Package ‘fungible’

October 7, 2019

Version 1.91
Date 2019-10-07
Title Psychometric Functions from the Waller Lab
Maintainer Niels Waller <nwaller@umn.edu>
Depends R (>= 3.5)
Imports clue, GPArotation, lattice, MASS, methods, mvtnorm, nleqslv, Rcsdp, RSpectra, utils, graphics, grDevices
Description Computes fungible coefficients and Monte Carlo data. Underlying theory for these functions is described in the following publications:
License GPL (>= 2)
Encoding UTF-8
NeedsCompilation no
RoxygenNote 6.1.1
Author Niels Waller [aut, cre],
Jeff Jones [ctb],
Casey Giordano [ctb]
Repository CRAN
Date/Publication 2019-10-07 18:10:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>adfCor</td>
<td>4</td>
</tr>
<tr>
<td>adfCov</td>
<td>5</td>
</tr>
<tr>
<td>AmzBoxes</td>
<td>6</td>
</tr>
<tr>
<td>BadRBY</td>
<td>7</td>
</tr>
<tr>
<td>BadRJN</td>
<td>8</td>
</tr>
<tr>
<td>BadRKtB</td>
<td>8</td>
</tr>
<tr>
<td>BadRLG</td>
<td>9</td>
</tr>
<tr>
<td>BadRRM</td>
<td>9</td>
</tr>
<tr>
<td>BiFAD</td>
<td>10</td>
</tr>
<tr>
<td>bigen</td>
<td>14</td>
</tr>
<tr>
<td>Box20</td>
<td>16</td>
</tr>
<tr>
<td>Box26</td>
<td>17</td>
</tr>
<tr>
<td>corSample</td>
<td>19</td>
</tr>
<tr>
<td>corSmooth</td>
<td>20</td>
</tr>
<tr>
<td>cosMat</td>
<td>21</td>
</tr>
<tr>
<td>d2r</td>
<td>23</td>
</tr>
<tr>
<td>eap</td>
<td>23</td>
</tr>
<tr>
<td>eigGen</td>
<td>25</td>
</tr>
<tr>
<td>enhancement</td>
<td>26</td>
</tr>
<tr>
<td>erf</td>
<td>28</td>
</tr>
<tr>
<td>faAlign</td>
<td>29</td>
</tr>
<tr>
<td>faEKC</td>
<td>33</td>
</tr>
<tr>
<td>fals</td>
<td>34</td>
</tr>
<tr>
<td>faMain</td>
<td>35</td>
</tr>
<tr>
<td>faMAP</td>
<td>43</td>
</tr>
<tr>
<td>fapa</td>
<td>44</td>
</tr>
<tr>
<td>fareg</td>
<td>47</td>
</tr>
<tr>
<td>faScores</td>
<td>49</td>
</tr>
<tr>
<td>faSort</td>
<td>52</td>
</tr>
<tr>
<td>faStandardize</td>
<td>54</td>
</tr>
<tr>
<td>faX</td>
<td>55</td>
</tr>
<tr>
<td>FMP</td>
<td>58</td>
</tr>
<tr>
<td>FMPMonotonicityCheck</td>
<td>61</td>
</tr>
<tr>
<td>fungible</td>
<td>62</td>
</tr>
<tr>
<td>fungibleExtrema</td>
<td>63</td>
</tr>
<tr>
<td>fungibleL</td>
<td>65</td>
</tr>
<tr>
<td>fungibleR</td>
<td>67</td>
</tr>
<tr>
<td>FUP</td>
<td>72</td>
</tr>
<tr>
<td>gen4PMData</td>
<td>75</td>
</tr>
<tr>
<td>genCorr</td>
<td>77</td>
</tr>
<tr>
<td>GenerateBoxData</td>
<td>78</td>
</tr>
<tr>
<td>genFMPData</td>
<td>82</td>
</tr>
<tr>
<td>genPhi</td>
<td>83</td>
</tr>
<tr>
<td>HS9Var</td>
<td>85</td>
</tr>
<tr>
<td>HW</td>
<td>86</td>
</tr>
<tr>
<td>irf</td>
<td>86</td>
</tr>
</tbody>
</table>
topics documented:

- itemDescriptives ......................................................... 88
- kurt ................................................................. 90
- Ledermann ............................................................. 91
- monte ................................................................. 92
- monte1 ................................................................. 100
- normalCor ............................................................. 102
- normF ................................................................. 103
- Omega ................................................................. 103
- orderFactors ......................................................... 105
- plot.monte ........................................................... 106
- print.faMain ......................................................... 106
- promaxQ ............................................................... 107
- r2d ................................................................. 111
- rarc ................................................................. 112
- rcone ................................................................. 114
- rcor ................................................................. 115
- rellipsoid ............................................................ 116
- restScore ............................................................. 117
- rGivens ............................................................... 119
- rMAP ................................................................. 120
- rmsd ................................................................. 121
- RnpdMAP .............................................................. 122
- SchmidLeiman ......................................................... 125
- seBeta ............................................................... 129
- seBetaCor ........................................................... 131
- seBetaFixed ........................................................ 132
- simFA ................................................................. 134
- skew ................................................................. 139
- SLi ................................................................. 140
- smoothAPA .......................................................... 144
- smoothBY ............................................................ 146
- smoothKB ............................................................ 148
- smoothLG ............................................................ 149
- summary.faMain ...................................................... 151
- summary.monte ..................................................... 156
- summary.monte1 .................................................... 157
- svdNorm .............................................................. 158
- tetcor ............................................................... 159
- tetcorQuasi .......................................................... 161
- ThurstoneBox20 ....................................................... 162
- ThurstoneBox26 ...................................................... 164
- vcos ................................................................. 165
- vnorm ............................................................... 166

Index ................................................................. 167
adfCor                  Asymptotic Distribution-Free Covariance Matrix of Correlations

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

Usage
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 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))) * 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 <- monte1(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)

#>        12       13       23
#> 12 0.001207845 0.0005331086 0.0004821594
#> 13 0.0005331086 0.0004980130 0.0002712080
#> 23 0.0004821594 0.0002712080 0.0005415301

---

**adfCov**

*Asymptotic Distribution-Free Covariance Matrix of Covariances*

**Description**

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

**Usage**

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**

## Generate non-normal data using `monte1`

```r
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 non-normal data for the predictors (X)
## x1 has expected skew = 1 and kurtosis = 3
## x2 has expected skew = 2 and kurtosis = 5
X <- monte1(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 Covariances
adfCov(X, y)
```

#>      11    12    13    22    23    33
#> 11 3.438760 2.317159 2.269080 2.442003 1.962584 1.688631
#> 12 2.317159 3.171722 2.278212 3.349173 2.692097 2.028701
#> 13 2.269080 2.278212 2.303659 2.395033 2.149316 2.106310
#> 22 2.442003 3.171722 2.303659 2.395033 2.149316 2.106310
#> 23 1.962584 2.692097 2.149316 4.086652 3.287088 2.501094
#> 33 1.688631 2.028701 2.106310 2.687647 2.501094 2.818664

---

### Description

Length, width, and height measurements for 98 Amazon shipping boxes

### Usage

`data(AmzBoxes)`
Format

A data set of measurements for 98 Amazon shipping boxes. These data were downloaded from the BoxDimensions website: (https://www.boxdimensions.com/). The data set includes five variables:

- Amazon Box Size
- Length (inches)
- Width (inches)
- Height (inches)
- Volume (inches)

Examples

data(AmzBoxes)

hist(AmzBoxes$‘Length (inches)’,
     main = “Histogram of Box Lengths”,
     xlab = “Length”,
     col = “blue”)

---

BadRBY

Improper correlation matrix reported by Bentler and Yuan

Description

Example improper R matrix reported by Bentler and Yuan (2011)

Format

A 12 by 12 non-positive definite correlation matrix.

Source


Examples

data(BadRBY)
<table>
<thead>
<tr>
<th></th>
<th>Improper R matrix reported by Joseph and Newman</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadRJN</td>
<td>Example NPD improper correlation matrix reported by Joseph and Newman</td>
</tr>
<tr>
<td>Format</td>
<td>A 14 by 14 non-positive definite correlation matrix.</td>
</tr>
<tr>
<td>Examples</td>
<td>data(BadRJN)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Improper R matrix reported by Knol and ten Berge</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadRKtB</td>
<td>Example improper R matrix reported by Knol and ten Berge</td>
</tr>
<tr>
<td>Format</td>
<td>A 6 by 6 non-positive definite correlation matrix.</td>
</tr>
<tr>
<td>Examples</td>
<td>data(BadRKtB)</td>
</tr>
</tbody>
</table>
BadRLG

Improper R matrix reported by Lurie and Goldberg

Description

Example improper R matrix reported by Lurie and Goldberg

Format

A 3 by 3 non-positive definite correlation matrix.

Source


Examples

data(BadRLG)

BadRRM

Improper R matrix reported by Rousseeuw and Molenberghs

Description

Example improper R matrix reported by Rousseeuw and Molenberghs

Format

A 3 by 3 non-positive definite correlation matrix.

Source


Examples

data(BadRRM)
BiFAD: Bifactor Analysis via Direct Schmid-Leiman (DSL) Transformations

Description

This function estimates the (rank-deficient) Direct Schmid-Leiman (DSL) bifactor solution as well as the (full-rank) Direct Bifactor (DBF) solution.

Usage

BiFAD(R, B = NULL, numFactors = NULL, facMethod = "fals", rotate = "oblimin", salient = 0.25, digits = NULL, rotateControl = NULL, faControl = NULL)

Arguments

R (Matrix) A correlation matrix.
B (Matrix) Bifactor target matrix. If B is NULL the program will create an empirically defined target matrix.
numFactors (Numeric) The number of group factors to estimate.
facMethod (Character) The method used for factor extraction (faX). The supported options are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa" for iterated principal axis factoring, "faregLS" for regularized least squares, "faregML" for regularized maximum likelihood factor, and "pca" for principal components analysis. The default method is "fals".
  • "fals": Factors are extracted using the unweighted least squares estimation procedure using the fals function.
  • "faml": Factors are extracted using the maximum likelihood estimation procedure using the factanal function.
  • "fapa": Factors are extracted using the iterated principal axis factoring estimation procedure using the fapa function.
  • "faregLS": Factors are extracted using regularized least squares factor analysis using the fareg function.
  • "faregML": Factors are extracted using regularized maximum likelihood factor using the fareg function.
  • "pca": Principal components are extracted.
rotate (Character) Designate which rotation algorithm to apply. See the faMain function for more details about possible rotations. An oblimin rotation is the default.
salient (Numeric) Threshold value for creating an empirical target matrix.
digits (Numeric) Rounds the values to the specified number of decimal places. Defaults to digits = NULL (no rounding).
rotateControl (List) A list of control values to pass to the factor rotation algorithms.
• **numberStarts**: (Numeric) The number of random (orthogonal) starting configurations for the chosen rotation method (e.g., oblimin). The first rotation will always commence from the unrotated factors orientation. Defaults to numberStarts = 10.

• **gamma**: (Numeric) This is a tuning parameter (between 0 and 1, inclusive) for an oblimin rotation. See the GPArotation library’s oblimin documentation for more details. Defaults to gamma = 0 (i.e., a quartimin rotation).

• **delta**: (Numeric) This is a tuning parameter for the geomin rotation. It adds a small number (default = .01) to the squared factor loadings before computing the geometric means in the discrepancy function.

• **kappa**: (Numeric) The main parameterization of the Crawford-Ferguson (CF) rotations (i.e., "cfT" and "cfQ" for orthogonal and oblique CF rotation, respectively). Defaults to kappa = 0.

• **k**: (Numeric) A specific parameter of the simplimax rotation. Defaults to k = the number of observed variables.

• **standardize**: (Character) The standardization routine used on the unrotated factor structure. The three options are "none", "Kaiser", and "CM". Defaults to standardize = "none".
  - "none": No standardization is applied to the unrotated factor structure.
  - "Kaiser": Use a factor structure matrix that has been normed by Kaiser’s method (i.e., normalize all rows to have a unit length).
  - "CM": Use a factor structure matrix that has been normed by the Cureton-Mulaik method.

• **epsilon**: (Numeric) The rotational convergence criterion to use. Defaults to epsilon = 1e-5.

• **power**: (Numeric) Raise factor loadings the n-th power in the promaxQ rotation. Defaults to power = 4.

• **maxItr**: (Numeric) The maximum number of iterations for the rotation algorithm. Defaults to maxItr = 15000.

faControl (List) A list of optional parameters passed to the factor extraction (faX) function.

• **treatHeywood**: (Logical) In fa1s, if treatHeywood is true, a penalized least squares function is used to bound the communality estimates below 1.0. Defaults to treatHeywood = TRUE.

• **nStart**: (Numeric) The number of starting values to be tried in fam1. Defaults to nStart = 10.

• **start**: (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to start = NULL.

• **maxCommunality**: (Numeric) In fam1, set the maximum communality value for the estimated solution. Defaults to maxCommunality = .995.

• **epsilon**: (Numeric) In fapa, the numeric threshold designating when the algorithm has converged. Defaults to epsilon = 1e-4.

• **communality**: (Character) The method used to estimate the initial communality values in fapa. Defaults to communality = "SMC".
  - "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
- "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
- "unity": Initial communalities equal 1.0 for all variables.

• maxItr: (Numeric) In fapa, the maximum number of iterations to reach convergence. Defaults to maxItr = 15,000.

Value

The following output are returned in addition to the estimated Direct Schmid-Leiman bifactor solution.

- B: (Matrix) The target matrix used for the Procrustes rotation.
- BstarFR: (Matrix) The resulting (full-rank) matrix of Direct Bifactor factor loadings.
- rmsrSL: (Scalar) The root mean squared residual (rmsr) between the known B matrix and the estimated (rank-deficient) Direct Schmid-Leiman rotation. If the B target matrix is empirically generated, this value is NULL.
- rmsrFR: (Scalar) The root mean squared residual (rmsr) between the known B matrix and the estimated (full-rank) Direct Bifactor rotation. If the B target matrix is empirically generated, this value is NULL.

Author(s)

- Niels G. Waller (nwaller@umn.edu)

References


See Also

Other Factor Analysis Routines: *Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain*

Examples

cat("\nExample 1:\nEmpirical Target Matrix:\n")
# Mansolf and Reise Table 2 Example
Btrue <- matrix(c(.48, .40, 0, 0, 0,
                 .51, .35, 0, 0, 0,
                 ,.51, .35, 0, 0, 0,
                 ,.51, .35, 0, 0, 0,
                 ,.51, .35, 0, 0, 0), nrow = 5, ncol = 5, byrow = TRUE)
BiFAD

.67, .62, 0, 0, 0,
.34, .55, 0, 0, 0,
.44, 0, .45, 0, 0,
.40, 0, .48, 0, 0,
.32, 0, .70, 0, 0,
.45, 0, .54, 0, 0,
.55, 0, 0, .43, 0,
.33, 0, 0, .33, 0,
.52, 0, 0, .51, 0,
.35, 0, 0, .69, 0,
.32, 0, 0, 0, .65,
.66, 0, 0, 0, .51,
.68, 0, 0, 0, .39,
.32, 0, 0, 0, .56), 16, 5, byrow=TRUE)

Rex1 <- Btrue %*% t(Btrue)
diag(Rex1) <- 1

out.ex1 <- BiFAD(R = Rex1,
                B = NULL,
                numFactors = 4,
                facMethod = "fals",
                rotate = "oblimin",
                salient = .25)

cat("Rank Deficient Bifactor Solution:
"
print( round(out.ex1$BstarSL, 2) )

cat("Full Rank Bifactor Solution:
"
print( round(out.ex1$BstarFR, 2) )

Bpattern <- matrix(c( 1, 1, 0, 0, 0,
1, 1, 0, 0, 0,
1, 1, 0, 0, 0,
1, 0, 1, 0, 0,
1, 0, 1, 0, 0,
1, 0, 1, 0, 0,
1, 0, 1, 0, 0,
1, 0, 1, 0, 0,
1, 0, 1, 0, 0), 16, 5, byrow=TRUE)

out.ex2 <- BiFAD(R = Rex1,
                B = Bpattern,
                numFactors = NULL,
                facMethod = "fals",
                rotate = "oblimin",
                salient = .25)
facMethod = "fals",
          rotate = "oblimin",
          salient = .25)

  cat("\nRank Deficient Bifactor Solution:\n")
  print( round(out.ex2$BstarSL, 2) )

  cat("\nFull Rank Bifactor Solution:\n")
  print( round(out.ex2$BstarFR, 2) )

bigen

Generate Correlated Binary Data

Description

Function for generating binary data with population thresholds.

Usage

bigen(data, n, thresholds = NULL, Smooth = FALSE, 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 data is a correlation matrix, thresholds must be a vector of threshold cut points.

Smooth (logical) Smooth = TRUE will smooth the tetrachoric correltion matrix.

seed Default = FALSE. Optional seed for random number generator.

Value

data Simulated binary data

r Input or calculated (tetrachoric) correlation matrix

Author(s)

Niels G Waller

Examples

## 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)

cat("\nnames in returned object\n")
print(names(z))

cat("\nCorrelation of Simulated binary scores\n")
print(round(cor(z$data), 3))

cat("Observed thresholds of simulated data:\n")
cat(apply(z$data, 2, mean))

Description

Length, width, and height measurements for Thurstone's 20 hypothetical boxes

Usage

data(Box20)

Format

A data set of measurements for Thurstone's 20 hypothetical boxes. The data set includes three variables:

- **x** Box length
- **y** Box width
- **z** Box height

Examples

data(Box20)

hist(Box20$x,
    main = "Histogram of Box Lengths",
    xlab = "Length",
    col = "blue")

# To create the raw data for Thurstone's 20 hypothetical
# box attributes:
data(Box20)
ThurstoneBox20 <- GenerateBoxData(XYZ = Box20,
    BoxStudy = 20,
    Reliability = 1,
    ModApproxErrVar = 0)$BoxData

RThurstoneBox20 <- cor(ThurstoneBox20)

# Smooth matrix to calculate factor indeterminacy values
RsmThurstoneBox20 <- smoothBY(RThurstoneBox20)$RBY

fout <- faMain(R = RsmThurstoneBox20,
    numFactors = 3,
    rotate = "varimax",
    facMethod = "faregLS",
    rotateControl = list(numberStarts = 100,
summary(fout, digits=3)

# Note that given the small ratio of subjects to variables,
# it is not possible to generate data for this example with model error
# (unless SampleSize is increased).

---

**Box26**

*R matrix for Thurstone’s 26 hypothetical box attributes.*

**Description**

Correlation matrix for Thurstone’s 26 hypothetical box attributes.

**Usage**

data(Box26)

**Format**

Correlation matrix for Thurstone’s 26 hypothetical box attributes. The so-called Thurstone invariant box problem contains measurements on the following 26 functions of length, width, and height. **Box26** variables:

1. x
2. y
3. z
4. xy
5. xz
6. yz
7. x^2 * y
8. x * y^2
9. x^2 * z
10. x * z^2
11. y^2 * z
12. y * z^2
13. x/y
14. y/x
15. x/z
16. z/x
17. y/z
18. z/y
19. \(2x + 2y\)
20. \(2x + 2z\)
21. \(2y + 2z\)
22. \(\sqrt{x^2 + y^2}\)
23. \(\sqrt{x^2 + z^2}\)
24. \(\sqrt{y^2 + z^2}\)
25. \(xyz\)
26. \(\sqrt{x^2 + y^2 + z^2}\)

- \(x\) Box length
- \(y\) Box width
- \(z\) Box height

Details

Two data sets have been described in the literature as Thurstone’s Box Data (or Thurstone’s Box Problem). The first consists of 20 measurements on a set of 20 hypothetical boxes (i.e., Thurstone made up the data). Those data are available in **Box20**. The second data set, which is described in this help file, was collected by Thurstone to provide an illustration of the invariance of simple structure factor loadings. In his classic textbook on multiple factor analysis (Thurstone, 1947), Thurstone states that “[m]easurements of a random collection of thirty boxes were actually made in the Psychometric Laboratory and recorded for this numerical example. The three dimensions, \(x\), \(y\), and \(z\), were recorded for each box. A list of 26 arbitrary score functions was then prepared” (p. 369). The raw data for this example were not published. Rather, Thurstone reported a correlation matrix for the 26 score functions (Thurstone, 1947, p. 370). Note that, presumably due to rounding error in the reported correlations, the correlation matrix for this example is non positive definite.

References


See Also

**Box20, AmzBoxes**

Other Factor Analysis Routines: **BiFAD, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain**

Examples

data(Box26)
fout <- faMain(R = Box26,
               numFactors = 3,
               facMethod = "faregLS",
               rotate = "varimax",
               bootstrapSE = FALSE,
# We now choose Cureton-Mulaik row standardization to reveal the underlying factor structure.

fout <- faMain(R = Box26,  
    numFactors = 3,  
    facMethod = "faregLS",  
    rotate = "varimax",  
    bootstrapSE = FALSE,  
    rotateControl = list(  
        numberStarts = 100,  
        standardize = "CM"),  
    Seed = 123)

summary(fout)
References


Examples

```r
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

```r
corSmooth(R, eps = 1e+08 * .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

### cosMat

**Examples**

```r
cosMat(A, B, align = FALSE, digits = NULL)
```

**Description**

This function will compute the cosines (i.e., the angle) between two vectors or matrices. When applied to matrices, it will compare the two matrices one vector (i.e., column) at a time. For instance, the cosine (angle) between factor 1 in matrix A and factor 1 in matrix B.

**Usage**

```r
cosMat(A, B, align = FALSE, digits = NULL)
```

**Arguments**

- `A` (Matrix, Vector) Either a matrix or vector.
- `B` (Matrix, Vector) Either a matrix or vector (must be of the same dimensions as A).
- `align` (Logical) Whether to run a factor alignment before computing the cosine.
- `digits` (Numeric) The number of digits to round the output to.

**Details**

- **Chance Congruence**: Factor cosines were originally described by Burt (1948) and later popularized by Tucker (1951). Several authors have noted the tendency for two factors to have spuriously large factor cosines. Paunonen (1997) provides a good overview and describes how factor cosines between two vectors of random numbers can appear to be congruent.
• **Effect Size Benchmarks:** When computing congruence coefficients (cosines) in factor analytic studies, it can be useful to know what constitutes large versus small congruence. Lorenzo-Seva and ten Berge (2006) currently provide the most popular (i.e., most frequently cited) recommended benchmarks for congruence. “A value in the range .85-.94 means that the two factors compared display fair similarity. This result should prevent congruence below .85 from being interpreted as indicative of any factor similarity at all. A value higher than .95 means that the two factors or components compared can be considered equal. That is what we have called a good similarity in our study” (Lorenzo-Seva & ten Berge, 2006, p. 61, emphasis theirs).

**Value**

A vector of cosines will be returned. When comparing two vectors, only one cosine can be computed. When comparing matrices, one cosine is computed per column.

- **cosine:** (Matrix) A matrix of cosines between the two inputs.
- **A:** (Matrix) The A input matrix.
- **B:** (Matrix) The B input matrix.
- **align:** (Logical) Whether Matrix B was aligned to A.

**Author(s)**

- Casey Giordano (Giord023@umn.edu)
- Niels G. Waller (nwaller@umn.edu)

**References**


**Examples**

```r
## Cosine between two vectors
A <- rnorm(5)
B <- rnorm(5)
cosMat(A, B)

## Cosine between the columns of two matrices
A <- matrix(rnorm(5 * 5), 5, 5)
B <- matrix(rnorm(5 * 5), 5, 5)
cosMat(A, B)
```
\textit{d2r} \hspace{1cm} \textit{Convert Degrees to Radians}

\textbf{Description}
A simple function to convert degrees to radians

\textbf{Usage}
d2r(deg)

\textbf{Arguments}
deg \hspace{1cm} Angle in degrees.

\textbf{Value}
Angle in radians.

\textbf{Examples}
d2r(90)

\textit{eap} \hspace{1cm} \textit{Compute eap trait estimates for FMP and FUP models}

\textbf{Description}
Compute eap trait estimates for items fit by filtered monotonic polynomial IRT models.

