

Package ‘corpcor’

March 10, 2010

Version 1.5.6

Date 2010-03-10

Title Efficient Estimation of Covariance and (Partial) Correlation

Author Juliane Schaefer, Rainer Opgen-Rhein, and Korbinian Strimmer.

Maintainer Korbinian Strimmer <strimmer@uni-leipzig.de>

Depends R (>= 2.7.0)

Suggests

Description This package implements a James-Stein-type shrinkage estimator for the covariance matrix, with separate shrinkage for variances and correlations. The details of the method are explained in Schaefer and Strimmer (2005) and Opgen-Rhein and Strimmer (2007). The approach is both computationally as well as statistically very efficient, it is applicable to "small n, large p" data, and always returns a positive definite and well-conditioned covariance matrix. In addition to inferring the covariance matrix the package also provides shrinkage estimators for partial correlations, partial variances, and regression coefficients. The inverse of the covariance and correlation matrix can be efficiently computed, and as well as any arbitrary power of the shrinkage correlation matrix. Furthermore, functions are available for fast singular value decomposition, for computing the pseudoinverse, and for checking the rank and positive definiteness of a matrix.

License GPL (>= 3)

URL <http://strimmerlab.org/software/corpcor/>

Repository CRAN

Date/Publication 2010-03-10 15:00:35

R topics documented:

corpcor-package	2
cor2pcor	3
cov.shrink	4

fast.svd	7
invcov.shrink	9
mpower	11
mvr.shrink	12
pcor.shrink	15
powcor.shrink	17
pseudoinverse	19
rank.condition	21
rebuild.cov	22
smtools	24
wt.scale	25
Index	27

corpcor-package *The corpcor Package*

Description

This package implements a James-Stein-type shrinkage estimator for the covariance matrix, with separate shrinkage for variances and correlations. The details of the method are explained in Sch\"afer and Strimmer (2005) and Opgen-Rhein and Strimmer (2007). The approach is both computationally as well as statistically very efficient, it is applicable to “small n, large p” data, and always returns a positive definite and well-conditioned covariance matrix. In addition to inferring the covariance matrix the package also provides shrinkage estimators for partial correlations, partial variances, and regression coefficients. The inverse of the covariance and correlation matrix can be efficiently computed, and as well as any arbitrary power of the shrinkage correlation matrix. Furthermore, functions are available for fast singular value decomposition, for computing the pseudoinverse, and for checking the rank and positive definiteness of a matrix.

The name of the package refers to **cor**relations and **partial cor**relations.

Author(s)

Juliane Sch\"afer, Rainer Opgen-Rhein, and Korbinian Strimmer (<http://strimmerlab.org/>)

References

See website: <http://strimmerlab.org/software/corpcor/>

See Also

[cov.shrink](#), [invcov.shrink](#), [powcor.shrink](#), [pcor.shrink](#), [mvr.shrink](#), [fast.svd](#).

cor2pcor	<i>Compute Partial Correlation from Correlation Matrix – and Vice Versa</i>
----------	---

Description

`cor2pcor` computes the pairwise *partial* correlation coefficients from either a correlation or a covariance matrix.

`pcor2cor` takes either a partial correlation matrix or a partial covariance matrix as input, and computes from it the corresponding correlation matrix.

Usage

```
cor2pcor(m, tol)
pcor2cor(m, tol)
```

Arguments

<code>m</code>	covariance matrix or (partial) correlation matrix
<code>tol</code>	tolerance - singular values larger than <code>tol</code> are considered non-zero (default value: <code>tol = max(dim(m)) * max(D) * .Machine\$double.eps</code>). This parameter is needed for the singular value decomposition on which pseudoinverse is based.

Details

The partial correlations are the negative standardized concentrations (which in turn are the off-diagonal elements of the inverse correlation or covariance matrix). In graphical Gaussian models the partial correlations represent the direct interactions between two variables, conditioned on all remaining variables.

In the above functions the [pseudoinverse](#) is employed for inversion - hence even singular covariances (with some zero eigenvalues) may be used. However, a better option may be to estimate a positive definite covariance matrix using [cov.shrink](#).

Note that for efficient computation of partial correlation coefficients from data `x` it is advised to use `pcor.shrink(x)` and *not* `cor2pcor(cor.shrink(x))`.

Value

A matrix with the pairwise partial correlation coefficients (`cor2pcor`) or with pairwise correlations (`pcor2cor`).

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

References

Whittaker J. 1990. Graphical Models in Applied Multivariate Statistics. John Wiley, Chichester.

See Also

[decompose.invcov](#), [pcor.shrink](#), [pseudoinverse](#).

