Package 'onlinePCA'

January 27, 2015

Type Package

Title Online Principal Component Analysis

Version 1.0-1

Date 2015-01-02

Author David Degras [aut, cre], Herve Cardot [ctb]

Maintainer David Degras <ddegrasv@depaul.edu>

Description Online PCA for multivariate and functional data using perturbation, incremental, and stochastic gradient methods.

License GPL-3

Depends R (>= 3.0.2)

Imports rARPACK, Rcpp (>= 0.11.2), splines

LinkingTo Rcpp, RcppArmadillo (>= 0.4.320.0)

URL http://cran.r-project.org/package=onlinePCA

Repository CRAN

NeedsCompilation yes

Date/Publication 2015-01-21 07:27:37

R topics documented:

`onlinePCA-package` *Online Principal Component Analysis*

Description

Online PCA using perturbation methods (`perturbationRpca`, `secularRpca`), incremental PCA (`incRpca`), and stochastic gradient approaches (`sgapca`, `snlpca`). Missing data are handled by the regression approach of Brand (2002). `batchpca` computes batch (offline) PCA based on covariance matrices. `create.basis`, `coef2fd`, `fd2coef` respectively create B-spline basis functions for functional data (FD), compute the basis coefficients of FD, and convert basis coefficients to FD. `updateMean` and `updateCovariance` update the sample mean and sample covariance.

Details

Package:	onlinePCA
Type:	Package
Version:	1.0-1
Date:	2014-12-10
License:	GPL-3
URL:	http://cran.r-project.org/package=onlinePCA

Author(s)

David Degras <ddegrasv@depaul.edu>

References

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision (ECCV).*

Gu, M. and Eisenstat, S. C. (1994). A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal of Matrix Analysis and Applications.*

Hegde et al. (2006) Perturbation-Based Eigenvector Updates for On-Line Principal Components Analysis and Canonical Correlation Analysis. *Journal of VLSI Signal Processing*.

Li, W., Yue, H. H., Valles-Cervantes, S. and Qin, S. J. (2000). Recursive PCA for adaptive process monitoring. *Journal of Process Control*.

Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

batchpca *Batch PCA of a covariance matrix*

Description

This function performs the PCA of a covariance matrix, returning an arbitrary number of principal components (eigenvectors) and corresponding eigenvalues.

Usage

```
batchpca(C, k)
```

Arguments

C	covariance matrix
k	number of PCs to compute

Details

If k is small ($k <= ncol(C)/10$), the eigendecomposition is efficiently carried out by the function eigs_sym of package rARPACK using an Implicitly Restarted Lanczos Method (IRLM). Otherwise the eigendecomposition is performed with the R base function eigen.

Value

A list with components

values	the first k eigenvalues of C
vectors	the first k PC/eigenvectors of C

References

http://www.caam.rice.edu/software/ARPACK/

Examples

```
n <- 1e4
p <- 500
k <- 10

X <- matrix(runif(n*p), n, p)
X <- X * rep(sqrt(12 * (1:p)), each = n)
C <- cov(X)
# The eigenvalues of cov(X) should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## k first eigenpairs of C using base function eigen
system.time(replicate(10, {
```

```
  eigenC1 <<- eigen(C, TRUE)
  eigenC1$values <<- eigenC1$values[1:k]
  eigenC1$vectors <<- eigenC1$vectors[,1:k]
  }))

## k first eigenpairs of C using function batchpca
system.time(replicate(10, eigenC2 <<- batchpca(C, k)))

## Check equality
all.equal(eigenC1$values, eigenC2$values)
# Reorient eigenvectors if needed
for (i in 1:k)
{
  if (sum(eigenC1$vectors[,i] * eigenC2$vectors[,i]) < 0)
    eigenC2$vectors[,i] <- - eigenC2$vectors[,i]
}
all.equal(eigenC1$vectors, eigenC2$vectors)
```

`coef2fd` *Recover functional data from their B-spline coefficients*

Description

This function computes functional data from their coefficients in a B-spline basis.

Usage

```
coef2fd(theta, basis, byrow = TRUE)
```

Arguments

`theta`	B-spline coefficients
`basis`	object created by `create.basis`
`byrow`	are the coefficients of each functional observation stored in rows (TRUE) or in columns (FALSE)?