\textbf{Usage}
eap(data, bParams, NQuad = 21, priorVar = 2, mintheta = -4, maxtheta = 4)
Arguments

data             N(subjects)-by-p(items) matrix of 0/1 item response data.
bParams         A p-by-9 matrix of FMP or FUP item parameters and model designations. Columns 1 - 8 hold the (possibly zero valued) polynomial coefficients; column 9 holds the value of k.
NQuad            Number of quadrature points used to calculate the eap estimates.
priorVar         Variance of the normal prior for the eap estimates. The prior mean equals 0.
imthetamu, maxtheta
                   NQuad quadrature points will be evenly spaced between imthetamu and maxtheta

Value

eap trait estimates.

Author(s)

Niels Waller

Examples

```r
## this example demonstrates how to calculate
## eap trait estimates for a scale composed of items
## that have been fit to FMP models of different
## degree

NSubjects <- 2000

## Assume that
## items 1 - 5 fit a k=0 model,
## items 6 - 10 fit a k=1 model, and
## items 11 - 15 fit a k=2 model.

itmParameters <- matrix(c(  
# b0 b1 b2 b3 b4 b5 b6 b7 k
-1.05, 1.63, 0.00, 0.00, 0.00, 0, 0, 0, 0, #1
-1.97, 1.75, 0.00, 0.00, 0.00, 0, 0, 0, 0, #2
-1.77, 1.82, 0.00, 0.00, 0.00, 0, 0, 0, 0, #3
-4.76, 2.67, 0.00, 0.00, 0.00, 0, 0, 0, 0, #4
-2.15, 1.93, 0.00, 0.00, 0.00, 0, 0, 0, 0, #5
-1.25, 1.17, -0.25, 0.12, 0.00, 0, 0, 0, 1, #6
1.65, 0.01, 0.02, 0.03, 0.00, 0, 0, 0, 1, #7
-2.99, 1.64, 0.17, 0.03, 0.00, 0, 0, 0, 1, #8
-3.22, 2.40, -0.12, 0.10, 0.00, 0, 0, 0, 1, #9
-0.75, 1.09, -0.39, 0.31, 0.00, 0, 0, 0, 1, #10
-1.21, 9.07, 1.20,-0.01,-0.01, 0.01, 0, 0, 2, #11
-1.92, 1.55, -0.17, 0.50,-0.01, 0.01, 0, 0, 2, #12
), byrow = T, ncol = 9)
```

# eigGen
eigGen

## Description

Generate eigenvalues for R matrices with underlying component structure

## Usage

```r
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,
            "%", sep = "")

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

---

**enhancement**

Find OLS Regression Coefficients that Exhibit Enhancement

**Description**

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

**Usage**

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

- **R**: Predictor correlation matrix.
- **br**: Model R-squared = $b^t r$. That is, $br$ is the model coefficient of determination: $b'Rb = Rsq = br$
- **rr**: Sum of squared predictor-criterion correlations ($r_{xy}$). That is, $rr = r'r = \text{Sum}(r_{xy}^2)$

Value

- **b**: Vector of standardized regression coefficients.
- **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)
```
Utility function to compute empirical response functions.

### Description

Utility function to compute empirical response functions.

### Usage

```r
erf(theta, data, whichItem, min = -3, max = 3, Ncuts = 12)
```

### Arguments

- `theta`: Vector of estimated latent trait scores.
- `data`: A matrix of binary item responses.
- `whichItem`: Data for an erf will be generated for whichItem.
- `Ncuts`: Number of score groups for erf.

### Value

- `probs`: A vector (of length Ncuts) of bin response probabilities for the empirical response function.
- `centers`: A vector of bin centers.
- `Ni`: Bin sample sizes.
- `se.p`: Standard errors of the estimated bin response probabilities.

### Author(s)

Niels Waller

### Examples

```r
NSubj <- 2000
#generate sample k=1 FMP data
b <- matrix(c(
  #b0  b1  b2  b3  b4  b5  b6  b7  k
  1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1,
  1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1,
  1.282, 1.063, -0.103, 0.003, 0, 0, 0, 0, 1,
  0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,
  1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,
), nrow = 10, byrow = TRUE)
```

-0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,
0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,
0.122, 0.601, -0.082, 0.119, 0, 0, 0, 0, 1,
1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,
-0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,
-0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,
0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,
-0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,
0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,
1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,
0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,
0.429, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,
0.089, 0.785, -0.005, 0.018, 0, 0, 0, 0, 1,
-0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,
0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,
0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,
-0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,
0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1,
                   nrow=23, ncol=9, byrow=TRUE)
theta <- rnorm(NSubj)
data<-genFMPData(NSubj = NSubj, bParam = b, theta = theta, seed = 345)$data
erfItem1 <- erf(theta, data, whichItem = 1, min = -3, max = 3, Ncuts = 12)
plot( erfItem1$centers, erfItem1$probs, type="b",
     main="Empirical Response Function",
     xlab = expression(theta),
     ylab="Probability",
     cex.lab=1.5)

faAlign

Align the columns of two factor loading matrices

Description

Align factor loading matrices across solutions using the Hungarian algorithm to locate optimal matches. faAlign will match the factors of F2 (the input matrix) to those in F1 (the target matrix) to minimize a least squares discrepancy function or to maximize factor congruence coefficients (i.e., vector cosines).

Usage

faAlign(F1, F2, Phi2 = NULL, MatchMethod = "LS")

Arguments

F1      target Factor Loadings Matrix.
F2

input Factor Loadings Matrix. F2 will be aligned with the target matrix, F1.

Phi2

optional factor correlation matrix for F2 (default = NULL).

MatchMethod

"LS" (Least Squares) or "CC" (congruence coefficients).

Value

F2
re-ordered and reflected loadings of F2.

Phi2
reordered and reflected factor correlations.

FactorMap
a 2 x k matrix (where k is the number of columns of F1) structured such that row 1: the original column order of F2; row 2: the sorted column order of F2.

UniqueMatch
(logical) indicates whether a unique match was found.

MatchMethod
"LS" (least squares) or "CC" (congruence coefficients, i.e., cosines).

CC
Congruence coefficients for the matched factors.

LS
Root-mean-squared-deviations (least squares criterion) for the matched factors.

Note


Author(s)

Niels Waller

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

# This example demonstrates the computation of
# non-parametric bootstrap confidence intervals
# for rotated factor loadings.
library(GPArotation)

data(HS9Var)

HS9 <- HS9Var[HS9Var$school == "Grant-White",7:15]

# Compute an R matrix for the HSVar9 Mental Abilities Data
R.HS9 <- cor(HS9)

varnames <- c( "vis.per", "cubes", "lozenges", "paragraph.comp", "sentence.comp", "word.mean", "speed.add", "speed.count.dots", "speed.dscr")

# Extract and rotate a 3-factor solution
# via unweighted least squares factor extraction
# and oblimin rotation.

NFac <- 3
NVar <- 9
B <- 200 # Number of bootstrap samples
NSubj <- nrow(HS9)

# Unrotated 3 factor uls solution
F3.uls <- fals(R = R.HS9, nfactors = NFac)

# Rotate via oblimin
F3.rot <- oblimin(F3.uls$loadings, gam = 0, normalize = FALSE)

F3.loadings <- F3.rot$loadings
F3.phi <- F3.rot$Phi

# Reflect factors so that salient loadings are positive
Dsgn <- diag(sign(colSums(F3.loadings^3)))
F3.loadings <- F3.loadings %*% Dsgn
F3.phi <- Dsgn %*% F3.phi %*% Dsgn

rownames(F3.loadings) <- varnames
colnames(F3.loadings) <- paste0("f", 1:3)
colnames(F3.phi) <- rownames(F3.phi) <- paste0("f", 1:3)

cat("Oblimin rotated factor loadings for 9 Mental Abilities Variables")
print(round(F3.loadings, 2))

cat("Factor correlation matrix")
print(round(F3.phi, 2))
# Declare variables to hold bootstrap output
Flist <- Philist <- as.list(rep(0, B))
UniqueMatchVec <- rep(0, B)
rows <- 1:NSubj

# Analyze bootstrap samples and record results
for(i in 1:B){
  cat("Working on sample ", i)
  set.seed(i)

  # Create bootstrap samples
  bsRows <- sample(rows, NSubj, replace= TRUE)
  Fuls <- fals(R = cor(HS9[bsRows, ]), nfactors = NFac)
  # rotated loadings
  Fboot <- oblimin(Fuls$loadings,
                  gam = 0,
                  normalize = FALSE)

  out <- faAlign(F1 = F3.loadings,
                 F2 = Fboot$loadings,
                 MatchMethod = "LS")
  Flist[[i]] <- out$F2 # aligned version of Fboot$loadings
  UniqueMatchVec[i] <- out$UniqueMatch
}

cat("Number of Unique Matches: ",
     100*round(mean(UniqueMatchVec),2),"%\n")

# Make a 3D array from list of matrices
arr <- array( unlist(Flist) , c(NVar, NFac, B) )

# Get quantiles of factor elements over third dimension (samples)
F95 <- apply( arr , 1:2 , quantile, .975 )
F05 <- apply( arr , 1:2 , quantile, .025 )
Fse <- apply( arr , 1:2, sd )

cat("Upper Bound 95% CI\n")
print( round(F95,3))
cat("Lower Bound 95% CI\n")
print( round(F05,3))

# plot distribution of bootstrap estimates
# for example element
hist(arr[5,1,], xlim=c(.4,1),
     main = "Bootstrap Distribution for F[5,1]",
     xlab = "F[5,1]"
)
print(round (F3.loadings, 2))
cat("Standard Errors")
print( round( Fse, 2))
Calculate Reference Eigenvalues for the Empirical Kaiser Criterion

**Description**

Calculate Reference Eigenvalues for the Empirical Kaiser Criterion

**Usage**

```r
faEKC(R = NULL, NSubj = NULL, Plot = FALSE)
```

**Arguments**

- **R** Input correlation matrix.
- **NSubj** Number of subjects (observations) used to create R.
- **Plot** (logical). If `Plot = TRUE` the function will plot the observed and reference eigenvalues of R.

**Value**

- `ljEKC`,
- `ljEKC1`,
- `dimensions` The estimated number of common factors.

**Author(s)**

Niels Waller

**See Also**

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

**Examples**

```r
data(AmzBoxes)
AmzBox20 <- GenerateBoxData(XYZ = AmzBoxes[,2:4],
                            BoxStudy = 20)$BoxData
RAmzBox20 <- cor(AmzBox20)
EKCout <- faEKC(R = RAmzBox20,
                NSubj = 98,
                Plot = TRUE)
```
fals

Unweighted least squares factor analysis

Description
Unweighted least squares factor analysis

Usage
fals(R, nfactors, TreatHeywood = TRUE)

Arguments
- R: Input correlation matrix.
- nfactors: Number of factors to extract.
- TreatHeywood: If TreatHeywood = TRUE then a penalized least squares function is used to bound the commonality estimates below 1.0. Default(TreatHeywood = TRUE).

Value
- loadings: Unrotated factor loadings. If a Heywood case is present in the initial solution then the model is re-estimated via non-iterated principal axes with max(rij^2) as fixed communality (h2) estimates.
- h2: Vector of final communality estimates.
- uniqueness: Vector of factor uniquenesses, i.e. (1 - h2).
- Heywood: (logical) TRUE if a Heywood case was produced in the LS solution.
- TreatHeywood: (logical) Value of the TreatHeywood argument.
- converged: (logical) TRUE if all values of the gradient are sufficiently close to zero.
- MaxAbsGrad: The maximum absolute value of the gradient at the solution.
- f.value: The discrepancy value associated with the final solution.

Author(s)
Niels Waller

See Also
Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain
Examples

Rbig <- fungible::rcor(120)
out1 <- fals(R = Rbig,
  nfactors = 2,
  TreatHeywood = TRUE)

faMain

Automatic Factor Rotation from Random Configurations with Bootstrap Standard Errors

Description

This function conducts factor rotations (using the GPArotation package) from a user-specified number of random (orthogonal) starting configurations. Based on the resulting complexity function value, the function determines the number of local minima and, among these local solutions, will find the "global minimum" (i.e., the minimized complexity value from the finite number of solutions). See Details below for an elaboration on the global minimum. This function can also return bootstrap standard errors of the factor solution.

Usage

faMain(X = NULL, R = NULL, n = NULL, numFactors = NULL,
  facMethod = "fals", urLoadings = NULL, rotate = "oblimin",
  targetMatrix = NULL, bootstrapSE = FALSE, numBoot = 1000,
  CILevel = 0.95, Seed = 1, digits = NULL, faControl = NULL,
  rotateControl = NULL, ...)

Arguments

X (Matrix) A raw data matrix (or data frame).
R (Matrix) A correlation matrix.
n (Numeric) Sample size associated with the correlation matrix. Defaults to n = NULL.
numFactors (Numeric) The number of factors to extract for subsequent rotation.
facMethod (Character) The method used for factor extraction (faX). The supported options are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa" for iterated principal axis factoring, "faregLS" for regularized least squares, "faregML" for regularized maximum likelihood, and "pca" for principal components analysis. The default method is "fals".
  • "fals": Factors are extracted using the unweighted least squares estimation procedure using the fals function.
  • "faml": Factors are extracted using the maximum likelihood estimation procedure using the factanal function.
• "fapa": Factors are extracted using the iterated principal axis factoring estimation procedure using the fapa function.
• "faregLS": Factors are extracted using regularized least squares factor analysis using the fareg function.
• "faregML": Factors are extracted using regularized maximum likelihood factor using the fareg function.
• "pca": Principal components are extracted.

urLoadings (Matrix) An unrotated factor-structure matrix to be rotated.

rotate (Character) Designate which rotation algorithm to apply. The following are available rotation options: "oblimin", "quartimin", "targetT", "targetQ", "oblimax", "entropy", "quartimax", "varimax", "simplimax", "bentlerT", "bentlerQ", "tandemI", "tandemII", "geominT", "geominQ", "cfT", "cfQ", "infomaxT", "infomaxQ", "mccammon", "bifactorT", "bifactorQ", and "none". Defaults to rotate = "oblimin". See GPArotation package for more details. Note that rotations ending in "T" and "Q" represent orthogonal and oblique rotations, respectively.

targetMatrix (Matrix) This argument serves two functions. First, if a user has requested either a "targetT" or "targetQ" rotation, then the target matrix is used to conduct a fully or partially specified target rotation. In the latter case, freely estimated factor loadings are designated by "NA" values and rotation will be conducted using Browne's (1972a, 1972b, 2001) method for a partially-specified target rotation. Second, if any other rotation option is chosen then all rotated loadings matrices (and assorted output) will be aligned (but not rotated) with the target solution.

bootstrapSE (Logical) Computes bootstrap standard errors. All bootstrap samples are aligned to the global minimum solution. Defaults to bootstrapSE = FALSE (no standard errors).

numBoot (Numeric) The number bootstraps. Defaults to numBoot = 1000.

CILevel (Numeric) The confidence level (between 0 and 1) of the bootstrap confidence interval. Defaults to CILevel = .95.

Seed (Numeric) Starting seed for reproducible bootstrap results and factor rotations. Defaults to Seed = 1.

digits (Numeric) Rounds the values to the specified number of decimal places. Defaults to digits = NULL (no rounding).

faControl (List) A list of optional parameters passed to the factor extraction (faX) function.
  • treatHeywood: (Logical) In fals, if treatHeywood is true, a penalized least squares function is used to bound the communality estimates below 1.0. Defaults to treatHeywood = TRUE.
  • nStart: (Numeric) The number of starting values to be tried in faml. Defaults to nStart = 10.
  • start: (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to start = NULL.
  • maxCommunality: (Numeric) In faml, set the maximum communality value for the estimated solution. Defaults to maxCommunality = .995.
  • epsilon: (Numeric) In fapa, the numeric threshold designating when the algorithm has converged. Defaults to epsilon = 1e-4.
• **communality**: (Character) The method used to estimate the initial communality values in fapa. Defaults to communality = 'SMC'.
  - "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
  - "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
  - "unity": Initial communalities equal 1.0 for all variables.

• **maxItr**: (Numeric) In fapa, the maximum number of iterations to reach convergence. Defaults to maxItr = 15,000.

rotateControl (List) A list of control values to pass to the factor rotation algorithms.

• **numberStarts**: (Numeric) The number of random (orthogonal) starting configurations for the chosen rotation method (e.g., oblimin). The first rotation will always commence from the unrotated factors orientation. Defaults to numberStarts = 10.

• **gamma**: (Numeric) This is a tuning parameter (between 0 and 1, inclusive) for an oblimin rotation. See the GPArotation library’s oblimin documentation for more details. Defaults to gamma = 0 (i.e., a quartimin rotation).

• **delta**: (Numeric) This is a tuning parameter for the geomin rotation. It adds a small number (default = .01) to the squared factor loadings before computing the geometric means in the discrepancy function.

• **kappa**: (Numeric) The main parameterization of the Crawford-Ferguson (CF) rotations (i.e., "cfT" and "cfQ" for orthogonal and oblique CF rotation, respectively). Defaults to kappa = 0.

• **k**: (Numeric) A specific parameter of the simplimax rotation. Defaults to k = the number of observed variables.

• **standardize**: (Character) The standardization routine used on the unrotated factor structure. The three options are "none", "Kaiser", and "CM". Defaults to standardize = "none".
  - "none": No standardization is applied to the unrotated factor structure.
  - "Kaiser": Use a factor structure matrix that has been normed by Kaiser’s method (i.e., normalize all rows to have a unit length).
  - "CM": Use a factor structure matrix that has been normed by the Cureton-Mulaik method.

• **epsilon**: (Numeric) The rotational convergence criterion to use. Defaults to epsilon = 1e-5.

• **power**: (Numeric) Raise factor loadings the n-th power in the promaxQ rotation. Defaults to power = 4.

• **maxItr**: (Numeric) The maximum number of iterations for the rotation algorithm. Defaults to maxItr = 15000.

Values to be passed to the cor function.

• **use**: (Character) A character string giving a method for computing correlations in the presence of missing values: "everything" (the default), "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".
• **method**: (Character) A character string indicating which correlation coefficient is to be computed: "pearson" (the default), "kendall", or "spearman".

• **na.rm**: (Logical) Should missing values be removed (TRUE) or not (FALSE)?

### Details

- **Global Minimum**: This function uses several random starting configurations for factor rotations in an attempt to find the global minimum solution. However, this function is not guaranteed to find the global minimum. Furthermore, the global minimum solution need not be more psychologically interpretable than any of the local solutions (cf. Rozeboom, 1992). As is recommended, our function returns all local solutions so users can make their own judgements.

- **Finding clusters of local minima**: We find local-solution sets by sorting the rounded rotation complexity values (to the number of digits specified in the epsilon argument of the rotateControl list) into sets with equivalent values. For example, by default epsilon = 1e-5. and thus will only evaluate the complexity values to five significant digits. Any differences beyond that value will not effect the final sorting.

### Value

The `faMain` function will produce a lot of output in addition to the rotated factor pattern matrix and the factor correlations.

- **R**: (Matrix) Returns the correlation matrix, useful when raw data are supplied.

- **loadings**: (Matrix) The rotated factor solution with the lowest evaluated discrepancy function. This solution has the lowest discrepancy function of the examined random starting configurations. It is not guaranteed to find the "true" global minimum. Note that multiple (or even all) local solutions can have the same discrepancy functions.

- **Phi**: (Matrix) The factor correlations of the rotated factor solution with the lowest evaluated discrepancy function (see Details).

- **facIndeterminacy**: (Vector) A vector (with length equal to the number of factors) containing Guttman’s (1955) index of factor indeterminacy for each factor.

- **h2**: (Vector) The vector of final communality estimates.

- **loadingsSE**: (Matrix) The matrix of factor-loading standard errors across the bootstrapped factor solutions. Each matrix element is the standard deviation of all bootstrapped factor loadings for that element position.

- **CILevel** (Numeric) The user-defined confidence level (between 0 and 1) of the bootstrap confidence interval. Defaults to CILevel = .95.

- **loadingsCIupper**: (Matrix) Contains the upper confidence interval of the bootstrapped factor loadings matrix. The confidence interval width is specified by the user.

- **loadingsCIlower**: (Matrix) Contains the lower confidence interval of the bootstrapped factor loadings matrix. The confidence interval width is specified by the user.

- **PhiSE**: (Matrix) The matrix of factor correlation standard errors across the bootstrapped factor solutions. Each matrix element is the standard deviation of all bootstrapped factor correlations for that element position.

- **PhiCIupper**: (Matrix) Contains the upper confidence interval of the bootstrapped factor correlation matrix. The confidence interval width is specified by the user.
• **PhiCIlower**: (Matrix) Contains the lower confidence interval of the bootstrapped factor correlation matrix. The confidence interval width is specified by the user.

• **facIndeterminacySE**: (Matrix) A row vector containing the standard errors of Guttman’s (1955) factor indeterminacy indices across the bootstrap factor solutions.

• **localSolutions**: (List) A list containing all local solutions in ascending order of their factor loadings, rotation complexity values (i.e., the first solution is the "global" minimum). Each solution returns the
  – **loadings**: (Matrix) the factor loadings,
  – **Phi**: (Matrix) factor correlations,
  – **RotationComplexityValue**: (Numeric) the complexity value of the rotation algorithm,
  – **facIndeterminacy**: (Vector) A vector of factor indeterminacy indices for each common factor, and
  – **RotationConverged**: (Logical) convergence status of the rotation algorithm.

• **numLocalSets** (Numeric) How many sets of local solutions with the same discrepancy value were obtained.

• **localSolutionSets**: (List) A list containing the sets of unique local minima solutions. There is one list element for every unique local solution that includes (a) the factor loadings matrix, (b) the factor correlation matrix (if estimated), and (c) the discrepancy value of the rotation algorithm.

• **loadingsArray**: (Array) Contains an array of all bootstrapped factor loadings. The dimensions are factor indicators, factors, and the number of bootstrapped samples (representing the row, column, and depth, respectively).

• **PhiArray**: (Array) Contains an array of all bootstrapped factor correlations. The dimensions are the number of factors, the number of factors, and the number of bootstrapped samples (representing the row, column, and depth, respectively).

• **facIndeterminacyArray**: (Array) Contains an array of all bootstrap factor indeterminacy indices. The dimensions are 1, the number of factors, and the number of bootstrap samples (representing the row, column, and depth order, respectively).

• **faControl**: (List) A list of the control parameters passed to the factor extraction (faX) function.

• **faFit**: (List) A list of additional output from the factor extraction routines.
  – **facMethod**: (Character) The factor extraction routine.
  – **df**: (Numeric) Degrees of Freedom from the maximum likelihood factor extraction routine.
  – **n**: (Numeric) Sample size associated with the correlation matrix.
  – **objectiveFunc**: (Numeric) The evaluated objective function for the maximum likelihood factor extraction routine.
  – **RMSEA**: (Numeric) Root mean squared error of approximation from Steiger & Lind (1980). Note that bias correction is computed if the sample size is provided.
  – **testStat**: (Numeric) The significance test statistic for the maximum likelihood procedure. Cannot be computed unless a sample size is provided.
  – **pValue**: (Numeric) The p value associated with the significance test statistic for the maximum likelihood procedure. Cannot be computed unless a sample size is provided.
  – **gradient**: (Matrix) The solution gradient for the least squares factor extraction routine.
- **maxAbsGradient**: (Numeric) The maximum absolute value of the gradient at the least squares solution.
- **Heywood**: (Logical) TRUE if a Heywood case was produced.
- **convergedX**: (Logical) TRUE if the factor extraction routine converged.
- **convergedR**: (Logical) TRUE if the factor rotation routine converged (for the local solution with the minimum discrepancy value).

- **rotateControl**: (List) A list of the control parameters passed to the rotation algorithm.
- **unSpunSolution**: (List) A list of output parameters (e.g., loadings, Phi, etc) from the rotated solution that was obtained by rotating directly from the unrotated (i.e., unspun) common factor orientation.
- **targetMatrix**: (Matrix) The input target matrix if supplied by the user.
- **Call**: (call) A copy of the function call.

**Author(s)**

- Niels G. Waller (nwaller@umn.edu)
- Casey Giordano (Giord023@umn.edu)
- The authors thank Allie Cooperman and Hoang Nguyen for their help implementing the standard error estimation and the Cureton-Mulaik standardization procedure.

