# Package 'bigpca'

September 1, 2014

**Type** Package

**Title** PCA, transpose and multicore functionality for big.matrix objects

**Version** 1.0.2

**Date** 2014-09-01

**Author** Nicholas Cooper

**Maintainer** Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

**Depends** R (>= 3.0), grDevices, graphics, stats, utils, reader (>= 1.0.1), NCmisc (>= 1.1), bigmemory (>= 4.0.0), biganalytics

**Imports** parallel, methods, bigmemory.sri, irlba

**OS_type** unix

**Description** This package adds wrappers to add functionality for big.matrix objects (see the bigmemory project). This allows fast scalable principle components analysis (PCA),or singular value decomposition (SVD). There are also functions for transposing, using multicore 'apply' functionality, data importing and for compact display of big.matrix objects. Most functions also work for standard matrices if RAM is sufficient.

**License** GPL (>= 2)

**Collate** 'bigpca.R'

## R topics documented:

---

`bigpca-package` *PCA, transpose and multicore functionality for big.matrix objects*

---

## Description

This package adds wrappers to add functionality for big.matrix objects (see the bigmemory project). This allows fast scalable principle components analysis (PCA), or singular value decomposition (SVD). There are also functions for transposing, using multicore 'apply' functionality, data importing, and for compact display of big.matrix objects. Most functions also work for standard matrices if RAM is sufficient.

## Details

| | |
|---|---|
| Package: | bigpca |
| Type: | Package |
| Version: | 1.0.2 |
| Date: | 2014-09-01 |
| License: | GPL (>= 2) |

The bigmemory project has provided a useful new data structure 'big.matrix', which allows fast and efficient access to an object that is only limited by disk-space and not RAM capacity. This package provides wrappers to extend the library of functions available for big.matrix objects. The focus of this package are functions for multicore functionality and Principle Components Analysis (PCA)/Singular Value Decomposition (SVD). bmcapply() works similarly to mcapply but is for big.matrix objects. There is a transpose function (which is not super-fast, but can be run with multiple cores to improve speed). There are several functions dedicated to PCA/SVD. These operations still require a large amount of RAM for large matrices, but the speed is greatly increased and there are useful tools allowing PCA/SVD of much larger matrices than would be feasible otherwise. There are also functions for determining the 'elbow' of the data, making scree plots, estimating variance explained for incomplete sets of eigenvalues, and for using the derived principle compo-

nents for correction of a dataset. The PC correction algorithm is fast and can be run with multiple cores simultaneously. There is also a new function prv.big.matrix() for compactly previewing large matrices, and get.big.matrix() for flexibly retrieving a big.matrix object from a range of different formats.

List of key functions:

- *big.algebra.install.help* install the big algebra package, or provide tips if it fails
- *big.PCA* PCA or SVD of a big.matrix object
- *big.select* select a subset of a big.matrix
- *bmcapply* multicore apply function for big.matrix
- *estimate.eig.vpcs* estimate uncalculated eigenvalues
- *generate.test.matrix* easily generate a random dataset for testing/simulation
- *get.big.matrix* obtain a big.matrix object via several possible methods
- *import.big.data* import data from text files efficiently into a big.matrix
- *PC.correct* correct a dataset (big.matrix) for n principle components
- *pca.scree.plot* draw a scree plot for a PCA / SVD
- *prv.big.matrix* compact preview for big.matrix objects
- *quick.elbow* calculate the elbow of a scree plot
- *quick.pheno.assocs* simple phenotype association test
- *select.least.assoc* choose subset of big.matrix variables least associated with a phenotype
- *subcor.select* choose a subset of a big.matrix that is most/least correlated with other variables
- *subpc.select* choose a subset of a big.matrix that is most representative of the principle components
- *svn.bigalgebra.install* install the big algebra package from SVN if command is available
- *big.t* transpose function for big.matrix (can be multicore)
- *thin* reduce the size of a big.matrix whilst preserving important data relationships
- *uniform.select* select a random or uniform subset of a big.matrix

## Author(s)

Nicholas Cooper

Maintainer: Nicholas Cooper <nick.cooper@cimr.cam.ac.uk>

## See Also

NCmisc ~~

## Examples

```
# # create a test big.matrix object (file-backed)
# bM <- filebacked.big.matrix(20, 50,
#        dimnames = list(paste("r",1:20,sep=""), paste("c",1:50,sep="")),
#        backingfile = "test.bck",  backingpath = getwd(), descriptorfile = "test.dsc")
# bM[1:20,] <- replicate(50,rnorm(20))
# prv.big.matrix(bM)
# # now transpose
# tbM <- big.t(bM,dir=getwd(),verbose=T)
# prv.big.matrix(tbM,row=10,col=4)
# colSDs <- bmcapply(tbM,2,sd,n.cores=10)
# rowSDs <- bmcapply(bM,1,sd,n.cores=10) # use up to 10 cores if available
# ##  generate some data with reasonable intercorrelations ##
# mat <- sim.cor(500,200,genr=function(n){ (runif(n)/2+.5) })
# bmat <- as.big.matrix(mat)
# # calculate PCA
# result <- big.PCA(bmat)
# corrected <- PC.correct(result2,bmat)
# corrected2 <- PC.correct(result2,bmat,n.cores=5)
# all.equal(corrected,corrected2)
```

---

`big.algebra.install.help`

*Attempt to install the bigalgebra package*

---

## Description

The bigalgebra package has now been submitted to CRAN, so this function is now mostly redundant. It may still be useful for some, and it will still work, as the first step to check CRAN, so at the risk of affecting existing code I will leave the function here for now. This function attempts to see whether bigalgebra is installed, then checks CRAN in case it has been updated, then check RForge. Failing that, it will attempt to install using svn.bigalgebra.install(). Returns TRUE if already installed. The bigalgebra package for efficient algebraic operations on big.matrix objects was not currently on CRAN, and used to fail a check on dependencies. Changing the description file was needed to add the dependency, and linking 'BH' allow3e the package to work. This function attempts to check-out the latest version of bigalgebra from SVN version management system and corrects the description file then installs. Note you must also have 'BLAS' installed on your system to utilise this package effectively. PCA functions in the present package are better with bigalgebra installed, but will still run without it. For more information on installation alternatives, type big.algebra.install.help().

## Usage

```
big.algebra.install.help(verbose = FALSE)
```

## Arguments

| | |
|---|---|
| `verbose` | whether to report on installation progress/steps |

### Value

If bigalgebra is already installed, or can be installed from RForge or SVN, this should load or install the bigalgebra package, else will return instructions on what to do next to fix the issue

### See Also

svn.bigalgebra.install

### Examples

```
# not run # big.algebra.install.help(TRUE)
```

---

## big.PCA *PCA/Singular Value Decomposition for big.matrix*

---

### Description

At the time of submission there was no existing native method to conduct principle components analysis (PCA) on big.matrix objects. This function allows singular value decomposition (SVD) of very large matrices, very efficiently, versus the default method. The major speed advantages occur when the 'bigalgebra' package is installed, and when the argument for this function 'SVD'=TRUE. Regular PCA can be conducted using SVD=FALSE but it will be slower and the maximum matrix size able to produce a result, given memory limits, will be smaller. SVD is not exactly the same as PCA, but from my testing the components produced will correlate R>.9 with components of PCA on the same matrix. This function is not completely native to big.matrix objects so there is one step where the matrix submitted needs to be loaded into memory, so if your big.matrix object is larger than the allowed size of a standard R-matrix [[which is roughly 3GB; you can check using NCmisc::estimate.memory()], then this function will fail unless you set the option 'thin' to a percentage that, multiplied by the original matrix memory-load, is under 3GB. For large matrices in my applications, components produced with thinning are still highly correlated with components produced using the full dataset. For a breakdown of thinning methods, see the description for the function thin() for more information. Even with medium sized matrices, for instance 15,000 x 50,000 in size, this function is orders of magnitude faster than the standard R PCA functions, usually running in a matter of minutes, rather than hours or days in examples that I have tested, due to much better handling of memory for internal transpose and eigen operations by using the 'bigmemory' architecture.

