Package ‘fungible’

November 9, 2016

Version  1.5
Date     2016-11-08
Title    Fungible Coefficients and Monte Carlo Functions
Author   Niels G. Waller <nwaller@umn.edu> and Jeff Jones <jeff.jones@kornferry.com>
Maintainer Niels G. Waller <nwaller@umn.edu>
Depends  R (>= 3.0)
Imports  e1071, lattice, MASS, mvtnorm, R2Cuba, stringr, nleqslv, methods
Description Computes fungible coefficients and Monte Carlo data.
Underlying theory for these functions is described in the following publications:
License  GPL (>= 2)
NeedsCompilation no
Repository  CRAN
Date/Publication  2016-11-09 23:54:09

R topics documented:

  adfCor .......................................................... 2
  adfCov .......................................................... 4
  bigen .......................................................... 5
  corSample ....................................................... 7
**Description**

Function for computing an asymptotic distribution-free covariance matrix of correlations.

**Usage**

```r
adfCor(X, y = NULL)
```
**Arguments**

- **X**  
  Data matrix.
- **y**  
  Optional vector of criterion scores.

**Value**

- **adfCorMat**  
  Asymptotic distribution-free estimate of the covariance matrix of correlations.

**Author(s)**

Jeff Jones and Niels Waller

**References**


**Examples**

```r
## Generate non-normal data using monte4
set.seed(123)
## we will simulate data for 1000 subjects
N <- 1000

## R = the desired population correlation matrix among predictors
R <- matrix(c(1, .5, .5, 1), 2, 2)

## Consider a regression model with coefficient of determination (Rsq):
Rsq <- .50

## and vector of standardized regression coefficients
Beta <- sqrt(Rsq/t(sqrt(c(.5, .5)))) * sqrt(c(.5, .5))

## generate non-normal data for the predictors (X)
## x1 has expected skew = 1 and kurtosis = 3
## x2 has expected skew = 2 and kurtosis = 5
X <- monte4(seed = 123, nvar = 2, nsub = N, cormat = R, skewvec = c(1, 2), kurtvec = c(3, 5))$data

## generate criterion scores
y <- X %*% Beta + sqrt(1-Rsq)*rnorm(N)

## Create ADF Covariance Matrix of Correlations
adfCor(X, y)
```

```r
#> 12 13 23
#> 12 0.0012078454 0.0005331086 0.0004821594
#> 13 0.0005331086 0.0004980130 0.0002712080
```
**Description**

Function for computing an asymptotic distribution-free covariance matrix of covariances.

**Usage**

```r
adfCov(X, y = NULL)
```

**Arguments**

- `X`: Data matrix.
- `y`: Optional vector of criterion scores.

**Value**

`adfCovMat`: Asymptotic distribution-free estimate of the covariance matrix of covariances

**Author(s)**

Jeff Jones and Niels Waller

**References**


**Examples**

```r
## Generate non-normal data using monte1
set.seed(123)

## we will simulate data for 1000 subjects
N <- 1000

## R = the desired population correlation matrix among predictors
R <- matrix(c(1, .5, .5, 1), 2, 2)

## Consider a regression model with coefficient of determination (Rsq):
Rsq <- .50

## and vector of standardized regression coefficients
Beta <- sqrt(Rsq/t(sqrt(c(.5, .5)))) %*% R %*% sqrt(c(.5, .5)) * sqrt(c(.5, .5))
```
## Generate Correlated Binary Data

**Description**

Function for generating binary data with population thresholds.

**Usage**

bigen(data, n, thresholds, seed = NULL)

**Arguments**

- **data**: Either a matrix of binary (0/1) indicators or a correlation matrix.
- **n**: The desired sample size of the simulated data.
- **thresholds**: If `x` is a correlation matrix, thresholds must be a vector of threshold cut points.
- **seed**: Default = NULL. Optional seed for random number generator.

**Value**

- **data**: Simulated binary data
- **r**: Input or calculated (tetrachoric) correlation matrix

**Author(s)**

Niels G Waller
Examples

```r
## Example: generating binary data to match
## an existing binary data matrix
##
## Generate correlated scores using factor analysis model
## X <- Z *L' + U*D
## Z is a vector of factor scores
## L is a factor loading matrix
## U is a matrix of unique factor scores
## D is a scaling matrix for U

N <- 5000

# Generate data from a single factor model
# factor patter matrix
L <- matrix( rep(.707, 5), nrow = 5, ncol = 1)

# common factor scores
Z <- as.matrix(rnorm(N))

# unique factor scores
U <- matrix(rnorm(N *5), nrow = N, ncol = 5)
D <- diag(as.vector(sqrt(1 - L^2)))

# observed scores
X <- Z %*% t(L) + U %*% D

cat("\nCorrelation of continuous scores\n")
print(round(cor(X),3))

# desired difficulties (i.e., means) of
# the dichotomized scores
difficulties <- c(.2, .3, .4, .5, .6)

# cut the observed scores at these thresholds
# to approximate the above difficulties
thresholds <- qnorm(difficulties)

Binary <- matrix(0, N, ncol(X))
for(i in 1:ncol(X)){
  Binary[X, i] <= thresholds[i], i] <- 1
}

cat("\nCorrelation of Binary scores\n")
print(round(cor(Binary), 3))

## Now use 'bigen' to generate binary data matrix with
## same correlations as in Binary

z <- bigen(data = Binary, n = N)
```
Sample Correlation Matrices from a Population Correlation Matrix

description

Sample correlation (covariance) matrices from a population correlation matrix (see Browne, 1968; Kshirsagar, 1959)

usage

corSample(R, n)

arguments

R
A population correlation matrix.

n
Sample correlation (covariance) matrices will be generated assuming a sample size of n.

value

cor.sample
Sample correlation matrix.
cov.sample
Sample covariance matrix.

author(s)

Niels Waller

references


examples

R <- matrix(c(1, .5, .5, 1), 2, 2)  # generate a sample correlation from pop R with n = 25
out <- corSample(R, n = 25)
out$cor.sample
out$cov.sample
corSmooth

Smooth a Non PD Correlation Matrix

Description

A function for smoothing a non-positive definite correlation matrix by the method of Knol and Berger (1991).

Usage

corSmooth(R, eps = 1E8 * .Machine$double.eps)

Arguments

R A non-positive definite correlation matrix.
eps Small positive number to control the size of the non-scaled smallest eigenvalue of the smoothed R matrix. Default = 1E8 * .Machine$double.eps

Value

rsmoothed A Smoothed (positive definite) correlation matrix.

Author(s)

Niels Waller

References


Examples

```r
## choose eigenvalues such that R is NPD
l <- c(3.0749126, 0.9328397, 0.5523868, 0.4408609, -0.0010000)

## Generate NPD R
R <- genCorr(eigenval = l, seed = 123)
print(eigen(R)$values)

#> [1] 3.0749126  0.9328397  0.5523868  0.4408609  -0.0010000

## Smooth R
Rsm<-corSmooth(R, eps = 1E8 * .Machine$double.eps)
print(eigen(Rsm)$values)

#> [1] 3.074184e+00  9.326669e-01  5.523345e-01  4.408146e-01  2.219607e-08
```
**d2r**

*Convert Degrees to Radians*

**Description**

A simple function to convert degrees to radians

**Usage**

\[ d2r(deg) \]

**Arguments**

- **deg**
  
  Angle in degrees.

**Value**

Angle in radians.

**Examples**

\[ d2r(90) \]

---

**eigGen**

*Generate eigenvalues for R matrices with underlying component structure*

**Description**

Generate eigenvalues for R matrices with underlying component structure

**Usage**

\[ eigGen(nDimensions = 15, nMajorFactors = 5, PrcntMajor = 0.8, threshold = 0.5) \]

**Arguments**

- **nDimensions**
  
  Total number of dimensions (variables).

- **nMajorFactors**
  
  Number of major factors.

- **PrcntMajor**
  
  Percentage of variance accounted for by major factors.

- **threshold**
  
  Minimum difference in eigenvalues between the last major factor and the first minor factor.

**Value**

A vector of eigenvalues that satisfies the above criteria.
Author(s)

Niels Waller

Examples

```r
## Example
set.seed(323)
ndim <- 25  # number of dimensions
nmaj <- 5   # number of major components
pmaj <- 0.70 # percentage of variance accounted for
# by major components
thresh <- 1 # eigenvalue difference between last major component
# and first minor component

L <- eigGen(nDimensions = ndim, nMajorFactors = nmaj,
            PrcntMajor = pmaj, threshold = thresh)

maxy <- max(L+1)

plotTitle <- paste("n Dimensions = ", ndim,
                        ", n Major Factors = ", nmaj,
                        "\n% Variance Major Factors = ", pmaj*100,
                        """)

plot(1:length(L), L, type = "b",
     main = plotTitle,
     ylim = c(0, maxy),
     xlab = "Dimensions",
     ylab = "Eigenvalues",
     cex.main = .9)
```

Find OLS Regression Coefficients that Exhibit Enhancement

---

Find OLS regression coefficients that exhibit a specified degree of enhancement.

