Package 'bootSVD'

July 22, 2025

Title Fast, Exact Bootstrap Principal Component Analysis for High Dimensional Data

Description Implements fast, exact bootstrap Principal Component Analysis and Singular Value Decompositions for high dimensional data, as described in <doi:10.1080/01621459.2015.1062383> (see also <doi:10.48550/arXiv.1405.0922>). For data matrices that are too large to operate on in memory, users can input objects with class 'ff' (see the 'ff' package), where the actual data is stored on disk. In response, this package will implement a block matrix algebra procedure for calculating the principal components (PCs) and bootstrap PCs. Depending on options set by the user, the 'parallel' package can be used to parallelize the calculation of the bootstrap PCs.

Version 1.2

URL http://arxiv.org/abs/1405.0922

Depends R (>= 3.0.2)

Imports ff, parallel

License GPL-2

LazyData true

RoxygenNote 7.3.2

Encoding UTF-8

Maintainer Aaron Fisher <afishe27@alumni.jh.edu>

NeedsCompilation no

Author Aaron Fisher [aut, cre]

Repository CRAN

Date/Publication 2025-06-12 23:00:07 UTC

Contents

`As2Vs` *Convert low dimensional bootstrap components to high dimensional bootstrap components*

Description

Let B be the number of bootstrap samples, indexed by $b = 1, 2, ...B$. `As2Vs` is a simple function converts the list of principal component (PC) matrices for the bootstrap scores to a list of principal component matrices on the original high dimensional space. Both of these lists, the input and the output of `As2Vs`, are indexed by b.

Usage

```
As2Vs(AsByB, V, pattern = NULL, ...)
```

Arguments

`AsByB`	a list of the PCs matrices for each bootstrap sample, indexed by b. Each element of this list should be a (n by K) matrix, where K is the number of PCs of interest, and n is the sample size.
`V`	a tall (p by n) matrix containing the PCs of the original sample, where n is sample size, and p is sample dimension.
`pattern`	if `V` is a class `ff` object, the returned value will also be a class `ff` object. `pattern` is passed to ff in creation of the output.
`...`	passed to `mclapply`.

Value

a `B`-length list of (`p` by `K`) PC matrices on the original sample coordinate space (denoted here as V^b). This is achieved by the matrix multiplication $V^b = VA^b$. Note that here, V^b denotes the b^{th} bootstrap PC matrix, not V raised to the power b. This notation is the same as the notation used in (Fisher et al., 2014).

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
svdY<-fastSVD(Y)
DUt<- tcrossprod(diag(svdY$d),svdY$u)
bInds<-genBootIndeces(B=50,n=dim(DUt)[2])
bootSVD_LD_output<-bootSVD_LD(DUt=DUt,bInds=bInds,K=3,verbose=interactive())

Vs<-As2Vs(As=bootSVD_LD_output$As,V=svdY$v)
# Yields the high dimensional bootstrap PCs (left singular
# vectors of the bootstrap sample Y),
# indexed by b = 1,2...B, where B is the number of bootstrap samples
```

---

bootPCA *Quickly calculates bootstrap PCA results (wrapper for bootSVD)*

Description

All arguments are passed to [bootSVD()]. This function should be used in exactly the same way as [bootSVD()]. The only difference is that PCA typically involves re-centering each bootstrap sample, whereas calculations involving the SVD might not.

Usage

```
bootPCA(...)
```

Arguments

...	passed to [bootSVD()], with centerSamples set to TRUE.

Value

```
bootSVD(...)
```

bootSVD *Calculates bootstrap distribution of PCA (i.e. SVD) results*

Description

Applies fast bootstrap PCA, using the method from (Fisher et al., 2014). Dimension of the sample is denoted by p, and sample size is denoted by n, with $p > n$.

Usage

