classes
        # give some arbitrary values to the parameters:
        set.seed(1)
y <- array(rnorm(n = q*n), dim = c(n,q))
# compute sufficient stats
suf_stat <- compute_sufficient_statistics(y = y,
 z = z, K = K, x_data = x_data)
```

---

compute_sufficient_statistics_given_mu

Compute sufficient statistics given mu

Description

Compute sufficient statistics given y, z and μ.

Usage

```
compute_sufficient_statistics_given_mu(y, z, K, x_data,mu)
```

Arguments

y	$n \times q$ matrix of factors
z	Allocation vector
K	Number of components
x_data	$n \times p$ matrix with observed data
mu	$K \times p$ matrix with marignal means per component

Value

A list with six entries of sufficient statistics.

cluster_size	Integer vector of length K
sx	$K \times p$ array
sy	$K \times q$ array
sxx	Not used
syy	$K \times q \times q$ array
sxy	$K \times p \times q$ array

Author(s)

Panagiotis Papastamoulis

Examples

```
        data(waveDataset1500)
        x_data <- as.matrix(waveDataset1500[ 1:20, -1]) # data
        z <-  waveDataset1500[ 1:20, 1] # class
        p <- dim(x_data)[2]
        n <- dim(x_data)[1]
        q <- 2
        K <- length(table(z))           # 3 classes
        # give some arbitrary values to the parameters:
        set.seed(1)
        mu <- array( runif(K * p), dim = c(K,p) )
y <- array(rnorm(n = q*n), dim = c(n,q))
# compute sufficient stats
suf_stat <- compute_sufficient_statistics_given_mu(y = y,
 z = z, K = K, x_data = x_data, mu = mu)
```

`compute_sufficient_statistics_q0` *Compute sufficient statistics for* $q = 0$

Description

Compute sufficient statistics given z.

Usage

```
compute_sufficient_statistics_q0(z, K, x_data)
```

Arguments

`z`	Allocation vector
`K`	Number of components
`x_data`	Data

Value

A list with six entries of sufficient statistics.

`cluster_size`	Integer vector of length K
`sx`	$K \times p$ array
`sy`	Not used here
`sxx`	Not used
`syy`	Not used here
`sxy`	Not used here

Author(s)

Panagiotis Papastamoulis

Examples

```
        data(waveDataset1500)
        x_data <- as.matrix(waveDataset1500[ 1:20, -1]) # data
        z <-  waveDataset1500[ 1:20, 1] # class
        p <- dim(x_data)[2]
        n <- dim(x_data)[1]
        q <- 2
        K <- length(table(z))           # 3 classes
# compute sufficient stats
suf_stat <- compute_sufficient_statistics_q0(
 z = z, K = K, x_data = x_data)
```

dealWithLabelSwitching *Apply label switching algorithms*

Description

This functions is a wrapper for the label.switching package and applies the ECR and ECR.ITERATIVE.1 algorithms. The model may have the same variance of error terms per cluster or not.

Usage

```
dealWithLabelSwitching(sameSigma, x_data, outputFolder, q, burn,
z.true, compute_regularized_expression, Km)
```

Arguments

sameSigma	Logical value indicating whether the parameterization with the same error variance per cluster is used.
x_data	Data
outputFolder	Name of the folder where the fabMix function has saved its output
q	Number of factors
burn	Discard observations as burn-in period (optional).
z.true	An (optional) vector of cluster assignments which is considered as the grountruth clustering of the data. Useful for direct comparisons against the real parameter values in simulated data.
compute_regularized_expression	Logical. Should regularized expression be computed?
Km	Number of components in the overfitted mixture model.

Value

The following files are produced in the output folder:

Author(s)

Panagiotis Papastamoulis

References

Papastamoulis, P. (2016). label.switching: An R Package for Dealing with the Label Switching Problem in MCMC Outputs. Journal of Statistical Software, 69(1), 1-24.

`fabMix` *Main function*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights.

Usage