Usage

```r
enhancement(R, br, rr)
```
Arguments

\( \mathbf{R} \)  
Predictor correlation matrix.

\( \mathbf{br} \)  
Model R-squared = \( \mathbf{b}' \mathbf{r} \). That is, \( \mathbf{br} \) is the model coefficient of determination:
\[ \mathbf{b}' \mathbf{R} \mathbf{b} = \mathbf{R}^2 = \mathbf{br} \]

\( \mathbf{rr} \)  
Sum of squared predictor-criterion correlations (\( r_{xy} \)). That is, \( \mathbf{rr} = \mathbf{r}' \mathbf{r} = \text{Sum}(r_{xy}^2) \)

Value

\( \mathbf{b} \)  
Vector of standardized regression coefficients.

\( \mathbf{r} \)  
Vector of predictor-criterion correlations.

Author(s)

Niels Waller

References


Examples

```r
## Example: For a given predictor correlation matrix (R) generate
## regression coefficient vectors that produce enhancement (br - rr > 0)

## Predictor correlation matrix
R <- matrix(c(1, .5, .25,
              .5, 1, .30,
              .25, .30, 1), 3, 3)

## Model coefficient of determination
Rsq <- .60

output <- enhancement(R, br = Rsq, rr = .40)

r <- output$r
b <- output$b

## Standardized regression coefficients
print(t(b))

## Predictor-criterion correlations
print(t(r))

## Coefficient of determinations (b'r)
print(t(b) %*% r)

## Sum of squared correlations (r'r)
print(t(r) %*% r)
```
fungible 

Generate Fungible Regression Weights

Description

Generate fungible weights for OLS Regression Models.

Usage

fungible(R.X, rxy, r.yhata.yhatb, sets, print = TRUE)

Arguments

- R.X: p x p Predictor correlation matrix.
- rxy: p x 1 Vector of predictor-criterion correlations.
- r.yhata.yhatb: Correlation between least squares (yhatb) and alternate-weight (yhata) composites.
- sets: Number of returned sets of fungible weights.
- print: Logical, if TRUE then print 5-point summaries of alternative weights.

Value

- a: Number of sets x p matrix of fungible weights.
- k: Number of sets x p matrix of k weights.
- b: p x 1 vector of LS weights.
- u: p x 1 vector of u weights.
- r.yhata.yhatb: Correlation between yhata and yhatb.
- r.y.yhatb: Correlation between y and yhatb.
- cov.a: Expected covariance matrix for a.
- cor.a: Expected correlation matrix for a.

Author(s)

Niels Waller

References

fungibleExtrema

Examples

```r
## Predictor correlation matrix
R.X <- matrix(c(1.00, 0.56, 0.77,
                 0.56, 1.00, 0.73,
                 0.77, 0.73, 1.00), 3, 3)

## vector of predictor-criterion correlations
rx.y <- c(0.39, 0.34, 0.38)

## OLS standardized regression coefficients
b <- solve(R.X) %*% r.x.y

## Coefficient of determination (Rsq)
OLSRSQ <- t(b) %*% R.X %*% b

## theta controls the correlation between
## yhatb: predicted criterion scores using OLS coefficients
## yhata: predicted criterion scores using alternate weights
theta <- .01

## desired correlation between yhata and yhatb
r.yhata.yhatb <- sqrt(1 - (theta)/OLSRSQ)

## number of returned sets of fungible weight vectors
Nsets <- 50

output <- fungible(R.X, r.x.y, r.yhata.yhatb, sets = Nsets, print = TRUE)
```

Description

Locate extrema of fungible regression weights.

Usage

`fungibleExtrema(R.X, r.x.y, r.yhata.yhatb, Nstarts, MaxMin)`

Arguments

- `R.X` p x p Predictor variable correlation matrix.
- `r.x.y` p x 1 Vector of predictor-criterion correlations.
- `r.yhata.yhatb` Correlation between least squares (yhatb) and alternate-weight (yhata) composites.
- `Nstarts` Maximum number of (max) minimizations from random starting configurations.
- `MaxMin` Character: "Max" = maximize cos(a,b); "Min" = minimize cos(a,b).
fungibleExtrema

Value

- **cos.ab**: cosine between OLS and alternate weights.
- **a**: extrema of fungible weights.
- **k**: k weights.
- **z**: z weights: a normalized random vector.
- **b**: OLS weights.
- **u**: p x 1 vector of u weights.
- **r.yhata.yhatb**: Correlation between yhata and yhatb.
- **r.y.yhatb**: Correlation between y and yhatb.
- **gradient**: Gradient of converged solution.

Author(s)

Niels Waller and Jeff Jones

References


Examples

```r
## Not run:
## Example
## This is Koopman's Table 2 Example

R.X <- matrix(c(1.00, .69, .49, .39,
                .69, 1.00, .38, .19,
                .49, .38, 1.00, .27,
                .39, .19, .27, 1.00),4,4)

b <- c(.39, .22, .02, .43)
ryxy <- R.X %*% b
OLSRSQ <- t(b) %*% R.X %*% b

## theta <- .02
## r.yhata.yhatb <- sqrt( 1 - (theta)/OLSRSQ)

r.yhata.yhatb <- .90
set.seed(5)
output <- fungibleExtrema(R.X, rxy, r.yhata.yhatb, Nstarts = 500,
                          MaxMin = "Min")
```
fungibleL

## Description
Generate fungible weights for Logistic Regression Models.

## Usage
```r
fungibleL(x, y, Nsets = 1000, method = "LLM", RsqDelta = NULL, rLaLb = NULL, s = .3, Print = TRUE)
```

## Arguments
- **x**: An n by nvar matrix of predictor scores without the leading column of ones.
- **y**: An n by 1 vector of dichotomous criterion scores.
- **Nsets**: The desired number of fungible coefficient vectors.
- **RsqDelta**: The desired decrement in the pseudo-R-squared - used when method = "LLM".
- **rLaLb**: The desired correlation between the logits - used when method = "EM".
- **s**: Scale factor for random deviates. s controls the range of random start values for the optimization routine. Recommended 0 <= s < 1. Default: s = 0.3.
- **Print**: Boolean (TRUE/FALSE) for printing output summary.

## Details
fungibleL provides two methods for evaluating parameter sensitivity in logistic regression models by computing fungible logistic regression weights. For additional information on the underlying theory of these methods see Jones and Waller (in press).
Value

model  A glm model object.
call  The function call to glm().
ftable  A data frame with the mle estimates and the minimum and maximum fungible coefficients.
lnLML  The maximum likelihood log likelihood value.
lnLf  The decremented, fungible log likelihood value.
pseudoRsq  The pseudo R-squared.
fungibleRsq  The fungible pseudo R-squared.
fungiblea  The Nsets by Nvar + 1 matrix of fungible (alternate) coefficients.
rlalb  The correlation between the logits.
maxPosCoefChange  The maximum positive change in a single coefficient holding all other coefficients constant.
maxNegCoefChange  The maximum negative change in a single coefficient holding all other coefficients constant.

Author(s)

Jeff Jones and Niels Waller

References


Examples

```r
# low : low birth rate (0 >= 2500 grams, 1 < 2500 grams)
# race: 1 = white, 2 = black, 3 = other
# ftv : number of physician visits during the first trimester

library(MASS)
attach(birthwt)

race <- factor(race, labels = c("white", "black", "other"))
predictors <- cbind(lwt, model.matrix(~ race)[, -1])

# compute mle estimates
BWght.out <- glm(low ~ lwt + race, family = "binomial")

# compute fungible coefficients
fungible.LLM <- fungibleL(X = predictors, y = low, method = "LLM",
                           Nsets = 10, RsqDelta = .005, s = .3)
```

fungibleR

# Compare with Table 2.3 (page 38) Hosmer Jr, D. W. & Lemeshow, S. (2000).

print(summary(EM$out))
print(fungible.LLM$call)
print(fungible.LLM$ftable)
cat("\nMLE log likelihood = ", fungible.LLM$lnML,
   "fungible log likelihood = ", fungible.LLM$lnL)
cat("\nPseudo Rsq = ", round(fungible.LLMpseudoRsq, 3))
cat("\nfungible Pseudo Rsq = ", round(fungible.LLMfungibleRsq, 3))

fungible.EM <- fungibleL(X = predictors, y = low, method = "EM",
   Nsets = 10, rLaLb = 0.99)

print(fungible.EM$call)
print(fungible.EM$ftable)
cat("\nrLaLb = ", round(fungible.EM$rLaLb, 3))

fungibleR

Generate Fungible Correlation Matrices

Description

Generate fungible correlation matrices. For a given vector of standardized regression coefficients, Beta, and a user-defined R-squared value, Rsq, find predictor correlation matrices, R, such that Beta’ R Beta = Rsq. The size of the smallest eigenvalue (Lp) of R can be defined.

