User manual for the psych package Version 1.0.23

William Revelle
Department of Psychology
Northwestern University

May 27, 2007

*Contents

`00psych-package` *A package for personality, psychometric, and psychological research*

Description

The psych package has been developed at Northwestern University to include functions most useful for personality and psychological research. Some of the functions (e.g., `describe` and `pairs.panels`) are useful for basic descriptive analyses.

Psychometric applications include routines for Very Simple Structure (`VSS`), Item Cluster Analysis (`ICLUST`) and principal axes factor analysis (`factor.pa`), as well as functions to do Schmid Leiman transformations (`schmid`) to transform a hierarchical factor structure into a bifactor solution and to graph both structures (`omega.graph`) and to calculate reliability coefficients alpha (`score.items`), beta (`ICLUST`) and McDonald's omega (`omega` and `omega.graph`).

Additional functions make for more convenient descriptions of item characteristics. Functions under development include 1 and 2 parameter Item Response measures.

A number of procedures have been developed as part of the Synthetic Aperture Personality Assessment (SAPA) project. These routines facilitate forming and analyzing composite scales equivalent to using the raw data but doing so by adding within and between cluster/scale item correlations. These functions include extracting clusters from factor loading matrices (`factor2cluster`), synthetically forming clusters from correlation matrices (`cluster.cor`), and finding multiple correlation from correlation matrices (`mat.regress`).

The most recent development version of the package is always available for download as a source file from the repository at `http://personality-project.org/r/src/contrib/`

Details

The psych package is a combination of multiple source files maintained at the `http://personality-project.org/r` repository: "useful.r", VSS.r., ICLUST.r, omega.r, etc."useful.r" is a set of routines for easy data entry (`read.clipboard`), simple descriptive statistics (`describe`), and splom plots combined with correlations (`pairs.panels`, adapted from the help files of pairs).

The `VSS` routines allow for testing the number of factors (`VSS`), showing plots (`VSS.plot`) of goodness of fit, and basic routines for estimating the number of factors/components to extract by examining the scree plot (`VSS.scree`) or comparing with the scree of an equivalent matrix of random numbers (`VSS.parallel`) .

In addition, there are routines for hierarchical factor analysis using Schmid Leiman tranformations (`omega`, `omega.graph`) as well as Item Cluster analysis (`ICLUST`, `ICLUST.graph`).

The more important functions in the package are for the analysis of multivariate data, with an emphasis upon those functions useful in scale construction of item composites.

When given a set of items from a personality inventory, one goal is to combine these into higher level item composites. This leads to several questions:

1) What are the basic properties of the data? `describe` reports basic summary statistics (mean, sd, median, mad, range, minimum, maximum, skew, kurtosis, standard error) for

vectors, columns of matrices, or data.frames. `describe.by` provides descriptive statistics, organized by a grouping variable. `pairs.panels` shows scatter plot matrices (SPLOMs) as well as histograms and the Pearson correlation for scales or items.

2) What is the most appropriate number of item composites to form? After finding either standard Pearson correlations, or finding tetrachoric or polychoric correlations using a wrapper (`poly.mat`) for John Fox's hetcor function, the dimensionality of the correlation matrix may be examined. The number of factors/components problem is a standard question of factor analysis, cluster analysis, or principal components analysis. Unfortunately, there is no agreed upon answer. The Very Simple Structure (`VSS`) set of procedures has been proposed as on answer to the question of the optimal number of factors. Other procedures (`VSS.scree`, `VSS.parallel`, and `fa.parallel`) also address this question.

3) What are the best composites to form? Although this may be answered using principal components (principal) or factor analysis (factor.pa) and to show the results graphically (fa.graph), it is sometimes more useful to address this question using cluster analytic techniques. Previous versions of `ICLUST` (e.g., Revelle, 1979) have been shown to be particularly successful at doing this. Graphical output from `ICLUST.graph` uses the Graphviz dot language and allows one to write files suitable for Graphviz. (As of May, 2007, Rgraphviz now work on the Intel-Mac and two graphic functions take advantage of this. Graphviz still produces cleaner output.)

4) How well does a particular item composite reflect a single construct? This is a question of reliability and general factor saturation. Multiple solutions for this problem result in (Cronbach's) alpha (`score.items`), (Revelle's) Beta (`ICLUST`), and (McDonald's) `omega`. Functions to estimate all three of these are included in psych.

5) For some applications, data matrices are synthetically combined from sampling different items for different people. So called Synthetic Aperture Personality Assessement (SAPA) techniques allow the formation of large correlation or covariance matrices even though no one person has taken all of the items. To analyze such data sets, it is easy to form item composites based upon the covariance matrix of the items, rather than original data set. These matrices may then be analyzed using a number of functions (e.g., `cluster.cor`, `factor.pa`, `ICLUST`, `principal` , `mat.regress`, and `factor2cluster`.

6) More typically, one has a raw data set to analyze. `score.items` will score data sets on multiple scales, reporting the scale scores, item-scale and scale-scale correlations, as well as coefficient alpha and alpha-1. Using a 'keys' matrix, scales can have overlapping or independent items.

Package:	psych
Type:	Package
Version:	1.0-23
Date:	2007-5-24,
License:	GPL version 2 or newer

Index:

psych A package for personality, psychometric, and psychological research.

Useful data entry and descriptive statistics

describe Basic descriptive statistics useful for psychometrics
describe.by Find summary statistics by groups
read.clipboard shortcut for reading from the clipboard
read.clipboard.csv shortcut for reading comma delimited files from clipboard
pairs.panels SPLOM and correlations for a data matrix
multi.hist Histograms of multiple variables arranged in matrix form
skew Calculate skew for a vector, each column of a matrix, or data.frame
kurtosi Calculate kurtosis for a vector, each column of a matrix or dataframe

error.crosses Two way error bars
geometric.mean Find the geometric mean of a vector or columns of a data.frame
harmonic.mean Find the harmonic mean of a vector or columns of a data.frame

Data reduction through cluster and factor analysis

factor.pa Do a principal Axis factor analysis
fa.graph Show the results of a factor analysis or principal components analysis graphically
principal Do an eigen value decomposition to find the principal components of a matrix
fa.parallel Scree test and Parallel analysis
`ICLUST` Apply the ICLUST algorithm
ICLUST.graph Graph the output from ICLUST using the dot language
ICLUST.rgraph Graph the output from ICLUST using rgraphviz
poly.mat Find the polychoric correlations for items (uses J. Fox's hetcor
omega Calculate the omega estimate of factor saturation (requires the GPArotation package
omega.graph Draw a hierarchical or SL orthogonalized solution
schmid Apply the Schmid Leiman transformation to a correlation matrix

score.items Combine items into multiple scales and find alpha
VSS Apply the Very Simple Structure criterion to determine the appropriate number of factors.
VSS.parallel Do a parallel analysis to determine the number of factors for a random matrix
VSS.plot Plot VSS output
VSS.scree Show the scree plot of the factor/principal components
VSS.simulate Generate simulated data for the factor model
make.hierarchical Generate simulated correlation matrices with hierarchical structure

Procedures particularly useful for Synthetic Aperture Personality Assessment

alpha.scale Find coefficient alpha for a scale (see also score.items)
correct.cor Correct a correlation matrix for unreliability
count.pairwise Count the number of complete cases when doing pair wise correlations
cluster.cor find correlations of composite variables from larger matrix
cluster.loadings find correlations of items with composite variables from a larger matrix
eigen.loadings Find the loadings when doing an eigen value decomposition
factor.pa Do a principal Axis factor analysis
factor2cluster extract cluster definitions from factor loadings
factor.congruence Factor congruence coefficient

factor.fit How well does a factor model fit a correlation matrix
factor.model Reproduce a correlation matrix based upon the factor model
factor.residuals Fit = data - model
factor.rotate "hand rotate" factors
mat.regress multiple regression from matrix input
principal Do an eigen value decomposition to find the principal components of a matrix

Functions for generating simulated data sets
circ.sim Generate a two dimensional circumplex item structure
item.sim Generate a two dimensional simple structrue with particular item characteristics
congeneric.sim Generate a one factor congeneric reliability structure
psycho.demo Create artificial data matrices for teaching purposes

Miscellaneous functions

fisherz Apply the Fisher r to z transform
paired.r Test for the difference of two paired correlations
phi2poly Given a phi coefficient, what is the polychoric correlation
poly.mat Use John Fox's hetcor to create a matrix of correlations from a data.frame or matrix of integer values
polychor.matrix Use John Fox's polycor to create a matrix of correlations (not yet very useful)

Functions that are under development and not recommended for casual use
irt.item.diff.rasch IRT estimate of item difficulty with assumption that theta = 0
irt.person.rasch Item Response Theory estimates of theta (ability) using a Rasch like model

test.psych Run a test of the major functions on 5 different data sets. Primarily for development purposes. Although the output can be used as a demo of the various functions.

Note

Development versions of this package are maintained at the local repository `http://personality-project.org/r` along with further documentation. Specify that you are downloading a source package.

Some functions require other packages. Specifically, omega and schmid require the GPArotation package, and poly.mat, phi2poly and polychor.matrix requires John Fox's polychor package.

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illiniois
`http://personality-project.org/revelle.html`

Maintainer: William Revelle <revelle@northwestern.edu>

References

A general guide to personality theory and research may be found at the personality-project http://personality-project.org. See also the short guide to R at http://personality-project.org/r. In addition, see An Introduction to Psychometric Theory with applications in R (Revelle, in preparation) at http://personality-project.org/r/book/

Examples

```
#See the separate man pages
```

`alpha.scale` *Cronbach alpha for a scale*

Description

Find Cronbach's coefficient alpha given a scale and a data.frame of the items in the scale. For X, a total score composed of items in the data.frame Y, find Cronbach's alpha.

Usage

```
alpha.scale(x, y)
```

Arguments

`x`	A scale made by summing items
`y`	A data frame of items

Details

Alpha is one of several estimates of the internal consistency reliability of a test. Perhaps because it is so easy to calculate, it is without doubt the most frequently reported measure of internal consistency reliability. Alpha is the mean of all possible spit half reliabilities (corrected for test length). For a unifactorial test, it is a reasonable estimate of the first factor saturation, although if the test has any microstructure (i.e., if it is "lumpy") coefficients beta and omega are more appropriate estimates of the general factor saturation.

Alpha is a positive function of the number of items in a test as well as the average intercorrelation of the items in the test. When calculated from the item variances and total test variance, as is done here, alpha is sensitive to differences in the item variances. Alternative functions `score.items` and `cluster.cor` will also score multiple scales and report more useful statistics. "Standardized" alpha is calculated from the inter-item correlations and will differ from raw alpha. Standardized alpha can be found by using `cluster.cor`.

Value

Coefficient alpha

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

http://personality-project.org/revelle/syllabi/405.syllabus.html

See Also

score.items, cluster.cor, ICLUST, omega, describe,pairs.panels, alpha in psychometrics

Examples

```
y <- attitude     #from the datasets package
x <- rowSums(y)   #find the sum of the seven attitudes
alpha.scale(x,y)
#[1] 0.8431428
#compare with standardized alpha:
st.alpha <- cluster.cor(rep(1,7),cor(attitude),digits=4)
st.alpha
#compare with score.items
si <- score.items(rep(1,7), attitude,digits=3)
si$alpha

## The function is currently defined as
function (x,y)   #find coefficient alpha given a scale and
#a data.frame of the items in the scale
        {
                n=length(y)          #number of variables
                Vi=sum(diag(var(y,na.rm=TRUE)))     #sum of item variance
                Vt=var(x,na.rm=TRUE)                #total test variance
                ((Vt-Vi)/Vt)*(n/(n-1))}              #alpha
```

---

circ.simulation *Simulations of circumplex and simple structure*

Description

Rotations of factor analysis and principal components analysis solutions typically try to represent correlation matrices as simple structured. An alternative structure, appealing to some, is a circumplex structure where the variables are uniformly spaced on the perimeter of a circle in a two dimensional space. Generating these data is straightforward, and is useful for exploring alternative solutions to affect and personality structure.

Usage

```
circ.simulation(samplesize=c(100,200,400,800), numberofvariables=c(16,32,48,72))
```

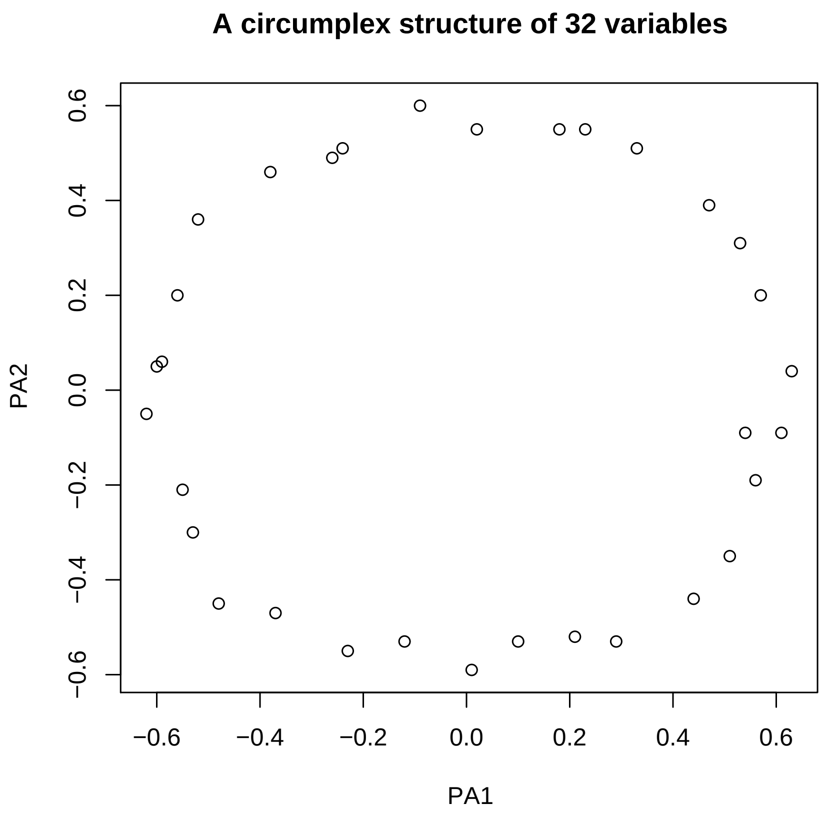

Figure 1: A circumplex structure is typically found for emotional words or for interpersonal behaviors. Tests of circumplex structure can be validated by analyzing simulated structures. Compare this structure to a simple structure Figure 7. Data generated using the circ.sim function.

Arguments

`samplesize`	a vector of sample sizes to simulate
`numberofvariables`	vector of the number of variables to simulate

Details

"A common model for representing psychological data is simple structure (Thurstone, 1947). According to one common interpretation, data are simple structured when items or scales have non-zero factor loadings on one and only one factor (Revelle & Rocklin, 1979). Despite the commonplace application of simple structure, some psychological models are defined by a lack of simple structure. Circumplexes (Guttman, 1954) are one kind of model in which simple structure is lacking.

"A number of elementary requirements can be teased out of the idea of circumplex structure. First, circumplex structure implies minimally that variables are interrelated; random noise does not a circumplex make. Second, circumplex structure implies that the domain in question is optimally represented by two and only two dimensions. Third, circumplex structure implies that variables do not group or clump along the two axes, as in simple structure, but rather that there are always interstitial variables between any orthogonal pair of axes (Saucier, 1992). In the ideal case, this quality will be reflected in equal spacing of variables along the circumference of the circle (Gurtman, 1994; Wiggins, Steiger, & Gaelick, 1981). Fourth, circumplex structure implies that variables have a constant radius from the center of the circle, which implies that all variables have equal communality on the two circumplex dimensions (Fisher, 1997; Gurtman, 1994). Fifth, circumplex structure implies that all rotations are equally good representations of the domain (Conte & Plutchik, 1981; Larsen & Diener, 1992)." (Acton and Revelle, 2004)

Acton and Revelle reviewed the effectiveness of 10 tests of circumplex structure and found that four did a particularly good job of discriminating circumplex structure from simple structure, or circumplexes from ellipsoidal structures. Unfortunately, their work was done in Pascal and is not easily available. Here we release R code to do the four most useful tests:

The Gap test of equal spacing

Fisher's test of equality of axes

A test of indifference to Rotation

A test of equal Variance of squared factor loadings across arbitrary rotations.

Included in this set of functions are simple procedure to generate circumplex structured or simple structured data, the four test statistics, and a simple simulation showing the effectiveness of the four procedures.

Value

A data.frame with simulation results for circumplex, ellipsoid, and simple structure data sets for each of the four tests.

Note

The simulations default values are for sample sizes of 100,200, 400, and 800 cases, with 16, 32, 48 and 72 items.

Author(s)

William Revelle

References

Acton, G. S. and Revelle, W. (2004) Evaluation of Ten Psychometric Criteria for Circumplex Structure. Methods of Psychological Research Online, Vol. 9, No. 1 http://www.dgps.de/fachgruppen/methoden/mpr-online/issue22/mpr110_10.pdf

See Also

See Also as circ.tests, circ.sim

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
demo <- circ.simulation()
 boxplot(demo[3:14])
 title("4 tests of Circumplex Structure",sub="Circumplex, Ellipsoid, Simple Structure")
## The function is currently defined as
function(samplesize=c(100,200,400,800), numberofvariables=c(16,32,48,72))  {
 ncases=length(samplesize)
 nvar <- length(numberofvariables)
  results <- matrix(NA,ncol=ncases,nrow=nvar*ncases)
  results.ls <- list()
   case <- 1
  for (ss in 1:ncases) {
    for (nv in 1:nvar) {

      circ.data <- circ.sim(nvar=numberofvariables[nv],nsub=samplesize[ss])
      sim.data <-  circ.sim(nvar=numberofvariables[nv],nsub=samplesize[ss],circum=FALSE)
      elipse.data <- circ.sim(nvar=numberofvariables[nv],nsub=samplesize[ss],yloading=.4)
      r.circ<- cor(circ.data)
      r.sim <- cor(sim.data)
      r.elipse <- cor(elipse.data)
      pc.circ <- principal(r.circ,2)
      pc.sim <- principal(r.sim,2)
      pc.elipse <- principal(r.elipse,2)
      case <- case + 1
      results.ls[[case]] <- list(numberofvariables[nv],samplesize[ss],circ.tests(pc.circ),circ.tests(pc.e
     }
     }
     results.mat <- matrix(unlist(results.ls),ncol=14,byrow=TRUE)
    colnames(results.mat) <- c("nvar","n","c-gap","c-fisher","c-RT","c-VT","e-gap","e-fisher","e-RT","e-V
    results.df <- data.frame(results.mat)
```

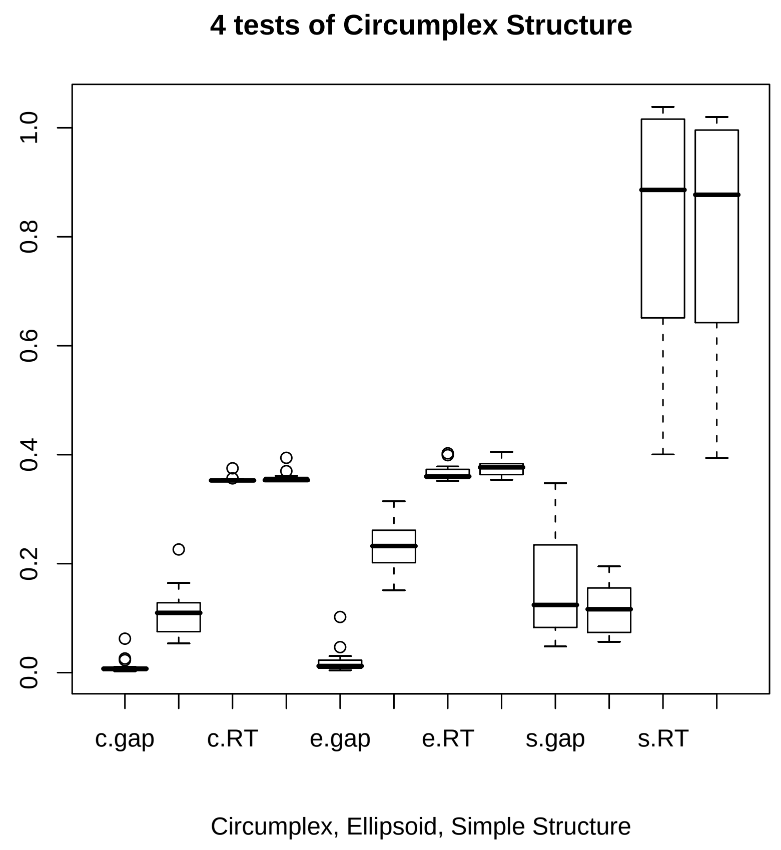

Figure 2: Tests for circumplex structure can be shown to discriminate circumplex and simple structure. Here we compare four tests of circumplex structure, the Gap Test, the Fisher Test, the Rotation Test, and the Variance Test of Circumplex structure. The data sets vary on whether or not they have a circumplex or simple structure. See Acton and Revelle (2004) for details.

```
        return(results.df)
}
```

`circ.tests` *Apply four tests of circumplex versus simple structure*

Description

Rotations of factor analysis and principal components analysis solutions typically try to represent correlation matrices as simple structured. An alternative structure, appealing to some, is a circumplex structure where the variables are uniformly spaced on the perimeter of a circle in a two dimensional space. Generating these data is straightforward, and is useful for exploring alternative solutions to affect and personality structure.

Usage

```
circ.tests(loads, loading = TRUE, sorting = TRUE)
```

Arguments

`loads`	A matrix of loadings `loads` here
`loading`	Are these loadings or a correlation matrix `loading` here
`sorting`	Should the variables be sorted `sorting` here

Details

A common model for representing psychological data is simple structure (Thurstone, 1947). According to one common interpretation, data are simple structured when items or scales have non-zero factor loadings on one and only one factor (Revelle & Rocklin, 1979). Despite the commonplace application of simple structure, some psychological models are defined by a lack of simple structure. Circumplexes (Guttman, 1954) are one kind of model in which simple structure is lacking.

A number of elementary requirements can be teased out of the idea of circumplex structure. First, circumplex structure implies minimally that variables are interrelated; random noise does not a circumplex make. Second, circumplex structure implies that the domain in question is optimally represented by two and only two dimensions. Third, circumplex structure implies that variables do not group or clump along the two axes, as in simple structure, but rather that there are always interstitial variables between any orthogonal pair of axes (Saucier, 1992). In the ideal case, this quality will be reflected in equal spacing of variables along the circumference of the circle (Gurtman, 1994; Wiggins, Steiger, & Gaelick, 1981). Fourth, circumplex structure implies that variables have a constant radius from the center of the circle, which implies that all variables have equal communality on the two circumplex dimensions (Fisher, 1997; Gurtman, 1994). Fifth, circumplex structure implies that all rotations are equally good representations of the domain (Conte & Plutchik, 1981; Larsen & Diener, 1992). (Acton and Revelle, 2004)

Acton and Revelle reviewed the effectiveness of 10 tests of circumplex structure and found that four did a particularly good job of discriminating circumplex structure from simple structure, or circumplexes from ellipsoidal structures. Unfortunately, their work was done in Pascal and is not easily available. Here we release R code to do the four most useful tests:

1 The Gap test of equal spacing

2 Fisher's test of equality of axes

3 A test of indifference to Rotation

4 A test of equal Variance of squared factor loadings across arbitrary rotations.

Value

A list of four items is returned. These are the gap, fisher, rotation and variance test results.

`gaps`	gap.test
`fisher`	fisher.test

`RT`	rotation.test
`VT`	variance.test

Note

Of the 10 criterion discussed in Acton and Revelle (2004), these tests operationalize the four most useful.

Author(s)

William Revelle

References

Acton, G. S. and Revelle, W. (2004) Evaluation of Ten Psychometric Criteria for Circumplex Structure. Methods of Psychological Research Online, Vol. 9, No. 1 ~http://www.dgps.de/fachgruppen/methoden/mpr-online/issue22/mpr110_10.pdf

See Also

`circ.simulation`, `circ.sim`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(loads,loading=TRUE,sorting=TRUE) {

circ.gap <- function(loads,loading=TRUE,sorting=TRUE) {
if (loading) {l <- loads$loadings} else {
       l <- loads}
       l<- l[,1:2]
    commun=rowSums(l*l)
    theta=sign(l[,2])*acos(l[,1]/sqrt(commun)) #vector angle in radians
    if(sorting) {theta<- sort(theta)}
    gaps <- diff(theta)
    test <- var(gaps)
    return(test)
    }

circ.fisher <- function(loads,loading=TRUE) {
if (loading) {l <- loads$loadings} else {
       l <- loads}
       l<- l[,1:2]
   radius <- rowSums(l * l)
   test <- sd(radius)/mean(radius)
  return (test)
  }
```

```
 circ.rt <- function(loads,loading=TRUE) {
if (loading) {l <- loads$loadings} else {
      l <- loads}
      l<- l[,1:2]
      qmc <- rep(0,10)
      for (i in 0:9) {theta <- 5*i
       rl <- factor.rotate(l,theta,1,2)
        l2 <- rl*rl
      qmc[i] <- sum(apply(l2,1,var)) }
      test <- sd(qmc)/mean(qmc)
}

circ.v2 <- function(loads,loading=TRUE) {
if (loading) {l <- loads$loadings} else {
      l <- loads}
      l<- l[,1:2]
  crit <- rep(0,10)
      for (i in 0:9) {
               theta <- 5*i
               rl <- factor.rotate(l,theta,1,2)
               l2 <- rl*rl
               suml2 <- sum(l2)
               crit[i] <- var(l2[,1]/suml2)
      }
      test <- sd(crit)/mean(crit)
 return (test)
 }


  gap.test <- circ.gap(loads,loading,sorting)
  fisher.test <- circ.fisher(loads,loading)
  rotation.test <- circ.rt(loads,loading)
  variance.test <- circ.v2(loads,loading)
  circ.tests <- list(gaps=gap.test,fisher=fisher.test,RT=rotation.test,VT=variance.test)
  }
```

cluster.cor *Find correlations of composite variables from a larger matrix*

Description

Given a n x c cluster definition matrix of -1s, 0s, and 1s (the keys) , and a n x n correlation matrix, find the correlations of the composite clusters. The keys matrix can be entered by hand, copied from the clipboard (read.clipboard), or taken as output from the factor2cluster function.

Usage

```
cluster.cor(keys, r.mat, correct = TRUE,digits=2)
```

Arguments

`keys`	A matrix of cluster keys
`r.mat`	A correlation matrix
`correct`	TRUE shows both raw and corrected for attenuation correlations
`digits`	round off answer to digits

Details

This is one of the functions used in the SAPA procedures to form synthetic correlation matrices. Given any correlation matrix of items, it is easy to find the correlation matrix of scales made up of those items. This can also be done from the original data matrix using `score.items`.

A typical use in the SAPA project is to form item composites by clustering or factoring (see `ICLUST`, `principal`), extract the clusters from these results (`factor2cluster`), and then form the composite correlation matrix using `cluster.cor`. The variables in this reduced matrix may then be used in multiple R procedures using `mat.regress`.

The original correlation is pre and post multiplied by the (transpose) of the keys matrix.

Value

`cor`	the (raw) correlation matrix of the clusters
`sd`	standard deviation of the cluster scores
`corrected`	raw correlations below the diagonal, alphas on diagonal, disattenuated above diagonal
`size`	How many items are in each cluster?

Note

See SAPA e.g., Revelle, W. (2006) Synthetic Aperture Personality Assessment. Invited paper at the Midwestern Psychological Association Annual Meeting, Chicago, May, 2006. pdf available at `http://personality-project.org/revelle/publications/sapa.mpa.key.pdf`

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

`http://personality-project.org/r/r.ICLUST.html`

See Also

`factor2cluster`, `mat.regress`, `alpha.scale`, `score.items`

Examples

```
## Not run:
data(attitude)
keys <- matrix(c(1,1,1,0,0,0,0,
                 0,0,0,1,1,1,1),ncol=2)
colnames(keys) <- c("first","second")
r.mat <- cor(attitude)
cluster.cor(keys,r.mat)
## End(Not run)
#$cor
#       first second
#first    1.0    0.6
#second   0.6    1.0
#
#$sd
# first second
#  2.57   3.01
#
#$corrected
#       first second
#first   0.82   0.77
#second  0.60   0.74
#
#$size
# first second
#     3      4

## The function is currently defined as
function(keys,r.mat,correct=TRUE) { #function to extract clusters according to the key vector
#default is to correct for attenuation and show this above the diagonal
#find the correlation matrix of scales made up of items defined in a keys matrix
#(e.g., extracted by factor2cluster)
 #takes as input the keys matrix as well as a correlation matrix of all the items
 if(!is.matrix(keys)) keys <- as.matrix(keys)
 #keys are sometimes a data frame - must be a matrix
 covar <- t(keys) %*% r.mat %*% keys    #matrix algebra is our friend
 var <- diag(covar)
 sd.inv <- 1/sqrt(var)
 ident.sd <- diag(sd.inv,ncol = length(sd.inv))
 cluster.correl <- ident.sd %*% covar  %*% ident.sd
 key.var <- diag(t(keys) %*% keys)
 key.alpha <- ((var-key.var)/var)*(key.var/(key.var-1))
 key.alpha[is.nan(key.alpha)] <- 1 #if only 1 variable to the cluster, then alpha is undefined
 key.alpha[!is.finite(key.alpha)] <- 1
 colnames(cluster.correl) <- names(key.alpha)
 rownames(cluster.correl) <- names(key.alpha)
 if (correct) {cluster.corrected <- correct.cor(cluster.correl,t(key.alpha))
 return(list(cor=cluster.correl,sd=sqrt(var),corrected= cluster.corrected,size=key.var))
 }  #correct for attenuation
 else {
 return(list(cor=cluster.correl,sd=sqrt(var),alpha=key.alpha,size=key.var))}
 }
```

`cluster.fit` *cluster Fit: fit of the cluster model to a correlation matrix*

Description

How well does the cluster model found by `ICLUST` fit the original correlation matrix? A similar algorithm `factor.fit` is found in `VSS`. This function is internal to ICLUST but has more general use as well.