```
fabMix(model, nChains, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar, rmDir, parallelModels)
```

Arguments

`model`	Any subset of "UUU" "CUU" "UCU" "CCU" "UCC" "UUC" "CUC", "CCC" indicating the fitted models. By default, all models are fitted.
`nChains`	The number of parallel heated chains. When 'dirPriorAlphas' is supplied, 'nChains' can be ignored.
`dirPriorAlphas`	vector of length `nChains` in the form of an increasing sequence of positive scalars. Each entry contains the (common) prior Dirichlet parameter for the corresponding chain. Default: `dirPriorAlphas = c(1, 1 + dN*(2:nChains - 1))/Kmax`, where `dN = 1`, for `nChains > 1`. Otherwise, `dirPriorAlphas = 1/Kmax`.
`rawData`	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
`outDir`	Name of the output folder. An error is thrown if the directory already exists inside the current working directory. Note: it should NOT correspond to an absolute path, e.g.: `outDir = \`fabMix_example\`` is acceptable, but `outDir = \`C:\Username\Documents\fabMix_example\`` is not.
`Kmax`	Number of components in the overfitted mixture. Default: 20.
`mCycles`	Number of MCMC cycles. Each cycle consists of `nIterPerCycle` MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
`burnCycles`	Number of cycles that will be discarded as burn-in period.
`g`	Prior parameter g. Default value: $g = 0.5$.
`h`	Prior parameter h. Default value: $h = 0.5$.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 0.5$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 0.5$.
`q`	A vector containing the number of factors to be fitted.
`normalize`	Should the observed data be normalized? Default value: TRUE. (Recommended)

`thinning` Optional integer denoting the thinning of the keeped MCMC cycles.

`zStart` Optional starting value for the allocation vector.

`nIterPerCycle` Number of iteration per MCMC cycle. Default value: 10.

`gibbs_z` Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.

`warm_up_overfitting` Number of iterations for the overfitting initialization scheme. Default value: 500.

`warm_up` Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 5000.

`overfittingInitialization` Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.

`progressGraphs` Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.

`gwar` Initialization parameter. Default: 0.05.

`rmDir` Logical value indicating whether to delete the `outDir` directory. Default: TRUE.

`parallelModels` Model-level parallelization: An optional integer specifying the number of cores that will be used in order to fit in parallel each member of `model`. Default: NULL (no model-level parallelization).

Details

Let $\boldsymbol{X}_i$; $i = 1, \ldots, n$ denote independent p-dimensional random vectors. Let $Y_i \in R^q$ with $q < p$ denote the latent factor for observation $i = 1, \ldots, n$. In the typical factor analysis model, each observation is modelled as $\boldsymbol{X}_i = \boldsymbol{\mu} + \boldsymbol{\Lambda}\boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma})$, where $\boldsymbol{\varepsilon}_i$ and Y_i are assumed independent, $i = 1, \ldots, n$. The $p \times q$ matrix Λ consists of the factor loadings. Assume that there are K clusters and let Z_i denotes the latent allocation of observation i to one amongs the k clusters, with prior probability $P(Z_i = k) = w_k$, $k = 1, \ldots, K$, independent for $i = 1, \ldots, n$. Following McNicholas et al (2008), the following parameterizations are used:

UUU model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k)$

UCU model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma})$

UUC model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \sigma_k \boldsymbol{I}_p)$

UCC model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \sigma \boldsymbol{I}_p)$

CUU model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}\boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma}_k)$

CCU model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}\boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma})$

CUC model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}\boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \sigma_k \boldsymbol{I}_p)$

CCC model: $(\boldsymbol{X}_i | Z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}\boldsymbol{Y}_i + \boldsymbol{\varepsilon}_i$, with $\boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \sigma \boldsymbol{I}_p)$

In all cases, $\boldsymbol{\varepsilon}_i$ and $\boldsymbol{Y}_i$ are assumed independent, $i = 1, \ldots, n$. Note that $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\Sigma}$ denote positive definite matrices, $\boldsymbol{I}_p$ denotes the $p \times p$ identity matrix and σ_k, σ denote positive scalars.

Value

An object of class `fabMix.object`, that is, a list consisting of the following entries:

`bic` Bayesian Information Criterion per model and number of factors.

`class` The estimated single best clustering of the observations according to the selected model.

`n_Clusters_per_model` The most probable number of clusters (number of non-empty components of the overfitted mixture) per model and number of factors.

`posterior_probability` The posterior probability of the estimated allocations according to the selected model.

`covariance_matrix` The estimated posterior mean of the covariance matrix per cluster according to the selected model.

`mu` The estimated posterior mean of the mean per cluster according to the selected model.

`weights` The estimated posterior mean of the mixing proportions according to the selected model.

`selected_model` Data frame containing the parameterization, number of clusters and factors of the selected model.

`mcmc` A list containing the MCMC draws for the parameters of the selected model. Each entry is returned as an `mcmc` object, a class imported from the `coda` package (Plummer et al, 2006). All component-specific parameters have been reordered according to the ECR algorithm in order to undo the label switching problem. However, the output corresponding to factor scores and factor loadings is not identifiable due to sign-switching across the MCMC trace.

`data` The observed data.

Note

It is recommended to use: `normalize = TRUE` (default). Tuning of `dirPriorAlphas` may be necessary to achieve reasonable acceptance rates of chain swaps. Note that the output is reordered in order to deal with the label switching problem, according to the ECR algorithm applied by `dealWithLabelSwitching` function.

Parallelization is enabled in both the chain-level as well as the model-level. By default all heated chains (specified by the `nchains` argument) run in parallel using (at most) the same number of threads (if available). If `parallelModels = NULL` (default), then the selected parameterizations will run (serially) on the same thread. Otherwise, if `parallelModels = x` (where `x` denotes a positive integer), the algorithm will first use `x` threads to fit the specified model parameterizations in parallel, and furthermore will also parallelize the heated chains (according to the remaining free cores on the user's system). The user should combine `parallelModels` with `nChains` efficiently, for example: if the number of available threads equals 12 and the user wishes to run 3 heated chains per model (recall that there are 8 parameterizations in total), then, an ideal allocation would be `parallelModels = 4` and `nChains = 3` because all available threads will be constantly

busy. If the user wishes to run `nChains = 4` heated chains per model using 12 threads, then an ideal allocation would be `parallelModels = 3` models running in parallel. In the case where `parallelModels*nChains > m`, with `m` denoting the available number of threads, the algorithm will first allocate min(`parallelModels`, `m`) threads to run the same number of parameterizations in parallel, and then the remaining threads (if any) will be used to process the parallel heated chains. If no other threads are available, the heated chains will be allocated to single threads.

Author(s)

Panagiotis Papastamoulis

References

Martyn Plummer, Nicky Best, Kate Cowles and Karen Vines (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC, R News, vol 6, 7-11.

McNicholas, P.D. and Murphy, T.B. Stat Comput (2008) 18: 285. https://doi.org/10.1007/s11222-008-9056-0.

Papastamoulis, P. (2018). Overfitting Bayesian mixtures of factor analyzers with an unknown number of components. Computational Statistics and Data Analysis, 124: 220-234. DOI: 10.1016/j.csda.2018.03.007.

See Also

`plot.fabMix.object`

Examples

```
# TOY EXAMPLE (very small numbers... only for CRAN check purposes)

######################################################################
# (a) using 2 cores in parallel, each one running 2 heated chains.
######################################################################
library('fabMix')

n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2      # true number of clusters

sINV_diag = 1/((1:p))  # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)

# Run `fabMix` for a _small_ number of iterations for the
# `UUU` (maximal model) and `CCC` (minimal model) parameterizations,
#  using the default prior parallel heating parameters `dirPriorAlphas`.
# NOTE: `dirPriorAlphas` may require some tuning in general.


qRange <- 2 # values for the number of factors (only the true number
```

```
#                                                    is considered here)
Kmax <- 4 # number of components for the overfitted mixture model
nChains <- 2 # number of parallel heated chains

set.seed(1)
fm <- fabMix( model = c("UUU", "CCC"), nChains = nChains,
rawData = syntheticDataset$data, outDir = "toyExample",
        Kmax = Kmax, mCycles = 4, burnCycles = 1, q = qRange,
        g = 0.5, h = 0.5, alpha_sigma = 0.5, beta_sigma = 0.5,
        warm_up_overfitting = 2, warm_up = 5)

# WARNING: the following parameters:
#  Kmax, nChains, mCycles, burnCycles, warm_up_overfitting, warm_up
#  should take (much) _larger_ values. E.g. a typical implementation consists of:
#        Kmax = 20, nChains >= 3, mCycles = 1100, burnCycles = 100,
#        warm_up_overfitting = 500, warm_up = 5000.

# You may also print and plot
# print(fm)
# plot(fm, what = "BIC")

######################################################################
# (b) using 12 cores_________________________________________________
#_______4 models with 3 heated chains running in parallel________
#_______considering all 8 model parameterizations_________________
######################################################################
## Not run:
library('fabMix')
set.seed(99)
n = 200                # sample size
p = 30                # number of variables
q = 2                # number of factors
K = 5      # number of clusters
sINV_diag = rep(1/20,p)  # diagonal of inverse variance of errors
syntheticDataset <- simData(sameLambda=FALSE,K.true = K, n = n, q = q, p = p,
sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
qRange <- 1:3 # range of values for the number of factors
Kmax <- 20 # number of components for the overfitted mixture model
nChains <- 3 # number of parallel heated chains

# the next command takes ~ 2 hours in a Linux machine with 12 threads.

fm <- fabMix( parallelModels = 4,
nChains = nChains,
model = c("UUU","CUU","UCU","CCU","UCC","UUC","CUC","CCC"),
rawData = syntheticDataset$data, outDir = "toyExample_b",
        Kmax = Kmax, mCycles = 1100, burnCycles = 100, q = qRange)

print(fm)
plot(fm, what = "BIC")
plot(fm, what = "classification_pairs")
# see also
```

```
# plot(fm)


## End(Not run)
```

---

fabMix_CxC *Function to estimate the* CUC *and* CCC *models*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights.

Usage