Usage

fungibleR(R, Beta, Lp = .00, eps = 1e-08, Print.Warnings = TRUE)

Arguments

R A p x p predictor correlation matrix.
Beta A p x 1 vector of standardized regression coefficients.
Lp Controls the size of the smallest eigenvalue of RstarLp.
eps Convergence criterion.
Print.Warnings Logical, default = TRUE. When TRUE, convergence failures are printed.

Value

R Any input correlation matrix that satisfies Beta’ R Beta = Rsq.
Beta Input vector of std reg coefficients.
Rstar A random fungible correlation matrix.
A fungible correlation matrix with a fixed minimum eigenvalue (RstarLp can be PD, PSD, or ID).

Scaling constant for Rstar.

Scaling constant for RstarLp.

Vector in the null space of vecp(Beta' * Beta).

Left null space of Beta.

Frobenius norm ||R - Rstar||_F.

Frobenius norm ||R - RstarLp||_F given random Delta.

An integer code. 0 indicates successful completion.

Author(s)

Niels Waller

References


Examples

library(fungible)

## ===== Example 1 =====
## Generate 5 random PD fungible R matrices
## that are consistent with a user-defined predictive
## structure: B' * Rxx * B = .30

set.seed(246)
## Create a 5 x 5 correlation matrix, R, with all r_ij = .25
R.ex1 <- matrix(.25, 5, 5)
diag(R.ex1) <- 1

## create a 5 x 1 vector of standardized regression coefficients,
## Beta.ex1
Beta.ex1 <- c(-.4, -.2, 0, .2, .4)
cat("\nModel Rsq = ", t(Beta.ex1) %*% R.ex1 %*% Beta.ex1)

## Generate fungible correlation matrices, Rstar, with smallest
## eigenvalues > 0.
Rstar.list <- list(rep(99,5))
i <- 0
while(i <= 5){
  out <- fungibleR(R = R.ex1, Beta = Beta.ex1, Lp = 1e-8, eps = 1e-8,
                   Print.Warnings = TRUE)
  if(out$converged==0){
    i <- i + 1
  }
Rstar.list[[i]] <- out$Rstar
}
}

## Check Results
cat("\n *** Check Results ***")
for(i in 1:5){
  cat("\n\n\n-------------------------------------------------------------")
  cat("\nRstar", i, "\n")
  print(round(Rstar.list[[i]], 2),)
  cat("\neigenvalues of Rstar", i, "\n")
  print(eigen(Rstar.list[[i]])$values)
  cat("\nBeta' Rstar", i, "Beta = ",
    t(Beta.ex1) %%% Rstar.list[[i]] %%% Beta.ex1)
}

## ===== Example 2 =====
## Generate a PD fungible R matrix with a fixed smallest
## eigenvalue (lp).

## Create a 5 x 5 correlation matrix, R, with all r_ij = .5
R <- matrix(.5, 5, 5)
diag(R) <- 1

## create a 5 x 1 vector of standardized regression coefficients, Beta,
## such that Beta_i = .1 for all i
Beta <- rep(.1, 5)

## Generate fungible correlation matrices (a) Rstar and (b) Rstarlp.
## Set lp = 0.12345678 so that the smallest eigenvalue (lp) of Rstarlp
## = 0.12345678
out <- fungibleR(R, Beta, lp = 0.12345678, eps = 1e-10, Print.Warnings = TRUE)

## print R
cat("\nR: a user-specified seed matrix")
print(round(out$R,3))

## Rstar
cat("\nRstar: A random fungible correlation matrix for R")
print(round(out$Rstar,3))

cat("\nCoefficient of determination when using R\n")
print( t(Beta) %%% R %%% Beta )

cat("\nCoefficient of determination when using Rstar\n")
print( t(Beta) %%% out$Rstar %%% Beta )

## Eigenvalues of R
cat("\nEigenvalues of R\n")
print(round(eigen(out$R)$values, 9))
## Eigenvalues of R

cat("\nEigenvalues of Rstar\n")
print(round(eigen(out$Rstar$)$values, 9))

## What is the Frobenius norm (Euclidean distance) between
## R and Rstar

cat("\nFrobenious norm ||R - Rstar||\n")
print(out$FrobNorm)

## RstarLp is a random fungible correlation matrix with
## a fixed smallest eigenvalue of 0.12345678

cat("\nRstarLp: a random fungible correlation matrix with a user-defined
smallest eigenvalue\n")
print(round(out$RstarLp, 3))

## Eigenvalues of RstarLp

cat("\nEigenvalues of RstarLp\n")
print(eigen(out$RstarLp$)$values, digits = 9)

cat("\nCoefficient of determination when using RstarLp\n")
print(t(Beta) %*% out$RstarLp %*% Beta)

## Check function convergence

if(out$converged) print("Failed to converge")

## Example 3

## This example demonstrates how fungibleR can be used
## to generate improper correlation matrices (i.e., pseudo
## correlation matrices with negative eigenvalues).

library(fungible)

## We desire an improper correlation matrix that
## is close to a user-supplied seed matrix. Create an
## interesting seed matrix that reflects a Big Five
## factor structure.

set.seed(123)
minCrossLoading <- -.2
maxCrossLoading <- .2
F1 <- c(rep(.6,5), runif(20, minCrossLoading, maxCrossLoading))
F2 <- c(runif(5, minCrossLoading, maxCrossLoading), rep(.6,5),
runif(15, minCrossLoading, maxCrossLoading))
F3 <- c(runif(10, minCrossLoading, maxCrossLoading), rep(.6,5),
runif(10, minCrossLoading, maxCrossLoading))
F4 <- c(runif(15, minCrossLoading, maxCrossLoading), rep(.6,5),
runif(15, minCrossLoading, maxCrossLoading))
F5 <- c(runif(20, minCrossLoading, maxCrossLoading), rep(.6,5))
FacMat <- cbind(F1, F2, F3, F4, F5)
R.bfi <- FacMat %*% t(FacMat)
diag(R.bfi) <- 1

## Set Beta to a null vector to inform fungibleR that we are
## not interested in placing constraints on the predictive structure
## of the fungible R matrices.
Beta <- rep(0, 25)

## We seek a NPD fungible R matrix that is close to the bfi seed matrix.
## To find a suitable matrix we generate a large number (e.g., 50000)
## fungible R matrices. For illustration purposes I will set Nmatrices
## to a smaller number: 10.
Nmatrices<-10

## Initialize a list to contain the Nmatrices fungible R objects
Rstarlp.list <- as.list( rep(0, Nmatrices ) )
## Initialize a vector for the Nmatrices Froebius norms ||R - Rstarlp||
FrobLP.vec <- rep(0, Nmatrices)

## Constraint the smallest eigenvalue of RStarlp by setting
## Lp = -.1 (or any suitably chosen user-defined value).

## Generate Nmatrices fungibleR matrices and identify the NPD correlation
## matrix that is "closest" (has the smallest Frobenious norm) to the bfi
## seed matrix.
BestR.i <- 0
BestFrob <- 99
i <- 0

set.seed(1)
while(i < Nmatrices){
  out<-fungibleR(R = R.bfi, Beta, Lp = -.1, eps=1e-10)
  ## retain solution if algorithm converged
  if(out$converged == 0)
    { i<- i + 1
      ## print progress
      cat("Generating matrix ", i, " Current minimum ||R - Rstarlp|| = ",BestFrob)
      tmp <- FrobLP.vec[i] <- out$FrobNormLp #Frobenious Norm ||R - Rstarlp||
      Rstarlp.list[[i]]<-out$Rstarlp
      if( tmp < BestFrob )
        { BestR.i <- i  # matrix with lowest ||R - Rstarlp||
          BestFrob <- tmp  # value of lowest ||R - Rstarlp||
        }
    }
}

# CloseR is an improper correlation matrix that is close to the seed matrix.
CloseR<-Rstarlp.list[[BestR.i]]

plot(1:25, eigen(R.bfi)$values,
     type = "b",
     col = "red",
     pch = 20,
     main = "Eigenvalues of Fungible R Matrices",
     xlab = "Eigenvalue Index",
     ylab = "Eigenvalue Value"
)