### Usage

```
big.PCA(bigMat, dir = getwd(), pcs.to.keep = 50, thin = FALSE,
  SVD = TRUE, LAP = FALSE, center = TRUE, save.pcs = FALSE,
  use.bigalgebra = TRUE, pcs.fn = "PCsEVsFromPCA.RData", verbose = FALSE,
  ...)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix object, or any argument accepted by get.big.matrix(), which includes paths to description files or even a standard matrix object. |
| dir | directory containing the filebacked.big.matrix, and also where the output file will be saved by default if the save.pcs option is set TRUE. |
| pcs.to.keep | integer, number of principle components to keep. Singular Value Decomposition methods are able to run faster if they don't need to calculate every single PC for a large matrix. Default is to calculate only the first 50; in practice even fewer than this are generally used directly. Apart from reducing processing time, this can also reduce storage/RAM burden for the resulting matrix. Set to NA, or a number >= min(dim(bigMat)) in order to keep all PCs. |
| thin | decimal, percentage of the original number of rows you want to thin the matrix to. see function thin() for details of how this can be done, pass arguments to thin() using ... Even though this PCA function uses mainly 'big.matrix' native methods, there is a step where the matrix must be stored fully in memory, so this limits the size of what matrix can be processed, depending on RAM limits. If you want to conduct PCA/SVD on a matrix larger than RAM you can thin the matrix to a percentage of the original size. Usually such a large matrix will contain correlated measures and so the exclusion of some data-rows (variables) will have only a small impact on the resulting principle components. In some applications tested using this function, using only 5 of 200,000 variables a PCA gave extremely similar results to using the full dataset. |
| SVD | logical, whether to use a Singular Value Decomposition method or a PCA method. The eigenvalues and eigenvectors of each alternative will be highly correlated so for most applications, such as PC-correction, this shouldn't make much difference to the result. However, using SVD=TRUE can provide significant advantages in speed, or even allow a solution on a matrix that would be to large to practically compute full PCA. Note that only in SVD mode, and with the bigalgebra package installed will the full speed advantage of this function be utilised. |
| LAP | logical, whether to use La.svd() instead of svd() when SVD=TRUE, see base:svd for more info. |
| center | whether to 'centre' the matrix rows by subtracting rowMeans() before conducting the PCA. This is usually advisable, although you may wish to skip this if the matrix is already centred to save extra processing. unlike prcomp there is no option to standardize or use the correlation matrix, if this is desired, please standardize the bigMat object before running this function. |
| save.pcs | whether to save the principle component matrix and eigenvalues to a binary file with name pcs.fn |
| pcs.fn | name of the binary when save.pcs=TRUE |
| verbose | whether to display detailed progress of the PCA |
| use.bigalgebra | logical, whether to use the bigalgebra package for algebraic operations. For large datasets bigalgebra should provide a substantial speedup, and also facilitates use of larger matrices. This relies on having bigalgebra installed and loaded, which requires some manual configuration as bigalgebra is not currently |

available on CRAN, but only SVN and RForge. See svn.bigalgebra.install() or big.algebra.install.help() Default is to use bigalgebra if available (TRUE), but setting this FALSE prevents the check for bigalgebra which would be cleaner if you know that you don't have it installed.

... if thin is TRUE, then these should be any additional arguments for thin(), e.g, 'keep', 'how', etc.

## Value

A list of principle components/singular vectors (may be incomplete depending on options selected), and of the eigenvalues/singular values.

## Author(s)

Nicholas Cooper

## See Also

get.big.matrix, PC.correct

## Examples

```
# create an example matrix and its transpose
min.dim <- 200; nvar <- 500; subset.size <- 50
mat <- matrix(rnorm(min.dim*nvar),ncol=min.dim)
prv.large(mat)
t.mat <- t(mat)
# create two alternative covariance matrices
MMs <- t.mat %*% mat
MsM <- mat %*% t.mat
# run singular value decomposition
pca <- svd(mat)
D <- pca$d # singular values (=sqrt(eigenvalues))
V <- pca$v # right singular vector
U <- pca$u # left singular vector
sig <- mat-mat; diag(sig) <- D;
MMs2 <- V %*% (t(sig) %*% sig) %*% t(V)
sig <- t.mat-t.mat; diag(sig) <- D;
MsM2 <- U %*% (sig %*% t(sig)) %*% t(U)
# show that the covariance matrices are equal to the functions of
# the left and right singular vectors
prv(MMs,MsM); prv(MMs2,MsM2)
pr <- princomp(mat) # PCA using eigendecomposition of cov matrix
L <- matrix(rep(0,40000),ncol=200); diag(L) <- pr[[1]]^2 # eigenvalues as diag
mat2 <- (pr[[2]]) %*% L %*%  solve(pr[[2]]) # = eigenvectors * eigenvalues * inv(eigenvectors)
prv.large(cov(mat)); prv.large(mat2) #  == COVmat (may be slight tolerance differences)
## Now demonstrate the correlation between SVD and PCA ##
# the right singular vector is highly correlated with the pca loadings:
median(abs(diag(cor(V,pr[["loadings"]]))))
# the left singular vector is highly correlated with the pca scores (eigenvectors):
median(abs(diag(cor(U,pr[["scores"]]))))
cor(pr$sdev,D) # the singular values are equivalent to the eigenvalues
```

```
## MAIN EXAMPLES ##
bmat <- as.big.matrix(mat,backingfile="testMyBig.bck",descriptorfile="testMyBig.dsc")
result <- big.PCA(bmat) #,verbose=TRUE)
headl(result)
# plot the eigenvalues with a linear fit line and elbow placed at 13
Eigv <- pca.scree.plot(result$Evalues,M=bmat,elbow=6,printvar=FALSE)
##  generate some data with reasonable intercorrelations ##
mat2 <- sim.cor(500,200,genr=function(n){ (runif(n)/2+.5) })
bmat2 <- as.big.matrix(mat2,backingfile="testMyBig.bck",descriptorfile="testMyBig.dsc")
# calculate PCA on decreasing subset size
result2 <- big.PCA(bmat2,thin=FALSE)
result3 <- big.PCA(bmat2,thin=TRUE,keep=.5)
result4 <- big.PCA(bmat2,thin=TRUE,keep=.5,how="cor")
result5 <- big.PCA(bmat2,thin=TRUE,keep=.5,how="pca")
result6 <- big.PCA(bmat2,thin=TRUE,keep=.2)
normal <- result2$PCs
thinned <- result3$PCs
corred <- result4$PCs
pced <- result5$PCs
thinner <- result6$PCs
## correlate the resulting PCs with the un-thinned PCs
cors.thin.with.orig <- apply(cor(normal,thinned),1,max)
cors.corred.with.orig <- apply(cor(normal,corred),1,max)
cors.pced.with.orig <- apply(cor(normal,pced),1,max)
cors.thinner.with.orig <-apply(cor(normal,thinner),1,max)
plot(cors.thin.with.orig,type="l",col="red",ylim=c(0,1))
lines(cors.thinner.with.orig,col="orange")
lines(cors.corred.with.orig,col="lightblue")
lines(cors.pced.with.orig,col="lightgreen")
# can see that the first component is highly preserved,
# and next components, somewhat preserved; try using different thinning methods
unlink(c("testMyBig.bck","testMyBig.dsc"))
```

---

`big.select` *Select a subset of a big.matrix*

---

## Description

Select a subset of big.matrix using indexes for a subset of rows and columns. Essentially a wrapper for bigmemory::deepcopy, but with slightly more flexible parameters. bigMat can be entered in any form accepted by get.big.matrix(), row and column selections can be vectors of indexes, names or file.names containing indexes. Default is to process using deepcopy, but processing without using bigmemory native methods is a faster option when matrices are small versus available RAM. File names for backing files are managed only requiring you to enter a prefix, or optionally use the default and gain filebacked functionality without having to bother choosing filename parameters.

## Usage

```
big.select(bigMat, select.rows = NULL, select.cols = NULL, dir = getwd(),
  deepC = TRUE, pref = "sel", delete.existing = FALSE, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix, matrix or any object accepted by get.big.matrix() |
| select.rows | selection of rows of bigMat, can be numbers, logical, rownames, or a file with names. If using a filename argument, must also use a filename argument for select.cols (cannot mix) |
| select.cols | selection of columns of bigMat, can be numbers, logical, colnames, or a file with names |
| dir | the directory containing the bigMat backing file (e.g, parameter for get.big.matrix()). |
| deepC | logical, whether to use bigmemory::deepcopy, which is slowish, but scalable, or alternatively to use standard indexing which converts the result to a regular matrix object, and is fast, but only feasible for matrices small enough to fit in memory. |
| pref | character, prefix for the big.matrix backingfile and descriptorfile, and optionally an R binary file containing a big.matrix.descriptor object pointing to the big.matrix result. |
| verbose | whether to display extra information about processing and progress |
| delete.existing | logical, if a big.matrix already exists with the same name as implied by the current 'pref' and 'dir' arguments, then default behaviour (FALSE) is to return an error. to overwrite any existing big.matrix file(s) of the same name(s), set this parameter to TRUE. |

## Value

A big.matrix with the selected (in order) rows and columns specified

## Author(s)

Nicholas Cooper

## Examples

```
bmat <- generate.test.matrix(5,big.matrix=TRUE)
# take a subset of the big.matrix without using deepcopy
sel <- big.select(bmat,c(1,2,8),c(2:10),deepC=FALSE,verbose=TRUE)
prv.big.matrix(sel)
# now select the same subset using row/column names from text files
writeLines(rownames(bmat)[c(1,2,8)],con="bigrowstemp.txt")
writeLines(colnames(bmat)[c(2:10)],con="bigcolstemp.txt")
sel <- big.select(bmat, "bigrowstemp.txt","bigcolstemp.txt", delete.existing=TRUE)
prv.big.matrix(sel)
unlink(c("bigcolstemp.txt","bigrowstemp.txt","sel.RData","sel.bck","sel.dsc"))
```

big.t *Transpose function for big.matrix objects*

## Description

At the time of writing, there is no transpose method for big.matrix() This function returns a new filebacked big.matrix which is the transpose of the input big.matrix. max.gb allows periodic manual flushing of the memory to be conducted in case the built-in memory management of R/bigmemory is not working as desired. This method is a non-native (not using the raw C objects from the package but merely standard R accessors and operations) algorithm to transpose a big matrix efficiently for memory usage and speed. A blank matrix is created on disk and the data is block-wise transposed and buffered into the new matrix.