```
fabMix_CxC(sameSigma, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar, cccStart)
```

Arguments

sameSigma	Logical value denoting the parameterization of the error variance per component. If TRUE, the parameterization CCU is fitted. Otherwise, the parameterization CUU is fitted.
dirPriorAlphas	The prior Dirichlet parameters for each chain.
rawData	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
outDir	Name of the output folder.
Kmax	Number of components in the overfitted mixture. Default: 20.
mCycles	Number of MCMC cycles. Each cycle consists of nIterPerCycle MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
burnCycles	Number of cycles that will be discarded as burn-in period.
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
q	Number of factors q, where $1 \leq q \leq L$. An error is thrown if the Ledermann bound (L) is exceeded.
normalize	Should the observed data be normalized? Default value: TRUE.

thinning	Optional integer denoting the thinning of the keeped MCMC cycles.
zStart	Optional starting value for the allocation vector.
nIterPerCycle	Number of iteration per MCMC cycle. Default value: 10.
gibbs_z	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
warm_up_overfitting	Number of iterations for the overfitting initialization scheme. Default value: 100.
warm_up	Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 500.
overfittingInitialization	Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.
progressGraphs	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
gwar	Initialization parameter. Default: 0.05.
cccStart	Initialization from the CCC model.

Value

List of files written to outDir

Note

It is recommended to always use: normalize = TRUE (default). Tuning of dirPriorAlphas may be necessary to achieve reasonable acceptance rates of chain swaps. Also note that the output is not identifiable due to label switching and the user has to subsequently call the dealWithLabelSwitching function. See the fabMix function for examples.

Author(s)

Panagiotis Papastamoulis

See Also

fabMix

---

fabMix_CxU *Function to estimate the* CCU *and* CUU *models*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights.

Usage

```
fabMix_CxU(sameSigma, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar)
```

Arguments

`sameSigma`	Logical value denoting the parameterization of the error variance per component. If `TRUE`, the parameterization CCU is fitted. Otherwise, the parameterization CUU is fitted.
`dirPriorAlphas`	The prior Dirichlet parameters for each chain.
`rawData`	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
`outDir`	Name of the output folder.
`Kmax`	Number of components in the overfitted mixture. Default: 20.
`mCycles`	Number of MCMC cycles. Each cycle consists of `nIterPerCycle` MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
`burnCycles`	Number of cycles that will be discarded as burn-in period.
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`q`	Number of factors q, where $1 \leq q \leq L$. An error is thrown if the Ledermann bound (L) is exceeded.
`normalize`	Should the observed data be normalized? Default value: TRUE.
`thinning`	Optional integer denoting the thinning of the keeped MCMC cycles.
`zStart`	Optional starting value for the allocation vector.
`nIterPerCycle`	Number of iteration per MCMC cycle. Default value: 10.
`gibbs_z`	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
`warm_up_overfitting`	Number of iterations for the overfitting initialization scheme. Default value: 100.
`warm_up`	Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 500.
`overfittingInitialization`	Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.
`progressGraphs`	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
`gwar`	Initialization parameter. Default: 0.05.

Value

List of files written to `outDir`

Note

It is recommended to always use: `normalize = TRUE` (default). Tuning of `dirPriorAlphas` may be necessary to achieve reasonable acceptance rates of chain swaps. Also note that the output is not identifiable due to label switching and the user has to subsequently call the `dealWithLabelSwitching` function. See the `fabMix` function for examples.

Author(s)

Panagiotis Papastamoulis

See Also

`fabMix`

---

fabMix_missing_values *Function to estimate the UUU or UCU models in case of missing values*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights. Missing values are simulated from their full conditional posterior distribution.

Usage

```
fabMix_missing_values(sameSigma, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z, warm_up,
progressGraphs, gwar)
```

Arguments

`sameSigma`	Logical value denoting the parameterization of the error variance per component. If `sameSigma = TRUE`, the parameterization `UCU` is fitted, otherwise the `UUU` model is fitted.
`dirPriorAlphas`	The prior Dirichlet parameters for each chain.
`rawData`	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
`outDir`	Name of the output folder.
`Kmax`	Number of components in the overfitted mixture. Default: 20.

mCycles	Number of MCMC cycles. Each cycle consists of nIterPerCycle MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
burnCycles	Number of cycles that will be discarded as burn-in period.
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
q	Number of factors q, where $1 \leq q \leq L$. An error is thrown if the Ledermann bound (L) is exceeded.
normalize	Should the observed data be normalized? Default value: TRUE.
thinning	Optional integer denoting the thinning of the keeped MCMC cycles.
zStart	Optional starting value for the allocation vector.
nIterPerCycle	Number of iteration per MCMC cycle. Default value: 10.
gibbs_z	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
warm_up	NUmber of iterations that will be used to initialize the models before starting proposing switchings. Default value: 500.
progressGraphs	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
gwar	Initialization parameter. Default: 0.05.

Value

List of files written to outDir

Note

It is recommended to always use: normalize = TRUE (default). Tuning of dirPriorAlphas may be necessary to achieve reasonable acceptance rates of chain swaps. Also note that the output is not identifiable due to label switching and the user has to subsequently call the dealWithLabelSwitching function.

Author(s)

Panagiotis Papastamoulis

`fabMix_parallelModels` *Function for model-level parallelization*

Description

This function runs multiple copies of the `fabMix` function in parallel.

Usage

```
fabMix_parallelModels(model, nChains, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar, rmDir, parallelModels)
```

Arguments

`model`	Any subset of "UUU" "CUU" "UCU" "CCU" "UCC" "UUC" "CUC", "CCC" indicating the fitted models.
`nChains`	The number of parallel heated chains. When 'dirPriorAlphas' is supplied, 'nChains' can be ignored.
`dirPriorAlphas`	vector of length `nChains` in the form of an increasing sequence of positive scalars. Each entry contains the (common) prior Dirichlet parameter for the corresponding chain. Default: `dirPriorAlphas = c(1, 1 + dN*(2:nChains - 1))/Kmax`, where `dN = 1`, for `nChains > 1`. Otherwise, `dirPriorAlphas = 1/Kmax`.
`rawData`	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
`outDir`	Name of the output folder. An error is thrown if this directory already exists.
`Kmax`	Number of components in the overfitted mixture. Default: 20.
`mCycles`	Number of MCMC cycles. Each cycle consists of `nIterPerCycle` MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
`burnCycles`	Number of cycles that will be discarded as burn-in period.
`g`	Prior parameter g. Default value: $g = 0.5$.
`h`	Prior parameter h. Default value: $h = 0.5$.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 0.5$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 0.5$.
`q`	A vector containing the number of factors to be fitted.
`normalize`	Should the observed data be normalized? Default value: TRUE. (Recommended)
`thinning`	Optional integer denoting the thinning of the keeped MCMC cycles.
`zStart`	Optional starting value for the allocation vector.
`nIterPerCycle`	Number of iteration per MCMC cycle. Default value: 10.

gibbs_z	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
warm_up_overfitting	Number of iterations for the overfitting initialization scheme. Default value: 500.
warm_up	Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 5000.
overfittingInitialization	Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.
progressGraphs	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
gwar	Initialization parameter. Default: 0.05.
rmDir	Logical value indicating whether to delete the outDir directory. Default: TRUE.
parallelModels	Model-level parallelization: An optional integer specifying the number of cores that will be used in order to fit in parallel each member of model.

Value

An object of class fabMix.object (see the fabMix function).

Note

See the fabMix function for examples.

Author(s)

Panagiotis Papastamoulis

---

fabMix_UxC *Function to estimate the* UUC *and* UCC *models*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights.

Usage