**References**


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

## Example 1

```
## Generate an orthogonal factor model
lambda <- matrix(c(0.41, 0.00, 0.00,
                   0.45, 0.00, 0.00,
                   0.53, 0.00, 0.00,
                   0.00, 0.66, 0.00,
                   0.00, 0.38, 0.00,
                   0.00, 0.66, 0.00,
                   0.00, 0.00, 0.68,
                   0.00, 0.00, 0.56,
                   0.00, 0.00, 0.55),
               nrow = 9, ncol = 3, byrow = TRUE)

## Generate factor correlation matrix
Phi <- matrix(0.50, nrow = 3, ncol = 3)
diag(Phi) <- 1

## Model-implied correlation matrix
R <- lambda %*% Phi %*% t(lambda)
diag(R) <- 1

## Load the MASS package to create multivariate normal data
library(MASS)

## Generate raw data to perfectly reproduce R
X <- mvrnorm(Sigma = R, mu = rep(0, nrow(R)), empirical = TRUE, n = 300)

## Not run:
## Execute 50 promax rotations from a least squares factor extraction
## Compute 100 bootstrap samples to compute standard errors and
## 80 percent confidence intervals
Out1 <- faMain(X = X,
               numFactors = 3,
               facMethod = "fals",
               rotate = "promaxQ",
               bootstrapSE = TRUE,
               numBoot = 100,
               CILevel = .80,
               faControl = list(treatHeywood = TRUE),
               rotateControl = list(numberStarts = 2,
                                      power = 4,
                                      standardize = "Kaiser"),
               digits = 2)
Out1[,c("loadings", "Phi")]
```
## Example 2

## Load Thurstone's (in)famous box data

```r
data(Thurstone, package = "GPArotation")
```

## Execute 5 oblimin rotations with Cureton-Mulaik standardization

```r
Out2 <- faMain(urLoadings = box26,
    rotate = "oblimin",
    bootstrapSE = FALSE,
    rotateControl = list(numberStarts = 5,
        standardize = "CM",
        gamma = 0,
        epsilon = 1e-6),
    digits = 2)
```

```r
Out2[,c("loadings", "Phi")]
```

## Example 3

## Factor matrix from Browne 1972

```r
lambda <- matrix(c(.664, .322, -.075,
    .688, .248, .192,
    .492, .304, .224,
    .837, -.291, .037,
    .705, -.314, .155,
    .820, -.377, -.104,
    .661, .397, .077,
    .457, .294, -.488,
    .765, .428, .009),
    nrow = 9, ncol = 3, byrow = TRUE)
```

## Create partially-specified target matrix

```r
Targ <- matrix(c(NA, 0, NA,
    NA, 0, NA,
    NA, NA, NA,
    NA, NA, 0,
    NA, NA, 0.
    .7, NA, NA,
    .7, NA, NA),
    nrow = 9, ncol = 3, byrow = TRUE)
```

## Perform target rotation

```r
Out3 <- faMain(urLoadings = lambda,
    rotate = "targetT",
    targetMatrix = Targ,
    digits = 3)$loadings
```

```r
Out3
```
faMAP

Description

Uses Velicer’s MAP (i.e., matrix of partial correlations) procedure to determine the number of components from a matrix of partial correlations.

Usage

faMAP(R, max.fac = 8, Print = TRUE, Plot = TRUE, ...)

Arguments

R input data in the form of a correlation matrix.
max.fac maximum number of dimensions to extract.
Print (logical) Print = TRUE will print complete results.
Plot (logical) Plot = TRUE will plot the MAP values.
... Arguments to be passed to the plot functions (see par).

Value

MAP Minimum partial correlations
MAP4 Minimum partial correlations
fm average of the squared partial correlations after the first m components are partialed out.
fm4 see Velicer, Eaton, & Fava, 2000.
PlotAvgSq A saved object of the original MAP plot (based on the average squared partial r's.)
PlotAvg4th A saved object of the revised MAP plot (based on the average 4th power of the partial r's.)

Author(s)

Niels Waller

References


Examples

```r
# Harman's data (1967, p 80)
# R = matrix(c(
# 1.000, .846, .805, .859, .473, .398, .301, .382,
# .846, 1.000, .881, .826, .376, .326, .277, .415,
# .805, .881, 1.000, .801, .380, .319, .237, .345,
# .859, .826, .801, 1.000, .436, .329, .327, .365,
# .473, .376, .380, .436, 1.000, .762, .730, .629,
# .398, .326, .319, .329, .762, 1.000, .583, .577,
# .301, .277, .237, .327, .730, .583, 1.000, .539,
# .382, .415, .345, .365, .629, .577, .539, 1.000), 8,8)

F <- matrix(c( .4, .1, .0,
               .5, .0, .1,
               .6, .03, .1,
               .4, -.2, .0,
               0, .6, .1,
               .1, .7, .2,
               .3, .7, .1,
               0, .4, .1,
               0, 0, .5,
               .1, -.2, .6,
               -.2, .1, .7),12,3)

R <- F %*% t(F)
diag(R) <- 1
faMAP(R, max.fac = 8, Print = TRUE, Plot = TRUE)
```

Description

This function applies the iterated principal axis factoring method to extract an unrotated factor structure matrix.

Usage

```r
fapa(R, numFactors = NULL, epsilon = 1e-04, communality = "SMC",
     maxItr = 15000, digits = NULL)
```

Arguments

- **R** *(Matrix)* A correlation matrix to be analyzed.
- **numFactors** *(Numeric)* The number of factors to extract.
fapa

epsilon (Numeric) A numeric threshold to designate whether the function has converged. The default value is 1e-4.

communality (Character) The routine requires an initial estimate of the communality values. There are three options (see below) with "SMC" (i.e., squared multiple correlation) being the default.

- "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables. The following equation is employed to find the squared multiple correlation: \(1 - \frac{1}{\text{diag}(R^{-1})}\).
- "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
- "unity": Initial communalities equal 1.0 for all variables.

maxItr (Numeric) The maximum number of iterations to reach convergence. The default is 15,000.

digits (Scalar) The number of digits with which to round all output.

Details

- **Initial communality estimate**: The choice of the initial communality estimate can impact the resulting principal axis factor solution.
  - **Impact on the Estimated Factor Structure**: According to Widaman and Herringer (1985), the initial communality estimate does not have much bearing on the resulting solution when a stringent convergence criterion is used. In their analyses, a convergence criterion of .001 (i.e., slightly less stringent than the default of 1e-4) is sufficiently stringent to produce virtually identical communality estimates irrespective of the initial estimate used. Based on their findings, it is not recommended to use a convergence criterion lower than 1e-3.
  - **Impact on the Iteration Procedure**: The initial communality estimates have little impact on the final factor structure but they can impact the iterated procedure. It is possible that poor communality estimates produce a non-positive definite correlation matrix (i.e., eigenvalues \(\leq 0\)) whereas different communality estimates result in a converged solution. If the fapa procedure fails to converge due to a non-positive definite matrix, try using different communality estimates before changing the convergence criterion.

Value

The main output is the matrix of unrotated factor loadings.

- loadings (Matrix) A matrix of unrotated factor loadings extracted via iterated principal axis factoring.
- h2 (Vector) A vector containing the resulting communality values.
- iterations (Numeric) The number of iterations required to converge.
- converged (Logical) TRUE if the iterative procedure converged.
- faControl (List) A list of the control parameters used to generate the factor structure.
  - epsilon (Numeric) The convergence criterion used for evaluating each iteration.
  - communality (Character) The method for estimating the initial communality values.
  - maxItr (Numeric) The maximum number of allowed iterations to reach convergence.
Author(s)

- Casey Giordano (Giord023@umn.edu)
- Niels G. Waller (nwaller@umn.edu)

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, Sli, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

```r
## Generate an example factor structure matrix
lambda <- matrix(c(.62, .00, .00,
                   .54, .00, .00,
                   .41, .00, .00,
                   .00, .31, .00,
                   .00, .58, .00,
                   .00, .62, .00,
                   .00, .00, .38,
                   .00, .00, .43,
                   .00, .00, .37),
                   nrow = 9, ncol = 3, byrow = TRUE)

## Find the model implied correlation matrix
R <- lambda %*% t(lambda)
diag(R) <- 1

## Extract factors using the fapa function
Out1 <- fapa(R = R,
             numFactors = 3,
             communality = "SMC")

## Call fapa through the factExtract function
Out2 <- faX(R = R,
            numFactors = 3,
            facMethod = "fapa",
            faControl = list(communality = "maxr",
                              epsilon = 1e-4))

## Check for equivalence of the two results
all.equal(Out1$loadings, Out2$loadings)
```
fareg Regularized Factor Analysis

Description

This function applies the regularized factoring method to extract an unrotated factor structure matrix.

Usage

fareg(R, numFactors = 1, facMethod = "rls")

Arguments

- R: (Matrix) A correlation matrix to be analyzed.
- facMethod: (Character) "rls" for regularized least squares estimation or "rml" for regularized maximum likelihood estimation. Default: facMethod = "rls".

Value

The main output is the matrix of unrotated factor loadings.

- loadings: (Matrix) A matrix of unrotated factor loadings.
- h2: (Vector) A vector of estimated communality values.
- L: (Numeric) Value of the estimated penalty parameter.
- Heywood: (Logical) TRUE if a Heywood case is detected (this should never happen).

Author(s)

Niels G. Waller (nwaller@umn.edu)

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, Sli, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, orderFactors, print.faMain, promaxQ, summary.faMain
Examples

data("HW")

# load first HW data set

RHW <- cor(x = HW$HW6)

# Compute principal axis factor analysis
fapaOut <- faMain(R = RHW,
  numFactors = 3,
  facMethod = "fapa",
  rotate = "oblimin",
  faControl = list(treatHeywood = FALSE))

fapaOut$faFit$Heywood
round(fapaOut$h2, 2)

# Conduct a regularized factor analysis
regOut <- fareg(R = RHW,
  numFactors = 3,
  facMethod = "rls")
regOut$L
regOut$Heywood

# rotate regularized loadings and align with population structure
regOutRot <- faMain(urLoadings = regOut$loadings,
  rotate = "oblimin")

# Align
FHW <- faAlign(HW$popLoadings, fapaOut$loadings)$F2
Freg <- faAlign(HW$popLoadings, regOutRot$loadings)$F2

AllSolutions <- round(cbind(HW$popLoadings, Freg, FHW),2)
colnames(AllSolutions) <- c("F1", "F2", "F3", "Fr1", "Fr2", "Fr3",
  "Fhw1", "Fhw2", "Fhw3")
AllSolutions

rmsdHW <- rmsd(HW$popLoadings, FHW,
  IncludeDiag = FALSE,
  Symmetric = FALSE)

rmsdReg <- rmsd(HW$popLoadings, Freg,
  IncludeDiag = FALSE,
  Symmetric = FALSE)

cat("\nrmsd HW = ", round(rmsdHW,3),
  "\nrmsd reg = ", round(rmsdReg,3))
faScores

Factor Scores

Description

This function computes factor scores by various methods. The function will accept an object of class FaMain or, alternatively, user-input factor pattern (i.e., Loadings) and factor correlation (Phi) matrices.

Usage

faScores(X = NULL, faMainObject = NULL, Loadings = NULL, Phi = NULL, Method = "Thurstone")

Arguments

X (Matrix) An N x variables data matrix. If X is a matrix of raw scores then faScores will convert the data to z scores.

faMainObject (Object of class FaMain) The returned object from a call to FaMain. Default = NULL

Loadings (Matrix) A factor pattern matrix. Default = NULL.

Phi (Matrix) A factor correlation matrix. Default = NULL. If a factor pattern is entered via the Loadings argument but Phi = NULL the program will set Phi to an identity matrix.

Method (Character) Factor scoring method. Defaults to the Thurstone or regression based method. Available options include:

- **Thurstone** Generates regression based factor score estimates.
- **Bartlett** Generates Bartlett method factor score estimates.
- **tenBerge** Generates factor score estimates with correlations identical to that found in Phi.
- **Anderson** The Anderson Rubin method. Generates uncorrelated factor score estimates. This method is only appropriate for orthogonal factor models.
- **Harman** Generates estimated factor scores by Harman’s idealized variables method.
- **PCA** Returns unrotated principal component scores.
faScores can be used to calculate estimated factor scores by various methods. In general, to calculate score estimates, users must input a data matrix $X$ and either (a) an object of class faMain or (b) a factor loadings matrix, Loadings and an optional (for oblique models) factor correlation matrix Phi. The one exception to this rule concerns scores for the principal components model. To calculate unrotated PCA scores (i.e., when Method = "PCA") users need only enter a data matrix, X.

Value

- fscores A matrix om common factor score estimates.
- Method The method used to create the factor score estimates.
- W The factor scoring coefficient matrix.
- Z A matrix of standardized data used to create the estimated factor scores.

Author(s)

Niels Waller

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

```r
lambda.Pop <- matrix(c(.41, .00, .00, .45, .00, .00, .53, .00, .00, .00, .66, .00, .00, .38, .00), nrow=3, byrow=TRUE)
```
faScores

\begin{verbatim}
.00, .66, .00, 
.00, .00, .68, 
.00, .00, .56, 
.00, .00, .55), 
nrow = 9, ncol = 3, byrow = TRUE)
NVar <- nrow(lambda.Pop)
NFac <- 3

## Factor correlation matrix
Phi.Pop <- matrix(.50, nrow = 3, ncol = 3)
diag(Phi.Pop) <- 1

# Model-implied correlation matrix
R <- lambda.Pop %*% Phi.Pop %*% t(lambda.Pop)
diag(R) <- 1

# Generate population data to perfectly reproduce pop R
Out <- simFA( Model = list(Model = "oblique"),
              Loadings = list(FacPattern = lambda.Pop),
              Phi = list(PhiType = "user",
                         UserPhi = Phi.Pop),
              FactorScores = list(FS = TRUE,
                                   CFSeed = 1,
                                   SFSeed = 2,
                                   EFSeed = 3,
                                   Population = TRUE,
                                   NFacScores = 100),
              Seed = 1)

PopFactorScores <- Out$Scores$FactorScores
X <- PopObservedScores <- Out$Scores$ObservedScores

fout <- faMain(X = X,
               numFactors = 3,
               facMethod = "fals",
               rotate = "oblimin")

print( round(fout$loadings, 2) )
print( round(fout$Phi,2) )

fload <- fout$loadings
Phi <- fout$Phi

fsOut <- faScores(X = X,
                   faMainObject = fout,
                   Method = "Thurstone")

fscores <- fsOut$fscores

print( round(cor(fscores), 2 ))
\end{verbatim}
faSort <- PopFactorScores[,1:NFac]
SpecificFS <- PopFactorScores[, (NFac+1):(NFac+NVar)]
ErrorFS <- PopFactorScores[, (NFac + NVar + 1):(NFac + 2*NVar)]

print( cor(fscores, CommonFS) )

faSort

Sort a factor loadings matrix

Description
faSort takes an unsorted factor pattern or structure matrix and returns a sorted matrix with (possibly) reflected columns. Sorting is done such that variables that load on a common factor are grouped together for ease of interpretation.

Usage
faSort(fmat, phi = NULL, BiFactor = FALSE, salient = 0.25,
       reflect = TRUE)

Arguments
fmat factor loadings (pattern or structure) matrix.
phi factor correlation matrix. Default = NULL. If reflect = TRUE then phi will be corrected to match the new factor orientations.
BiFactor (logical) Is the solution a bifactor model?
salient factor markers with loadings >= abs(salient) will be saved in the markers list. Note that a variable can be a marker of more than one factor.
reflect (logical) if reflect = TRUE then the factors will be reflected such that salient loadings are mostly positive.

Value
loadings sorted factor loadings matrix.
phi reflected factor correlation matrix when phi is given as an argument.
markers A list of factor specific markers with loadings >= abs(salient). Markers are sorted by the absolute value of the salient factor loadings.
sortOrder sorted row numbers.
SEmat The SEmat is a so-called Start-End matrix that lists the first (start) and last (end) row for each factor in the sorted pattern matrix.

Author(s)
Niels Waller
See Also

faSort

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

```r
set.seed(123)
F <- matrix(c(.5, 0,
             .6, 0,
             0, .6,
             .6, 0,
             0, .5,
             .7, 0,
             0, .7,
             0, .6), nrow = 8, ncol = 2, byrow=TRUE)
Rex1 <- F %*% t(F); diag(Rex1) <- 1

Items <- c("1. I am often tense.\n",  
           "2. I feel anxious much of the time.\n",  
           "3. I am a naturally curious individual.\n",  
           "4. I have many fears.\n",  
           "5. I read many books each year.\n",  
           "6. My hands perspire easily.\n",  
           "7. I have many interests.\n",  
           "8. I enjoy learning new words.\n")

exampleOut <- fals(R = Rex1, nfactors = 2)

# Varimax rotation
Fload <- varimax(exampleOut$loadings)$loadings$

# Add some row labels
rownames(Fload) <- paste0("V", 1:nrow(Fload))

cat("Unsorted factor loadings\n")
print(round( Fload, 2) )

# Sort items and reflect factors
out1 <- faSort(fmat = Fload,
               salient = .25,  
               reflect = TRUE)

FloadSorted <- out1$loadings

cat("Sorted factor loadings\n")
print(round( FloadSorted, 2) )
```
faStandardize

standardize the unrotated factor loadings using two methods: Kaiser's normalization and Cureton-Mulaik standardization.

faStandardize

# Print sorted items
cat("\n Items sorted by Factor\n")
cat("\n",Items[out1$sortOrder])

faStandardize

Standardize the Unrotated Factor Loadings

Description

This function standardizes the unrotated factor loadings using two methods: Kaiser's normalization and Cureton-Mulaik standardization.

Usage

faStandardize(method, lambda)

Arguments

method

(Character) The method used for standardization. There are three options: "none", "Kaiser", and "CM".
- "none": No standardization is conducted on the unrotated factor loadings matrix.
- "Kaiser": The rows of the unrotated factor loadings matrix are rescaled to have unit-lengths.
- "CM": Apply the Cureton-Mulaik standardization to the unrotated factor loadings matrix.

lambda

(Matrix) The unrotated factor loadings matrix (or data frame).

Value

The resulting output can be used to standardize the factor loadings as well as providing the inverse matrix used to unstandardize the factor loadings after rotating the factor solution.

- Dv: (Matrix) A diagonal weight matrix used to standardize the unrotated factor loadings. Pre-multiplying the loadings matrix by the diagonal weight matrix (i.e., Dv
- DvInv: (Matrix) The inverse of the diagonal weight matrix used to standardize. To unstandardize the ultimate rotated solution, pre-multiply the rotated factor loadings by the inverse of Dv (i.e., DvInv
- lambda: (Matrix) The standardized, unrotated factor loadings matrix.
- unstndLambda: (Matrix) The original, unstandardized, unrotated factor loadings matrix.

References


faX

Factor Extraction (faX) Routines

Description
This function can be used to extract an unrotated factor structure matrix using the following algorithms: (a) unweighted least squares ("fals"); (b) maximum likelihood ("faml"); (c) iterated principal axis factoring ("fapa"); and (d) principal components analysis ("pca").

Usage
faX(R, n = NULL, numFactors = NULL, facMethod = "fals", faControl = NULL, digits = NULL)

Arguments
- **R** (Matrix) A correlation matrix used for factor extraction.
- **n** (Numeric) Sample size associated with the correlation matrix. Defaults to n = NULL.
- **numFactors** (Numeric) The number of factors to extract for subsequent rotation.
- **facMethod** (Character) The method used for factor extraction. The supported options are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa" for iterated principal axis factoring, and "pca" for principal components analysis. The default method is "fals".
  - "fals": Factors are extracted using the unweighted least squares estimation procedure using the fals function.
  - "faml": Factors are extracted using the maximum likelihood estimation procedure using the factanal function.
  - "faregLS": Factors are extracted using regularized least squares factor analysis using the fareg function.
  - "faregML": Factors are extracted using regularized maximum likelihood factor using the fareg function.
  - "fapa": Factors are extracted using the iterated principal axis factoring estimation procedure using the fapa function.
  - "pca": Principal components are extracted.
- **faControl** (List) A list of optional parameters passed to the factor extraction (faX) function.
  - **treatHeywood** (Logical) In fals, if treatHeywood is true, a penalized least squares function is used to bound the communality estimates below 1.0. Defaults to treatHeywood = TRUE.

See Also
Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain
• **nStart**: (Numeric) The number of starting values to be tried in `faml`. Defaults to `nStart = 10`.

• **start**: (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to `start = NULL`.

• **maxCommunality**: (Numeric) In `faml`, set the maximum communality value for the estimated solution. Defaults to `maxCommunality = .995`.

• **epsilon**: (Numeric) In `fapa`, the numeric threshold designating when the algorithm has converged. Defaults to `epsilon = 1e-4`.

• **communality**: (Character) The method used to estimate the initial communality values in `fapa`. Defaults to `communality = 'SMC'`.
  - "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
  - "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
  - "unity": Initial communalities equal 1.0 for all variables.

• **maxItr**: (Numeric) In `fapa`, the maximum number of iterations to reach convergence. Defaults to `maxItr = 15,000`.

• **digits**: (Numeric) Rounds the values to the specified number of decimal places. Defaults to `digits = NULL` (no rounding).

**Details**

- **Initial communality estimate**: According to Widaman and Herringer (1985), the initial communality estimate does not have much bearing on the resulting solution *when a stringent convergence criterion is used*. In their analyses, a convergence criterion of .001 (i.e., slightly less stringent than the default of 1e-4) is sufficiently stringent to produce virtually identical communality estimates irrespective of the initial estimate used. It should be noted that all four methods for estimating the initial communality in Widaman and Herringer (1985) are the exact same used in this function. Based on their findings, it is not recommended to use a convergence criterion lower than 1e-3.

**Value**

This function returns a list of output relating to the extracted factor loadings.

- **loadings**: (Matrix) An unrotated factor structure matrix.

- **h2**: (Vector) Vector of final communality estimates.

- **faFit**: (List) A list of additional factor extraction output.
  - **facMethod**: (Character) The factor extraction routine.
  - **df**: (Numeric) Degrees of Freedom from the maximum likelihood factor extraction routine.
  - **n**: (Numeric) Sample size associated with the correlation matrix.
  - **objectiveFunc**: (Numeric) The evaluated objective function for the maximum likelihood factor extraction routine.
  - **RMSEA**: (Numeric) Root mean squared error of approximation from Steiger & Lind (1980). Note that bias correction is computed if the sample size is provided.
- **testStat**: (Numeric) The significance test statistic for the maximum likelihood procedure. Cannot be computed unless a sample size is provided.
- **pValue**: (Numeric) The p value associated with the significance test statistic for the maximum likelihood procedure. Cannot be computed unless a sample size is provided.
- **gradient**: (Matrix) The solution gradient for the least squares factor extraction routine.
- **maxAbsGradient**: (Numeric) The maximum absolute value of the gradient at the least squares solution.
- **Heywood**: (Logical) TRUE if a Heywood case was produced.
- **converged**: (Logical) TRUE if the least squares or principal axis factor extraction routine converged.

**Author(s)**
- Casey Giordano (Giord023@umn.edu)
- Niels G. Waller (nwaller@umn.edu)

**References**

**See Also**
Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

**Examples**
```r
## Generate an example factor structure matrix
lambda <- matrix(c(.62, .00, .00,
                   .54, .00, .00,
                   .41, .00, .00,
                   .00, .31, .00,
                   .00, .58, .00,
                   .00, .62, .00,
                   .00, .00, .38,
                   .00, .00, .43,
                   .00, .00, .37),
                   nrow = 9, ncol = 3, byrow = TRUE)

## Find the model implied correlation matrix
R <- lambda %*% t(lambda)
diag(R) <- 1
```
## Extract (principal axis) factors using the factExtract function

Out1 <- faX(R = R,
    numFactors = 3,
    facMethod = "fapa",
    faControl = list(communality = "maxr",
                      epsilon = 1e-4))

## Extract (least squares) factors using the factExtract function

Out2 <- faX(R = R,
    numFactors = 3,
    facMethod = "fals",
    faControl = list(treatHeywood = TRUE))

---

**FMP**

*Estimate the coefficients of a filtered monotonic polynomial IRT model*

### Description

Estimate the coefficients of a filtered monotonic polynomial IRT model.