In general, the cluster model is a Very Simple Structure model of complexity one. That is, every item is assumed to represent only one factor/cluster. Cluster fit is an analysis of how well this model reproduces a correlation matrix. Two measures of fit are given: cluster fit and factor fit. Cluster fit assumes that variables that define different clusters are orthogonal. Factor fit takes the loadings generated by a cluster model, finds the cluster loadings on all clusters, and measures the degree of fit of this somewhat more complicated model. Because the cluster loadings are similar to, but not identical to factor loadings, the factor fits found here and by `factor.fit` will be similar.

Usage

```
cluster.fit(original, load, clusters, diagonal = FALSE, digits = 2)
```

Arguments

`original`	The original correlation matrix being fit
`load`	Cluster loadings – that is, the correlation of individual items with the clusters, corrected for item overlap
`clusters`	The cluster structure
`diagonal`	Should we fit the diagonal as well?
`digits`	Number of digits to be used in the output

Details

The cluster model is similar to the factor model: R is fitted by C'C. Where C <- Cluster definition matrix x the loading matrix. How well does this model approximate the original correlation matrix and how does this compare to a factor model?

The fit statistic is a comparison of the original (squared) correlations to the residual correlations. Fit = 1 - r*2/r2 where r* is the residual correlation of data - model and model = C'C.

Value

`clusterfit`	The cluster model is a reduced form of the factor loading matrix. That is, it is the product of the elements of the cluster matrix * the loading matrix.
`factorfit`	How well does the complete loading matrix reproduce the correlation matrix?

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

http://personality-project.org/r/r.ICLUST.html

See Also

VSS, ICLUST, factor2cluster, cluster.cor, factor.fit

Examples

```
 r.mat<- Harman74.cor$cov
 iq.clus <- ICLUST(r.mat,nclusters =2)
 fit <- cluster.fit(r.mat,iq.clus$loadings,iq.clus$clusters)
 fit


## The function is currently defined as
function (original, load, clusters, diagonal = FALSE, digits = 2)
{
    sqoriginal <- original * original
    totaloriginal <- sum(sqoriginal) - diagonal * sum(diag(sqoriginal))
    model <- load %*% t(load)
    residual <- original - model
    sqresid <- residual * residual
    totalresid <- sum(sqresid) - diagonal * sum(diag(sqresid))
    fit <- 1 - totalresid/totaloriginal
    model.1 <- (load * clusters) %*% t(load * clusters)
    residual <- original - model.1
    sqresid <- residual * residual
    totalresid <- sum(sqresid) - diagonal * sum(diag(sqresid))
    fit.1 <- 1 - totalresid/totaloriginal
    cluster.fit <- list(clusterfit = round(fit.1, digits), factorfit = round(fit,
        digits))
  }
```

---

cluster.loadings *Find item by cluster correlations, corrected for overlap and reliability*

Description

Given a n x n correlation matrix and a n x c matrix of -1,0,1 cluster weights for those n items on c clusters, find the correlation of each item with each cluster. If the item is part of the cluster, correct for item overlap. Part of the ICLUST set of functions, but useful for many item analysis problems.

Usage

```
cluster.loadings(keys, r.mat, correct = TRUE, digits = 2)
```

Arguments

`keys`	Cluster keys: a matrix of -1,0,1 cluster weights
`r.mat`	A correlation matrix
`correct`	Correct for reliability
`digits`	Number of digits output

Details

Given a set of items to be scored as (perhaps overlapping) clusters and the intercorrelation matrix of the items, find the clusters and then the correlations of each item with each cluster. Correct for item overlap by replacing the item variance with its average within cluster inter-item correlation.

Although part of ICLUST, this may be used in any SAPA application where we are interested in item- whole correlations of items and composite scales.

These loadings are particularly interpretable when sorted by absolute magnitude for each cluster (see `ICLUST.sort`).

Value

`loadings`	A matrix of item-cluster correlations (loadings)
`cor`	Correlation matrix of the clusters
`corrected`	Correlation matrix of the clusters, raw correlations below the diagonal, alpha on diagonal, corrected for reliability above the diagonal
`sd`	Cluster standard deviations
`alpha`	alpha reliabilities of the clusters
`count`	Number of items in the cluster

Note

Although part of ICLUST, this may be used in any SAPA application where we are interested in item- whole correlations of items and composite scales.

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

ICLUST: `http://personality-project.org/r/r.iclust.html`

See Also

`ICLUST`, `factor2cluster`, `cluster.cor`

Examples

```
## Not run:
 r.mat<- Harman74.cor$cov
 clusters <- matrix(c(1,1,1,rep(0,24),1,1,1,1,rep(0,17)),ncol=2)
 cluster.loadings(clusters,r.mat)
## End(Not run)

## The function is currently defined as
function (keys, r.mat, correct = TRUE, digits = 2)
{
    if (!is.matrix(keys))
        keys <- as.matrix(keys)
    item.covar <- r.mat %*% keys
    covar <- t(keys) %*% item.covar
    var <- diag(covar)
    sd.inv <- 1/sqrt(var)
    key.count <- diag(t(keys) %*% keys)
    if (correct) {
        cluster.correct <- diag((key.count/(key.count - 1)))
        for (i in 1:ncol(keys)) {
            if (key.count[i] == 1) {
                cluster.correct[i, i] <- 1
            }
            else {
                item.covar[, i] <- item.covar[, i] - keys[, i]
            }
        }
        correction.factor <- keys %*% cluster.correct
        correction.factor[correction.factor < 1] <- 1
        item.covar <- item.covar * correction.factor
    }
    ident.sd <- diag(sd.inv, ncol = length(sd.inv))
    cluster.loading <- item.covar %*% ident.sd
    cluster.correl <- ident.sd %*% covar %*% ident.sd
    key.alpha <- ((var - key.count)/var) * (key.count/(key.count -
        1))
    key.alpha[is.nan(key.alpha)] <- 1
    key.alpha[!is.finite(key.alpha)] <- 1
    colnames(cluster.loading) <- colnames(keys)
    colnames(cluster.correl) <- colnames(keys)
    rownames(cluster.correl) <- colnames(keys)
    rownames(cluster.loading) <- rownames(r.mat)
    if (ncol(keys) > 1) {
        cluster.corrected <- correct.cor(cluster.correl, t(key.alpha))
    }
    else {
        cluster.corrected <- cluster.correl
    }
    return(list(loadings = round(cluster.loading, digits), cor = round(cluster.correl,
        digits), corrected = round(cluster.corrected, digits),
        sd = round(sqrt(var), digits), alpha = round(key.alpha,
            digits), size = key.count))
```

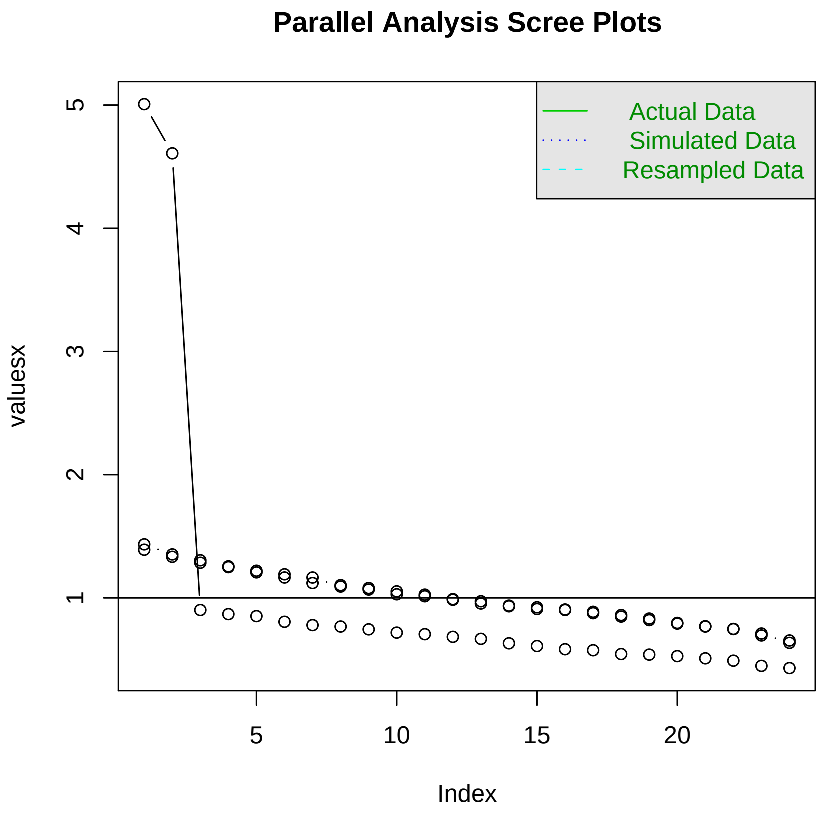

Figure 3: Plots of eigen values for real and simulated "parallel" data give two ways to determine the optimal number of factors: Examine the scree for large breaks or find when the eigen values of random data are greater than for the real data. For this data set, both methods suggest two factors are optimal.

```
}
```

`congeneric.sim` *Simulate a congeneric data set*

Description

Classical Test Theory (CTT) considers four or more tests to be congenerically equivalent if all tests may be expressed in terms of one factor and a residual error. Parallel tests are the special case where (usually two) tests have equal factor loadings. Tau equivalent tests have equal factor loadings but may have unequal errors. Congeneric tests may differ in both factor loading and error variances.

Usage

```
congeneric.sim(N = 1000, loads = c(0.8, 0.7, 0.6, 0.5), err=NULL, short = TRUE)
```

Arguments

`N`	How many subjects to simulate
`loads`	A vector of factor loadings for the tests
`err`	A vector of error variances – if NULL then error = 1 - loading 2
`short`	short=TRUE: Just give the test scores, short=FALSE, report observed test scores as well as the implied pattern matrix

Details

When constructing examples for reliability analysis, it is convenient to simulate congeneric data structures. These are the most simple of item structures, having just one factor.

The implied covariance matrix is just pattern %*% t(pattern).

Value

`observed`	a matrix of test scores for n tests
`pattern`	The pattern matrix implied by the loadings and error variances

Author(s)

William Revelle

References

See Also

`item.sim`

Examples

```
#test <- congeneric.sim()

## The function is currently defined as

function(N = 1000, loads = c(0.8, 0.7, 0.6, 0.5),err=NULL, short=TRUE) {
 n <- length(loads)
loading <- matrix(loads, nrow = n)
error <- diag(1, nrow = n)
if (!is.null(err)) {diag(error) <- err} else {
 diag(error) <- sqrt(1 - loading^2) }
pattern <- cbind(loading, error)
colnames(pattern) <- c("theta", paste("e", seq(1:n), sep = ""))
```

```
A graphNEL graph with directed edges
Number of Nodes = 5
Number of Edges = 4
```

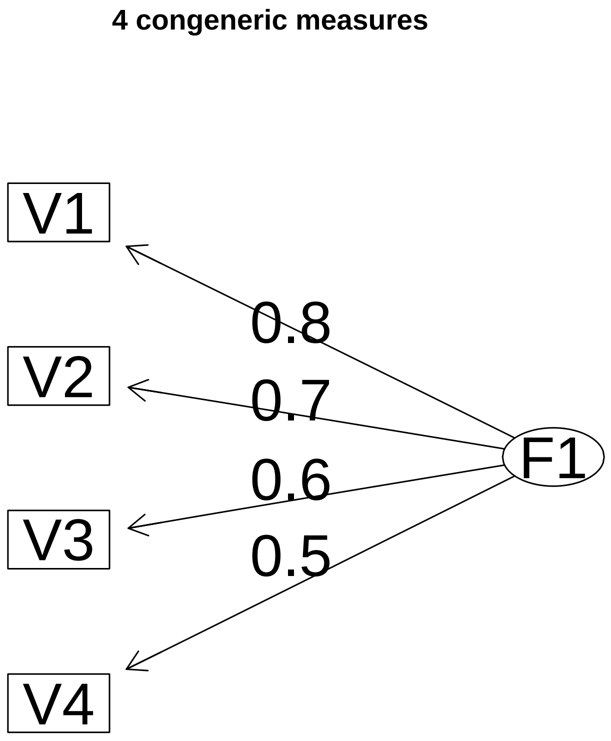

Figure 4: Congeneric tests have one common factor and can differ in their error variances. A demonstration of congeneric.sim, factor.pa, and fa.graph.

```
rownames(pattern) <- c(paste("V", seq(1:n), sep = ""))
latent <- matrix(rnorm(N * (n + 1)), ncol = (n + 1))
observed <- latent
if (short) {return(observed)}  else result <- list(observed=observed,pattern=pattern)
 return(result)
}
```

`correct.cor` *Find dis-attenuated correlations and give alpha reliabilities*

Description

Given a raw correlation matrix and a vector of reliabilities, report the disattenuated correlations above the diagonal.

Usage

```
correct.cor(x, y)
```

Arguments

`x`	A raw correlation matrix
`y`	Vector of reliabilities

Details

Disattenuated correlations may be thought of as correlations between the latent variables measured by a set of observed variables. That is, what would the correlation be between two (unreliable) variables be if both variables were measured perfectly reliably.

Examples of the output of this function are seen in `cluster.loadings` and `cluster.cor`

Value

Raw correlations below the diagonal, reliabilities on the diagonal, disattenuated above the diagonal.

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

http://personality-project.org/revelle/syllabi/405.syllabus.html

See Also

`cluster.loadings` and `cluster.cor`

Examples

```
# attitude from the datasets package
#example 1 is a rather clunky way of doing things
## Not run:
a1 <- attitude[,c(1:3)]
a2 <- attitude[,c(4:7)]
x1 <- rowSums(a1)  #find the sum of the first 3 attitudes
x2 <- rowSums(a2)   #find the sum of the last 4 attitudes
alpha1 <- alpha.scale(x1,a1)
alpha2 <- alpha.scale(x2,a2)
x <- matrix(c(x1,x2),ncol=2)
x.cor <- cor(x)
alpha <- c(alpha1,alpha2)
round(correct.cor(x.cor,alpha),2)
#
#much better - although uses standardized alpha
clusters <- matrix(c(rep(1,3),rep(0,7),rep(1,4)),ncol=2)
cluster.loadings(clusters,cor(attitude))
# or
clusters <- matrix(c(rep(1,3),rep(0,7),rep(1,4)),ncol=2)
cluster.cor(clusters,cor(attitude))
#
## End(Not run)

## The function is currently defined as
"correct.cor" <-
function(x,y) { n=dim(x)[1]
        { diag(x) <- y
        if (n> 1)  {
        for (i in 2:n) {
           k=i-1
           for (j in 1:k) {
              x[j,i] <- x[j,i]/sqrt(y[i]*y[j])  }   #fix the upper triangular part of the matrix
             }}
           return(x)  }}
```

`count.pairwise` *Count number of pairwise cases for a data set with missing (NA) data.*

Description

When doing cor(x, use= ”pairwise”), it is nice to know the number of cases for each pairwise correlation. This is particularly useful when doing SAPA type analyses.

Usage

```
count.pairwise(x, y = NULL)
```

Arguments

x	An input matrix, typically a data matrix ready to be correlated.
y	An optional second input matrix

Value

result = matrix of counts of pairwise observations

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

Examples

```
## Not run:
x <- matrix(rnorm(1000),ncol=6)
y <- matrix(rnorm(500),ncol=3)
x[x < 0] <- NA
y[y> 1] <- NA

count.pairwise(x)
count.pairwise(y)
count.pairwise(x,y)
## End(Not run)

## The function is currently defined as

function (x, y=NULL)
{
    sizex <- dim(x)[2]
    if (length(y)>0)
        {sizey <- dim(y)[2]}  else  {sizey <- dim(x)[2]}
    result <- matrix(1, nrow = sizey, ncol = sizex)
    xnames <- names(x)
    colnames(result) <- names(x)
    if (((is.data.frame(y)) | (is.matrix(y)))) {
        rownames(result) <- names(y) }  else {rownames(result) <- names(x)}
    if (length(y) ==0) {
        for (i in 1:sizex) {
            for (j in 1:(i)) {
                result[j, i] <- sum((!is.na(x[, j])) & (!is.na(x[,
                  i])))
                result[i, j] <- result[j, i]
            }
        }
    } else {
        for (i in 1:sizex) {
            for (j in 1:sizey) {
                result[j, i] <- sum((!is.na(x[, i])) & (!is.na(y[,
                  j])))
            }
        }
    }
    return(result) }
```

describe.by	*Basic summary statistics by group*

Description

Report basic summary statistics by a grouping variable. Useful if the grouping variable is some experimental variable and data are to be aggregated for plotting. Just a wrapper for by and `describe`.

Usage

```
describe.by(x, group,...)
```

Arguments

`x`	a data.frame or matrix
`group`	a grouping variable
`...`	parameters to be passed to describe

Details

Value

A data.frame of the relevant statistics broken down by group:
item name
item number
number of valid cases
mean
standard deviation
median
mad: median absolute deviation (from the median)
minimum
maximum
skew
standard error

Author(s)

William Revelle

See Also

describe

Examples

```
set.seed(42)  #to get the same values each time
 x.df <- data.frame(group=sample(2,20,replace=TRUE), matrix(rnorm(100),ncol=5))
 x <- describe.by(x.df,x.df$group)
 x  #shows all the results
 x[1]  #shows just the first group
 x <- matrix(sample(4,200,replace=TRUE),ncol=5)
 y <- describe.by(x,x[,1])
y
#
#group: 1
#      var n  mean   sd median  mad   min  max range   se
#group   1 7  1.00 0.00   1.00 0.00  1.00 1.00  0.00 0.00
#X1      2 7 -0.36 1.66  -0.31 2.16 -2.66 1.90  4.55 0.63
#X2      3 7 -0.10 1.04  -0.43 0.95 -1.37 1.44  2.81 0.39
#...
#---------------------------------------------------------------------------
#group: 2
#      var  n  mean   sd median  mad   min  max range   se
#group   1 13  2.00 0.00   2.00 0.00  2.00 2.00  0.00 0.00
#X1      2 13 -0.07 1.26  -0.26 0.57 -2.44 2.29  4.73 0.35
#...
#>  x[1]  #shows just the first group
#$`1`
#      var n  mean   sd median  mad   min  max range   se
#group   1 7  1.00 0.00   1.00 0.00  1.00 1.00  0.00 0.00
#X1      2 7 -0.36 1.66  -0.31 2.16 -2.66 1.90  4.55 0.63
#X2      3 7 -0.10 1.04  -0.43 0.95 -1.37 1.44  2.81 0.39
#X3      4 7  0.56 0.90   0.92 0.97 -0.73 1.58  2.30 0.34
#X4      5 7  0.29 0.88   0.46 0.87 -1.19 1.51  2.71 0.33
#X5      6 7  0.23 1.12   0.19 1.27 -1.46 1.85  3.31 0.42

## The function is currently defined as
function (x,group,...) {               #data are x, grouping variable is group
answer <- by(x,group,describe,...)
return(answer)}
```

`describe` *Basic descriptive statistics useful for psychometrics*

Description

There are many summary statistics available in R; this function provides the ones most useful for scale construction and item analysis in classic psychometrics. Range is most useful for the first pass in a data set, to check for coding errors.

Usage

```
describe(x, digits = 2, na.rm = TRUE, skew = TRUE, ranges = TRUE)
```

Arguments

`x`	A data frame or matrix
`digits`	How many significant digits to report
`na.rm`	The default is to delete missing data
`skew`	Should the skew and kurtosis be calculated?
`ranges`	Should the range be calculated?

Details

In basic data analysis it is vital to get basic descriptive statistics. Procedures such as `summary` and hmisc::describe do so. The describe function in the `psych` package is meant to produce the most frequently requested stats in psychometric and psychology studies, and to produce them in an easy to read data.frame. The results from describe can be used in graphics functions (e.g., `error.crosses`).

The range statistics (min, max, range) are most useful for data checking to detect coding errors, and should be found in early analyses of the data.

Although describe will work on data frames as well as matrices, it is important to realize that for data frames, descriptive statistics will be reported only for those variables where this makes sense (i.e., not for factors or for alphanumeric data).

In a typical study, one might read the data in from the clipboard (`read.clipboard`), show the splom plot of the correlations (`pairs.panels`), and then describe the data.

Value

A data.frame of the relevant statistics:
item name
item number
number of valid cases
mean
standard deviation
median
mad: median absolute deviation (from the median)
minimum
maximum
skew
kurtosis
standard error

Note

Describe uses either the mean or colMeans functions depending upon whether the data are a data.frame or a matrix. The mean function supplies means for the columns of a data.frame, but the overall mean for a matrix. Mean will throw a warning for non-numeric data, but colMeans stops with non-numeric data. Thus, the describe function uses either mean (for data frames) or colMeans (for matrices). This is true for skew and kurtosi as well.

Author(s)

http://personality-project.org/revelle.html

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

See Also

describe.by, skew, kurtosi, pairs.panels, read.clipboard, error.crosses

Examples

```
describe(attitude)
#           var  n  mean    sd median   mad min max range  skew kurtosis   se
#rating       1 30 64.63 12.17   65.5 10.38  40  85    45 -0.36    -0.77 2.22
#complaints   2 30 66.60 13.31   65.0 14.83  37  90    53 -0.22    -0.68 2.43
#privileges   3 30 53.13 12.24   51.5 10.38  30  83    53  0.38    -0.41 2.23
#learning     4 30 56.37 11.74   56.5 14.83  34  75    41 -0.05    -1.22 2.14
#raises       5 30 64.63 10.40   63.5 11.12  43  88    45  0.20    -0.60 1.90
#critical     6 30 74.77  9.89   77.5  7.41  49  92    43 -0.87     0.17 1.81
#advance      7 30 42.93 10.29   41.0  8.90  25  72    47  0.85     0.47 1.88

describe(attitude,skew=FALSE)   #attitude is taken from R data sets

#           var  n  mean    sd median   mad min max range   se
#rating       1 30 64.63 12.17   65.5 10.38  40  85    45 2.22
#complaints   2 30 66.60 13.31   65.0 14.83  37  90    53 2.43
#privileges   3 30 53.13 12.24   51.5 10.38  30  83    53 2.23
#learning     4 30 56.37 11.74   56.5 14.83  34  75    41 2.14
#raises       5 30 64.63 10.40   63.5 11.12  43  88    45 1.90
#critical     6 30 74.77  9.89   77.5  7.41  49  92    43 1.81
#advance      7 30 42.93 10.29   41.0  8.90  25  72    47 1.88



## The function is currently defined as

function (x, digits = 2,na.rm=TRUE,skew=TRUE,ranges=TRUE)   #basic stats after dropping non-numeric data
                                                 #slightly faster if we don't do skews
{                          #first, define a local function
    valid <- function(x) {sum(!is.na(x))}

    if (is.vector(x) )  {        #do it for vectors or
            len  <- 1
            stats = matrix(rep(NA,8),ncol=8)    #create a temporary array
                        stats[1, 1] <-  valid(x )
                        stats[1, 2] <-  mean(x, na.rm=na.rm )
                        stats[1, 3] <-  median(x,na.rm=na.rm  )
                        stats[1, 4] <-  min(x, na.rm=na.rm )
                        stats[1, 5] <-  max(x, na.rm=na.rm )
                        stats[1, 6] <-  skew(x,na.rm=na.rm  )
                        stats[1,7] <-  mad(x,na.rm=na.rm)
                        stats[1,8] <-  kurtosi(x,na.rm=na.rm)

        }   else  {
        len = dim(x)[2]     #do it for matrices or data.frames

   stats = matrix(rep(NA,len*8),ncol=8)    #create a temporary array
   stats[,1] <- apply(x,2,valid)
   if(is.matrix(x))  {stats[,2] <- colMeans(x, na.rm=na.rm )} else {stats[,2] <- mean(x,na.rm=na.rm)}
    if (skew) {stats[, 6] <-  skew(x,na.rm=na.rm  )
               stats[,8] <- kurtosi(x,na.rm=na.rm)}

    for (i in 1:len) {
        if (is.numeric(x[,i])) {   #just do this for numeric data

                if (ranges) {
                        stats[i, 3] <-  median(x[,i],na.rm=na.rm  )
                        stats[i,7] <-   mad(x[,i], na.rm=na.rm)
                        stats[i, 4] <-  min(x[,i], na.rm=na.rm )
                        stats[i, 5] <-  max(x[,i], na.rm=na.rm )
                                        } #ranges
                                 }#is.numeric
                }# i loop
        } #else loop
    if (ranges)
        {if(skew){temp <-  data.frame(var = seq(1:len),n = stats[,1],mean=stats[,2], sd = sd(x,na.rm=TRUE
        3],mad = stats[,7], min= stats[,4],max=stats[,5], range=stats[,5]-stats[,4],skew = stats[, 6], ku

         else {temp <-  data.frame(var = seq(1:len),n = stats[,1],mean=stats[,2], sd = sd(x,na.rm=TRUE), 
        3],mad = stats[,7],min= stats[,4],max=stats[,5], range=stats[,5]-stats[,4])}}

        else {if(skew){temp <-  data.frame(var = seq(1:len),n = stats[,1],mean=stats[,2], sd = sd(x,na.rm
       else {temp <-  data.frame(var = seq(1:len),n = stats[,1],mean=stats[,2], sd = sd(x,na.rm=TRUE))}}

    answer <-  round(data.frame(temp, se = temp$sd/sqrt(temp$n)),  digits)
     return(answer)
}
```

`eigen.loadings`	*Convert eigen vectors and eigen values to the more normal (for psychologists) component loadings*

Description

The default procedures for principal component returns values not immediately equivalent to the loadings from a factor analysis. eigen.loadings translates them into the more typical metric of eigen vectors multiplied by the squareroot of the eigenvalues. This lets us find pseudo factor loadings if we have used princomp or princ. If we use `principal` to do our principal components analysis, then we do not need this routine.

Usage

```
eigen.loadings(x)
```

Arguments

`x`	a matrix of loadings from princ or princomp

Value

A matrix of PA loadings more typical for what is expected in psychometrics.

Note

Useful for SAPA analyses

Author(s)

⟨ revelle@northwestern.edu ⟩
`http://personality-project.org/revelle.html`

Examples

```
## The function is currently defined as
"eigen.loadings" <-
function (x) { #convert eigen vectors to loadings by unnormalizing them
               #used if using princomp or princ, not needed for principal
    return(x$vectors %*% sqrt(diag(x$values)))
    }
```

`error.crosses` *Plot x and y error bars*

Description

Given two vectors of data, plot the means and show standard errors in both X and Y directions.

Usage

```
error.crosses(x, y, labels = NULL, pos = NULL, arrow.len = 0.2, ...)
```

Arguments

`x`	A vector of summary statistics (from Describe)
`y`	A second vector of summary statistics (also from Describe)
`labels`	name the pair
`pos`	Labels are located where with respect to the mean?
`arrow.len`	Arrow length
`...`	Other parameters for plot

Details

For an example of two way error bars describing the effects of mood manipulations upon positive and negative affect, see `http://personality-project.org/revelle/publications/happy-sad-appendix/FIG.A-6.pdf`)

Author(s)

William Revelle
⟨revelle@northwestern.edu⟩

Examples

```
#desc <- describe(attitude)
#x <- desc[1,]
#y <- desc[2,]
#plot(x$mean,y$mean)    #in graphics window
#error.crosses(x,y)     #in graphics window

## The function is currently defined as

function (x,y,labels=NULL,pos=NULL,arrow.len=.2,...)  # x  and y are data frame with
    {z <- dim(x)[1]
     if (length(pos)==0) {locate <- rep(1,z)} else {locate <- pos}
     if (length(labels)==0) lab <- rep("",z) else lab <-labels
        for (i in 1:z)
        {xcen <- x$mean[i]
         ycen <- y$mean[i]
         xse  <- x$se[i]
         yse <-  y$se[i]
         arrows(xcen-xse,ycen,xcen+xse,ycen,length=arrow.len, angle = 90,
         code=3,col = par("fg"), lty = NULL, lwd = par("lwd"), xpd = NULL)
         arrows(xcen,ycen-yse,xcen,ycen+yse,length=arrow.len, angle = 90,
         code=3,col = par("fg"), lty = NULL, lwd = par("lwd"), xpd = NULL)
        text(xcen,ycen,labels=lab[i],pos=pos[i],cex=1,offset=arrow.len+1)
        #puts in labels for all points
        }
   }
```

fa.graph	*Graph factor loading matrices*

Description

Factor analysis or principal components analysis results are typically interpreted in terms of the major loadings on each factor. These structures may be represented as a table of loadings or graphically, where all loadings with an absolute value > some cut point are represented as an edge (path).