```
bootSVD(
  Y = NULL,
  K,
  V = NULL,
  d = NULL,
  U = NULL,
  B = 50,
  output = "HD_moments",
  verbose = getOption("verbose"),
  bInds = NULL,
  percentiles = c(0.025, 0.975),
  centerSamples = TRUE,
  pattern_V = "V_",
  pattern_Vb = "Vb_"
)
```

Arguments

Y	initial data sample, which can be either a matrix or a ff matrix. Y can be either tall (p by n) or wide (n by p). If Y is entered and V, d and U (see definitions below) are not entered, then bootSVD will also compute the SVD of Y. In this case where the SVD is computed, bootSVD will assume that the larger dimension of Y is p, and the smaller dimension of Y is n (i.e. bootSVD assumes that $p > n$. This assumption can be overriden by manually entering V, U and d. For cases where the entire data matrix can be easily stored in memory (e.g. $p < 50000$), it is generally appropriate to enter Y as a standard matrix. When Y is large enough that matrix algebra on Y is too demanding for memory though, Y should be entered as a ff object, where the actual data is stored on disk. If Y has class ff, and V, d or U is not entered, then block matrix algebra will be used to calculate the PCs and bootstrap PCs. The results of these calculations will be returned as ff objects as well.
K	number of PCs to calculate the bootstrap distribution for.
V	(optional) the (p by n) full matrix of p-dimensional PCs for the sample data matrix. If Y is wide, these are the right singular vectors of Y (i.e. $Y = UDV'$). If Y is tall, these are the left singular vectors of Y (i.e. $Y = VDU'$). In general it is assumed that $p > n$, however, this can be overridden by setting V and U

appropriately.
Like Y, the argument V can be either a standard matrix or a ff matrix. If V is a ff object, the bootstrap PCs, if requested, will be returned as ff objects as well.

d — (optional) n-length vector of the singular values of Y. For example, if Y is tall, then we have $Y = VDU'$ with D=diag(d).

U — (optional) the (n by n) full set of n-dimensional singular vectors of Y. If Y is wide, these are the left singular vectors of Y (i.e. $Y = UDV'$). If Y is tall, these are the right singular vectors of Y (i.e. $Y = VDU'$).

B — number of bootstrap samples to compute.

output — a vector telling which descriptions of the bootstrap distribution should be calculated. Can include any of the following: 'initial_SVD', 'HD_moments', 'full_HD_PC_dist', and 'HD_percentiles'. See below for explanations of these outputs.
For especially high dimensional cases, caution should be used if requesting 'full_HD_PC_dist' due to potential storage limitations.

verbose — if TRUE, the function will print progress during calculation procedure.

bInds — a (B by n) matrix of bootstrap indeces, where B is the number of bootstrap samples, and n is the sample size. The purpose of setting a specific bootstrap sampling index is to allow the results to be more precisely compared against standard bootstrap PCA calculations. If entered, the bInds argument will override the B argument.

percentiles — a vector containing percentiles to be used to calculate element-wise percentiles across the bootstrap distribution (both across the distribution of p-dimensional components and the distribution of n-dimensional components). For example, percentiles=c(.025,.975) will return the 2.5 and 97.5 percentiles, which can be used as 95 percent bootstrap percentile CIs. Alternatively, a longer vector of percentiles can be entered.

centerSamples — whether each bootstrap sample should be centered before calculating the SVD.

pattern_V — if Y is a class ff object, then the returned PCs of Y will also be a class ff object. pattern_V is passed to ff in creation of the initial_SVD output. Specifically, pattern_V is a filename prefix used for storing the high dimensional PCs of the original sample.

pattern_Vb — if Y or V is a class ff object, then the returned bootstrap PCs will also be class ff objects. pattern_Vb is passed to ff in creation of the full_HD_PC_dist output. Specifically, pattern_Vb is a filename prefix used for storing the high dimensional bootstrap PCs.

Details

Users might also consider changing the global options ffbatchbytes, from the ff package, and mc.cores, from the parallel package. When ff objects are entered as arguments for bootSVD, the required matrix algebra is done using block matrix alebra. The ffbatchbytes option determines the size of the largest block matrix that will be held in memory at any one time. The mc.cores option (set to 1 by default) determines the level of parallelization to use when calculating the high dimensional distribution of the bootstrap PCs (see mclapply).

Value

bootSVD returns a list that can include any of the following elements, depending on what is specified in the output argument:

initial_SVD The singular value decomposition of the centered, original data matrix. initial_SVD is a list containing V, the matrix of p-dimensional principal components, d, the vector of singular values of Y, and U, the matrix of n-dimensional singular vectors of Y.

HD_moments A list containing the bootstrap expected value (EPCs), element-wise bootstrap variance (varPCs), and element-wise bootstrap standard deviation (sdPCs) for each of the p-dimensional PCs. Each of these three elements of HD_moments is also a list, which contains K vectors, one for each PC. HD_moments also contains momentCI, a K-length list of (p by 2) matrices containing element-wise moment based confidence intervals for the PCs.

full_HD_PC_dist A B-length list of matrices (or ff matrices), with the b^{th} list element equal to the (p by K) matrix of high dimensional PCs for the b^{th} bootstrap sample.