### Usage

FMP(data, thetaInit, item, startvals, k = 0, eps = 1e-06)

### Arguments

- **data**: N(subjects)-by-p(items) matrix of 0/1 item response data.
- **thetaInit**: Initial theta (θ) surrogates (e.g., calculated by svdNorm).
- **item**: Item number for coefficient estimation.
- **startvals**: Start values for function minimization. Start values are in the gamma metric (see Liang & Browne, 2015)
- **k**: Order of monotonic polynomial = 2k+1 (see Liang & Browne, 2015). k can equal 0, 1, 2, or 3.
- **eps**: Step size for gradient approximation, default = 1e-6. If a convergence failure occurs during function optimization reducing the value of eps will often produce a converged solution.

### Details

As described by Liang and Browne (2015), the filtered polynomial model (FMP) is a quasi-parametric IRT model in which the IRF is a composition of a logistic function and a polynomial function, \( m(\theta) \), of degree \( 2k + 1 \). When \( k = 0 \), \( m(\theta) = b_0 + b_1 \theta \) (the slope intercept form of the 2PL). When \( k = 1 \), \( 2k + 1 \) equals 3 resulting in \( m(\theta) = b_0 + b_1 \theta + b_2 \theta^2 + b_3 \theta^3 \). Acceptable values of \( k = 0,1,2,3 \). According to Liang and Browne, the "FMP IRF may be used to approximate any IRF with a continuous derivative arbitrarily closely by increasing the number of parameters in the monotonic polynomial" (2015, p. 2) The FMP model assumes that the IRF is monotonically increasing, bounded by 0 and 1, and everywhere differentiable with respect to theta (the latent trait).
**Value**

- **b** Vector of polynomial coefficients.
- **gamma** Polynomial coefficients in gamma metric (see Liang & Browne, 2015).
- **FHAT** Function value at convergence.
- **counts** Number of function evaluations during minimization (see optim documentation for further details).
- **AIC** Pseudo scaled Akaike Information Criterion (AIC). Candidate models that produce the smallest AIC suggest the optimal number of parameters given the sample size. Scaling is accomplished by dividing the non-scaled AIC by sample size.
- **BIC** Pseudo scaled Bayesian Information Criterion (BIC). Candidate models that produce the smallest BIC suggest the optimal number of parameters given the sample size. Scaling is accomplished by dividing the non-scaled BIC by sample size.
- **convergence** Convergence = 0 indicates that the optimization algorithm converged; convergence=1 indicates that the optimization failed to converge.

**Author(s)**

Niels Waller

**References**


**Examples**

```r
## Not run:
## In this example we will generate 2000 item response vectors
## for a k = 1 order filtered polynomial model and then recover
## the estimated item parameters with the FMP function.

k <- 1 # order of polynomial
NSubjects <- 2000

## generate a sample of 2000 item response vectors
## for a k = 1 FMP model using the following
## coefficients
b <- matrix(c(1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1, 1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1, 1.282, 1.063, -0.103, 0.003, 0, 0, 0, 0, 1), nrow = 3, byrow = TRUE)
```
0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,
1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,
-0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,
0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,
0.122, 0.601, -0.082, 0.044, 0, 0, 0, 0, 1,
1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,
-0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,
-0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,
0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,
-0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,
0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,
1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,
0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,
0.429, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,
0.089, 0.785, -0.065, 0.018, 0, 0, 0, 0, 1,
-0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,
0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,
0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,
-0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,
0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1,
rown=23, ncol=9, byrow=TRUE)

ex1.data<-genFMPData(NSubj = NSubjects, bParams = b, seed = 345)$data

## number of items in the data matrix
NItems <- ncol(ex1.data)

# compute (initial) surrogate theta values from
# the normed left singular vector of the centered
# data matrix
thetaInit <- svdNorm(ex1.data)

## earlier we defined k = 1
if(k == 0) {
    startVals <- c(1.5, 1.5)
    bmat <- matrix(0, NItems, 6)
    colnames(bmat) <- c(paste("b", 0:1, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}
if(k == 1) {
    startVals <- c(1.5, 1.5, .10, .10)
    bmat <- matrix(0, NItems, 8)
    colnames(bmat) <- c(paste("b", 0:3, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}
if(k == 2) {
    startVals <- c(1.5, 1.5, .10, .10, .10, .10)
    bmat <- matrix(0, NItems, 10)
    colnames(bmat) <- c(paste("b", 0:5, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}
if(k == 3) {
    startVals <- c(1.5, 1.5, .10, .10, .10, .10, .10, .10)
    bmat <- matrix(0, NItems, 12)
    colnames(bmat) <- c(paste("b", 0:7, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}
# estimate item parameters and fit statistics
for(i in 1:NItems){
  out <- FMP(data = ex1.data, thetaInit, item = i, startvals = startVals, k = k)
  Nb <- length(out$b)
  bmat[i,1:Nb] <- out$b
  bmat[i,Nb+1] <- out$FHAT
  bmat[i,Nb+2] <- out$AIC
  bmat[i,Nb+3] <- out$BIC
  bmat[i,Nb+4] <- out$convergence
}

# print output
print(bmat)

## End(Not run)

---

**FMPMonotonicityCheck**  
*Utility function for checking FMP monotonicity*

### Description

Utility function for checking whether candidate FMP coefficients yield a monotonically increasing polynomial.

### Usage

```r
FMPMonotonicityCheck(b, lower = -20, upper = 20, PLOT = FALSE)
```

### Arguments

- `b`  
  A vector of 8 polynomial coefficients ($b$) for $m(\theta) = b_0 + b_1 \theta + b_2 \theta^2 + b_3 \theta^3 + b_4 \theta^4 + b_5 \theta^5 + b_6 \theta^6 + b_7 \theta^7$.
- `lower`, `upper`  
  $\theta$ bounds for monotonicity check.
- `PLOT`  
  Logical (default = FALSE). If PLOT = TRUE the function will plot the original polynomial function for $\theta$ between lower and upper.

### Value

- `increasing`  
  Logical indicating whether function is monotonically increasing.
- `minDeriv`  
  Minimum value of the derivative for the polynomial.
- `minTheta`  
  Value of $\theta$ at derivative minimum.

### Author(s)

Niels Waller
Examples

```r
## A set of candidate coefficients for an FMP model.
## These coefficients fail the test and thus
## should not be used with genFMPdata to generate
## item response data that are consistent with an
## FMP model.
b <- c(1.21, 1.87, -1.02, 0.18, 0.18, 0, 0, 0)
FMPMonotonicityCheck(b)
```

---

**fungible**

*Generate Fungible Regression Weights*

### Description

Generate fungible weights for OLS Regression Models.

### Usage

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

### Arguments

- `R.X`: p x p Predictor correlation matrix.
- `r.xy`: 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


Examples

```r
## Predictor correlation matrix
R.X <- matrix(c(1.00, .56, .77, .56, 1.00, .73, .77, .73, 1.00), 3, 3)

## vector of predictor-criterion correlations
rxy <- c(.39, .34, .38)

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

## 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, rxy, r.yhata.yhatb, sets = Nsets, print = TRUE)
```

fungible Extrema

Locate Extrema of Fungible Regression Weights

Description

Locate extrema of fungible regression weights.

Usage

```r
fungibleExtrema(R.X, rxy, r.yhata.yhatb, Nstarts = 100, MaxMin = "Max")
```
Arguments

- **R.X**: \( p \times p \) Predictor variable correlation matrix.
- **rxy**: \( p \times 1 \) 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 \( \cos(a,b) \).

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**: \( p \times 1 \) 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, 0.69, 0.49, 0.39, 0.69, 1.00, 0.38, 0.19, 0.49, 0.38, 1.00, 0.27, 0.39, 0.19, 0.27, 1.00), nrow=4)

b <- c(0.39, 0.22, 0.02, 0.43)
```
rxy <- 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")

## Scale to replicate Koopman
a <- output$a
a.old <- a
aRa <- t(a) %*% R.X %*% a

## Scale a such that a' R a = .68659
## vc = variance of composite
vc <- aRa
## sf = scale factor
sf <- .68659/vc
a <- as.numeric(sqrt(sf)) * a
cat("\n\nKoopman Scaling\n")
print(round(a,2))

## End(Not run)

---

table

fungibleL Generate Fungible Logistic Regression Weights

Description

Generate fungible weights for Logistic Regression Models.

Usage

fungibleL(X, y, Nsets = 1000, method = "LLM", RsqDelta = NULL,
          rLaLb = NULL, s = 0.3, 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.

fungibleL

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.
lnLr: 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

# low : low birth rate (Ø >= 2500 grams, 1 < 2500 grams)
# race: 1 = white, 2 = black, 3 = other
fungibleR

# 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 = 0.005, s = 0.3)

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

print(summary(BWght.out))
print(fungible.LLM$call)
print(fungible.LLM$ftable)
cat("\nMLE log likelihood = ", fungible.LLM$lnLML,
    "fungible log likelihood = ", fungible.LLM$lnLf)
cat("\nPseudo Rsq = ", round(fungible.LLM$pseudoRsq, 3))
cat("fungible Pseudo Rsq = ", round(fungible.LLM$fungibleRsq, 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-define 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 = 0, 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.
RstarLp A fungible correlation matrix with a fixed minimum eigenvalue (RstarLp can be PD, PSD, or ID).
s Scaling constant for Rstar.
sLp Scaling constant for RstarLp.
Delta Vector in the null space of vecp(Beta Beta’).
Q Left null space of Beta.
FrobNorm Frobenius norm ||R - Rstar||_F.
FrobNormLp Frobenius norm ||R - RstarLp||_F given random Delta.
converged 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_1j = .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("Model 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(" *** Check Results ***")
for(i in 1:5){
cat("+++++++++++++++++++++++++++++++++++++++++++++++++")
cat("Rstar", i,"\n")
print(round(Rstar.list[[i]], 2),)
cat("eigenvalues of Rstar", i,"\n")
print(eigen(Rstar.list[[i]])$values)
cat("Beta's 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("R: a user-specified seed matrix")
print(round(out$R,3))

## Rstar

print(round(out$Rstar,3))

## Rstar: A random fungible correlation matrix for R

print( round(out$Rstar,3) )

## Coefficient of determination when using R

print( t(Beta) %*% R %*% Beta )

## Coefficient of determination when using Rstar

print( t(Beta) %*% out$Rstar %*% Beta )

### Eigenvalues of R

print(round(eigen(out$R)$values, 9))

### Eigenvalues of Rstar

print(round(eigen(out$Rstar)$values, 9))

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

print(out$FrobNorm)

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

print(round(out$RstarLp, 3))

### Eigenvalues of RstarLp

print(eigen(out$RstarLp)$values, digits = 9)

### Coefficient of determination when using RstarLp

print( t(Beta) %*% out$RstarLp %*% Beta )

### Check function convergence

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

# Example 3

library(fungible)

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(5,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 Frobeius 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)$values,
     type = "b",
     lwd = 2,
     main = "Scree Plots for R and RstarLp",
     cex.main = 1.5,
     ylim = c(-2,6),
     ylab = "Eigenvalues",
     xlab = "Dimensions")
points(1:25,eigen(CloseR)$values,
       type = "b",
       lty = 2,
       lwd = 2,
       col = "red")
abline(h = 0, col = "grey")
legend(legend=c(expression(paste(lambda[i]~" of R",sep = "")),
                 expression(paste(lambda[i]~" of RstarLp",sep = ""))),
        lty=c(1,2),
        x = 17,y = 5.75,
        cex = 1.5,
        col=c("black","red"),
        text.width = 5.5,
        lwd = 2)

---

**FUP**

*Estimate the coefficients of a filtered unconstrained polynomial IRT model*

---

**Description**

Estimate the coefficients of a filtered unconstrained polynomial IRT model.

**Usage**

FUP(data, thetaInit, item, startvals, k = 0)
Arguments

data N(subjects)-by-p(items) matrix of 0/1 item response data.
thetaInit Initial theta surrogates (e.g., calculated by svdNorm).
item item number for coefficient estimation.
startvals start values for function minimization.
k order of monotonic polynomial = 2k+1 (see Liang & Browne, 2015).

Value

b Vector of polynomial coefficients.
FHAT Function value at convergence.
counts Number of function evaluations during minimization (see optim documentation for further details).
AIC Pseudo scaled Akaike Information Criterion (AIC). Candidate models that produce the smallest AIC suggest the optimal number of parameters given the sample size. Scaling is accomplished by dividing the non-scaled AIC by sample size.
BIC Pseudo scaled Bayesian Information Criterion (BIC). Candidate models that produce the smallest BIC suggest the optimal number of parameters given the sample size. Scaling is accomplished by dividing the non-scaled BIC by sample size.
convergence Convergence = 0 indicates that the optimization algorithm converged; convergence=1 indicates that the optimization failed to converge.

Author(s)

Niels Waller

References


Examples

## Not run:
NSubjects <- 2000

## generate sample k=1 FMP data
b <- matrix(c(
    #b0  b1  b2  b3  b4  b5  b6  b7  k
    1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1,
    1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1,
    1.282, 1.063, -0.103, 0.003, 0, 0, 0, 0, 1,
), ncol = 8, byrow = TRUE)

0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,
1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,
-0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,
0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,
0.122, 0.601, -0.082, 0.119, 0, 0, 0, 0, 1,
1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,
-0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,
-0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,
0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,
-0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,
0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,
1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,
0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,
0.429, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,
0.089, 0.785, -0.065, 0.018, 0, 0, 0, 0, 1,
-0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,
0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,
0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,
-0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,
0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1)
rownames(ex1.data) <- NULL
ncol(ex1.data) <- 9
byrow(ex1.data) <- TRUE

# generate data using the above item parameters
ex1.data <- genFMPData(NSubj = NSubjects, bParams = b, seed = 345)$data

NItems <- ncol(ex1.data)

# compute (initial) surrogate theta values from
# the normed left singular vector of the centered
# data matrix
thetaInit <- svdNorm(ex1.data)

# Choose model
k <- 1  # order of polynomial = 2k+1

# Initialize matrices to hold output
if(k == 0) {
  startVals <- c(1.5, 1.5)
  bmat <- matrix(0, NItems, 6)
  colnames(bmat) <- c(paste("b", 0:1, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}

if(k == 1) {
  startVals <- c(1.5, 1.5, .10, .10)
  bmat <- matrix(0, NItems, 8)
  colnames(bmat) <- c(paste("b", 0:3, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}

if(k == 2) {
  startVals <- c(1.5, 1.5, .10, .10, .10, .10)
  bmat <- matrix(0, NItems, 10)
  colnames(bmat) <- c(paste("b", 0:5, sep = ""), "FHAT", "AIC", "BIC", "convergence")
}
if(k == 3) {
    startVals <- c(1.5, 1.5, .10, .10, .10, .10, .10, .10)
    bmat <- matrix(0,NItems,12)
    colnames(bmat) <- c(paste("b", 0:7, sep = ""),"FHAT", "AIC", "BIC", "convergence")
}

# estimate item parameters and fit statistics
for(i in 1:NItems){
    out<-FUP(data = ex1.data,thetaInit = thetaInit, item = i, startvals = startVals, k = k)
    Nb <- length(out$b)
    bmat[i,1:Nb] <- out$b
    bmat[i,Nb+1] <- out$FHAT
    bmat[i,Nb+2] <- out$AIC
    bmat[i,Nb+3] <- out$BIC
    bmat[i,Nb+4] <- out$convergence
}

# print results
print(bmat)

## End(Not run)

---

**gen4PMData** Generate item response data for 1, 2, 3, or 4-parameter IRT models

**Description**
Generate item response data for or 1, 2, 3 or 4-parameter IRT Models.

**Usage**

```r
gen4PMData(NSubj = NULL, abcdParams, D = 1.702, seed = NULL,
theta = NULL, thetaMN = 0, thetaVar = 1)
```

**Arguments**

- `NSubj` the desired number of subject response vectors.
- `abcdParams` a p(items)-by-4 matrix of IRT item parameters: a = discrimination, b = difficulty, c = lower asymptote, and d = upper asymptote.
- `D` Scaling constant to place the IRF on the normal ogive or logistic metric. Default = 1.702 (normal ogive metric)
- `seed` Optional seed for the random number generator.
- `theta` Optional vector of latent trait scores. If theta = NULL (the default value) then gen4PMData will simulate theta from a normal distribution.
- `thetaMN` Mean of simulated theta distribution. Default = 0.
- `thetaVar` Variance of simulated theta distribution. Default = 1
Value

- **data**: N(subject)-by-p(items) matrix of item response data.
- **theta**: Latent trait scores.
- **seed**: Value of the random number seed.

Author(s)

Niels Waller

Examples

```r
## Generate simulated 4PM data for 2,000 subjects
# 4PM Item parameters from MMPI-A CYN scale

Params<-matrix(c(1.41, -0.79, .01, .98, #1
1.19, -0.81, .02, .96, #2
0.79, -1.11, .05, .94, #3
0.94, -0.53, .02, .93, #4
0.90, -1.02, .04, .95, #5
1.00, -0.21, .02, .84, #6
1.05, -0.27, .02, .97, #7
0.90, -0.75, .04, .73, #8
0.80, -1.42, .06, .98, #9
0.71, 0.13, .05, .94, #10
1.01, -0.14, .02, .81, #11
0.63, 0.18, .18, .97, #12
0.68, 0.18, .02, .87, #13
0.60, -0.14, ,09, .96, #14
0.85, -0.71, .04, .99, #15
0.83, -0.07, .05, .97, #16
0.86, -0.36, .03, .95, #17
0.66, -0.64, .04, .77, #18
0.60, 0.52, .04, .94, #19
0.90, -0.06, .02, .96, #20
0.62, -0.47, .05, .86, #21
0.57, 0.13, .06, .93, #22
0.77, -0.43, .04, .97),23,4, byrow=TRUE)
data <- gen4PMData(NSubj=2000, abcdParams = Params, D = 1.702,
seed = 123, thetaMN = 0, thetaVar = 1)$data
cat("\nClassical item difficulties for simulated data")
print( round( apply(data,2,mean),2 ) )
```
**genCorr**

*Generate Correlation Matrices with User-Defined Eigenvalues*

**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-structure 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))
```
GenerateBoxData

Generate Thurstone’s Box Data From length, width, and height box measurements

Description

Generate data for Thurstone’s 20 variable and 26 variable Box Study From length, width, and height box measurements.

Usage

GenerateBoxData(XYZ, BoxStudy = 20, Reliability = 0.75,
    ModApproxErrVar = 0.1, SampleSize = NULL, NMinorFac = 50,
    epsTKL = 0.2, Seed = 1, SeedErrorFactors = 2,
    SeedMinorFactors = 3, PRINT = FALSE, LB = FALSE, LBVal = 1,
    Constant = 0)

Arguments

XYZ (Matrix) Length, width, and height measurements for N boxes. The Amazon Box data can be accessed by calling data(AmxBoxes). The Thurstone Box data (20 hypothetical boxes) can be accessed by calling data(Thurstone20Boxes).

BoxStudy (Integer) If BoxStudy = 20 then data will be generated for Thurstone’s classic 20 variable box problem. If BoxStudy = 26 then data will be generated for Thurstone’s 26 variable box problem. Default: BoxStudy = 20.

Reliability (Scalar [0, 1]) The common reliability value for each measured variable. Default: Reliability = .75.

ModApproxErrVar (Scalar [0, 1]) The proportion of reliable variance (for each variable) that is due to all minor common factors. Thus, if x (i.e., error free length) has variance var(x) and ModApproxErrVar = .10, then var(e.ma)/var(x + e.ma) = .10.

SampleSize (Integer) Specifies the number of boxes to be sampled from the population. If SampleSize = NULL then measurements will be generated for the original input box sizes.
GenerateBoxData

NMinorFac (Integer) The number of minor factors to use while generating model approximation error. Default: NMinorFac = 50.

epsTKL (Numeric [0, 1]) A parameter of the Tucker, Koopman, and Linn (1969) algorithm that controls the spread of the influence of the minor factors. Default: epsTKL = .20.

Seed (Integer) Starting seed for box sampling.

SeedErrorFactors (Integer) Starting seed for the error-factor scores.

SeedMinorFactors (Integer) Starting seed for the minor common-factor scores.

PRINT (Logical) If PRINT = TRUE then the computed reliabilities will be printed. Default: PRINT = FALSE. Setting PRINT to TRUE can be useful when LB = TRUE.

LB (lower bound; logical) If LB = TRUE then minimum box measurements will be set to LBVal (inches) if they fall below 0 after adding measurement error. If LB = FALSE then negative attribute values will not be modified. This argument has no effect on data that include model approximation error.

LBVal (Numeric) If LB = TRUE then values in BoxDataE will be bounded from below at LBVal. This can be used to avoid negative or very small box measurements.

Constant (Numeric) Optional value to add to all box measurements. Default: Constant = 0.

Details

This function can be used with the Amazon boxes dataset (data(AmzBoxes)) or with any collection of user-supplied scores on three variables. The Amazon Boxes data were downloaded from the BoxDimensions website: (https://www.boxdimensions.com/). These data contain length (x), width (y), and height (z) measurements for 98 Amazon shipping boxes. In his classical monograph on Multiple Factor Analysis (Thurstone, 1947) Thurstone describes two data sets (one that he created from fictitious data and a second data set that he created from actual box measurements) that were used to illustrate topics in factor analysis. The first (fictitious) data set is known as the Thurstone Box problem (see Kaiser and Horst, 1975). To create his data for the Box problem, Thurstone constructed 20 nonlinear combinations of fictitious length, width, and height measurements. Box20 variables:

1. x^2
2. y^2
3. z^2
4. xy
5. xz
6. yz
7. sqrt(x^2 + y^2)
8. sqrt(x^2 + z^2)
9. sqrt(y^2 + z^2)
10. 2x + 2y
11. \(2x + 2z\)
12. \(2y + 2z\)
13. \(\log(x)\)
14. \(\log(y)\)
15. \(\log(z)\)
16. \(xyz\)
17. \(\sqrt{x^2 + y^2 + z^2}\)
18. \(\exp(x)\)
19. \(\exp(y)\)
20. \(\exp(z)\)

The second Thurstone Box problem contains measurements on the following 26 functions of length, width, and height. **Box26** variables:

1. \(x\)
2. \(y\)
3. \(z\)
4. \(xy\)
5. \(xz\)
6. \(yz\)
7. \(x^2 \cdot y\)
8. \(x \cdot y^2\)
9. \(x^2 \cdot z\)
10. \(x \cdot z^2\)
11. \(y^2 \cdot z\)
12. \(y \cdot z^2\)
13. \(x/y\)
14. \(y/x\)
15. \(x/z\)
16. \(z/x\)
17. \(y/z\)
18. \(z/y\)
19. \(2x + 2y\)
20. \(2x + 2z\)
21. \(2y + 2z\)
22. \(\sqrt{x^2 + y^2}\)
23. \(\sqrt{x^2 + z^2}\)
24. \(\sqrt{y^2 + z^2}\)
25. \(xyz\)
26. \(\sqrt{x^2 + y^2 + z^2}\)

Note that when generating unreliable data (i.e., variables with reliability values less than 1) and/or data with model error, **SampleSize** must be greater than **NMinorFac**.
**Value**

- **XYZ** The length (x), width (y), and height (z) measurements for the sampled boxes. If `SampleSize = NULL` then `XYZ` contains the x, y, z values for the original 98 boxes.

- **BoxData** Error free box measurements.

- **BoxDataE** Box data with added measurement error.

- **BoxDataEME** Box data with added (reliable) model approximation and (unreliable) measurement error.

- **Rel.E** Classical reliabilities for the scores in `BoxDataE`.

- **Rel.EME** Classical reliabilities for the scores in `BoxDataEME`.

- **NMinorFac** Number of minor common factors used to generate `BoxDataEME`.

- **epsTKL** Minor factor spread parameter for the Tucker, Koopman, Linn algorithm.

- **SeedErrorFactors** Starting seed for the error-factor scores.

- **SeedMinorFactors** Starting seed for the minor common-factor scores.