Examples

```
# load corpcor library
library("corpcor")

# covariance matrix
m.cov = rbind(
  c(3,1,1,0),
  c(1,3,0,1),
  c(1,0,2,0),
  c(0,1,0,2)
)
m.cov

# corresponding correlation matrix
m.cor.1 = cov2cor(m.cov)
m.cor.1

# compute partial correlations (from covariance matrix)
m.pcor.1 = cor2pcor(m.cov)
m.pcor.1

# compute partial correlations (from correlation matrix)
m.pcor.2 = cor2pcor(m.cor.1)
m.pcor.2

zapsmall( m.pcor.1 ) == zapsmall( m.pcor.2 )

# backtransformation
m.cor.2 = pcor2cor(m.pcor.1)
m.cor.2
zapsmall( m.cor.1 ) == zapsmall( m.cor.2 )
```

Description

The functions `var.shrink`, `cor.shrink`, and `cov.shrink` compute shrinkage estimates of variance, correlation, and covariance, respectively.

Usage

```
var.shrink(x, lambda.var, w, verbose=TRUE)
cor.shrink(x, lambda, w, collapse=FALSE, verbose=TRUE)
cov.shrink(x, lambda, lambda.var, w, collapse=FALSE, verbose=TRUE)
```

Arguments

<code>x</code>	a data matrix
<code>lambda</code>	the correlation shrinkage intensity (range 0-1). If <code>lambda</code> is not specified (the default) it is estimated using an analytic formula from Sch\"afer and Strimmer (2005) - see details below. For <code>lambda=0</code> the empirical correlations are recovered.
<code>lambda.var</code>	the variance shrinkage intensity (range 0-1). If <code>lambda.var</code> is not specified (the default) it is estimated using an analytic formula from Opgen-Rhein and Strimmer (2007) - see details below. For <code>lambda.var=0</code> the empirical variances are recovered.
<code>w</code>	optional: weights for each data point - if not specified uniform weights are assumed (<code>w = rep(1/n, n)</code> with <code>n = nrow(x)</code>).
<code>collapse</code>	return vector instead of matrix if estimated or specified <code>lambda</code> equals 1.
<code>verbose</code>	output some status messages while computing (default: TRUE)

Details

`var.shrink` computes the empirical variance of each considered random variable, and shrinks them towards their median. The shrinkage intensity is estimated using Eq. 6 from Sch\"afer and Strimmer (2005), which leads to

$$\lambda_{var}^* = \left(\sum_{k=1}^p Var(s_{kk}) \right) / \sum_{k=1}^p (s_{kk} - medians)^2$$

where *medians* denotes the median of the empirical variances.

Similarly `cor.shrink` computes a shrinkage estimate of the correlation matrix by shrinking the empirical correlations towards the identity matrix. In this case the shrinkage intensity is

$$\lambda^* = \sum_{k \neq l} Var(r_{kl}) / \sum_{k \neq l} r_{kl}^2.$$

`cov.shrink` computes the corresponding full covariance matrix on the basis of the shrunken correlation matrix and the shrunken variances.

In comparison with the standard empirical estimates (`var`, `cov`, and `cor`) the shrinkage estimates exhibit a number of favorable properties. For instance,

1. they are typically much more efficient, i.e. they show (sometimes dramatically) better mean squared error,
2. the estimated covariance and correlation matrices are always positive definite and well conditioned (so that there are no numerical problems when computing their inverse),
3. they are inexpensive to compute, and
4. they are fully automatic and do not require any tuning parameters (as the shrinkage intensity is analytically estimated from the data), and
5. they assume nothing about the underlying distributions, except for the existence of the first two moments.

These properties also carry over to derived quantities, such as partial variances and partial correlations (`pvar.shrink` and `pcor.shrink`).

As an extra benefit, the shrinkage estimators have a form that can be *very* efficiently inverted, especially if the number of variables is large and the sample size is small. Thus, instead of inverting the matrix output by `cov.shrink` and `cor.shrink` please use the functions `invcov.shrink` and `invcor.shrink`, respectively.

Value

`var.shrink` returns a vector with estimated variances.

`cov.shrink` returns a covariance matrix.

`cor.shrink` returns the corresponding correlation matrix.

Author(s)

Juliane Sch\u00f6fer, Rainer Opgen-Rhein, and Korbinian Strimmer (<http://strimmerlab.org>).

References

Opgen-Rhein, R., and K. Strimmer. 2007. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statist. Appl. Genet. Mol. Biol.* **6**:9. (<http://www.bepress.com/sagmb/vol6/iss1/art9/>)

Sch\u00f6fer, J., and K. Strimmer. 2005. A shrinkage approach to large-scale covariance estimation and implications for functional genomics. *Statist. Appl. Genet. Mol. Biol.* **4**:32. (<http://www.bepress.com/sagmb/vol4/iss1/art32/>)