For especially high dimensional cases when the output is returned as ff matrices, caution should be used if requesting 'full_HD_PC_dist' due to potential storage limitations.
To reindex these PCs by k (the PC index) as opposed to b (the bootstrap index), see [reindexMatricesByK()]. Again though, caution shoulded be used when reindexing PCs stored as ff objects, as this will double the number of files stored.

HD_percentiles A list of K matrices, each of dimension (p by q), where q is the number of percentiles requested (i.e. q = length(percentiles)). The k^{th} matrix in HD_percentiles contains element-wise percentiles for the k^{th}, p-dimensional PC.

In addition, the following results are always included in the output, regardless of what is specified in the output argument:

full_LD_PC_dist
: A B-length list of matrices, with the b^{th} list element equal to the (p by K) matrix of PCs of the scores in the b^{th} bootstrap sample. To reindex these vectors by k (the PC index), see [reindexMatricesByK()].

d_dist
: A B-length list of vectors, with the b^{th} element of d_dist containing the n-length vector of singular values from the b^{th} bootstrap sample. To reindex these values by k (the PC index), see [reindexVectorsByK()].

U_dist
: A B-length list of (n by K) matrices, with the columns of the b^{th} matrix containing the n-length singular vectors from the b^{th} bootstrap sample. To reindex these vectors by k (the PC index), see [reindexMatricesByK()].

LD_moments
: A list that is comparable to HD_moments, but that instead describes the variability of the n-dimensional principal components of the resampled score matrices. LD_moments contains the bootstrap expected value (EPCs), element-wise bootstrap variances (varPCs), and element-wise bootstrap standard deviations (sdPCs) for each of the n-dimensional PCs. Each of these three elements of LD_moments is also a list, which contains K vectors, one for each PC. LD_moments also contains momentCI, a list of K (n by 2) matrices containing element-wise, moment-based confidence intervals for the PCs.

LD_percentiles
: A list of K matrices, each of dimension (p by q), where q is the number of percentiles requested (i.e. q = length(percentiles)). The k^{th} matrix in LD_percentiles contains element-wise percentiles for the k^{th} n-dimensional PC.

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
b<-bootSVD(Y, B=50, K=2, output=
  c('initial_SVD', 'HD_moments', 'full_HD_PC_dist',
  'HD_percentiles'), verbose=interactive())
b

#explore results
matplot(b$initial_SVD$V[,1:4],type='l',main='Fitted PCs',lty=1)
legend('bottomright',paste0('PC',1:4),col=1:4,lty=1,lwd=2)

######################
# look specifically at 2nd PC
k<-2

######
#looking at HD variability

#plot several draws from bootstrap distribution
VsByK<-reindexMatricesByK(b$full_HD_PC_dist)
matplot(t(VsByK[[k]][1:20,]),type='l',lty=1,
main=paste0('20 Draws from bootstrap\ndistribution of HD PC ',k))

#plot pointwise CIs
matplot(b$HD_moments$momentCI[[k]],type='l',col='blue',lty=1,
main=paste0('CIs For HD PC ',k))
matlines(b$HD_percentiles[[k]],type='l',col='darkgreen',lty=1)
lines(b$initial_SVD$V[,k])
legend('topright',c('Fitted PC','Moment CIs','Percentile CIs'),
lty=1,col=c('black','blue','darkgreen'))
abline(h=0,lty=2,col='darkgrey')

######
# looking at LD variability

# plot several draws from bootstrap distribution
AsByK<-reindexMatricesByK(b$full_LD_PC_dist)
matplot(t(AsByK[[k]][1:50,]),type='l',lty=1,
main=paste0('50 Draws from bootstrap\ndistribution of LD PC ',k),
xlim=c(1,10),xlab='PC index (truncated)')

# plot pointwise CIs
matplot(b$LD_moments$momentCI[[k]],type='o',col='blue',
lty=1,main=paste0('CIs For LD PC ',k),xlim=c(1,10),
xlab='PC index (truncated)',pch=1)
```

```
matlines(b$LD_percentiles[[k]],type='o',pch=1,col='darkgreen',lty=1)
abline(h=0,lty=2,col='darkgrey')
legend('topright',c('Moment CIs','Percentile CIs'),lty=1,
pch=1,col=c('blue','darkgreen'))
#Note: variability is mostly due to rotations with the third and fourth PC.

# Bootstrap eigenvalue distribution
dsByK<-reindexVectorsByK(b$d_dist)
boxplot(dsByK[[k]]^2,main=paste0('Covariance Matrix Eigenvalue ',k),
ylab='Bootstrap Distribution',
ylim=range(c(dsByK[[k]]^2,b$initial_SVD$d[k]^2)))
points(b$initial_SVD$d[k]^2,pch=18,col='red')
legend('bottomright','Sample Value',pch=18,col='red')


##################
#Example with ff input
library(ff)
Yff<-as.ff(Y, pattern='Y_')
# If desired, change options in 'ff' package to
# adjust the size of matrix blocks held in RAM.
# For example:
# options('ffbatchbytes'=100000)
ff_dir<-tempdir()
pattern_V <- paste0(ff_dir,'/V_')
pattern_Vb <- paste0(ff_dir,'/Vb_')
bff <- bootSVD(Yff, B=50, K=2, output=c('initial_SVD', 'HD_moments',
  'full_HD_PC_dist', 'HD_percentiles'), pattern_V= pattern_V,
  pattern_Vb=pattern_Vb, verbose=interactive())


# Note that elements of full_HD_PC_dist and initial_SVD
# have class 'ff'
str(lapply(bff,function(x) class(x[[1]])))
#Show some results of bootstrap draws
plot(bff$full_HD_PC_dist[[1]][,k],type='l')
#Reindexing by K will create a new set of ff files.
VsByKff<-reindexMatricesByK(bff$full_HD_PC_dist,
  pattern=paste0(ff_dir,'/Vk_'))
physical(bff$full_HD_PC_dist[[1]])$filename
physical(VsByKff[[1]])$filename
matplot(t(VsByKff[[k]][1:10,]),type='l',lty=1,
main=paste0('Bootstrap Distribution of PC',k))


# Saving and moving results:
saveRDS(bff,file=paste0(ff_dir,'/bff.rds'))
close(bff$initial_SVD$V)
physical(bff$initial_SVD$V)$filename
# If the 'ff' files on disk are moved or renamed,
# this filename attribute can be changed:
old_ff_path <- physical(bff$initial_SVD$V)$filename
new_ff_path <- paste0(tempdir(),'/new_V_file.ff')
```

```
file.rename(from= old_ff_path, to= new_ff_path)
physical(bff$initial_SVD$V)$filename <- new_ff_path
matplot(bff$initial_SVD$V[,1:4],type='l',lty=1)
```