## For illustration purposes I will set Nmatrices
to a smaller number: 10.
Nmatrices<-10

## Initialize a list to contain the Nmatrices fungible R objects
Rstarlp.list <- as.list( rep(0, Nmatrices ) )
## Initialize a vector for the Nmatrices Froebius norms ||R - Rstarlp||
FrobLP.vec <- rep(0, Nmatrices)

## Constraint the smallest eigenvalue of RStarlp by setting
## Lp = -.1 (or any suitably chosen user-defined value).

## Generate Nmatrices fungibleR matrices and identify the NPD correlation
## matrix that is "closest" (has the smallest Frobenious norm) to the bfi
## seed matrix.
BestR.i <- 0
BestFrob <- 99
i <- 0

set.seed(1)
while(i < Nmatrices){
  out<-fungibleR(R = R.bfi, Beta, Lp = -.1, eps=1e-10)
  ## retain solution if algorithm converged
  if(out$converged == 0)
    { i<- i + 1
      ## print progress
      cat("Generating matrix ", i, " Current minimum ||R - Rstarlp|| = ",BestFrob)
      tmp <- FrobLP.vec[i] <- out$FrobNormLp #Frobenious Norm ||R - Rstarlp||
      Rstarlp.list[[i]]<-out$Rstarlp
      if( tmp < BestFrob )
        { BestR.i <- i  # matrix with lowest ||R - Rstarlp||
          BestFrob <- tmp  # value of lowest ||R - Rstarlp||
        }
    }
}

# CloseR is an improper correlation matrix that is close to the seed matrix.
CloseR<-Rstarlp.list[[BestR.i]]

plot(1:25, eigen(R.bfi)$values,
     type = "b",
     col = "red",
     pch = 20,
     main = "Eigenvalues of Fungible R Matrices",
     xlab = "Eigenvalue Index",
     ylab = "Eigenvalue Value"
**Description**

Uses the Marsaglia and Olkin (1984) algorithm to generate correlation matrices with user-defined eigenvalues.

**Usage**

```r
gencorr(eigenval, seed='rand')
```

**Arguments**

- `eigenval` A vector of eigenvalues that must sum to the order of the desired correlation matrix. For example: if you want a correlation matrix of order 4, then you need 4 eigenvalues that sum to 4. A warning message will display if `sum(eigenval) != length(eigenval)`
- `seed` Either a user supplied seed for the random number generator or ‘rand’ for a function generated seed. Default seed='rand'.

**Value**

Returns a correlation matrix with the eigen-stucture specified by `eigenval`.

**Author(s)**

Jeff Jones
References


Examples

```r
# Example
# Generate a correlation matrix with user-specified eigenvalues
set.seed(123)
R <- genCorr(c(2.5, 1, 1, .3, .2))

print(round(R, 2))

#> [1,] 1.00 0.08 -0.07 -0.07 0.00
#> [2,] 0.08 1.00 0.00 -0.60 0.53
#> [3,] -0.07 0.00 1.00 0.51 -0.45
#> [4,] -0.07 -0.60 0.51 1.00 -0.75
#> [5,] 0.00 0.53 -0.45 -0.75 1.00

print(eigen(R)$values)

# [1] 2.5 1.0 1.0 0.3 0.2
```

---

`kurt`  
*Calculate Univariate Kurtosis for a Vector or Matrix*

Description

Calculate univariate kurtosis for a vector or matrix (algorithm G2 in Joanes & Gill, 1998).

Usage

`kurt(x)`

Arguments

- `x`  
  Either a vector or matrix of numeric values.

Value

Kurtosis for each column in `x`. 
Author(s)
Niels Waller

References

See Also
skew

Examples
```r
x <- matrix(rnorm(1000), 100, 10)
print(kurt(x))
```

---

**monte**

*Simulate Clustered Data with User-Defined Properties*

**Description**
Function for simulating clustered data with user defined characteristics such as: within cluster indicator correlations, within cluster indicator skewness values, within cluster indicator kurtosis values, and cluster separations as indexed by each variable (indicator validities).

**Usage**
```
monte(seed, nvar, nclus, clus.size, eta2, cor.list, random.cor, skew.list, kurt.list, secor, compactness, sortmeans)
```

**Arguments**
- **seed**: Required: An integer to be used as the random number seed.
- **nvar**: Required: Number of variables to simulate.
- **nclus**: Required: Number of clusters to simulate. *Note* that number of clusters must be equal to or greater than 2.
- **clus.size**: Required: Number of objects in each cluster.
- **eta2**: Required: A vector of indicator validities that range from 0 to 1. Higher numbers produce clusters with greater separation on that indicator.
- **cor.list**: Optional: A list of correlation matrices. There should be one correlation matrix for each cluster. The first correlation matrix will represent the indicator correlations within cluster 1. The second correlation matrix will represent the indicator correlations for cluster 2. Etc.
- **random.cor**: Optional: Set to TRUE to generate a common within cluster correlation matrix.
skew.list  Optional: A list of within cluster indicator skewness values.
kurt.list  Optional: A list of within cluster indicator kurtosis values.
secor      Optional: If `random.cor = TRUE` then 'secor' determines the standard error of
            the simulated within group correlation matrices.
compactness Optional: A vector of cluster compactness parameters. The meaning of this
            option is explained Waller et al. (1999). Basically, 'compactness' allows users
            some control over cluster overlap without changing indicator validities. See the
            example below for an illustration.
sortMeans  Optional: A logical that determines whether the latent means will be sorted by
            taxon. Default = TRUE

Value

data        The simulated data. The 1st column of 'data' denotes cluster membership.
1mn         The cluster indicator means.
fl          The factor loading matrix as described in Waller, et al. 1999.
fs          The unique values of the linearized factor scores.
call        The call.
nclus       Number of clusters.
nvar        Number of variables.
cor.list    The input within cluster correlation matrices.
skew.list   The input within cluster indicator skewness values.
kurt.list   The input within cluster indicator kurtosis values.
clus.size   The number of observations in each cluster.
etta2       Vector of indicator validities.
seed        The random number seed.

Author(s)

Niels Waller

References


Examples

```r
## Example 1
## Simulating Fisher's Iris data
# The original data were reported in:
# problems. Annals of Eugenics, 7, Part II, 179-188.
# This example includes 3 clusters. Each cluster represents
# an Iris species: Setosa, Versicolor, and Virginica.
# On each species, four variables were measured: Sepal Length,
# Sepal Width, Petal Length, and Petal Width.
# The within species (cluster) correlations of the flower
# indicators are as follows:
#
# Iris Type 1:
# [1,]  1.000 0.743 0.267 0.178
# [2,]  0.743 1.000 0.278 0.233
# [3,]  0.267 0.278 1.000 0.332
# [4,]  0.178 0.233 0.332 1.000
#
# Iris Type 2
# [1,]  1.000 0.526 0.754 0.546
# [2,]  0.526 1.000 0.561 0.664
# [3,]  0.754 0.561 1.000 0.787
# [4,]  0.546 0.664 0.787 1.000
#
# Iris Type 3
# [1,]  1.000 0.457 0.864 0.281
# [2,]  0.457 1.000 0.401 0.538
# [3,]  0.864 0.401 1.000 0.322
# [4,]  0.281 0.538 0.322 1.000
#
# 'monte' expects a list of correlation matrices
#
# create a list of within species correlations
data(iris)
cormat <- cm <- lapply(split(iris[,1:4], iris[,5]), cor)

# create a list of within species indicator
# skewness and kurtosis
sk.lst <- list(c(0.120, 0.041, 0.106, 1.254),
               c(0.105, -0.363, -0.607, -0.031),
               c(0.118, 0.366, 0.549, -0.129))

kt.lst <- list(c(-0.253, 0.955, 1.022, 1.719),
               c(-0.533, -0.366, 0.048, -0.410),
               c(1.022, -0.363, -0.129, 0.048))
```

```bash
monte
```
c( 0.033, 0.706, -0.154, -0.602 )

# Generate a new sample of iris data
my.iris <- monte(seed=123, nvar = 4, nclus = 3, cor.list = cormat, 
                 clus.size = c(50, 50, 50),
                 eta2=c(0.619, 0.401, 0.941, 0.929),
                 random.cor = FALSE,
                 skew.list = sk.lst,
                 kurt.list = kt.lst,
                 secor = .3, compactness=c(1, 1, 1),
                 sortMeans = TRUE)

summary(my.iris)
plot(my.iris)