See Also

`invcov.shrink`, `pcor.shrink`, `cor2pcor`

Examples

```
# load corpcor library
library("corpcor")

# small n, large p
p = 100
n = 20
```

```
# generate random p x p covariance matrix
sigma = matrix(rnorm(p*p), ncol=p)
sigma = crossprod(sigma) + diag(rep(0.1, p))

# simulate multinormal data of sample size n
sigsvd = svd(sigma)
Y = t(sigsvd$v %*% (t(sigsvd$u) * sqrt(sigsvd$d)))
X = matrix(rnorm(n * ncol(sigma)), nrow = n) %*% Y

# estimate covariance matrix
s1 = cov(X)
s2 = cov.shrink(X)

# squared error
sum((s1-sigma)^2)
sum((s2-sigma)^2)

# compare positive definiteness
is.positive.definite(sigma)
is.positive.definite(s1)
is.positive.definite(s2)

# compare ranks and condition
rank.condition(sigma)
rank.condition(s1)
rank.condition(s2)

# compare eigenvalues
e0 = eigen(sigma, symmetric=TRUE)$values
e1 = eigen(s1, symmetric=TRUE)$values
e2 = eigen(s2, symmetric=TRUE)$values
m = max(e0, e1, e2)
y1 = c(0, m)

par(mfrow=c(1,3))
plot(e1, main="empirical")
plot(e2, ylim=y1, main="full shrinkage")
plot(e0, ylim=y1, main="true")
par(mfrow=c(1,1))
```

Description

`fast.svd` returns the singular value decomposition of a rectangular real matrix

$$M = UDV',$$

where U and V are orthogonal matrices with $U'U = I$ and $V'V = I$, and D is a diagonal matrix containing the singular values (see `svd`).

The main difference to the native version `svd` is that `fast.svd` is substantially faster for "fat" (small n , large p) and "thin" (large n , small p) matrices. In this case the decomposition of M can be greatly sped up by first computing the SVD of either MM' (fat matrices) or $M'M$ (thin matrices), rather than that of M .

A second difference to `svd` is that `fast.svd` only returns the *positive* singular values (thus the dimension of D always equals the rank of M). Note that the singular vectors computed by `fast.svd` may differ in sign from those computed by `svd`.

Usage

```
fast.svd(m, tol)
```

Arguments

<code>m</code>	matrix
<code>tol</code>	tolerance - singular values larger than <code>tol</code> are considered non-zero (default value: <code>tol = max(dim(m)) * max(D) * .Machine\$double.eps</code>)

Details

For "fat" M (small n , large p) the SVD decomposition of MM' yields

$$MM' = UD^2U$$

As the matrix MM' has dimension $n \times n$ only, this is faster to compute than SVD of M . The V matrix is subsequently obtained by

$$V = M'UD^{-1}$$

Similarly, for "thin" M (large n , small p), the decomposition of $M'M$ yields

$$M'M = VD^2V'$$

which is also quick to compute as $M'M$ has only dimension $p \times p$. The U matrix is then computed via

$$U = MVD^{-1}$$

Value

A list with the following components:

d	a vector containing the <i>positive</i> singular values
u	a matrix with the corresponding left singular vectors
v	a matrix with the corresponding left singular vectors

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

[svd](#), [solve](#).

Examples

```
# load corpcor library
library("corpcor")

# generate a "fat" data matrix
n = 50
p = 5000
X = matrix(rnorm(n*p), n, p)

# compute SVD
system.time( (s1 = svd(X)) )
system.time( (s2 = fast.svd(X)) )

eps = 1e-10
sum(abs(s1$d-s2$d) > eps)
sum(abs(abs(s1$u)-abs(s2$u)) > eps)
sum(abs(abs(s1$v)-abs(s2$v)) > eps)
```

invcov.shrink	<i>Fast Computation of the Inverse of the Covariance and Correlation Matrix</i>
---------------	---

Description

The functions `invcov.shrink` and `invcor.shrink` implement an algorithm to *efficiently* compute the inverses of shrinkage estimates of covariance ([cov.shrink](#)) and correlation ([cor.shrink](#)).

Usage

```
invcov.shrink(x, lambda, lambda.var, w, collapse=FALSE, verbose=TRUE)
invcor.shrink(x, lambda, w, collapse=FALSE, verbose=TRUE)
```

Arguments

<code>x</code>	a data matrix
<code>lambda</code>	the correlation shrinkage intensity (range 0-1). If <code>lambda</code> is not specified (the default) it is estimated using an analytic formula from Sch\"afer and Strimmer (2005) - see cor.shrink . For <code>lambda=0</code> the empirical correlations are recovered.
<code>lambda.var</code>	the variance shrinkage intensity (range 0-1). If <code>lambda.var</code> is not specified (the default) it is estimated using an analytic formula from Sch\"afer and Strimmer (2005) - see var.shrink . For <code>lambda.var=0</code> the empirical variances are recovered.
<code>w</code>	optional: weights for each data point - if not specified uniform weights are assumed (<code>w = rep(1/n, n)</code> with <code>n = nrow(x)</code>).
<code>collapse</code>	return vector instead of matrix if estimated or specified <code>lambda</code> equals 1.
<code>verbose</code>	output status while computing (default: TRUE)

Details

Both `invcov.shrink` and `invcor.shrink` rely on [powcor.shrink](#). This allows to compute the inverses in a very efficient fashion (much more efficient than directly inverting the matrices - see the example).

Value

`invcov.shrink` returns the inverse of the output from [cov.shrink](#).