---

bootSVD_LD *Calculate bootstrap distribution of n-dimensional PCs*

Description

bootSVD_LD Calculates the bootstrap distribution of the principal components (PCs) of a low dimensional matrix. If the score matrix is inputted, the output of bootSVD_LD can be used to to calculate bootstrap standard errors, confidence regions, or the full bootstrap distribution of the high dimensional components. Most users may want to instead consider using [bootSVD()], which also calculates descriptions of the high dimensional components. Note that [bootSVD()] calls bootSVD_LD.

Usage

```
bootSVD_LD(
  UD,
  DUt = t(UD),
  bInds = genBootIndeces(B = 1000, n = dim(DUt)[2]),
  K,
  warning_type = "silent",
  verbose = getOption("verbose"),
  centerSamples = TRUE
)
```

Arguments

UD	(optional) a (n by n) matrix of scores, were rows denote individuals, and columns denote measurements in the PC space.
DUt	the transpose of UD. If both UD and UDt are entered and t(UD)!=DUt, the DUt argument will override the UD argument.
bInds	a (B by n) matrix of bootstrap indeces, where B is the number of bootstrap samples, and n is the sample size. Each row should be an indexing vector that can be used to generate a new bootstrap sample (i.e. sample(n,replace=TRUE)). The matrix of bootstrap indeces is taken as input, rather than being calculated within bootSVD_LD, so that this method can be more easily compared against traditional bootstrap SVD methods on the exact same bootstrap samples. The bInds matrix can be calculated using the helper function genBootIndeces).
K	the number of PCs to be estimated.
warning_type	passed to qrSVD, when taking the SVD of the low dimensional bootstrap score matrices.
verbose	if TRUE, a progress bar will appear.
centerSamples	whether each bootstrap sample should be centered before calculating the SVD.

Value

For each bootstrap matrix $(DU')^b$, let $svd(DU') =: A^b D^b U^b$, where A^b and U^b are (n by n) orthonormal matrices, and D^b is a (n by n) diagonal matrix K. Here we calculate only the first K columns of A^b, but all n columns of U^b. The results are stored as a list containing

As	a B-length list of the (n by K) matrices containing the first K PCs from each bootstrap sample. This list is indexed by b, with the b^{th} element containing the results from the b^{th} bootstrap sample.
ds	a B-length list of vectors, indexed by the bootstrap index b, with each vector containing the singular values of the corresponding bootstrap sample.
Us	a B-length list, indexed by the bootstrap index b, of the (n by n) matrices U^b.
time	The computation time required for the procedure, taken using system.time.

If the score matrix is inputted to bootSVD_LD, the results can be transformed to get the PCs on the original space by multiplying each matrix A^b by the PCs of the original sample, V (see [As2Vs()]). The bootstrap scores of the original sample are equal to $U^b D^b$.

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
svdY<-fastSVD(Y)
DUt<- tcrossprod(diag(svdY$d),svdY$u)
bInds<-genBootIndeces(B=50,n=dim(DUt)[2])
bootSVD_LD_output<-bootSVD_LD(DUt=DUt,bInds=bInds,K=3,verbose=interactive())
```

EEG_leadingV *Leading 5 Principal Components (PCs) from EEG dataset*

Description

This package is based on (Fisher et al., 2014), which uses as an example a subset of the electroencephalogram (EEG) measurements from the Sleep Heart Health Study (SHHS) (Quan et al. 1997). Since we cannot publish the EEG recordings from SHHS participants in this package, we instead include the summary statistics of the PCs from our subsample of the processed SHHS EEG data. These summary statistics were generated from measurements of smoothed Normalized Delta Power. This data is used by the simEEG to simulate data examples to demonstrate our functions.

Details

Specifically, EEG_leadingV is a matrix whose columns contain the leading 5 principal components of the EEG dataset.

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Stuart F Quan, Barbara V Howard, Conrad Iber, James P Kiley, F Javier Nieto, George T O'Connor, David M Rapoport, Susan Redline, John Robbins, JM Samet, et al. *The sleep heart health study: design, rationale, and methods*. Sleep, 20(12):1077-1085, 1997. 1.1

See Also

EEG_mu, EEG_score_var

EEG_mu *Functional mean from EEG dataset*

Description

This package is based on (Fisher et al., 2014), which uses as an example a subset of the electroencephalogram (EEG) measurements from the Sleep Heart Health Study (SHHS) (Quan et al. 1997). Since we cannot publish the EEG recordings from SHHS participants in this package, we instead include the summary statistics of the PCs from our subsample of the processed SHHS EEG data. These summary statistics were generated from measurements of smoothed Normalized Delta Power. This data is used by the simEEG to simulate data examples to demonstrate our functions.

Details

Specifically, EEG_mu is a vector containing the mean normalized delta power function across all subjects, for the first 7.5 hours of sleep.

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Stuart F Quan, Barbara V Howard, Conrad Iber, James P Kiley, F Javier Nieto, George T O'Connor, David M Rapoport, Susan Redline, John Robbins, JM Samet, et al. *The sleep heart health study: design, rationale, and methods*. Sleep, 20(12):1077-1085, 1997. 1.1

See Also

EEG_leadingV, EEG_score_var

EEG_score_var *Empirical variance of the first 5 score variables from EEG dataset*

Description

This package is based on (Fisher et al., 2014), which uses as an example a subset of the electroencephalogram (EEG) measurements from the Sleep Heart Health Study (SHHS) (Quan et al. 1997). Since we cannot publish the EEG recordings from SHHS participants in this package, we instead include the summary statistics of the PCs from our subsample of the processed SHHS EEG data. These summary statistics were generated from measurements of smoothed Normalized Delta Power. This data is used by the `simEEG` to simulate data examples to demonstrate our functions.

Details

Specifically, `EEG_score_var` is a vector containing the variances of the first 5 empirical score variables. Here, we refer to the score variables refer to the n-dimensional, uncorrelated variables, whose coordinate vectors are the principal components `EEG_leadingV`.

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Stuart F Quan, Barbara V Howard, Conrad Iber, James P Kiley, F Javier Nieto, George T O'Connor, David M Rapoport, Susan Redline, John Robbins, JM Samet, et al. *The sleep heart health study: design, rationale, and methods*. Sleep, 20(12):1077-1085, 1997. 1.1

See Also

`EEG_mu`, `EEG_leadingV`

fastSVD *Fast SVD of a wide or tall matrix*

Description

`fastSVD` uses the inherent low dimensionality of a wide, or tall, matrix to quickly calculate its SVD. For a matrix A, this function solves $svd(A) = UDV'$. This function can be applied to either standard matrices, or, when the data is too large to be stored in memeory, to matrices with class ff. ff objects have a representation in memory, but store their contents on disk. In these cases, `fastSVD` will implement block matrix algebra to compute the SVD.

Usage