Value

A matrix of functional data stored in the same format (row or columns) as the coefficients `theta`.

Note

In view of (online or offline) functional PCA, the coefficients `theta` are left- or right- multiplied by `M^{-1/2}` (depending on their row/column format)) before applying the B-spline matrix `B`, with `M` the Gram matrix associated to `B`.

See Also

`create.basis`, `fd2coef`

Examples

```
n <- 100 # number of curves
p <- 500  # number of observation points
x <- (1:p)/p # observation grid
nknot <- 50 # number of interior knots
par(mfrow = c(1,2))

# Simulate Brownian motion
brownian <- rnorm(n*p)/sqrt(p)
dim(brownian) <- c(n,p)
brownian <- t(apply(brownian,1,cumsum))
matplot(x, t(brownian), type = "l", ylab = "y",
main = "Sample paths of Brownian motion")

# Create B-spline basis
mybasis <- create.basis(x, nknot, 1e-4)

# Compute smooth basis coefficients
theta <- fd2coef(brownian, mybasis)

# Recover smooth functional data
brownian.smooth <- coef2fd(theta, mybasis)
matplot(x, t(brownian.smooth), type = "l", ylab = "y",
main = "Smoothed sample paths")

# Standard PCA and Functional PCA
pca <- prcomp(brownian)
fpca <- prcomp(theta)
matplot(x, pca$rotation[,1:5], type = "l", ylab = "y",
main = "Standard PCA")
matplot(x, coef2fd(fpca$rotation[,1:5], mybasis, FALSE),
type = "l", ylab = "y", main = "Functional PCA")
```

`create.basis` *Create a smooth B-spline basis*

Description

This function creates a smooth B-spline basis and provides tools to find the coefficients of functional data in the basis and to recover functional data from basis coefficients.

Usage

```
create.basis(x, nknot, lambda = 1e-09, degree = 3, nderiv = 2)
```

Arguments

`x`	vector of observation times
`nknot`	number of interior knots

lambda	smoothing parameter
degree	degree of the B splines
nderiv	order of the derivative to penalize for smoothing

Details

The knots of the B-spline basis are taken as regular quantiles of x. The total number of knots is nknot + degree + 1. The function output is typically not meant to be used directly but rather through the functions coef2fd and fd2coef.

Value

A list with fields

B	matrix of B-splines evaluated at x (each column represents a basis function)
S	matrix that maps functional data to their (smoothed) coefficients of their projection in the basis set. For the purpose of PCA, the coefficients are premultiplied by M^{1/2}, where M is the Gram matrix associated with B
invsqrtM	matrix M^{-1/2} used to recover functional # data from their coefficients in the basis set

See Also

coef2fd, fd2coef

Examples

```
n <- 100 # number of curves
p <- 500 # number of observation points
x <- (1:p)/p # observation grid
nknot <- 50  # number of interior knots
par(mfrow = c(1,2))

# Simulate Brownian motion
brownian <- rnorm(n*p)/sqrt(p)
dim(brownian) <- c(n,p)
brownian <- t(apply(brownian,1,cumsum))
matplot(x, t(brownian), type = "l", ylab = "y",
main="Sample paths of Brownian motion")

# Create B-spline basis
mybasis <- create.basis(x, nknot, 1e-4)

# Compute smooth basis coefficients
theta <- fd2coef(brownian, mybasis)

# Recover smooth functional data
brownian.smooth <- coef2fd(theta, mybasis)
matplot(x, t(brownian.smooth), type = "l", ylab = "y",
main = "Smoothed sample paths")
```

```
# Standard PCA and Functional PCA
pca <- prcomp(brownian)
fpca <- prcomp(theta)
matplot(x, pca$rotation[,1:5], type = "l", ylab = "y",
 main = "Standard PCA")
matplot(x, coef2fd(fpca$rotation[,1:5], mybasis, FALSE),
 type = "l", ylab = "y", main = "Functional PCA")
```

`fd2coef` *Compute the coefficients of functional data in a B-spline basis*

Description

This function computes the coefficients of functional data in a B-spline basis.

Usage

```
fd2coef(fd, basis, byrow = TRUE)
```

Arguments

`fd`	matrix of functional data.
`basis`	object created by `create.basis`.
`byrow`	are the functional data stored in rows (TRUE) or in columns (FALSE)?