## Usage

```
big.t(bigMat, dir = NULL, name = "t.bigMat", R.descr = NULL,
  max.gb = NA, verbose = F, tracker = NA, file.ok = T)
```

## Arguments

| | |
|---|---|
| bigMat | default, a big.matrix(), although if 'file.ok' is set TRUE, then this can be a big.matrix descriptor, or a file location |
| dir | the directory for the matrix backing file (preferably for both the original and the proposed transposed matrix). If this is left NULL and bigMat contains a path, this path (via dirname(bigMat)) will be used; if it doesn't contain a path the current working directory will be used |
| name | the basename of the new transposed matrix |
| R.descr | the name of a binary file that will store the big.matrix.descriptor for the transposed matrix. If "" then the descriptor won't be saved. If NULL, then it will be <name>.RData |
| max.gb | the maximum number of GB of data to process before flushing the big.matrix |
| verbose | whether to print messages about each stage of the process |
| tracker | whether to use a progress bar. NA means it will only be used if the matrix in question is larger than 1GB. |
| file.ok | whether to accept big.matrix.descriptors or filenames as input for 'bigMat'; if T, then anything that works with get.big.matrix(bigMat,dir) is acceptable |

## Value

A big.matrix that is the transpose (rows and columns switched) of the original matrix

## Examples

```
bM <- filebacked.big.matrix(200, 500,
       dimnames = list(paste("r",1:200,sep=""), paste("c",1:500,sep="")),
       backingfile = "test.bck",  backingpath = getwd(), descriptorfile = "test.dsc")
bM[1:200,] <- replicate(500,rnorm(200))
prv.big.matrix(bM)
tbM <- big.t(bM,verbose=TRUE)
prv.big.matrix(tbM)
unlink(c("t.bigMat.RData","t.bigMat.bck","t.bigMat.dsc","test.bck","test.dsc"))
```

---

bmcapply *A multicore 'apply' function for big.matrix objects*

---

## Description

# to put into NCmisc Multicore method to run a function for a big.matrix that could be run using 'apply' on a regular matrix (when parameter use.apply=T [default]). Otherwise for a function that might be more efficient done in done in chunks (e.g, utilising vectorised functions) use.apply=F can be set so that processing is done on larger submatrices, rather than 1 row/column at a time. Input to specify whether to perform the function row or columnwise is equivalent to 'apply' syntax, 1=by-rows, 2=by-columns. This function is useful for big.matrix processing even without multiple cores, particulary when MARGIN=1 (row-wise). While native colmean, colmin and colsd functions for big.matrix objects are very fast (and will probably outperform bmcapply even with 1 core versus many), these are only natively implemented for column-wise operations and the equivalent operations if needing to be row-wise should be faster with bmcapply for matrices larger than available RAM. Can also be used for regular matrices although there is unlikely to be a speed advantage.

## Usage

```
bmcapply(bigMat, MARGIN, FUN, dir = NULL, by = 200, n.cores = 1,
  use.apply = TRUE, convert = !use.apply, combine.fn = NULL, ...)
```

## Arguments

| | |
|---|---|
| bigMat | the big.matrix object to apply the function upon, can enter as a filename, description object or any other valid parameter to get.big.matrix(). Can also use with a standard matrix |
| MARGIN | 1=row-wise, 2=column-wise, see same argument for base:::apply() |
| FUN | the function to apply, should return a result with 1 dimension that has the same length as dim(bigMat)[MARGIN]=L; i.e, a vector length L, matrix (L,x) or (x,L) or list[[L]]. Note that using a custom 'combine.fn' parameter might allow exceptions to this. |
| dir | directory argument for get.big.matrix(), ie. the location of the bigMat backing file if not in the current working directory. |

| | |
|---|---|
| by | integer, the number of rows/columns to process at once. The default should work in most situations however, if the dimension not specified by MARGIN is very large, this might need to be smaller, or if the function being applied is much more efficient performed on a large matrix than several smaller ones then this 'by' parameter should be increased within memory contraints. You should make sure 'estimate.memory(c(by,dim(bigMat)[-MARGIN]))' doesn't exceed available RAM. |
| n.cores | integer, the number of parallel cores to utilise; note that sometimes if a machine has only a few cores this can result in slower performance by tying up resources which should be available to perform background and system operations. |
| use.apply | logical, if TRUE then use the 'apply' function to apply FUN to each submatrix, or if FALSE, then directly apply FUN to submatrices, which means that FUN must return results with at least 1 dimension the same as the input, or you can use a custom 'combine.fn' parameter to recombine results from submatrices. |
| convert | logical, only need to change this parameter when use.apply=FALSE. If use are using a function that can natively run on big.matrix objectsthen you can increase speed by setting convert=FALSE. Most functions will expect a regular matrix and may fail with a big.matrix, so default convert=TRUE behaviour will convert submatrices to a regular matrix just before processing. |
| combine.fn | a custom function to recombine input from sub.matrix processing. Default combine functions are list(), cbind() and rbind(); so a custom function should expect the same input as these; ie., a list of unspecified length, which will be the list of results from parallel calls on submatrices of bigMat, usually of size by*X. |
| ... | if use.apply=TRUE, then additional arguments for apply(); else additional arguments for FUN. |

## Value

Result depends on the function 'FUN' called, and the parameter 'combine.fn', but if MARGIN=1 usually is a vector of length nrow(bigMat), or if MARGIN=2 a vector of length ncol(bigMat).

## See Also

get.big.matrix

## Examples

```
# set up a toy example of a big.matrix (functions most relevant when matrix is huge)
bM <- filebacked.big.matrix(20, 50,
       dimnames = list(paste("r",1:20,sep=""), paste("c",1:50,sep="")),
       backingfile = "test.bck",  backingpath = getwd(), descriptorfile = "test.dsc")
bM[1:20,] <- replicate(50,rnorm(20))
prv.big.matrix(bM)
# compare native bigmemory column-wise function to multicore [native probably faster]
v1 <- colsd(bM) # native bigmemory function
v2 <- bmcapply(bM,2,sd,n.cores=2) # use up to 2 cores if available
print(all.equal(v1,v2))
# compare row-means approaches
v1 <- rowMeans(as.matrix(bM))
```

```
v2 <- bmcapply(bM,1,mean,n.cores=2) # use up to 2 cores if available
v3 <- bmcapply(bM,1,rowMeans,use.apply=FALSE)
print(all.equal(v1,v2)); print(all.equal(v2,v3))
# example using a custom combine function; taking the mean of column means
weight.means.to.scalar <- function(...) { X <- list(...); mean(unlist(X)) }
v1 <- bmcapply(bM, 2, sd, combine.fn=weight.means.to.scalar)
v2 <- mean(colsd(bM))
print(all.equal(v1,v2))
## note that this function works with normal matrices, however, multicore
# operation is only likely to benefit speed when operations take more than 10 seconds
# so this function will mainly help using large matrices or intensive functions
test.size <- 5 # try increasing this number, or use more intensive function than sd()
# to test relative speed for larger matrices
M <- matrix(runif(10^test.size),ncol=10^(test.size-2)) # normal matrix
system.time(bmcapply(M,2,sd,n.cores=2)) # use up to 2 cores if available
system.time(apply(M,2,sd)) #
unlink(c("test.bck","test.dsc"))
```

---

`estimate.eig.vpcs` *Estimate the variance percentages for uncalculated eigenvalues*

---

## Description

If using a function like irlba() to calculate PCA, then you can choose (for speed) to only calculate a subset of the eigenvalues. So there is no exact percentage of variance explained by the PCA, or by each component as you will get as output from other routines. This code uses a linear, or b*1/x model, to estimate the AUC for the unknown eigenvalues, providing a reasonable estimate of the variances accounted for by each unknown eigenvalue, and the predicted eigenvalue sum of the unknown eigenvalues.