```
fastSVD(
  A,
  nv = min(dim(A)),
  warning_type = "silent",
  center_A = FALSE,
  pattern = NULL
)
```

Arguments

A	matrix of dimension (n by m). This can be either of class `matrix` or `ff`.
nv	number of high dimensional singular vectors to obtain. If $n > m$, this is the number of n-dimensional left singular vectors to be computed. If $n < m$, this is the number of m-dimensional right singular vectors to be computed.
warning_type	passed to qrSVD, which calculates either `svd(tcrossprod(A))` or `svd(crossprod(A))`, whichever is of lower dimension.
center_A	Whether the matrix `A` should be centered before taking it's SVD. Centering is done along whichever dimension of `A` is larger. For example, if `A` is tall, then setting `center_A=TRUE` will return the SVD of `A` after centering the rows of `A`. This centering is implemented as a low dimensional matrix operation that does not require creating a copy of the original matrix `A`.
pattern	passed to ff. When `A` has class `ff`, the returned high dimensional singular vectors will also have class `ff`. The argument `pattern` is passed to ff when creating the files on disk for the high dimensional singular vectors.

Details

Users might also consider changing the global option `ffbatchbytes`, from the `ff` package. When a `ff` object is entered, the `ffbatchbytes` option determines the maximum block size in the block matrix algebra used to calculate the SVD.

Value

Let r be the rank of the matrix `A`. `fastSVD` solves $svd(A) = UDV'$, where U is an (n by r) orthonormal matrix, D is an (r by r) diagonal matrix; and V is a (m by r) orthonormal matrix. When `A` is entered as an `ff` object, the high dimensional singular vectors of `A` will be returned as an `ff` object as well. For matrices where one dimension is substantially large than the other, calculation times are considerably faster than the standard `svd` function.

Examples

```
Y<-simEEG(n=100,centered=TRUE,wide=TRUE)
svdY<-fastSVD(Y)
svdY
matplot(svdY$v[,1:5],type='l',lty=1) #sample PCs for a wide matrix are the right singular vectors
```

```
#Note: For a tall, demeaned matrix Y, with columns corresponding
#to subjects and rows to measurements,
#the PCs are the high dimensional left singular vectors.

#Example with 'ff'
dev.off()
library(ff)
Yff<-as.ff(Y)
svdYff<-fastSVD(Yff)
svdYff
matplot(svdYff$v[,1:5],type='l',lty=1)
```

`ffmatrixmult` *Matrix multiplication with "ff_matrix" or "matrix" inputs*

Description

A function for `crossprod(x,y)`, for `tcrossprod(x,y)`, or for regular matrix multiplication, that is compatible with `ff` matrices. Multiplication is done without creating new matrices for the transposes of `x` or `y`. Note, the crossproduct function can't be applied directly to objects with class `ff`.

Usage

```
ffmatrixmult(
  x,
  y = NULL,
  xt = FALSE,
  yt = FALSE,
  ram.output = FALSE,
  override.big.error = FALSE,
  ...
)
```

Arguments

`x`	a matrix or ff_matrix
`y`	a matrix or ff_matrix. If NULL, this is set equal to x, although a second copy of the matrix x is not actually stored.
`xt`	should the x matrix be transposed before multiplying
`yt`	should the y matrix be transposed before multiplying (e.g. `xt=TRUE`, `yt=FALSE` leads to `crossprod(x,y)`).
`ram.output`	force output to be a normal matrix, as opposed to an object with class `ff`.
`override.big.error`	If the dimension of the final output matrix is especially large, `ffmatrixmult` will abort, giving an error. This is meant to avoid the accidental creation of very large matrices. Set override.big.error=TRUE to bypass this error.
`...`	passed to ff.

Value

A standard matrix, or a matrix with class `ff` if one of the input matrices has class `ff`.

Examples