Value

A matrix of B-spline coefficients stored in the same format (row or columns) as functional data.

Note

In view of (online or offline) functional PCA, the coefficients are smoothed and premultiplied by `M^{1/2}`, with `M` the Gram matrix associated to the B-spline matrix.

See Also

`create.basis`, `coef2fd`

Examples

```
n <- 100 # number of curves
p <- 500 # number of observation points
x <- (1:p)/p # observation grid
nknot <- 50  # number of interior knots
par(mfrow=c(1,2))

# Simulate Brownian motion
brownian <- rnorm(n*p)/sqrt(p)
dim(brownian) <- c(n,p)
```

```
brownian <- t(apply(brownian,1,cumsum))
matplot(x, t(brownian), type="l", ylab="y",
main="Sample paths of Brownian motion")

# Create B-spline basis
mybasis <- create.basis(x, nknot, 1e-4)

# Compute smooth basis coefficients
theta <- fd2coef(brownian, mybasis)

# Recover smooth functional data
brownian.smooth <- coef2fd(theta, mybasis)
matplot(x, t(brownian.smooth), type = "l", ylab = "y",
main = "Smoothed sample paths")

# Standard PCA and Functional PCA
pca <- prcomp(brownian)
fpca <- prcomp(theta)
matplot(x, pca$rotation[,1:5], type = "l", ylab = "y",
main = "Standard PCA")
matplot(x, coef2fd(fpca$rotation[,1:5], mybasis, FALSE),
type = "l", ylab = "y", main = "Functional PCA")
```

`incRpca` *Incremental PCA*

Description

This function updates the PCA using the incremental method of Arora et al. (2012).

Usage

```
incRpca(d, Q, x, n, ff = 1/n, k = ncol(Q), center)
```

Arguments

`d`	eigenvalues
`Q`	principal components (PCs) stored in columns
`x`	new data vector
`n`	sample size prior to observing `x`
`ff`	forgetting factor ($0 \leq$ `ff` ≤ 1)
`k`	maximal number of PCs to compute
`center`	optional centering vector for `x`

Details

The function provides an efficient eigendecomposition of the matrix `C = (1-ff)Qdiag(D)Q'+ \sqrt{ff(1+ff)} xx'`. After applying the function, the number of PCs is `min(k, rank([Q,x]))`.

Value

A list with components

`values`	updated eigenvalues in decreasing order
`vectors`	updated principal components

References

Arora et al. (2012). Stochastic Optimization for PCA and PLS. *50th Annual Conference on Communication, Control, and Computing (Allerton).*

Examples

```
## Simulation of Brownian motion
N <- 100  # number of paths
p <- 50  # number of observation points
k <- 10  # number of PCs to compute
X <- matrix(rnorm(N*p,sd=1/sqrt(p)), N, p)
X <- t(apply(X,1,cumsum))

## Initial PCA
n0 <- 50
pca <- eigen(cov(X[1:n0,]))
xbar <- colMeans(X[1:n0,])

## Incremental PCA
for (n in n0:(N-1))
{
  xbar <- updateMean(xbar, X[n+1,], n)
  pca <- incRpca(pca$values, pca$vectors, X[n+1,], n, k = k,
center = xbar)
}

## Batch PCA
pca1 <- eigen(cov(X))

## Error in the approximation of batch PCA (p eigenpairs)
## by incremental PCA (k<p eigenpairs)
par(mfrow=c(1,2))
plot(abs(1 - pca$values/pca1$values[1:k]), xlab = "PC index", log = "y",
ylab = "Relative error", main = "Eigenvalues", type = "b")
error <- sqrt(2 * abs(1 - abs(colSums(pca$vectors * pca1$vectors[,1:k] ))))
plot(error, xlab = "PC index", ylab = "Relative error", log = "y",
main = "Eigenvectors", type = "b")
```

`perturbationRpca` *Recursive PCA using a rank 1 perturbation method*

Description

This function recursively updates the PCA with respect to a single new data vector, using the (fast) perturbation method of Hegde et al. (2006).