**Author(s)**

Niels G. Waller (nwaller@umn.edu)

**References**


**See Also**

Other Factor Analysis Routines: `BiFAD`, `Box26`, `Ledermann`, `SL1`, `SchmidLeiman`, `faAlign`, `faEKC`, `faMain`, `faScores`, `faSort`, `faStandardize`, `faX`, `fals`, `fapa`, `fareg`, `orderFactors`, `print.faMain`, `promaxQ`, `summary.faMain`

**Examples**

```r
data(AmzBoxes)
BoxList <- GenerateBoxData (XYZ = AmzBoxes[,2:4],
                           BoxStudy = 20,
                           Reliability = .75,
                           ModApproxErrVar = .10,
                           SampleSize = 300,
                           NMinorFac = 50,
                           epsTKL = .20,
                           Seed = 1,
                           SeedErrorFactors = 1,
```
```r
SeedMinorFactors = 2,
PRINT = FALSE,
LB = FALSE,
LBVal = 1,
Constant = 0)

BoxData <- BoxList$BoxData

RBoxes <- cor(BoxData)
fout <- faMain(R = RBoxes,
   numFactors = 3,
   facMethod = "fals",
   rotate = "geominQ",
   rotateControl = list(numberStarts = 100,
   standardize = "CM"))

summary(fout)
```

---

**genFMPData**

*Generate item response data for a filtered monotonic polynomial IRT model*

**Description**

Generate item response data for the filtered polynomial IRT model.

**Usage**

```r
genFMPData(NSubj, bParams, theta = NULL, thetaMN = 0, thetaVar = 1, seed)
```

**Arguments**

- `NSubj` the desired number of subject response vectors.
- `bParams` a `p(items)`-by-9 matrix of polynomial coefficients and model designations. Columns 1 - 8 hold the polynomial coefficients; column 9 holds the value of `k`.
- `theta` A user-supplied vector of latent trait scores. Default `theta = NULL`.
- `thetaMN` If `theta = NULL` genFMPData will simulate random normal deviates from a population with mean `thetaMN` and variance `thetaVar`.
- `thetaVar` If `theta = NULL` genFMPData will simulate random normal deviates from a population with mean `thetaMN` and variance `thetaVar`.
- `seed` initial seed for the random number generator.

**Value**

- `theta` theta values used for data generation
- `data` N(subject)-by-p(items) matrix of item response data.
- `seed` Value of the random number seed.
# Examples

# The following code illustrates data generation for
# an FMP of order 3 (i.e., 2k+1)
# data will be generated for 2000 examinees
NSubjects <- 2000

## Example item parameters, k=1 FMP
b <- matrix(c(
    #b0  b1  b2  b3  b4  b5  b6  b7  k
    1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1,  
    1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1,  
    1.282, 1.063, -0.103, 0.003, 0, 0, 0, 0, 1,  
    0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,  
    1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,  
    -0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,  
    0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,  
    0.122, 0.601, -0.082, 0.119, 0, 0, 0, 0, 1,  
    1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,  
    -0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,  
    -0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,  
    0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,  
    -0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,  
    0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,  
    1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,  
    0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,  
    0.249, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,  
    0.089, 0.785, -0.065, 0.018, 0, 0, 0, 0, 1,  
    -0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,  
    0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,  
    0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,  
    -0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,  
    0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1,  
),
  nrow=23, ncol=9, byrow=TRUE)

data<-genFMPData(NSubj = NSubjects, bParams=b, seed=345)$data
Description

Create a random Phi matrix with maximum factor correlation.

Usage

genPhi(NFac, EigenValPower = 6, MaxAbsPhi = 0.5)

Arguments

NFac Number of factors.
EigenValPower (Scalar > 1) A scalar than controls the positive skewness of the distribution of eigenvalues of Phi.
MaxAbsPhi (Scaler in [0,1]) The maximum off diagonal of Phi (the factor correlation matrix).

Value

A factor correlation matrix. Note that the returned matrix is not guaranteed to be positive definite. However, a PD check is performed in simFA so that simFA always produces a PD Phi matrix.

Author(s)

Niels Waller

Examples

```r
NFac <- 5
par(mfrow=c(2,2))
for(i in 1:4){
  R <- genPhi(NFac,
              EigenValPower = 6,
              MaxAbsPhi = 0.5)

  L <- eigen(R)$values
  plot(1:NFac, L,
       type="b",
       ylab = "Eigenvalues of Phi",
       xlab = "Dimensions",
       ylim=c(0,L[1]+.5))
}
```
Description
Mental abilities data on seventh- and eighth-grade children from the classic Holzinger and Swineford (1939) dataset.

Format
A data frame with 301 observations on the following 15 variables.

<table>
<thead>
<tr>
<th>id</th>
<th>subject identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>sex</td>
<td>gender</td>
</tr>
<tr>
<td>ageyr</td>
<td>age, year part</td>
</tr>
<tr>
<td>agemo</td>
<td>age, month part</td>
</tr>
<tr>
<td>school</td>
<td>school name (Pasteur or Grant-White)</td>
</tr>
<tr>
<td>grade</td>
<td>grade</td>
</tr>
<tr>
<td>x1</td>
<td>Visual perception</td>
</tr>
<tr>
<td>x2</td>
<td>Cubes</td>
</tr>
<tr>
<td>x3</td>
<td>Lozenges</td>
</tr>
<tr>
<td>x4</td>
<td>Paragraph comprehension</td>
</tr>
<tr>
<td>x5</td>
<td>Sentence completion</td>
</tr>
<tr>
<td>x6</td>
<td>Word meaning</td>
</tr>
<tr>
<td>x7</td>
<td>Speeded addition</td>
</tr>
<tr>
<td>x8</td>
<td>Speeded counting of dots</td>
</tr>
<tr>
<td>x9</td>
<td>Speeded discrimination straight and curved capitals</td>
</tr>
</tbody>
</table>

Source
These data were retrieved from the lavaan package. The complete data for all 26 tests are available in the MBESS package.

References

Examples
```r
data(HS9Var)
head(HS9Var)
```
**HW**  
*Six data sets that yield a Heywood case*

**Description**
Six data sets that yield a Heywood case in a 3-factor model.

**Usage**
data(HW)

**Format**
Each data set is a matrix with 150 rows and 12 variables:

- Each data set (HW1, HW2, ... HW6) represents a hypothetical sample of 150 subjects from a population 3-factor model. The population factor loadings are given in HW$popLoadings.

**Examples**
data(HW)

```r
# Compute a principal axis factor analysis
# on the first data set
RHW <- cor(HW$HW1)
fapaOut <- faMain(R = RHW,
numFactors = 3,
facMethod = "fapa",
rotate = "oblimin",
faControl = list(treatHeywood = FALSE))

fapaOut$faFit$Heywood
round(fapaOut$h2, 2)
```

---

**irf**  
*Plot item response functions for polynomial IRT models.*

**Description**
Plot model-implied (and possibly empirical) item response function for polynomial IRT models.

**Usage**
irf(data, bParams, item, plotERF = TRUE, thetaEAP = NULL, minCut = -3, maxCut = 3, NCuts = 9)
Arguments

data: N(subjects)-by-p(items) matrix of 0/1 item response data.
bParams: p(items)-by-9 matrix. The first 8 columns of the matrix should contain the FMP or FUP polynomial coefficients for the p items. The 9th column contains the value of k for each item (where the item specific order of the polynomial is 2k+1).

item: The IRF for item will be plotted.
plotERF: A logical that determines whether to plot discrete values of the empirical response function.
thetaEAP: If plotERF=TRUE, the user must supply previously calculated eap trait estimates to thetaEAP.
minCut, maxCut: If plotERF=TRUE, the program will (attempt to) plot NCuts points of the empirical response function between trait values of minCut and maxCut. Default minCut = -3. Default maxCut = 3.
NCuts: Desired number of bins for the empirical response function.

Author(s)

Niels Waller

Examples

NSubjects <- 2000
NItems <- 15
itmParameters <- matrix(c(
  # b0  b1  b2  b3  b4  b5  b6  b7  k
  -1.05, 1.63, 0.00, 0.00, 0.00, 0, 0, 0, 0, #1
  -1.97, 1.75, 0.00, 0.00, 0.00, 0, 0, 0, 0, #2
  -1.77, 1.82, 0.00, 0.00, 0.00, 0, 0, 0, 0, #3
  -4.76, 2.67, 0.00, 0.00, 0.00, 0, 0, 0, 0, #4
  -2.15, 1.93, 0.00, 0.00, 0.00, 0, 0, 0, 0, #5
  -1.25, 1.17, -0.25, 0.12, 0.00, 0, 0, 0, 0, #6
  1.65, 0.01, 0.02, 0.03, 0.00, 0, 0, 0, 1, #7
  -2.99, 1.64, 0.17, 0.03, 0.00, 0, 0, 0, 1, #8
  -3.22, 2.40, -0.12, 0.10, 0.00, 0, 0, 0, 1, #9
  -0.75, 1.09, -0.39, 0.31, 0.00, 0, 0, 0, 1, #10
  -1.21, 9.07, 1.20,-0.01,-0.01, 0.01, 0, 0, 2, #11
  -1.92, 1.55, -0.17, 0.50,-0.01, 0.01, 0, 0, 2, #12
  -1.76, 1.29, -0.13, 1.60,-0.01, 0.01, 0, 0, 2, #13
  -2.32, 1.40, 0.55, 0.05,-0.01, 0.01, 0, 0, 2, #14
  -1.24, 2.48, -0.65, 0.60,-0.01, 0.01, 0, 0, 2), byrow=TRUE)

ex1.data<-genFMPData(NSubj = NSubjects, bParams = itmParameters, seed = 345)$data
## compute initial theta surrogates
thetaInit <- svdNorm(ex1.data)

## For convenience we assume that the item parameter
## estimates equal their population values. In practice,
## item parameters would be estimated at this step.
itmEstimates <- itmParameters

## calculate eap estimates for mixed models
thetaEAP <- eap(data = ex1.data, bParams = itmEstimates, NQuad = 21,
        priorVar = 2,
        mintheta = -4, maxtheta = 4)

## plot irf and erf for item 1
irf(data = ex1.data, bParams = itmEstimates,
        item = 1,
        plotERF = TRUE,
        thetaEAP)

## plot irf and erf for item 12
irf(data = ex1.data, bParams = itmEstimates,
        item = 12,
        plotERF = TRUE,
        thetaEAP)

---

**itemDescriptives**

Compute basic descriptives for binary-item analysis

**Description**

Compute basic descriptives for binary item analysis

**Usage**

itemDescriptives(X, digits = 3)

**Arguments**

- `X` 
  a matrix of binary (0/1) item responses.

- `digits` 
  number of digits to print.

**Value**

- `alpha` 
  Coefficient alpha for the total scale.

- `means` 
  item means.

- `standard deviations` 
  item standard deviations.
pt. biserial correlations
corrected item-total point biserial correlations.

biserial correlations
corrected item-total point biserial correlations.

corrected.alpha
corrected (leave item out) alpha coefficients.

Author(s)
Niels Waller

Examples

## Example 1: 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

Nsubj <- 2000
L <- matrix(rep(.707,5), nrow = 5, ncol = 1)
Z <- as.matrix(rnorm(Nsubj))
U <- matrix(rnorm(Nsubj * 5),nrow = Nsubj, ncol = 5)
tmp <- sqrt(1 - L^2)
D <- matrix(0, 5, 5)
diag(D) <- tmp
X <- Z %*% t(L) + U%*%D

cat("Correlation of continuous scores")
print(round(cor(X),3))

thresholds <- c(.2,.3,.4,.5,.6)
Binary<-matrix(0,Nsubj,5)
for(i in 1:5){
  Binary[X[,i]< thresholds[i],i]<-1
}

cat("Correlation of Binary scores")
print(round(cor(Binary),3))

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

z <- bigen(data = Binary, n = 5000)
cat("\n\nnames in returned object\n")
print(names(z))

cat("\n\nCorrelation of Simulated binary scores\n")
print(round( cor(z$data), 3))

cat("\nObserved thresholds of simulated data:\n")
cat( apply(z$data, 2, mean) )

itemDescriptives(z$data)

---

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))
```

---

**Ledermann's inequality for factor solution identification**

**Description**

Ledermann's (1937) inequality to determine either (a) how many factor indicators are needed to
uniquely estimate a user-specified number of factors or (b) how many factors can be uniquely esti-
mated from a user-specified number of factor indicators. See the Details section for more informa-
tion.

**Usage**

```r
Ledermann(numFactors = NULL, numVariables = NULL)
```

**Arguments**

- `numFactors` (Numeric) Determine the number of variables needed to uniquely estimate the
  [user-specified] number of factors. Defaults to `numFactors = NULL`.
- `numVariables` (Numeric) Determine the number of factors that can be uniquely estimated from
  the [user-specified] number of variables. Defaults to `numVariables = NULL`.

**Details**

The user will specify either (a) `numFactors` or (b) `numVariables`. When one value is specified,
the obtained estimate for the other may be a non-whole number. If estimating the number of required
variables, the obtained estimate is rounded up (using `ceiling`). If estimating the number of factors,
the obtained estimate is rounded down (using `floor`). For example, if `numFactors = 2`, roughly
4.56 variables are required for an identified solution. However, the function returns an estimate of
5.

For the relevant equations, see Thurstone (1947, p. 293) Equations 10 and 11.

**Value**

- `numFactors` (Numeric) Given the inputs, the number of factors to be estimated from the
  `numVariables` number of factor indicators.
- `numVariables` (Numeric) Given the inputs, the number of variables needed to estimate `numFactors`.

**Author(s)**

Casey Giordano
References


Thurstone, L. L. (1947). Multiple-factor analysis; a development and expansion of The Vectors of Mind.

See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Sli, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

```r
## To estimate 3 factors, how many variables are needed?
Ledermann(numFactors = 3,
          numVariables = NULL)

## Provided 10 variables are collected, how many factors
## can be estimated?
Ledermann(numFactors = NULL,
          numVariables = 10)
```

---

### 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

```r
monte(seed = 123, nvar = 4, nclus = 3, clus.size = c(50, 50, 50),
      eta2 = c(0.619, 0.401, 0.941, 0.929), cor.list = NULL,
      random.cor = FALSE, skew.list = NULL, kurt.list = NULL,
      secor = NULL, compactness = NULL, sortMeans = TRUE)
```

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.
lmn        The cluster indicator means.
f1         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.
eta2       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:
# 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, 0.306, 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, -0.602))

#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,
```r
#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 ###

```r
## 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.).

---

```r
library(lattice)

# Condition 1
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, 

```
```r
monte

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
############################
## Within taxon indicator skew and kurtosis
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"
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")

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 3
############################
## Set within group correlations to .80
cormat <- matrix(.80, 3, 3)
```
# Code Snippet

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

## 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)

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, .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)
```

---

*Monte Carlo Simulation*
cor.list = in.cor.list,
compactness = in.compactness)

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")
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, .0, 1, .5), more = TRUE)

---

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.
**monte1**

- `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.monte1`

**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

data(Harman23.cor)
normalCor(Harman23.cor$cov, Nobs = 305)
**normF**

*Compute the Frobenius norm of a matrix*

**Description**

A function to compute the Frobenius norm of a matrix

**Usage**

```r
normF(X)
```

**Arguments**

- `X` A matrix.

**Value**

The Frobenius norm of X.

**Author(s)**

Niels Waller

**Examples**

```r
data(BadRLG)
out <- smoothLG(R = BadRLG, Penalty = 50000)
cat("Gradient at solution:", out$gr, "\n")
cat("Nearest Correlation Matrix\n")
print( round(out$RLG, 8) )
cat("Frobenius norm of (NPD - PSD) matrix\n")
print(normF(BadRLG - out$RLG ))
```

---

**Omega**

*Compute Omega hierarchical*

**Description**

This function computes McDonald’s Omega hierarchical to determine the proportions of variance (for a given test) associated with the latent factors and with the general factor.

**Usage**

```r
Omega(lambda, genFac = 1, digits = NULL)
```
Arguments

lambda (Matrix) A factor pattern matrix to be analyzed.
genFac (Scalar, Vector) Which column(s) contains the general factor(s). The default value is the first column.
digits (Scalar) The number of digits to round all output to.

Details

• **Omega Hierarchical**: For a reader-friendly description (with some examples), see the Rodriguez et al., (2016) *Psychological Methods* article. Most of the relevant equations and descriptions are found on page 141.

Value

• **omegaTotal**: (Scalar) The total reliability of the latent, common factors for the given test.
• **omegaGeneral**: (Scalar) The proportion of total variance that is accounted for by the general factor(s).

Author(s)

• Casey Giordano (Giord023@umn.edu)
• Niels G. Waller (nwaller@umn.edu)

References


Examples

```r
## Create a bifactor structure
bifactor <- matrix(c(.21, .49, .00, .00,
                    .12, .28, .00, .00,
                    .17, .38, .00, .00,
                    .23, .00, .34, .00,
                    .34, .00, .52, .00,
                    .22, .00, .34, .00,
                    .41, .00, .00, .42,
                    .46, .00, .00, .47,
                    .48, .00, .00, .49),
                   nrow = 9, ncol = 4, byrow = TRUE)

## Compute Omega
Out1 <- Omega(lambda = bifactor)
```
orderFactors

Order factor-loadings matrix by the sum of squared factor loadings

Description

Order the columns of a factor loadings matrix in descending order based on the sum of squared factor loadings.

Usage

orderFactors(Lambda, PhiMat, salient = 0.29, reflect = TRUE)

Arguments

Lambda (Matrix) Factor loadings matrix to be reordered.
PhiMat (Matrix, NULL) Factor correlation matrix to be reordered.
salient (Numeric) Indicators with loadings < salient will be suppressed when computing the factor sum of squares values. Defaults to salient = .29.
reflect (Logical) If true, negatively-keyed factors will be reflected. Defaults to reflect = TRUE.

Value

Returns the sorted factor loading and factor correlation matrices.

• Lambda: (Matrix) The sorted factor loadings matrix.
• Phi: (Matrix) The sorted factor correlation matrix.

See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, print.faMain, promaxQ, summary.faMain

Examples

## Not run:
Loadings <-
matrix(c(.49, .41, .00, .00,
       .73, .45, .00, .00,
       .47, .53, .00, .00,
       .54, .00, .66, .00,
       .60, .00, .38, .00,
       .55, .00, .66, .00,
       .39, .00, .00, .68,
       .71, .00, .00, .56,
       .63, .00, .00, .55),
nrow = 9, ncol = 4, byrow = TRUE)
fungible::orderFactors(Lambda = Loadings, 
PhiMat = NULL)$Lambda

## End(Not run)

---

### 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)
```

---

### print.faMain

#### Print Method for an Object of Class faMain

**Description**

Print Method for an Object of Class FaMain

**Usage**

```r
## S3 method for class 'faMain'
print(x, ..., digits = 2, Set = 1, itemSort = FALSE)
```
Arguments

x  (Object of class faMain) The returned object from a call to faMain.

... Additional arguments affecting the summary produced.

digits  (Integer) Print output with user-specified number of significant digits. Default digits = 2.

Set

• integer (Integer) Summarize the solution from the specified solution set.
• ’UnSpun’ (Character) Summarize the solution from the rotated output that was produced by rotating from the unrotated (i.e., unspun) factor orientation.

itemSort  (Logical) If TRUE, sort the order of the observed variables to produce a "staircase"-like pattern. In bifactor models (i.e., bifactorT and bifactorQ) item sorting is determined by the magnitudes of the group factor loadings. Defaults to itemSort = FALSE.

See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, promaxQ, summary.faMain

promaxQ  Conduct an Oblique Promax Rotation

Description

This function is an extension of the promax function. This function will extract the unrotated factor loadings (with three algorithm options, see faX) if they are not provided. The factor intercorrelations (Phi) are also computed within this function.

Usage

promaxQ(R = NULL, urLoadings = NULL, facMethod = "fals", numFactors = NULL, power = 4, standardize = "Kaiser", epsilon = 1e-04, maxItr = 15000, digits = NULL, faControl = NULL)

Arguments

R  (Matrix) A correlation matrix.

urLoadings  (Matrix) An unrotated factor-structure matrix to be rotated.

facMethod  (Character) The method used for factor extraction (faX). The supported options are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa" for iterated principal axis factoring, "faregLS" for regularized least squares, "faregML" for regularized maximum likelihood, and "pca" for principal components analysis. The default method is "fals".
• "fals": Factors are extracted using the unweighted least squares estimation procedure using the `fals` function.
• "faml": Factors are extracted using the maximum likelihood estimation procedure using the `factanal` function.
• "fapa": Factors are extracted using the iterated principal axis factoring estimation procedure using the `fapa` function.
• "faregLS": Factors are extracted using regularized least squares factor analysis using the `fareg` function.
• "faregML": Factors are extracted using regularized maximum likelihood factor using the `fareg` function.
• "pca": Principal components are extracted.

--

`numFactors` (Scalar) The number of factors to extract if the lambda matrix is not provided.

`power` (Scalar) The power with which to raise factor loadings for minimizing trivial loadings. The default value is 4.

`standardize` (Character) Which standardization routine is applied to the unrotated factor structure. The three options are "none", "Kaiser", and "CM". The default option is "Kaiser" as is recommended by Kaiser and others. See `faStandardize` for more details.
  - "none": Do not rotate the normalized factor structure matrix.
  - "Kaiser": Use a factor structure matrix that has been normed by Kaiser's method (i.e., normalize all rows to have a unit length).
  - "CM": Use a factor structure matrix that has been normed by the Cureton-Mulaik method.

`epsilon` (Scalar) The convergence criterion used for evaluating the varimax rotation. The default value is 1e-4 (i.e., .0001).

`maxItr` (Scalar) The maximum number of iterations allowed for computing the varimax rotation. The default value is 15,000 iterations.

`digits` (Numeric) Rounds the values to the specified number of decimal places. Defaults to digits = NULL (no rounding).

`faControl` (List) A list of optional parameters passed to the factor extraction (`faX`) function.
  - `treatHeywood` (Logical) In `fals`, if `treatHeywood` is true, a penalized least squares function is used to bound the communality estimates below 1.0. Defaults to `treatHeywood` = TRUE.
  - `nStart` (Numeric) The number of starting values to be tried in `faml`. Defaults to nStart = 10.
  - `start` (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to start = NULL.
  - `maxCommunality` (Numeric) In `faml`, set the maximum communality value for the estimated solution. Defaults to maxCommunality = .995.
  - `epsilon` (Numeric) In `fapa`, the numeric threshold designating when the algorithm has converged. Defaults to epsilon = 1e-4.
  - `communality` (Character) The method used to estimate the initial communality values in `fapa`. Defaults to communality = 'SMC'.


- "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
- "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
- "unity": Initial communalities equal 1.0 for all variables.

- **maxItr**: (Numeric) In `fapa`, the maximum number of iterations to reach convergence. Defaults to `maxItr = 15,000`.

### Details

- **Varimax Standardization**: When conducting the varimax rotation, it is recommended to standardize the factor loadings using Kaiser’s normalization (i.e., rescaling the factor indicators [rows] so that the vectors have unit length). The standardization/normalization occurs by pre-multiplying the unrotated factor structure, \( A \), by the inverse of \( H \), where \( H^2 \) is a diagonal matrix with the communality estimates on the diagonal. A varimax rotation is then applied to the normalized, unrotated factor structure. Then, the varimax-rotated factor structure is rescaled to its original metric by pre-multiplying the varimax factor structure by \( H \). For details, see Mulaik (2009).

- **Oblique Procrustes Rotation of the Varimax Solution**: According to Hendrickson & White (1964), an unrestricted (i.e., oblique) Procrustes rotation is applied to the orthogonal varimax solution. Specifically, a target matrix is generated by raising the varimax factor loadings to the user-specified power (typically, power = 4) (must retain the signs of the original factor loadings). This should quickly diminish trivial factor loadings while retaining larger factor loadings. The Procrustes rotation takes the varimax solution and rotates it toward the promax-generated target matrix. For a modern description of this approach, see Mulaik (2009, ch. 12, p. 342-343).

- **Choice of a Power**: Changing the power in which varimax factor loadings are raised will change the target matrix in the oblique Procrustes rotation. After raising factor loadings to some power, there will be a larger discrepancy between high and low loadings than before (e.g., squaring factor loadings of .6 and .7 yields loadings of .36 and .49 and cubing yields loadings of .216 and .343). Furthermore, increasing the power will increase the number of near-zero loadings, resulting in larger factor intercorrelations. Many (cf. Gorsuch, 1983; Hendrickson & White, 1964; Mulaik, 2009) advocate for raising varimax loadings to the fourth power (the default) but some (e.g., Gorsuch) advocate for trying power = 2 and power = 6 to see if there is an improvement in the simple structure without overly inflating factor correlations.

### Value

A list of the following elements are produced:

- **loadings**: (Matrix) The oblique, promax-rotated, factor-pattern matrix.
- **vmaxLoadings**: (Matrix) The orthogonal, varimax-rotated, factor-structure matrix used as the input matrix for the promax rotation.
- **rotMatrix**: (Matrix) The (rescaled) transformation matrix used in an attempt to minimize the Euclidean distance between the varimax loadings and the generated promax target matrix (cf. Hendrickson & White, 1964; Mulaik, 2009, p. 342-343, eqn. 12.44).
**Phi**: (Matrix) The factor correlation matrix associated with the promax solution. Phi is found by taking the inverse of the inner product of the (rescaled) rotation matrix (rotMatrix) with itself (i.e., solve(T^T), where T is the (rescaled) rotation matrix).

**vmaxDiscrepancy**: (Scalar) The value of the minimized varimax discrepancy function. promax does not have a rotational criterion but the varimax rotation does.

**convergence**: (Logical) Whether the varimax rotation converged.

**Table**: (Matrix) The table returned from **GPForth** from the **GPArotation** package.

**rotateControl**: (List) A list containing (a) the power parameter used, (b) whether the varimax rotation used Kaiser normalization, (c) the varimax epsilon convergence criterion, and (d) the maximum number of iterations specified.
  - **power**: The power in which the varimax-rotated factor loadings are raised.
  - **standardize**: Which standardization routine was used.
  - **epsilon**: The convergence criterion set for the varimax rotation.
  - **maxItr**: The maximum number of iterations allowed for reaching convergence in the varimax rotation.

**Author(s)**
- Casey Giordano (Giord023@umn.edu)
- Niels G. Waller (nwaller@umn.edu)

**References**

**See Also**

**Examples**
```r
## Generate an orthogonal factor model
lambda <- matrix(c(.41, .00, .00,
                  .45, .00, .00,
                  .53, .00, .00,
                  .00, .66, .00,
                  .00, .38, .00,
                  .00, .66, .00,
                  .00, .00, .68,
                  .00, .00, .56,
                  .00, .00, .55),
                 nrow = 9, ncol = 3, byrow = TRUE)
```
## Model-implied correlation (covariance) matrix