```
fabMix_UxC(sameSigma, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar)
```

Arguments

sameSigma	Logical value denoting the parameterization of the error variance per component. If TRUE, the parameterization CCU is fitted. Otherwise, the parameterization CUU is fitted.
dirPriorAlphas	The prior Dirichlet parameters for each chain.
rawData	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
outDir	Name of the output folder.
Kmax	Number of components in the overfitted mixture. Default: 20.
mCycles	Number of MCMC cycles. Each cycle consists of nIterPerCycle MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
burnCycles	Number of cycles that will be discarded as burn-in period.
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
q	Number of factors q, where $1 \leq q \leq L$. An error is thrown if the Ledermann bound (L) is exceeded.
normalize	Should the observed data be normalized? Default value: TRUE.
thinning	Optional integer denoting the thinning of the keeped MCMC cycles.
zStart	Optional starting value for the allocation vector.
nIterPerCycle	Number of iteration per MCMC cycle. Default value: 10.
gibbs_z	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
warm_up_overfitting	Number of iterations for the overfitting initialization scheme. Default value: 100.
warm_up	Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 500.
overfittingInitialization	Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.
progressGraphs	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
gwar	Initialization parameter. Default: 0.05.

Value

List of files written to outDir

Note

It is recommended to always use: `normalize = TRUE` (default). Tuning of `dirPriorAlphas` may be necessary to achieve reasonable acceptance rates of chain swaps. Also note that the output is not identifiable due to label switching and the user has to subsequently call the `dealWithLabelSwitching` function. See the `fabMix` function for examples.

Author(s)

Panagiotis Papastamoulis

See Also

`fabMix`

---

fabMix_UxU *Function to estimate the* UUU *and* UCU *model*

Description

This function runs parallel chains under a prior tempering scheme of the Dirichlet prior distribution of mixture weights.

Usage

```
fabMix_UxU(sameSigma, dirPriorAlphas, rawData, outDir, Kmax, mCycles,
burnCycles, g, h, alpha_sigma, beta_sigma, q, normalize,
thinning, zStart, nIterPerCycle, gibbs_z,
warm_up_overfitting, warm_up, overfittingInitialization,
progressGraphs, gwar)
```

Arguments

`sameSigma`	Logical value denoting the parameterization of the error variance per component. If `TRUE`, the parameterization $\Sigma_1 = \ldots = \Sigma_K$ is fitted.
`dirPriorAlphas`	The prior Dirichlet parameters for each chain.
`rawData`	The observed data as an $n \times p$ matrix. Clustering is performed on the rows of the matrix.
`outDir`	Name of the output folder.
`Kmax`	Number of components in the overfitted mixture. Default: 20.
`mCycles`	Number of MCMC cycles. Each cycle consists of `nIterPerCycle` MCMC iterations. At the end of each cycle a swap of the state of two randomly chosen adjacent chains is attempted.
`burnCycles`	Number of cycles that will be discarded as burn-in period.
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.

alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
q	Number of factors q, where $1 \leq q \leq L$. An error is thrown if the Ledermann bound (L) is exceeded.
normalize	Should the observed data be normalized? Default value: TRUE.
thinning	Optional integer denoting the thinning of the keeped MCMC cycles.
zStart	Optional starting value for the allocation vector.
nIterPerCycle	Number of iteration per MCMC cycle. Default value: 10.
gibbs_z	Select the gibbs sampling scheme for updating latent allocations of mixture model. Default value: 1.
warm_up_overfitting	Number of iterations for the overfitting initialization scheme. Default value: 100.
warm_up	Number of iterations that will be used to initialize the models before starting proposing switchings. Default value: 500.
overfittingInitialization	Logical value indicating whether the chains are initialized via the overfitting initialization scheme. Default: TRUE.
progressGraphs	Logical value indicating whether to plot successive states of the chains while the sampler runs. Default: FALSE.
gwar	Initialization parameter. Default: 0.05.

Value

List of files written to `outDir`

Note

It is recommended to always use: `normalize = TRUE` (default). Tuning of `dirPriorAlphas` may be necessary to achieve reasonable acceptance rates of chain swaps. Also note that the output is not identifiable due to label switching and the user has to subsequently call the `dealWithLabelSwitching` function. See the `fabMix` function for examples.

Author(s)

Panagiotis Papastamoulis

See Also

`fabMix`

getStuffForDIC *Compute information criteria*

Description

This function computes four information criteria for a given run of the fabMix algorithm, namely: AIC, BIC, DIC and DIC_2. Given various runs with different number of factors, the selected model corresponds to the one with the smalled value of the selected criterion.

Usage

```
getStuffForDIC(sameSigma, sameLambda, isotropic, x_data, outputFolder, q, burn,
Km, normalize, discardLower)
```

Arguments

sameSigma	Logical value indicating whether the parameterization with the same variance of errors per component is used. Default: TRUE.
sameLambda	Logical value indicating whether the parameterization with same loadings per component is used. Default: FALSE.
isotropic	Logical value indicating whether the parameterization with isotropic error variance per component is used. Default: FALSE.
x_data	Observed data.
outputFolder	Name of the folder where the fabMix function has saved its output.
q	Number of factors. Note that this should coincide with the number of factors in the fabMix run.
burn	Discard observations as burn-in period (optional).
Km	Number of components in the overfitted mixture model. Note that this should coincide with the same entry in the fabMix run.
normalize	Should the observed data be normalized? Note that this should coincide with the same entry in the fabMix run. Default value: TRUE.
discardLower	Discard draws with log-likelihood values lower than the specific quantile. This applied only for the DIC computation.

Details

If necessary, more details than the description above

Value

The information criteria are saved to the informationCriteria_map_model.txt file in the code-outputFolder.

Note

It is well known that DIC tends to overfit, so it advised to compare models with different number of factors using AIC or BIC. The main function of the package uses BIC.

Author(s)

Panagiotis Papastamoulis

---

log_dirichlet_pdf *Log-density function of the Dirichlet distribution*

Description

Log-density function of the Dirichlet distribution

Usage

```
log_dirichlet_pdf(alpha, weights)
```

Arguments

alpha	Parameter vector
weights	Vector of weights

Value

Log-density of the $D(alpha_1, \ldots, \alpha_k)$ evaluated at $w_1, \ldots, w_k$.

Author(s)

Panagiotis Papastamoulis

---

myDirichlet *Simulate from the Dirichlet distribution*

Description

Generate a random draw from the Dirichlet distribution $D(\alpha_1, \ldots, \alpha_k)$.

Usage

```
myDirichlet(alpha)
```

Arguments

alpha	Parameter vector

Value

Simulated vector

Author(s)

Panagiotis Papastamoulis

`observed.log.likelihood0`
Log-likelihood of the mixture model

Description

Log-likelihood of the mixture model evaluated only at the alive components.

Usage

```
observed.log.likelihood0(x_data, w, mu, Lambda, Sigma, z)
```

Arguments

`x_data`	The observed data
`w`	Vector of mixture weights
`mu`	Vector of marginal means
`Lambda`	Factor loadings
`Sigma`	Diagonal of the common covariance matrix of the errors per cluster
`z`	Allocation vector

Value

Log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
```

```
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
Lambda <- array( runif(K*p*q), dim = c(K,p,q) )
SigmaINV <- array(1, dim = c(p,p))
Sigma <- 1/diag(SigmaINV)
# compute the complete.log.likelihood
observed.log.likelihood0(x_data = x_data, w = w,
mu = mu, Lambda = Lambda, Sigma = Sigma, z = z)
```

`observed.log.likelihood0_q0_sameSigma` *Log-likelihood of the mixture model for $q = 0$ and same variance of errors*

Description

Log-likelihood of the mixture model evaluated only at the alive components.