# Now generate a new data set with the sample indicator validities
# as before but with different cluster compactness values.
my.iris2<-monte(seed = 123, nvar = 4, nclus = 3, 
                 cor.list = cormat, clus.size = c(50, 50, 50),
                 eta2 = c(0.619, 0.401, 0.941, 0.929), random.cor = FALSE,
                 skew.list = sk.lst ,kurt.list = kt.lst,
                 secor = .3,
                 compactness=c(2, .5, .5),
                 sortMeans = TRUE)

summary(my.iris2)

# Notice that cluster 1 has been blow up whereas clusters 2 and 3 have been shrunk.
plot(my.iris2)

### Now compare your original results with the actual
### Fisher iris data
library(lattice)
data(iris)
super.sym <- trellis.par.get("superpose.symbol")
splom(~iris[1:4], groups = Species, data = iris,
      panel = panel.superpose,
      key = list(title = "Three Varieties of Iris",
                 columns = 3,
                 points = list(pch = super.sym$pch[1:3],
                                col = super.sym$col[1:3]),
                 text = list(c("Setosa", "Versicolor", "Virginica"))))

##############################################################

## Example 2
## Simulating data for Taxometric
## Monte Carlo Studies.

## In this four part example we will
## generate two group mixtures
## (Complement and Taxon groups)
## under four conditions.
##
## In all conditions
## base rate (BR) = .20
## 3 indicators
## indicator validities = .50
## (This means that 50 percent of the total
## variance is due to the mixture.)
##
## Condition 1:
## All variables have a slight degree
## of skewness (.10) and kurtosis (.10).
## Within group correlations = 0.00.
##
## Condition 2:
## In this condition we generate data in which the
## complement and taxon distributions differ in shape.
## In the complement group all indicators have
## skewness values of 1.75 and kurtosis values of 3.75.
## In the taxon group all indicators have skewness values
## of .50 and kurtosis values of 0.
## As in the previous condition, all within group
## correlations (nuisance covariance) are 0.00.
##
## Condition 3:
## In this condition we retain all previous
## characteristics except that the within group
## indicator correlations now equal .80
## (they can differ between groups).
##
## Condition 4:
## In this final condition we retain
## all previous data characteristics except that
## the variances of the indicators in the complement
## class are now 5 times the indicator variances
## in the taxon class (while maintaining indicator skewness,
## kurtosis, correlations, etc.).

library(lattice)
# Condition 1

```r
in.nvar <- 3  # Number of variables
in.nclus <- 2  # Number of taxa
in.seed <- 123
BR <- .20  # Base rate of higher taxon

## Within taxon indicator skew and kurtosis
in.skew.list <- list(c(.1, .1, .1), c(.1, .1, .1))
in.kurt.list <- list(c(.1, .1, .1), c(.1, .1, .1))

## Indicator validities
in.eta2 <- c(.50, .50, .50)

## Groups sizes for Population
BigN <- 100000
in.clus.size <- c(BigN*(1-BR), BR * BigN)

## Generate Population of scores with "monte"
sample.data <- monte(seed = in.seed,
   nvar=in.nvar,
   nclus = in.nclus,
   clus.size = in.clus.size,
   eta2 = in.eta2,
   skew.list = in.skew.list,
   kurt.list = in.kurt.list)

output <- summary(sample.data)

z <- data.frame(sample.data$data[,sample(1:BigN, 600, replace=FALSE),])
z[,2:4] <- scale(z[,2:4])
names(z) <- c("id","v1","v2","v3")

trellis.device()
trellis.par.set( col.whitebg() )
print(
   cloud(v3 ~ v1 * v2,
      groups = as.factor(id),data=z,
      subpanel = panel.superpose,
      zlim=c(-4, 4),
      xlim=c(-4, 4),
      ylim=c(-4, 4),
      main="",
      screen = list(z = 20, x = -70)),
      position=c(.1, .5, .5, 1), more = TRUE)
```

# Condition 2

```r
### Condition 2
### Condition 2
```
## Condition 2

### Within taxon indicator skew and kurtosis

```r
in.skew.list <- list(c(1.75, 1.75, 1.75), c(.50, .50, .50))
in.kurt.list <- list(c(3.75, 3.75, 3.75), c(0, 0, 0))
```

### Generate Population of scores with "monte"

```r
sample.data <- monte(seed = in.seed,
                     nvar = in.nvar,
                     nclus = in.nclus,
                     clus.size = in.clus.size,
                     eta2 = in.eta2,
                     skew.list = in.skew.list,
                     kurt.list = in.kurt.list)
```

```r
output <- summary(sample.data)
```

```r
z <- data.frame(sample.data$sample[1:BigN, 600, replace=FALSE])
z[2:4] <- scale(z[, 2:4])
```

```r
names(z) <- c("id", "v1", "v2", "v3")
```

```r
print(
      cloud(v3 ~ v1 * v2,
            groups = as.factor(id), data = z,
            subpanel = panel.superpose,
            zlim = c(-4, 4),
            xlim = c(-4, 4),
            ylim = c(-4, 4),
            main="",
            screen = list(z = 20, x = -70)),
      position = c(.5, .5, 1, 1), more = TRUE)
```

## Condition 3

### Set within group correlations to .80

```r
cormat <- matrix(.80, 3, 3)
diag(cormat) <- rep(1, 3)
in.cor.list <- list(cormat, cormat)
```

### Generate Population of scores with "monte"

```r
sample.data <- monte(seed = in.seed,
                      nvar = in.nvar,
                      nclus = in.nclus,
                      clus.size = in.clus.size,
                      eta2 = in.eta2,
                      skew.list = in.skew.list,
                      kurt.list = in.kurt.list,
                      cor.list = in.cor.list)
```
output <- summary(sample.data)

z <- data.frame(sample.data$sample[1:BigN, 600, replace = FALSE], )
z[, 2:4] <- scale(z[, 2:4])
names(z) <- c("id", "v1", "v2", "v3")

## trellis.device()
## trellis.par.set( col.whitebg() )
print(
  cloud(v3 ~ v1 * v2,
        groups = as.factor(id), data = z,
        subpanel = panel.superpose,
        zlim = c(-4, 4),
        xlim = c(-4, 4),
        ylim = c(-4, 4),
        main = "",
        screen = list(z = 20, x = -70),
        position = c(.1, .0, .5, .5), more = TRUE)


##############################################################
##  Condition 4
##############################################################

## Change compactness so that variance of
## complement indicators is 5 times
## greater than variance of taxon indicators

v <- ( 2 * sqrt(5))/(1 + sqrt(5))
in.compactness <- c(v, 2-v)

## Generate Population of scores with "monte"
sample.data <- monte(seed = in.seed,
                     nvar = in.nvar,
                     nclus = in.nclus,
                     clus.size = in.clus.size,
                     eta2 = in.eta2,
                     skew.list = in.skew.list,
                     kurt.list = in.kurt.list,
                     cor.list = in.cor.list,
                     compactness = in.compactness)

output <- summary(sample.data)

z <- data.frame(sample.data$sample[1:BigN, 600, replace = FALSE], )
z[, 2:4] <- scale(z[, 2:4])
names(z) <- c("id", "v1", "v2", "v3")
print(
  cloud(v3 ~ v1 * v2,
        groups = as.factor(id), data = z,
        subpanel = panel.superpose,
monte1

Simulate Multivariate Non-normal Data by Vale & Maurelli (1983)

Method

Description

Function for simulating multivariate nonnormal data by the methods described by Fleishman (1978) and Vale & Maurelli (1983).

Usage

monte1(seed, nvar, nsub, cormat, skewvec, kurtvec)

Arguments

- **seed**: An integer to be used as the random number seed.
- **nvar**: Number of variables to simulate.
- **nsub**: Number of simulated subjects (response vectors).
- **cormat**: The desired correlation matrix.
- **skewvec**: A vector of indicator skewness values.
- **kurtvec**: A vector of indicator kurtosis values.

Value

- **data**: The simulated data.
- **call**: The call.
- **nsub**: Number of subjects.
- **nvar**: Number of variables.
- **cormat**: The desired correlation matrix.
- **skewvec**: The desired indicator skewness values.
- **kurtvec**: The desired indicator kurtosis values.
- **seed**: The random number seed.

Author(s)

Niels Waller
References


See Also

`monte, summary.monte, summary.monteQ`

Examples

```r
## Generate dimensional data for 4 variables.
## All correlations = .60; all variable
## skewness = 1.75;
## all variable kurtosis = 3.75

cormat <- matrix(.60, 4, 4)
diag(cormat) <- 1

nontaxon.dat <- monte1(seed = 123, nsub = 100000, nvar = 4, skewvec = rep(1.75, 4),
                       kurtvec = rep(3.75, 4), cormat = cormat)

print(cor(nontaxon.dat$data), digits = 3)
print(apply(nontaxon.dat$data, 2, skew), digits = 3)
print(apply(nontaxon.dat$data, 2, kurt), digits = 3)
```

normalCor

*Compute Normal-Theory Covariances for Correlations*

Description

Compute normal-theory covariances for correlations

Usage

```
normalCor(R, Nobs)
```

Arguments

- **R**: A p x p matrix of correlations.
- **Nobs**: Number of observations.