`invcor.shrink` returns the inverse of the output from [cor.shrink](#).

Author(s)

Juliane Sch\"afer and Korbinian Strimmer (<http://strimmerlab.org>).

References

Sch\"afer, J., and K. Strimmer. 2005. A shrinkage approach to large-scale covariance estimation and implications for functional genomics. *Statist. Appl. Genet. Mol. Biol.* **4**:32. (<http://www.bepress.com/sagmb/vol4/iss1/art32/>)

See Also

[powcor.shrink](#), [cov.shrink](#), [pcor.shrink](#), [cor2pcor](#)

Examples

```
# load corpcor library
library("corpcor")

# generate data matrix
p = 500
n = 10
```

```

X = matrix(rnorm(n*p), nrow = n, ncol = p)

lambda = 0.23 # some arbitrary lambda

# slow
system.time(
  (W1 = solve(cov.shrink(X, lambda)))
)

# very fast
system.time(
  (W2 = invcov.shrink(X, lambda))
)

# no difference
sum((W1-W2)^2)

```

mpower

Compute the Power of a Real Symmetric Matrix

Description

mpower computes m^{α} , i.e. the α -th power of the real symmetric matrix m .

Usage

```
mpower(m, alpha, pseudo=FALSE, tol)
```

Arguments

<code>m</code>	a real-valued symmetric matrix.
<code>alpha</code>	exponent.
<code>pseudo</code>	if <code>pseudo=TRUE</code> then all zero eigenvalues are dropped (e.g. for computing the pseudoinverse). The default is to use all eigenvalues.
<code>tol</code>	tolerance - eigenvalues with absolute value smaller or equal to <code>tol</code> are considered identically zero (default: <code>tol = max(dim(m)) * max(abs(eval)) * .Machine\$double.eps</code>).

Details

The matrix power of m is obtained by first computing the spectral decomposition of m , and subsequent modification of the resulting eigenvalues.

Note that m is assumed to be symmetric, and only its lower triangle (diagonal included) is used in [eigen](#).

For computing the matrix power of [cov.shrink](#) use the vastly more efficient function [powcov.shrink](#).

Value

mpower returns a matrix of the same dimensions as m .

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

[powcor.shrink](#), [eigen](#).

Examples

```
# load corpcor library
library("corpcor")

# generate symmetric matrix
p = 10
n = 20
X = matrix(rnorm(n*p), nrow = n, ncol = p)
m = cor(X)

m %*% m
mpower(m, 2)

solve(m)
mpower(m, -1)

msq = mpower(m, 0.5)
msq %*% msq
m

mpower(m, 1.234)
```

mvr.shrink

Multivariate Shrinkage Regression

Description

`mvr.shrink` implements an analytic shrinkage approach for multivariate linear regression that can be applied for "small n, large p" data.

`mvr.predict` is a function to predict the response given the (estimated) regression coefficients.

Usage

```
mvr.shrink(x, y, lambda, lambda.var, w, verbose=TRUE)
mvr.predict(coef, x)
```

Arguments

<code>x</code>	a predictor matrix.
<code>y</code>	a response vector or matrix. If <code>y</code> is missing then each individual column of <code>x</code> serves as univariate response vector that is regressed in turn against the remaining predictors.
<code>lambda</code>	the correlation shrinkage intensity (range 0-1). If <code>lambda</code> is not specified (the default) it is estimated using an analytic formula from Sch" afer and Strimmer (2005) - see details below. For <code>lambda=0</code> the empirical correlations are recovered.
<code>lambda.var</code>	the variance shrinkage intensity (range 0-1). If <code>lambda.var</code> is not specified (the default) it is estimated using an analytic formula from Opgen-Rhein and Strimmer (2007) - see details below. For <code>lambda.var=0</code> the empirical variances are recovered.
<code>w</code>	optional: weights for each data point - if not specified uniform weights are assumed (<code>w = rep(1/n, n)</code> with <code>n = nrow(x)</code>).
<code>verbose</code>	output some status messages while computing (default: TRUE)
<code>coef</code>	the matrix of regression coefficients, as output by <code>mvr.shrink()</code> .

Details

Regression coefficients and partial correlations/variances are interrelated, see for example Cox and Wermuth (1993). Specifically, the regression coefficients are the partial correlations times the square-root of the ratio of the partial variances.

The function `mvr.shrink` first computes shrinkage estimates of partial correlation and variance following the method described in Sch" afer and Strimmer (2005) and Opgen-Rhein and Strimmer (2007), and then calculates from these the respective regression coefficients.

Note that if the response is multivariate, the the type of regression performed is block regression (Cox and Wermuth 1993).

Value

`mvr.shrink` returns a matrix with the estimated regression coefficients.

`mvr.predict` returns a response matrix.

Author(s)

Rainer Opgen-Rhein and Korbinian Strimmer (<http://strimmerlab.org>).

References

Cox, D. R., and N. Wermuth. 1993. Linear dependencies represented by chain graphs. *Statistical Science* **8**:204-283.