Usage

```
fa.graph(fa.results, out.file = NULL, labels = NULL, cut = 0.3, simple = TRUE, size = c(8, 6),
```

Arguments

`fa.results`	The output of factor analysis or principal components analysis
`out.file`	If it exists, a dot representation of the graph will be stored here
`labels`	Variable labels
`cut`	Loadings with abs(loading) > cut will be shown
`simple`	Only the biggest loading per item is shown
`size`	
`node.font`	
`edge.font`	
`rank.direction`	
`digits`	Number of digits to show as an edgelable
`title`	Graphic title
`...`	other parameters

Details

Path diagram representations have become standard in confirmatory factor analysis, but are not yet common in exploratory factor analysis. Representing factor structures graphically helps some people understand the structure.

Value

A graph is drawn using rgraphviz. If an output file is specified, the graph instructions are also saved in the dot language.

Note

Requires rgraphviz

Author(s)

William Revelle

See Also

omega.graph, ICLUST.graph

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
# test.simple <- factor.pa(item.sim(16),2)
#fa.graph(test.simple)

## The function is currently defined as
function(fa.results,out.file=NULL,labels=NULL,cut=.3,simple=TRUE,
   size=c(8,6), node.font=c("Helvetica", 14),
    edge.font=c("Helvetica", 10), rank.direction="RL", digits=1,title="Factor Analysis", ...){
    require(Rgraphviz)

factors <- as.matrix(fa.results$loadings)
   rank.direction <- match.arg(rank.direction)
  #first some basic setup parameters

   num.var <- dim(factors)[1]   #how many variables?
   if (is.null(num.var) ){num.var <- length(factors)
       num.factors <- 1} else {
   num.factors <- dim(factors)[2]}
   if (simple) {k=1} else {k <- num.factors}
   vars <- paste("V",1:num.var,sep="")
   fact <- paste("F",1:num.factors,sep="")
   clust.graph <-  new("graphNEL",nodes=c(vars,fact),edgemode="directed")
   graph.shape <- c(rep("box",num.var),rep("ellipse",num.factors))
   graph.rank <- c(rep("sink",num.var),rep("",num.factors))
   names(graph.shape) <- nodes(clust.graph)
   names(graph.rank) <- nodes(clust.graph)
   edge.label <- rep("",num.var*k)
   edge.name <- rep("",num.var*k)
   names(edge.label) <-  seq(1:num.var*k)

  #show the cluster structure with ellipses

  l <- factors
  if (num.factors ==1) {

    for (i in 1:num.var) { clust.graph <- addEdge(fact[1], vars[i], clust.graph,1)
                         edge.label[i] <- round(factors[i],digits)
                         edge.name[i] <- paste(fact[1],"~",vars[i],sep="")
                        }
      } else {
     if(simple){   #very simple structure is one loading per variable
```

```
        m1 <- matrix(apply(t(apply(l, 1, abs)), 1, which.max),
        ncol = 1)
        for (i in 1:num.var) {clust.graph <- addEdge(fact[m1[i]], vars[i], clust.graph,1)
                         edge.label[i] <- round(factors[i,m1[i]],digits)
                         edge.name[i] <- paste(fact[m1[i]],"~",vars[i],sep="")
                        }
                  }   else {   #all loadings > cut in absolute value
                   k <- 1
                   for (i in 1:num.var) {
                   for (f in 1:num.factors) { if (abs(factors[i,f]) > cut) {clust.graph <- addEdge(fact[f
                         edge.label[k] <- round(factors[i,f],digits)
                         edge.name[k] <- paste(fact[f],"~",vars[i],sep="")
                         k <- k+1 }   #end of if
                         }  #end of factor
                        } # end of variable loop
            }  #end of if simple else
       }   #end of if num.factors ==1

 nAttrs <- list()  #node attributes
 eAttrs <- list()  #edge attributes

 if (!is.null(labels)) {var.labels <- c(labels,fact)
  names(var.labels) <-  nodes(clust.graph)
  nAttrs$label <- var.labels
  names(edge.label) <- edge.name
  }
    names(edge.label) <- edge.name
 nAttrs$shape <- graph.shape
 nAttrs$rank <- graph.rank
 eAttrs$label <- edge.label
 attrs <- list(node = list(shape = "ellipse", fixedsize = FALSE),graph=list(rankdir=rank.direction, fonts
 obs.var <- subGraph(vars,clust.graph)
 cluster.vars <- subGraph(fact,clust.graph)
 observed <- list(list(graph=obs.var,cluster=TRUE,attrs=c(rank="")))
 plot(clust.graph, nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attrs,subGList=observed)
if(!is.null(out.file) ){toDot(clust.graph,out.file,nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attrs)
return(clust.graph)
   }
```

`fa.parallel` *Scree plots of data or correlation matrix compared to random "parallel" matrix*

Description

One way to determine the number of factors or components in a data matrix or a correlation matrix is to examine the "scree" plot of the successive eigenvalues. Sharp breaks in the plot suggest the appropriate number of components or factors to extract. "Parallel" analyis is an alternative technique that compares the scree of the observed data with that of a random data matrix of the same size as the original.

Usage

```
fa.parallel(x, ncases = 0, main = "Parallel Analysis Scree Plots")
```

Arguments

`x`	A data.frame or data matrix of scores. If the matrix is square, it is assumed to be a correlation matrix. Otherwise, correlations (with pairwise deletion) will be found
`ncases`	ncases=0 implies a data matrix/data.frame. Otherwise, how many cases were used to find the correlations.
`main`	a title for the analysis

Details

Cattell's "scree" test is one of most simple tests for the number of factors problem. Humphreys and Montanelli's "parallel" analysis is an equally compelling procedure. Other procedures for determining the most optimal number of factors include finding the Very Simple Structure (VSS) criterion (`VSS`).

Value

A plot of the eigen values for the original data, a resampling of the original data, and of a equivalent size matrix of random normal deviates.

Author(s)

William Revelle

References

See Also

`VSS`,`VSS.plot`, `VSS.parallel`

Examples

```
#not run
#test.data <- Harman74.cor$cov
#fa.parallel(test.data,ncases=200)
#
#fa.parallel(attitude)
#

## The function is currently defined as
function(x,ncases=0,main="Parallel Analysis Scree Plots")  {
 nsub <- dim(x)[1]
 nvariables <- dim(x)[2]
 if (ncases >0) { nsub <- ncases
  rx <- x} else {
  rx <- cor(x,use="pairwise")
  }
 if((ncases ==0) && (nsub==nvariables)) {warning("It seems as if you are using a correlation matrix, but I
  if(ncases == 0) {sampledata <- matrix(sample(unlist(x),size=nsub*nvariables,replace=TRUE),nrow=nsub,nco
  values.samp <- eigen(cor(sampledata,use="pairwise"))$values}
  simdata=matrix(rnorm(nsub*nvariables),nrow=nsub,ncol=nvariables)    #make up simulated data
  values.sim <- eigen(cor(simdata))$values

 valuesx  <- eigen(rx)$values

plot(valuesx,type="b", main = main )
points(values.sim,type ="b",lty="dotted")
if(ncases ==0) {points(values.samp,type ="b",lty="dashed")}
legend("topright", c("   Actual Data", "   Simulated Data", "  Resampled Data"), col = c(3,4,5),
       text.col = "green4", lty = c("solid","dotted", "dashed"),
       merge = TRUE, bg = 'gray90')
abline(h=1) }
```

---

factor.congruence *Coefficient of factor congruence*

Description

Given two sets of factor loadings, report their degree of congruence.

Usage

```
factor.congruence(x, y, loading=TRUE)
```

Arguments

x	A matrix of factor loadings
y	A second matrix of factor loadings
loading	Is the input a loading matrix from a factor analysis or principal components, or is it just a matrix

Details

Find the coefficient of factor congruence between two sets of factor loadings.

It is an interesting exercise to compare factor congruences with the correlations of factor loadings. Factor congruences are based upon the raw cross products, while correlations are based upon centered cross products.

Value

A matrix of factor congruences.

Author(s)

⟨revelle@northwestern.edu⟩
http://personality-project.org/revelle.html

References

Gorsuch, Richard, (1983) Factor Analysis. Lawrence Erlebaum Associates.
Revelle, W. (In preparation) An Introduction to Psychometric Theory with applications in R (http://personality-project.org/r/book/)

See Also

principal, factor.pa

Examples

```
#fa <- factanal(x,4,covmat=Harman74.cor$cov)
#pc <- principal(Harman74.cor$cov,4)
#pcv <- varimax(pc$loading)
#round(factor.congruence(fa,pcv),2)
#
#round(factor.congruence(pcv,fa),2)
#    Factor1 Factor2 Factor3 Factor4
#PC1    1.00    0.60    0.45    0.55
#PC2    0.44    0.49    1.00    0.56
#PC3    0.54    0.99    0.44    0.55
#PC4    0.47    0.52    0.48    0.99

#compare with
#round(cor(fa$loading,pcv$loading),2)


## The function is currently defined as
#function (x,y,loading=TRUE) {
# if (loading) {x <- x$loadings
#      y <- y$loadings }

#  nx<- dim(x)[2]
#  ny<- dim(y)[2]
#  cross<- t(y) %*% x   #inner product will have dim of ny * nx
#   sumsx<- sqrt(1/diag(t(x) %*%x))
 #  sumsy<- sqrt(1/diag(t(y) %*%y))

 #  result<- matrix(rep(0,nx*ny),ncol=nx)
#    result<-  sumsy * (cross * rep(sumsx, each = ny))
#  return(t(result))
#   }
```

`factor.fit`	*How well does the factor model fit a correlation matrix. Part of the VSS package*

Description

The basic factor or principal components model is that a correlation or covariance matrix may be reproduced by the product of a factor loading matrix times its transpose: F'F or P'P. One simple index of fit is the 1 - sum squared residuals/sum squared original correlations. This fit index is used by `VSS`, `ICLUST`, etc.

Usage

```
factor.fit(r, f)
```

Arguments

`r`	a correlation matrix
`f`	A factor matrix of loadings.

Details

There are probably as many fit indices as there are psychometricians. This fit is a plausible estimate of the amount of reduction in a correlation matrix given a factor model. Note that it is sensitive to the size of the original correlations. That is, if the residuals are small but the original correlations are small, that is a bad fit.

Value

fit

Author(s)

William Revelle

See Also

`VSS`, `ICLUST`

Examples

```
## Not run:
#compare the fit of 4 to 3 factors for the Harman 24 variables
fa4 <- factanal(x,4,covmat=Harman74.cor$cov)
round(factor.fit(Harman74.cor$cov,fa4$loading),2)
#[1] 0.9
fa3 <- factanal(x,3,covmat=Harman74.cor$cov)
round(factor.fit(Harman74.cor$cov,fa3$loading),2)
#[1] 0.88
```

```
## End(Not run)

## The function is currently defined as
function (r,f) {
     r2 <-sum( r*r)
     rstar <- factor.residuals(r,f)
     rstar2 <- sum(rstar*rstar)
     fit<- 1- rstar2/r2
     return(fit) }
```

`factor.model` *Find R = F F' + U2 is the basic factor model*

Description

The basic factor or principal components model is that a correlation or covariance matrix may be reproduced by the product of a factor loading matrix times its transpose. Find this reproduced matrix. Used by `factor.fit`, `VSS`, `ICLUST`, etc.

Usage

```
factor.model(f)
```

Arguments

`f`	A matrix of loadings.

Details

Value

A correlation or covariance matrix.

Author(s)

⟨revelle@northwestern.edu ⟩
`http://personality-project.org/revelle.html`

References

Gorsuch, Richard, (1983) Factor Analysis. Lawrence Erlebaum Associates.
Revelle, W. In preparation) An Introduction to Psychometric Theory with applications in R (`http://personality-project.org/r/book/`)

See Also

ICLUST.graph,ICLUST.cluster, cluster.fit , VSS, omega

Examples

```
## The function is currently defined as
function(f) {
    result<- f %*% t(f)
    return (result)}
```

---

factor.pa *Principal Axis Factor Analysis*

Description

Among the many ways to do factor analysis, one of the most conventional is principal axes. An eigen value decomposition of a correlation matrix is done and then the communalities for each variable are estimated by the first n factors. These communalities are entered onto the diagonal and the procedure is repeated until the sum(diag(r)) does not vary. For well behaved matrices, maximum likelihood factor analysis (factanal) is probably preferred.

Usage

```
factor.pa(r, nfactors=1, residuals = FALSE, rotate = "varimax", min.err = 0.001, digits = 2, ma
```

Arguments

r	A correlation matrix or a raw data matrix. If raw data, the correlation matrix will be found using pairwise deletion.
nfactors	Number of factors to extract, default is 1
residuals	Should the residual matrix be shown
rotate	”none”, ”varimax”, ”promax”
min.err	Iterate until the change in communalities is less than min.err
digits	How many digits of output should be returned
max.iter	Maximum number of iterations for convergence

Details

Factor analysis is an attempt to approximate a correlation or covariance matrix with one of lesser rank. The basic model is that $nRn \approx nFkkFn'$ where k is much less than n. There are many ways to do factor analysis, and maximum likelihood procedures are probably the most preferred (see factanal).

Principal axes factor analysis has a long history in exploratory analysis and is a straightforward procedure. Successive eigen value decompositions are done on a correlation matrix with the diagonal replaced with diag (FF’) until sum(diag(FF’)) does not change (very

```
A graphNEL graph with directed edges
Number of Nodes = 28
Number of Edges = 35
```

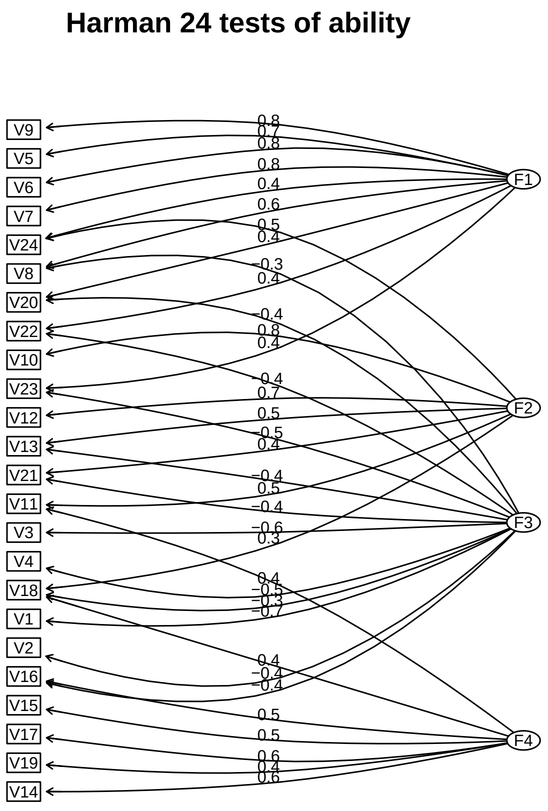

Figure 5: Four factors of the Harman 24 mental tests suggest that a simple structure is difficult to obtain. Note the cross loadings for several items. factor.pa, and fa.graph. Compare with a hierarchical using the omega function (figure 8) or a cluster solution with ICLUST (figure 6)

much). The current limit of max.iter =50 seems to work for most problems, but the Holzinger-Harmon 24 variable problem needs about 203 iterations to converge for a 5 factor solution.

Principal axes may be used in cases when maximum likelihood solutions fail to converge.

Value

If it is a LIST, use

`values`	Eigen values of the final solution
`loadings`	An item by factor loading matrix. Suitable for use in other programs (e.g., GPA rotation or factor2cluster.
`fit`	How well does the factor model reproduce the correlation matrix. (See VSS, ICLUST, and principal for this fit statistic.
`communality`	The history of the communality estimates. Probably only useful for teaching what happens in the process of iterative fitting.

Author(s)

William Revelle

References

Gorsuch, Richard, (1983) Factor Analysis. Lawrence Erlebaum Associates.

See Also

`principal`, `VSS`, `ICLUST`

Examples

```
#using the Harmon 24 mental tests, compare a principal factor with a principal components solution
pc <- principal(Harman74.cor$cov,4,rotate=TRUE)
pa <- factor.pa(Harman74.cor$cov,4,rotate="varimax")
round(factor.congruence(pc,pa),2)

#then compare with a maximum likelihood solution using factanal
mle <- factanal(x,4,covmat=Harman74.cor$cov)
round(factor.congruence(mle,pa),2)
#note that the order of factors and the sign of some of factors differ

## The function is currently defined as
"factor.pa" <-
function(r,nfactors=1,residuals=FALSE,rotate="varimax",min.err = .001,digits=2,max.iter=50) {
    n <- dim(r)[1]
    if (n!=dim(r)[2]) r <- cor(r,use="pairwise") # if given a rectangular matrix, the find the correlatio
    if (!residuals) { result <- list(values=c(rep(0,n)),loadings=matrix(rep(0,n*n),ncol=n),fit=0)} else {

    orig <- diag(r)
    r.mat <- r
    comm <- sum(diag(r.mat))
```

```
 err <- comm
  i <- 1
 comm.list <- list()
 while(err > min.err)    #iteratively replace the diagonal with our revised communality estimate
   {
     eigens <- eigen(r.mat)
      loadings <- eigens$vectors[,1:nfactors]
      model <- loadings \%*\% t(loadings)
      new <- diag(loadings \%*\% t(loadings))
      comm1 <- sum(new)
      diag(r.mat) <- new
      err <- abs(comm-comm1)
      comm <- comm1
      comm.list[[i]] <- comm1
      i <- i + 1
      if(i > max.iter) {warning("maximum iteration exceeded")
             err <-0 }
    }

    #make each vector signed so that the maximum loading is positive
 if (nfactors >1) {
             maxabs <- apply(apply(loadings,2,abs),2,which.max)
                     sign.max <- vector(mode="numeric",length=nfactors)
             for (i in 1: nfactors) {sign.max[i] <- sign(loadings[maxabs[i],i])}
             loadings <- loadings %*% diag(sign.max)

     } else {
             mini <- min(loadings)
                     maxi <- max(loadings)
             if (abs(mini) > maxi) {loadings <- -loadings }
             loadings <- as.matrix(loadings)
     } #sign of largest loading is positive
     colnames(loadings) <- paste("PA",1:nfactors,sep='')
 rownames(loadings) <- rownames(r)

 if(rotate != "none") {if (nfactors ==1) {warning("Must have at least two factors to rotate")} else {
     if (rotate=="varimax") {
                     loadings <- varimax(loadings)$loadings } else {
     if (rotate=="promax") {loadings <- promax(loadings)$loadings }
     }
  }}
 if(nfactors<1) nfactors <- n

 residual<- r - loadings \%*\% t(loadings)
 r2 <- sum(r*r)
 rstar2 <- sum(residual*residual)
 if (residuals) {result$residual <- round(residual,digits)}
r2.off <- r
diag(r2.off) <- 0
r2.off <- sum(r2.off^2)
 diag(residual) <- 0
 rstar.off <- sum(residual^2)
 result$fit <- round(1-rstar2/r2,digits)
```

```
    result$fitoff <- round(1-rstar.off/r2.off,digits)
    result$values <- round(eigens$values,digits)
    result$loadings <- round(loadings,digits)
    result$communality <- round(unlist(comm.list),digits)

    return(result) }
```

`factor.residuals` *R* = R- F F'*

Description

The basic factor or principal components model is that a correlation or covariance matrix may be reproduced by the product of a factor loading matrix times its transpose. Find the residuals of the original minus the reproduced matrix. Used by `factor.fit`, `VSS`, `ICLUST`, etc.

Usage

```
factor.residuals(r, f)
```

Arguments

`r`	A correlation matrix
`f`	A factor model matrix

Details

R* = R - F'F is the basic factor equation.

Value

rstar is the residual correlation matrix.

Author(s)

Maintainer: William Revelle <revelle@northwestern.edu>

Examples

```
## The function is currently defined as
function(r, f) {
   rstar<- r- factor.model(f)
   return(rstar)}
```

`factor.rotate` *"Hand" rotate a factor loading matrix*

Description

Given a factor or components matrix, it is sometimes useful to do arbitrary rotations of particular pairs of variables. This supplements the much more powerful rotation package GRProtation and is meant for specific requirements to do unusual rotations.

Usage

```
factor.rotate(f, angle, col1, col2)
```

Arguments

`f`	original loading matrix
`angle`	angle (in degrees!) to rotate
`col1`	column in factor matrix defining the first variable
`col2`	column in factor matrix defining the second variable

Details

Partly meant as a demonstration of how rotation works, factor.rotate is useful for those cases that require specific rotations that are not available in more advanced packages such as GPAroation.

Value

the resulting rotated matrix of loadings.

Note

For a complete rotation package, see GPArotation

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu ⟩

References

`http://personality-project.org/revelle/syllabi/405.syllabus.html`

Examples

```
## The function is currently defined as
function(f,angle,col1,col2)  {
   #hand rotate two factors from a loading matrix
   #see the GPArotation package for much more elegant procedures
     nvar<- dim(f)[2]
     rot<- matrix(rep(0,nvar*nvar),ncol=nvar)
     rot[cbind(1:nvar, 1:nvar)] <- 1
     theta<- 2*pi*angle/360
     rot[col1,col1]<-cos(theta)
     rot[col2,col2]<-cos(theta)
     rot[col1,col2]<- -sin(theta)
     rot[col2,col1]<- sin(theta)
     result <- f \%*\% rot
     return(result) }
```

`factor2cluster` *Extract cluster definitions from factor loadings*

Description

Given a factor or principal components loading matrix, assign each item to a cluster corresponding to the largest (signed) factor loading for that item. Essentially, this is a Very Simple Structure approach to cluster definition that corresponds to what most people actually do: highlight the largest loading for each item and ignore the rest.

Usage

```
factor2cluster(loads, loading = TRUE, cut = 0)
```

Arguments

`loads`	either a matrix of loadings, or the result of a factor analysis/principal components analyis with a loading component
`loading`	TRUE if taking the output of a FA/PC or ICLUST, FALSE if just a matrix
`cut`	Extract items with absolute loadings > cut

Details

A factor/principal components analysis loading matrix is converted to a cluster (-1,0,1) definition matrix where each item is assigned to one and only one cluster. This is a fast way to extract items that will be unit weighted to form cluster composites. Use this function in combination with cluster.cor to find the corrleations of these composite scores.

A typical use in the SAPA project is to form item composites by clustering or factoring (see `ICLUST`, `principal`), extract the clusters from these results (`factor2cluster`), and then form the composite correlation matrix using `cluster.cor`. The variables in this reduced matrix may then be used in multiple R procedures using mat.regress.

Value

a matrix of -1,0,1 cluster definitions for each item.

Author(s)

http://personality-project.org/revelle.html

Maintainer: William Revelle ⟨ revelle@northwestern.edu ⟩

References

http://personality-project.org/r/r.vss.html

See Also

cluster.cor, factor2cluster ,principal, ICLUST

Examples

```
## Not run:
f  <- factanal(x,4,covmat=Harman74.cor$cov)
factor2cluster(f)
## End(Not run)
#                        Factor1 Factor2 Factor3 Factor4
#VisualPerception              0       1       0       0
#Cubes                         0       1       0       0
#PaperFormBoard                0       1       0       0
#Flags                         0       1       0       0
#GeneralInformation            1       0       0       0
#PargraphComprehension         1       0       0       0
#SentenceCompletion            1       0       0       0
#WordClassification            1       0       0       0
#WordMeaning                   1       0       0       0
#Addition                      0       0       1       0
#Code                          0       0       1       0
#CountingDots                  0       0       1       0
#StraightCurvedCapitals        0       0       1       0
#WordRecognition               0       0       0       1
#NumberRecognition             0       0       0       1
#FigureRecognition             0       0       0       1
#ObjectNumber                  0       0       0       1
#NumberFigure                  0       0       0       1
#FigureWord                    0       0       0       1
#Deduction                     0       1       0       0
#NumericalPuzzles              0       0       1       0
#ProblemReasoning              0       1       0       0
#SeriesCompletion              0       1       0       0
#ArithmeticProblems            0       0       1       0


## The function is currently defined as
```

```
function (loads,loading=TRUE,cut=.0)
{
    if (loading) {l <- loads$loadings
       colnames(l) <- colnames(loads$loading) } else {
       l <- loads
       colnames(l) <- colnames(loads) }
    l <- as.matrix(l)
    nrows <- dim(l)[1]
    ncols <- dim(l)[2]
   if (ncols ==1) {m1 <- matrix(rep(1,nrows),ncol=1) } else {
    m1 <- matrix(apply(t(apply(l, 1, abs)), 1, which.max),
        ncol = 1)}
    id <- matrix(c(1:nrows, m1), ncol = 2)  #index row and column
   factor2cluster <- matrix(rep(0, ncols * nrows), ncol = ncols)
   factor2cluster[id] <- sign(l[id])*( (abs(l[id]) >cut)+0)  #only loadings > cut
  rownames(factor2cluster) <- rownames(l)
  colnames(factor2cluster) <- colnames(l)
   nitems <- colSums(abs(factor2cluster))
   for (i in ncols:1) {if (nitems[i]<1) {factor2cluster <- factor2cluster[,-i]} }#remove columns with no
    return(factor2cluster)
}
```

`fisherz` *Fisher z transform of r*

Description

convert a correlation to a z score using the Fisher transformation.

Usage

```
fisherz(rho)
```

Arguments

`rho`	a Pearson r

Value

z value corresponding to r

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu ⟩

Examples

```
## Not run:
cors <- seq(-.9,.9,.1)
round(fisherz(cors),2)
## End(Not run)

## The function is currently defined as
function(rho)  {0.5*log((1+rho)/(1-rho)) }   #converts r to z
```

`geometric.mean` *Find the geometric mean of a vector or columns of a data.frame.*

Description

The geometric mean is the nth root of n products or e to the mean log of x. Useful for describing non-normal, i.e., geometric distributions.

Usage

```
geometric.mean(x)
```

Arguments

`x`	a vector or data.frame

Details

Useful for teaching how to write functions, also useful for showing the different ways of estimating central tendency.

Value

geometric mean(s) of x or x.df.

Note

Not particularly useful if there are elements that are <= 0.

Author(s)

William Revelle

See Also

`harmonic.mean`, `mean`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
x <- seq(1,5)
x2 <- x^2
geometric.mean(x)
geometric.mean(x2)
## The function is currently defined as
function(x) { exp(mean(log(x))) }
```

`harmonic.mean` *Find the harmonic mean of a vector, matrix, or columns of a data.frame*

Description

The harmonic mean is merely the reciprocal of the arithmetic mean of the reciprocals.

Usage

```
harmonic.mean(x)
```

Arguments

`x`	a vector, matrix, or data.frame

Details

Included as an example for teaching about functions. As well as for a discussion of how to estimate central tendencies.