Value

A normal-theory covariance matrix of correlations.
Author(s)

Jeff Jones and Niels Waller

References


See Also

`adfCor`

Examples

```r
data(Harman23.cor)
normalCor(Harman23.cor$cov, Nobs = 305)
```

---

**plot.monte**  
*Plot Method for Class Monte*

Description

plot method for class "monte"

Usage

```r
## S3 method for class 'monte'
plot(x, ...)
```

Arguments

- `x` An object of class 'monte', usually, a result of a call to `monte`.
- `...` Optional arguments passed to plotting function.

Value

The function `plot.monte` creates a scatter plot of matrices plot (a splom plot). Cluster membership is denoted by different colors in the plot.

Examples

```r
#plot(monte.object)
```
**r2d** 

*Convert Radians to Degrees*

**Description**

Convert radian measure to degrees.

**Usage**

\[ r2d(\text{radian}) \]

**Arguments**

- \( \text{radian} \)  
  Radian measure of an angle

**Value**

Degree measure of an angle

**Examples**

\[ r2d(.5\pi) \]

---

**rarc** 

*Rotate Points on the Surface on an N-Dimensional Ellipsoid*

**Description**

Rotate between two points on the surface on an n-dimensional ellipsoid. The hyper-ellipsoid is composed of all points, B, such that B' Rxx B = Rsq. Vector B contains standardized regression coefficients.

**Usage**

\[ rarc(\text{Rxx}, \text{Rsq}, \text{b1}, \text{b2}, \text{Npoints}) \]

**Arguments**

- \( \text{Rxx} \)  
  Predictor correlation matrix.
- \( \text{Rsq} \)  
  Model coefficient of determination.
- \( \text{b1} \)  
  First point on ellipsoid. If b1 and b2 are scalars then choose scaled eigenvectors v[\text{b1}] and v[\text{b2}] as the start and end vectors.
- \( \text{b2} \)  
  Second point on ellipsoid. If b1 and b2 are scalars then choose scaled eigenvectors v[\text{b1}] and v[\text{b2}] as the start and end vectors.
- \( \text{Npoints} \)  
  Generate “Npoints” +1 OLS coefficient vectors between b1 and b2.
Value

\[ b \]

N+1 sets of OLS coefficient vectors between b1 and b2.

Author(s)

Niels Waller and Jeff Jones.

References


Examples

```r
## Example
## GRE/GPA Data
##------------------------------
R <- Rxx <- matrix(c(1.00, .56, .77, .56, 1.00, .73, .77, .73, 1.00), 3, 3)

## GPA validity correlations
rxy <- c(.39, .34, .38)
b <- solve(Rxx) %*% rxy

Rsq <- t(b) %*% Rxx %*% b
N <- 200

b <- rarc(Rxx = R, Rsq, b1 = 1, b2 = 3, Npoints = N)

## compute validity vectors
r <- Rxx %*% b
N <- N + 1
Rsq.r <- Rsq.unit <- rep(0, N)

for(i in 1:N){
  ## eval performance of unit weights
  Rsq.unit[i] <- (t(sign(r[,i])) %*% r[,i])^2 / (t(sign(r[,i])) %*% R %*% sign(r[,i]))

  ## eval performance of correlation weights
  Rsq.r[i] <- (t(r[,i]) %*% r[,i])^2 /(t(r[,i]) %*% R %*% r[,i])
}

cat("\nAverage relative performance of unit weights across elliptical arc:",
     round(mean(Rsq.unit)/Rsq,3) )
cat("\n\nAverage relative performance of r weights across elliptical arc:",
     round(mean(Rsq.r)/Rsq,3) )
```
rcone

Generate a Cone of Regression Coefficient Vectors

Description

Compute a cone of regression vectors with a constant R-squared around a target vector.

Usage

rcone(R, Rsq, b, axis1, axis2, deg, Npoints = 360)

Arguments

R Predictor correlation matrix.
Rsq Coefficient of determination.
b Target vector of OLS regression coefficients.
axis1 1st axis of rotation plane.
axis2 2nd axis of rotation plane.
deg All vectors b.i will be 'deg' degrees from b.
Npoints Number of rotation vectors, default = 360.

Value

b.i Npoints values of b.i

Author(s)

Niels Waller and Jeff Jones
References


Examples

R <- matrix(.5, 4, 4)
diag(R) <- 1

Npoints <- 1000
Rsq <- .40
NumDeg <- 20
V <- eigen(R)$vectors

## create b parallel to v[,3]
## rotate in the 2 - 4 plane
b <- V[,3]
bsq <- t(b) %*% R %*% b
b <- b * sqrt(Rsq/bsq)
b.i <- rcone(R, Rsq, b, V[,2], V[,4], deg = NumDeg, Npoints)

t(b.i[,1]) %*% R %*% b.i[,1]
t(b.i[,25]) %*% R %*% b.i[,25]

rcor

Generate Random PSD Correlation Matrices

Description

Generate random PSD correlation matrices.

Usage

rcor(Nvar)

Arguments

Nvar

An integer that determines the order of the random correlation matrix.

Details

rcor generates random PSD correlation matrices by (1) generating Nvar squared random normal deviates, (2) scaling the deviates to sum to Nvar, and then (3) placing the scaled values into a diagonal matrix L. Next, (4) an Nvar x Nvar orthogonal matrix, Q, is created by performing a QR decomposition of a matrix, M, that contains random normal deviates. (5) A PSD covariance matrix, C, is created from Q L Q^T and then (6) scaled to a correlation metric.
rellipsoid

Value
A random correlation matrix.

Author(s)
Niels Waller

See Also
genCorr

Examples
R <- rcor(4)
print(R)

rellipsoid

Generate Uniformly Spaced OLS Regression Coefficients that Yield a User-Supplied R-Squared Value

Description
Given predictor matrix R, generate OLS regression coefficients that yield a user-supplied R-Squared value. These regression coefficient vectors will be uniformly spaced on the surface of a (hyper) ellipsoid.

Usage
rellipsoid(R, Rsq, Npoints)

Arguments
R A p x p predictor correlation matrix.
Rsq A user-supplied R-squared value.
Npoints Desired number of generated regression vectors.

Value
b A p x Npoints matrix of regression coefficients

Author(s)
Niels Waller and Jeff Jones.

Examples

```r
## generate uniformly distributed regression vectors
## on the surface of a 14-dimensional ellipsoid
N <- 10000
Rsq <- .21

# Correlations from page 224 WAIS-III manual
# The Psychological Corporation (1997).
wais3 <- matrix(
  c(1, .76, .58, .43, .75, .75, .42, .54, .41, .57, .64, .54, .50, .53,
    .76, 1, .57, .36, .69, .71, .45, .52, .36, .63, .68, .51, .47, .54,
    .58, .57, 1, .45, .65, .60, .47, .48, .43, .59, .60, .49, .56, .47,
    .43, .36, .45, 1, .37, .40, .60, .30, .32, .34, .35, .28, .35, .29,
    .75, .69, .65, .37, 1, .70, .44, .54, .34, .59, .62, .54, .45, .50,
    .75, .71, .60, .40, .70, 1, .42, .51, .44, .53, .60, .50, .52, .44,
    .42, .45, .47, .60, .44, .42, 1, .46, .49, .47, .43, .27, .50, .42,
    .54, .52, .48, .30, .54, .51, .46, 1, .45, .50, .58, .55, .53, .56,
    .41, .36, .43, .32, .34, .44, .49, .45, 1, .47, .49, .41, .70, .38,
    .57, .63, .59, .34, .59, .53, .47, .50, .47, 1, .63, .62, .58, .66,
    .64, .68, .60, .35, .62, .60, .43, .58, .49, .63, 1, .59, .50, .59,
    .54, .51, .49, .28, .54, .50, .27, .55, .41, .62, .59, 1, .48, .53,
    .50, .47, .56, .35, .45, .52, .50, .53, .70, .58, .50, .48, 1, .51,
    .53, .54, .47, .29, .50, .44, .42, .56, .38, .66, .59, .53, .51, 1),
  nrow = 14, ncol = 14)

R <- wais3[1:6, 1:6]
b <- rellipsoid(R, Rsq, Npoints = N)
b <- b$b

# plot(b[1,], b[2,])
```

**rgivens**

*Generate Correlation Matrices with Specified Eigenvalues*

**Description**

rgivens generates correlation matrices with user-specified eigenvalues via a series of Givens rotations by methods described in Bendel & Mickey (1978) and Davis & Higham (2000).