Opgen-Rhein, R., and K. Strimmer. 2007. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statist. Appl. Genet. Mol. Biol.* **6**:9. (<http://www.bepress.com/sagmb/vol6/iss1/art9/>)

Schaffer, J., and K. Strimmer. 2005. A shrinkage approach to large-scale covariance estimation and implications for functional genomics. *Statist. Appl. Genet. Mol. Biol.* 4:32. (<http://www.bepress.com/sagmb/vol14/iss1/art32/>)

See Also

[cov.shrink](#)

Examples

```
# load corpcor library
library("corpcor")

# example data
data(longley)
x = as.matrix(longley[ 1:13, -2])
y = as.matrix(longley[ 1:13,  2, drop=FALSE])
xtest = as.matrix(longley[14:16, -2])
ytest = as.matrix(longley[14:16,  2, drop=FALSE])

# least squares regression is recovered if
# shrinkage intensities are all set to zero
lm(y ~ x)
coefs.ols = mvr.shrink(x, y, lambda=0, lambda.var=0)
coefs.ols

# shrinkage regression coefficients
# note that these are quite different!
coefs.shrink = mvr.shrink(x, y)
coefs.shrink

# prediction
mvr.predict(coefs.ols, xtest)
mvr.predict(coefs.shrink, xtest)

# if no response matrix is given, then each predictor
# is regressed in turn against all others
coefs = mvr.shrink(longley)
coefs

# connection between partial correlations and regression coefficients
# (note that the mvr.shrink() function is specifically constructed so that
# this relationship holds for the shrinkage estimates)

beta = coefs[,-1] # shrinkage regression coefficients w/o intercept
pc = pcor.shrink(longley) # shrunken partial correlations
```

```

k = 3
l = 4
# partial correlations as computed by pcor.shrink()
pc[k,l]
# partial correlations obtained from mvr.shrink() regression coefficients
sqrt(beta[k,l] * beta[l,k]) * sign(beta[l,k])

```

pcor.shrink

Shrinkage Estimates of Partial Correlation and Partial Variance

Description

The functions `pcor.shrink` and `pvar.shrink` compute shrinkage estimates of partial correlation and partial variance, respectively.

Usage

```

pcor.shrink(x, lambda, w, verbose=TRUE)
pvar.shrink(x, lambda, lambda.var, w, verbose=TRUE)

```

Arguments

<code>x</code>	a data matrix
<code>lambda</code>	the correlation shrinkage intensity (range 0-1). If <code>lambda</code> is not specified (the default) it is estimated using an analytic formula from Sch" afer and Strimmer (2005) - see cor.shrink . For <code>lambda=0</code> the empirical correlations are recovered.
<code>lambda.var</code>	the variance shrinkage intensity (range 0-1). If <code>lambda.var</code> is not specified (the default) it is estimated using an analytic formula from Opgen-Rhein and Strimmer (2007) - see details below. For <code>lambda.var=0</code> the empirical variances are recovered.
<code>w</code>	optional: weights for each data point - if not specified uniform weights are assumed (<code>w = rep(1/n, n)</code> with <code>n = nrow(x)</code>).
<code>verbose</code>	report progress while computing (default: TRUE)

Details

The partial variance $var(X_k|rest)$ is the variance of X_k conditioned on the remaining variables. It equals the inverse of the corresponding diagonal entry of the precision matrix (see Whittaker 1990).

The partial correlations $corr(X_k, X_l|rest)$ is the correlation between X_k and X_l conditioned on the remaining variables. It equals the sign-reversed entries of the off-diagonal entries of the precision matrix, standardized by the the squared root of the associated inverse partial variances.

Note that using `pcor.shrink(x)` *much* faster than `cor2pcor(cor.shrink(x))`.

For details about the shrinkage procedure consult Sch" afer and Strimmer (2005), Opgen-Rhein and Strimmer (2007), and the help page of [cov.shrink](#).

Value

`pcor.shrink` returns the partial correlation matrix. Attached to this matrix are the standardized partial variances (i.e. PVAR/VAR) that can be retrieved using `attr` under the attribute "spv".

`pvar.shrink` returns the partial variances.

Author(s)

Juliane Schöfer and Korbinian Strimmer (<http://strimmerlab.org>).

References

Opgen-Rhein, R., and K. Strimmer. 2007. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. *Statist. Appl. Genet. Mol. Biol.* **6**:9. (<http://www.bepress.com/sagmb/vol6/iss1/art9/>)

Schöfer, J., and K. Strimmer. 2005. A shrinkage approach to large-scale covariance estimation and implications for functional genomics. *Statist. Appl. Genet. Mol. Biol.* **4**:32. (<http://www.bepress.com/sagmb/vol4/iss1/art32/>)

Whittaker J. 1990. *Graphical Models in Applied Multivariate Statistics*. John Wiley, Chichester.