```
R <- lambda %*% t(lambda)
```

## Unit diagonal elements

```
diag(R) <- 1
```

## Start from just a correlation matrix

```
Out1 <- promaxQ(R,
facMethod = "fals",
numFactors = 3,
power = 4,
standardize = "Kaiser")
```

## Iterate the promaxQ rotation using the rotate function

```
Out2 <- faMain(R,
facMethod = "fals",
numFactors = 3,
rotate = "promaxQ",
rotateControl = list(power = 4,
standardize = "Kaiser"))
```

## Align the factors to have the same orientation

```
Out1 <- faAlign(F1 = Out2,
F2 = Out1)
```

## Show the equivalence of factor solutions from promaxQ and rotate

```
all.equal(Out1, Out2, check.attributes = FALSE)
```

---

### r2d

**Convert Radians to Degrees**

**Description**

Convert radian measure to degrees.

**Usage**

```
r2d(radian)
```

**Arguments**

- `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(Rxx, Rsq, b1, b2, Npoints)

Arguments

Rxx          Predictor correlation matrix.
Rsq          Model coefficient of determination.
b1           First point on ellipsoid. If b1 and b2 are scalars then choose scaled eigenvectors v[b1] and v[b2] as the start and end vectors.
b2           Second point on ellipsoid. If b1 and b2 are scalars then choose scaled eigenvectors v[b1] and v[b2] as the start and end vectors.
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("Average relative performance of unit weights across elliptical arc:",
     round(mean(Rsq.unit)/Rsq,3) )
cat("Average relative performance of r weights across elliptical arc:",
     round(mean(Rsq.r)/Rsq,3) )

plot(seq(0, 90, length = N), Rsq.r, typ = "l",
     ylim = c(0, .20),
     xlim = c(0, 95),
     lwd = 3,
     ylab = expression(R^2),
     xlab = expression(paste("Degrees from ",b[1]," in the direction of ",b[2])),
     cex.lab = 1.25, lab = c(10, 5, 5))
points(seq(0, 90, length = N), Rsq.unit,
       type = "l",
       lty = 2, lwd = 3)
legend(x = 0,y = .12,
       legend = c("r weights", "unit weights"),
       lty = c(1, 2),
```

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
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

```r
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.

Value

A random correlation matrix.
rellipsoid

Author(s)
Niels Waller and Jeff Jones.

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.

References
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,])
```

---

restScore

*Plot an ERF using rest scores*

**Description**

Plot an empirical response function using rest scores.

**Usage**

```r
restScore(data, item, NCuts = 10)
```

**Arguments**

- `data` N(subjects)-by-p(items) matrix of 0/1 item response data.
- `item` Generate a rest score plot for item `item`.
- `NCuts` Divide the rest scores into `NCuts` bins of equal width.
Value

A restscore plot with 95% confidence interval bars for the conditional probability estimates.

item
The item number.

bins
A vector of bin limits and bin sample sizes.

binProb
A vector of bin conditional probabilities.

Author(s)

Niels Waller

Examples

NSubj <- 2000

#generate sample k=1 FMP data
b <- matrix(c(
  #b0 b1 b2 b3 b4 b5 b6 b7 k
  1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1,
  1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1,
  1.282, 1.063, -0.103, 0.003, 0, 0, 0, 0, 1,
  0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,
  1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,
  -0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,
  0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,
  0.122, 0.601, -0.082, 0.119, 0, 0, 0, 0, 1,
  1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,
  -0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,
  -0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,
  0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,
  -0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,
  0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,
  1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,
  0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,
  0.429, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,
  0.009, 0.765, -0.065, 0.018, 0, 0, 0, 0, 1,
  -0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,
  0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,
  0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,
  -0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,
  0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1),
nrow=23, ncol=9, byrow=TRUE)

data<-genFMPData(NSubj = NSubj, bParam = b, seed = 345)$data

## generate a rest score plot for item 12.
## the grey horizontal lines in the plot
## respresent pseudo asymptotes that
## are significantly different from the
## (0,1) boundaries


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

rGivens(eigs, Seed = NULL)

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

## 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
rMAP

Generate Correlation Matrices with Specified Eigenvalues

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

Usage

rMAP(eigenval, eps = 1e-12, maxits = 5000, 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'.

---

```r
# print(out)
#$R
# [1,] 1.000000 1.000000 0.46497370 0.2392817
# [2,] -0.1104098 -0.2451233 0.33564370 -0.7645915
# [3,] -0.2451233 0.3356437 1.00000000 -0.02935466 -0.2024926
# [4,] 0.4649737 -0.4664016 -0.2935466 1.0000000 0.6225880
# [5,] 0.2392817 -0.7645915 -0.02935466 0.62258797 1.0000000
#
#$Frob
# [1] 2.691613
#
#$S0
# [1,] 1.0349665 0.2253775 -0.46827121 -0.10448336 -0.24730565
# [2,] 0.2253775 0.31833805 -0.23208078 0.06591368 -0.14504161
# [3,] -0.4682712 -0.23208078 2.28911499 -0.05430754 0.06964858
# [4,] -0.1044834 0.06591368 0.05430754 0.94884439 -0.14439623
# [5,] -0.2473056 -0.14504161 0.06964858 -0.14439623 0.40873606
#
#$convergence
# [1] TRUE
```
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
```

### rmsd

**Root Mean Squared Deviation of (A - B)**

Description

Calculates the root mean squared deviation of matrices A and B. If these matrices are symmetric (Symmetric = TRUE) then the calculation is based on the upper triangles of each matrix. When the matrices are symmetric, the diagonal of each matrix can be included or excluded from the calculation (IncludeDiag = FALSE)

Usage

```r
rmsd(A, B, Symmetric = TRUE, IncludeDiag = FALSE)
```
Arguments

A  A possibly non square matrix.
B  A matrix of the same dimensions as matrix A.
Symmetric Logical indicating whether A and B are symmetric matrices. (Default: Symmetric = TRUE)
IncludeDiag Logical indicating whether to include the diagonals in the calculation. (Default: IncludeDiag = FALSE).

Value

Returns the root mean squared deviation of (A - B).

Author(s)

Niels Waller

Examples

A <- matrix(rnorm(9), nrow = 3)
B <- matrix(rnorm(9), nrow = 3)
(rmsd(A, B, Symmetric = FALSE, IncludeDiag = TRUE))

RnpdMAP

Generate Random NPD R matrices from a user-supplied population R

Description

Generate a list of Random NPD (pseudo) R matrices with a user-defined fixed minimum eigenvalue from a user-supplied population R using the method of alternating projections.

Usage

RnpdMAP(Rpop, Lp = NULL, NNegEigs = 1, NSmoothPosEigs = 4,
         NSubjects = NULL, NSamples = 0, MaxIts = 15000, PRINT = FALSE,
         Seed = NULL)

Arguments

Rpop  input (PD or PSD) p x p Population correlation matrix.
Lp  desired minimum eigenvalue in the NPD matrices.
NNegEigs  number of eigenvalues < 0 in Rnpd.
NSmoothPosEigs  number of eigenvalues > 0 to smooth: the smallest NSmoothPosEigs > 0 be smoothed toward 0.
NSubjects sample size (required when NSamples > 0) parameter used to generate sample correlation matrices. Default = NULL.

NSamples generate NSamples sample R matrices. If NSamples = 0 the program will attempt to find Rnpd such that ||Rpop - Rnpd||_2 is minimized.

MaxIts maximum number of projection iterations.

PRINT (logical) If TRUE the program will print the iteration history for Lp. Default = NULL.

Seed Optional seed for random number generation.

Value

Rpop population (PD) correlation matrix.

R sample correlation matrix.

Rnpd NPD improper (pseudo) correlation matrix.

Lp desired value of minimum eigenvalue.

minEig observed value of minimum eigenvalue of Rnpd.

convergence 0 = converged; 1 = not converged in MaxIts iterations of the alternating projections algorithm.

feasible logical) TRUE if max(abs(r_ij)) <= 1. If FALSE then one or more values in Rnpd > 1 in absolute value.

Seed saved seed for random number generator.

prbs1 vector probabilities used to generate eigenvalues < 0.

prbs2 vector of probabilities used to smooth the smallest NSmoothPosEigs towards zero.

Author(s)

Niels G. Waller

Examples

library(MASS)

Nvar = 20
Nfac = 4
NSubj = 600
Seed = 123

set.seed(Seed)

## Generate a vector of classical item difficulties
p <- runif(Nvar)

cat("\nClassical Item Difficulties:\n")
print(rbind(1:Nvar,round(p,2)) )

summary(p)

## Convert item difficulties to quantiles
b <- qnorm(p)

## Fnc to compute root mean squared standard deviation
RMSD <- function(A, B){
  sqrt(mean( ( A[lower.tri(A, diag = FALSE)] - B[lower.tri(B, diag = FALSE)] )^2))
}

## Generate vector of eigenvalues with clear factor structure
L <- eigGen(nDimensions = Nvar, 
            nMajorFactors = Nfac, 
            PrcntMajor = .60, 
            threshold = .50)

## Generate a population R matrix with the eigenvalues in L
Rpop <- rGivens(eigs = L)$R

## Generate continuous data that will reproduce Rpop (exactly)
X <- mvrnorm(n = NSubj, mu = rep(0, Nvar), 
             Sigma = Rpop, empirical = TRUE)

if( any(colSums(X) == 0) ){ stop("One or more variables have zero variance. Generate a new data set.") }

## Cut X at thresholds given in b to produce binary data U
U <- matrix(0, nrow(X), ncol(X))
for(j in 1:Nvar){
  U[X[,j] <= b[j],j] <- 1
}

## Compute tetrachoric correlations
Rtet <- tetcor(U, Smooth = FALSE, PRINT = TRUE)$r
# Calculate eigenvalues of tetrachoric R matrix
Ltet <- eigen(Rtet)$values

if(Ltet[Nvar] >= 0) stop("Rtet is P(S)D")

## Simulate NPD R matrix with minimum eigenvalue equal to
# min(Ltet)
out <- RnpdMAP(Rpop, 
               Lp = Ltet[Nvar], 
               NNegEigs = Nvar/5, 
               NSmoothPosEigs = Nvar/5, 
               NSubjects = 150,
NSamples = 1,
MaxIts = 15000,
PRINT = FALSE,
Seed = Seed)

## RLp is a NPD pseudo R matrix with min eigenvalue = min(Ltet)
RLp <- out[[1]]$Rnpd

## Calculate eigenvalues of simulated NPD R matrix (Rnpd)
Lnpd <- eigen(RLp, only.values = TRUE)$values

## Scree plots for observed and simulated NPD R matrices.
ytop <- max(c(L,Lnpd,Ltet))
pointSize = .8
plot(1:Nvar, L, typ = "b", col = "darkgrey", lwd=3,
     lty=1,
     main = "Eigenvalues of Rpop, Tet R, and Sim Tet R:
     Simulated vs Observed npd Tetrachoric R Matrices",
     ylim = c(-1, ytop),
     xlab = "Dimensions",
     ylab = "Eigenvalues",
     cex = pointSize,cex.main = 1.2)
points(1:Nvar, Lnpd, typ="b",
      col = "red", lwd = 3, lty=2, cex=pointSize)
points(1:Nvar, Ltet, typ="b",
      col = "darkgreen", lwd = 3, lty = 3, cex= pointSize)
legend("topright",
      legend = c("eigs Rpop", "eigs Sim Rnpd", "eigs Emp Rnpd"),
      col = c("darkgrey", "red","darkgreen"),
      lty = c(1,2,3),
      lwd = c(4,4,4), cex = 1.5)
abline(h = 0, col = "grey", lty = 2, lwd = 4)
cat("\nRMSD(Rpop, Rtet) = ", round(rmsd(Rpop, Rtet), 3))
cat("\nRMSD(Rpop, RLp) = ", round(rmsd(Rpop, RLp), 3))
Usage

SchmidLeiman(R, numFactors, facMethod = "fals", rotate = "oblimin",
rescaleH2 = 0.98, digits = NULL, faControl = NULL,
rotateControl = NULL)

Arguments

R (Matrix) A correlation matrix.
numFactors (Vector) The number of latent factors at each level of analysis. For example,
c(3, 1) estimates three latent factors in the first-order common factor model and
one latent factor in the second-order common factor model (i.e., 3 group factors
and 1 general factor). This function can orthogonalize up to (and including) a
three-order factor solution.
facMethod (Character) The method used for factor extraction (faX). The supported options
are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa"
for iterated principal axis factoring, "faregLS" for regularized least squares,
"faregML" for regularized maximum likelihood, and "pca" for principal com-
ponents analysis. The default method is "fals".
  • "fals": Factors are extracted using the unweighted least squares estimation
    procedure using the fals function.
  • "faml": Factors are extracted using the maximum likelihood estimation
    procedure using the factanal function.
  • "fapa": Factors are extracted using the iterated principal axis factoring
    estimation procedure using the fapa function.
  • "faregLS": Factors are extracted using regularized least squares factor
    analysis using the fareg function.
  • "faregML": Factors are extracted using regularized maximum likelihood
    factor using the fareg function.
  • "pca": Principal components are extracted.
rotate (Character) Designate which rotation algorithm to apply. See the faMain func-
tion for more details about possible rotations. Defaults to rotate = "oblimin".
rescaleH2 (Numeric) If a Heywood case is detected at any level of the higher-order factor
analyses, rescale the communality value to continue with the matrix algebra.
When a Heywood case occurs, the uniquenesses (i.e., specific-factor variances)
will be negative and the SL orthogonalization of the group factors is no longer
correct.
digits (Numeric) Rounds the values to the specified number of decimal places. De-
faults to digits = NULL (no rounding).
faControl (List) A list of optional parameters passed to the factor extraction (faX) function.
  • treatHeywood: (Logical) In fals, if treatHeywood is true, a penalized
    least squares function is used to bound the communality estimates below
    1.0. Defaults to treatHeywood = TRUE.
  • nStart: (Numeric) The number of starting values to be tried in faml. De-
faults to nStart = 10.
• **start**: (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to start = NULL.

• **maxCommunality**: (Numeric) In `fam1`, set the maximum communality value for the estimated solution. Defaults to maxCommunality = .995.

• **epsilon**: (Numeric) In `fapa`, the numeric threshold designating when the algorithm has converged. Defaults to epsilon = 1e-4.

• **communality**: (Character) The method used to estimate the initial communality values in `fapa`. Defaults to communality = 'SMC'.
  - "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
  - "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
  - "unity": Initial communalities equal 1.0 for all variables.

• **maxItr**: (Numeric) In `fapa`, the maximum number of iterations to reach convergence. Defaults to maxItr = 15,000.

• **rotateControl** (List) A list of control values to pass to the factor rotation algorithms.
  - **numberStarts**: (Numeric) The number of random (orthogonal) starting configurations for the chosen rotation method (e.g., oblimin). The first rotation will always commence from the unrotated factors orientation. Defaults to numberStarts = 10.
  - **gamma**: (Numeric) This is a tuning parameter (between 0 and 1, inclusive) for an oblimin rotation. See the `GPArotation` library’s oblimin documentation for more details. Defaults to gamma = 0 (i.e., a quartimin rotation).
  - **delta**: (Numeric) This is a tuning parameter for the geomin rotation. It adds a small number (default = .01) to the squared factor loadings before computing the geometric means in the discrepancy function.
  - **kappa**: (Numeric) The main parameterization of the Crawford-Ferguson (CF) rotations (i.e., "cfT" and "cfQ" for orthogonal and oblique CF rotation, respectively). Defaults to kappa = 0.
  - **k**: (Numeric) A specific parameter of the simplimax rotation. Defaults to k = the number of observed variables.
  - **standardize**: (Character) The standardization routine used on the unrotated factor structure. The three options are "none", "Kaiser", and "CM". Defaults to standardize = "none".
    - "none": No standardization is applied to the unrotated factor structure.
    - "Kaiser": Use a factor structure matrix that has been normed by Kaiser’s method (i.e., normalize all rows to have a unit length).
    - "CM": Use a factor structure matrix that has been normed by the Cureton-Mulaik method.
  - **epsilon**: (Numeric) The rotational convergence criterion to use. Defaults to epsilon = 1e-5.
  - **power**: (Numeric) Raise factor loadings the the n-th power in the `promaxQ` rotation. Defaults to power = 4.
  - **maxItr**: (Numeric) The maximum number of iterations for the rotation algorithm. Defaults to maxItr = 15000.
Value

- **L1**: (Matrix) The first-order (oblique) factor pattern matrix.
- **L2**: (Matrix) The second-order (oblique) factor pattern matrix.
- **L3**: (Matrix, NULL) The third-order (oblique) factor pattern matrix (if applicable).
- **Phi1**: (Matrix) The first-order factor correlation matrix.
- **Phi2**: (Matrix) The second-order factor correlation matrix.
- **Phi3**: (Matrix, NULL) The third-order factor pattern matrix (if applicable).
- **Usq1**: (Matrix) The first-order factor uniquenesses (variances).
- **Usq2**: (Matrix) The second-order factor uniquenesses (variances).
- **Usq3**: (Matrix, NULL) The third-order factor uniquenesses (variances) (if applicable).
- **B**: (Matrix) The resulting Schmid-Leiman transformation.
- **rotateControl**: (List) A list of the control parameters passed to the `faMain` function.
- **faControl**: (List) A list of optional parameters passed to the factor extraction (`faX`) function.

Author(s)

- Casey Giordano (Giord023@umn.edu)
- Niels G. Waller (nwaller@umn.edu)

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain

Examples

```r
## Dataset used in Schmid & Leiman (1957) rounded to 2 decimal places
SLdata <- matrix(c(1.0, .72, .31, .27, .10, .05, .13, .04, .29, .16, .06, .08, .72, 1.0, .35, .30, .11, .06, .15, .04, .33, .18, .07, .08, .31, .35, 1.0, .42, .08, .04, .10, .03, .22, .12, .05, .06, .27, .30, .42, 1.0, .06, .03, .08, .02, .19, .11, .04, .05, .10, .11, .08, .06, 1.0, .32, .13, .04, .11, .06, .02, .03, .05, .06, .04, .03, .32, 1.0, .07, .02, .05, .03, .01, .01,
```
seBeta

Standard Errors and CIs for Standardized Regression Coefficients

Description

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

Usage

seBeta(X = NULL, y = NULL, 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.

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')
```
seBetaCor

# 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

seBetaCor

Standard Errors and CIs for Standardized Regression Coefficients from Correlations

Description

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

Usage

seBetaCor(R, rxy, Nobs, alpha = 0.05, digits = 3, covmat = "normal")

Arguments

R
A p x p predictor correlation matrix.
rxy
A p x 1 vector of predictor-criterion correlations
Nobs
Number of observations.
alpha
Desired Type 1 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-1 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)

# 95% CIs for Standardized Regression Coefficients:
#
#  lbound estimate ubound
# beta_1 0.107 0.263 0.419
# beta_2 0.231 0.391 0.552
# beta_3 0.337 0.495 0.653
```

seBetaFixed

*Covariance Matrix and Standard Errors for Standardized Regression Coefficients for Fixed Predictors*

Description

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

Usage

```r
seBetaFixed(X = NULL, y = NULL, 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.
seBetaFixed

$\text{cov.} \; \text{xy}$  Vector of covariances or correlations between predictors and criterion.
$\text{var.} \; \text{y}$  Criterion variance.
$\text{var.} \; \text{error}$  Optional argument to supply the error variance: $\text{var}(y - \hat{y})$.
$\text{Nobs}$  Number of observations.

**Value**

$\text{cov.Beta}$  Normal theory covariance matrix of standardized regression coefficients for fixed predictors.
$\text{se.Beta}$  Standard errors for standardized regression coefficients for fixed predictors.