```
## Not run:
 library(ff)
#Tall data
y_tall<-matrix(rnorm(5000),500,10) #y tall
x_tall<-matrix(rnorm(5000),500,10)
y_wide<-t(y_tall)
x_wide<-t(x_tall)
y_tall_ff<-as.ff(y_tall) #y tall and ff
x_tall_ff<-as.ff(x_tall)
y_wide_ff<-as.ff(y_wide) #y tall and ff
x_wide_ff<-as.ff(x_wide)

 #Set options to ensure that block matrix algebra is actually done,
 #and the entire algebra isn't just one in one step.
 #Compare ffmatrixmult against output from standard methods
 options('ffbytesize'=100)

 #small final matrices
#x'x
range(  crossprod(x_tall) - ffmatrixmult(x_tall_ff, xt=TRUE)  )
range(  tcrossprod(x_wide) - ffmatrixmult(x_wide_ff, yt=TRUE)  )
range(  crossprod(x_tall,y_tall) - ffmatrixmult(x_tall_ff,y_tall_ff, xt=TRUE)  )
range(  tcrossprod(x_wide,y_wide) - ffmatrixmult(x_wide_ff,y_wide_ff, yt=TRUE)  )
range(  (x_wide%*%y_tall) - ffmatrixmult(x_wide_ff,y_tall_ff)  )

#ff + small data
s_tall <- matrix(rnorm(80),10,8)
s_wide <- matrix(rnorm(80),8,10)

#tall output
range(  crossprod(x_wide, s_tall) - ffmatrixmult(x_wide_ff, s_tall,xt=TRUE)[]  )
range(  tcrossprod(x_tall, s_wide) - ffmatrixmult(x_tall_ff, s_wide,yt=TRUE)[]  )
range( x_tall%*%s_tall - ffmatrixmult(x_tall_ff, s_tall)[])

#Wide output
range(  crossprod(s_tall, y_wide) - ffmatrixmult( s_tall, y_wide_ff,xt=TRUE)[]  )
range(  tcrossprod(s_wide, y_tall) - ffmatrixmult( s_wide,y_tall_ff,yt=TRUE)[]  )
range( s_wide%*%y_wide - ffmatrixmult(s_wide,y_wide_ff)[])

 #Reset options for more practical use
 options('ffbytesize'=16777216)


## End(Not run)
```

genBootIndeces *Generate a random set of bootstrap resampling indeces*

Description

Let n be the original sample size, p be the number of measurements per subject, and B be the number of bootstrap samples. `genBootIndeces` generates a (B by n) matrix containing B indexing vectors that can be used to create B bootstrap samples, each of size n.

Usage

```
genBootIndeces(B, n)
```

Arguments

`B`	number of desired bootstrap samples
`n`	size of original sample from which we'll be resampling.

Value

A (B by n) matrix of bootstrap indeces. Let `bInds` denote the output of `getBootIndeces`, and `Y` denote the original (p by n) sample. Then `Y[,bInds[b,]]` is the b^{th} bootstrap sample.

Examples

```
bInds<-genBootIndeces(B=50,n=200)
```

genQ *Generate random orthonormal matrix*

Description

`genQ` generates a square matrix of random normal noise, and then takes the QR decomposition to return Q, a random orthogonal square matrix.

Usage

```
genQ(n, lim_attempts = 200)
```

Arguments

`n`	the dimension of the desired random orthonormal matrix
`lim_attempts`	the random matrix of normal noise must be full rank to generate the appropriate QR decomposition. `lim_attempts` gives the maximum number of attempts for generating a full rank matrix of normal noise.

Value

a random orthonormal (n by n) matrix

Examples

```
A<-genQ(3)
round(crossprod(A),digits=10)
```

`getMomentsAndMomentCI` *Calculate bootstrap moments and moment-based confidence intervals for the PCs.*

Description

Let K be the number of PCs of interest, let B be the number of bootstrap samples, and let p be the number of measurements per subject, also known as the dimension of the sample. In general, we use k to refer to the principal component (PC) index, where $k = 1, 2, ...K$, and use b to refer to the bootstrap index, where $b = 1, 2, ...B$.

Usage

```
getMomentsAndMomentCI(AsByK, V, K = length(AsByK), verbose = FALSE)
```

Arguments

`AsByK`	a list of the bootstrap PC matrices. This list should be indexed by k, with the k^{th} element of the lsit containing a b by p matrix of results for the k^{th} PC, across bootstrap samples.
`V`	a (p by n) matrix containing the coordinate vectors for the matrices within the `AsByK` list, where n is sample size and p is sample dimension. Generally for bootstrap PCA, `AsByK` should contain the PCs for the bootstrap scores, and `V` should be the matrix of PCs from the original sample. The argument `V` may also be a ff object.
`K`	the number of leading PCs for which moments and confidence intervals should be obtained.
`verbose`	setting to `TRUE` will cause the function to print its progress in calculating the bootstrap variance for each PC.

Value

a list containing

`EVs`	a list containing element-wise bootstrap means for each of the `K` fitted PCs, indexed by `k`.
`varVs`	a list containing element-wise bootstrap variances for each of the `K` fitted PCs, indexed by `k`.

sdVs	a list containing element-wise bootstrap standard errors for each of the K fitted PCs, indexed by k.
momentCI	a list of (p by 2) matrices, indexed by k, where momentCI[[k]][j,] is the pointwise moment-based CI for the j^{th} element of the k^{th} PC.

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
svdY<-fastSVD(Y)
V<-svdY$v #right singular vectors of the wide matrix Y
DUt<- tcrossprod(diag(svdY$d),svdY$u)
bInds<-genBootIndeces(B=50,n=dim(DUt)[2])
bootSVD_LD_output<-bootSVD_LD(DUt=DUt,bInds=bInds,K=3,verbose=interactive())

AsByB<-bootSVD_LD_output$As
AsByK<-reindexMatricesByK(AsByB)
moments<-getMomentsAndMomentCI(AsByK,V,verbose=interactive())
plot(V[,1],type='l',ylim=c(-.1,.1),main='Original PC1, with CI in blue')
matlines(moments$momentCI[[1]],col='blue',lty=1)

#Can also use this function to get moments for low dimensional
#vectors A^b[,k], by setting V to the identity matrix.
moments_A<- getMomentsAndMomentCI(As=AsByK,V=diag(ncol(AsByK[[1]])))
```