Value

The harmonic mean(s)

Note

Included as a simple demonstration of how to write a function

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
x <- seq(1,5)
x2 <- x^2
harmonic.mean(x)
harmonic.mean(x2)
## The function is currently defined as
function(x) { 1/(mean(1/x)) }
```

`ICLUST.cluster` *Function to form hierarchical cluster analysis of items*

Description

The guts of the `ICLUST` algorithm. Called by `ICLUST`.

Usage

```
ICLUST.cluster(r.mat, ICLUST.options)
```

Arguments

`r.mat`	A correlation matrix
`ICLUST.options`	A list of options (see `ICLUST`)

Details

See `ICLUST`

Value

A list of cluster statistics, described more fully in `ICLUST`

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Note

Although the main code for ICLUST is here in ICLUST.cluster, the more extensive documentation is for `ICLUST`.

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illinois
⟨ revelle@northwestern.edu ⟩
`http://personality-project.org/revelle.html`

References

Revelle, W. 1979, Hierarchical Cluster Analysis and the Internal Structure of Tests. Multivariate Behavioral Research, 14, 57-74. `http://personality-project.org/revelle/publications/iclust.pdf`
See also more extensive documentation at `http://personality-project.org/r/r.ICLUST.html`

See Also

ICLUST.graph,ICLUST, cluster.fit , VSS, omega

Examples

```
## The function is currently defined as
function (r.mat,ICLUST.options) {#should allow for raw data, correlation or covariances

#options:  alpha =1  (minimum alpha)  2 (average alpha)    3 (maximum alpha)
#          beta =1   (minimum beta)   2 (average beta)     3 (maximum beta)
#          correct  for reliability
#          reverse score items if negative correlations
#          stop clustering if beta for new clusters < beta.min
#          output =1 (short)    2  (show steps)     3 show rejects as we go

#initialize  various arrays and get ready for the first pass
output <- ICLUST.options$output
num.var <- nrow(r.mat)
keep.clustering <- TRUE          #used to determine when we are finished clustering
results <- data.frame(matrix(rep(0,18*(num.var-1)),ncol=18))
names(results) <- c("Item/Cluster", "Item/Cluster","similarity","correlation","alpha1","alpha2","beta1",
"beta2","size1","size2","rbar1","rbar2","r1","r2","alpha","beta","rbar","size")
rownames(results) <- paste("C",1:(num.var-1),sep="")

clusters <- diag(1,nrow =nrow(r.mat))    #original cluster structure is 1 item clusters
rownames(clusters) <- rownames(r.mat)
colnames(clusters) <- paste("V",1:num.var,sep="")
count=1

#master loop

while (keep.clustering) {   #loop until we figure out we should stop
#find similiarities
cluster.stats <- cluster.cor(clusters,r.mat,FALSE)   #we will do most of the work on a copy of the r.mat
sim.mat <- cluster.stats$cor    #the correlation matrix
diag(sim.mat) <- 0   #we don't want 1's on the diagonal to mess up the maximum

#two ways to estimate reliability -- for 1 item clusters, max correlation, for >1, alpha
#this use of initial max should be an option
if (ICLUST.options$correct) {
        row.range <- apply(sim.mat,1,range,na.rm=TRUE)      #find the largest and smallest similarities f
        row.max <- pmax(abs(row.range[1,]),abs(row.range[2,]))  #find the largest absolute similarity
   } else {row.max <- rep(1, nrow(sim)) }         #don't correct for largest similarity

   item.rel <-  cluster.stats$alpha
for (i in 1: length(item.rel)) { if (cluster.stats$size[i]<2) {
        item.rel[i] <- row.max[i]
        #figure out item betas here?
        }}
sq.max <- diag(1/sqrt(item.rel))     #used to correct for reliabilities
```

```
if (ICLUST.options$correct) {sim <- sq.max
     } else {sim <- sim.mat}    #this is the corrected for maximum r similarities
diag(sim) <- NA                  #we need to not consider the diagonal when looking for maxima
#find the most similar pair and apply tests if we should combine
test.alpha <- FALSE
test.beta <- FALSE

while(!(test.alpha&test.beta)){

max.cell <- which.max(sim)             #global maximum
if (length(max.cell) < 1) {
   keep.clustering <- FALSE
   break}   #there are no non-NA values left
sign.max <- 1
if ( ICLUST.options$reverse ) {            #normal case is to reflect if necessary
        min.cell <- which.min(sim)            #location of global minimum
        if (sim[max.cell] <  abs(sim[min.cell] )) {
                sign.max <- -1
                max.cell <- min.cell }
        if (sim[max.cell] < 0.0) {sign.max <- -1 }}  #this is a weird case where all the similarities are

max.col <- trunc(max.cell/nrow(sim))+1    #is in which row and column?
max.row <- max.cell - (max.col-1)*nrow(sim) #need to fix the case of first column
if (max.row < 1) {max.row <- nrow(sim)
 max.col <- max.col-1 }

#combine these two rows if the various criterion are passed
beta.combined <-  2* sign.max*sim.mat[max.cell]/(1+sign.max* sim.mat[max.cell])   #unweighted beta

size1 <- cluster.stats$size[max.row]
if(size1 < 2) {V1 <- 1
        beta1 <-  item.rel[max.row]
        alpha1 <-  item.rel[max.row]
        rbar1 <- item.rel[max.row] } else {
        rbar1 <- results[cluster.names[max.row],"rbar"]
        beta1 <- results[cluster.names[max.row],"beta"]
        alpha1 <- results[cluster.names[max.row],"alpha"]}

 V1 <- size1 + size1*(size1-1) * rbar1

size2 <- cluster.stats$size[max.col]
if(size2 < 2) {V2 <- 1
        beta2 <-  item.rel[max.col]
        alpha2 <-  item.rel[max.col]
        rbar2 <- item.rel[max.col] } else {
        rbar2 <- results[cluster.names[max.col],"rbar"]
        beta2 <- results[cluster.names[max.col],"beta"]
        alpha2 <- results[cluster.names[max.col],"alpha"]}

 V2 <- size2 + size2 * (size2-1) * rbar2

Cov12 <- sign.max* sim.mat[max.cell] * sqrt(V1*V2)
V12 <- V1 + V2 + 2 * Cov12
```

```
size12 <- size1 + size2
alpha <- (V12 - size12)*(size12/(size12-1))/V12
rbar <- alpha/(size12-alpha*(size12-1))

#what is the correlation of this new cluster with the two subclusters?
#this considers item overlap problems
c1 <-  sign.max*rbar1*size1*size1 + sign.max* Cov12   #corrects for item overlap
c2 <-  rbar2*size2*size2 + Cov12     #only flip one of the two correlations with the combined cluster
if(size1 > size2) {r1 <- c1/sqrt(V1*V12)
r2 <- sign.max* c2/sqrt(V2*V12) } else {r1 <-sign.max* c1/sqrt(V1*V12)    #flip the smaller of the two cl
r2 <-  c2/sqrt(V2*V12) }

#test if we should combine these two clusters
#first, does alpha increase?
test.alpha <- TRUE

if (ICLUST.options$alpha.size < min(size1,size2)) {
  switch(ICLUST.options$alpha, {if (alpha < min(alpha1,alpha2)) {if (output>2) {print(paste ('do not combi
  cluster.names[max.row],"with", cluster.names[max.col],'new alpha =', round (alpha,2),
  'old alpha1 =',round( alpha1,2),"old alpha2 =",round(alpha2,2)))}
                                                                                            test.alph
  {if (alpha < mean(alpha1,alpha2)) {if (output>2) {print(paste ('do not combine ', cluster.names[max.row]
  "with", cluster.names[max.col],'new alpha =', round (alpha,2),'old alpha1 =',
  round( alpha1,2),"old alpha2 =",round(alpha2,2)))}
                                                                                            test.alph
  {if (alpha < max(alpha1,alpha2)) {if (output>2) {print(paste ('do not combine ', cluster.names[max.row],
  "with", cluster.names[max.col],'new alpha =', round (alpha,2),'old alpha1 =',
  round( alpha1,2),"old alpha2 =",round(alpha2,2)))}
                                                                                            test.alph
  }   #end if options$alpha.size

#second, does beta increase ?
test.beta <- TRUE

if (ICLUST.options$beta.size < min(size1,size2)) {
  switch(ICLUST.options$beta, {if (beta.combined < min(beta1,beta2)) {if (output>2) {print(
  paste ('do not combine ', cluster.names[max.row],"with", cluster.names[max.col],'new beta =',
  round (beta.combined,2),'old beta1 =',round( beta1,2),"old beta2 =",round(beta2,2)))}
                                                                                            test.beta
  {if (beta.combined < mean(beta1,beta2)) {if (output>2) {print(paste ('do not combine ', cluster.names[ma
  "with", cluster.names[max.col],'new beta =', round (beta.combined,2),'old beta1 =',round( beta1,2),"old b
                                                                                            test.beta
  {if (beta.combined < max(beta1,beta2)) {if (output>2) {print(paste ('do not combine ', cluster.names[max
  "with", cluster.names[max.col],'new beta =', round (beta.combined,2),'old beta1 =',round( beta1,2),"old b
                                                                                            test.beta
  }   #end if options$beta.size




if(test.beta&test.alpha)   {break } else  {
if (beta.combined < ICLUST.options$beta.min) {
                        keep.clustering <- FALSE       #the most similiar pair is not very similar, we sh
                        break}  else {sim[max.row,max.col] <- NA
                sim[max.col,max.row] <- NA }
    }   #end of test.beta&test.alpha

}    #end of while test.alpha&test.beta.loop

#combine and summarize
if (keep.clustering)
   {          # we have based the alpha and beta tests, now combine these two variables
clusters[,max.row] <- clusters[,max.row] + sign.max *  clusters[,max.col]
cluster.names <- colnames(clusters)

#summarize the results
results[count,1] <- cluster.names[max.row]
results[count,2] <- cluster.names[max.col]
results[count,"similarity"] <- sim[max.cell]
results[count,"correlation"] <- sim.mat[max.cell]
results[count,"alpha1"] <- item.rel[max.row]
results[count,"alpha2"] <- item.rel[max.col]
size1 <- cluster.stats$size[max.row]
size2 <- cluster.stats$size[max.col]
results[count,"size1"] <- size1
results[count,"size2"] <- size2
results[count,"beta1"] <-  beta1
results[count,"beta2"] <-  beta2
results[count,"rbar1"] <-  rbar1
results[count,"rbar2"] <-  rbar2
results[count,"r1"] <- r1
results[count,"r2"] <- r2

results[count,"beta"] <- beta.combined

results[count,'alpha'] <- alpha
results[count,'rbar'] <- rbar
results[count,"size"] <- size12
results[count,3:18] <- round(results[count,3:18],ICLUST.options$digits)
#update
cluster.names[max.row] <- paste("C",count,sep="")
colnames(clusters) <- cluster.names
clusters <- clusters[,-max.col]
cluster.names<- colnames(clusters)

#row.max <- row.max[-max.col]


        }  #end of combine section

if(output > 1) print(results[count,],digits=2)
count=count+1
if ((num.var - count) < ICLUST.options$n.clus) {keep.clustering <- FALSE}
if(num.var - count < 1) {keep.clustering <- FALSE}   #only one cluster left

}  #end of keep clustering loop
```

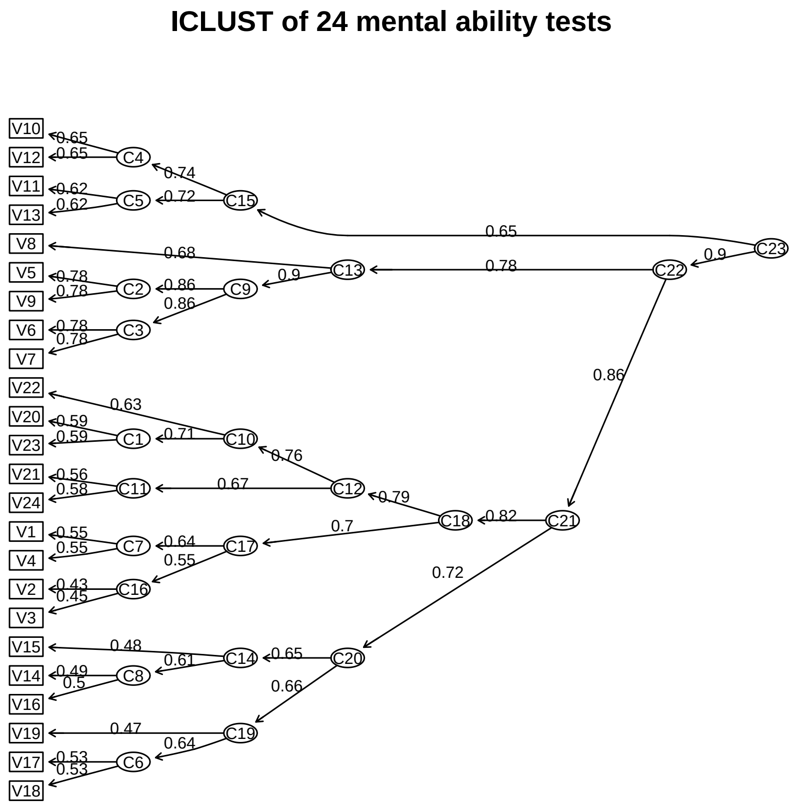

Figure 6: The Harman 24 variables may also be cluster analyzed to highlight the hierarchical structure of abilities. Compare to the hierarchical factor solution using omega Figure 8 or the orthogonal factor solution Figure 5

```
ICLUST.cluster <- list(results=results,clusters=clusters,number <- num.var - count)
}   # end ICLUST.cluster
```

`ICLUST.graph`   *create control code for ICLUST graphical output*

Description

Given a cluster structure determined by `ICLUST`, create dot code to describe the `ICLUST` output. To use the dot code, use either `http://www.graphviz.org/` Graphviz or a commercial viewer (e.g., OmniGraffle).

Usage

```
ICLUST.graph(ic.results, out.file,min.size=1, short = FALSE,labels=NULL,
size = c(8, 6), node.font = c("Helvetica", 14), edge.font = c("Helvetica", 12),
rank.direction = "RL", digits = 2, title = "ICLUST", ...)
```

Arguments

`ic.results`	output list from ICLUST
`out.file`	name of output file (defaults to console)
`min.size`	draw a smaller node (without all the information) for clusters < min.size – useful for large problems
`short`	if short==TRUE, don't use variable names
`labels`	vector of text labels (contents) for the variables
`size`	size of output
`node.font`	Font to use for nodes in the graph
`edge.font`	Font to use for the labels of the arrows (edges)
`rank.direction`	LR or RL
`digits`	number of digits to show
`title`	any title
`...`	other options to pass

Details

Will create (or overwrite) an output file and print out the dot code to show a cluster structure. This dot file may be imported directly into a dot viewer (e.g., http://www.graphviz.org/). The ”dot” language is a powerful graphic description language that is particulary appropriate for viewing cluster output. Commercial graphics programs (e.g., OmniGraffle) can also read (and clean up) dot files.

ICLUST.graph takes the output from `ICLUST` results and processes it to provide a pretty picture of the results. Original variables shown as rectangles and ordered on the left hand side (if rank direction is RL) of the graph. Clusters are drawn as ellipses and include the alpha, beta, and size of the cluster. Edges show the cluster intercorrelations.

It is possible to trim the output to not show all cluster information. Clusters < min.size are shown as small ovals without alpha, beta, and size information.

Value

Output is a set of dot commands written either to console or to the output file. These commands may then be used as input to any ”dot” viewer, e.g., Graphviz.

Author(s)

⟨revelle@northwestern.edu ⟩
http://personality-project.org/revelle.html

References

ICLUST: http://personality-project.org/r/r.iclust.html

See Also

VSS.plot, ICLUST

Examples

```
## Not run:
test.data <- Harman74.cor$cov
ic.out <- ICLUST(test.data)
out.file <- file.choose(new=TRUE)   #create a new file to write the plot commands to
ICLUST.graph(ic.out,out.file)
now go to graphviz (outside of R) and open the out.file you created
print(ic.out,digits=2)
## End(Not run)


#test.data <- Harman74.cor$cov
#my.iclust <- ICLUST(test.data)
#ICLUST.graph(my.iclust)
#
#
#digraph ICLUST {
#  rankdir=RL;
#  size="8,8";
#  node [fontname="Helvetica" fontsize=14 shape=box, width=2];
#  edge [fontname="Helvetica" fontsize=12];
# label = "ICLUST";
#       fontsize=20;
#V1  [label = VisualPerception];
#V2  [label = Cubes];
#V3  [label = PaperFormBoard];
#V4  [label = Flags];
#V5  [label = GeneralInformation];
#V6  [label = PargraphComprehension];
#V7  [label = SentenceCompletion];
#V8  [label = WordClassification];
#V9  [label = WordMeaning];
#V10  [label = Addition];
#V11  [label = Code];
#V12  [label = CountingDots];
#V13  [label = StraightCurvedCapitals];
#V14  [label = WordRecognition];
#V15  [label = NumberRecognition];
#V16  [label = FigureRecognition];
#V17  [label = ObjectNumber];
#V18  [label = NumberFigure];
#V19  [label = FigureWord];
#V20  [label = Deduction];
#V21  [label = NumericalPuzzles];
```

```
#V22  [label = ProblemReasoning];
#V23  [label = SeriesCompletion];
#V24  [label = ArithmeticProblems];
#node [shape=ellipse, width ="1"];
#C1-> V9 [ label = 0.78 ];
#C1-> V5 [ label = 0.78 ];
#C2-> V12 [ label = 0.66 ];
#C2-> V10 [ label = 0.66 ];
#C3-> V18 [ label = 0.53 ];
#C3-> V17 [ label = 0.53 ];
#C4-> V23 [ label = 0.59 ];
#C4-> V20 [ label = 0.59 ];
#C5-> V13 [ label = 0.61 ];
#C5-> V11 [ label = 0.61 ];
#C6-> V7 [ label = 0.78 ];
#C6-> V6 [ label = 0.78 ];
#C7-> V4 [ label = 0.55 ];
#C7-> V1 [ label = 0.55 ];
#C8-> V16 [ label = 0.5 ];
#C8-> V14 [ label = 0.49 ];
#C9-> C1 [ label = 0.86 ];
#C9-> C6 [ label = 0.86 ];
#C10-> C4 [ label = 0.71 ];
#C10-> V22 [ label = 0.62 ];
#C11-> V21 [ label = 0.56 ];
#C11-> V24 [ label = 0.58 ];
#C12-> C10 [ label = 0.76 ];
#C12-> C11 [ label = 0.67 ];
#C13-> C8 [ label = 0.61 ];
#C13-> V15 [ label = 0.49 ];
#C14-> C2 [ label = 0.74 ];
#C14-> C5 [ label = 0.72 ];
#C15-> V3 [ label = 0.48 ];
#C15-> C7 [ label = 0.65 ];
#C16-> V19 [ label = 0.48 ];
#C16-> C3 [ label = 0.64 ];
#C17-> V8 [ label = 0.62 ];
#C17-> C12 [ label = 0.8 ];
#C18-> C17 [ label = 0.82 ];
#C18-> C15 [ label = 0.68 ];
#C19-> C16 [ label = 0.66 ];
#C19-> C13 [ label = 0.65 ];
#C20-> C19 [ label = 0.72 ];
#C20-> C18 [ label = 0.83 ];
#C21-> C20 [ label = 0.87 ];
#C21-> C9 [ label = 0.76 ];
#C22-> 0 [ label = 0 ];
#C22-> 0 [ label = 0 ];
#C23-> 0 [ label = 0 ];
#C23-> 0 [ label = 0 ];
#C1  [label =   "C1\n  alpha= 0.84\n beta=  0.84\nN= 2"] ;
#C2  [label =   "C2\n  alpha= 0.74\n beta=  0.74\nN= 2"] ;
#C3  [label =   "C3\n  alpha= 0.62\n beta=  0.62\nN= 2"] ;
```

```
#C4  [label =   "C4\n  alpha= 0.67\n beta=  0.67\nN= 2"] ;
#C5  [label =   "C5\n  alpha= 0.7\n beta=  0.7\nN= 2"] ;
#C6  [label =   "C6\n  alpha= 0.84\n beta=  0.84\nN= 2"] ;
#C7  [label =   "C7\n  alpha= 0.64\n beta=  0.64\nN= 2"] ;
#C8  [label =   "C8\n  alpha= 0.58\n beta=  0.58\nN= 2"] ;
#C9  [label =   "C9\n  alpha= 0.9\n beta=  0.87\nN= 4"] ;
#C10  [label =   "C10\n  alpha= 0.74\n beta=  0.71\nN= 3"] ;
#C11  [label =   "C11\n  alpha= 0.62\n beta=  0.62\nN= 2"] ;
#C12  [label =   "C12\n  alpha= 0.79\n beta=  0.74\nN= 5"] ;
#C13  [label =   "C13\n  alpha= 0.64\n beta=  0.59\nN= 3"] ;
#C14  [label =   "C14\n  alpha= 0.79\n beta=  0.74\nN= 4"] ;
#C15  [label =   "C15\n  alpha= 0.66\n beta=  0.58\nN= 3"] ;
#C16  [label =   "C16\n  alpha= 0.65\n beta=  0.57\nN= 3"] ;
#C17  [label =   "C17\n  alpha= 0.81\n beta=  0.71\nN= 6"] ;
#C18  [label =   "C18\n  alpha= 0.84\n beta=  0.75\nN= 9"] ;
#C19  [label =   "C19\n  alpha= 0.74\n beta=  0.65\nN= 6"] ;
#C20  [label =   "C20\n  alpha= 0.87\n beta=  0.74\nN= 15"] ;
#C21  [label =   "C21\n  alpha= 0.9\n beta=  0.77\nN= 19"] ;
#C22  [label =   "C22\n  alpha= 0\n beta=  0\nN= 0"] ;
#C23  [label =   "C23\n  alpha= 0\n beta=  0\nN= 0"] ;
#{ rank=same;
#V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12;V13;V14;V15;V16;V17;V18;V19;V20;V21;V22;V23;V24;}}
#
#copy the above output to Graphviz and draw it
#see \url{http://personality-project.org/r/r.ICLUST.html} for an example.

## The function is currently defined as
function(ic.results, out.file,short=FALSE,
   size=c(8,8), node.font=c("Helvetica", 14),
    edge.font=c("Helvetica", 12), rank.direction="RL", digits=2,title="ICLUST", ...){

      if(!missing(out.file)){
        out <- file(out.file, "w")
        on.exit(close(out))
        }
        else out <- stdout()
    results <- ic.results$results
    var.labels <- rownames(ic.results$loadings)
    clusters <- ic.results$clusters
    if(length(clusters)==length(var.labels) ){clusters <- as.matrix(clusters)}
        num <- nrow(results)
    if (short) {var.labels <- paste("V",1:nrow(var.labels),sep="")}

  rank.direction <- match.arg(rank.direction)
  #first some basic setup parameters
  cat( file=out,paste('digraph ICLUST', ' {\n', sep=""))
  cat(file=out, paste('  rankdir=', rank.direction, ';\n', sep=""))
  cat(file=out, paste('  size="',size[1],',',size[2],'";\n', sep=""))
  cat(file=out, paste('  node [fontname="', node.font[1],
        '" fontsize=', node.font[2], ' shape=box, width=2];\n', sep=""))
  cat(file=out, paste('  edge [fontname="', edge.font[1],
        '" fontsize=', edge.font[2], '];\n', sep=""))
  cat(file=out, paste(' label = "' ,title,'";
        fontsize=20;\n', sep=""))

   #create the items as boxes
   #add the sign from the clusters
  num.var <- nrow(results)+1   #how many variables?
     for (i in 1:num.var) { if (max(clusters[i,]) > 0 ) {
        cat(file=out,paste('V',i,'  [label = "',var.labels[i], '"];\n', sep="")) } else {
        cat(file=out,paste('V',i,'  [label = "-',var.labels[i], '"];\n', sep="")) }
         }

  #show the cluster structure with ellipses

  cat(file=out,paste('node [shape=ellipse, width ="1"];\n', sep=""))
  for (i in 1:num) {if(results[i,1]>0) { #avoid printing null results
     cat(file=out,paste(row.names(results)[i],  '-> ', results[i,1],
     ' [ label = ',round(results[i,"r1"],digits),' ];\n', sep=""))
     cat(file=out,paste(row.names(results)[i],  '-> ', results[i,2],
     ' [ label = ',round(results[i,"r2"],digits),' ];\n', sep=""))
     }}

   #label the clusters with alpha and beta
   for (i in 1:num) {if(results[i,1]>0) { #don't print blank results
     cat(file=out,paste(row.names(results)[i],  '  [label =   "',row.names(results)[i],
     '\n  alpha= ',round(results[i,"alpha"],digits),'\n beta=  ' ,
     round(results[i,"beta"],digits),'\nN= ',results[i,"size"], '"] ;\n', sep=""))
     }}

  #keep the boxes all at the same rank (presumably the left side)
  cat(file=out, paste('{ rank=same;\n', sep=""))
  for (i in 1:num.var) { cat(file=out,paste('V',i,';', sep=""))
  }
   cat(file=out, paste('}}', sep=""))   # we are finished
 } # end of ICLUST.graph
```

`ICLUST.rgraph`   *Draw an ICLUST graph using the Rgraphviz package*

Description

Given a cluster structure determined by ICLUST, create a rgraphic directly using Rgraphviz. To create dot code to describe the ICLUST output with more precision, use ICLUST.graph. As an option, dot code is also generated and saved in a file. To use the dot code, use either http://www.graphviz.org/ Graphviz or a commercial viewer (e.g., OmniGraffle).

Usage

```
ICLUST.rgraph(ic.results, out.file = NULL, min.size = 1, short = FALSE, labels = NULL, size = c
```

Arguments

`ic.results`	output list from ICLUST
`out.file`	File name to save optional dot code.
`min.size`	draw a smaller node (without all the information) for clusters < min.size – useful for large problems
`short`	if short==TRUE, don't use variable names
`labels`	vector of text labels (contents) for the variables
`size`	size of output
`node.font`	Font to use for nodes in the graph
`edge.font`	Font to use for the labels of the arrows (edges)
`rank.direction`	LR or RL
`digits`	number of digits to show
`title`	any title
`...`	other options to pass

Details

Will create (or overwrite) an output file and print out the dot code to show a cluster structure. This dot file may be imported directly into a dot viewer (e.g., `http://www.graphviz.org/`). The "dot" language is a powerful graphic description language that is particulary appropriate for viewing cluster output. Commercial graphics programs (e.g., OmniGraffle) can also read (and clean up) dot files.

ICLUST.graph takes the output from `ICLUST` results and processes it to provide a pretty picture of the results. Original variables shown as rectangles and ordered on the left hand side (if rank direction is RL) of the graph. Clusters are drawn as ellipses and include the alpha, beta, and size of the cluster. Edges show the cluster intercorrelations.

It is possible to trim the output to not show all cluster information. Clusters < min.size are shown as small ovals without alpha, beta, and size information.

Value

Output is a set of dot commands written either to console or to the output file. These commands may then be used as input to any "dot" viewer, e.g., Graphviz.

Additional output is drawn to main graphics screen.

Note

Requires Rgraphviz

Author(s)

⟨revelle@northwestern.edu ⟩
`http://personality-project.org/revelle.html`

References

ICLUST: http://personality-project.org/r/r.iclust.html

See Also

VSS.plot, ICLUST

Examples

```
test.data <- Harman74.cor$cov
ic.out <- ICLUST(test.data)
ICLUST.rgraph(ic.out)

## The function is currently defined as
function(ic.results,out.file = NULL, min.size=1,short=FALSE,labels=NULL,
   size=c(8,6), node.font=c("Helvetica", 14),
    edge.font=c("Helvetica", 10), rank.direction="RL", digits=2,title="ICLUST", ...){
    require(Rgraphviz)
   clusters <- as.matrix(ic.results$clusters)
   results <- ic.results$results

   rank.direction <- match.arg(rank.direction)
  #first some basic setup parameters


   #create the items as boxes
   #add the sign from the clusters
   num.var <- dim(clusters)[1]   #how many variables?
   num.clust <- num.var - dim(clusters)[2]

   vars <- paste("V",1:num.var,sep="")
   clust <- paste("C",1:num.clust,sep="")
   clust.graph <-  new("graphNEL",nodes=c(vars,clust),edgemode="directed")
   graph.shape <- c(rep("box",num.var),rep("ellipse",num.clust))
   graph.rank <- c(rep("sink",num.var),rep("",num.clust))
   names(graph.shape) <- nodes(clust.graph)
   names(graph.rank) <- nodes(clust.graph)
   edge.label <- rep("",num.clust*2)
   edge.name <- rep("",num.clust*2)
   names(edge.label) <-  seq(1:num.clust*2)
  #show the cluster structure with ellipses
  for (i in 1:num.clust) {if(results[i,1]>0) { #avoid printing null results
     clust.graph <- addEdge(row.names(results)[i], results[i,1], clust.graph,1)
     edge.label[(i-1)*2+1] <- results[i,"r1"]
     edge.name [(i-1)*2+1]  <- paste(row.names(results)[i],"~", results[i,1],sep="")
     clust.graph <- addEdge(row.names(results)[i], results[i,2], clust.graph,1)
      edge.label[i*2] <- results[i,"r2"]
      edge.name [i*2]  <- paste(row.names(results)[i],"~", results[i,2],sep="")
     }}
 nAttrs <- list()  #node attributes
 eAttrs <- list()  #edge attributes
```

```
if (!is.null(labels)) {var.labels <- c(labels,row.names(results)) #note how this combines variable label
 names(var.labels) <-  nodes(clust.graph)
 nAttrs$label <- var.labels
 names(edge.label) <- edge.name
 }
   names(edge.label) <- edge.name
nAttrs$shape <- graph.shape
nAttrs$rank <- graph.rank
eAttrs$label <- edge.label
attrs <- list(node = list(shape = "ellipse", fixedsize = FALSE),graph=list(rankdir="RL", fontsize=10,bgc
obs.var <- subGraph(vars,clust.graph)
cluster.vars <- subGraph(clust,clust.graph)
observed <- list(list(graph=obs.var,cluster=TRUE,attrs=c(rank="sink")))
plot(clust.graph, nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attrs,subGList=observed)
if (!is.null(out.file)) {toDot(clust.graph,out.file,nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attr
  }
```

`ICLUST.sort` *sort items by absolute size of cluster loadins*

Description

Given a cluster analysis or factor analysis loadings matrix, sort the items by the (absolute) size of each column of loadings. Used as part of ICLUST and SAPA analyses.

Usage

```
ICLUST.sort(ic.load, cut = 0, labels = NULL,loading=TRUE)
```

Arguments

`ic.load`	A loading matrix from a factor or principal components analysis, or from ICLUST.
`cut`	Do not include items in clusters with absolute loadings less than cut
`labels`	labels for each item.
`loading`	if coming from a factor analysis or ICLUST output, use loading=TRUE, but, if just sorting a matrix use loading=FALSE

Details

When interpreting cluster or factor analysis outputs, is is useful to group the items in terms of which items have their biggest loading on each factor/cluster and then to sort the items by size of the absolute factor loading.

A stable cluster solution will be one in which the output of these cluster definitions does not vary when clusters are formed from the clusters so defined.

Value

`cluster`	A matrix of -1, 0, 1s defining each item by the factor/cluster with the row wise largest absolute loading.
`load`	A data.frame of item numbers, item contents, and item x factor loadings.

...

Note

Although part of the ICLUST set of programs, this is generally more useful for factor or principal components analysis.

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illinois
⟨ revelle@northwestern.edu ⟩
http://personality-project.org/revelle.html

References

http://personality-project.org/r/r.ICLUST.html

See Also

ICLUST.graph,ICLUST.cluster, cluster.fit , VSS, factor2cluster

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ic.load,cut=0,labels=NULL,loading=FALSE) {
    if (length(labels)==0) {
    var.labels <- rownames(ic.load)} else {var.labels=labels}
    if(loading) {loadings <- ic.load$loadings} else {loadings <- ic.load}
     nclust <- dim(loadings)[2]
     nitems <- dim(loadings)[1]

  load <- data.frame(item=seq(1:nitems),content=var.labels,cluster=rep(0,nitems),loadings)

  #first find the maximum for each row and assign it to that cluster
   load$cluster <- apply(abs(loadings),1,which.max)
  for (i in 1:nitems) {if (abs(loadings[i,load$cluster[i]]) < cut) {load$cluster[i] <-nclust+1}}

  ord <- sort(load$cluster,index.return=TRUE)
  load[1:nitems,] <- load[ord$ix,]
  rownames(load)[1:nitems] <- rownames(load)[ord$ix]
```

```
 items <- c(table(load$cluster),1)   #how many items are in each cluster?
 #now sort the loadings that have their highest loading on each cluster
  first <- 1
       for (i in 1:nclust) {
       last <- first + items[i]- 1
       ord <- sort(abs(load[first:last,i+3]),decreasing=TRUE,index.return=TRUE)
  load[first:last,] <- load[ord$ix+first-1,]
   rownames(load)[first:last] <- rownames(load)[ord$ix+first-1]
   first <- first + items[i]
   }
   if (first < nitems) load[first:nitems,"cluster"] <- 0   #assign items less than cut to 0
ICLUST.sort <- load }
```

`ICLUST` *ICLUST: Item Cluster Analysis – Hierarchical cluster analysis using psychometric principles*

Description

A common data reduction technique is to cluster cases (subjects). Less common, but particularly useful in psychological research, is to cluster items (variables). This may be thought of as an alternative to factor analysis, based upon a much simpler model. The cluster model is that the correlations between variables reflect that each item loads on at most one cluster, and that items that load on those clusters correlate as a function of their respective loadings on that cluster and items that define different clusters correlate as a function of their respective cluster loadings and the intercluster correlations. Essentially, the cluster model is a Very Simple Structure factor model of complexity one (see `VSS`).