## Usage

```
estimate.eig.vpcs(eigenv = NULL, min.dim = length(eigenv), M = NULL,
  elbow = NA, linear = TRUE, estimated = FALSE, print.est = TRUE,
  print.coef = FALSE, add.fit.line = FALSE, col = "blue",
  ignore.warn = FALSE)
```

## Arguments

| | |
|---|---|
| `eigenv` | the vector of eigenvalues actually calculated |
| `min.dim` | the size of the smaller dimension of the matrix submitted to singular value decomposition, e.g, number of samples - i.e, the max number of possible eigenvalues, alternatively use 'M'. |
| `M` | optional enter the original dataset 'M'; simply used to derive the dimensions, alternatively use 'min.dim'. |
| `elbow` | the number of components which you think explain the important portion of the variance of the dataset, so further components are assumed to be reflecting noise or very subtle effects, e.g, often the number of components used is decided by the 'elbow' in a scree plot (see 'pca.scree.plots') |

| | |
|---|---|
| linear | whether to use a linear model to model the 'noise' eigenvalues; alternative is a 1/x model with no intercept. |
| estimated | logical, whether to return the estimated variance percentages for unobserved eigenvalues along with the real data; will also generate a factor describing which values in the returned vector are observed versus estimated. |
| print.est | whether to output the estimate result to the console |
| print.coef | whether to output the estimate regression coefficients to the console |
| add.fit.line | logical, if there is an existing scree plot, adds the fit line from this estimate to the plot ('pca.scree.plots' can use this option using the parameter of the same name) |
| col | colour for the fit line |
| ignore.warn | ignore warnings when an estimate is not required (i.e, all eigenvalues present) |

## Value

By default returns a list where the first element "variance.pcs' are the known variance percentages for each eigenvalue based on the estimated divisor, the second element 'tail.auc' is the area under the curve for the estimated eigenvalues. If estimate =TRUE then a third element is return with separate variance percentages for each of the estimated eigenvalues.

## See Also

pca.scree.plots

## Examples

```
nsamp <- 100; nvar <- 300; subset.size <- 25; elbow <- 6
mat <- matrix(rnorm(nsamp*nvar),ncol=nsamp)
# or use: # mat <- crimtab-rowMeans(crimtab) ; subset.size <- 10 # crimtab centred
prv.large(mat)
pca <- svd(mat,nv=subset.size,nu=0) # calculates subset of V, but all D
require(irlba)
pca2 <- irlba(mat,nv=subset.size,nu=0) # calculates subset of V & D
pca3 <- princomp(mat,cor=TRUE) # calculates all
# number of eigenvalues for svd is the smaller dimension of the matrix
eig.varpc <- estimate.eig.vpcs(pca$d^2,M=mat)$variance.pcs
cat("sum of all eigenvalue-variances=",sum(eig.varpc),"\n")
print(eig.varpc[1:elbow])
# number of eigenvalues for irlba is the size of the subset if < min(dim(M))
eig.varpc <- estimate.eig.vpcs((pca2$d^2)[1:subset.size],M=mat)$variance.pcs
print(eig.varpc[1:elbow])  ## using 1/x model, underestimates total variance
eig.varpc <- estimate.eig.vpcs((pca2$d^2)[1:subset.size],M=mat,linear=TRUE)$variance.pcs
print(eig.varpc[1:elbow])  ## using linear model, closer to exact answer
eig.varpc <- estimate.eig.vpcs((pca3$sdev^2),M=mat)$variance.pcs
print(eig.varpc[1:elbow])  ## different analysis, but fairly similar var.pcs
```

`generate.test.matrix` *Generate a test matrix of random data*

## Description

Generates a test matrix of easily specified size and type. Options allow automated row and column names (which might resemble labels for a SNP analysis) and return of several different formats, matrix, data.frame or big.matrix. You can specify the randomisation function (e.g, rnorm, runif, etc), as well as parameters determining the matrix size. Can also generate big.matrix objects, and an important feature is that the method to generate big.matrix objects is scalable so that very large matrices for simulation can be generated only limited by disk space and not by RAM.

## Usage

```
generate.test.matrix(size = 5, row.exp = 2, rand = rnorm,
  dimnames = TRUE, data.frame = FALSE, big.matrix = FALSE,
  file.name = NULL, tracker = TRUE)
```

## Arguments

| | |
|---|---|
| `size` | 10^size is the total number of datapoints simulated. 6 or less are fairly quick to generate, while 7 takes a few seconds. 8 will take under a minute, 9 around ten minutes, 10, perhaps over an hour. Values are coerced to the range of integers c(2:10). |
| `row.exp` | similar to 'nrow' when creating a matrix, except this is exponential, giving 10^row.exp rows. |
| `rand` | a function, must return 'n' values, when rand(n) is called, eg., rnorm(), runif(), numeric() |
| `dimnames` | logical, whether to generate some row and column names |
| `data.frame` | logical, whether to return as a data.frame (FALSE means return a matrix) |
| `big.matrix` | logical, whether to return as a big.matrix (overrides data.frame). If a file.name is used then the big.matrix will be filebacked and this function returns a list with a a big.matrix, and the description and backing filenames. |
| `file.name` | if a character, then will write the result to tab file instead of returning the object, will return the filename; overrides data.frame. Alternatively, if big.matrix=TRUE, then this provides the basename for a filebacked big.matrix. |
| `tracker` | logical, whether to display a progress bar for large matrices (size>7) where progress will be slow |

## Value

Returns a random matrix of data for testing/simulation, can be a data.frame or big.matrix if those options are selected

## Author(s)

Nicholas Cooper

## Examples

```
mat <- (generate.test.matrix(5)); prv(mat)
lst <- (generate.test.matrix(5,3,big.matrix=TRUE,file.name="bigtest"))
mat <- lst[[1]]; prv(mat); headl(lst[2:3]);
unlink(unlist(lst[2:3]))
```

---

# get.big.matrix *Retrieve a big.matrix object*

---

## Description

This function can load a big.matrix object using a big.matrix.descriptor object, the name of a description file, the name of a binary file containing a big.matrix.descriptor or if passed a big.matrix object, it will just return that object. Only the object or file name plus the directory containing the backing file are required.

## Usage

```
get.big.matrix(fn, dir = "", verbose = FALSE)
```

## Arguments

| | |
|---|---|
| fn | the name of a description file, the name of a binary file containing a big.matrix.descriptor, a big.matrix object or a big.matrix.descriptor object. |
| dir | directory containing the backing file (if not the working directory) |
| verbose | whether to display information on method being used, or minor warnings |

## Value

Returns a big.matrix object, regardless of what method was used as reference/input

## Examples

```
# set up a toy example of a big.matrix
bM <- filebacked.big.matrix(20, 50,
       dimnames = list(paste("r",1:20,sep=""), paste("c",1:50,sep="")),
       backingfile = "test.bck",  backingpath = getwd(), descriptorfile = "test.dsc")
bM[1:20,] <- replicate(50,rnorm(20))
# Now have a big matrix which can be retrieved using this function in 4 ways:
d.bM <- describe(bM)
save(d.bM,file="fn.RData")
bM1 <- get.big.matrix("test.dsc")
bM2 <- get.big.matrix(d.bM)
bM3 <- get.big.matrix("fn.RData")
```

```
bM4 <- get.big.matrix(bM)
prv.big.matrix(bM)
prv.big.matrix(bM1)
prv.big.matrix(bM2)
prv.big.matrix(bM3)
prv.big.matrix(bM4)
unlink(c("fn.RData","test.bck","test.dsc"))
```

---

`import.big.data` *Load a text file into a big.matrix object*

---

## Description

This provides a faster way to import text data into a big.matrix object than bigmemory::read.big.matrix(). The method allows import of a data matrix with size exceeding RAM limits. Can import from a matrix delimited file with or without row/column names, or from a long format dataset with no row/columns names (these should be specified as separate lists).

## Usage

```
import.big.data(input.fn = NULL, dir = getwd(), long = FALSE,
  rows.fn = NULL, cols.fn = NULL, pref = "", delete.existing = TRUE,
  ret.obj = FALSE, verbose = TRUE, row.names = NULL, col.names = NULL,
  dat.type = "double", ram.gb = 2, hd.gb = 1000, tracker = TRUE)
```