---

os *Quickly print an R object's size*

Description

Quickly print an R object's size

Usage

```
os(x, units = "Mb")
```

Arguments

x	an object of interest
units	measure to print size in

Value

print(object.size(x),units=units)

Examples

```
Y<-simEEG(n=50)
os(Y)
```

qrSVD *Wrapper for svd, which uses random preconditioning to restart when svd fails to converge*

Description

In order to generate the SVD of the matrix x, qrSVD calls genQ to generate a random orthonormal matrix, and uses this random matrix to precondition x. The svd of the preconditioned matrix is calculated, and adjusted to account for the preconditioning process in order to find svd(x).

Usage

```
qrSVD(
  x,
  lim_attempts = 50,
  warning_type = "silent",
  warning_file = "qrSVD_warnings.txt",
  ...
)
```

Arguments

x	a matrix to calculate the svd for
lim_attempts	the number of tries to randomly precondition x. We generally find that one preconditioning attempt is sufficient.
warning_type	controls whether the user should be told if an orthogonal preconditioning matrix is required, or if svd gives warnings. 'silent' ignores these warnings, 'print' prints the warning to the console, and 'file' saves the warnings in a text file.
warning_file	gives the location of a file to print warnings to, if warning_type is set to 'file'.
...	parameters passed to svd, such as nv and nu.

Value

Solves $svd(x) = UDV'$, where U is an matrix containing the left singular vectors of x, D is a diagonal matrix containing the singular values of x; and V is a matrix containing the right singular vectors of x (output follows the same notation convention as the svd function).

qrSVD will attempt the standard svd function before preconditioning the matrix x.

See Also

[fastSVD()]

Examples

```
x <-matrix(rnorm(3*5),nrow=3,ncol=5)
svdx <- qrSVD(x)
svdx
```

`reindexMatricesByK` *Used for calculation of low dimensional standard errors & percentiles, by re-indexing the A^b by PC index (k) rather than bootstrap index (b).*

Description

This function is used as a precursor step for calculate bootstrap standard errors, or percentiles. For very high dimensional data, we recommend that the this function be applied to the low dimensional components A^b, but the function can also be used to reorder a list of high dimensional bootstrap PCs. It can equivalently be used to reorder a list of scores. In general, we recommend that as many operations as possible be applied to the low dimensional components, as opposed to their high dimensional counterparts. This function is called by [getMomentsAndMomentCI()].

Usage

```
reindexMatricesByK(matricesByB, pattern)
```

Arguments

`matricesByB`	a `B`-length list of (`r` by `K`) matrices from each bootstrap sample. If the list elements have class `ff`, the returned matrices will also have class `ff`.
`pattern`	(optional) passed to ff.

Value

a `K`-length list of (B by r) matrices. If elements of `matricesByB` have class `ff`, then the returned, reordered matrices will also have class `ff`.

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
svdY<-fastSVD(Y)
V<- svdY$v #original sample PCs
DUt<- tcrossprod(diag(svdY$d),svdY$u)
bInds<-genBootIndeces(B=50,n=dim(DUt)[2])
bootSVD_LD_output<-bootSVD_LD(DUt=DUt,bInds=bInds,K=3,verbose=interactive())

########
# to get 'low dimensional PC' moments and lower percentiles
AsByB<-bootSVD_LD_output$As
AsByK<-reindexMatricesByK(AsByB)

meanA1<-apply(AsByK[[1]],2,mean)
seA1<-apply(AsByK[[1]],2,sd)
pA1<-apply(AsByK[[1]],2,function(x) quantile(x,.05))
#can also use lapply to get a list (indexed by k=1,...K) of
#the means, standard errors, or percentiles for each PC.
```

```
#See example below, for high dimensional bootstrap PCs.