This function applies the ICLUST algorithm to hierarchically cluster items to form composite scales. Clusters are combined if coefficients alpha and beta will increase in the new cluster.

Alpha, the mean split half correlation, and beta, the worst split half correlation, are estimates of the reliability and general factor saturation of the test. (See also the `omega` function to estimate McDonald's coeffient omega.)

Usage

```
ICLUST(r.mat, nclusters=1, alpha=3, beta=1, beta.size=4, alpha.size=3,
correct=TRUE, reverse=TRUE, beta.min=.5, output=1, digits=2,labels=NULL,cut=0,
n.iterations = 0,title="ICLUST")

#ICLUST(r.mat)    #use all defaults
#ICLUST(r.mat,nclusters =3)    #use all defaults and if possible stop at 3 clusters
#ICLUST(r.mat, output =3)     #long output shows clustering history
#ICLUST(r.mat, n.iterations =3)  #clean up solution by item reassignment
```

Arguments

`r.mat`	A correlation matrix or data matrix/data.frame. (If r.mat is not square i.e, a correlation matrix, the data are correlated using pairwise deletion.
`nclusters`	Extract clusters until nclusters remain (default =1)
`alpha`	Apply the increase in alpha criterion (0) never or for (1) the smaller, 2) the average, or 3) the greater of the separate alphas. (default = 3)
`beta`	Apply the increase in beta criterion (0) never or for (1) the smaller, 2) the average, or 3) the greater of the separate betas. (default =1)
`beta.size`	Apply the beta criterion after clusters are of beta.size (default = 4)
`alpha.size`	Apply the alpha criterion after clusters are of size alpha.size (default =3)
`correct`	Correct correlations for reliability (default = TRUE)
`reverse`	Reverse negative keyed items (default = TRUE
`beta.min`	Stop clustering if the beta is not greater than beta.min (default = .5)
`output`	1) short, 2) medium, 3) long output (default =1)
`labels`	vector of item content or labels
`cut`	sort cluster loadings > absolute(cut) (default = 0)
`n.iterations`	
`digits`	Precision of digits of output (default = 2)
`title`	Title for this run

Details

Extensive documentation and justification of the algorithm is available in the original MBR 1979 `http://personality-project.org/revelle/publications/iclust.pdf` paper. Further discussion of the algorithm and sample output is available on the personality-project.org web page: `http://personality-project.org/r/r.ICLUST.html`

The results are best visualized using `ICLUST.graph`, the results of which can be saved as a dot file for the Graphviz program. `http://www.graphviz.org/`

A common problem in the social sciences is to construct scales or composites of items to measure constructs of theoretical interest and practical importance. This process frequently involves administering a battery of items from which those that meet certain criteria are selected. These criteria might be rational, empirical,or factorial. A similar problem is to analyze the adequacy of scales that already have been formed and to decide whether the putative constructs are measured properly. Both of these problems have been discussed in numerous texts, as well as in myriad articles. Proponents of various methods have argued for the importance of face validity, discriminant validity, construct validity, factorial homogeneity, and theoretical importance.

Revelle (1979) proposed that hierachical cluster analysis could be used to estimate a new coefficient (beta) that was an estimate of the general factor saturation of a test. More recently, Zinbarg, Revelle, Yovel and Li (2005) compared McDonald's Omega to Chronbach's alpha and Revelle's beta. They conclude that omega is the best estimate. An algorithm for estimating `omega` is available as part of this package.

This R version is a completely new version of ICLUST. Although early testing suggests it is stable, let me know if you have problems. Please email me if you want help with this version of ICLUST or if you desire more features.

The program currently has three primary functions: cluster, loadings, and graphics.

Clustering 24 tests of mental ability

A sample output using the 24 variable problem by Harman can be represented both graphically and in terms of the cluster order. Note that the graphic is created using GraphViz in the dot language. `ICLUST.graph` produces the dot code for Graphviz. I have also tried doing this in Rgraphviz with less than wonderful results. Dot code can be viewed directly in Graphviz or can be tweaked using commercial software packages (e.g.,OmniGraffle)

Note that for this problem, with these parameters, the data formed one large cluster. (This is consistent with the Very Simple Structure (`VSS`) output as well, which shows a clear one factor solution for complexity 1 data.) See below for an example with this same data set, but with more stringent parameter settings.

To see the graphic output go to `http://personality-project.org/r/r.ICLUST.html` .

Value

`title`	Name of this run
`results`	A list containing
`clusters`	a matrix of -1,0, and 1 values to define cluster membership.
`corrected`	The raw and corrected for alpha reliability cluster intercorrelations.
`purified`	A list of the cluster definitions and cluster loadings of the purified solution

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illinois
⟨ revelle@northwestern.edu ⟩
`http://personality-project.org/revelle.html`

References

Revelle, W. Hierarchical Cluster Analysis and the Internal Structure of Tests. Multivariate Behavioral Research, 1979, 14, 57-74. `http://personality-project.org/revelle/publications/iclust.pdf`
See also more extensive documentation at `http://personality-project.org/r/r.ICLUST.html`

See Also

`ICLUST.graph`,`ICLUST.cluster`, `cluster.fit` , `VSS`, `omega`

Examples

```
## Not run:
test.data <- Harman74.cor$cov
ic.out <- ICLUST(test.data)         #use all defaults
out.file <- file.choose(new=TRUE)   #create a new file to write the plot commands to
ICLUST.graph(ic.out,out.file,title = "ICLUST of Harman's 24 mental variables" )

ic.out <- ICLUST(test.data,nclusters =3)  #use all defaults and if possible stop at 3 clusters
ICLUST.graph(ic.out,out.file,title = "ICLUST of 24 mental variables with forced 3 cluster solution")

ICLUST(test.data, output =3)     #long output shows clustering history

ic.out <- ICLUST(test.data,,nclusters=4, n.iterations =3)  #clean up solution by item reassignment
ICLUST.graph(ic.out,out.file,title = "ICLUST of 24 mental variables with forced 4 cluster solution")
ic.out    #shows the output on the console
## End(Not run)

#produces this output
#ICLUST(Harman74.cor$cov)
#$title
#[1] "ICLUST"
#
#$clusters
#      VisualPerception                  Cubes         PaperFormBoard                  Flags     GeneralI
#                     1                      1                      1                      1
# PargraphComprehension     SentenceCompletion     WordClassification            WordMeaning
#                     1                      1                      1                      1
#                  Code           CountingDots StraightCurvedCapitals        WordRecognition      NumberR
#                     1                      1                      1                      1
#     FigureRecognition           ObjectNumber           NumberFigure             FigureWord
#                     1                      1                      1                      1
#      NumericalPuzzles       ProblemReasoning       SeriesCompletion     ArithmeticProblems
#                     1                      1                      1                      1
#
#$corrected
#     [,1]
#[1,]    1
#
#$loadings
#                        [,1]
#VisualPerception       0.57
#Cubes                  0.36
#PaperFormBoard         0.40
#Flags                  0.46
#GeneralInformation     0.62
#PargraphComprehension  0.62
#SentenceCompletion     0.60
#WordClassification     0.63
#WordMeaning            0.62
#Addition               0.43
#Code                   0.54
#CountingDots           0.44
```

```
#StraightCurvedCapitals 0.57
#WordRecognition        0.41
#NumberRecognition      0.38
#FigureRecognition      0.50
#ObjectNumber           0.45
#NumberFigure           0.51
#FigureWord             0.44
#Deduction              0.59
#NumericalPuzzles       0.58
#ProblemReasoning       0.58
#SeriesCompletion       0.66
#ArithmeticProblems     0.62
#
#$fit
#$fit$clusterfit
#[1] 0.78
#
#$fit$factorfit
#[1] 0.78
#
#
#$results
#    Item/Cluster Item/Cluster similarity correlation alpha1 alpha2 beta1 beta2 size1 size2 rbar1 rbar2
#C1           V23          V20       1.00        0.51   0.51   0.51  0.51  0.51     1     1  0.51  0.51 0
#C2            V9           V5       1.00        0.72   0.72   0.72  0.72  0.72     1     1  0.72  0.72 0
#C3            V7           V6       1.00        0.72   0.72   0.72  0.72  0.72     1     1  0.72  0.72 0
#C4           V12          V10       1.00        0.58   0.58   0.58  0.58  0.58     1     1  0.58  0.58 0
#C5           V13          V11       1.00        0.54   0.54   0.54  0.54  0.54     1     1  0.54  0.54 0
#C6           V18          V17       1.00        0.45   0.45   0.45  0.45  0.45     1     1  0.45  0.45 0
#C7            V4           V1       0.99        0.47   0.47   0.48  0.47  0.48     1     1  0.47  0.48 0
#C8           V16          V14       0.98        0.41   0.43   0.41  0.43  0.41     1     1  0.43  0.41 0
#C9            C2           C3       0.93        0.78   0.84   0.84  0.84  0.84     2     2  0.72  0.72 0
#C10           C1          V22       0.91        0.56   0.67   0.56  0.68  0.56     2     1  0.51  0.56 0
#C11          V21          V24       0.87        0.45   0.51   0.53  0.51  0.53     1     1  0.51  0.53 0
#C12          C10          C11       0.86        0.58   0.74   0.62  0.72  0.62     3     2  0.49  0.45 0
#C13           C9           V8       0.84        0.64   0.90   0.64  0.88  0.64     4     1  0.69  0.64 0
#C14           C8          V15       0.84        0.41   0.58   0.41  0.58  0.41     2     1  0.41  0.41 0
#C15           C5           C4       0.82        0.59   0.70   0.74  0.70  0.73     2     2  0.54  0.58 0
#C16           V3           V2       0.81        0.32   0.41   0.38  0.41  0.38     1     1  0.41  0.38 0
#C17          C16           C7       0.81        0.45   0.48   0.64  0.48  0.64     2     2  0.32  0.47 0
#C18          C12          C17       0.81        0.59   0.79   0.67  0.73  0.62     5     4  0.43  0.34 0
#C19          V19           C6       0.80        0.40   0.40   0.62  0.40  0.62     1     2  0.40  0.45 0
#C20          C19          C14       0.77        0.49   0.64   0.64  0.57  0.58     3     3  0.38  0.37 0
#C21          C18          C20       0.74        0.58   0.83   0.74  0.74  0.66     9     6  0.35  0.32 0
#C22          C21          C13       0.70        0.62   0.86   0.90  0.73  0.78    15     5  0.29  0.64 0
#C23          C22          C15       0.65        0.55   0.90   0.79  0.77  0.74    20     4  0.31  0.49 0
#    beta rbar size
#C1  0.68 0.51    2
#C2  0.84 0.72    2
#C3  0.84 0.72    2
#C4  0.73 0.58    2
#C5  0.70 0.54    2
#C6  0.62 0.45    2
```

```
#C7  0.64 0.47    2
#C8  0.58 0.41    2
#C9  0.88 0.69    4
#C10 0.72 0.49    3
#C11 0.62 0.45    2
#C12 0.73 0.43    5
#C13 0.78 0.64    5
#C14 0.58 0.37    3
#C15 0.74 0.49    4
#C16 0.48 0.32    2
#C17 0.62 0.34    4
#C18 0.74 0.35    9
#C19 0.57 0.38    3
#C20 0.66 0.32    6
#C21 0.73 0.29   15
#C22 0.77 0.31   20
#C23 0.71 0.30   24
#
#$cor
#     [,1]
#[1,]    1
#
#$alpha
#[1] 0.91
#
#$size
#[1] 24
#
#$sorted
#$sorted$sorted
#                        item                content cluster loadings
#SeriesCompletion         23       SeriesCompletion       1     0.66
#WordClassification        8     WordClassification       1     0.63
#GeneralInformation        5     GeneralInformation       1     0.62
#PargraphComprehension     6  PargraphComprehension       1     0.62
#WordMeaning               9            WordMeaning       1     0.62
#ArithmeticProblems       24     ArithmeticProblems       1     0.62
#SentenceCompletion        7     SentenceCompletion       1     0.60
#Deduction                20              Deduction       1     0.59
#NumericalPuzzles         21       NumericalPuzzles       1     0.58
#ProblemReasoning         22       ProblemReasoning       1     0.58
#VisualPerception          1       VisualPerception       1     0.57
#StraightCurvedCapitals   13 StraightCurvedCapitals       1     0.57
#Code                     11                   Code       1     0.54
#NumberFigure             18           NumberFigure       1     0.51
#FigureRecognition        16      FigureRecognition       1     0.50
#Flags                     4                  Flags       1     0.46
#ObjectNumber             17           ObjectNumber       1     0.45
#CountingDots             12           CountingDots       1     0.44
#FigureWord               19             FigureWord       1     0.44
#Addition                 10               Addition       1     0.43
#WordRecognition          14        WordRecognition       1     0.41
#PaperFormBoard            3         PaperFormBoard       1     0.40
```

```
#NumberRecognition        15      NumberRecognition       1     0.38
#Cubes                     2                  Cubes       1     0.36
#
#
#$p.fit
#$p.fit$clusterfit
#[1] 0.78
#
#$p.fit$factorfit
#[1] 0.78
#
#
#$p.sorted
#$p.sorted$sorted
#                        item                content cluster loadings
#SeriesCompletion         23       SeriesCompletion       1     0.66
#WordClassification        8     WordClassification       1     0.63
#GeneralInformation        5     GeneralInformation       1     0.62
#PargraphComprehension     6  PargraphComprehension       1     0.62
#WordMeaning               9            WordMeaning       1     0.62
#ArithmeticProblems       24     ArithmeticProblems       1     0.62
#SentenceCompletion        7     SentenceCompletion       1     0.60
#Deduction                20              Deduction       1     0.59
#NumericalPuzzles         21       NumericalPuzzles       1     0.58
#ProblemReasoning         22       ProblemReasoning       1     0.58
#VisualPerception          1       VisualPerception       1     0.57
#StraightCurvedCapitals   13 StraightCurvedCapitals       1     0.57
#Code                     11                   Code       1     0.54
#NumberFigure             18           NumberFigure       1     0.51
#FigureRecognition        16      FigureRecognition       1     0.50
#Flags                     4                  Flags       1     0.46
#ObjectNumber             17           ObjectNumber       1     0.45
#CountingDots             12           CountingDots       1     0.44
#FigureWord               19             FigureWord       1     0.44
#Addition                 10               Addition       1     0.43
#WordRecognition          14        WordRecognition       1     0.41
#PaperFormBoard            3         PaperFormBoard       1     0.40
#NumberRecognition        15      NumberRecognition       1     0.38
#Cubes                     2                  Cubes       1     0.36
#
#
#$purified
#$purified$cor
#     [,1]
#[1,]    1
#
#$purified$sd
#[1] 13.79
#
#$purified$corrected
#     [,1]
#[1,] 0.91
#
```

```
#$purified$size
#[1] 24
#
#
### The function is currently defined as
function (r.mat,nclusters=1,alpha=3,beta=2,beta.size=4,alpha.size=3,correct=TRUE,reverse=TRUE,beta.min=.5
 #ICLUST.options <- list(n.clus=1,alpha=3,beta=2,beta.size=4,alpha.size=3,correct=TRUE,reverse=TRUE,beta.
        ICLUST.options <- list(n.clus=nclusters,alpha=alpha,beta=beta,beta.size=beta.size,alpha.size=alph
        if(dim(r.mat)[1]!=dim(r.mat)[2]) {r.mat <- cor(r.mat,use="pairwise") }    #cluster correlation ma
        iclust.results <- ICLUST.cluster(r.mat,ICLUST.options)
        load <- cluster.loadings(iclust.results$clusters,r.mat,digits=digits)
        fits <- cluster.fit(r.mat,load$loadings,iclust.results$clusters,digits=digits)
        sorted <- ICLUST.sort(ic.load=load$loadings,labels=labels,cut=cut) #sort the loadings

        #now, iterate the cluster solution to clean it up (if desired)

                clusters <- iclust.results$clusters
                old.clusters <- clusters
                old.fit <- fits$clusterfit
                clusters <- factor2cluster(load$loadings,cut=cut,loading=FALSE)
                load <- cluster.loadings(clusters,r.mat,digits=digits)
                if (n.iterations > 0) {  #it is possible to iterate the solution to perhaps improve it
                for (steps in 1:n.iterations) {   #
                        load <- cluster.loadings(clusters,r.mat,digits=digits)
                        clusters <- factor2cluster(load$loadings,cut=cut,loading=FALSE)
                        if(dim(clusters)[2]!=dim(old.clusters)[2]) {change <- 999
                        load <- cluster.loadings(clusters,r.mat,digits=digits) } else {
                        change <- sum(abs(clusters)-abs(old.clusters)) }  #how many items are changing?
                        fit <- cluster.fit(r.mat,load$loadings,clusters,digits=digits)
                old.clusters <- clusters
                print(paste("iterations ",steps," change in clusters ", change, "current fit " , fit$clus
                if ((abs(change) < 1) | (fit$clusterfit <= old.fit)) {break}    #stop iterating if it get
                old.fit <- fit$cluster.fit
                                        }
                }
        p.fit <- cluster.fit(r.mat,load$loadings,clusters,digits=digits)
        p.sorted <- ICLUST.sort(ic.load=load$loadings,labels=labels,cut=cut)
        purified <- cluster.cor(clusters,r.mat,digits=digits)
        list(title=title,clusters=iclust.results$clusters,corrected=load$corrected,loadings=load$loadings
}
```

irt.item.diff.rasch *Simple function to estimate item difficulties using IRT concepts*

Description

Steps toward a very crude and preliminary IRT program. These two functions estimate item difficulty and discrimination parameters.

Usage

```
irt.item.diff.rasch(items)
irt.discrim(item.diff,theta,items)
```

Arguments

`items`	a matrix of items
`item.diff`	a vector of item difficulties (found by irt.item.diff)
`theta`	ability estimate from irt.person.theta

Details

Item Response Theory (aka ”The new psychometrics”) models individual responses to items with a logistic function and an individual (theta) and item difficulty (diff) parameter.

irt.item.diff.rasch finds item difficulties with the assumption of theta=0 for all subjects and that all items are equally discriminating.

irt.discrim takes those difficulties and theta estimates from `irt.person.rasch` to find item discrimination (beta) parameters.

A far better package with these features is the ltm package. The IRT functions in the psych-package are for pedagogical rather than production purposes. They are believed to be accurate, but are not guaranteed. They do seem to be slightly more robust to missing data structures associated with SAPA data sets than the ltm package.

Value

a vector of item difficulties or item discriminations.

Note

Under development. Not recommended for public consumption.

Author(s)

William Revelle

References

See Also

`irt.person.rasch`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(items) {
 ncases <- nrow(items)
 item.mean <- colMeans(items,na.rm=TRUE)
 item.mean[item.mean<(1/ncases)] <- 1/ncases
 irt.item.diff.rasch <- log((1/item.mean)- 1) }


 "irt.discrim" <-
function(item.diff,theta,items) {
#find the item discrimination parameter  (beta)
#find the item discrimination parameter -- optimized by item.discrim
 irt.item.discrim <- function(x,diff,theta,scores) {
  fit <- -1*(log(scores/(1+exp(x*(diff-theta))) + (1-scores)/(1+exp(x*(theta-diff)))))
  mean(fit,na.rm=TRUE)
  }
 nitems <- length(item.diff)
 discrim <- matrix(NA,nitems,2)
 for (i in 1:nitems) {
    item.fit <- optimize(irt.item.discrim,c(-5,5),diff=item.diff[i],theta=theta,scores = items[,i])
   discrim[i,1] <- item.fit$minimum
   discrim[i,2] <- item.fit$objective}
  irt.discrim <- discrim
 }
```

`irt.1p` *Item Response Theory estimate of theta (ability) using a Rasch (like) model*

Description

Item Response Theory models individual responses to items by estimating individual ability (theta) and item difficulty (diff) parameters. This is an early and crude attempt to capture this modeling procedure.

Usage

```
irt.person.rasch(diff, items)
irt.0p(items,possible=20)
irt.1p(delta,items)
irt.2p(delta,beta,items)
```

Arguments

`diff`	A vector of item difficulties –probably taken from irt.item.diff.rasch
`items`	A matrix of 0,1 items nrows = number of subjects, ncols = number of items
`possible`	Number of items in the scale – used to determine values of all wrong or all right
`delta`	delta is the same as diff and is the item difficulty parameter
`beta`	beta is the item discrimination parameter found in `irt.discrim`

Details

A very preliminary IRT estimation procedure. Given scores xij for ith individual on jth item
Classical Test Theory ignores item difficulty and defines ability as expected score : abilityi = theta(i) = x(i.) A zero parameter model rescales these mean scores from 0 to 1 to a quasi logistic scale ranging from - 4 to 4 This is merely a non-linear transform of the raw data to reflect a logistic mapping.

Basic 1 parameter (Rasch) model considers item difficulties (delta j): p(correct on item j for the ith subject |theta i, deltaj) = 1/(1+exp(deltaj - thetai)) If we have estimates of item difficulty (delta), then we can find theta i by optimization

Two parameter model adds item sensitivity (beta j): p(correct on item j for subject i |thetai, deltaj, betaj) = 1/(1+exp(betaj *(deltaj- theta i))) Estimate delta, beta, and theta to maximize fit of model to data.

The procedure used here is to first find the item difficulties assuming theta = 0 Then find theta given those deltas Then find beta given delta and theta.

This is not an "official" way to do IRT, but is useful for basic item development.

Value

a data.frame with estimated ability (theta) and quality of fit. (for irt.person.rasch)
a data.frame with the raw means, theta0, and the number of items completed

Note

Not recommended for serious use. This code is under development.

Author(s)

William Revelle

References

See Also

`irt.item.diff.rasch`

Examples

```
## The function is currently defined as
function(diff,items) {
#
#
#the basic one parameter model
 irt <- function(x,diff,scores) {
  fit <- -1*(log(scores/(1+exp(diff-x)) + (1-scores)/(1+exp(x-diff))))
rowMeans(fit,na.rm=TRUE)
  }
 #
 diff<- diff
 items <-items
 num <- dim(items)[1]
 fit <- matrix(NA,num,2)
 total <- rowMeans(items,na.rm=TRUE)
 count <- rowSums(!is.na(items))

 for (i in 1:num) {

        if (count[i]>0)  {myfit <- optimize(irt,c(-4,4),diff=diff,scores=items[i,]) #how to do an apply?
                fit[i,1] <- myfit$minimum
                fit[i,2] <- myfit$objective  #fit of optimizing program
                } else {
                fit[i,1] <- NA
                        fit[i,2] <- NA
                        }    #end if else
    }  #end loop
        irt.person.rasch <-data.frame(total,theta=fit[,1],fit=fit[,2],count)}


irt.0p <- function(items,possible=20) {
raw <- rowMeans(items,na.rm=TRUE)
ave <- raw
valid <- rowSums(!is.na(items))
ave[(!is.na(ave))&(ave<.0001)] <- 1/(possible+1)
ave[(!is.na(ave))&(ave > .9999)] <-  (possible)/(possible+1)
theta <- -log((1/ave) -1)
irt.0p <- matrix(c(theta,raw,valid),ncol=3)
colnames(irt.0p ) <- c("theta","raw","valid")
return(irt.0p)
}
```

`item.sim` *Generate simulated data structures for circumplex or simple structure*

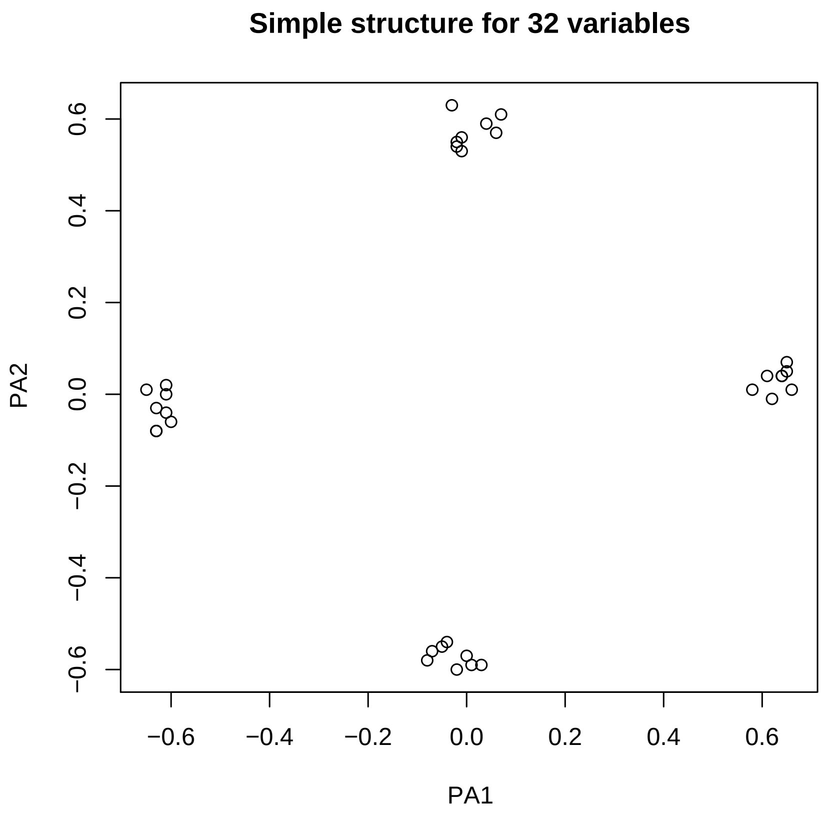

Figure 7: Simple structure is a goal for many factor rotation and extraction procedures. Data may be generated with simple or circumplex structure with varying degrees of skew and correlation. Compare with Figure 1)

Description

Rotations of factor analysis and principal components analysis solutions typically try to represent correlation matrices as simple structured. An alternative structure, appealing to some, is a circumplex structure where the variables are uniformly spaced on the perimeter of a circle in a two dimensional space. Generating these data is straightforward, and is useful for exploring alternative solutions to affect and personality structure.

Usage

```
item.sim(nvar = 72, nsub = 500, circum = FALSE, xloading = 0.6, yloading = 0.6, gloading = 0, x
circ.sim(nvar = 72, nsub = 500, circum = TRUE, xloading = 0.6, yloading = 0.6, gloading = 0, xb
```

Arguments

`nvar`	Number of variables to simulate
`nsub`	Number of subjects to simulate
`circum`	circum=TRUE is circumplex structure, FALSE is simple structure
`xloading`	the average loading on the first dimension
`yloading`	Average loading on the second dimension
`gloading`	Average loading on a general factor (default=0)
`xbias`	To introduce skew, how far off center is the first dimension
`ybias`	To introduce skew on the second dimension
`categorical`	continuous or categorical variables.
`low`	values less than low are forced to low
`high`	values greater than high are forced to high
`truncate`	Change all values less than cutpoint to cutpoint.
`cutpoint`	What is the cutpoint

Details

This simulation was originally developed to compare the effect of skew on the measurement of affect (see Rafaeli and Revelle, 2005). It has been extended to allow for a general simulation of affect or personality items with either a simple structure or a circumplex structure. Items can be continuous normally distributed, or broken down into n categories (e.g, -2, -1, 0, 1, 2). Items can be distorted by limiting them to these ranges, even though the items have a mean of (e.g., 1).

Value

A data matrix of (nsub) subjects by (nvar) variables.

Author(s)

William Revelle

References

Variations of a routine used in Rafaeli and Revelle, 2006; Rafaeli, E. & Revelle, W. (2006). A premature consensus: Are happiness and sadness truly opposite affects? Motivation and Emotion.

Acton, G. S. and Revelle, W. (2004) Evaluation of Ten Psychometric Criteria for Circumplex Structure. Methods of Psychological Research Online, Vol. 9, No. 1 http://www.dgps.de/fachgruppen/methoden/mpr-online/issue22/mpr110_10.pdf

See Also

See Also the implementation in this to generate numerous simulations. circ.simulation, circ.tests

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--     or do  help(data=index)  for the standard data sets.
circ.data <- circ.sim(nvar=8,nsub=200)
round(cor(circ.data),2)

## The function is currently defined as
function (nvar = 72 ,nsub = 500,
    circum = TRUE, xloading =.6, yloading = .6, gloading=0, xbias=0,  ybias = 0,categorical=FALSE, low=-3
        {
        avloading <- (xloading+yloading)/2

        errorweight <- sqrt(1-(avloading^2  + gloading^2))  #squared errors and true score weights add to
    g <- rnorm(nsub)
        truex <- rnorm(nsub)* xloading  +xbias #generate normal true scores for x + xbias
        truey <- rnorm(nsub) * yloading + ybias #generate normal true scores for y + ybias

        if (circum)  #make a vector of radians (the whole way around the circle) if circumplex
        {radia <- seq(0,2*pi,len=nvar+1)
      rad <- radia[which(radia<2*pi)]        #get rid of the last one
     } else rad <- rep(seq(0,3*pi/2,len=4),nvar/4) #simple structure

        error<- matrix(rnorm(nsub*(nvar)),nsub)    #create normal error scores

        #true score matrix for each item reflects structure in radians
        trueitem <- outer(truex, cos(rad)) + outer(truey,sin(rad))

        item<- gloading * g +  trueitem  + errorweight*error   #observed item = true score + error score
    if (categorical) {

        item = round(item)       #round all items to nearest integer value
                item[(item<= low)] <- low
                item[(item>high) ] <- high
                }
        if (truncate) {item[item < cutpoint] <- 0  }
        return (item)
    }
```

kurtosi	*Kurtosis of a vector, matrix, or data frame*

Description

Find the kurtosis of a vector, matrix, or dataframe.