See Also

[invcov.shrink](#), [cov.shrink](#), [cor2pcor](#)

Examples

```
# load corpcor library
library("corpcor")

# generate data matrix
p = 50
n = 10
X = matrix(rnorm(n*p), nrow = n, ncol = p)

# partial variance
pv = pvar.shrink(X)
pv

# partial correlations (fast and recommend way)
pcr1 = pcor.shrink(X)

# other possibilities to estimate partial correlations
pcr2 = cor2pcor( cor.shrink(X) )

# all the same
sum((pcr1 - pcr2)^2)
```

Description

The function `powcor.shrink` efficiently computes the α -th power of the shrinkage correlation matrix produced by `cor.shrink`.

For instance, this function may be used for fast computation of the (inverse) square root of the shrinkage correlation matrix (needed, e.g., for decorrelation).

Usage

```
powcor.shrink(x, alpha, lambda, w, collapse=FALSE, verbose=TRUE)
```

Arguments

<code>x</code>	a data matrix
<code>alpha</code>	exponent
<code>lambda</code>	the correlation shrinkage intensity (range 0-1). If <code>lambda</code> is not specified (the default) it is estimated using an analytic formula from Sch\"afer and Strimmer (2005) - see <code>cor.shrink</code> . For <code>lambda=0</code> the empirical correlations are recovered.
<code>w</code>	optional: weights for each data point - if not specified uniform weights are assumed (<code>w = rep(1/n, n)</code> with <code>n = nrow(x)</code>).
<code>collapse</code>	return vector instead of matrix if estimated or specified <code>lambda</code> equals 1.
<code>verbose</code>	output status while computing (default: TRUE)

Details

This function employs a special matrix identity to speed up the computation of the matrix power of the shrinkage correlation matrix (see Zuber and Strimmer 2009 for details).

Apart from a scaling factor the shrinkage correlation matrix computed by `cor.shrink` takes on the form

$$Z = I_p + VMV^T,$$

where VMV^T is a multiple of the empirical correlation matrix. Crucially, Z is a matrix of size p times p whereas M is a potentially much smaller matrix of size m times m , where m is the rank of the empirical correlation matrix.

In order to calculate the α -th power of Z the function uses the identity

$$Z^\alpha = I_p - V(I_m - (I_m + M)^\alpha)V^T$$

requiring only the computation of the α -th power of the m by m matrix $I_m + M$. This trick enables substantial computational savings especially when the number of observations is much smaller than the number of variables.

Note that the above identity is related but not identical to the Woodbury matrix identity for inversion of a matrix. For $\alpha = -1$ the above identity reduces to

$$Z^{-1} = I_p - V(I_m - (I_m + M)^{-1})V^T,$$

whereas the Woodbury matrix identity equals

$$Z^{-1} = I_p - V(I_m + M^{-1})^{-1}V^T.$$

Value

`powcor.shrink` returns a matrix containing the output from `cor.shrink` taken to the power α .

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

References

Zuber, V., and K. Strimmer. 2009. Gene ranking and biomarker discovery under correlation. *Bioinformatics* **25**:2700-2707. (<http://arxiv.org/abs/0902.0751>)

See Also

[invcor.shrink](#), [cor.shrink](#), [mpower](#).

Examples

```
# load corpcor library
library("corpcor")

# generate data matrix
p = 500
n = 10
X = matrix(rnorm(n*p), nrow = n, ncol = p)

lambda = 0.23 # some arbitrary lambda

### computing the inverse ###
# slow
system.time(
  (W1 = solve(cor.shrink(X, lambda=lambda)))
)

# very fast
system.time(
```

```

    (W2 = powcor.shrink(X, alpha=-1, lambda=lambda))
)

# no difference
sum((W1-W2)^2)

### computing the square root ###

system.time(
  (W1 = mpower(cor.shrink(X, lambda=lambda), alpha=0.5))
)

# very fast
system.time(
  (W2 = powcor.shrink(X, alpha=0.5, lambda=lambda))
)

# no difference
sum((W1-W2)^2)

### computing an arbitrary power (alpha=1.23) ###

system.time(
  (W1 = mpower(cor.shrink(X, lambda=lambda), alpha=1.23))
)

# very fast
system.time(
  (W2 = powcor.shrink(X, alpha=1.23, lambda=lambda))
)

# no difference
sum((W1-W2)^2)

```

pseudoinverse

Pseudoinverse of a Matrix

Description

The standard definition for the inverse of a matrix fails if the matrix is not square or singular. However, one can generalize the inverse using singular value decomposition. Any rectangular real matrix M can be decomposed as

$$M = UDV'$$

where U and V are orthogonal, V' means V transposed, and D is a diagonal matrix containing only the positive singular values (as determined by `tol`, see also [fast.svd](#)).

The pseudoinverse, also known as Moore-Penrose or generalized inverse is then obtained as

$$iM = VD^{-1}U'$$

Usage

```
pseudoinverse(m, tol)
```

Arguments

m	matrix
tol	tolerance - singular values larger than tol are considered non-zero (default value: <code>tol = max(dim(m)) * max(D) * .Machine\$double.eps</code>)

Details

The pseudoinverse has the property that the sum of the squares of all the entries in $m M^{-1} - I$, where I is an appropriate identity matrix, is minimized. For non-singular matrices the pseudoinverse is equivalent to the standard inverse.