Usage

```
observed.log.likelihood0_q0_sameSigma(x_data, w, mu, Sigma, z)
```

Arguments

`x_data`	The observed data
`w`	Vector of mixture weights
`mu`	Vector of marginal means
`Sigma`	Covariance matrix of the errors per cluster
`z`	Allocation vector

Value

Log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
```

```
set.seed(1)
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
SigmaINV <- array(1, dim = c(p,p))
Sigma <- 1/diag(SigmaINV)
# compute the complete.log.likelihood
observed.log.likelihood0_q0_sameSigma(x_data = x_data, w = w,
mu = mu,  Sigma = Sigma, z = z)
```

`observed.log.likelihood0_Sj` *Log-likelihood of the mixture model*

Description

Log-likelihood of the mixture model evaluated only at the alive components.

Usage

```
observed.log.likelihood0_Sj(x_data, w, mu, Lambda, Sigma, z)
```

Arguments

`x_data`	The observed data
`w`	Vector of mixture weights
`mu`	Vector of marginal means
`Lambda`	Factor loadings
`Sigma`	$K \times p$ matrix with each row containing the diagonal of the covariance matrix of the errors per cluster
`z`	Allocation vector

Value

Log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
```

```
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
Lambda <- array( runif(K*p*q), dim = c(K,p,q) )
Sigma <- matrix(1:K, nrow = K, ncol = p)
# compute the complete.log.likelihood
observed.log.likelihood0_Sj(x_data = x_data, w = w,
mu = mu, Lambda = Lambda, Sigma = Sigma, z = z)
```

`observed.log.likelihood0_Sj_q0` *Log-likelihood of the mixture model for* $q = 0$

Description

Log-likelihood of the mixture model evaluated only at the alive components.

Usage

```
observed.log.likelihood0_Sj_q0(x_data, w, mu, Sigma, z)
```

Arguments

`x_data`	The observed data
`w`	Vector of mixture weights
`mu`	Vector of marginal means
`Sigma`	$K \times p$ matrix with each row containing the diagonal of the covariance matrix of the errors per cluster
`z`	Allocation vector

Value

Log-likelihood value

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
data(waveDataset1500)
x_data <- waveDataset1500[ 1:20, -1] # data
z <-  waveDataset1500[ 1:20, 1] # class
p <- dim(x_data)[2]
q <- 2
```

```
K <- length(table(z)) # 3 classes
# give some arbitrary values to the parameters:
set.seed(1)
w <- rep(1/K, K)
mu <- array( runif(K * p), dim = c(K,p) )
Sigma <- matrix(1:K, nrow = K, ncol = p)
# compute the complete.log.likelihood
observed.log.likelihood0_Sj_q0(x_data = x_data, w = w,
mu = mu, Sigma = Sigma, z = z)
```

`overfittingMFA` *Basic MCMC sampler for the* `UCU` *model*

Description

Gibbs sampling for fitting a mixture model of factor analyzers.

Usage

```
overfittingMFA(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

---

overfittingMFA_CCC *Basic MCMC sampler for the* CCC *model*

Description

Gibbs sampling for fitting a CCC mixture model of factor analyzers.

Usage

```
overfittingMFA_CCC(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder
Kmax	Number of mixture components
m	Number of iterations
thinning	Thinning of chain
burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_CCC <- overfittingMFA_CCC(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
```

```
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

`overfittingMFA_CCU` *Basic MCMC sampler for the* `CCU` *model*

Description

Gibbs sampling for fitting a CCU mixture model of factor analyzers.

Usage

```
overfittingMFA_CCU(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_CCU(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

---

overfittingMFA_CUC *Basic MCMC sampler for the* CUC *model*

Description

Gibbs sampling for fitting a CUC mixture model of factor analyzers.

Usage

```
overfittingMFA_CUC(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder

Kmax	Number of mixture components
m	Number of iterations
thinning	Thinning of chain
burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

Set of files written in outputDirectory.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_CUC(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

overfittingMFA_CUU *Basic MCMC sampler for the* CUU *model*

Description

Gibbs sampling for fitting a CUU mixture model of factor analyzers.

Usage

```
overfittingMFA_CUU(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder
Kmax	Number of mixture components
m	Number of iterations
thinning	Thinning of chain
burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_CUU(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

---

overfittingMFA_missing_values

Basic MCMC sampler for the case of missing data

Description

Gibbs sampling for fitting a mixture model of factor analyzers.

Usage

```
overfittingMFA_missing_values(missing_entries, x_data, originalX, outputDirectory, Kmax,
m, thinning, burn, g, h, alpha_prior, alpha_sigma,
beta_sigma, start_values, q, zStart, gibbs_z)
```

Arguments

missing_entries	list which contains the row number (1st entry) and column indexes (subsequent entries) for every row containing missing values.
x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder

Kmax	Number of mixture components
m	Number of iterations
thinning	Thinning of chain
burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

List of files

Author(s)

Panagiotis Papastamoulis

---

overfittingMFA_Sj *Basic MCMC sampler for the* UUU *model*

Description

Gibbs sampling for fitting a mixture model of factor analyzers.

Usage

```
overfittingMFA_Sj(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder
Kmax	Number of mixture components
m	Number of iterations

thinning	Thinning of chain
burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

Set of files written in outputDirectory.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_Sj(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

`overfittingMFA_Sj_missing_values` *Basic MCMC sampler for the case of missing data and different error variance*

Description

Gibbs sampling for fitting a mixture model of factor analyzers.

Usage

```
overfittingMFA_Sj_missing_values(missing_entries, x_data, originalX,
outputDirectory, Kmax,
m, thinning, burn, g, h, alpha_prior, alpha_sigma,
beta_sigma, start_values, q, zStart, gibbs_z)
```

Arguments

`missing_entries`	list which contains the row number (1st entry) and column indexes (subsequent entries) for every row containing missing values.
`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

List of files

Author(s)

Panagiotis Papastamoulis

`overfittingMFA_UCC` *Basic MCMC sampler for the* `UCC` *model*

Description

Gibbs sampling for fitting a UCC mixture model of factor analyzers.

Usage

```
overfittingMFA_UCC(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_UCC(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

---

overfittingMFA_UUC *Basic MCMC sampler for the* UUC *model*

Description

Gibbs sampling for fitting a UUC mixture model of factor analyzers.

Usage

```
overfittingMFA_UUC(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma,
start_values, q, zStart, gibbs_z)
```

Arguments

x_data	normalized data
originalX	observed raw data (only for plotting purpose)
outputDirectory	Name of the output folder
Kmax	Number of mixture components
m	Number of iterations
thinning	Thinning of chain

burn	Burn-in period
g	Prior parameter g. Default value: $g = 2$.
h	Prior parameter h. Default value: $h = 1$.
alpha_prior	Parameters of the Dirichlet prior distribution of mixture weights.
alpha_sigma	Prior parameter α. Default value: $\alpha = 2$.
beta_sigma	Prior parameter β. Default value: $\beta = 1$.
start_values	Optional (not used)
q	Number of factors.
zStart	Optional (not used)
gibbs_z	Optional

Value

Set of files written in `outputDirectory`.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
colnames(syntheticDataset$data) <- paste0("x_",1:p)
Kmax <- 4       # number of components for the overfitted mixture model

set.seed(1)
overfittingMFA_UUC(x_data = syntheticDataset$data,
originalX = syntheticDataset$data, outputDirectory = 'outDir',
Kmax = Kmax, m = 5, burn = 1,
g = 0.5, h = 0.5, alpha_prior = rep(1, Kmax),
alpha_sigma = 0.5, beta_sigma = 0.5,
start_values = FALSE, q = 2,  gibbs_z = 1)
list.files('outDir')
unlink('outDir', recursive = TRUE)
```

`overfitting_q0` *MCMC sampler for* $q = 0$

Description

Gibbs sampling for fitting a mixture model with diagonal covariance structure.

Usage