## Arguments

| | |
|---|---|
| `input.fn` | character, or list, either a single file name of the data, or a list of multiple file name if the data is stored as multiple files. If multiple, then the corresponding list of row or column names that is unique between files should be a list of the same length. |
| `dir` | character, the directory containing all files. Or, if files are split between directories, then either include the directories explicitly in the filenames, or multiple directories can be entered as a list, with names 'big', 'ano' and 'col', where big is the location for big.matrix objects to file-back to, 'ano' is the location of row and column names, and 'col' is the location of the raw text datafiles. |
| `long` | logical, if TRUE, then the data is assumed to be in long format, where each datapoint is on a new line, and the file is structured so that the data for each case/sample/id is consecutive and ordered consistently between samples. If using long format the file should contain no row or column names, these should be specified in either rows.fn/cols.fn file name arguments, or row.names/col.names vector arguments. If long=FALSE, then the dimensions of the file will be automatically detected; including if the file is in long format, however, if you know the data is in long format, specifying this explicitly will be quicker and guarantees the correct import method. |

| | |
|---|---|
| rows.fn | character, with the name of a text file containing the list of row labels for the dataset. Unnecessary if importing from a matrix with row/column names in the file, or if using the row.names parameter. Must be a list of filenames if row names are split across multiple input.fn files. |
| cols.fn | character, with the name of a text file containing the list of column labels for the dataset. Unnecessary if importing from a matrix with row/column names in the file, or if using the col.names parameter. Must be a list of filenames if column names are split across multiple input.fn files. |
| pref | character, optional prefix to use in naming the big.matrix files (description/backing files) |
| delete.existing | logical, if a big.matrix already exists with the same name as implied by the current 'pref' and 'dir' arguments, then default behaviour (FALSE) is to return an error. to overwrite any existing big.matrix file(s) of the same name(s), set this parameter to TRUE. |
| ret.obj | logical, whether to return a big.matrix.descriptor object (TRUE), or just the file name of the big.matrix description file of the imported dataset. |
| verbose | logical, whether to display extra information about import progress and notifications. |
| row.names | character vector, optional alternative to specifying rows.fn file name(s), directly specify row names as a single vector, or a list of vectors if multiple input files with differing row names are being imported. |
| col.names | character vector, optional alternative to specifying cols.fn file name(s), directly specify oclumn names as a single vector, or a list of vectors if multiple input files with differing column names are being imported. |
| dat.type | character, data type being imported, default is "double", but can specify any type supported by a filebacked.big.matrix(), namely, "integer","char","short"; note these are C-style data types; double=numeric, char=character, integer=integer, short=numeric (although will be stored with less precision in the C-based big.matrix object). |
| ram.gb | numeric, the number of gigabytes of free RAM that it is ok for the import to use. The higher this amount, the quicker the import will be, as flushing RAM contents to the hard drive more regularly slows down the process. Setting this lower will reduce the RAM footprint of the import. Note that if you set it too high, it can't be guaranteed, but usually R and bigmemory will do a reasonable job of managing the memory, and it shouldn't crash your computer. |
| hd.gb | numeric, the amount of free space on your hard disk; if you set this parameter accurately the function will stop if it believes there is insufficient disk space to import the object you have specified. By default this is set to 1 terabyte, so if importing an object larger than that, you will have to increase this parameter to make it work. |
| tracker | logical, whether to display a progress bar for the importing process |

## Value

Returns a big.matrix containing the data imported (single big.matrix even when text input is split across multiple files)

## Examples

```
# Collate all file names to use in this example #
all.fn <- c("rownames.txt","colnames.txt","functestdn.txt","funclongcol.txt","functest.txt",
 paste("rn",1:3,".txt",sep=""),paste("cn",1:3,".txt",sep=""),
 paste("split",1:3,".txt",sep=""),
 paste("splitmatCd",1:3,".txt",sep=""),paste("splitmatRd",1:3,".txt",sep=""),
 paste("splitmatC",1:3,".txt",sep=""), paste("splitmatR",1:3,".txt",sep=""))
any.already <- file.exists(all.fn)
if(any(any.already)) {
 warning("files already exist in the working directory with the same names as some example files") }
# SETUP a test matrix and reference files #
test.size <- 4 # try increasing this number for larger matrices
M <- matrix(runif(10^test.size),ncol=10^(test.size-2)) # normal matrix
write.table(M,sep="\t",col.names=FALSE,row.names=FALSE,
 file="functest.txt",quote=FALSE) # no dimnames
rown <- paste("rs",sample(10:99,nrow(M),replace=TRUE),sample(10000:99999,nrow(M)),sep="")
coln <- paste("ID",sample(1:9,ncol(M),replace=TRUE),sample(10000:99999,ncol(M)),sep="")
r.fn <- "rownames.txt"; c.fn <- "colnames.txt"
Mdn <- M; colnames(Mdn) <- coln; rownames(Mdn) <- rown
# with dimnames
write.table(Mdn,sep="\t",col.names=TRUE,row.names=TRUE,file="functestdn.txt",quote=FALSE)
prv.large(Mdn)
writeLines(paste(as.vector(M)),con="funclongcol.txt")
in.fn <- "functest.txt"

### IMPORTING SIMPLE 1 FILE MATRIX ##
writeLines(rown,r.fn); writeLines(coln,c.fn)
#1. import without specifying row/column names
ii <- import.big.data(in.fn); prv.big.matrix(ii) # SLOWER without dimnames!
#2. import using row/col names from file
ii <- import.big.data(in.fn,cols.fn="colnames.txt",rows.fn="rownames.txt")
prv.big.matrix(ii)
#3. import by passing colnames/rownames as objects
ii <- import.big.data(in.fn, col.names=coln,row.names=rown)
prv.big.matrix(ii)

### IMPORTING SIMPLE 1 FILE MATRIX WITH DIMNAMES ##
#1. import without specifying row/column names, but they ARE in the file
in.fn <- "functestdn.txt"
ii <- import.big.data(in.fn); prv.big.matrix(ii)

### IMPORTING SIMPLE 1 FILE MATRIX WITH MISORDERED rownames ##
rown2 <- rown; rown <- sample(rown);
# re-run test3 using in.fn with dimnames
ii <- import.big.data(in.fn, col.names=coln,row.names=rown)
prv.big.matrix(ii)
# restore rownames:
rown <- rown2

### IMPORTING SIMPLE 1 FILE LONG FORMAT by columns ##
in.fn <- "funclongcol.txt"; #rerun test 2 #
ii <- import.big.data(in.fn,cols.fn="colnames.txt",rows.fn="rownames.txt")
```

```
prv.big.matrix(ii)

### IMPORTING multifile LONG by cols ##
# create the dataset and references
splF <- factor(rep(c(1:3),ncol(M)*c(.1,.5,.4)))
colnL <- split(coln,splF); MM <- as.data.frame(t(M))
Ms2 <- split(MM,splF)
Ms2 <- lapply(Ms2,
   function(X) { X <- t(X); dim(X) <- c(nrow(M),length(X)/nrow(M)); X } )
# preview Ms2 - not run # lapply(Ms2,prv.large)
colfs <- paste("cn",1:length(colnL),".txt",sep="")
infs <- paste("split",1:length(colnL),".txt",sep="")
# create multiple column name files and input files
for(cc in 1:length(colnL)) { writeLines(colnL[[cc]],con=colfs[cc]) }
for(cc in 1:length(infs)) {
  writeLines(paste(as.vector((Ms2[[cc]]))),con=infs[cc]) }

# Now test the import using colnames and rownames lists
ii <- import.big.data(infs, col.names=colnL,row.names=rown)
prv.big.matrix(ii)

### IMPORTING multifile MATRIX by rows ##
# create the dataset and references
splF <- factor(rep(c(1,2,3),nrow(M)*c(.1,.5,.4)))
rownL <- split(rown,splF)
Ms <- split(M,splF)
Ms <- lapply(Ms,function(X) { dim(X) <- c(length(X)/ncol(M),ncol(M)); X } )
# preview Ms - not run # lapply(Ms,prv.large)
# create multiple row name files and input files
rowfs <- paste("rn",1:length(rownL),".txt",sep="")
for(cc in 1:length(rownL)) { writeLines(rownL[[cc]],con=rowfs[cc]) }
infs <- paste("splitmatR",1:length(colnL),".txt",sep="")
for(cc in 1:length(infs)) {
 write.table(Ms[[cc]],sep="\t",col.names=FALSE,row.names=FALSE,file=infs[cc],quote=FALSE) }

# Now test the import using colnames and rownames files
ii <- import.big.data(infs, col.names="colnames.txt",rows.fn=rowfs)
prv.big.matrix(ii)

# DELETE ALL FILES ##
unlink(all.fn[!any.already]) # prevent deleting users files
## many files to clean up! ##
unlink(c("funclongcol.bck","funclongcol.dsc","functest.bck","functest.dsc",
 "functestdn.RData","functestdn.bck","functestdn.dsc","functestdn_file_rowname_list_check_this.txt",
 "split1.bck","split1.dsc","splitmatR1.bck","splitmatR1.dsc"))
```

---

PC.correct *Correct a big.matrix by principle components*

---

## Description

Principle components (PC) can be used as a way of capturing bias (when common variance represents bias) and so PC correction is a way to remove such bias from a dataset. Using the first 'n' PCs from an an analysis performed using big.pca(), this function will transform the original matrix by regressing onto the 'n' principle components (and optionally gender) and returing the residuals. The result is returned as a big.matrix object, so that objects larger than available RAM can be processed, and multiple processors can be utilised for greater speed for large datasets.