Value

A matrix (the pseudoinverse of m).

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

[solve](#), [fast.svd](#)

Examples

```
# load corpcor library
library("corpcor")

# a singular matrix
m = rbind(
  c(1,2),
  c(1,2)
)

# not possible to invert exactly
try(solve(m))

# pseudoinverse
p = pseudoinverse(m)
p

# characteristics of the pseudoinverse
zapsmall(m %*% p %*% m) == zapsmall(m)
zapsmall(p %*% m %*% p) == zapsmall(p)
zapsmall(p %*% m) == zapsmall(t(p %*% m))
zapsmall(m %*% p) == zapsmall(t(m %*% p))
```

```
# example with an invertable matrix
m2 = rbind(
  c(1,1),
  c(1,0)
)
zapsmall( solve(m2) ) == zapsmall( pseudoinverse(m2) )
```

rank.condition *Positive Definiteness of a Matrix, Rank and Condition Number*

Description

`is.positive.definite` tests whether all eigenvalues of a symmetric matrix are positive.

`make.positive.definite` computes the nearest positive definite of a real symmetric matrix, using the algorithm of NJ Higham (1988, Linear Algebra Appl. 103:103-118).

`rank.condition` estimates the rank and the condition of a matrix by computing its singular values $D[i]$ (using `svd`). The rank of the matrix is the number of singular values $D[i] > \text{tol}$ and the condition is the ratio of the largest and the smallest singular value.

Note that the definition `tol = max(dim(m)) * max(D) * Machine$double.eps` is exactly compatible with the conventions used in "Octave" or "Matlab".

Usage

```
is.positive.definite(m, tol, method=c("eigen", "chol"))
make.positive.definite(m, tol)
rank.condition(m, tol)
```

Arguments

<code>m</code>	matrix
<code>tol</code>	tolerance for singular values and for absolute eigenvalues - only those with values larger than <code>tol</code> are considered non-zero (default: <code>tol = max(dim(m)) * max(D) * Machine\$double.eps</code>)
<code>method</code>	Determines the method to check for positive definiteness: eigenvalue computation (<code>eigen</code> , default) or Cholesky decomposition (<code>chol</code>).

Value

`is.positive.definite` returns a logical value (TRUE or FALSE).

`rank.condition` returns a list object with the following components:

<code>rank</code>	Rank of the matrix.
<code>condition</code>	Condition number.
<code>tol</code>	Tolerance.

`make.positive.definite` returns a symmetric positive definite matrix.

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

[svd](#), [pseudoinverse](#).

Examples

```
# load corpcor library
library("corpcor")

# Hilbert matrix
hilbert = function(n) { i = 1:n; 1 / outer(i - 1, i, "+") }

# positive definite ?
m = hilbert(8)
is.positive.definite(m)

# numerically ill-conditioned
m = hilbert(15)
rank.condition(m)

# make positive definite
m2 = make.positive.definite(m)
is.positive.definite(m2)
rank.condition(m2)
m2 - m
```

rebuild.cov

Rebuild and Decompose the (Inverse) Covariance Matrix

Description

`rebuild.cov` takes a correlation matrix and a vector with variances and reconstructs the corresponding covariance matrix.

Conversely, `decompose.cov` decomposes a covariance matrix into correlations and variances.

`decompose.invcov` decomposes a concentration matrix (=inverse covariance matrix) into partial correlations and partial variances.

`rebuild.invcov` takes a partial correlation matrix and a vector with partial variances and reconstructs the corresponding concentration matrix.

Usage

```
rebuild.cov(r, v)
rebuild.invcov(pr, pv)
decompose.cov(m)
decompose.invcov(m)
```

Arguments

r	correlation matrix
v	variance vector
pr	partial correlation matrix
pv	partial variance vector
m	a covariance or a concentration matrix

Details

The diagonal elements of the concentration matrix (=inverse covariance matrix) are the precisions, and the off-diagonal elements are the concentrations. Thus, the partial variances correspond to the inverse precisions, and the partial correlations to the negative standardized concentrations.

Value

`rebuild.cov` and `rebuild.invcov` return a matrix.

`decompose.cov` and `decompose.invcov` return a list containing a matrix and a vector.

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

`cor`, `cov`, `pcor.shrink`

Examples

```
# load corpcor library
library("corpcor")

# a correlation matrix and some variances
r = matrix(c(1, 1/2, 1/2, 1), nrow = 2, ncol=2)
r
v = c(2, 3)

# construct the associated covariance matrix
c = rebuild.cov(r, v)
c

# decompose into correlations and variances
decompose.cov(c)

# the corresponding concentration matrix
conc = pseudoinverse(c)
conc

# decompose into partial correlation matrix and partial variances
```

```
tmp = decompose.invcov(conc)
tmp
# note: because this is an example with two variables,
# the partial and standard correlations are identical!

# reconstruct the concentration matrix from partial correlations and
# partial variances
rebuild.invcov(tmp$pr, tmp$pv)
```

smtools

Some Tools for Handling Symmetric Matrices

Description

`sm2vec` takes a symmetric matrix and puts the lower triangular entries into a vector (cf. `lower.tri`).
`sm.index` lists the corresponding x-y-indices for each entry in the vector produced by `sm2vec`.
`vec2sm` reverses the operation by `sm2vec` and converts the vector back to a symmetric matrix. If `diag=FALSE` the diagonal of the resulting matrix will consist of NAs. If `order` is supplied then the input vector `vec` will first be rearranged accordingly.