```
overfitting_q0(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma, start_values, q, zStart, gibbs_z)
```

Arguments

`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

List of files

Author(s)

Panagiotis Papastamoulis

`overfitting_q0_sameSigma` *MCMC sampler for $q = 0$ and same error variance parameterization*

Description

Gibbs sampling for fitting a mixture model with diagonal covariance structure.

Usage

```
overfitting_q0_sameSigma(x_data, originalX, outputDirectory, Kmax, m, thinning, burn,
g, h, alpha_prior, alpha_sigma, beta_sigma, start_values, q, zStart, gibbs_z)
```

Arguments

`x_data`	normalized data
`originalX`	observed raw data (only for plotting purpose)
`outputDirectory`	Name of the output folder
`Kmax`	Number of mixture components
`m`	Number of iterations
`thinning`	Thinning of chain
`burn`	Burn-in period
`g`	Prior parameter g. Default value: $g = 2$.
`h`	Prior parameter h. Default value: $h = 1$.
`alpha_prior`	Parameters of the Dirichlet prior distribution of mixture weights.
`alpha_sigma`	Prior parameter α. Default value: $\alpha = 2$.
`beta_sigma`	Prior parameter β. Default value: $\beta = 1$.
`start_values`	Optional (not used)
`q`	Number of factors.
`zStart`	Optional (not used)
`gibbs_z`	Optional

Value

List of files

Author(s)

Panagiotis Papastamoulis

`plot.fabMix.object` *Plot function*

Description

This function plots `fabMix` function.

Usage

```
## S3 method for class 'fabMix.object'
plot(x, what, variableSubset, class_mfrow, ...)
```

Arguments

`x`	An object of class `fabMix.object`, which is returned by the `fabMix` function.
`what`	One of the "BIC", "classification_matplot", "classification_pairs", "correlation", "regularized_expression". The plot will display the BIC values per model and number of factors (along with the most probable number of clusters as text), a matplot per cluster for the selected model, scatterplots pairs, the estimated correlation matrix per cluster, and the estimated regularized expression of each variable to the factor space for the selected model, respectively.
`variableSubset`	An optional subset of the variables. By default, all variables are selected.
`class_mfrow`	An optional integer vector of length 2, that will be used to set the `mfrow` for "classification_matplot" and "correlation" plots. By default, each plot is printed to a new plotting area.
`...`	ignored.

Details

When the BIC values are plotted, a number indicates the most probable number of "alive" clusters. The pairwise scatterplots (`what = "classification_pairs"`) are created using the `coordProj` function of the `mclust` package. The `what = "correlation"` is plotted using the `corrplot` package. Note that the `what = "classification_matplot"` plots the original data (before scaling and centering). On the other hand, the option `what = "classification_pairs"` plots the centered and scaled data.

Author(s)

Panagiotis Papastamoulis

References

Luca Scrucca and Michael Fop and Thomas Brendan Murphy and Adrian E. Raftery (2017). mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. The R Journal, 8(1): 205–233.

Taiyun Wei and Viliam Simko (2017). R package "corrplot": Visualization of a Correlation Matrix (Version 0.84). Available from https://github.com/taiyun/corrplot

print.fabMix.object *Print function*

Description

This function prints a summary of objects returned by the fabMix function.

Usage

```
## S3 method for class 'fabMix.object'
print(x, printSubset, ...)
```

Arguments

x	An object of class fabMix.object, which is returned by the fabMix function.
printSubset	Logical.
...	ignored.

Details

The function prints some basic information for a fabMix.object.

Author(s)

Panagiotis Papastamoulis

readLambdaValues *Read Lambda values.*

Description

Function to read Lambda values from file.

Usage

```
readLambdaValues(myFile,K,p,q)
```

Arguments

myFile	File containing Lambda values
K	Number of components
p	Number of variables
q	Number of factors

Value

$K \times p \times q$ array of factor loadings.

Author(s)

Panagiotis Papastamoulis

`simData` *Synthetic data generator*

Description

Simulate data from a multivariate normal mixture using a mixture of factor analyzers mechanism.

Usage

```
simData(sameSigma, sameLambda, p, q, K.true, n, loading_means, loading_sd, sINV_values)
```

Arguments

`sameSigma`	Logical.
`sameLambda`	Logical.
`p`	The dimension of the multivariate normal distribution ($p > 1$).
`q`	Number of factors. It should be strictly smaller than p.
`K.true`	The number of mixture components (clusters).
`n`	Sample size.
`loading_means`	A vector which contains the means of blocks of factor loadings. Default: `loading_means = c(-30,-20,-10,10, 20, 30)`.
`loading_sd`	A vector which contains the standard deviations of blocks of factor loadings. Default: `loading_sd <- rep(2, length(loading_means))`.
`sINV_values`	A vector which contains the values of the diagonal of the (common) inverse covariance matrix, if `sigmaTrue = TRUE`. An $K \times p$ matrix which contains the values of the diagonal of the inverse covariance matrix per component, if `sigmaTrue = FALSE`. Default: `sINV_values = rgamma(p, shape = 1, rate = 1)`.

Value

A list with the following entries:

`data`	$n \times p$ array containing the simulated data.
`class`	n-dimensional vector containing the class of each observation.
`factorLoadings`	$K.true \times p \times q$-array containing the factor loadings Λ_{krj} per cluster k, feature r and factor j, where $k = 1, \ldots, K$; $r = 1, \ldots, p$; $j = 1, \ldots, q$.

means	$K.true \times p$ matrix containing the marginal means μ_{kr}, $k = 1, \ldots, K$; $r = 1, \ldots, p$.
variance	$p \times p$ diagonal matrix containing the variance of errors σ_{rr}, $r = 1, \ldots, p$. Note that the same variance of errors is assumed for each cluster.
factors	$n \times q$ matrix containing the simulated factor values.
weights	$K.true$-dimensional vector containing the weight of each cluster.

Note

The marginal variance for cluster k is equal to $\Lambda_k \Lambda_k^T + \Sigma$.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')

n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                         sINV_values = sINV_diag)
summary(syntheticDataset)
```

---

update_all_y *Gibbs sampling for y in* xCx *model*

Description

Gibbs sampling for updating the factors y for models with same variance of errors per component.