**Usage**

```r
rgivens(eigs, Seed = NULL)
```
rGivens

Arguments

eigs A vector of eigenvalues that must sum to the order of the desired correlation matrix. A fatal error will occur if sum(eigs) != length(eigs).

Seed Either a user supplied seed for the random number generator or ‘NULL’ for a function generated seed. Default Seed = ‘NULL’.

Value

R A correlation matrix with desired spectrum.

Frob The Frobenius norm of the difference between the initial and final matrices with the desired spectrum.

convergence (Logical) TRUE if rGivens converged to a feasible solution, otherwise FALSE.

References


Examples

```r
## Example
## Generate a correlation matrix with user-specified eigenvalues

out <- rGivens(c(2.5, 1, 1, .3, .2), Seed = 123)

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.

#> eigen(out$R)$values
# [,1] 2.5 1.0 1.0 0.3 0.2

correlation matrix with desired spectrum.
rMAP

Generate Correlation Matrices with Specified Eigenvalues

Description

rMAP uses the method of alternating projections (MAP) to generate correlation matrices with specified eigenvalues.

Usage

rMAP(eigenval, eps, maxits, Seed = NULL)

Arguments

eigenval A vector of eigenvalues that must sum to the order of the desired correlation matrix. A fatal error will occur if sum(eigenval) != length(eigenval).
eps Convergence criterion. Default = 1e-12.
maxits Maximum number of iterations of MAP.
Seed Either a user supplied seed for the random number generator or 'NULL' for a function generated seed. Default Seed = 'NULL'.

Value

R A correlation matrix with the desired spectrum.
evals Eigenvalues of the returned matrix, R.
convergence (Logical) TRUE if MAP converged to a feasible solution, otherwise FALSE.

Author(s)

Niels Waller

References

Examples

```r
## Example
## Generate a correlation matrix with user-specified eigenvalues

R <- rMAP(c(2.5, 1, 1, .3, .2), Seed = 123)$R
print(R, 2)

# [1,]  1.000 0.5355 -0.746 -0.0688 -0.545
# [2,]  0.535 1.0000 -0.671 -0.0016 -0.056
# [3,] -0.746 -0.6711 1.000  0.0608  0.298
# [4,] -0.069 -0.0016  0.061  1.0000  0.002
# [5,] -0.545 -0.0564  0.298  0.0020  1.000

eigen(R)$values
# [1] 2.5 1.0 1.0 0.3 0.2
```

---

**seBeta**

*Standard Errors and CIs for Standardized Regression Coefficients*

**Description**

Computes Normal Theory and ADF Standard Errors and CIs for Standardized Regression Coefficients

**Usage**

```r
seBeta(x, y, cov.x = NULL, cov.xy = NULL, var.y = NULL, nobs = NULL, alpha = 0.05, estimator = "ADF", digits = 3)
```

**Arguments**

- `x` Matrix of predictor scores.
- `y` Vector of criterion scores.
- `cov.x` Covariance or correlation matrix of predictors.
- `cov.xy` Vector of covariances or correlations between predictors and criterion.
- `var.y` Criterion variance.
- `nobs` Number of observations.
- `alpha` Desired Type I error rate; default = .05.
- `estimator` 'ADF' or 'Normal' confidence intervals - requires raw X and raw y; default = 'ADF'.
- `digits` Number of significant digits to print; default = 3.
seBetaCor

**Value**

- `cov.Beta`: Normal theory or ADF covariance matrix of standardized regression coefficients.
- `se.Beta`: standard errors for standardized regression coefficients.
- `alpha`: desired Type-I error rate.
- `CI.Beta`: Normal theory or ADF (1-alpha)% confidence intervals for standardized regression coefficients.
- `estimator`: estimator = "ADF" or "Normal".

**Author(s)**

Jeff Jones and Niels Waller

**References**


**Examples**

```r
library(MASS)

set.seed(123)

R <- matrix(.5, 3, 3)
diag(R) <- 1
X <- mvrnorm(n = 200, mu = rep(0, 3), Sigma = R, empirical = TRUE)
Beta <- c(.2, .3, .4)
y <- X %*% Beta + .64 * scale(rnorm(200))
seBeta(X, y, Nobs = 200, alpha = .05, estimator = 'ADF')

# 95% CIs for Standardized Regression Coefficients:
#
#   lbound estimate ubound
# beta_1  0.104  0.223  0.341
# beta_2  0.245  0.359  0.473
# beta_3  0.245  0.360  0.476
```

**Description**

Computes Normal Theory and ADF Standard Errors and CIs for Standardized Regression Coefficients from Correlations.
Usage

```r
seBetaCor(R, rxy, Nobs, alpha = .05, digits = 3, covmat = 'normal')
```

Arguments

- **rxy**: A p x 1 vector of predictor-criterion correlations.
- **Nobs**: Number of observations.
- **alpha**: Desired Type I error rate; default = .05.
- **digits**: Number of significant digits to print; default = 3.
- **covmat**: String = 'normal' (the default) or a (p+1)p/2 x (p+1)p/2 covariance matrix of correlations. The default option computes an asymptotic covariance matrix under the assumption of multivariate normal data. Users can supply a covariance matrix under asymptotic distribution free (ADF) or elliptical distributions when available.

Value

- **cov.Beta**: Covariance matrix of standardized regression coefficients.
- **se.Beta**: Vector of standard errors for the standardized regression coefficients.
- **alpha**: Type-I error rate.
- **CI.Beta**: (1-alpha)% confidence intervals for standardized regression coefficients.

Author(s)

Jeff Jones and Niels Waller

References


Examples

```r
R <- matrix(c(1.0000, 0.3511, 0.3661,
             0.3511, 1.0000, 0.4359,
             0.3661, 0.4359, 1.0000), 3, 3)

rxy <- c(0.5820, 0.6997, 0.7621)
Nobs <- 46
out <- seBetaCor(R = R, rxy = rxy, Nobs = Nobs)
```
Description

Computes Normal Theory Covariance Matrix and Standard Errors for Standardized Regression Coefficients for Fixed Predictors

Usage

seBetaFixed(X, y, cov.x = NULL, cov.xy = NULL, var.y = NULL, var.error = NULL, Nobs = NULL)

Arguments

X  Matrix of predictor scores.
y  Vector of criterion scores.
cov.x  Covariance or correlation matrix of predictors.
cov.xy  Vector of covariances or correlations between predictors and criterion.
var.y  Criterion variance.
var.error  Optional argument to supply the error variance: var(y - yhat).
Nobs  Number of observations.

Value

cov.Beta  Normal theory covariance matrix of standardized regression coefficients for fixed predictors.
se.Beta  Standard errors for standardized regression coefficients for fixed predictors.

Author(s)

Jeff Jones and Niels Waller

References

skew

See Also

seBeta

Examples

```r
## We will generate some data and pretend that the Predictors are being held fixed
library(MASS)
R <- matrix(.5, 3, 3); diag(R) <- 1
Beta <- c(.2, .3, .4)

rm(list = ".Random.seed", envir = globalenv()); set.seed(123)
X <- mvrnorm(n = 200, mu = rep(0, 3), Sigma = R, empirical = TRUE)
y <- X %*% Beta + .64*scale(rnorm(200))
seBetaFixed(X, y)

# $covBeta
#   b1   b2   b3
# b1  0.003275127 -0.001235665 -0.001274303
# b2 -0.001235665  0.003037100 -0.001491736
# b3 -0.001274303 -0.001491736  0.002830157
#
# $seBeta
#   b1   b2   b3
# 0.05722872 0.05510989 0.05319922

## you can also supply covariances instead of raw data
seBetaFixed(cov.x = cov(X), cov.xy = cov(X, y), var.y = var(y), Nobs = 200)

# $covBeta
#   b1   b2   b3
# b1  0.003275127 -0.001235665 -0.001274303
# b2 -0.001235665  0.003037100 -0.001491736
# b3 -0.001274303 -0.001491736  0.002830157
#
# $seBeta
#   b1   b2   b3
# 0.05722872 0.05510989 0.05319922
```

### skew

**Calculate Univariate Skewness for a Vector or Matrix**

**Description**

Calculate univariate skewness for vector or matrix (algorithm G1 in Joanes & Gill, 1998).
Usage

\texttt{skew(x)}

Arguments

\texttt{x} \\
Either a vector or matrix of numeric values.

Value

Skewness for each column in \texttt{x}.

Author(s)

Niels Waller

References


See Also

\texttt{kurt}

Examples

\begin{verbatim}
  x <- matrix(rnorm(1000), 100, 10)
  skew(x)
\end{verbatim}

---

\texttt{summary.monte} 
\textit{Summary Method for an Object of Class Monte}

Description

summary method for class “monte”

Usage