Usage

```
sm2vec(m, diag = FALSE)
sm.index(m, diag = FALSE)
vec2sm(vec, diag = FALSE, order = NULL)
```

Arguments

<code>m</code>	symmetric matrix
<code>diag</code>	logical. Should the diagonal be included in the conversion to and from a vector?
<code>vec</code>	vector of unique elements from a symmetric matrix
<code>order</code>	order of the entries in <code>vec</code>

Value

A vector (`sm2vec`), a two-column matrix with indices (`sm.index`), or a symmetric matrix (`vec2sm`).

Author(s)

Korbinian Strimmer (<http://strimmerlab.org/>).

See Also

`lower.tri`.

Examples

```

# load corpcor library
library("corpcor")

# a symmetric matrix
m = rbind(
  c(3,1,1,0),
  c(1,3,0,1),
  c(1,0,2,0),
  c(0,1,0,2)
)
m

# convert into vector (including the diagonals)
v = sm2vec(m, diag=TRUE)
v.idx = sm.index(m, diag=TRUE)
v
v.idx

# put back to symmetric matrix
vec2sm(v, diag=TRUE)

# convert from vector with specified order of the elements
sv = sort(v)
sv
ov = order(v)
ov
vec2sm(sv, diag=TRUE, order=ov)

```

wt.scale

Weighted Expectations and Variances

Description

wt.var estimate the unbiased variance taking into account data weights.

wt.moments produces the weighted mean and weighted variance for each column of a matrix.

wt.scale centers and standardized a matrix using the weighted means and variances.

Usage

```

wt.var(xvec, w)
wt.moments(x, w)
wt.scale(x, w, center=TRUE, scale=TRUE)

```

Arguments

xvec	a vector
x	a matrix

w	data weights
center	logical value
scale	logical value

Value

A rescaled matrix (`wt.scale`), a list containing the column means and variances (`wt.moments`), or single number (`wt.var`)

Author(s)

Korbinian Strimmer (<http://strimmerlab.org>).

See Also

[weighted.mean](#), [cov.wt](#).

Examples

```
# load corpcor library
library("corpcor")

# generate some data
p = 5
n = 5
X = matrix(rnorm(n*p), nrow = n, ncol = p)
w = c(1,1,1,3,3)/9

# standardize matrix
scale(X)
wt.scale(X)
wt.scale(X, w) # take into account data weights
```

Index

*Topic **algebra**

fast.svd, 7
mpower, 10
pseudoinverse, 19
rank.condition, 20

*Topic **multivariate**

cor2pcor, 2
corpcor-package, 2
cov.shrink, 4
invcov.shrink, 9
mvr.shrink, 12
pcor.shrink, 14
powcor.shrink, 16
rebuild.cov, 22
wt.scale, 25

*Topic **utilities**

smtools, 23

attr, 15

cor, 5, 23

cor.shrink, 9, 11, 14, 16, 17

cor.shrink(cov.shrink), 4

cor2pcor, 2, 6, 10, 15

corpcor-package, 2

cov, 5, 23

cov.shrink, 2, 3, 4, 9, 10, 13, 15

cov.wt, 25

decompose.cov(rebuild.cov), 22

decompose.invcov, 3

decompose.invcov(rebuild.cov), 22

eigen, 11

fast.svd, 2, 7, 19

invcov.shrink, 5, 17

invcov.shrink(invcov.shrink), 9

invcov.shrink, 2, 5, 6, 9, 15

is.positive.definite
(rank.condition), 20

lower.tri, 23, 24

make.positive.definite
(rank.condition), 20

mpower, 10, 17

mvr.predict(mvr.shrink), 12

mvr.shrink, 2, 12

pcor.shrink, 2, 3, 5, 6, 10, 14, 23

pcor2cor(cor2pcor), 2

powcor.shrink, 2, 9–11, 16

pseudoinverse, 3, 19, 21

pvar.shrink, 5

pvar.shrink(pcor.shrink), 14

rank.condition, 20

rebuild.cov, 22

rebuild.invcov(rebuild.cov), 22

sm.index(smtools), 23

sm2vec(smtools), 23

smtools, 23

solve, 8, 19

svd, 7, 8, 20, 21

var, 5

var.shrink, 9

var.shrink(cov.shrink), 4

vec2sm(smtools), 23

weighted.mean, 25

wt.moments(wt.scale), 25

wt.scale, 25

wt.var(wt.scale), 25