## Usage

```
PC.correct(pca.result, bigMat, dir = getwd(), num.pcs = 9, n.cores = 1,
  pref = "corrected", big.cor.fn = NULL, write = FALSE,
  sample.info = NULL, correct.sex = FALSE, add.int = FALSE,
  preserve.median = FALSE, tracker = TRUE, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| pca.result | result returned by 'big.pca()', or a list with 2 elements containing the principle components and the eigenvalues respectively (or SVD equivalents). Alternatively, can be the name of an R binary file containing such an object. |
| bigMat | a big.matrix with exactly corresponding samples (columns) to those submitted to PCA prior to correction |
| dir | directory containing the big.matrix backing file |
| num.pcs | number of principle components (or SVD components) to correct for |
| n.cores | number of cores to use in parallel for processing |
| pref | prefix to add to the file name of the resulting corrected matrix backing file |
| big.cor.fn | instead of using 'pref' directly specify the desired file name |
| write | whether to write the result to a file.backed big.matrix or to simply return a pointer to the resulting corrected big.matrix |
| sample.info | if using 'correct.sex=TRUE' then this object should be a dataframe containing the sex of each sample, with sample names as rownames |
| correct.sex | if sample.info is a dataframe containing a column named 'gender' or 'sex' (case insensitive), then add a sex covariate to the PC correction linear model |
| add.int | logical, whether to maintain the pre-corrected means of each variable, i.e, post-correction add the mean back onto the residuals which will otherwise have mean zero for each variable. |
| preserve.median | logical, if add.int=TRUE, then setting this parameter to TRUE will preserve the median of the original data, instead of the mean. This is because after PC-correction the skew may change. |
| tracker | logical, whether to display a progress bar |
| verbose | logical, whether to display preview of pre- and post- corrected matrix |

## Value

A big.matrix of the same dimensions as original, corrected for n PCs and an optional covariate (sex)

### Author(s)

Nicholas Cooper

### See Also

big.pca

### Examples

```
mat2 <- sim.cor(500,200,genr=function(n){ (runif(n)/2+.5) })
bmat2 <- as.big.matrix(mat2,backingfile="testMyBig.bck",descriptorfile="testMyBig.dsc")
## calculate PCA ##
# result2 <- big.PCA(bmat2,thin=FALSE)
# corrected <- PC.correct(result2,bmat2)
# corrected2 <- PC.correct(result2,bmat2,n.cores=2)
# c1 <- get.big.matrix(corrected) ; c2 <- get.big.matrix(corrected2)
# all.equal(as.matrix(c1),as.matrix(c2))
unlink(c("testMyBig.bck","testMyBig.dsc"))
```

---

## pca.scree.plot *Make scree plots for any PCA*

---

### Description

Make a scree plot using eigenvalues from princomp(), prcomp(), svd(), irlba(), big.pca(), etc. Note that most these return values which need to be squared to be proper eigenvalues. There is also an option to use the estimate.eig.vpcs() function to estimate any missing eigenvalues (e.g, if using a function like irlba' to calculate PCA) and then to visualise the fitline of the estimate on the scree plot.

### Usage

```
pca.scree.plot(eigenv, elbow = NA, printvar = TRUE, min.dim = NA,
  M = NULL, add.fit.line = FALSE, n.xax = max(30, length(eigenv)),
  linear = TRUE, verbose = FALSE, return.data = FALSE, ...)
```

### Arguments

| | |
|---|---|
| eigenv | the vector of eigenvalues actually calculated |
| elbow | the number of components which you think explain the important chunk of the variance of the dataset, so further components are modelled as reflecting noise or very subtle effects, e.g, often the number of components used is decided by the 'elbow' in a scree plot (see 'pca.scree.plots') |
| min.dim | the size of the smaller dimension of the matrix submitted to singular value decomposition, e.g, number of samples - i.e, the max number of possible eigenvalues, alternatively use 'M'. |

| | |
|---|---|
| M | optional enter the original dataset 'M'; simply used to derive the dimensions, alternatively use 'min.dim'. |
| linear | whether to use a linear model to model the 'noise' eigenvalues; alternative is a 1/x model with no intercept. |
| printvar | logical, whether to print summary of variance calculations |
| add.fit.line | logical, if there is an existing scree plot, adds the fit line from this estimate to the plot ('pca.scree.plots' can use this option using the parameter of the same name) |
| n.xax | number of components to include on the x-axis |
| verbose | logical, whether to display additional output |
| return.data | logical, whether to return the percentages of variance explained for each component, or nothing (just plot) |
| ... | further arguments to the plot function |

## Value

Either a vector of variance percentages explained, or nothing (just a plot), depending on value of 'return.data'

## See Also

pca.scree.plots

## Examples

```
require(irlba)
nsamp <- 100; nvar <- 300; subset.size <- 25; elbow <- 6
mat <- matrix(rnorm(nsamp*nvar),ncol=nsamp)
#this gives the full solution
pca <- svd(mat,nv=subset.size,nu=0)
pca2 <- irlba(mat,nv=subset.size,nu=0)
# show alternate fits for linear versus 1/x fit
pca.scree.plot((pca2$d^2)[1:subset.size],n.xax=100,add.fit.line=TRUE,
               min.dim=min(dim(mat)),linear=TRUE, elbow=6, ylim=c(0,1400))
pca.scree.plot((pca2$d^2)[1:subset.size],n.xax=100,add.fit.line=TRUE,
              min.dim=min(dim(mat)),linear=FALSE, elbow=40, ylim=c(0,1400))
subset.size <- 75
pca2 <- irlba(mat,nv=subset.size,nu=0)
pca.scree.plot((pca2$d^2)[1:subset.size],n.xax=100,add.fit.line=TRUE,
              min.dim=min(dim(mat)),linear=TRUE, elbow=6, ylim=c(0,1400))
pca.scree.plot((pca2$d^2)[1:subset.size],n.xax=100,add.fit.line=TRUE,
              min.dim=min(dim(mat)),linear=FALSE, elbow=40, ylim=c(0,1400))
```

`prv.big.matrix` *Tidier display function for big matrix objects*

## Description

This function prints the first and last columns and rows of a big matrix, and a few more than this if desired. Allows previewing of a big.matrix without overloading the console.

## Usage

```
prv.big.matrix(bigMat, dir = "", rows = 3, cols = 2, name = NULL,
  dat = TRUE, descr = NULL, bck = NULL, mem = FALSE, rcap = "",
  ccap = "", ...)
```

## Arguments

| | |
|---|---|
| `bigMat` | the description file, big.matrix object, or big.matrix.descriptor object, anything that can be read by get.big.matrix() |
| `dir` | the directory containing the big.matrix backing/description files |
| `name` | logical, whether to print a name for the matrix |
| `dat` | logical, whether to print any of the matrix contents (overrides row/col) |
| `descr` | character, optional name of the description file, which if not null will be displayed |
| `bck` | character, optional name of the backing file, which if not null will be displayed |
| `mem` | logical, whether to display the amount of memory used by the object |
| `rows` | integer, number of rows to display |
| `cols` | integer, number of columns to display |
| `rcap` | character, caption to display for the rows |
| `ccap` | character, caption to display for the columns |
| `...` | additional arguments to prv.large (from NCmisc) which displays the end result |

## Value

Prints to console a compact representation of the bigMat matrix, with the first few rows and columns, and the last row and column. Note that sometimes the initial printing of a big.matrix can take a little while. But subsequently the printout should be almost instantaneous.

## See Also

get.big.matrix()

### Examples

```
bM <- filebacked.big.matrix(20, 50,
       dimnames = list(paste("r",1:20,sep=""), paste("c",1:50,sep="")),
       backingfile = "test.bck",  backingpath = getwd(), descriptorfile = "test.dsc")
bM[1:20,] <- replicate(50,rnorm(20))
prv.big.matrix(bM)
prv.big.matrix(bM,rows=10,cols=4)
unlink(c("test.dsc","test.bck"))  # clean up files
```

---

## quick.elbow *Quickly estimate the 'elbow' of a scree plot (PCA)*

---

### Description

This function uses a rough algorithm to estimate a sensible 'elbow' to choose for a PCA scree plot of eigenvalues. The function looks at an initial arbitrarily 'low' level of variance and looks for the first eigenvalue lower than this. If the very first eigenvalue is actually lower than this (i.e, when the PCs are not very explanatory) then this 'low' value is iteratively halved until this is no longer the case. After starting below this arbitrary threshold the drop in variance explained by each pair of consecutive PCs is standardized by dividing over the larger of the pair. The largest percentage drop in the series below 'low'

### Usage

```
quick.elbow(varpc, low = 0.08, max.pc = 0.9)
```

### Arguments

| | |
|---|---|
| varpc | numeric, vector of eigenvalues, or 'percentage of variance' explained datapoints for each principle component. If only using a partial set of components, should first pass to estimate.eig.vpcs() to estimate any missing eigenvalues. |
| low | numeric, between zero and one, the threshold to define that a principle component does not explain much 'of the variance'. |
| max.pc | maximum percentage of the variance to capture before the elbow (cumulative sum to PC 'n') |

### Value

The number of last principle component to keep, prior to the determined elbow cutoff

### Author(s)

Nicholas Cooper

### See Also

estimate.eig.vpcs

## Examples