Usage

```
perturbationRpca(d, Q, x, n, ff, center)
```

Arguments

`d`	vector of eigenvalues
`Q`	matrix of principal components (eigenvectors of the covariance matrix) stored in columns
`x`	new data vector
`n`	current sample size (prior to observation of new data)
`ff`	forgetting factor: a number between 0 and 1, set to `1/n` by default
`center`	centering vector (optional)

Details

The forgetting factor `ff` determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments `n` and `ff` must be specified. If `ff` is specified, its value overrides the argument `n`; otherwise, `ff` is set to `1/n` which corresponds to the assumption of a stationnary observation process. If `center` is not specified, then the data `x` is not centered prior to PCA. Missing values in `x` are imputed using the BLUP method of Brand (2002).

Value

A list with components

`values`	the updated eigenvalues.
`vectors`	the updated eigenvectors.

Note

This perturbation method is based on large sample approximations. It may be highly inaccurate for small/medium sized samples and should not be used in this case.

References

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision (ECCV).*
Hegde et al. (2006) Perturbation-Based Eigenvector Updates for On-Line Principal Components Analysis and Canonical Correlation Analysis. *Journal of VLSI Signal Processing*.

See Also

secularRpca

Examples

```
N <- 1e4
n0 <- 5e3
p <- 10
X <- matrix(runif(N*p), N, p)
X <- X * rep(sqrt(12 * (1:p)), each = N)
# The eigenvalues of cov(X) are approximately equal to 1, 2, ..., p
# and the corresponding eigenvectors are approximately equal to
# the canonical basis of R^p

## Perturbation-based recursive PCA
xbar <- colMeans(X[1:n0,])
pca <- batchpca(cov(X[1:n0,]))

for (n in (n0+1):N)
{
x <- X[n,]
xbar <- updateMean(xbar, x,  n)
pca <- perturbationRpca(pca$values, pca$vectors, x, n, center = xbar)
}

## Comparison with batch PCA
pca1 <- batchpca(cov(X))
par(mfrow=c(1,2))
plot(abs(1 - pca$values/pca1$values), xlab = "PC index", log = "y",
ylab = "Relative error", main = "Eigenvalues", type = "b")
error <- sqrt(2 * abs(1 - abs(colSums(pca$vectors * pca1$vectors))))
plot(error, xlab = "PC index", ylab = "Relative error", log = "y",
main = "Eigenvectors", type = "b")
```

---

secularRpca *Recursive PCA using secular equations*

Description

This function recursively updates a PCA with respect to a new data vector. The eigenvalues are computed as the roots of a secular equations. The principal components are either deduced by direct calculation or computed with the approximation method of Gu and Eisenstat (1994).

Usage

```
secularRpca(d, Q, x, n, ff, center, tol = 1e-10, reortho = FALSE)
```

Arguments

d	vector of eigenvalues
Q	matrix of principal components (PCs) stored in columns
x	new data vector
n	sample size before observing x
ff	forgetting factor; a real number between 0 and 1 set to 1/n by default.
center	optional centering vector for x
tol	error tolerance for the computation of eigenvalues
reortho	logical; if FALSE (default) . If TRUE, the approximation method of Gu and Eisenstat (1994) is used to compute the eigenvectors.

Details

The method of secular equations provides accurate eigenvalues in all but pathological cases. On the other hand, the perturbation method implemented by perturbationRpca typically runs much faster but is only accurate for a large sample size n.
The default method for computing PCs/eigenvalues (reortho = FALSE) is that of Li et al. (2000). It is accurate for the first few PCs but loss of accuracy and orthogonality may occur for the next PCs. In contrast the method of Gu and Eisenstat (1994) is robust against small errors in the computation of eigenvalues. It provides PCs that may be less accurate than the default method but for which strict orthogonality is guaranteed.
Lower values of the forgetting factor ff place more weight on the current PCA while higher values place more weight on the new data. The default value ff = 1/n corresponds to the usual PCA (eigenpairs of the empirical covariance matrix).
Missing values in x are imputed using the BLUP method of Brand (2002).

Value

A list with components

values	updated eigenvalues in decreasing order
vectors	updated principal components

References

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision (ECCV).*
Gu, M. and Eisenstat, S. C. (1994). A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM Journal of Matrix Analysis and Applications.*
Li, W., Yue, H. H., Valles-Cervantes, S. and Qin, S. J. (2000). Recursive PCA for adaptive process monitoring. *Journal of Process Control*.