Usage

```
kurtosi(x, na.rm = TRUE)
```

Arguments

`x`	vector, matrix, or data frame
`na.rm`	na.rm =TRUE removes missing data from the column

Details

Kurtosis in the E1071 program finds the kurtosis for a single vector. This does it for matrices and dataframes. Used in the describe function

Value

`kurtosi`	a vector of the kurtosis for each column of the matrix

Note

The mean function supplies means for the columns of a data.frame, but the overall mean for a matrix. Mean will throw a warning for non-numeric data, but colMeans stops with non-numeric data. Thus, the function uses either mean (for data frames) or colMeans (for matrices). This is true for skew and kurtosi as well.

Author(s)

William Revelle

See Also

`skew`, `describe`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
 round(kurtosi(attitude),2)

## The function is currently defined as
function (x, na.rm = TRUE)
{
    if (length(dim(x)) == 0) {
        if (na.rm) {
            x <- x[!is.na(x)]
                                }
       if (is.matrix(x) ) { mx <- colMeans(x,na.rm=na.rm)} else {mx <- mean(x,na.rm=na.rm)}

         sdx <- sd(x,na.rm=na.rm)
        kurt <- sum((x - mx)^4)/(length(x) * sd(x)^4)  -3
        } else {

    kurt <- rep(NA,dim(x)[2])
  #  mx <- mean(x,na.rm=na.rm)
  if (is.matrix(x) ) { mx <- colMeans(x,na.rm=na.rm)} else {mx <- mean(x,na.rm=na.rm)}

    sdx <- sd(x,na.rm=na.rm)
    for (i in 1:dim(x)[2]) {
    kurt[i] <- sum((x[,i] - mx[i])^4,  na.rm = na.rm)/((length(x[,i]) - sum(is.na(x[,i]))) * sdx[i]^4)  -
            }
    }
    return(kurt)
}
```

`make.hierarchical` *Create a population or sample correlation matrix with hierachical structure.*

Description

Create a population hierarchical correlation matrix from a set of factor loadings and factor intercorrelations. Samples of size n may be then be drawn from this population. Return either the sample data, sample correlations, or population correlations. This is used to create sample data sets for instruction and demonstration.

Usage

```
make.hierarchical(gload, fload, n = 0, raw = FALSE)
```

Arguments

`gload`	Loadings of group factors on a general factor
`fload`	Loadings of items on the group factor
`n`	Number of subjects to generate: N=0 => population values
`raw`	raw=TRUE, report the raw data, raw=FALSE, report the sample correlation matrix.

Details

Many personality and cognitive tests have a hierarchical factor structure. For demonstration purposes, it is useful to be able to create such matrices, either with population values, or sample values.

Given a matrix of item factor loadings (fload) and of loadings of these factors on a general factor (gload), we create a population correlation matrix by using the general factor law (R = F' theta F where theta = g'g).

To create sample values, we use the `mvrnorm` function from MASS.

The default is to return population correlation matrices. Sample correlation matrices are generated if n >0. Raw data are returned if raw = TRUE.

Value

a matrix of correlations or a data matrix

Author(s)

William Revelle

References

`http://personality-project.org/r/r.omega.html`

See Also

`omega`, `schmid`, `ICLUST`, `VSS`, `mvrnorm`

Examples

```
## Not run:
gload <-  gload<-matrix(c(.9,.8,.7),nrow=3)    # a higher order factor matrix
fload <-matrix(c(                    #a lower order (oblique) factor matrix
           .8,0,0,
           .7,0,.0,
           .6,0,.0,
            0,.7,.0,
            0,.6,.0,
            0,.5,0,
            0,0,.6,
            0,0,.5,
            0,0,.4),   ncol=3,byrow=TRUE)
```

```
jensen <- make.hierarchical(gload,fload)    #the test set used by omega
round(jensen,2)
## End(Not run)

## The function is currently defined as
function (gload,fload,n=0,raw=FALSE) {
   require(MASS)
  fcor <- gload %*% t(gload)           #the factor correlation matrix
  diag(fcor) <-1                       #put ones on the diagonal
  model <-  fload%*% fcor %*% t(fload) #the model correlation matrix for oblique factors
  diag(model)<- 1                       # put ones along the diagonal
  if(n>0) {
        model <- mvrnorm(n = n,mu, Sigma=model, tol = 1e-6, empirical = FALSE)
        if (!raw ) { model <- cor(model) } }
  make.hierarchical <- model }
```

mat.regress *Multiple Regression from matrix input*

Description

This function extracts subsets of variables (x and y) from a correlation matrix (m) and then find the multiple correlation and beta weights of the (x) set predicting each member of the (y) set.

Usage

```
mat.regress(m, x, y,digits=2)
```

Arguments

`m`	a matrix of correlations
`x`	the column numbers of the x set (e.g., c(1,3,5)
`y`	the column numbers of the y set (e.g., c(2,4,6)
`digits`	round the answer to digits

Details

Although it is more common to calculate multiple regression from raw data, it is, of course, possible to do so from a set of correlations. The input to the function is a square covariance or correlation matrix, as well as the column numbers of the x (predictor) and y (criterion) variables.

The output is a set of multiple correlations, one for each dependent variable in the y set.

A typical use in the SAPA project is to form item composites by clustering or factoring (see `ICLUST`, `principal`), extract the clusters from these results (`factor2cluster`), and then form the composite correlation matrix using `cluster.cor`. The variables in this reduced matrix may then be used in multiple R procedures using mat.regress.

Value

`beta`	the beta weights for each variable in X for each variable in Y
`R2`	The multiple R2 for each equation

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illiniois

Maintainer: William Revelle <revelle@northwestern.edu>

References

For an application of this procedure, see http://personality-project.org/revelle/publications/sapa.pdf

See Also

`cluster.cor`, `factor2cluster`,`principal`,`ICLUST`

Examples

```
## Not run:
test.data <- Harman74.cor$cov     #24 mental variables
#choose 3 of them to regress against another 4 -- arbitrary choice of variables
print(mat.regress(test.data,c(1,2,3),c(4,5,10,12)),digits=2)
## End(Not run)
#gives this output
#print(mat.regress(test.data,c(1,2,3),c(4,5,10,12)),digits=2)
#$beta
#                 Flags GeneralInformation Addition CountingDots
#VisualPerception 0.397              0.22    0.162        0.296
#Cubes            0.064              0.18    0.056        0.049
#PaperFormBoard   0.125              0.10   -0.158        0.005
#
#$R2
#             Flags GeneralInformation           Addition       CountingDots
#             0.239              0.148              0.034              0.101
#

## The function is currently defined as
function(m,x,y)  {
 #a function to extract subsets of variables (a and b) from a correlation matrix m
  #and find the multiple correlation beta weights + R2 of the a set predicting the b set

    #first reorder the matrix to select the right variables
         nm <- dim(m)[1]
        t.mat <- matrix(0,ncol=nm,nrow=nm)
        xy <- c(x,y)
 numx <- length(x)
numy <- length(y)
nxy <- numx+numy
for (i in 1:nxy) {
t.mat[i,xy[i]] <- 1 }

reorder <- t.mat %*% m %*% t(t.mat)
a.matrix <- reorder[1:numx,1:numx]
b.matrix <- reorder[1:numx,(numx+1):nxy]
model.mat <- solve(a.matrix,b.matrix)       #solve the equation bY~aX
if (length(y) >1 ) { rownames(model.mat) <- rownames(m)[x]
 colnames(model.mat) <- colnames(m)[y]

R2 <- colSums(model.mat * b.matrix) }
 else { R2 <- sum(model.mat * b.matrix)
 names(model.mat) <- rownames(m)[x]
 names(R2) <- colnames(m)[y]}
mat.regress <- list(beta=model.mat,R2=R2)
return(mat.regress)
}
```

`matrix.addition` *A function to add two vectors or matrices*

Description

It is sometimes convenient to add two vectors or matrices in an operation analogous to matrix multiplication. For matrices nXm and mYp, the matrix sum of the i,jth element of nSp = sum(over m) of iXm + mYj.

Usage

```
x %+% y
```

Arguments

x	a n by m matrix (or vector if m= 1)
y	a m by p matrix (or vector if m = 1)

Details

Used in such problems as Thurstonian scaling. Although not technically matrix addition, as pointed out by Krus, there are many applications where the sum or difference of two vectors or matrices is a useful operation. This can be done, of course, through

Value

a n by p matix of sums

Author(s)

William Revelle

References

Krus, D. J. (2001) Matrix addition. Journal of Visual Statistics, 1, (February, 2001).

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
x <- seq(1,4)
z <- x %+% -t(x)
x
z
x <- matrix(seq(1,6),ncol=2)
y <- matrix(seq(1,10),nrow=2)
z <- x %+% y
x
y
z
## The function is currently defined as
"%+%" <- function(x,y) {
 if(!is.matrix(x)) {
        if(is.vector(x)) {x <- as.matrix(x)} else stop("x must be either a vector or a matrix")}
if(!is.matrix(y)) {
        if(is.vector(y)) {y <- as.matrix(y)} else stop("y must be either a vector or a matrix")}
n.x <- dim(x)[1]
n.y <- dim(y)[2]
n.k <- dim(x)[2]
if (n.k != dim(y)[1]) {warning("Matrices should be comparable")}
#first find sum vectors
x <- rowSums(x)
y <- colSums(y)
one <- as.vector(rep(1,n.y))  #to duplicate x n.y times
one.y <- as.vector(rep(1,n.x)) #to duplicate y n.x times
xy  <- x
  return(xy) }
```

`multi.hist` *Multiple histograms on one screen*

Description

Given a matrix or data.frame, produce histograms for each variable in a ”matrix” form.

Usage

```
multi.hist(x)
```

Arguments

`x`	matrix or data.frame

Author(s)

William Revelle
Northwestern University
Evanston, Illinois
⟨ revelle@northwestern.edu ⟩
http://personality-project.org/revelle.html

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x) {nvar <- dim(x)[2]  #number of variables
     nsize=trunc(sqrt(nvar))+1   #size of graphic
     old.par <- par(no.readonly = TRUE) # all par settings which can be changed
     par(mfrow=c(nsize,nsize))       #set new graphic parameters
     for (i in 1:nvar) {
     name=names(x)[i]                #get the names for the variables
     hist(x[,i],main=name,xlab=name) }  #draw the histograms for each variable
     on.exit(par(old.par))   #set the graphic parameters back to the original
     }
```

`omega.graph` *Graph hierarchical factor structures*

Description

Hierarchical factor structures represent the correlations between variables in terms of a smaller set of correlated factors which themselves can be represented by a higher order factor.

Two alternative solutions to such structures are found by the omega function. The correlated factors solutions represents the effect of the higher level, general factor, through its effect on the correlated factors. The other representation makes use of the Schmid Leiman transformation to find the direct effect of the general factor upon the original variables as well as the effect of orthogonal residual group factors upon the items.

Graphic presentations of these two alternatives are helpful in understanding the structure. omega.graph draws both such structures. Graphs are drawn directly onto the graphics window or expressed in "dot" commands for conversion to graphics using implementations of Graphviz.

Usage

```
omega.graph(om.results, out.file = NULL,  sl = TRUE, labels = NULL, size = c(8, 6), node.font =
```

Arguments

`om.results`	The output from the omega function
`out.file`	Optional output file for off line analysis using Graphviz
`sl`	Orthogonal clusters using the Schmid-Leiman transform (sl=TRUE) or oblique clusters
`labels`	variable labels
`size`	size of graphics window
`node.font`	What font to use for the items
`edge.font`	What font to use for the edge labels
`rank.direction`	Defaults to left to right
`digits`	Precision of labels
`title`	Figure title
`...`	Other options to pass into the graphics packages

Details

Requires the Rgraphviz package. omega requires the GPArotation package.

Value

`clust.graph`	A graph object

Note

Requires rgraphviz. – omega requires GPARotation

Author(s)

http://personality-project.org/revelle.html
Maintainer: William Revelle ⟨ revelle@northwestern.edu ⟩

References

http://personality-project.org/r/r.omega.html

Revelle, W. (1979). Hierarchical cluster analysis and the internal structure of tests. Multivariate Behavioral Research, 14, 57-74. (http://personality-project.org/revelle/publications/iclust.pdf)

Zinbarg, R.E., Revelle, W., Yovel, I., & Li. W. (2005). Cronbach's Alpha, Revelle's Beta, McDonald's Omega: Their relations with each and two alternative conceptualizations of reliability. Psychometrika. 70, 123-133. http://personality-project.org/revelle/publications/zinbarg.revelle.pmet.05.pdf

Zinbarg, R., Yovel, I., Revelle, W. & McDonald, R. (2006). Estimating generalizability to a universe of indicators that all have one attribute in common: A comparison of estimators for omega. Applied Psychological Measurement, 30, 121-144. DOI: 10.1177/0146621605278814 http://apm.sagepub.com/cgi/reprint/30/2/121

See Also

omega, ICLUST.rgraph

Examples

```
om24 <- omega(Harman74.cor$cov,4)  #run omega
om24pn <- omega.graph(om24,sl=FALSE,title="Harman's 24 tests of mental ability") #show the structure
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(om.results,out.file=NULL,sl=TRUE,labels=NULL,
   size=c(8,6), node.font=c("Helvetica", 14),
    edge.font=c("Helvetica", 10), rank.direction="RL", digits=2,title="Omega", ...){
    require(Rgraphviz)

   if (sl) {factors <- as.matrix(om.results$schmid$sl)   } else{factors <- as.matrix(om.results$schmid$ob
   rank.direction <- match.arg(rank.direction)
  #first some basic setup parameters

   num.var <- dim(factors)[1]   #how many variables?
  if (sl) {num.factors <- dim(factors)[2] -3 } else {num.factors <- dim(factors)[2]}

   vars <- paste("V",1:num.var,sep="")
   fact <- c("g",paste("F",1:num.factors,sep=""))
   clust.graph <-  new("graphNEL",nodes=c(vars,fact),edgemode="directed")
   graph.shape <- c(rep("box",num.var),rep("ellipse",num.factors+1))
   graph.rank <- c(rep("sink",num.var),rep("",num.factors+1))
   names(graph.shape) <- nodes(clust.graph)
   names(graph.rank) <- nodes(clust.graph)
   edge.label <- rep("",num.var*2)
   edge.name <- rep("",num.var*2)
   names(edge.label) <-  seq(1:num.var*2)
  #show the cluster structure with ellipses
   if (sl) {
   l <- matrix(factors[,2:(num.factors+1)],ncol=num.factors) } else { l <- factors }
   m1 <- matrix(apply(t(apply(l, 1, abs)), 1, which.max),
        ncol = 1)

  if (sl) { for (i in 1:num.var) {
     clust.graph <- addEdge(fact[1], vars[i], clust.graph,1) } } else {
        for (i in 1:num.factors) {clust.graph <- addEdge(fact[1], fact[i+1], clust.graph,1) } }
   for (i in 1:num.var) {  clust.graph <- addEdge(fact[1+m1[i]], vars[i], clust.graph,1) }
```

```
 if(FALSE) {
     edge.label[(i-1)*2+1] <- results[i,"r1"]
     edge.name [(i-1)*2+1]  <- paste(row.names(results)[i],"~", results[i,1],sep="")
     clust.graph <- addEdge(row.names(results)[i], results[i,2], clust.graph,1)
      edge.label[i*2] <- results[i,"r2"]
      edge.name [i*2]  <- paste(row.names(results)[i],"~", results[i,2],sep="")
     }

 nAttrs <- list()  #node attributes
 eAttrs <- list()  #edge attributes

 if (!is.null(labels)) {var.labels <- c(labels,fact)
  names(var.labels) <-  nodes(clust.graph)
  nAttrs$label <- var.labels
  names(edge.label) <- edge.name
  }
    names(edge.label) <- edge.name
 nAttrs$shape <- graph.shape
 nAttrs$rank <- graph.rank
 eAttrs$label <- edge.label
 attrs <- list(node = list(shape = "ellipse", fixedsize = FALSE),graph=list(rankdir="RL", fontsize=10,bgc
 obs.var <- subGraph(vars,clust.graph)
 cluster.vars <- subGraph(fact,clust.graph)
 observed <- list(list(graph=obs.var,cluster=TRUE,attrs=c(rank="")))
 plot(clust.graph, nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attrs,subGList=observed)
if(!is.null(out.file) ){toDot(clust.graph,out.file,nodeAttrs = nAttrs, edgeAttrs = eAttrs, attrs = attrs)
return(clust.graph)
   }
```

`omega` *Calculate the omega estimate of factor saturation*

Description

McDonald has proposed coefficient omega as an estimate of the general factor saturation of a test. One way to find omega is to do a factor analysis of the original data set, rotate the factors obliquely, do a Schmid Leiman transformation, and then find omega. This function estimates omega as suggested by McDonald by using hierarchical factor analysis (following Jensen).

Usage

```
omega(m, nfactors, pc = "pa",...)
```

Arguments

`m`	A correlation matrix or a data.frame/matrix of data
`nfactors`	Number of factors believed to be group factors

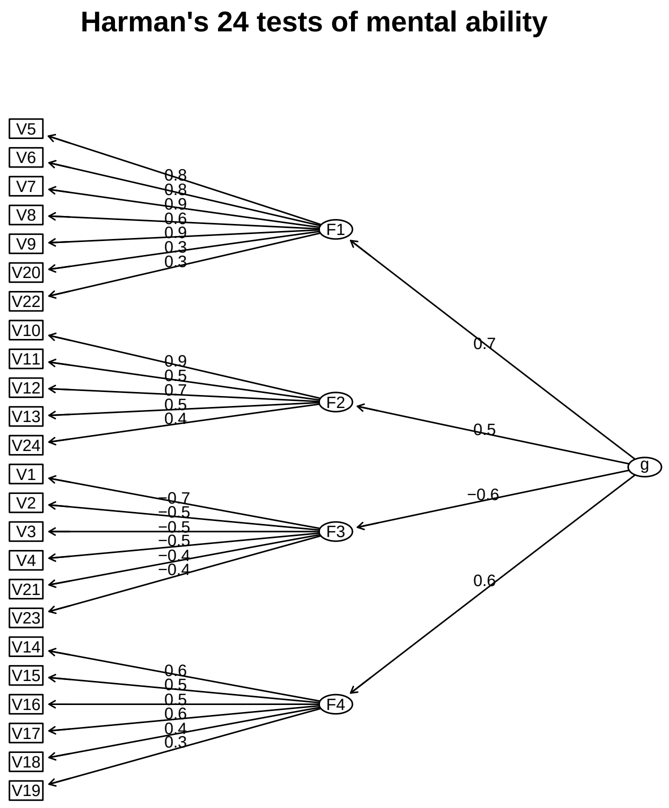

Figure 8: Hierarchical factor solutions are typical in the ability domain where a g factor is thought to reflect the correlations among lower level factors. An alternative transformation is to ortogonalize the g factor from the residual group factors using the Schmid-Leiman transformaton (Figure 9)

Harman's 24 tests of mental ability

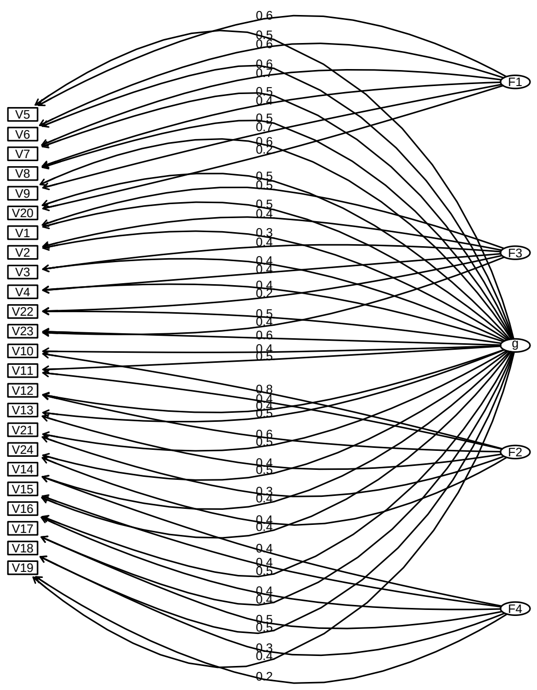

Figure 9: An alternative to the standard hierarchical factor solutions which are typical in the ability domain is to ortogonalize the g factor from the residual group factors using the Schmid-Leiman transformaton. For the hierarchical solution, see Figure 8

`pc`	pc="pa" for principal axes, pc="pc" for principal components, pc="mle" for maximum likelihood .
`...`	Allows additional parameters to be passed through to the factor routines

Details

"Many scales are assumed by their developers and users to be primarily a measure of one latent variable. When it is also assumed that the scale conforms to the effect indicator model of measurement (as is almost always the case in psychological assessment), it is important to support such an interpretation with evidence regarding the internal structure of that scale. In particular, it is important to examine two related properties pertaining to the internal structure of such a scale. The first property relates to whether all the indicators forming the scale measure a latent variable in common.

"The second internal structural property pertains to the proportion of variance in the scale scores (derived from summing or averaging the indicators) accounted for by this latent variable that is common to all the indicators (Cronbach, 1951; McDonald, 1999; Revelle, 1979). That is, if an effect indicator scale is primarily a measure of one latent variable common to all the indicators forming the scale, then that latent variable should account for the majority of the variance in the scale scores. Put differently, this variance ratio provides important information about the sampling fluctuations when estimating individuals' standing on a latent variable common to all the indicators arising from the sampling of indicators (i.e., when dealing with either Type 2 or Type 12 sampling, to use the terminology of Lord, 1956). That is, this variance proportion can be interpreted as the square of the correlation between the scale score and the latent variable common to all the indicators in the infinite universe of indicators of which the scale indicators are a subset. Put yet another way, this variance ratio is important both as reliability and a validity coefficient. This is a reliability issue as the larger this variance ratio is, the more accurately one can predict an individual's relative standing on the latent variable common to all the scale's indicators based on his or her observed scale score. At the same time, this variance ratio also bears on the construct validity of the scale given that construct validity encompasses the internal structure of a scale." (Zinbarg, Yovel, Revelle, and McDonald, 2006).

McDonald has proposed coefficient omega as an estimate of the general factor saturation of a test. Zinbarg, Revelle, Yovel and Li (2005) `http://personality-project.org/revelle/publications/zinbarg.revelle.pmet.05.pdf` compare McDonald's Omega to Cronbach's alpha and Revelle's beta. They conclude that omega is the best estimate. (See also Zinbarg et al., 2006)

One way to find omega is to do a factor analysis of the original data set, rotate the factors obliquely, do a Schmid-Leiman (schmid) transformation, and then find omega. Here we present code to do that.

Omega differs as a function of how the factors are estimated. Three options are available, pc="pa" does a principle axes factor analysis (factor.pa), pc="mle" uses the factanal function, and pc="pc" does a principal components analysis (principal).

Beta, an alternative to omega, is defined as the worst split half reliability. It can be estimated by using ICLUST (a hierarchical clustering algorithm originally developed for main frames and written in Fortran and that is now available in R. (For a very complimentary review of why the ICLUST algorithm is useful in scale construction, see Cooksey and Soutar, 2005).

Value

`alpha`	Cronbach's alpha
`schmid`	The Schmid Leiman transformed factor matrix and associated matrices
`schmid$sl`	The g factor loadings as well as the residualized factors
`schmid$orthog`	Varimax rotated solution of the original factors
`schmid$oblique`	The oblimin transformed factors
`schmid$fcor`	the correlation matrix of the oblique factors
`schid$gloading`	The loadings on the higher order, g, factor of the oblimin factors

Note

Requires the GPArotation package

Author(s)

`http://personality-project.org/revelle.html`
Maintainer: William Revelle ⟨ revelle@northwestern.edu ⟩

References

`http://personality-project.org/r/r.omega.html`

Revelle, W. (1979). Hierarchical cluster analysis and the internal structure of tests. Multivariate Behavioral Research, 14, 57-74. (`http://personality-project.org/revelle/publications/iclust.pdf`)

Zinbarg, R.E., Revelle, W., Yovel, I., & Li. W. (2005). Cronbach's Alpha, Revelle's Beta, McDonald's Omega: Their relations with each and two alternative conceptualizations of reliability. Psychometrika. 70, 123-133. `http://personality-project.org/revelle/publications/zinbarg.revelle.pmet.05.pdf`

Zinbarg, R., Yovel, I., Revelle, W. & McDonald, R. (2006). Estimating generalizability to a universe of indicators that all have one attribute in common: A comparison of estimators for omega. Applied Psychological Measurement, 30, 121-144. DOI: 10.1177/0146621605278814 `http://apm.sagepub.com/cgi/reprint/30/2/121`

See Also

`ICLUST`, `ICLUST.graph`, `VSS`, `schmid`

Examples

```
## Not run:
test.data <- Harman74.cor$cov
my.omega <- omega(test.data,3)
print(my.omega,digits=2)
## End(Not run)
```

```
#produces this output

#$omega
#[1] 0.64
#
#$alpha
#[1] 0.91
#
#$schmid
#$schmid$sl
#                       g factor Factor1 Factor2 Factor3    h2   u2
#VisualPerception           0.53   0.018  0.4688 0.02089 0.494 0.51
#Cubes                      0.34   0.022  0.3029 0.04544 0.209 0.79
#PaperFormBoard             0.38   0.033  0.3971 0.18505 0.398 0.60
#Flags                      0.43   0.109  0.3233 0.06148 0.261 0.74
#GeneralInformation         0.57   0.564  0.0078 0.09900 0.606 0.39
#PargraphComprehension      0.57   0.599  0.0244 0.02960 0.671 0.33
#SentenceCompletion         0.56   0.624  0.0352 0.02783 0.730 0.27
#WordClassification         0.56   0.394  0.1343 0.09255 0.341 0.66
#WordMeaning                0.58   0.637  0.0135 0.06034 0.762 0.24
#Addition                   0.35   0.047  0.0939 0.81706 0.858 0.14
#Code                       0.44   0.061  0.1442 0.44377 0.300 0.70
#CountingDots               0.37   0.100  0.1576 0.57732 0.491 0.51
#StraightCurvedCapitals     0.50   0.036  0.2677 0.33168 0.301 0.70
#WordRecognition            0.34   0.127  0.1506 0.10015 0.093 0.91
#NumberRecognition          0.32   0.060  0.2005 0.07972 0.105 0.90
#FigureRecognition          0.44   0.022  0.4080 0.00192 0.374 0.63
#ObjectNumber               0.37   0.064  0.1769 0.22163 0.139 0.86
#NumberFigure               0.43   0.076  0.3290 0.26170 0.339 0.66
#FigureWord                 0.37   0.053  0.2431 0.10447 0.151 0.85
#Deduction                  0.53   0.231  0.2814 0.00299 0.277 0.72
#NumericalPuzzles           0.50   0.025  0.2877 0.30211 0.301 0.70
#ProblemReasoning           0.52   0.222  0.2840 0.00067 0.272 0.73
#SeriesCompletion           0.59   0.198  0.3304 0.07553 0.325 0.68
#ArithmeticProblems         0.52   0.211  0.1106 0.40982 0.320 0.68
#
#$schmid$orthog
#                       Factor1 Factor2  Factor3
#VisualPerception         0.025   0.702 -0.02336
#Cubes                    0.030   0.454 -0.05080
#PaperFormBoard           0.045   0.595 -0.20689
#Flags                    0.149   0.484 -0.06874
#GeneralInformation       0.771  -0.012  0.11068
#PargraphComprehension    0.817   0.037 -0.03310
#SentenceCompletion       0.852  -0.053  0.03111
#WordClassification       0.538   0.201  0.10348
#WordMeaning              0.870   0.020 -0.06746
#Addition                 0.065  -0.141  0.91350
#Code                     0.083   0.216  0.49615
#CountingDots            -0.136   0.236  0.64546
#StraightCurvedCapitals   0.049   0.401  0.37082
#WordRecognition          0.173   0.226  0.11197
#NumberRecognition        0.082   0.300  0.08912
```