#Alternatively, moments can be calculated with
seA1_v2<- getMomentsAndMomentCI(As=AsByK,
V=diag(dim(AsByK[[1]])[2]))$sdPCs[[1]]
all(seA1_v2==seA1)

#Additional examples of exploring the low dimensional bootstrap
#PC distribution are given in the documentation for
#the 'bootSVD' function.
#########

#########
#High dimensional percentiles for each PC
VsByB<-As2Vs(As=AsByB,V=V)
VsByK<-reindexMatricesByK(VsByB)
percentileCI_Vs<-lapply(VsByK,function(mat_k){
apply(mat_k,2,function(x) quantile(x,c(.025,.975)))
})
k=2 # the 2nd PC is a little more interesting here.
matplot(t(percentileCI_Vs[[k]]),type='l',lty=1,col='blue')
lines(V[,k])
########

# Note: This function can also be used to reorganize the
#   high dimensional PCs. For 'ff' matrices, this will
#   create a new set of files on disk.
```

`reindexVectorsByK` *Used to study of the bootstrap distribution of the k^th singular values, by re-indexing the list of d^b vectors to be organized by PC index (k) rather than bootstrap index (b).*

Description

Used to study of the bootstrap distribution of the k^th singular values, by re-indexing the list of d^b vectors to be organized by PC index (k) rather than bootstrap index (b).

Usage

```
reindexVectorsByK(vectorsByB)
```

Arguments

`vectorsByB`	a `B`-length list, containing vectors with the `n` values from each bootstrap sample.

Value

a `K`-length list of (B by n) matrices, where each matrices' rows refers to the values from a different bootstrap sample.

Examples

```
#use small n, small B, for a quick illustration
set.seed(0)
Y<-simEEG(n=100, centered=TRUE, wide=TRUE)
svdY<-fastSVD(Y)
DUt<- tcrossprod(diag(svdY$d),svdY$u)
bInds<-genBootIndeces(B=50,n=dim(DUt)[2])
bootSVD_LD_output<-bootSVD_LD(DUt=DUt,bInds=bInds,K=3,verbose=interactive())

dsByK<-reindexVectorsByK(bootSVD_LD_output$ds)

boxplot(dsByK[[1]],main='Bootstrap distribution of 1st singular value')
```

`simEEG` *Simulation functional EEG data*

Description

Our data from (Fisher et al. 2014) consists of EEG measurements from the Sleep Heart Health Study (SHHS) (Quan et al. 1997). Since we cannot publish the EEG recordings from the individuals in the SHHS, we instead include the summary statistics of the PCs from our subsample of the processed SHHS EEG data. This data is used by the `simEEG` to simulate functional data that is approximately similar to the data used in our work. The resulting simulated vectors are always of length 900, and are generated from 5 basis vectors (see `EEG_leadingV`).

Usage

```
simEEG(n = 100, centered = TRUE, propVarNoise = 0.45, wide = TRUE)
```

Arguments

`n`	the desired sample size
`centered`	if TRUE, the sample will be centered to have mean zero for each dimension. If FALSE, measurements will be simulated from a population where the mean is equal to that observed in the sample used in (Fisher et al. 2014) (see `EEG_mu`).
`propVarNoise`	the approximate proportion of total sample variance attributable to random noise.
`wide`	if TRUE, the resulting data is outputted as a `n` by 900 matrix, with each row corresponding to a different subject. If FALSE, the resulting data is outputted as a 900 by `n` matrix, with each column corresponding to a different subject.

Value

A matrix containing `n` simulated measurement vectors of Normalized Delta Power, for the first 7.5 hours of sleep. These vectors are generated according to the equation:

$y = \sum_{j=1}^{5} B_j * s_j + e$

Where y is the simulated measurement for a subject, B_j is the j^{th} basis vector, s_j is a random normal variable with mean zero, and e is a vector of random normal noise. The specific values for B_j and $var(s_j)$ are determined from the EEG data sample studied in (Fisher et al., 2014), and are respectively equal to the j^{th} empirical principal component vector (see EEG_leadingV), and the empirical variance of the j^{th} score variable (see EEG_score_var).

References

Aaron Fisher, Brian Caffo, and Vadim Zipunnikov. *Fast, Exact Bootstrap Principal Component Analysis for p>1 million*. 2014. http://arxiv.org/abs/1405.0922

Stuart F Quan, Barbara V Howard, Conrad Iber, James P Kiley, F Javier Nieto, George T O'Connor, David M Rapoport, Susan Redline, John Robbins, JM Samet, et al. *The sleep heart health study: design, rationale, and methods*. Sleep, 20(12):1077-1085, 1997. 1.1

Examples

```
set.seed(0)

#Low noise example, for an illustration of smoother functions
Y<-simEEG(n=20,centered=FALSE,propVarNoise=.02,wide=FALSE)
matplot(Y,type='l',lty=1)

#Higher noise example, for PCA
Y<-simEEG(n=100,centered=TRUE,propVarNoise=.5,wide=TRUE)
svdY<-fastSVD(Y)
V<-svdY$v #since Y is wide, the PCs are the right singular vectors (svd(Y)$v).
d<-svdY$d
head(cumsum(d^2)/sum(d^2),5) #first 5 PCs explain about half the variation

# Compare fitted PCs to true, generating basis vectors
# Since PCs have arbitrary sign, we match the sign of
# the fitted sample PCs to the population PCs first
V_sign_adj<- array(NA,dim=dim(V))
for(i in 1:5){
V_sign_adj[,i]<-V[,i] * sign(crossprod(V[,i],EEG_leadingV[,i]))
}
par(mfrow=c(1,2))
matplot(V_sign_adj[,1:5],type='l',lty=1,
main='PCs from simulated data,\n sign adjusted')
matplot(EEG_leadingV,type='l',lty=1,main='Population PCs')
```

Index