See Also

perturbationRpca

Examples

```
# Initial data set
n <- 100
p <- 50
X <- matrix(runif(n*p),n,p)
xbar <- colMeans(X)
pca0 <- eigen(cov(X))

# New observation
newx <- runif(p)

# Recursive PCA with secular equations
xbar <- updateMean(xbar, newx, n)
secpca <- secularRpca(pca0$values, pca0$vectors, newx, n, center = xbar)

# Direct PCA
pca1 <- eigen(cov(rbind(X,newx)))

# Comparison of methods for eigenvalues
all.equal(secpca$values, pca1$values) # TRUE

# Comparison of methods for eigenvectors
error <- sqrt(2 * abs(1 - abs(colSums(secpca$vectors * pca1$vectors ))))
plot(error, xlab = "PC", ylab = "Relative error",
main = "Recursive calculation of PCs\n using secular equations")
```

---

sgapca *Stochastic Gradient Ascent PCA*

Description

This function updates the PCA using the Stochastic Gradient Ascent algorithm of Oja (1992).

Usage

```
sgapca(d, Q, x, gamma, center, type = c("exact", "nn"))
```

Arguments

d	vector of eigenvalues (optional).
Q	matrix of principal components stored in columns.
x	new data vector.
gamma	vector of gain parameters.
center	centering vector for x (optional).
type	string specifying the type of implementation: "exact" or "nn" (neural network).

Details

The argument d can be omitted if the interest lies in principal components only and not eigenvalues. Otherwise, the length of d and number of columns of Q must match.
The gain vector gamma determines the weight placed on the new data in updating each principal component. The first coefficient of gamma corresponds to the first principal component, etc.. It can be specified as a single positive number (which is recycled by the function) or as a vector of length ncol(Q). For larger values of gamma, more weight is placed on x and less on Q. A common choice for (the components of) gamma is of the form c/n, with n the sample size and c a suitable positive constant.
The Stochastic Gradient Ascent PCA can be implemented exactly or through a neural network. The latter is less accurate but faster.
Missing values in x are imputed using the BLUP method of Brand (2002).

Value

A list with components

values	updated eigenvalues in decreasing order.
vectors	updated principal components.

References

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision (ECCV).*
Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

See Also

snlpca

Examples

```
## Initialization
n <- 1e4  # sample size
n0 <- 5e3 # initial sample size
p <- 10   # number of variables
mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## SGA PCA
xbar <- colMeans(mat[1:n0,])
pca <- batchpca(cov(mat[1:n0,]))
for (i in (n0+1):n) {
  xbar <- updateMean(xbar, mat[i,],  i - 1)
  pca <- sgapca(pca$values, pca$vectors, mat[i,], 2 / i, xbar)
}
```

```
pca

# Compare to batch PCA
eigen(cov(mat), TRUE)
```

`snlpca` *Subspace Network Learning PCA*

Description

This function updates the PCA using the Subspace Network Learning algorithm of Oja (1992).

Usage

```
snlpca(d, Q, x, gamma, center, type = c("exact", "nn"))
```

Arguments

`d`	vector of eigenvalues (optional).
`Q`	matrix of principal components stored in columns.
`x`	vector of new data.
`gamma`	vector of gain parameters.
`center`	vector used to center `x` (optional).
`type`	string specifying the algorithm implementation: "exact" or "nn" (neural network).

Details

The argument `d` can be omitted if the interest lies in principal components (PC's) only and not eigenvalues.
The gain vector `gamma` determines the weight placed on the new data in updating each PC. The first coefficient of `gamma` corresponds to the first PC, etc.. `gamma` can be specified as a single positive number (recycled by the function) or as a vector of length `ncol(Q)`. It For larger values of `gamma`, more weight is placed on `x` and less on `Q`. A common choice is of the form `c/n`, with `n` the sample size and `c` a suitable positive constant.
The Subspace Network Learning PCA can be implemented exactly or through a neural network. The latter is less accurate but much faster.
Missing values in `x` are imputed using the BLUP method of Brand (2002).

Value

A matrix of (rotated) principal components stored in columns.

Note

Unlike the Stochastic Gradient Ascent algorithm which consistently estimates the principal components (i.e., the eigenvectors of the theoretical covariance matrix), the Subspace Network Learning algorithm only provides the linear space spanned by the PCs and not the PCs themselves.