```
#FigureRecognition       -0.029   0.611  0.00214
#ObjectNumber             0.087   0.265  0.24779
#NumberFigure            -0.104   0.493  0.29259
#FigureWord               0.073   0.364  0.11681
#Deduction                0.315   0.421 -0.00334
#NumericalPuzzles         0.035   0.431  0.33777
#ProblemReasoning         0.303   0.425 -0.00074
#SeriesCompletion         0.270   0.495  0.08445
#ArithmeticProblems       0.288   0.166  0.45820
#
#$schmid$fcor
#     [,1] [,2] [,3]
#[1,] 1.00 0.51 0.30
#[2,] 0.51 1.00 0.33
#[3,] 0.30 0.33 1.00
#
#$schmid$gloading
#
#Loadings:
#     Factor1
#[1,] 0.681
#[2,] 0.744
#[3,] 0.447
#
#               Factor1
#SS loadings      1.218
#Proportion Var   0.406
#
#
# The function is currently defined as
function(m,nfactors=3,pc="pa",...) {
      #m is a correlation matrix
      #nfactors is the number of factors to extract
      require(GPArotation)
      nvar <-dim(m)[2]
      gf<-schmid(m,nfactors,pc,...)
      Vt <- sum(m)   #find the total variance in the scale
      Vitem <-sum(diag(m)) #
      gload <- gf$sl[,1]
      gsq <- (sum(gload))^2
      alpha <- ((Vt-Vitem)/Vt)*(nvar/(nvar-1))
      omega <- list(omega= gsq/Vt,alpha=alpha,schmid=gf)
      }
```

`paired.r` *Test the difference between paired correlations*

Description

Test the difference between paired correlations. Given 3 variables, x, y, z, is the correlation between xy different than that between xz? If y and z are independent, this is a simple t-test. But, if they are dependent, it is a bit more complicated.

Usage

```
paired.r(xy, xz, yz, n)
```

Arguments

`xy`	r(xy)
`xz`	r(xz)
`yz`	r(yz)
`n`	Number of subjects

Details

Find a t2 statistic for the difference of two dependent correlations.

Value

t2

Author(s)

William Revelle
Northwestern University
Evanston, Illinois
⟨ revelle@northwestern.edu ⟩
http://personality-project.org/revelle.html

References

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(xy,xz,yz,n) {
       diff <- xy-xz
       determin=1-xy*xy - xz*xz - yz*yz + 2*xy*xz*yz
       av=(xy+xz)/2
       cube= (1-yz)*(1-yz)*(1-yz)
       t2 = diff * sqrt((n-1)*(1+yz)/(((2*(n-1)/(n-3))*determin+av*av*cube)))
       return(t2)
        }
```

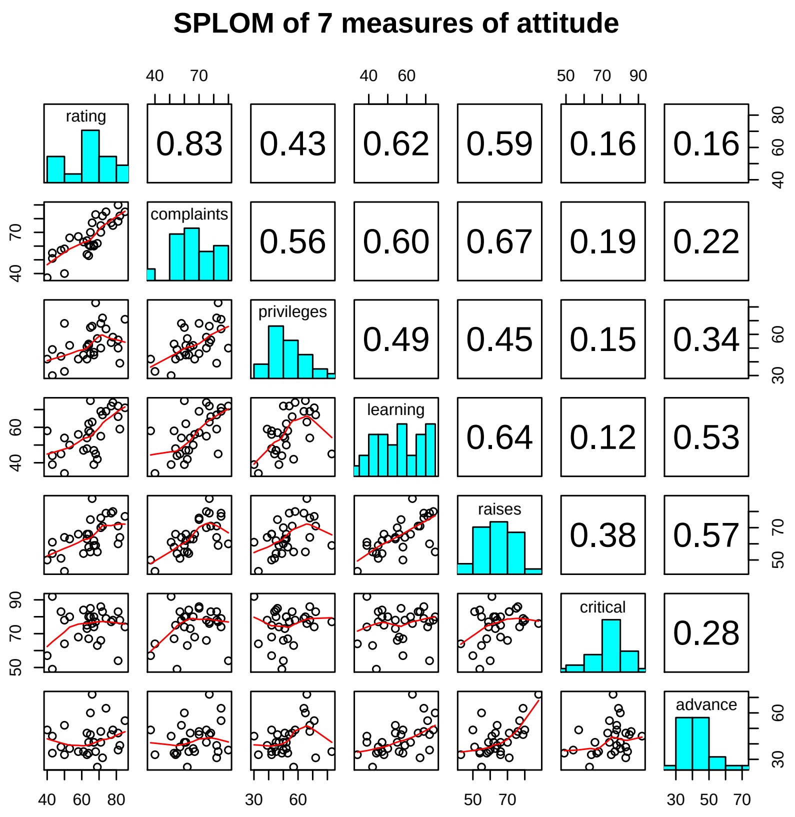

Figure 10: Graphical presentations of correlations (SPLOMS or Scatter Plots of Matrices using the pairs.panels function) allows for quick detection of strange relationships.

`pairs.panels` *SPLOM, histograms and correlations for a data matrix*

Description

Adapted from the help page for pairs, pairs.panels shows a scatter plot of matrices (SPLOM), with bivariate scatter plots below the diagonal, histograms on the diagonal, and the Pearson correlation above the diagonal. Useful for descriptive statistics of small data sets.

Usage

```
pairs.panels(x, y, smooth = TRUE, scale = FALSE, digits = 2, ...)
```

Arguments

`x`	a data.frame or matrix
`y`	an optional data.frame or matrix
`smooth`	TRUE draws loess smooths
`scale`	TRUE scales the correlation font by the size of the absolute correlation.
`digits`	the number of digits to show
`...`	other options for pairs

Details

Shamelessly adapted from the pairs help page. Uses panel.cor, panel.cor.scale, and panel.hist, all taken from the help pages for pairs.

Value

a scatter plot matrix (SPLOM) is drawn in the graphic window. The lower off diagonal draws scatter plots, the diagonal histograms, the upper off diagonal reports the Pearson correlation (with pairwise deletion).

Author(s)

⟨ revelle@northwestern.edu ⟩

See Also

`pairs`

Examples

```
#pairs.panels(attitude)   #see the graphics window

## The function is currently defined as
function (x, y, smooth = TRUE, scale = FALSE, digits = 2, ...)
{
    if (smooth) {
        if (scale) {
            pairs(x, diag.panel = panel.hist, upper.panel = panel.cor.scale,
                lower.panel = panel.smooth, ...)
        }
        else {
            pairs(x, diag.panel = panel.hist, upper.panel = panel.cor,
                lower.panel = panel.smooth, ...)
        }
    }
    else {
        if (scale) {
            pairs(x, diag.panel = panel.hist, upper.panel = panel.cor.scale,
                ...)
        }
        else {
            pairs(x, diag.panel = panel.hist, upper.panel = panel.cor,
                ...)
        }
    }
  }

  #
  ## The function is currently defined as
"panel.cor"
function(x, y, digits=2, prefix="", cex.cor)
     {
         usr <- par("usr"); on.exit(par(usr))
         par(usr = c(0, 1, 0, 1))
         r = (cor(x, y,use="pairwise"))
         txt <- format(c(round(r,digits), 0.123456789), digits=digits)[1]
         txt <- paste(prefix, txt, sep="")
         if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
         text(0.5, 0.5, txt, cex = cex )
     }

 ## The function is currently defined as
 "panel.cor.scale"
function(x, y, digits=2, prefix="", cex.cor)
     {
         usr <- par("usr"); on.exit(par(usr))
         par(usr = c(0, 1, 0, 1))
         r = (cor(x, y,use="pairwise"))
         txt <- format(c(r, 0.123456789), digits=digits)[1]
         txt <- paste(prefix, txt, sep="")
         if(missing(cex.cor)) cex <- 0.8/strwidth(txt)
         text(0.5, 0.5, txt, cex = cex * abs(r))
     }
 "panel.hist"
 function(x, ...)
{
    usr <- par("usr"); on.exit(par(usr))
    par(usr = c(usr[1:2], 0, 1.5) )
    h <- hist(x, plot = FALSE)
    breaks <- h$breaks; nB <- length(breaks)
    y <- h$counts; y <- y/max(y)
    rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
  }
```

`phi` *Find the phi coefficient of correlation between two dichotomous variables*

Description

Given a 1 x 4 vector or a 2 x 2 matrix of frequencies, find the phi coefficient of correlation. Typical use is in the case of predicting a dichotomous criterion from a dichotomous predictor.

Usage

```
phi(t, digits = 2)
```

Arguments

`t`	a 1 x 4 vector or a 2 x 2 matrix
`digits`	round the result to digits

Details

In many prediction situations, a dichotomous predictor (accept/reject) is validated against a dichotomous criterion (success/failure). Although a polychoric correlation estimates the underlying Pearson correlation as if the predictor and criteria were continuous and bivariate normal variables, the phi coefficient is the Pearson applied to a matrix of 0's and 1s.

The calculation follows J. Wiggins discussion of personality assessment.

Value

phi coefficient of correlation

Author(s)

William Revelle with modifications by Leo Gurtler

See Also

phi2poly

Examples

```
phi(c(30,20,20,30))
phi(c(40,10,10,40))
x <- matrix(c(40,5,20,20),ncol=2)
phi(x)

## The function is currently defined as
function(t,digits=2)
{  # expects: t is a 2 x 2 matrix or a vector of length(4)
   stopifnot(prod(dim(t)) == 4 || length(t) == 4)
   if(is.vector(t)) t <- matrix(t, 2)
   r.sum <- rowSums(t)
   c.sum <- colSums(t)
   total <- sum(r.sum)
   r.sum <- r.sum/total
   c.sum <- c.sum/total
   v <- prod(r.sum, c.sum)
   phi <- (t[1,1]/total - c.sum[1]*r.sum[1]) /sqrt(v)
return(round(phi,2))  }
```

`phi2poly` *Convert a phi coefficient to a polychoric correlation*

Description

Given a phi coefficient (a Pearson r calculated on two dichotomous variables), and the marginal frequencies, what is the corresponding estimate of the polychoric correlation?

Usage

```
phi2poly(ph, cp, cc)
```

Arguments

`ph`	phi
`cp`	probability of the predictor – the so called selection ratio
`cc`	probability of the criterion – the so called success rate.

Details

Uses John Fox's polycor function.

Value

a polychoric correlation

Author(s)

William Revelle

See Also

polychor.matrix

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(ph,cp,cc) {
        require(polycor)
     #ph is the phi coefficient
     #cp is the selection ratio of the predictor
     #cc is the success rate of the criterion
     r.marg<-rep(0,2)
     c.marg<- rep(0,2)
     p<-array(rep(0,4),dim=c(2,2))
     r.marg[1]<- cp
     r.marg[2]<- 1 -cp
     c.marg[1]<- cc
     c.marg[2]<- 1-cc

     p[1,1]<- r.marg[1]*c.marg[1]+ ph*sqrt(prod(r.marg,c.marg))
     p[2,2]<- r.marg[2]*c.marg[2]+ ph*sqrt(prod(r.marg,c.marg))
     p[1,2]<- r.marg[1]*c.marg[2]- ph*sqrt(prod(r.marg,c.marg))
     p[2,1]<- r.marg[2]*c.marg[1]- ph*sqrt(prod(r.marg,c.marg))

     result<-polychor(p )
     return(result)}
```

`poly.mat` *Find polychoric correlations of item data*

Description

Uses John Fox's hetcor function (from polychor package) to find a matrix of polychoric correlations for integer data. Essentially a wrapper for hetcor to convert integer item data into factor (categorical) data and then use hetcor. Just a useful shortcut for subsequent factor analysis.

Usage

```
poly.mat(x, short = TRUE, std.err = FALSE, ML = FALSE)
```

Arguments

`x`	A matrix or data frame of integer data
`short`	short=TRUE, just show the correlations, short=FALSE give the full hetcor output
`std.err`	std.err=FALSE does not report the standard errors (faster)
`ML`	ML=FALSE do a quick two step procedure, ML=TRUE, do longer maximum likelihood

Details

Typical personality and item data are integer values (0,1 for ability; 1,2, 3, 4 for attitude scales). The normal correlation procedures will find Pearson correlations (cor). The polycor and hetcor functions from John Fox's polychor package will find polychoric correlations for categorical data. This wrapper function converts integer data to categorical data and then calls hetcor.

Value

A matrix of polychoric correlations (if short=TRUE), otherwise a list of various estimates (see hetcor).

Note

requires polycor

Author(s)

William Revelle

Examples

```
## The function is currently defined as
function(x,short=TRUE,std.err=FALSE,ML=FALSE) {
require(polycor)  #John Fox's Polycor package
xm <- as.matrix(x)
xm.cat <- matrix(as.factor(xm),ncol=dim(xm)[2])
colnames(xm.cat) <- colnames(xm)
r.het <- hetcor(xm.cat,std.err=std.err,ML=ML)
colnames(r.het$correlations) <- colnames(xm)
if(short) {return(r.het$correlations)} else {return(r.het)}
  }
```

`polychor.matrix`	*Actually, what does this do? Convert a matrix of phi coefficients to a matrix of polycoric correlations*

Description

Given a vector of a vector of frequencies, use John Fox's polycor function to convert these to polychoric correlations.

Not ready for public consumption.

Usage

```
polychor.matrix(x, y = NULL)
```

Arguments

`x`	a matrix of phi coefficients
`y`	perhaps

Details

This is a stub of a function to convert a matrix of phi coefficients with an accompanying vector of frequencies into polychoric correlations. Clearly it is not there yet, but is included in the package as a stopgap.

Please do not use.

Value

Describe the value returned If it is a LIST, use

`comp1`	Description of 'comp1'
`comp2`	Description of 'comp2'

...

Author(s)

William Revelle

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(x,y=NULL) {
        require(polycor)
```

```
 sizex <- dim(x)[2]
if (((is.data.frame(y))|(is.matrix(y))))  sizey<-dim(y)[2]
   else  sizey <- dim(x)[2]

 result<-matrix(1,nrow=sizey,ncol=sizex)   #create the output array

 xnames<- names(x)
 colnames(result)<- names(x)

if (((is.data.frame(y))|(is.matrix(y))))  rownames(result) <- names(y)
    else  rownames(result) <- names(x)


if (!((is.data.frame(y))|(is.matrix(y)))) {     #default case returns a square matrix
   for (i in 2: sizex ) {
     for (j in 1:( i-1)) {
      result[j,i]<-polychor(table(x[,j],x[,i]) )
       result[i,j] <- result[j,i]
       }
      }
      }
     else {                   #if y is input, then return the rectangular array
        for (i in 1: sizex ) {
            for (j in 1:sizey) {
                 result[j,i]<-polychor(table(x[,i],y[,j]) )
                     }
                } }
   return (result) }
```

`principal` *Principal components analysis*

Description

Does an eigen value decomposition and returns eigen values, loadings, and degree of fit for a specified number of components. Basically just is doing a principal components for n principal components. Can show the residual correlations as well. The quality of reduction in the squared correlations is reported by comparing residual correlations to original correlations. Unlike princomp, this returns a subset of just the best nfactors. The eigen vectors are rescaled by the sqrt of the eigen values to produce the component loadings more typical in factor analysis.

Usage

```
principal(r, nfactors = 0, residuals = FALSE,rotate=FALSE, digits=2)
```

Arguments

`r`	a correlation matrix

`nfactors`	Number of components to extract
`residuals`	FALSE, do not show residuals, TRUE, report residuals
`rotate`	
`digits`	digits =2 Accuracy of answers as well as display

Details

Useful for those cases where the correlation matrix is improper (perhaps because of SAPA techniques).

Value

`values`	Eigen Values of all components – useful for a scree plot
`loadings`	A standard loading matrix
`fit`	Fit of the model to the correlation matrix
`residual`	Residual matrix – if requested

Author(s)

William Revelle

See Also

`VSS`,`factor2cluster`,`factor.pa`, `factor.congruence`

Examples

```
#Four principal components of the Harmon 24 variable problem
#compare to a four factor principal axes solution using factor.congruence
pc <- principal(Harman74.cor$cov,4,rotate=TRUE)
pa <- factor.pa(Harman74.cor$cov,4,rotate=TRUE)
round(factor.congruence(pc,pa),2)

## The function is currently defined as
function(r,nfactors=0,residuals=FALSE,rotate=FALSE,digits=2) {
   n <- dim(r)[1]

   if (n!=dim(r)[2]) r <- cor(r,use="pairwise") # if given a rectangular matrix, the find the correlation

    if (!residuals) { result <- list(values=c(rep(0,n)),loadings=matrix(rep(0,n*n),ncol=n),fit=0)} else {
    eigens <- eigen(r)    #call the eigen value decomposition routine
    result$values <- round(eigens$values,digits)
   if(nfactors >0) {loadings <- eigen.loadings(eigens)[,1:nfactors] } else {loadings <- eigen.loadings(ei
   nfactors <- n}

        if(nfactors >1) {
        maxabs <- apply(apply(loadings,2,abs),2,which.max)
        sign.max <- vector(mode="numeric",length=nfactors)
        for (i in 1: nfactors) {sign.max[i] <- sign(loadings[maxabs[i],i])}
```

```
            loadings <- loadings
    } else {if(nfactors >0) {mini <- min(loadings)
                    maxi <- max(loadings)
            if (abs(mini) > maxi) {loadings <- -loadings }
            loadings <- as.matrix(loadings)
     }}

colnames(loadings) <- paste("PC",1:nfactors,sep='')
rownames(loadings) <- rownames(r)

if(rotate) {loadings <- varimax(loadings)$loadings }

residual<- factor.residuals(r,loadings)
r2 <- sum(r*r)
rstar2 <- sum(residual*residual)
if (residuals) {result$residual <-residual}
result$fit <- round(1-rstar2/r2,digits)
result$loadings <- round(loadings,digits)
return(result)
}
```

`psycho.demo` *Create demo data for psychometrics*

Description

A not very interesting demo of what happens if bivariate continuous data are dichotomized. Bascially a demo of r, phi, and polychor.

Usage

```
psycho.demo()
```

Details

Not one of my more interesting demonstrations. See `http://personality-project.org/r/simulating-personality.html` and `http://personality-project.org/r/r.datageneration.html` for better demonstrations of data generation.

Value

a matrix of correlations)

Author(s)

William Revelle

References

`http://personality-project.org/r/simulating-personality.html` and `http://personality-project.org/r/r.datageneration.html` for better demonstrations of data generation.

See Also

VSS.simulate,

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function() {
 #simulate correlation matrix with variable cut points -- psychometric demo
 #make up some correlations with different cut points
cuts <-c(-2,-1,0,1,2)
nsub<-1000    #how many subjects
r<-.6         #population correlation of observed with theta
theta <-rnorm(nsub)  #make up some random normal theta scores
err<- rnorm(nsub)    #random normal error scores

obser<- theta*(r) + err*sqrt(1-r*r)  #observed = T + E

#convert to 0/1  with different values of cuts
trunc<- matrix(rep(obser,length(cuts)),ncol=length(cuts))
for (i in 1:length(cuts)) {
   trunc[obser>cuts[i],i]<- 1
   trunc[obser< cuts[i],i]<- 0}

d.mat<- data.frame(theta,obser,trunc)  #combine into a data frame
trunc.cor<- cor(d.mat)                 #find the correlations
freq <- mean(d.mat)                    #find the frequencies of scores

#now, convert the upper diagonal to polychorics using John Fox's polychor and my phi2poly

for (i in 4:length(d.mat)) {
   for (j in 3:i) {
       trunc.cor[j,i]<- phi2poly(trunc.cor[i,j],freq[i],freq[j])
       }}

  }
```

---

read.clipboard   *shortcut for reading from the clipboard*

Description

input from the keyboard is easy but a bit obscure, particularly for Mac users. This is just an easier mnemonic to do so.

Usage

```
read.clipboard(header = TRUE, ...)

#my.data <- read.clipboad()         #assumes headers and tab delimited
#my.data <- read.clipboard.csv()     #assumes heades and comma delimited
```

Arguments

`header`	Does the first row have variable labels
`...`	Other parameters to pass to read

Details

A typical session of R might involve data stored in text files, generated on line, etc. Although it is easy to just read from a file (particularly if using file.locate(), copying from the file to the clipboard and then reading from the clipboard is also very convenient (and somewhat more intuitive to the naive user.)

Value

the contents of the clipboard.

Author(s)

William Revelle

Examples

```
#my.data <- read.clipboad()
#my.data <- read.clipboard.csv()

## The function is currently defined as
function(header=TRUE,...) {
    MAC<-Sys.info()[1]=="Darwin"    #are we on a Mac using the Darwin system?
   if (!MAC ) {if (header) read.clipboard<-read.table(file("clipboard"),header=TRUE,...)
            else read.clipboard<-read.table(file("clipboard"),...) }
    else {
   if (header) read.clipboard<-  read.table(pipe("pbpaste"),header=TRUE,...)
   else read.clipboard<- read.table(pipe("pbpaste"),...)}
   }
```

`schmid` *Apply the Schmid Leiman transformation to a correlation matrix*

Description

One way to find omega is to do a factor analysis of the original data set, rotate the factors obliquely, do a Schmid Leiman transformation, and then find omega. Here is the code for Schmid Leiman. The S-L transform takes a factor or PC solution, transforms it to an oblique solution, factors the oblique solution to find a higher order (g) factor, and then residualizes g out of the the group factors.

Usage

```
schmid(model, nfactors = 3, pc = "pa",...)
```

Arguments

`model`	A correlation matrix
`nfactors`	Number of factors to extract
`pc`	pc="pa" for principal axes, pc="pc" for principal components, pc="mle" for maximum likelihood
`...`	Allows additional parameters to be passed to the factoring routines

Details

Schmid Leiman orthogonalizations are typical in the ability domain, but are not seen as often in the non-cognitive personality domain. S-L is one way of finding the loadings of items on the general factor for estimating omega.

A typical example would be in the study of anxiety and depression. A general neuroticism factor (g) accounts for much of the variance, but smaller group factors of tense anxiety, panic disorder, depression, etc. also need to be considerd.

An alternative model is to consider hierarchical cluster analysis

Requires the GPArotation package. (`ICLUST`).

Value

`sl`	loadings on g + nfactors group factors, communalities, uniqueness
`orthog`	original orthogonal factor loadings
`oblique`	oblique factor loadings
`fcor`	correlations among the transformed factors
`gload`	loadings of the lower order factors on g
...	

Author(s)

William Revelle

References

`http://personality-project.org/r/r.omega.html` gives an example taken from Jensen and Weng, 1994 of a S-L transformation.

See Also

omega, ICLUST,VSS

Examples

```
## The function is currently defined as
function (model, nfactors = 3, pc = "pa",...)
{
    if (pc=="pc") {
        fact <- principal(model, nfactors,...)
    } else {if (pc=="pa") {fact <- factor.pa(model, nfactors,...) } else {
        fact <- factanal(x, covmat = model, factors = nfactors,...)
    }}
       obminfact <- oblimin(loadings(fact))
    rownames(obminfact$loadings) <- attr(model,"dimnames")[[1]]
    fload <- obminfact$loadings
    factr <- t(obminfact$Th) %*% (obminfact$Th)
    gfactor <- factanal(x, covmat = factr, factors = 1)
    gload <- loadings(gfactor)
    gprimaryload <- fload %*% gload
    colnames(gprimaryload) <- "g factor"
    u2 <- 1 - diag(fload %*% t(fload))
    h2 <- 1 - u2
    uniq <- 1 - fload^2
    guniq <- 1 - gprimaryload^2
    Ig <- matrix(0, ncol = nfactors, nrow = nfactors)
    diag(Ig) <- gload
    primeload <- fload %*% Ig
    uniq2 <- 1 - uniq - primeload^2
    sm <- sqrt(uniq2)
    schmid <- list(sl = cbind(gprimaryload, sm, h2, u2), orthog = fload,
        fcor = factr, gloading = gload)
}
```

---

score.alpha — *Score scales and find Cronbach's alpha as well as associated statistics*

Description

Given a matrix or data.frame of k keys for m items (-1, 0, 1), and a matrix or data.frame of items scores for m items and n people, find the sum scores or average scores for each person and each scale. In addition, report Cronbach's alpha, the average r, the scale intercorrelations, and the item by scale correlations.

Usage

```
score.alpha(keys, items, labels = NULL, totals=TRUE,digits = 2)
```

Arguments

`keys`	A matrix or dataframe of -1, 0, or 1 weights for each item on each scale
`items`	Data frame or matrix of raw item scores
`labels`	column names for the resulting scales
`totals`	Find sum scores (default) or average score
`digits`	Number of digits for answer (default =2)

Details

The process of finding sum or average scores for a set of scales given a larger set of items is a typical problem in psychometric research. Although the structure of scales can be determined from the item intercorrelations, to find scale means, variances, and do further analyses, it is typical to find the sum or the average scale score.

Various estimates of scale reliability include "Cronbach's alpha", and the average interitem correlation. For k = number of items in a scale, and av.r = average correlation between items in the scale, alpha = k * av.r/(1+ (k-1)*av.r). Thus, alpha is an increasing function of test length as well as the test homeogeneity.

Alpha is a poor estimate of the general factor saturation of a test (see Zinbarg et al., 2005) for it can seriously overestimate the size of a general factor, and a better but not perfect estimate of total test reliability because it underestimates total reliability. None the less, it is a useful statistic to report.

Value

`scores`	Sum or average scores for each subject on the k scales
`alpha`	Cronbach's coefficient alpha. A simple (but non-optimal) measure of the internal consistency of a test. See also beta and omega.
`av.r`	The average correlation within a scale, also known as alpha 1 is a useful index of the internal consistency of a domain.
`n.items`	Number of items on each scale
`cor`	The intercorrelation of all the scales
`item.cor`	The correlation of each item with each scale. Because this is not corrected for item overlap, it will overestimate the amount that an item correlates with the other items in a scale.

Author(s)

William Revelle

References

An introduction to psychometric theory with applications in R (in preparation). `http://personality-project.org/r/book`

See Also

`alpha.scale`, `correct.cor`, `alpha.scale`, `cluster.loadings`, `omega`

Examples

```
y <- attitude     #from the datasets package
keys <- matrix(c(rep(1,7),rep(1,4),rep(0,7),rep(-1,3)),ncol=3)
labels <- c("first","second","third")
x <- score.alpha(keys,y,labels)

## The function is currently defined as
function (keys,items,labels=NULL,totals=TRUE,digits=2) {
    keys <- as.matrix(keys)   #just in case they were not matrices to start with
    items <- as.matrix(items)
    scores <- items %*%  keys  #this actually does all the work
    if (length(labels)>0) {colnames(scores) <- labels} #add labels
    abskeys <- abs(keys)
    item.var <- diag(var(items))  #find the item variances
    cov.scales  <- cov(scores)    #and total scale variance
    var.scales <- diag(cov.scales)
    cor.scales <- cor(scores)     #could do this as matrix operation, but why bother
    sum.item.var <- item.var %*% abskeys
    num.item <- diag(t(abskeys) %*% abskeys) #how many items in each scale
    alpha.scale <- (var.scales - sum.item.var)*num.item/((num.item-1)*var.scales)
   if (length(labels)>0) {colnames(alpha.scale) <- labels}
   av.r <- alpha.scale/(num.item - alpha.scale*(num.item-1))  #alpha 1 = average r
   item.cor <- cor(items,scores)
    results <- list(scores=scores,alpha=round(alpha.scale,digits), av.r=round(av.r,digits), n.items = num
 }
```

`score.items` *Score item composite scales and find Cronbach's alpha as well as associated statistics*

Description

Given a matrix or data.frame of k keys for m items (-1, 0, 1), and a matrix or data.frame of items scores for m items and n people, find the sum scores or average scores for each person and each scale. In addition, report Cronbach's alpha, the average r, the scale intercorrelations, and the item by scale correlations. Replace missing values with the item mean if desired. Will adjust scores for reverse scored items.

Usage