```
# correlated data
mat <- sim.cor(100,50)
result <- princomp(mat)
eig <- result$sdev^2
elb.a <- quick.elbow(eig)
pca.scree.plot(eig,elbow=elb.a,M=mat)
elb.b <- quick.elbow(eig,low=.05) # decrease low to select more components
pca.scree.plot(eig,elbow=elb.b,M=mat)
# random (largely independent) data, usually higher elbow #
mat2 <- generate.test.matrix(5,3)
result2 <- princomp(mat2)
eig2 <- result2$sdev^2
elb2 <- quick.elbow(result2$sdev^2)
pca.scree.plot(eig2,elbow=elb2,M=mat2)
```

---

# quick.pheno.assocs *Quick association tests for phenotype*

---

## Description

Simplistic association tests, only meant for purposes of preliminary variable selection or creation of priors, etc. Quickly obtain association p-values for a big.matrix against a list of phenotypes for each row, where columns are samples and column labels correspond to the rownames of the sample.info dataframe which contains the phenotype information, in a column labelled 'use.col'.

## Usage

```
quick.pheno.assocs(bigMat, sample.info = NULL, use.col = "phenotype",
  dir = "", p.values = TRUE, F.values = TRUE, n.cores = 1,
  verbose = FALSE)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix object, or any argument accepted by get.big.matrix(), which includes paths to description files or even a standard matrix object. |
| dir | directory containing the filebacked.big.matrix, same as dir for get.big.matrix. |
| sample.info | a data.frame with rownames corresponding to colnames of the bigMat. Must also contain a column named 'use.col' (default 'phenotype') which contains the categorical variable to perform the association test for phenotype, etc. This file may contain extra ids not in colnames(bigMat), although if any column names of bigMat are missing from sample.info a warning will be given, and the call is likely to give incorrect results. |
| use.col | the name of the phenotype column in the data.frame 'sample.info' |
| p.values | logical, whether to return p.values from the associations |
| F.values | logical, whether to return F.values from the associations |

| | |
|---|---|
| n.cores | integer, if wanting to process the analysis using multiple cores, specify the number |
| verbose | logical, whether to display additional output on progress |

## Value

Depending on options selected returns either a list of F values and p values, or just F, or just p-values for association with each variable in the big.matrix.

If both F.values and p.values are TRUE, returns dataframe of both statistics for each variable, else a vector. If the phenotype has 20 more or more unique categories, it will be assumed to be continuous and the association test applied will be correlation. If there are two categories a t-test will be used, and 3 to 19 categories, an ANOVA# will be used. Regardless of the analysis function, output will be converted to an F statistic and/or associated p-values. Except if p.values and F.values are both set to false and the phenotype is continuous, then pearsons correlation values will be returned

## Author(s)

Nicholas Cooper

## See Also

get.big.matrix

## Examples

```
bmat <- generate.test.matrix(5,big.matrix=TRUE)
pheno <- rep(1,ncol(bmat)); pheno[which(runif(ncol(bmat))<.5)] <- 2
ids <- colnames(bmat); samp.inf <- data.frame(phenotype=pheno); rownames(samp.inf) <- ids
both <- quick.pheno.assocs(bmat,samp.inf); prv(both)
Fs <- quick.pheno.assocs(bmat,samp.inf,verbose=TRUE,p.values=FALSE); prv(Fs)
Ps <- quick.pheno.assocs(bmat,samp.inf,F.values=FALSE); prv(Ps)
```

---

# select.least.assoc *Select subset of rows least associated with a categorical variable*

---

## Description

Runs a quick association analysis on the dataset against a phenotype/categorical variable stored in a dataframe, and uses the results as a way to select a subset of the original matrix, so you may wish to select the 'N' least associated variables, or the 'N' most associated.

## Usage

```
select.least.assoc(bigMat, keep = 0.05, phenotype = NULL, least = TRUE,
  dir = "", n.cores = 1, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix object, or any argument accepted by get.big.matrix(), which includes paths to description files or even a standard matrix object. |
| keep | numeric, by default a proportion (decimal) of the original number of rows/columns to choose for the subset. Otherwise if an integer>2 then will assume this is the size of the desired subset, e.g, for a dataset with 10,000 rows where you want a subset size of 1,000 you could set 'keep' as either 0.1 or 1000. |
| dir | directory containing the filebacked.big.matrix, same as dir for get.big.matrix. |
| phenotype | a vector which contains the categorical variable to perform an association test for phenotype, etc. This should be the same length as the number of columns (e.g, samples) in bigMat. |
| least | logical, whether to select TRUE, the top least associated variables, or FALSE, the most associated. |
| n.cores | integer, if wanting to process the analysis using multiple cores, specify the number |
| verbose | logical, whether to display additional output |

## Value

A set of row or column indexes (depents on 'rows' parameter) of the variables most dependent (or indepent) variables measured by association with a [continuous/categorical] phenotype.

## Author(s)

Nicholas Cooper

## See Also

quick.pheno.assocs

## Examples

```
bmat <- generate.test.matrix(5,big.matrix=TRUE)
pheno <- rep(1,ncol(bmat)); pheno[which(runif(ncol(bmat))<.5)] <- 2
most.correl <- select.least.assoc(bmat,phenotype=pheno,least=FALSE)
least.correl <- select.least.assoc(bmat,phenotype=pheno,least=TRUE)
cor(bmat[least.correl,][1,],pheno)  # least correlated
cor(bmat[most.correl,][1,],pheno)  # most correlated
```

subcor.select *Selection of the most correlated variable subset*

## Description

Returns a subset (size='keep') of row or column numbers that are most correlated to other variables in the dataset (or if hi.cor=F), then those that are least correlated. This function performs cor() on a small subset of columns and all rows (when rows=TRUE, or vice -versa when rows=FALSE), and selects rows (rows=TRUE) with greatest/least absolute sum of correlations.

## Usage

```
subcor.select(bigMat, keep = 0.05, rows = TRUE, hi.cor = TRUE,
  dir = getwd(), random = TRUE, ram.gb = 0.1)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix, matrix or any object accepted by get.big.matrix() |
| keep | numeric, by default a proportion (decimal) of the original number of rows/columns to choose for the subset. Otherwise if an integer>2 then will assume this is the size of the desired subset, e.g, for a dataset with 10,000 rows where you want a subset size of 1,000 you could set 'keep' as either 0.1 or 1000. |
| rows | logical, whether the subset should be of the rows of bigMat. If rows=FALSE, then the subset is chosen from columns, would be equivalent to calling subpc.select(t(bigMat)), but avoids actually performing the transpose which can save time for large matrices. |
| hi.cor | logical, whether to choose the most correlated (TRUE) or least correlated subset (FALSE). |
| dir | the directory containing the bigMat backing file (e.g, parameter for get.big.matrix()). |
| random | logical, passed to uniform.select(), whether to take a random or uniform selection of columns (or rows if rows=FALSE) to run the subset PCA. |
| ram.gb | maximum size of the matrix in gigabytes for the subset PCA, 0.1GB is the default which should result in minimal processing time on a typical system. Increasing this increases the processing time, but also the representativeness of the subset chosen. Note that some very large matrices will not be able to be processed by this function unless this parameter is increased; basically if the dimension being thinned is more than 5 this memory limit (see estimate.memory() from NCmisc). |

## Value

A set of row or column indexes (depents on 'rows' parameter) of the most inter-correlated (or least) variables in the matrix.

### Author(s)

Nicholas Cooper

### See Also

thin, uniform.select, get.big.matrix

### Examples

```
mat <- matrix(rnorm(200*2000),ncol=200)
bmat <- as.big.matrix(mat)
ii1 <- subcor.select(bmat,.05,rows=TRUE) # thin down to 5% of the rows
ii2 <- subcor.select(bmat,45,rows=FALSE) # thin down to 45 columns
prv(ii1,ii2)
# show that rows=T is equivalent to rows=F of the transpose (random must be FALSE)
ii1 <- subcor.select(mat,.4,rows=TRUE,random=FALSE)
ii2 <- subcor.select(t(mat),.4,rows=FALSE,random=FALSE)
print(all.equal(ii1,ii2))
```

---

## subpc.select *Selection of a representative variable subset*

---

### Description

Returns a subset (size='keep') of row or column numbers that are most representative of a dataset. This function performs PCA on a small subset of columns and all rows (when rows=TRUE, or vice -versa when rows=FALSE), and selects rows (rows=TRUE) most correlated to the first 'n' principle components, where 'n' is chosen by the function quick.elbow(). The number of variables selected corresponding to each component is weighted according to how much of the variance is explained by each component.