References

Brand, M. (2002). Incremental singular value decomposition of uncertain data with missing values. *European Conference on Computer Vision (ECCV).*
Oja (1992). Principal components, Minor components, and linear neural networks. *Neural Networks.*

See Also

sgapca

Examples

```
## Initialization
n <- 1e4  # sample size
n0 <- 5e3 # initial sample size
p <- 10   # number of variables
mat <- matrix(runif(n*p), n, p)
mat <- mat * rep(sqrt(12 * (1:p)), each = n)
# The eigenvalues of mat should be close to 1, 2, ..., p
# and the corresponding eigenvectors should be close to
# the canonical basis of R^p

## SNL PCA
xbar <- colMeans(mat[1:n0,])
pca <- batchpca(cov(mat[1:n0,]))
for (i in (n0+1):n) {
  xbar <- updateMean(xbar, mat[i,],  i - 1)
  pca <- snlpca(pca$values, pca$vectors, mat[i,], 1 / i, xbar)
}
pca
# Corresponding projection matrix
snlP <- tcrossprod(pca$vectors)

# Compare to batch PCA
(batch <- eigen(cov(mat), TRUE)$vectors)
batchP <- tcrossprod(batch)

# Relative distance between the two projectors (Frobenius norm)
norm(snlP - batchP, "2") / norm(batchP, "2")
```

---

updateCovariance *Recursive update of a covariance matrix*

Description

This function efficiently updates a covariance matrix for new observations without entirely recomputing the covariance.

Usage

```
updateCovariance(C, x, n, xbar, ff, byrow = TRUE)
```

Arguments

C	covariance matrix.
x	vector/matrix of new data.
n	sample size prior to the observation of new data.
xbar	sample mean (vector) prior to the observation of new data.
ff	forgetting factor: a number beween 0 and 1; set to 1/n by default.
byrow	logical: should be TRUE if the new data vectors are stored in rows (i.e., variables in columns) and FALSE otherwise.

Details

The forgetting factor `ff` determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments `n` and `ff` must be specified. If `ff` is specified, its value overrides the argument `n`; otherwise, `ff` is set to `1/n` which corresponds to the assumption of a stationnary observation process.
If unspecified, the argument `byrow` defaults to TRUE if `x` is a matrix and to FALSE if `x` is a vector.

Value

The updated covariance matrix.

See Also

`updateMean`

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10
mat <- matrix(runif(n*p), n, p)

## Direct computation of the covariance
C <- cov(mat)

## Recursive computation of the covariance
xbar0 <- colMeans(mat[1:n0,])
C0 <- cov(mat[1:n0,])
Crec <- updateCovariance(C0, mat[(n0+1):n,], n0, xbar0)

## Check equality
all.equal(C, Crec)
```

updateMean *Recursive update of the sample mean vector*

Description

This functions updates the sample mean vector with respect to new data.

Usage

```
updateMean(xbar, x, n, ff, byrow = TRUE)
```

Arguments

xbar	sample mean vector.
x	new data.
n	sample size prior to the observation of new data.
ff	forgetting factor: a number between 0 and 1, set to 1/n by default.
byrow	logical; should be set to TRUE if the new data vectors are in rows (i.e., variables in columns) and FALSE otherwise.

Details

The forgetting factor ff determines the balance between past and present observations in the PCA update: the closer it is to 1 (resp. to 0), the more weight is placed on current (resp. past) observations. At least one of the arguments n and ff must be specified. If ff is specified, its value overrides the argument n; otherwise, ff is set to 1/n which corresponds to the assumption of a stationnary observation process.
If unspecified, the argument byrow defaults to TRUE if x is a matrix and to FALSE if x is a vector.

Value

The updated sample mean vector.

See Also

updateCovariance

Examples

```
n <- 1e4
n0 <- 5e3
p <- 10
mat <- matrix(runif(n*p), n, p)

## Direct computation of the mean
xbar <- colMeans(mat)
```

```
## Recursive computation of the covariance
xbar0 <- colMeans(mat[1:n0,])
xbarrec <- updateMean(xbar0, mat[(n0+1):n,], n0)

## Check equality
all.equal(xbar, xbarrec)
```

Index