```
score.items(keys, items, totals = TRUE, ilabels = NULL, missing = TRUE, min = NULL, max = NULL,
```

Arguments

`keys`	A matrix or dataframe of -1, 0, or 1 weights for each item on each scale
`items`	Matrix or dataframe of raw item scores
`totals`	if TRUE (default) find total scores, if FALSE, find average scores

`ilabels`	a vector of item labels
`missing`	TRUE: Replace missing values with the corresponding item mean. FALSE: do not score that subject
`min`	May be specified as minimum item score allowed, else will be calculated from data
`max`	May be specified as maximum item score allowed, else will be calculated from data
`digits`	Number of digits to report

Details

The process of finding sum or average scores for a set of scales given a larger set of items is a typical problem in psychometric research. Although the structure of scales can be determined from the item intercorrelations, to find scale means, variances, and do further analyses, it is typical to find scores based upon the sum or the average item score.

Various estimates of scale reliability include "Cronbach's alpha", and the average interitem correlation. For k = number of items in a scale, and av.r = average correlation between items in the scale, alpha = k * av.r/(1+ (k-1)*av.r). Thus, alpha is an increasing function of test length as well as the test homeogeneity.

Alpha is a poor estimate of the general factor saturation of a test (see Zinbarg et al., 2005) for it can seriously overestimate the size of a general factor, and a better but not perfect estimate of total test reliability because it underestimates total reliability. None the less, it is a useful statistic to report.

Correlations between scales are attenuated by a lack of reliability. Correcting correlations for reliability (by dividing by the square roots of the reliabilities of each scale) sometimes help show structure.

Value

`scores`	Sum or average scores for each subject on the k scales
`alpha`	Cronbach's coefficient alpha. A simple (but non-optimal) measure of the internal consistency of a test. See also beta and omega. Set to 1 for scales of length 1.
`av.r`	The average correlation within a scale, also known as alpha 1 is a useful index of the internal consistency of a domain. Set to 1 for scales with 1 item.
`n.items`	Number of items on each scale
`item.cor`	The correlation of each item with each scale. Because this is not corrected for item overlap, it will overestimate the amount that an item correlates with the other items in a scale.
`cor`	The intercorrelation of all the scales
`corrected`	The correlations of all scales (below the diagonal), alpha on the diagonal, and the unattenuated correlations (above the diagonal)

Author(s)

William Revelle

References

An introduction to psychometric theory with applications in R (in preparation). http://personality-project.org/r/book

See Also

alpha.scale, correct.cor, cluster.cor , cluster.loadings, omega

Examples

```
y <- attitude     #from the datasets package
keys <- matrix(c(rep(1,7),rep(1,4),rep(0,7),rep(-1,3)),ncol=3)
colnames(keys) <- c("first","second","third")
x <- score.items(keys,y)
#x    #to see the output
## The function is currently defined as
function (keys,items,totals=TRUE,ilabels=NULL, missing=TRUE, min=NULL,max=NULL,digits=2) {
    keys <- as.matrix(keys)   #just in case they were not matrices to start with
    items <- as.matrix(items)
    item.means <- colMeans(items,na.rm=TRUE)
    if (is.null(min)) {min <- min(items,na.rm=TRUE)}
    if (is.null(max)) {max <- max(items,na.rm=TRUE)}
    if(missing) { miss.rep <- rowSums(is.na(items))
       item.means <- colMeans(items,na.rm=TRUE)
            miss <- which(is.na(items),arr.ind=TRUE)
           items[miss]<- item.means[miss[,2]]}
    n.keys <- dim(keys)[2]
    n.items <- dim(keys)[1]
    n.subjects <- dim(items)[1]
     scores<- items \%*\%  keys  #this actually does all the work but doesn't handle missing

    scores<- items \%*\%  keys  #this actually does all the work
    slabels <- colnames(keys)
    if (is.null(slabels)) {slabels<- paste("S",1:n.keys,sep="")}
    colnames(scores) <- slabels

    abskeys <- abs(keys)
    item.var <- diag(var(items,use="pairwise"))  #find the item variances
    cov.scales  <- cov(scores,use="pairwise")    #and total scale variance
    var.scales <- diag(cov.scales)
    cor.scales <- cor(scores,use="pairwise")     #could do this as matrix operation, but why bother
    sum.item.var <- item.var \%*\% abskeys
    num.item <- diag(t(abskeys) \%*\% abskeys) #how many items in each scale
    alpha.scale <- (var.scales - sum.item.var)*num.item/((num.item-1)*var.scales)
     alpha.scale[is.nan(alpha.scale)] <- 1      #set to 1 for one item scales
    colnames(alpha.scale) <- slabels
   av.r <- alpha.scale/(num.item - alpha.scale*(num.item-1))  #alpha 1 = average r
   item.cor <- cor(items,scores,use="pairwise")
```

```
    if(is.null(ilabels)) {ilabels <-  paste("I",1:n.items,sep="")}
    rownames(item.cor) <- ilabels
    correction <- (colSums(abs(keys)-(keys))/2)*(max+min)
    scores <- scores  + matrix(rep(correction,n.subjects),byrow=TRUE,nrow=n.subjects)

    if (!totals) {scores <- scores \%*\% diag(1/num.item)   #find averages
                  colnames(scores) <- paste("A",1:n.keys,sep="") }
  scale.cor <- correct.cor(cor.scales,t(alpha.scale))
    if (missing){results <-list(scores=scores,missing = miss.rep,alpha=round(alpha.scale,digits), av.r=rou
     results <- list(scores=scores,alpha=round(alpha.scale,digits), av.r=round(av.r,digits), n.items = num
  }
```

`skew` *Calculate skew for a vector, matrix, or data.frame*

Description

Find the skew for each variable in a data.frame or matrix. Unlike skew in e1071, this calculates a different skew for each variable or column of a data.frame/matrix.

Usage

```
skew(x, na.rm = TRUE)
```

Arguments

`x`	A data.frame or matrix
`na.rm`	how to treat missing data

Details

given a matrix or data.frame x, find the skew for each column.

Value

if input is a matrix or data.frame, skew is a vector of skews

Note

The mean function supplies means for the columns of a data.frame, but the overall mean for a matrix. Mean will throw a warning for non-numeric data, but colMeans stops with non-numeric data. Thus, the function uses either mean (for data frames) or colMeans (for matrices). This is true for skew and kurtosi as well.

Author(s)

William Revelle

See Also

describe, describe.by, kurtosi

Examples

```
round(skew(attitude),2)
## The function is currently defined as
function (x, na.rm = TRUE)
{
    if (length(dim(x)) == 0) {
        if (na.rm) {
            x <- x[!is.na(x)]
                                    }
        sdx <- sd(x,na.rm=na.rm)
        mx <- mean(x)
        skewer <- sum((x - mx)^3)/(length(x) * sd(x)^3)
        } else {

    skewer <- rep(NA,dim(x)[2])
    mx <- colMeans(x,na.rm=na.rm)
    sdx <- sd(x,na.rm=na.rm)
    for (i in 1:dim(x)[2]) {
    skewer[i] <- sum((x[,i] - mx[i])^3,  na.rm = na.rm)/((length(x[,i]) - sum(is.na(x[,i]))) * sdx[i]^3)
            }
    }
    return(skewer)
}
```

`test.psych` *Testing of functions in the psych package*

Description

Test to make sure the psych functions run on basic test data sets

Usage

```
test.psych(first=1,last=5,short=TRUE)
```

Arguments

`first`	first=1: start with dataset first
`last`	last=5: test for datasets until last
`short`	short=TRUE - don't return any analyses

Details

When modifying the psych package, it is useful to make sure that adding some code does not break something else. This function tests the major functions on various standard data sets.

Value

`out`	if short=FALSE, then list of the output from all functions tested

Warning

Warning messages will be thrown by fa.parallel

Note

Not for general consumption – used to make sure functions throw error messages or correct for weird conditions.

The datasets tested are part of the standard R data sets and represent some of the basic problems encountered.

Author(s)

William Revelle

Examples

```
#test <- psych.test()
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
test.psych <-
function(first=1,last=5,short=TRUE) {
s1 <- USArrests         #  Violent Crime Rates by US State  (4 variables)
s2 <- attitude          #The Chatterjee-Price Attitude Data
s3 <- Harman23.cor$cov     #   Harman Example 2.3 8 physical measurements
s4 <- Harman74.cor$cov     #   Harman Example 7.4  24 mental measurements
s5 <- ability.cov$cov       #  8 Ability and Intelligence Tests

#convert covariance to correlation
d5 <- diag(1/sqrt(diag(s5)))
s5 <- d5

datasets <- list(s1,s2,s3,s4,s5)
out <- list()

for (i in first:last) {
   test.data <- datasets[[i]]
        pc <-   principal(test.data)
        pc2 <-    principal(test.data,2)
        fa2 <- factor.pa(test.data,2)
        fp <-    fa.parallel(test.data)
        ic <-   ICLUST(test.data)
        om <-  omega(test.data)
        fc <- factor.congruence(pc2,fa2)
        vss2 <- VSS(test.data)
        vsspc <- VSS(test.data,pc="pc")
        vss.scree <- VSS.scree(test.data)
        d <- describe(test.data)


        keys <- matrix(rep(0,dim(test.data)[2]*2),ncol=2)
        keys[,1] <- 1
        keys[1:3,2] <- 1
        if( dim(test.data)[1] != dim(test.data)[2]) {test.score <- score.items(keys,test.data)} else {tes

        out <- list(out,paste("test",i),pc,pc2,fa2,fp,ic,om,fc,vss2,vsspc,vss.scree,d,test.score)
  } #end loop

  #a few more tests

  simple <- item.sim(nvar=24)
  circ <-  circ.sim(nvar=24)
  fa.simple <- factor.pa(simple)
  psych.d <- psycho.demo()

 if (!short) { return(out)}

}#end function
```

VSS.parallel *Compare real and random VSS solutions*

Description

Another useful test for the number of factors is when the eigen values of a random matrix are greater than the eigen values of a a real matrix. Here we show VSS solutions to random data.

Usage

```
VSS.parallel(ncases, nvariables,scree=FALSE,rotate="none")
```

Arguments

`ncases`	Number of simulated cases
`nvariables`	number of simulated variables
`scree`	Show a scree plot for random data – see `omega`
`rotate`	rotate=”none” or rotate=”varimax”

Value

VSS like output to be plotted by VSS.plot

Author(s)

William Revelle

References

Very Simple Structure (VSS)

See Also

VSS.plot, ICLUST, omega

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(ncases,nvariables,scree=FALSE,rotate="none")       #function call
{
 simdata=matrix(rnorm(ncases*nvariables),nrow=ncases,ncol=nvariables)    #make up simulated data
 if(scree) {VSS.scree(simdata)
 testsim <- simdata}  else {testsim <- VSS(simdata,8,rotate)}
 return(testsim)
 }
```

VSS.plot *Plot VSS fits*

Description

The Very Simple Structure criterion (VSS) for estimating the optimal number of factors is plotted as a function of the increasing complexity and increasing number of factors.

Usage

```
VSS.plot(x, plottitle = "Very Simple Structure", line = FALSE)
```

Arguments

`x`	output from VSS
`plottitle`	any title
`line`	connect different complexities

Details

Item-factor models differ in their "complexity". Complexity 1 means that all except the greatest (absolute) loading for an item are ignored. Basically a cluster model (e.g., `ICLUST`). Complexity 2 implies all except the greatest two, etc.

Different complexities can suggest different number of optimal number of factors to extract. For personality items, complexity 1 and 2 are probably the most meaningful.

The Very Simple Structure criterion will tend to peak at the number of factors that are most interpretable for a given level of complexity. Note that some problems, the most interpretable number of factors will differ as a function of complexity. For instance, when doing the Harman 24 psychological variable problems, an unrotated solution of complexity one suggests one factor (g), while a complexity two solution suggests that a four factor solution is most appropriate. This latter probably reflects a bi-factor structure.

For examples of VSS.plot output, see `http://personality-project.org/r/r.vss.html`

Value

A plot window showing the VSS criterion varying as the number of factors and the complexity of the items.

Author(s)

Maintainer: William Revelle ⟨revelle@northwestern.edu⟩

References

`http://personality-project.org/r/r.vss.html`

See Also

`VSS`, `ICLUST`, `omega`

Examples

```
#test.data <- Harman74.cor$cov
#my.vss <- VSS(test.data)         #suggests that 4 factor complexity two solution is optimal
#VSS.plot(my.vss)                 #see the graphics window

## The function is currently defined as
function(x,plottitle="Very Simple Structure",line=FALSE)
  {
        n=dim(x)
        symb=c(49,50,51,52)              #plotting sym
        plot(x$cfit.1,ylim=c(0,1),type="b",ylab="Very Simple Structure Fit",xlab="Number of Factors",pch=
    if (line) lines(x$fit)


        title(main=plottitle)
        x$cfit.2[1]<-NA
        x$cfit.3[1]<-NA
        x$cfit.3[2]<-NA
```

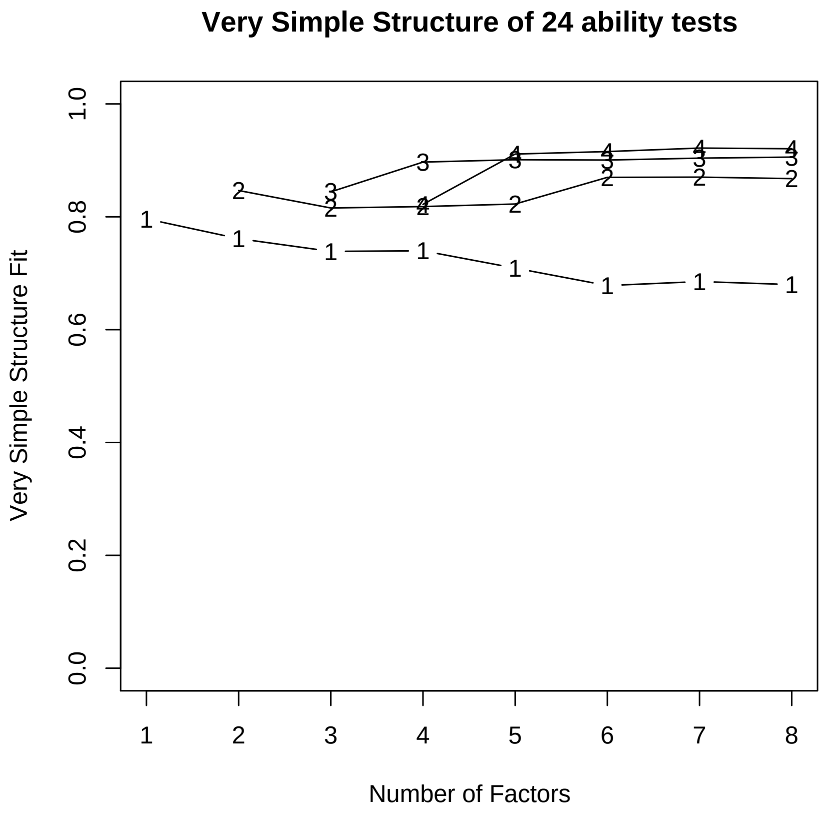

Figure 11: The Very Simple Structure criterion plots goodness of fit as a function of the number of factors extracted and the factorial complexity for each item/test. Note how a complexity one solution best fits the data if only one factor is extracted, but a complexity three solution is optimized at 4 factors.

```
        x$cfit.4[1]<-NA
        x$cfit.4[2]<-NA
        x$cfit.4[3]<-NA
        lines(x$cfit.2)
        points(x$cfit.2,pch=50)
        lines(x$cfit.3)
        points(x$cfit.3,pch=symb[3])
        lines(x$cfit.4)
        points(x$cfit.4,pch=symb[4])
    }
```

`VSS.scree` *Plot a scree test*

Description

Cattell's scree test is one of most simple ways of testing the number of components in a correlation matrix. Here we plot the eigen values of a correlation matrix.

Usage

```
VSS.scree(rx, main = "scree plot")
```

Arguments

`rx`	a correlation matrix or a data matrix. If data, then correlations are found using pairwise deletions.
`main`	Title

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illiniois

Maintainer: William Revelle <revelle@northwestern.edu>

References

http://personality-project.org/r/vss.html

See Also

VSS.plot, ICLUST, omega

Examples

```
#VSS.scree(attitude)
#VSS.scree(cor(attitude)

## The function is currently defined as

function(rx,main="scree plot") {
 nvar <- dim(rx)[2]
 if (nvar <dim(rx)[1]) {rx <- cor(rx,use="pairwise")}
 values <-  eigen(rx)$values
plot(values,type="b", main = main )
abline(h=1) }
```

`VSS.simulate` *create VSS like data*

Description

Simulation is one of most useful techniques in statistics and psychometrics. Here we simulate a correlation matrix with a simple structure composed of a specified number of factors. Each item is assumed to have complexity one.

Usage

```
VSS.simulate(ncases=1000, nvariables=16, nfactors=4, meanloading=.5)
```

Arguments

`ncases`	number of simulated subjects
`nvariables`	Number of variables
`nfactors`	Number of factors to generate
`meanloading`	with a mean loading

Value

a ncases x nvariables matrix

Author(s)

William Revelle

See Also

`VSS`, `ICLUST`

Examples

```
## Not run:
simulated <- VSS.simulate(1000,20,4,.6)
vss <- VSS(simulated,rotate="varimax")
VSS.plot(vss)
## End(Not run)
## The function is currently defined as
function(ncases=1000,nvariables=16,nfactors=4,meanloading=.5)
#generates a simple structure factor matrix
                                #with nfactors

{
weight=sqrt(1-meanloading*meanloading)    #loadings are path coefficients
theta=matrix(rnorm(ncases*nfactors),nrow=ncases,ncol=nvariables)
#generates nfactor independent columns, repeated nvariable/nfactor times)
error=matrix(rnorm(ncases*nvariables),nrow=ncases,ncol=nvariables)
```

```
#errors for all variables
items=meanloading*theta+weight*error
#observed score = factor score + error score
return(items)
  }
```

VSS	*Apply the Very Simple Structure criterion to determine the appropriate number of factors.*

Description

There are multiple ways to determine the appropriate number of factors in exploratory factor analysis. Routines for the Very Simple Structure (VSS) criterion allow one to compare solutions of varying complexity and for different number of factors. Graphic output indicates the "optimal" number of factors for different levels of complexity.

Usage

```
VSS(x, n = 8, rotate = "none", diagonal = FALSE, pc = "pa", n.obs=1000,...)
```

Arguments

x	a correlation matrix or a data matrix
n	Number of factors to extract – should be more than hypothesized!
rotate	what rotation to use
diagonal	Should we fit the diagonal as well
pc	pc="pa" Principal Axis Factor Analysis, pc="mle" Maximum Likelihood FA, pc="pc" Principal Components"
n.obs	Number of observations if doing a factor analysis of correlation matrix. This value is ignored by VSS but is necessary for the ML factor analysis package.
...	parameters to pass to the factor analysis program The most important of these is if using a correlation matrix is covmat= xx

Details

Determining the most interpretable number of factors from a factor analysis is perhaps one of the greatest challenges in factor analysis. There are many solutions to this problem, none of which is uniformly the best. "Solving the number of factors problem is easy, I do it everyday before breakfast. But knowing the right solution is harder" (Kaiser, 195x).

Techniques most commonly used include

1) Extracting factors until the chi square of the residual matrix is not significant.

2) Extracting factors until the change in chi square from factor n to factor n+1 is not significant.

3) Extracting factors until the eigen values of the real data are less than the corresponding eigen values of a random data set of the same size (parallel analysis).

4) Plotting the magnitude of the successive eigen values and applying the scree test (a sudden drop in eigen values analogous to the change in slope seen when scrambling up the talus slope of a mountain and approaching the rock face.

5) Extracting principal components until the eigen value <1. <

6) Extracting factors as long as they are interpetable.

7) Using the Very Structure Criterion.

Each of the procedures has its advantages and disadvantages. Using either the chi square test or the change in square test is, of course, sensitive to the number of subjects and leads to the nonsensical condition that if one wants to find many factors, one simlpy runs more subjects. Parallel analysis is partially sensitive to sample size in that for large samples the eigen values of random factors will be very small. The scree test is quite appealling but can lead to differences of interpretation as to when the scree "breaks". The eigen value of 1 rule, although the default for many programs, seems to be a rough way of dividing the number of variables by 3. Extracting interpretable factors means that the number of factors reflects the investigators creativity more than the data. VSS, while very simple to understand, will not work very well if the data are very factorially complex. (Simulations suggests it will work fine if the complexities of some of the items are no more than 2).

Most users of factor analysis tend to interpret factor output by focusing their attention on the largest loadings for every variable and ignoring the smaller ones. Very Simple Structure operationalizes this tendency by comparing the original correlation matrix to that reproduced by a simplified version (S) of the original factor matrix (F). R = SS' + U2. S is composed of just the c greatest (in absolute value) loadings for each variable. C (or complexity) is a parameter of the model and may vary from 1 to the number of factors.

The VSS criterion compares the fit of the simplified model to the original correlations: VSS = 1 -sumsquares(r*)/sumsquares(r) where R* is the residual matrix R* = R - SS' and r* and r are the elements of R* and R respectively.

VSS for a given complexity will tend to peak at the optimal (most interpretable) number of factors (Revelle and Rocklin, 1979).

Although originally written in Fortran for main frame computers, VSS has been adapted to micro computers (e.g., Macintosh OS 6-9) using Pascal. We now release R code for calculating VSS.

Note that if using a correlation matrix (e.g., my.matrix) and doing a factor analysis, the parameters n.obs should be specified for the factor analysis: the call is VSS(my.matrix,n.obs=500). Otherwise it defaults to 1000.

Value

A data.frame with entries: dof: degrees of freedom (if using FA)
chisq: chi square (from the factor analysis output (if using FA)
prob: probability of residual matrix > 0 (if using FA)
sqresid: squared residual correlations
fit: factor fit of the complete model
cfit.1: VSS fit of complexity 1
cfit.2: VSS fit of complexity 2

...
cfit.8: VSS fit of complexity 8
cresidiual.1: sum squared residual correlations for complexity 1
...: sum squared residual correlations for complexity 2 ..8

Author(s)

William Revelle
Department of Psychology
Northwestern University
Evanston, Illiniois

Maintainer: William Revelle <revelle@northwestern.edu>

References

http://personality-project.org/r/vss.html see also Revelle, W. and Rocklin, T. 1979, Very Simple Structure: an Alternative Procedure for Estimating the Optimal Number of Interpretable Factors, Multivariate Behavioral Research, 14, 403-414. http://personality-project.org/revelle/publications/vss.pdf

See Also

VSS.plot, ICLUST, omega

Examples

```
## Not run:
test.data <- Harman74.cor$cov
my.vss <- VSS(test.data)         #suggests that 4 factor complexity two solution is optimal
print(my.vss[,1:12],digits =2)
VSS.plot(my.vss)                 #see graphic window for a plot

#produces this output
#  dof chisq     prob sqresid  fit cfit.1 cfit.2 cfit.3 cfit.4 cfit.5 cfit.6 cfit.7
#1 252  4583  0.0e+00    17.2 0.79   0.79   0.00   0.00   0.00   0.00   0.00   0.00
#2 229  3105  0.0e+00    12.9 0.84   0.75   0.84   0.00   0.00   0.00   0.00   0.00
#4 186  1689 2.3e-240     8.0 0.90   0.66   0.87   0.90   0.90   0.00   0.00   0.00
#5 166  1398 9.3e-194     7.3 0.91   0.68   0.86   0.90   0.91   0.91   0.00   0.00
#6 147  1183 2.9e-161     6.5 0.92   0.53   0.83   0.88   0.91   0.92   0.92   0.00
#7 129  1002 5.8e-135     5.7 0.93   0.47   0.78   0.88   0.91   0.92   0.93   0.93
#8 112   803 5.3e-105     5.3 0.94   0.49   0.76   0.86   0.90   0.92   0.93   0.93

#compare the above solution to a "varimax" rotated solution which suggests 1 factor (g)

my.vss <- VSS(test.data,rotate="varimax")         #suggests that 1 factor complexity one solution is opti
print(my.vss[,1:14],digits =2)
VSS.plot(my.vss)                 #see graphic window for a plot

#  dof chisq     prob sqresid  fit cfit.1 cfit.2 cfit.3 cfit.4 cfit.5 cfit.6 cfit.7 cfit.8 cresidual.1
```

```
#1 252  4583  0.0e+00    17.2 0.79   0.79   0.00   0.00    0.0   0.00   0.00   0.00   0.00          17
#2 229  3105  0.0e+00    12.9 0.84   0.54   0.84   0.00    0.0   0.00   0.00   0.00   0.00          38
#3 207  2193  0.0e+00    10.1 0.88   0.46   0.79   0.88    0.0   0.00   0.00   0.00   0.00          45
#4 186  1689 2.3e-240     8.0 0.90   0.42   0.73   0.87    0.9   0.00   0.00   0.00   0.00          48
#5 166  1398 9.3e-194     7.3 0.91   0.40   0.70   0.86    0.9   0.91   0.00   0.00   0.00          50
#6 147  1183 2.9e-161     6.5 0.92   0.39   0.69   0.86    0.9   0.92   0.92   0.00   0.00          51
#7 129  1002 5.8e-135     5.7 0.93   0.39   0.70   0.84    0.9   0.92   0.93   0.93   0.00          50
#8 112   803 5.3e-105     5.3 0.94   0.39   0.69   0.83    0.9   0.92   0.93   0.93   0.94          50
## End(Not run)

## The function is currently defined as
function (x,n=8,rotate="none",diagonal=FALSE,pc="pa",n.obs=1000,...)     #apply the Very Simple Structure
  #x is a data matrix
  #n is the maximum number of factors to extract  (default is 8)
  #rotate is a string "none" or "varimax" for type of rotation (default is "none"
  #diagonal is a boolean value for whether or not we should count the diagonal  (default=FALSE)
  # ... other parameters for factanal may be passed as well
  #e.g., to do VSS on a covariance/correlation matrix with up to 8 factors and 3000 cases:
  #VSS(covmat=msqcovar,n=8,rotate="none",n.obs=3000)


 {             #start Function definition
  #first some preliminary functions
  #complexrow sweeps out all except the c largest loadings
  #complexmat applies complexrow to the loading matrix


complexrow <- function(x,c)     #sweep out all except c loadings
    {  n=length(x)              #how many columns in this row?
       temp <- x                #make a temporary copy of the row
       x <- rep(0,n)            #zero out x
       for (j in 1:c)
       {
        locmax <- which.max(abs(temp))                    #where is the maximum (absolute) value
         x[locmax] <- sign(temp[locmax])*max(abs(temp))    #store it in x
        temp[locmax] <- 0                                 #remove this value from the temp copy
       }
     return(x)                                            #return the simplified (of complexity c) row
    }

 complexmat <- function(x,c)           #do it for every row   (could tapply somehow?)
        {
        nrows <- dim(x)[1]
        ncols <- dim(x)[2]
        for (i in 1:nrows)
                {x[i,] <- complexrow(x[i,],c)}   #simplify each row of the loading matrix
        return(x)
         }

  #now do the main Very Simple Structure  routine

  complexfit <- array(0,dim=c(n,n))        #store these separately for complex fits
  complexresid <-  array(0,dim=c(n,n))
  vss.df <- data.frame(dof=rep(0,n),chisq=0,prob=0,sqresid=0,fit=0) #keep the basic results here

  if (dim(x)[1]!=dim(x)[2]) x <- cor(x,use="pairwise") # if given a rectangular

 for (i in 1:n)                            #loop through 1 to the number of factors requested
 {
   if(!(pc=="pc")) { if ( pc=="pa") {
                f <- factor.pa(x,i,rotate=rotate,...)   #do a factor analysis with i factors and the rota
         if (i==1)
                 {original <- x         #just find this stuff once
                 sqoriginal <- original*original    #squared correlations
                 totaloriginal <- sum(sqoriginal) - diagonal*sum(diag(sqoriginal) )   #sum of squared cor
                }}  else {
        f <- factanal(x,i,rotation=rotate,covmat=x,n.obs=n.obs,...)  #do a factor analysis with i factors
         if (i==1)
                 {original <- x         #just find this stuff once
                 sqoriginal <- original*original    #squared correlations
                 totaloriginal <- sum(sqoriginal) - diagonal*sum(diag(sqoriginal) )   #sum of squared cor
                }}
          } else {f <- principal(x,i)
            if (i==1)
                         {original <- x       #the input to pc is a correlation matrix, so we don't need t
                         sqoriginal <- original*original    #squared correlations
                        totaloriginal <- sum(sqoriginal) - diagonal*sum(diag(sqoriginal) )   #sum of squa
                        }
                if((rotate=="varimax") & (i>1)) {f <- varimax(f$loadings)}
            }

        load <- as.matrix(f$loadings )                    #the loading matrix
        model <- load
        residual <- original-model              #find the residual  R* = R - FF'
        sqresid <- residual*residual            #square the residuals
        totalresid <- sum(sqresid)- diagonal * sum(diag(sqresid) )      #sum squared residuals - the main
        fit <- 1-totalresid/totaloriginal       #fit is 1-sumsquared residuals/sumsquared original     (o

        if ((pc=="mle")) {
                        vss.df[i,1] <- f$dof                   #degrees of freedom from the factor analys
                        vss.df[i,2] <- f$STATISTIC             #chi square from the factor analysis
                        vss.df[i,3] <- f$PVAL                  #probability value of this complete soluti
                         }
        vss.df[i,4] <- totalresid              #residual given complete model
        vss.df[i,5] <- fit                     #fit of complete model



     #now  do complexities -- how many factors account for each item

  for (c in 1:i)

         {
                simpleload <- complexmat(load,c)             #find the simple structure version of the lo
```

```
                model <- simpleload
                residual <- original- model                   #R* = R - SS'
                sqresid <- residual*residual
                totalsimple <- sum(sqresid) -diagonal * sum(diag(sqresid))    #default is to not count th
                simplefit <- 1-totalsimple/totaloriginal
                complexresid[i,c] <-totalsimple
                complexfit[i,c] <- simplefit
         }

}     #end of i loop for number of factors

vss.stats <- data.frame(vss.df,cfit=complexfit,cresidual=complexresid)
return(vss.stats)

    }     #end of VSS function
```