Usage

```
update_all_y(x_data, mu, SigmaINV, Lambda, z)
```

Arguments

x_data	$n \times p$ matrix with obseved data
mu	$n \times p$ matrix of marginal means
SigmaINV	$p \times p$ precision matrix
Lambda	$p \times q$ matrix of factor loadings
z	Allocation vector

Value

A matrix with generated factors

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')

n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
# use the real values as input and simulate factors
update_all_y(x_data = syntheticDataset$data,
mu = syntheticDataset$means,
SigmaINV = diag(1/diag(syntheticDataset$variance)),
Lambda = syntheticDataset$factorLoadings,
z = syntheticDataset$class)
```

---

update_all_y_Sj *Gibbs sampling for y in* xUx *model*

Description

Gibbs sampling for updating the factors y for models with different variance of errors per component.

Usage

```
update_all_y_Sj(x_data, mu, SigmaINV, Lambda, z)
```

Arguments

x_data	$n \times p$ matrix with obseved data
mu	$n \times p$ matrix of marginal means
SigmaINV	$K \times p \times p$ array containing the precision matrix per component
Lambda	$p \times q$ matrix of factor loadings
z	Allocation vector

Value

A matrix with generated factors

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')

n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters

sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
# add some noise here:
SigmaINV <- array(data = 0, dim = c(K,p,p))
for(k in 1:K){
     diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# use the real values as input and simulate factors
update_all_y_Sj(x_data = syntheticDataset$data,
mu = syntheticDataset$means,
SigmaINV = SigmaINV,
Lambda = syntheticDataset$factorLoadings,
z = syntheticDataset$class)
```

---

update_OmegaINV *Gibbs sampling for* $\Omega\hat{}-1$

Description

Gibbs sampling for Ω^{-1}

Usage

```
update_OmegaINV(Lambda, K, g, h)
```

Arguments

Lambda	Factor loadings
K	Number of components
g	Prior parameter
h	Prior parameter

Value

$q \times q$ matrix Ω^{-1}

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
SigmaINV <- array(data = 0, dim = c(K,p,p))
for(k in 1:K){
     diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# use the real values as input and simulate allocations
update_OmegaINV(Lambda = syntheticDataset$factorLoadings,
        K = K, g=0.5, h = 0.5)
```

---

update_OmegaINV_Cxx *Gibbs sampling for $\Omega\hat{}-1$ for Cxx model*

Description

Gibbs sampling for Ω^{-1} for Cxx model

Usage

```
update_OmegaINV_Cxx(Lambda, K, g, h)
```

Arguments

Lambda	Factor loadings, in the form of $K \times p \times q$ matrix, under the restriction that all components share the factor loadings.
K	Number of components
g	Prior parameter
h	Prior parameter

Value

$q \times q$ matrix Ω^{-1}

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
SigmaINV <- array(data = 0, dim = c(K,p,p))
for(k in 1:K){
     diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# Use the real values as input and simulate allocations.
# Mmake sure that in this case Lambda[k,,] is the same
# for all k = 1,..., K
update_OmegaINV_Cxx(Lambda = syntheticDataset$factorLoadings,
        K = K, g=0.5, h = 0.5)
```

---

update_SigmaINV_faster

Gibbs sampling for $\Sigma\hat{}-1$

Description

Gibbs sampling for updating Σ^{-1} for the xCU model.

Usage

```
update_SigmaINV_faster(x_data, z, y, Lambda, mu, K, alpha_sigma, beta_sigma)
```

Arguments

x_data	$n \times p$ matrix containing the observed data
z	Allocation vector
y	$n \times q$ matrix containing the latent factors
Lambda	$K \times p \times q$ array with factor loadings
mu	$K \times p$ array containing the marginal means
K	Number of components
alpha_sigma	Prior parameter $alpha$
beta_sigma	Prior parameter $beta$

Value

$p \times p$ matrix with the common variance of errors per component Σ^{-1}.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)

# use the real values as input and update SigmaINV
update_SigmaINV_faster(x_data = syntheticDataset$data,
z = syntheticDataset$class,
y = syntheticDataset$factors,
Lambda = syntheticDataset$factorLoadings,
mu = syntheticDataset$means,
K = K,
alpha_sigma = 0.5, beta_sigma = 0.5)
```

`update_SigmaINV_faster_q0`
Gibbs sampling for $\Sigma\hat{}-1$ *per component for* $q = 0$

Description

Gibbs sampling for Σ^{-1} per component

Usage

```
update_SigmaINV_faster_q0( z, mu, K, alpha_sigma, beta_sigma, x_data)
```

Arguments

`z`	Allocation vector
`mu`	Marginal means
`K`	Number of components
`alpha_sigma`	Prior parameter
`beta_sigma`	Prior parameter
`x_data`	Data

Value

Σ^{-1}

Author(s)

Panagiotis Papastamoulis

`update_SigmaINV_faster_q0_sameSigma`
Gibbs sampling for $\Sigma\hat{}-1$ *per component for* $q = 0$

Description

Gibbs sampling for Σ^{-1} per component

Usage

```
update_SigmaINV_faster_q0_sameSigma( z, mu, K, alpha_sigma, beta_sigma, x_data)
```

Arguments

z	Allocation vector
mu	Marginal means
K	Number of components
alpha_sigma	Prior parameter
beta_sigma	Prior parameter
x_data	Data

Value

Σ^{-1}

Author(s)

Panagiotis Papastamoulis

---

update_SigmaINV_faster_Sj

Gibbs sampling for $\Sigma\hat{}-1$ *per component*

Description

Gibbs sampling for updating Σ^{-1} for the xUU model.

Usage

```
update_SigmaINV_faster_Sj(x_data, z, y, Lambda, mu, K, alpha_sigma, beta_sigma)
```

Arguments

x_data	$n \times p$ matrix containing the observed data
z	Allocation vector
y	$n \times q$ matrix containing the latent factors
Lambda	$K \times p \times q$ array with factor loadings
mu	$K \times p$ array containing the marginal means
K	Number of components
alpha_sigma	Prior parameter α
beta_sigma	Prior parameter β

Value

$K \times p \times p$ array with the variance of errors per component Σ_k^{-1}, $k = 1, \ldots, K$.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)

# use the real values as input and update SigmaINV
update_SigmaINV_faster_Sj(x_data = syntheticDataset$data,
z = syntheticDataset$class,
y = syntheticDataset$factors,
Lambda = syntheticDataset$factorLoadings,
mu = syntheticDataset$means,
K = K,
alpha_sigma = 0.5, beta_sigma = 0.5)
```

---

update_SigmaINV_xCC *Gibbs sampling for $\Sigma\hat{}-1$ for xCC models*

Description

Gibbs sampling for Σ^{-1} for xCC models

Usage

```
update_SigmaINV_xCC(x_data, z, y, Lambda, mu, K, alpha_sigma, beta_sigma)
```

Arguments

x_data	$n \times p$ matrix containing the observed data
z	Allocation vector
y	$n \times q$ matrix containing the latent factors
Lambda	$K \times p \times q$ array with factor loadings
mu	$K \times p$ array containing the marginal means
K	Number of components
alpha_sigma	Prior parameter $alpha$
beta_sigma	Prior parameter $beta$

Value

$p \times p$ matrix with the common variance of errors per component $\Sigma^{-1} = \sigma I_p$.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)

# use the real values as input and update SigmaINV
update_SigmaINV_xCC(x_data = syntheticDataset$data,
z = syntheticDataset$class,
y = syntheticDataset$factors,
Lambda = syntheticDataset$factorLoadings,
mu = syntheticDataset$means,
K = K,
alpha_sigma = 0.5, beta_sigma = 0.5)
```

---

update_SigmaINV_xUC *Gibbs sampling for $\Sigma\hat{}-1$ per component for xUC models*

Description

Gibbs sampling for Σ^{-1} per component for xUC models

Usage