### Usage

```
subpc.select(bigMat, keep = 0.05, rows = TRUE, dir = getwd(),
  random = TRUE, ram.gb = 0.1, ...)
```

### Arguments

| | |
|---|---|
| bigMat | a big.matrix, matrix or any object accepted by get.big.matrix() |
| keep | numeric, by default a proportion (decimal) of the original number of rows/columns to choose for the subset. Otherwise if an integer>2 then will assume this is the size of the desired subset, e.g, for a dataset with 10,000 rows where you want a subset size of 1,000 you could set 'keep' as either 0.1 or 1000. |
| rows | logical, whether the subset should be of the rows of bigMat. If rows=FALSE, then the subset is chosen from columns, would be equivalent to calling subpc.select(t(bigMat)), but avoids actually performing the transpose which can save time for large matrices. |

| | |
|---|---|
| dir | the directory containing the bigMat backing file (e.g, parameter for get.big.matrix()). |
| random | logical, passed to uniform.select(), whether to take a random or uniform selection of columns (or rows if rows=F) to run the subset PCA. |
| ram.gb | maximum size of the matrix in gigabytes for the subset PCA, 0.1GB is the default which should result in minimal processing time on a typical system. Increasing this increases the processing time, but also the representativeness of the subset chosen. Note that some very large matrices will not be able to be processed by this function unless this parameter is increased; basically if the dimension being thinned is more than 5 this memory limit (see estimate.memory() from NCmisc). |
| ... | further parameters to pass to big.PCA() which performs the subset PCA used to determine the most representative rows (or columns). |

## Value

A set of row or column indexes (depents on 'rows' parameter) of the most representative variables in the matrix, as defined by most correlated to principle components

## Author(s)

Nicholas Cooper

## See Also

thin, uniform.select, big.PCA, get.big.matrix

## Examples

```
mat <- matrix(rnorm(200*2000),ncol=200) # normal matrix
bmat <- as.big.matrix(mat)              # big matrix
ii <- subpc.select(bmat,.05,rows=TRUE) # thin down to 5% of the rows
ii <- subpc.select(bmat,45,rows=FALSE) # thin down to 45 columns
# show that rows=T is equivalent to rows=F of the transpose (random must be FALSE)
ii1 <- subpc.select(mat,.4,rows=TRUE,random=FALSE)
ii2 <- subpc.select(t(mat),.4,rows=FALSE,random=FALSE)
print(all.equal(ii1,ii2))
```

---

# svn.bigalgebra.install

*Attempt to install the bigalgebra package using SVN*

---

## Description

The bigalgebra package for efficient algebraic operations on big.matrix objects has now been submitted to CRAN, so this function is now mostly redundant. It used to require installation from SVN and some tinkering, such as changing the description file to add the dependency, and linking 'BH' to allow the package to work. This may still be required on older versions of R that do not support the bigalgebra package uploaded to CRAN, but I cannot confirm this. This function automatically performs these corrections. First, it attempts to check-out the latest version of bigalgebra from SVN version management system and then corrects the description file, then tries to install the package. Note you must also have 'BLAS' installed on your system to utilise this package effectively. PCA functions in the present package are better with bigalgebra installed, but will still run without it. For more information on installation alternatives, type big.algebra.install.help(). Returns TRUE if bigalgebra is already installed.

## Usage

```
svn.bigalgebra.install(verbose = FALSE)
```

## Arguments

| | |
|---|---|
| verbose | whether to report on installation progress/steps |

## Value

If SVN is installed on your system, along with BLAS, this function should install the bigalgebra package, else it will return instructions on what to do to fix the issue

## See Also

big.algebra.install.help

## Examples

```
# not run # svn.bigalgebra.install(TRUE)
```

---

# thin *Reduce one dimension of a large matrix in a strategic way*

---

## Description

Thin the rows (or columns) of a large matrix or big.matrix in order to reduce the size of the dataset while retaining important information. Percentage of the original size or a new number of rows/columns is selectable, and then there are four methods to choose the data subset. Simple uniform and random selection can be specified. Other methods look at the correlation structure of a subset of the data to derive non-arbitrary selections, using correlation, PCA, or association with a phenotype or some other categorical variable. Each of the four methods has a separate function in this package, which you can see for more information, this function is merely a wrapper to select one of the four.

## Usage

```
thin(bigMat, keep = 0.05, how = c("uniform", "correlation", "pca",
  "association"), dir = "", rows = TRUE, random = TRUE, hi.cor = TRUE,
  least = TRUE, pref = "thin", verbose = FALSE, ret.obj = TRUE, ...)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix object, or any argument accepted by get.big.matrix(), which includes paths to description files or even a standard matrix object. |
| keep | numeric, by default a proportion (decimal) of the original number of rows/columns to choose for the subset. Otherwise if an integer>2 then will assume this is the size of the desired subset, e.g, for a dataset with 10,000 rows where you want a subset size of 1,000 you could set 'keep' as either 0.1 or 1000. |
| how | character, only the first two characters are required and they are not case sensitive, select what method to use to perform subset selection, options are: 'uniform': evenly spaced selection when random=FALSE, or random selection otherwise; see uniform.select(). 'correlation': most correlated subset when hi.cor=TRUE, least correlated otherwise; see subcor.select(). 'pca': most representative variables of the principle components of a subset; see subpc.select(). 'association': most correlated subset with phenotype if least=FALSE, or least correlated otherwise; see select.least.assoc(). |
| dir | directory containing the filebacked.big.matrix, same as 'dir' for get.big.matrix. |
| rows | logical, whether to choose a subset of rows (TRUE), or columns (FALSE). rows is always TRUE when using 'association' methods. |
| random | logical, whether to use random selections and subsets (TRUE), or whether to use uniform selections that should give the same result each time for the same dataset (FALSE) |
| hi.cor | logical, if using 'correlation' methods, then whether to choose the most correlated (TRUE) or least correlated (FALSE). |
| least | logical, if using 'association' methods, whether to choose the least associated (TRUE) or most associated variables with phenotype |
| pref | character, a prefix for big.matrix backing files generated by this selection |
| verbose | logical, whether to display more information about processing |
| ret.obj | logical, whether to return the result as a big.matrix object (TRUE), or as a reference to the binary file containing the big.matrix.descriptor object [either can be read with get.big.matrix() or prv.big.matrix()] |
| ... | other arguments to be passed to uniform.select, subpc.select, subcor.select, or select.least.assoc |

## Value

A smaller big.matrix with fewer rows and/or columns than the original matrix

## Author(s)

Nicholas Cooper

## See Also

uniform.select, subpc.select, subcor.select, select.least.assoc, big.select, get.big.matrix

## Examples

```
bmat <- generate.test.matrix(5,big.matrix=TRUE)
prv.big.matrix(bmat)
# make 5% random selection:
lmat <- thin(bmat)
prv.big.matrix(lmat)
# make 10% most orthogonal selection (lowest correlations):
lmat <- thin(bmat,.10,"cor",hi.cor=FALSE)
prv.big.matrix(lmat)
# make 10% most representative selection:
lmat <- thin(bmat,.10,"PCA",ret.obj=FALSE) # return file name instead of object
print(lmat)
prv.big.matrix(lmat)
# make 25% selection most correlated to phenotype
# create random phenotype variable
pheno <- rep(1,ncol(bmat)); pheno[which(runif(ncol(bmat))<.5)] <- 2
lmat <- thin(bmat,.25,"assoc",phenotype=pheno,least=FALSE,verbose=TRUE)
prv.big.matrix(lmat)
# tidy up temporary files:
unlink(c("thin.bck","thin.dsc","thin.RData"))
```

---

# uniform.select *Derive a subset of a large dataset*

---

## Description

Either randomly or uniformly select rows or columns from a large dataset to form a new smaller dataset.

## Usage

```
uniform.select(bigMat, keep = 0.05, rows = TRUE, dir = "",
  random = TRUE, ram.gb = 0.1)
```

## Arguments

| | |
|---|---|
| bigMat | a big.matrix object, or any argument accepted by get.big.matrix(), which includes paths to description files or even a standard matrix object. |
| keep | numeric, by default a proportion (decimal) of the original number of rows/columns to choose for the subset. Otherwise if an integer>2 then will assume this is the size of the desired subset, e.g, for a dataset with 10,000 rows where you want a subset size of 1,000 you could set 'keep' as either 0.1 or 1000. |
| dir | directory containing the filebacked.big.matrix, same as dir for get.big.matrix. |

| | |
|---|---|
| rows | logical, whether the subset should be of the rows of bigMat. If rows=FALSE, then the subset is chosen from columns, would be equivalent to calling subpc.select(t(bigMat)), but avoids actually performing the transpose which can save time for large matrices. |
| random | logical, passed to uniform.select(), whether to take a random or uniform selection of columns (or rows if rows=FALSE) to run the subset PCA. |
| ram.gb | maximum size of the matrix in gigabytes for the subset PCA, 0.1GB is the default which should result in minimal processing time on a typical system. Increasing this increases the processing time, but also the representativeness of the subset chosen. Note that some very large matrices will not be able to be processed by this function unless this parameter is increased; basically if the dimension being thinned is more than 5 this memory limit (see estimate.memory() from NCmisc). |

## Value

A set of row or column indexes (depents on 'rows' parameter) of uniformly distributed (optionally reproduceable) or randomly selected variables in the matrix.

## Author(s)

Nicholas Cooper

## See Also

subpc.select

## Examples

```
mat <- matrix(rnorm(200*100),ncol=200)  # standard matrix
bmat <- as.big.matrix(mat)              # big.matrix
ii1 <- uniform.select(bmat,.05,rows=TRUE) # thin down to 5% of the rows
ii2 <- uniform.select(bmat,45,rows=FALSE,random=TRUE) # thin down to 45 columns
prv(ii1,ii2)
```

# Index