**Author(s)**

Jeff Jones and Niels Waller

**References**


**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)

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

## you can also supply covariances instead of raw data
```
Generate Factor Analysis Models and Data Sets for Simulation Studies

Description

A function to simulate factor loadings matrices and Monte Carlo data sets for common factor models and bifactor models.

Usage

\[ \text{simFA} \left( \text{Model} = \text{list}, \text{Loadings} = \text{list}, \text{CrossLoadings} = \text{list}, \right. \]
\[ \left. \text{Phi} = \text{list}, \text{ModelError} = \text{list}, \text{Bifactor} = \text{list}, \right. \]
\[ \left. \text{MonteCarlo} = \text{list}, \text{FactorScores} = \text{list}, \text{Missing} = \text{list}, \right. \]
\[ \left. \text{Control} = \text{list}, \text{Seed} = \text{NULL} \right) \]

Arguments

- **Model** (list)
  - `NFac` (scalar) Number of common or group factors; defaults to `NFac = 3`.
  - `NItemPerFac` (scalar) All factors have the same number of primary loadings.
  - `NItemPerFac` (vector) A vector of length `NFac` specifying the number of primary loadings for each factor; defaults to `NItemPerFac = 3`.
  - `Model` (character) "orthogonal" or "oblique"; defaults to `Model = "orthogonal"`.
- **Loadings** (list)
  - `FacPattern` (NULL or matrix).
    - `FacPattern = M` where `M` is a user-defined factor pattern matrix.
    - `FacPattern = NULL`; `simFA` will generate a factor pattern based on the arguments specified under other keywords (e.g., `Model`, `CrossLoadings`, etc.); defaults to `FacPattern = NULL`.

\begin{verbatim}
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
\end{verbatim}
• FacLoadDist (character) Specifies the sampling distribution for the common factor loadings. Possible values are "runif", "rnorm", "sequential", and "fixed"; defaults to FacLoadDist = "runif".

• FacLoadRange (vector of length NFac, 2, or 1); defaults to FacLoadRange = c(.3,.7).
  - If FacLoadDist = "runif" the vector defines the bounds of the uniform distribution;
  - If FacLoadDist = "rnorm" the vector defines the mean and standard deviation of the normal distribution from which loadings are sampled.
  - If FacLoadDist = "sequential" the vector specifies the lower and upper bound of the loadings sequence.
  - If FacLoadDist = "fixed" and FacLoadRange is a vector of length 1 then all common loadings will equal the constant specified in FacLoadRange. If FacLoadDist = "fixed" and FacLoadRange is a vector of length NFac then each factor will have fixed loadings as specified by the associated element in FacLoadRange.

• h2 (vector) An optional vector of communalities used to constrain the population communalities to user-defined values; defaults to h2 = NULL.

CrossLoadings (list)

• ProbCrossLoad (scalar) A value in the (0,1) interval that determines the probability that a cross loading will be present in elements of the loadings matrix that do not have salient (primary) factor loadings. If set to ProbCrossLoad = 1, a single cross loading will be added to each factor; defaults to ProbCrossLoad = 0.

• CrossLoadRange (vector of length 2) Controls size of the crossloadings; defaults to CrossLoadRange = c(.20,.25).

• CrossLoadPositions (matrix) Specifies the row and column positions of (optional) cross-loadings; defaults to CrossLoadPositions = NULL.

• CrossLoadValues (vector) If CrossLoadPositions is specified then CrossLoadValues is a vector of user-supplied cross-loadings; defaults to CrossLoadValues = NULL.

• CrudFactor (scalar) Controls the size of tertiary factor loadings. If CrudFactor != 0 then elements of the loadings matrix with neither primary nor secondary (i.e., cross) loadings will be sampled from a [-CrudFactor, CrudFactor] uniform distribution; defaults to CrudFactor = 0.

Phi (list)

• MaxAbsPhi (scalar) Upper (absolute) bound on factor correlations; defaults to MaxAbsPhi = .5.

• EigenValPower (scalar) Controls the skewness of the eigenvalues of Phi. Larger values of EigenValPower result in a Phi spectrum that is more right-skewed (and thus closer to a unidimensional model); defaults to EigenValPower = 2.

• PhiType (character); defaults to PhiType = "free".
  - If PhiType = "free" factor correlations will be randomly generated under the constraints of MaxAbsPhi and EigenValPower.
- If PhiType = "fixed" all factor correlations will equal the value specified in MaxAbsPhi. A fatal error will be produced if Phi is not positive semidefinite.
- If PhiType = "user" the factor correlations are defined by the matrix specified in UserPhi (see below).

  • UserPhi (matrix) A positive semidefinite (PSD) matrix of user-defined factor correlations; defaults to UserPhi = NULL.

ModelError (list)

  • ModelError (logical) If ModelError = TRUE model error will be introduced into the factor pattern via the method described by Tucker, Koopman, and Linn (TKL, 1969); defaults to ModelError = FALSE.
  • NMinorFac (scalar) Number of minor factors in the TKL model; defaults to NMinorFac = 150.
  • ModelErrorType (character) If ModelErrorType = "U" then ModelErrorVar is the proportion of uniqueness variance that is due to model error. If ModelErrorType = "V" then ModelErrorVar is the proportion of total variance that is due to model error; defaults to ModelErrorType = "U".
  • ModelErrorVar (scalar [0,1]) The proportion of uniqueness (U) or total (V) variance that is due to model error; defaults to ModelErrorVar = .10.
  • epsTKL (scalar [0,1]) Controls the size of the factor loadings in successive minor factors; defaults to epsTKL = .20.
  • RSpecific (matrix) Optional correlation matrix for specific factors; defaults to RSpecific = NULL.

Bifactor (list)

  • Bifactor (logical) If Bifactor = TRUE parameters for the bifactor model will be generated; defaults to Bifactor = FALSE.
  • Hierarchical (logical) If Hierarchical = TRUE then a hierarchical Schmid Leiman (1957) bifactor model will be generated; defaults to Hierarchical = FALSE.
  • F1FactorDist (character) Specifies the sampling distribution for the general factor loadings. Possible values are "runif", "rnorm", "sequential", and "fixed"; defaults to F1FactorDist = "sequential".
  • F1FactorRange (vector of length 1 or 2) Controls the sizes of the general factor loadings in nonhierarchical bifactor models; defaults to F1FactorRange = c(.4,.7).
    - If F1FactorDist = "runif", the vector of length 2 defines the bounds of the uniform distribution, c(lower, upper).
    - If F1FactorDist = "rnorm", the vector defines the mean and standard deviation of the normal distribution from which loadings are sampled, c(MN, SD).
    - If F1FactorDist = "sequential", the vector specifies the lower and upper bound of the loadings sequence, c(lower, upper).

MonteCarlo (list)

  • NSamples (integer) Defines number of Monte Carlo Samples; defaults to NSamples = 0.
• SampleSize (integer) Sample size for each Monte Carlo sample; defaults to SampleSize = 250.

• Raw (logical) If Raw = TRUE, simulated data sets will contain raw data. If Raw = FALSE, simulated data sets will contain correlation matrices; defaults to Raw = FALSE.

• Thresholds (list) List elements contain thresholds for each item. Thresholds are required when generating Likert variables.

FactorScores (list)

• FS (logical) If FS = TRUE (true) factor scores will be simulated; defaults to FS = FALSE.

• CFSeed (integer) Optional starting seed for the common factor scores; defaults to CFSeed = NULL in which case a random seed is used.

• SFSeed (integer) Optional starting seed for the specific factor scores; defaults to SFSeed = NULL in which case a random seed is used.

• EFSeed (integer) Optional starting seed for the error factor scores; defaults to EFSeed = NULL in which case a random seed is used. Note that CFSeed, SFSeed, and EFSeed must be different numbers (a fatal error is produced when two or more seeds are specified as equal).

• VarRel (vector) A vector of manifest variable reliabilities. The specific factor variance for variable $i$ will equal $VarRel[i] - h^2[i]$ (the manifest variable reliability minus its commonality). By default, $VarRel = h^2$ (resulting in uniformly zero specific factor variances).

• Population (logical) If Population = TRUE, factor scores will fit the correlational constraints of the factor model exactly (e.g., the common factors will be orthogonal to the unique factors); defaults to Population = FALSE.

• NFacScores (scalar) Sample size for the factor scores; defaults to NFacScores = 250.

• Thresholds (list) A list of quantiles used to polychotomize the observed data that will be generated from the factor scores.

Missing (list)

• Missing (logical) If Missing = TRUE all data sets will contain missing values; defaults to Missing = FALSE.

• Mechanism (character) Specifies the missing data mechanism. Currently, the program only supports missing completely at random (MCAR): Missing = "MCAR".

• MSProb (scalar or vector of length NVar) Specifies the probability of missingness for each variable; defaults to MSprob = 0.

Control (list)

• Maxh2 (scalar) Rows of the loadings matrix will be rescaled to have a maximum communality of Maxh2; defaults to Maxh2 = .98. itemReflect (logical) If Reflect = TRUE loadings on the common factors will be randomly reflected; defaults to Reflect = FALSE.

Seed (integer) Starting seed for the random number generator; defaults to Seed = NULL. When no seed is specified by the user, the program will generate a random seed.
Details

`sima` was specifically designed to simplify the process of running Monte Carlo studies of factor analysis models. Thus, `sima` can save all relevant output for a user-specified model. Saved output can be accessed by calling one or more of the following object names.

Value

- `loadings` A common factor or bifactor loadings matrix.
- `Phi` A factor correlation matrix.
- `urloadings` The unrotated loadings matrix.
- `h2` A vector of item commonalities.
- `h2PopME` A vector item commonalities that may include model approximation error.
- `Rpop` The model-implied population correlation matrix.
- `RpopME` The model-implied population correlation matrix with model error.
- `CovMatrices` A list containing:
  - `CovMajor` The model implied covariances from the major factors.
  - `CovMinor` The model implied covariances from the minor factors.
  - `CovUnique` The model implied variances from the uniqueness factors.
- Bifactor A list containing:
  - `loadingsHier` Factor loadings of the 1st order solution of a hierarchical bifactor model.
  - `PhiHier` Factor correlations of the 1st order solution of a hierarchical bifactor model.
- `Scores` A list containing:
  - `FactorScores` Factor scores for the common and uniqueness factors.
  - `FacInd` Factor indeterminacy indices for the error free population model.
  - `FacIndME` Factor score indeterminacy indices for the population model with model error.
  - `ObservedScores` A matrix of model implied ObservedScores. If `Thresholds` were supplied under Keyword `FactorScores`, `ObservedScores` will be transformed into Likert scores.
- `Monte` A list containing output from the Monte Carlo simulations if generated.
- `IRT` Factor loadings expressed in the normal ogive IRT metric. If `Thresholds` were given then IRT difficulty values will also be returned.
- `Seed` The initial seed for the random number generator.
- `call` A copy of the function call.
- `cn` A list of all active and nonactive function arguments.

Author(s)

Niels G. Waller

References


Examples

# Ex 1. Three Factor Simple Structure Model with Crossloadings and
# Ideal Nonsalient Loadings
out <- simFA(Seed = 1)
print( round( out$loadings, 2 ) )

# Ex 2. Non Hierarchical bifactor model 3 group factors
# with constant loadings on the general factor
out <- simFA(Bifactor = list(Bifactor = TRUE,
Hierarchical = FALSE,
F1FactorRange = c(.4, .4),
F1FactorDist = "runif"),
Seed = 1)
print( round( out$loadings, 2 ) )

skew

Calculate Univariate Skewness for a Vector or Matrix

Description

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

Usage

skew(x)

Arguments

x

Either a vector or matrix of numeric values.

Value

Skewness for each column in x.

Author(s)

Niels Waller

References


See Also

kurt
Examples

```r
x <- matrix(rnorm(1000), 100, 10)
skew(x)
```

---

**SLi**

Conduct a Schmid-Leiman Iterated (SLi) Target Rotation

**Description**

Compute an iterated Schmid-Leiman target rotation (SLi). This algorithm applies Browne’s partially-specified Procrustes target rotation to obtain a full-rank bifactor solution from a rank-deficient (Direct) Schmid-Leiman procedure. Note that the target matrix is automatically generated based on the salient argument. Note also that the algorithm will converge when the partially-specified target pattern in the n-th iteration is equivalent to the partially-specified target pattern in the (n-1)th iteration.

**Usage**

```r
SLi(R, SL = NULL, rotate = "geominQ", numFactors = NULL, 
facMethod = "fals", salient = 0.2, urLoadings = NULL, 
freelyEstG = TRUE, gFac = 1, maxSLiItr = 20, digits = NULL, 
rotateControl = NULL, faControl = NULL)
```

**Arguments**

**R**

(Matrix) A correlation matrix

**SL**

(Matrix, NULL) A (rank-deficient) Schmid-Leiman (SL) bifactor solution (e.g., from a Schmid-Leiman or Direct Schmid-Leiman rotation). If NULL, the function will estimate the SL solution using the `SchmidLeiman` function.

**rotate**

(Character) Designate which rotation algorithm to apply. See the `faMain` function for more details about possible rotations. A geomin rotation is the default.

**numFactors**

(Vector) The number of latent factors at each level of analysis. For example, c(3, 1) estimates three latent factors in the first-order common factor model and one latent factor in the second-order common factor model (i.e., 3 group factors and 1 general factor).

**facMethod**

(Character) The method used for factor extraction (faX). The supported options are "fals" for unweighted least squares, "faml" for maximum likelihood, "fapa" for iterated principal axis factoring, "faregLS" for regularized least squares, "faregML" for regularized maximum likelihood, and "pca" for principal components analysis. The default method is "fals".

- "fals": Factors are extracted using the unweighted least squares estimation procedure using the `fals` function.
- "faml": Factors are extracted using the maximum likelihood estimation procedure using the `factanal` function.
• "fapa": Factors are extracted using the iterated principal axis factoring estimation procedure using the `fapa` function.
• "faregLS": Factors are extracted using regularized least squares factor analysis using the `fareg` function.
• "faregML": Factors are extracted using regularized maximum likelihood factor using the `fareg` function.
• "pca": Principal components are extracted.

```
slifapa

SLi

fapa

SLi

fapa

SLi

fapa

SLi

fapa
```

salient

(Numeric) A threshold parameter used to dichotomize factor loadings to create the target matrix. The default value is .20 (in absolute value) which is based on the Abad et al., 2017 application of this method.

```
salient

SLi

salient

SLi

salient

SLi
```

urLoadings

(Matrix, NULL) A full-rank matrix of unrotated factor loadings to be rotated using the (automatically generated) target matrix. If specified as NULL, a full-rank matrix of factor loadings will be extracted using the `faX` function. An unweighted least squares ("fals") procedure is the default.

```
urLoadings

SLi

urLoadings

SLi

urLoadings

SLi
```

freelyEstG

(Logical) Specify whether the general factor loadings are freely estimated (in the partially-specified target matrix). If set to FALSE, only general factor loadings above the salient threshold will be estimated in the partially-specified target rotation.

```
freelyEstG

SLi

freelyEstG

SLi

freelyEstG
```

gFac

(Numeric, Vector) The position of the general factor(s) to be estimated. Solutions with multiple general factors may be estimated. Must either (a) freely estimate all loadings on the general factors or (b) only freely estimate general factor loadings that are above the salient threshold. The default column position is 1.

```
gFac

SLi

gFac

SLi

gFac
```

maxSLiIttr

(Numeric) The maximum number of iterations for the SLi procedure. Typically, 10 iterations is usually sufficient to converge (cf. Abad et al., 2017). The default is 20 iterations.

```
maxSLiIttr

SLi

maxSLiIttr

SLi

maxSLiIttr
```

digits

(Numeric) The number of digits to round all output to. The default is no rounding.

```
digits

SLi

digits

SLi

digits
```

rotateControl

(List) A list of control values to pass to the factor rotation algorithms.

```
rotateControl

SLi

rotateControl

SLi

rotateControl
```

- `numberStarts`: (Numeric) The number of random (orthogonal) starting configurations for the chosen rotation method (e.g., oblimin). The first rotation will always commence from the unrotated factors orientation. Defaults to `numberStarts = 10`.
- `gamma`: (Numeric) This is a tuning parameter (between 0 and 1, inclusive) for an oblimin rotation. See the GPArotation library’s oblimin documentation for more details. Defaults to `gamma = 0` (i.e., a quartimin rotation).
- `delta`: (Numeric) This is a tuning parameter for the geomin rotation. It adds a small number (default = .01) to the squared factor loadings before computing the geometric means in the discrepancy function.
- `kappa`: (Numeric) The main parameterization of the Crawford-Ferguson (CF) rotations (i.e., "cfT" and "cfQ" for orthogonal and oblique CF rotation, respectively). Defaults to `kappa = 0`.
- `k`: (Numeric) A specific parameter of the simplimax rotation. Defaults to `k = the number of observed variables`. 

```
rotateControl

SLi

rotateControl

SLi

rotateControl
```
• **standardize**: (Character) The standardization routine used on the unrotated factor structure. The three options are "none", "Kaiser", and "CM". Defaults to standardize = "none".
  - "none": No standardization is applied to the unrotated factor structure.
  - "Kaiser": Use a factor structure matrix that has been normed by Kaiser’s method (i.e., normalize all rows to have a unit length).
  - "CM": Use a factor structure matrix that has been normed by the Cureton-Mulaik method.

• **epsilon**: (Numeric) The rotational convergence criterion to use. Defaults to epsilon = 1e-5.

• **power**: (Numeric) Raise factor loadings the n-th power in the promaxQ rotation. Defaults to power = 4.

• **maxItr**: (Numeric) The maximum number of iterations for the rotation algorithm. Defaults to maxItr = 15000.

**faControl**
(List) A list of optional parameters passed to the factor extraction (faX) function.

• **treatHeywood**: (Logical) In fa1s, if treatHeywood is true, a penalized least squares function is used to bound the communality estimates below 1.0. Defaults to treatHeywood = TRUE.

• **nStart**: (Numeric) The number of starting values to be tried in fam1. Defaults to nStart = 10.

• **start**: (Matrix) NULL or a matrix of starting values, each column giving an initial set of uniquenesses. Defaults to start = NULL.

• **maxCommunality**: (Numeric) In fam1, set the maximum communality value for the estimated solution. Defaults to maxCommunality = .995.

• **epsilon**: (Numeric) In fapa, the numeric threshold designating when the algorithm has converged. Defaults to epsilon = 1e-4.

• **communality**: (Character) The method used to estimate the initial communality values in fapa. Defaults to communality = 'SMC'.
  - "SMC": Initial communalities are estimated by taking the squared multiple correlations of each indicator after regressing the indicator on the remaining variables.
  - "maxr": Initial communalities equal the largest (absolute value) correlation in each column of the correlation matrix.
  - "unity": Initial communalities equal 1.0 for all variables.

• **maxItr**: (Numeric) In fapa, the maximum number of iterations to reach convergence. Defaults to maxItr = 15,000.

**Value**

This function iterates the Schmid-Leiman target rotation and returns several relevant output.

• **loadings**: (Matrix) The bifactor solution obtain from the SLi procedure.

• **iterations**: (Numeric) The number of iterations required for convergence

• **rotateControl**: (List) A list of the control parameters passed to the faMain function.

• **faControl**: (List) A list of optional parameters passed to the factor extraction (faX) function.
Author(s)

• Casey Giordano (Giord023@umn.edu)
• Niels G. Waller (nwaller@umn.edu)

References


See Also

Other Factor Analysis Routines: *BiFAD, Box26, GenerateBoxData, Ledermann, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ, summary.faMain*

Examples

```r
## Generate a bifactor model
bifactor <- matrix(c(.35, .61, .00, .00,
                     .35, .61, .00, .00,
                     .35, .61, .00, .00,
                     .35, .00, .61, .00,
                     .35, .00, .61, .00,
                     .35, .00, .61, .00,
                     .35, .00, .00, .61,
                     .35, .00, .00, .61,
                     .35, .00, .00, .61),
                    nrow = 9, ncol = 4, byrow = TRUE)

## Model-implied correlation (covariance) matrix
R <- bifactor %*% t(bifactor)

## Unit diagonal elements
diag(R) <- 1

Out1 <- SLi(R = R,
            numFactors = c(3, 1),
            digits = 2)
```

```
Smooth a Non positive definite (NPD) correlation matrix to PD using the Alternating Projection Algorithm with Dykstra's correction via Theory described in Higham 2002.

Usage

smoothAPA(R, delta = 1e-06, fixR = NULL, Wghts = NULL, maxTries = 1000)

Arguments

- **delta**: Desired value of the smallest eigenvalue of smoothed matrix, RAPA. (Default = 1e-06).
- **fixR**: User-supplied integer list that instructs the program to constrain elements in RAPA to equal corresponding elements in RAPA. For example if fixR = c(1,2) then smoothed matrix, RAPA[1:2,1:2] = R[1:2,1:2]. Default (fixR = NULL).
- **maxTries**: Maximum number of iterations in the alternating projections algorithm. Default (maxTries = 1000).

Value

- **RAPA**: A smoothed matrix.
- **delta**: User-supplied delta value.
- **Wghts**: User-supplied weight vector.
- **fixR**: User-supplied integer list that instructs the program to constrain elements in RAPA to equal corresponding elements in R.
- **convergence**: A value of 0 indicates that the algorithm located a feasible solution. A value of 1 indicates that no feasible solution was located within maxTries.

Author(s)

- Niels Waller
Examples

data(BadRKtB)

#########################################################################
## Replicate analyses in Table 2 of Knol and ten Berge (1989).
#########################################################################

## n1 = 0, 1
out<-smoothAPA(R = BadRKtB, delta = .0, fixR = NULL, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
eigen(S)$val

## n1 = 2
out<-smoothAPA(R = BadRKtB, fixR =c(1,2), delta=.0, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
eigen(S)$val

## n1 = 4
out<-smoothAPA(R = BadRKtB, fixR = 1:4, delta=.0, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
eigen(S)$val

## n1 = 5
out<-smoothAPA(R = BadRKtB, fixR = 1:5, delta=0, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
eigen(S)$val

#########################################################################
## Replicate analyses in Table 3 of Knol and ten Berge (1989).
#########################################################################

## n1 = 0, 1
out<-smoothAPA(R = BadRKtB, delta = .05, fixR = NULL, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
eigen(S)$val

## n1 = 2
out<-smoothAPA(R = BadRKtB, fixR =c(1,2), delta=.05, Wghts = NULL, maxTries=1e06)
S <- out$RAPA
round(S - BadRKtB,3)
normF(S - BadRKtB)
## smoothBY

Smooth an NPD R matrix to PD using the Bentler Yuan 2011 method

### Description
Smooth a NPD correlation matrix to PD using the Bentler and Yuan method.

### Usage

```r
smoothBY(R, const = 0.98, eps = 0.001)
```

### Arguments

- **R**: Indefinite Matrix.
- **const**: A user-defined parameter that is defined as k in Bentler and Yuan (2011). If 0 < const < 1, then const is treated as a fixed value. If const = 1 then the program will attempt to find the highest value of const such that R is positive (semi) definite.
- **eps**: If const = 1 then the program will iteratively reduce const by eps until either (a) the program converges or (b) const <= 0.
### Value

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBY</td>
<td>smoothed correlation matrix.</td>
</tr>
<tr>
<td>constant</td>
<td>The final value of const.</td>
</tr>
<tr>
<td>convergence</td>
<td>(Logical) a value of TRUE indicates that the function converged.</td>
</tr>
<tr>
<td>outStatus</td>
<td>Convergence state for Rsdp::csdp function.</td>
</tr>
</tbody>
</table>

0:
Success. Problem solved to full accuracy

1:
Success. Problem is primal infeasible

2:
Success. Problem is dual infeasible

3:
Partial Success. Solution found but full accuracy was not achieved

4:
Failure. Maximum number of iterations reached

5:
Failure. Stuck at edge of primal feasibility

6:
Failure. Stuck at edge of dual infeasibility

7:
Failure. Lack of progress

8:
Failure. X or Z (or Newton system O) is singular

9:
Failure. Detected NaN or Inf values

**g1b**
Greatest lower bound reliability estimates.
smoothKB

Description

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

Usage

smoothKB(R, eps = 1e+08 * .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

eps
Default value (eps = 1E-03) or user-supplied value of eps.

Author(s)

Code modified from that reported in Debelak, R. & Tran, U. S. (2011).

References


Examples

data(BadRBY)

out<-smoothBY(R = BadRBY, const = .98)
cat("Smoothed Correlation Matrix\n")
print( round(out$RBY,8) )
cat("Eigenvalues of smoothed matrix\n")
print( eigen(out$RBY)$val )
smoothLG

Value

RKB A Smoothed (positive definite) correlation matrix.
eps Small positive number to control the size of the non-scaled smallest eigenvalue of the smoothed R matrix.

Author(s)

Niels Waller

References


Examples

data(BadRLG)

## RKB = smoothed R
RKB<-smoothLG(R=BadRLG, eps = 1E8 * .Machine$double.eps)$RKB
print(eigen(RKB)$values)

smoothLG

Smooth NPD to Nearest PSD or PD Matrix

Description

Smoothing an indefinite matrix to a PSD matrix via theory described by Lurie and Goldberg

Usage

smoothLG(R, start.val = NULL, Wghts = NULL, PD = FALSE,
Penalty = 50000, eps = 1e-07)

Arguments

R Indefinite Matrix.
start.val Optional vector of start values for Cholesky factor of S.
Wghts An optional matrix of weights such that the objective function minimizes wij(rij - sij)^2, where wij is Wghts[i,j].
PD Logical (default = FALSE). If PD = TRUE then the objective function will smooth the least squares solution to insure Positive Definitness.
Penalty A scalar weight to scale the Lagrangian multiplier. Default = 50000.
eps A small value to add to zero eigenvalues if smoothed matrix must be PD. Default = 1e-07.
**Value**

- **RLG**: Lurie Goldberg smoothed matrix.
- **RKB**: Knol and Berger smoothed matrix.
- **convergence**: 0 = converged solution, 1 = convergence failure.
- **start.val**: Vector of start.values.
- **gr**: Analytic gradient at solution.
- **Penalty**: Scalar used to scale the Lagrange multiplier.
- **PD**: User-supplied value of PD.
- **Wghts**: Weights used to scale the squared euclidean distances.
- **eps**: Value added to zero eigenvalue to produce PD matrix.