```
## S3 method for class 'monte'
summary(object, digits = 3, compute.validities = FALSE, Total.stats=TRUE, ...)
```
summary.monte

Arguments

object An object of class monte, usually, a result of a call to monte.
digits Number of digits to print. Default = 3.
compute.validities Logical: If TRUE then the program will calculate the indicator validities (eta^2) for the generated data.
Total.stats Logical: If TRUE then the program will return the following statistics for the total sample: (1) indicator correlation matrix, (2) indicator skewness, (3) indicator kurtosis.
... Optional arguments.

Value

Various descriptive statistics will be computed within groups including:

1. clus.size Number of objects within each group.
2. centroids Group centroids.
3. var.matrix Within group variances.
4. Ratio of within group variances (currently printed but not saved.
5. cor.list Expected within group correlations.
6. obs.cor Observed within group correlations.
7. skew.list Expected within group indicator skewness values.
8. obs.skew Observed within group indicator skewness values.
9. kurt.list Expected within group indicator kurtosis values.
10. obs.kurt Observed within group indicator kurtosis values.
11. validities Observed indicator validities.
12. Total.cor Total sample correlation matrix.
13. Total.skew Total sample indicator skewness.
14. Total.kurt Total sample indicator kurtosis.

Examples

## set up a 'monte' run for the Fisher iris data

```r
sk.lst <- list(c(0.120, 0.041, 0.106, 1.254),
               c(0.105, -0.363, -0.607, -0.031),
               c(0.118, 0.366, 0.549, -0.129))

kt.lst <- list(c(-0.253, 0.955, 1.022, 1.719),
               c(-0.533, -0.366, 0.048, -0.410),
               c( 0.033, 0.706, -0.154, -0.002))

cormat <- lapply(split(iris[,1:4], iris[,5]), cor)
```
my.iris <- monte(seed = 123, nvar = 4, nclus = 3, cor.list = cormat, clus.size = c(50, 50, 50),
    eta2 = c(0.619, 0.401, 0.941, 0.929),
    random.cor = FALSE,
    skew.list = sk.lst, kurt.list = kt.lst,
    secor = .3,
    compactness = c(1, 1, 1),
    sortMeans = TRUE)
summary(my.iris)

### Summary Method for an Object of Class Monte1

**Description**

summary method for class "monte1"

**Usage**

```r
## S3 method for class 'monte1'
summary(object, digits=3, ...)
```

**Arguments**

- `object` An object of class `monte1`, usually, a result of a call to `monte1`.
- `digits` Number of significant digits to print in final results.
- `...` Additional argument affecting the summary produced.

**Value**

Various descriptive statistics will be computed including:

1. Expected correlation matrix.
2. Observed correlation matrix.
3. Expected indicator skewness values.
4. Observed indicator skewness values.
5. Expected indicator kurtosis values.
6. Observed indicator kurtosis values.

**Examples**

```r
## Generate dimensional data for 4 variables.
## All correlations = .60; all variable
## skewness = 1.75;
## all variable kurtosis = 3.75

cormat <- matrix(.60, 4, 4)
```
tetcor

compute ml tetrachoric correlations

description
compute ml tetrachoric correlations with optional bias correction and smoothing.

usage
	tetcor(x, y = NULL, biascorrect, stderror, smooth = true, max.iter, print = true)

arguments
	x	either a matrix or vector of (0/1) binary data.
	y	an optional (if x is a matrix) vector of (0/1) binary data.

biascorrect	a logical that determines whether bias correction (brown & benedetti, 1977) is performed. default = true.

stderror	a logical that determines whether standard errors are calculated. default = false.

smooth	a logical which determines whether the tetrachoric correlation matrix should be smoothed. a smoothed matrix is always positive definite.

max.iter	maximum number of iterations. default = 50.

print	a logical that determines whether to print progress updates during calculations. default = true

value
if stderror = false, tetcor returns a matrix of tetrachoric correlations. if stderror = true then tetcor returns a list the first component of which is a matrix of tetrachoric correlations and the second component is a matrix of standard errors (see hamdan, 1970).

author(s)
niels waller
References


Examples

```r
## generate bivariate normal data
library(MASS)
set.seed(123)
rho <- .85
xy <- mvrnorm(100000, mu = c(0, 0), Sigma = matrix(c(1, rho, rho, 1), ncol = 2))

# dichotomize at difficulty values
p1 <- .7
p2 <- .1
xy[,1] <- xy[,1] < qnorm(p1)
xy[,2] <- xy[,2] < qnorm(p2)

print(apply(xy,2,mean), digits = 2)
# [1] 0.700 0.099

tetcor(x = xy, BiasCorrect = TRUE, stderror = TRUE, Smooth = TRUE, max.iter = 5000)

# $r
# [,1]        [,2]
# [1,] 1.00000000 0.8552535
# [2,] 0.8552535 1.0000000
#
# $se
# [,1]        [,2]
# [1,] NA       0.01458171
# [2,] 0.01458171 NA
#
# $Warnings
# list()
```

tetcorQuasi  Correlation between a Naturally and an Artificially Dichotomized Variable
Description

A function to compute Ulrich and Wirtz's correlation of a naturally and an artificially dichotomized variable.

Usage

tetcorQuasi(x, y = NULL)

Arguments

x  An N x 2 matrix or an N x 1 vector of binary responses coded 0/1.
y  An optional (if x is a vector) vector of 0/1 responses.

Value

A quasi tetrachoric correlation

Author(s)

Niels Waller

References


Examples

```r
set.seed(321)
Nsubj <- 5000

## Generate mvn data with rxy = .5
R <- matrix(c(1, .5, .5, 1), 2, 2)
X <- MASS::mvrnorm(n = Nsubj, mu = c(0, 0), Sigma = R, empirical = TRUE)

## dichotomize data
thresholds <- qnorm(c(.2, .3))
binaryData <- matrix(0, Nsubj, 2)

for(i in 1:2){
  binaryData[X[,i] <= thresholds[i],i] <- 1
}

## calculate Pearson correlation
cat("\nPearson r: ", round(cor(X)[1,2], 2))

## calculate Pearson Phi correlation
cat("\nPhi r: ", round(cor(binaryData)[1,2], 2))

## calculate tetrachoric correlation
```
cat("\nTetrachoric r: ", round(tetc%r(binaryData)$r[1,2], 2))

## calculate Quasi-tetrachoric correlation
cat("\nQuasi-tetrachoric r: ", round(tetc%r(binaryData), 2))

---

**vcos**  
*Compute the Cosine Between Two Vectors*

**Description**  
Compute the cosine between two vectors.

**Usage**  
vcos(x, y)

**Arguments**
- **x**: A p x 1 vector.
- **y**: A p x 1 vector.

**Value**  
Cosine between x and y

**Examples**  
x <- rnorm(5)
y <- rnorm(5)
vcos(x, y)

---

**vnorm**  
*Norm a Vector to Unit Length*

**Description**  
Norm a vector to unit length.

**Usage**  
vnorm(x)

**Arguments**
- **x**: An n by 1 vector.
vnorm

Value
   x scaled to unit length.

Author(s)
   Niels Waller

Examples
   x <- rnorm(5)
   v <- vnorm(x)
Index

*Topic **Statistics**
- adfCor, 2
- adfCov, 4
- corSmooth, 8
- d2r, 9
- eigGen, 9
- kurt, 23
- normalCor, 33
- r2d, 35
- rcor, 38
- seBeta, 43
- seBetaCor, 44
- seBetaFixed, 46
- skew, 47
- tetcor, 51
- tetcorQuasi, 52
- vcos, 54
- vnorm, 54

*Topic **datagen**
- bigen, 5
- corSample, 7
- enhancement, 10
- genCorr, 22
- monte, 24
- monte1, 32
- rarc, 35
- rcone, 37
- rellipsoid, 39
- rGivens, 40
- rMAP, 42

*Topic **fungible**
- fungible, 12
- fungibleExtrema, 13
- fungibleL, 15
- fungibleR, 17

- sBeta, 43, 47
- seBetaCor, 44
- seBetaFixed, 46
- skew, 24, 47
- summary.monte, 33, 48
- summary.monte1, 33, 50
INDEX

tetcor, 51
tetcorQuasi, 52

vcos, 54
vnorm, 54