```
update_SigmaINV_xUC(x_data, z, y, Lambda, mu, K, alpha_sigma, beta_sigma)
```

Arguments

x_data	$n \times p$ matrix containing the observed data
z	Allocation vector
y	$n \times q$ matrix containing the latent factors
Lambda	$K \times p \times q$ array with factor loadings

mu	$K \times p$ array containing the marginal means
K	Number of components
alpha_sigma	Prior parameter $alpha$
beta_sigma	Prior parameter $beta$

Value

$K \times p \times p$ array containing the inverse variance of errors per component under the restriction: $\Sigma_k^{-1} = \sigma_k I_p$, where $\sigma_k > 0$.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)

# use the real values as input and update SigmaINV
update_SigmaINV_xUC(x_data = syntheticDataset$data,
z = syntheticDataset$class,
y = syntheticDataset$factors,
Lambda = syntheticDataset$factorLoadings,
mu = syntheticDataset$means,
K = K,
alpha_sigma = 0.5, beta_sigma = 0.5)
```

---

update_z2 *Collapsed Gibbs for z using matrix inversion lemma*

Description

Collapsed Gibbs for z using matrix inversion lemma

Usage

```
update_z2(w, mu, Lambda, SigmaINV, K, x_data)
```

Arguments

w	vector with length K consisting of mixture weights
mu	$K \times p$ array containing the marginal means
Lambda	$K \times p$ array with factor loadings
SigmaINV	$p \times p$ precision matrix
K	Number of components
x_data	$n \times p$ matrix containing the observed data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
# use the real values as input and simulate allocations
update_z2(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
SigmaINV = diag(1/diag(syntheticDataset$variance)),
K = K, x_data = syntheticDataset$data)$z
```

---

update_z2_Sj *Collapsed Gibbs for z using matrix inversion lemma*

Description

Collapsed Gibbs for z using matrix inversion lemma

Usage

```
update_z2_Sj(w, mu, Lambda, SigmaINV, K, x_data)
```

Arguments

w	vector with length K consisting of mixture weights
mu	$K \times p$ array containing the marginal means
Lambda	$K \times p$ array with factor loadings
SigmaINV	$K \times p \times p$ array containing the precision matrix per component
K	Number of components
x_data	$n \times p$ matrix containing the observed data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
SigmaINV <- array(data = 0, dim = c(K,p,p))
for(k in 1:K){
diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# use the real values as input and simulate allocations
update_z2_Sj(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
SigmaINV = SigmaINV,
K = K, x_data = syntheticDataset$data)$z
```

---

update_z4 *Collapsed Gibbs for z*

Description

Collapsed Gibbs for z.

Usage

```
update_z4(w, mu, Lambda, SigmaINV, K, x_data)
```

Arguments

w	vector with length K consisting of mixture weights
mu	$K \times p$ array containing the marginal means
Lambda	$K \times p$ array with factor loadings
SigmaINV	$p \times p$ precision matrix
K	Number of components
x_data	$n \times p$ matrix containing the observed data

Value

A vector of length n with the simulated allocation of each observation among the K components.

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
# use the real values as input and simulate allocations
update_z4(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
SigmaINV = diag(1/diag(syntheticDataset$variance)),
K = K, x_data = syntheticDataset$data)$z
```

---

update_z4_Sj *Collapsed Gibbs for z*

Description

Collapsed Gibbs for z

Usage

```
update_z4_Sj(w, mu, Lambda, SigmaINV, K, x_data)
```

Arguments

w	vector with length K consisting of mixture weights
mu	$K \times p$ array containing the marginal means
Lambda	$K \times p$ array with factor loadings
SigmaINV	$K \times p \times p$ array containing the precision matrix per component
K	Number of components
x_data	$n \times p$ matrix containing the observed data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
SigmaINV <- array(data = 0, dim = c(K,p,p))
for(k in 1:K){
diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# use the real values as input and simulate allocations
update_z4_Sj(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
SigmaINV = SigmaINV,
K = K, x_data = syntheticDataset$data)$z
```

update_z_b *Gibbs sampling for z*

Description

Gibbs sampling for z: here the full conditional distribution is being used (that is, the distribution is also conditioned on the values of factors y).

Usage

```
update_z_b(w, mu, Lambda, y, SigmaINV, K, x_data)
```

Arguments

w	vector with length K consisting of mixture weights
mu	$K \times p$ array containing the marginal means
Lambda	$K \times p$ array with factor loadings
y	$n \times q$ Matrix of factors
SigmaINV	Precision matrix
K	Number of components
x_data	Data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)

# use the real values as input and simulate allocations
update_z_b(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
y = syntheticDataset$factors,
```

```
SigmaINV = diag(1/diag(syntheticDataset$variance)),
K = K, x_data = syntheticDataset$data)$z
```

`update_z_b_Sj` *Gibbs sampling for z*

Description

Gibbs sampling for z: here the full conditional distribution is being used (that is, the distribution is also conditioned on the values of factors y).

Usage

```
update_z_b_Sj(w, mu, Lambda, y, SigmaINV, K, x_data)
```

Arguments

`w`	vector with length K consisting of mixture weights
`mu`	$K \times p$ array containing the marginal means
`Lambda`	$K \times p$ array with factor loadings
`y`	$n \times q$ Matrix of factors
`SigmaINV`	$K \times p \times p$ array containing the precision matrix per component
`K`	Number of components
`x_data`	$n \times p$ matrix containing the observed data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

Examples

```
library('fabMix')
# simulate some data
n = 8                # sample size
p = 5                # number of variables
q = 2                # number of factors
K = 2                # true number of clusters
sINV_diag = 1/((1:p))    # diagonal of inverse variance of errors
set.seed(100)
syntheticDataset <- simData(sameLambda=TRUE,K.true = K, n = n, q = q, p = p,
                        sINV_values = sINV_diag)
SigmaINV <- array(data = 0, dim = c(K,p,p))
```

```
for(k in 1:K){
diag(SigmaINV[k,,]) <- 1/diag(syntheticDataset$variance) + rgamma(p, shape=1, rate = 1)
}

# use the real values as input and simulate allocations
update_z_b_Sj(w = syntheticDataset$weights, mu = syntheticDataset$means,
Lambda = syntheticDataset$factorLoadings,
y = syntheticDataset$factors,
SigmaINV = SigmaINV,
K = K, x_data = syntheticDataset$data)$z
```

`update_z_q0` *Gibbs sampling for z for $q = 0$*

Description

Gibbs sampling for z

Usage

```
update_z_q0(w, mu, SigmaINV, K, x_data)
```

Arguments

`w`	Mixture weights
`mu`	Marginal means
`SigmaINV`	Precision matrix per component
`K`	Number of components
`x_data`	Data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

update_z_q0_sameSigma *Gibbs sampling for z for $q = 0$*

Description

Gibbs sampling for z

Usage

```
update_z_q0_sameSigma(w, mu, SigmaINV, K, x_data)
```

Arguments

w	Mixture weights
mu	Marginal means
SigmaINV	Precision matrix per component
K	Number of components
x_data	Data

Value

Allocation vector

Author(s)

Panagiotis Papastamoulis

waveDataset1500 *Wave dataset*

Description

A subset of 1500 randomly sampled observations from the wave dataset (version 1), available from the UCI machine learning repository. It contains 3 classes of waves (variable `class` with values "1", "2" and "3") and 21 attributes. Each class is generated from a combination of 2 of 3 base waves with noise.

Usage

```
waveDataset1500
```

Format

A data frame with 1500 rows and 22 columns. The first column denotes the class of each observation.

Source

https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)

References

Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.

Breiman,L., Friedman,J.H., Olshen,R.A. and Stone,C.J. (1984). Classification and Regression Trees. Wadsworth International Group: Belmont, California.

Index