**Author(s)**

Niels Waller

**Examples**

```r
data(BadRLG)
out<-smoothLG(R = BadRLG, Penalty = 50000)
cat("\nGradient at solution:", out$gr,\n")
cat("\nNearest Correlation Matrix\n")
print( round(out$RLG,8) )

################################
## Rousseeuw Molenbergh example
data(BadRRM)
out <- smoothLG(R = BadRRM, PD=TRUE)
cat("\nGradient at solution:", out$gr,\n")
cat("\nNearest Correlation Matrix\n")
print( round(out$RLG,8) )

## Weights for the weighted solution
W <- matrix(c(1, 1, .5,
             1, 1, 1,
             .5, 1, 1), nrow = 3, ncol = 3)
tmp <- smoothLG(R = BadRRM, PD = TRUE, eps=.001)
cat("\nGradient at solution:", out$gr,\n")
cat("\nNearest Correlation Matrix\n")
print( round(out$RLG,8) )
print( eigen(out$RLG)$val )

## Rousseeuw Molenbergh
## non symmetric matrix
T <- matrix(c(.8, -.9, -.9,
            -1.2, 1.1, .3,
            -.8, .4, .9), nrow = 3, ncol = 3,byrow=TRUE)
```

**smoothLG**
out <- smoothLG(R = T, PD = FALSE, eps=.001)

cat("\nGradient at solution:", out$gr,"\n")
cat("\nNearest Correlation Matrix\n")
print( round(out$RLG,8) )

summary.faMain

Summary Method for an Object of Class faMain

Description

This function summarizes results from a call to faMain.

Usage

## S3 method for class 'faMain'
summary(object, digits = 2, Set = 1,
        HPthreshold = 0.05, PrintLevel = 1, DiagnosticsLevel = 1,
        itemSort = FALSE, ...)

Arguments

object

( Object of class faMain ) The returned object from a call to faMain.

digits

(Integer) Print output with user-specified number of significant digits. Default digits = 2.

Set

The argument Set can be specified as either an integer value (i.e., 1 through the number of unique solution sets) or a character value (i.e., 'UnSpun').

  • Integer Summarize the solution from the specified solution set. If Set = 1, the "global minimum" solution is reported. See faMain for more details about finding the "global" and local minima.

  • 'UnSpun' Summarize the solution from the rotated output that was produced by rotating from the unrotated (i.e., unspun) factor orientation. All other solutions are rotated from a randomly 'spun' rotation (i.e., by orientating the unrotated factor solution via a random orthonormal matrix).

HPthreshold

(Numeric) User-defined threshold for declaring that the absolute value of a factor pattern coefficient is in a hyperplane. The hyperplane count is the number of near-zero (as defined by HPthreshold; see Cattell, 1978, p. 105) elements in the factor pattern matrix. Default HPthreshold = .05.

PrintLevel

(Integer) Controls the level of printing. If PrintLevel = 0 then no output is printed. If PrintLevel = 1 then the standard output will be printed. If PrintLevel = 2 more extensive output (e.g., the Factor Structure Matrix) will be printed. Default PrintLevel = 1.
DiagnosticsLevel

(Integer) Controls the amount of diagnostics information that is computed on the rotation local minima. If DiagnosticsLevel = 1 then only the number of local solution sets will be reported. If DiagnosticsLevel = 2 then the program will determine whether all solutions within a solution set are identical. Default DiagnosticsLevel = 1.

itemSort

(Logical) If TRUE, sort the order of the observed variables to produce a "staircase"-like pattern. Note that this argument cannot handle bifactor models at this time. Defaults to itemSort = FALSE.

Additional arguments affecting the summary produced.

Details

summary.faMain provides various criteria for judging the adequacy of the rotated factor solution(s). After reporting the number of solution sets, (i.e., rotated solutions with the same complexity value) the following measures of factor adequacy are reported for each solution set:

- **Complexity Value**: The rotation complexity value (see faMain for details).
- **Hyperplane Count**: The number of near-zero loadings (defined by HPthreshold) for all factor patterns in a solution set (if MaxWithinSetRMSD > 0 then Hyperplane Count refers to the first factor pattern in the solution set).
- **% Cases (x 100) in Set**: The percentage of factor patterns in each solution set.
- **RMSD**: The root mean squared deviation between the first factor pattern in each solution set with the first factor pattern in the solution set specified by the Set parameter. By default, Set = 1.
- **MaxWithinSetRMSD**: The maximum root mean squared deviation between all within set solutions and the first element in the solution set. When MaxWithinSetRMSD > 0 then the solution set contains non-identical rotated factor patterns with identical complexity values.
- **Converged**: A Logical (TRUE/FALSE) that indicates whether all within set rotations converged.

Note that the printed factor pattern is not sorted even if itemSort is requested in faMain.

Value

- **loadings** (Matrix) Factor loadings for the solution associated with the minimum (maximum) rotation complexity value (default) or the user-chosen solution.
- **Phi** (Matrix) Factor correlation matrix for the solution associated with the minimum (maximum) rotation complexity value (default) or the user-chosen solution.
- **FS** (Matrix) Factor structure matrix for the solution associated with the minimum (maximum) rotation complexity value (default) or the user-chosen solution.
- **Set** (Integer) The returned Set number.
- **h2** (Matrix) Communalities for the returned factor solution. If Bootstrap = TRUE then h2 also returns the bootstrap standard errors and associated confidence bounds from the bootstrap distribution.
• facIndeterminacyMatrix: Factor Indeterminacy values. If Bootstrap = TRUE then facIndeterminacy also returns the bootstrap standard errors and associated confidence bounds from the bootstrap distribution.

• SetComplexityValues: Rotation complexity value for each solution set.

• HP_counts: Hyperplane count for each solution set.

• MaxWithinSetRMSD: If DiagnosticsLevel = 2 then the program will compute within set RMSD values. These values represent the root mean squared deviations of each within set solution with the first solution in a set. If the MaxWithinSetRMSD = 0 for a set, then all within set solutions are identical. If MaxWithinSetRMSD > 0 then at least one solution differs from the remaining solutions within a set (i.e., two solutions with different factor loadings produced identical complexity values).

• RMSD: The root mean squared deviation between the observed and model-implied correlation matrix.

• RMSD: The root mean squared absolute deviation between the observed and model-implied correlation matrix.

• NumberLocalSolutions: The number of local solution sets.

• LocalSolutions: A list of local solutions (factor loadings, factor correlations, etc).

• rotate: Designates which rotation method was applied.

Author(s)

• Niels G. Waller (nwaller@umn.edu)

• Casey Giordano (Giord023@umn.edu)

References


See Also

Other Factor Analysis Routines: BiFAD, Box26, GenerateBoxData, Ledermann, SLi, SchmidLeiman, faAlign, faEKC, faMain, faScores, faSort, faStandardize, faX, fals, fapa, fareg, orderFactors, print.faMain, promaxQ

Examples

## Load Thurstone's Box data from the fungible library
library(fungible)
data(Box26)

## Create a matrix from Thurstone's solution
## Used as a target matrix to sort columns of the estimated solution
ThurstoneSolution <- matrix(c( .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03, .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03, .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03, .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03, .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03, .95, .01, .01, .02, .92, .01, .02, .05, .91, .59, .64, -.03), nrow = 5, byrow = TRUE)

## Calculate the root mean squared deviation between the observed and model-implied correlation matrix
RMSD <- function(matrix1, matrix2) sqrt(rowMeans((matrix1 - matrix2)^2))

## Calculate the root mean squared absolute deviation between the observed and model-implied correlation matrix
RMSD_ab <- function(matrix1, matrix2) rowMeans(abs(matrix1 - matrix2))

## Compute within set RMSD values
MaxWithinSetRMSD <- function(matrix, firstSolution) apply(matrix - firstSolution, 2, function(x) RMSD(x, firstSolution[1,])}

## Compute within set RMSD values
MaxWithinSetRMSD <- function(matrix, firstSolution) apply(matrix - firstSolution, 2, function(x) RMSD(x, firstSolution[1,])}

## Create a list of local solutions
LocalSolutions <- list()

## Rotate the solutions
rotate <- function(matrix) matrix
.60, .00, .62, 
-.04, .60, .58, 
.81, .38, .01, 
.35, .79, .01, 
.79, -.01, .41, 
.40, -.02, .79, 
-.04, .74, .40, 
-.02, .41, .74, 
.74, -.77, .06, 
-.74, .77, -.06, 
.74, .02, -.73, 
-.74, -.02, .73, 
-.07, .80, -.76, 
-.07, -.80, .76, 
.51, .70, -.03, 
.56, -.04, .69, 
-.02, .60, .58, 
.50, .69, -.03, 
.52, -.01, .68, 
-.01, .60, .55, 
.43, .46, .45, 
.31, .51, .46), nrow = 26, ncol = 3, 
byrow=TRUE)

## Example 1: Multiple solution sets.
## Ignore warnings about non-positive definite sample correlation matrix
suppressWarnings(
  fout <- faMain(R = Box26,
                 numFactors = 3,
                 facMethod = 'faregLS',
                 rotate = 'infomaxQ',
                 targetMatrix = ThurstoneSolution,
                 rotateControl =
                 list(numberStarts = 25, ## increase in real problem
                       standardize = 'none'),
                 Seed = 123)
)

## Summarize the factor analytic output
summary(object = fout, 
        digits = 2, 
        Set = 2, 
        HPthreshold = .10, 
        PrintLevel = 1, 
        DiagnosticsLevel = 2)

## Example 2: Bootstrap Illustration
## Step 1: In an initial analysis, confirm that all rotations converge
## to a single minimum complexity value.
## Step 2: If Step 1 is satisfied then generate bootstrap samples.

## Load Amazon box data
data("AmzBoxes")
## Convert box dimensions into Thurstone's indicators

```r
BoxData <- GenerateBoxData(AmzBoxes[, 2:4],
                             BoxStudy = 26,       ## Select columns 2, 3, & 4
                             Reliability = 0.75,  ## 26 indicators
                             SampleSize = 200,    ## Add unreliability
                             ModApproxErrVar = 0.1,   ## Add sampling error
                             NMinorFac = 50,       ## Number of minor factors
                             epsTKL = 0.2,        ## Spread of minor factor influence
                             SeedErrorFactors = 1,   ## Reproducible starting seed
                             SeedMinorFactors = 2,   ## Reproducible starting seed
                             PRINT = FALSE,       ## Suppress some output
                             LB = FALSE,         ## Do not set lower-bounds
                             LBVal = 1,          ## Lower bound value (ignored)
                             Constant = 0)      ## Do not add constant to data
```

## Analyze new box data with added measurement error

```r
fout <- faMain(X = BoxData$BoxDataE,
                numFactors = 3,
                facMethod = 'fapa',
                rotate = 'infomaxQ',
                targetMatrix = ThurstoneSolution,
                bootstrapSE = FALSE,
                rotateControl =
                    list(numberStarts = 25, ## increase in real problem
                         standardize = 'CM'),
                Seed = 1)
```

## Summarize factor analytic output

```r
sout <- summary(object = fout,
                 Set = 1,
                 PrintLevel = 1)
```

## Generate bootstrap samples

```r
fout <- faMain(X = BoxData$BoxDataE,
                numFactors = 3,
                facMethod = 'fapa',
                rotate = 'infomaxQ',
                targetMatrix = ThurstoneSolution,
                bootstrapSE = TRUE,
                numBoot = 25,   ## increase in real problem
                rotateControl =
                    list(numberStarts = 1,
                         standardize = 'CM'),
                Seed = 1)
```

## Summarize factor analytic output with bootstraps

```r
sout <- summary(object = fout,
                 Set = 1,
                 PrintLevel = 2)
```

## To print a specific solution without computing diagnostics and
## summary information, use the print function.

print(fout,
    Set = 1)

summary.monte  

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, ...)

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

```r
## set up a 'monte' run for the Fisher iris data
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, -0.602))

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 = 0.3,
                  compactness = c(1, 1, 1),
                  sortMeans = TRUE)

summary(my.iris)
```

summary.monte1

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.
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)
diag(cormat) <- 1

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

summary(nontaxon.dat)
```

svdNorm

Compute theta surrogates via normalized SVD scores

Description

Compute theta surrogates by calculating the normalized left singular vector of a (mean-centered) data matrix.

Usage

`svdNorm(data)`

Arguments

data N(subjects)-by-p(items) matrix of 0/1 item response data.

Value

the normalized left singular vector of the mean centered data matrix.

svdNorm will center the data automatically.
tetcor

Author(s)

Niels Waller

Examples

NSubj <- 2000

## example item parameters for sample data: k=1 FMP
b <- matrix(c(
  #b0 b1 b2 b3 b4 b5 b6 b7 k
  1.675, 1.974, -0.068, 0.053, 0, 0, 0, 0, 1,
  1.550, 1.805, -0.230, 0.032, 0, 0, 0, 0, 1,
  1.282, 1.063, -0.193, 0.003, 0, 0, 0, 0, 1,
  0.704, 1.376, -0.107, 0.040, 0, 0, 0, 0, 1,
  1.417, 1.413, 0.021, 0.000, 0, 0, 0, 0, 1,
  -0.008, 1.349, -0.195, 0.144, 0, 0, 0, 0, 1,
  0.512, 1.538, -0.089, 0.082, 0, 0, 0, 0, 1,
  0.122, 0.601, -0.082, 0.119, 0, 0, 0, 0, 1,
  1.801, 1.211, 0.015, 0.000, 0, 0, 0, 0, 1,
  -0.207, 1.191, 0.066, 0.033, 0, 0, 0, 0, 1,
  -0.215, 1.291, -0.087, 0.029, 0, 0, 0, 0, 1,
  0.259, 0.875, 0.177, 0.072, 0, 0, 0, 0, 1,
  -0.423, 0.942, 0.064, 0.094, 0, 0, 0, 0, 1,
  0.113, 0.795, 0.124, 0.110, 0, 0, 0, 0, 1,
  1.030, 1.525, 0.200, 0.076, 0, 0, 0, 0, 1,
  0.140, 1.209, 0.082, 0.148, 0, 0, 0, 0, 1,
  0.429, 1.480, -0.008, 0.061, 0, 0, 0, 0, 1,
  0.089, 0.785, -0.065, 0.018, 0, 0, 0, 0, 1,
  -0.516, 1.013, 0.016, 0.023, 0, 0, 0, 0, 1,
  0.143, 1.315, -0.011, 0.136, 0, 0, 0, 0, 1,
  0.347, 0.733, -0.121, 0.041, 0, 0, 0, 0, 1,
  -0.074, 0.869, 0.013, 0.026, 0, 0, 0, 0, 1,
  0.630, 1.484, -0.001, 0.000, 0, 0, 0, 0, 1),
nrow=23, ncol=9, byrow=TRUE)

# generate data using the above item parameters
data<-genFMPData(NSubj=NSubj, bParam=b, seed=345)$data

# compute (initial) surrogate theta values from
# the normed left singular vector of the centered
# data matrix
thetaInit<-svdNorm(data)

tetcor

Compute ML Tetrachoric Correlations

Description

Compute ML tetrachoric correlations with optional bias correction and smoothing.
Usage

tetcor(X, y = NULL, BiasCorrect = TRUE, stderr = FALSE,
  Smooth = TRUE, max.iter = 5000, 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.
stderr  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 stderr = FALSE, tetcor returns a matrix of tetrachoric correlations. If stderr = 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

  Divgi, D. R. (1979) Calculation of the tetrachoric correlation coefficient. Psychometrika, 44, 169-
  172.

Examples

  ## 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
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("Pearson r: ", round(cor(X)[1,2], 2))

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

## calculate tetrachoric correlation
cat("Tetrachoric r: ", round(tetcor(binaryData)$r[1,2], 2))

## calculate Quasi-tetrachoric correlation
cat("Quasi-tetrachoric r: ", round(tetcorQuasi(binaryData), 2))
```

**Description**

Factor Pattern and Factor Correlations for Thurstone’s 20 hypothetical box attributes.

**Usage**

`data(ThurstoneBox20)`
Format

This is a list containing the Loadings (original factor pattern) and Phi matrix (factor correlation matrix) from Thurstone’s 20 Box problem (Thurstone, 1940, p. 227). The original 20-variable Box problem contains measurements on the following score functions of box length (x), width (y), and height (z). Box20 variables:

1. \(x^2\)
2. \(y^2\)
3. \(z^2\)
4. \(xy\)
5. \(xz\)
6. \(yz\)
7. \(\sqrt{x^2 + y^2}\)
8. \(\sqrt{x^2 + z^2}\)
9. \(\sqrt{y^2 + z^2}\)
10. \(2x + 2y\)
11. \(2x + 2z\)
12. \(2y + 2z\)
13. \(\log(x)\)
14. \(\log(y)\)
15. \(\log(z)\)
16. \(xyz\)
17. \(\sqrt{x^2 + y^2 + z^2}\)
18. \(\exp(x)\)
19. \(\exp(y)\)
20. \(\exp(z)\)

Details

Two data sets have been described in the literature as Thurstone’s Box Data (or Thurstone’s Box Problem). The first consists of 20 measurements on a set of 20 hypothetical boxes (i.e., Thurstone made up the data). Those data are available in Box20.

References


See Also

AmzBoxes, Box20, Box26, GenerateBoxData

Examples

data(ThurstoneBox20)
ThurstoneBox20
Description

Factor Pattern Matrix for Thurstone’s 26 box attributes.

Usage

data(ThurstoneBox26)

Format

The original factor pattern (3 graphically rotated centroid factors) from Thurstone’s 26 hypothetical box data as reported by Thurstone (1947, p. 371). The so-called Thurstone invariant box problem contains measurements on the following 26 functions of length (x), width (y), and height (z). Box 26 variables:

1. x
2. y
3. z
4. xy
5. xz
6. yz
7. x^2 * y
8. x * y^2
9. x^2 * z
10. x * z^2
11. y^2 * z
12. y * z^2
13. x/y
14. y/x
15. x/z
16. z/x
17. y/z
18. z/y
19. 2x + 2y
20. 2x + 2z
21. 2y + 2z
22. sqrt(x^2 + y^2)
23. sqrt(x^2 + z^2)
24. sqrt(y^2 + z^2)
25. xyz
26. sqrt(x^2 + y^2 + z^2)
Details

Two data sets have been described in the literature as Thurstone’s Box Data (or Thurstone’s Box Problem). The first consists of 20 measurements on a set of 20 hypothetical boxes (i.e., Thurstone made up the data). Those data are available in Box20. The second data set was collected by Thurstone to provide an illustration of the invariance of simple structure factor loadings. In his classic textbook on multiple factor analysis (Thurstone, 1947), Thurstone states that “[m]easurements of a random collection of thirty boxes were actually made in the Psychometric Laboratory and recorded for this numerical example. The three dimensions, x, y, and z, were recorded for each box. A list of 26 arbitrary score functions was then prepared” (p. 369). The raw data for this example were not published. Rather, Thurstone reported a correlation matrix for the 26 score functions (Thurstone, 1947, p. 370). Note that, presumably due to rounding error in the reported correlations, the correlation matrix for this example is non positive definite. This file includes the rotated centroid solution that is reported in his book (Thurstone, 1947, p. 371).

References


See Also

Box20, AmzBoxes

Examples

data(ThurstoneBox26)
ThurstoneBox26

vcos(x, y)

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

```r
x <- rnorm(5)
y <- rnorm(5)
vcos(x, y)
```

---

**vnorm**

*Norm a Vector to Unit Length*

**Description**

Norm a vector to unit length.

**Usage**

```r
vnorm(x)
```

**Arguments**

- `x` An n by 1 vector.

**Value**

the scaled (i.e., unit length) input vector

**Author(s)**

Niels Waller

**Examples**

```r
x <- rnorm(5)
v <- vnorm(x)
print(v)
```
Index

*Topic Statistics
  adfCor, 4
  adfCov, 5
  corSmooth, 20
  d2r, 23
  eigGen, 25
  faAlign, 29
  faEKC, 33
  fals, 34
  kurt, 90
  normalCor, 102
  r2d, 111
  rcor, 115
  seBeta, 129
  seBetaCor, 131
  seBetaFixed, 132
  skew, 139
  smoothKB, 148
  tetcor, 159
  tetcorQuasi, 161
  vcos, 165

*Topic Statistics
  faSort, 52

*Topic datagen
  bigen, 14
  corSample, 19
  enhancement, 26
  genCorr, 77
  monte, 92
  montel, 100
  rarc, 112
  rcone, 114
  rEllipsoid, 116
  rGivens, 119
  rMAP, 120

*Topic datasets
  AmzBoxes, 6
  Box20, 16
  Box26, 17

  HS9Var, 85
  HW, 86
  ThurstoneBox20, 162
  ThurstoneBox26, 164

*Topic fungible
  faAlign, 29
  fungible, 62
  fungibleExtrema, 63
  fungibleL, 65
  fungibleR, 67
  RnpdMAP, 122

*Topic statistics
  BadRBY, 7
  BadRJN, 8
  BadRKtB, 8
  BadRLG, 9
  BadRRM, 9
  eap, 23
  erf, 28
  faMAP, 43
  FMP, 58
  FMPMonotonicityCheck, 61
  FUP, 72
  gen4PMDdata, 75
  genFMPData, 82
  irf, 86
  itemDescriptives, 88
  normF, 103
  restScore, 117
  smoothAPA, 144
  smoothBY, 146
  smoothLG, 149
  svdNorm, 158
  vnorm, 166

*Topic stats
  genPhi, 83
  rmsd, 121
  simFA, 134

  adfCor, 4, 102
INDEX

faScores, 12, 18, 30, 33, 34, 41, 46, 47, 49, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
faSort, 12, 18, 30, 33, 34, 41, 46, 47, 50, 52, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
faStandardize, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 54, 57, 81, 92, 105, 107, 108, 110, 128, 143, 153
faX, 10–12, 18, 30, 33–36, 39, 41, 46, 47, 50, 53, 55, 55, 81, 92, 105, 107, 108, 110, 126, 128, 140–143, 153

floor, 91
FMP, 58
FMPMonotonicityCheck, 61
fungible, 62
d2r, 23

eap, 23
eigGen, 25
enhancement, 26
erf, 28

defadCov, 5
AmzBoxes, 6, 18, 163, 165
BadRBY, 7
BadRJN, 8
BadRktB, 8
BadRLG, 9
BadRMM, 9
BiFAD, 10, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
bigen, 14
Box20, 16, 18, 163, 165
Box26, 12, 17, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153, 163

cor, 37
corSample, 19
corSmooth, 20
cosMat, 21
d2r, 23
eap, 23
eigGen, 25
enhancement, 26
erf, 28

faAlign, 12, 18, 29, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
factanal, 10, 35, 55, 108, 126, 140
faEKC, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
fals, 10, 12, 18, 30, 33, 34, 35, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 108, 110, 126, 128, 140, 143, 153
faMain, 10, 12, 18, 30, 33, 34, 35, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 126, 128, 140, 142, 143, 151–153
faMAP, 43
fapa, 10, 12, 18, 30, 33, 34, 36, 41, 44, 47, 50, 53, 55, 57, 81, 92, 105, 107, 108, 110, 126, 128, 141, 143, 153
fareg, 10, 12, 18, 30, 33, 34, 36, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 108, 110, 126, 128, 141, 143, 153

kurt, 90, 139
Ledermann, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 91, 105, 107, 110, 128, 143, 153

monte, 92, 101
monte1, 100

normalCor, 102
normF, 103
Omega, 103
INDEX

orderFactors, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 153
plot.monte, 106
print.faMain, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 106, 110, 128, 143, 153
print.summary.monte(summary.monte), 156
print.summary.monte1(summary.monte1), 157
promax, 107
promaxQ, 11, 12, 18, 30, 33, 34, 37, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 107, 127, 128, 142, 143, 153
r2d, 111
rarc, 112
rcone, 114
rcor, 115
rellipsoid, 116
restScore, 117
rGivens, 119
rMAP, 120
rmsd, 121
RnpdMAP, 122
SchmidLeiman, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 125, 140, 143, 153
seBeta, 129, 133
seBetaCor, 131
seBetaFixed, 132
simFA, 134
skew, 90, 139
SLi, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 140, 153
smoothAPA, 144
smoothBY, 146
smoothKB, 148
smoothLG, 149
summary.faMain, 12, 18, 30, 33, 34, 41, 46, 47, 50, 53, 55, 57, 81, 92, 105, 107, 110, 128, 143, 151
summary.monte, 101, 156
summary.monte1, 101, 157
svdNorm, 58, 73, 158
tecor, 159
tecorQuasi, 161
ThurstoneBox20, 162
ThurstoneBox26, 164
tecorQuasi, 161
vcon, 165
vnorm, 166