Package 'multiway'

November 13, 2017

Type Package

Title Component Models for Multi-Way Data

Version 1.0-4

Date 2017-11-13

Author Nathaniel E. Helwig <helwig@umn.edu>

Maintainer Nathaniel E. Helwig <helwig@umn.edu>

Depends parallel, quadprog

Description Fits multi-way component models via alternating least squares algorithms with optional constraints: orthogonal, non-negative, unimodal, monotonic, periodic, smooth, or structure. Fit models include Individual Differences Scaling, Parallel Factor Analysis (1 and 2), Simultaneous Component Analysis, and Tucker Factor Analysis.

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2017-11-13 10:26:26 UTC

R topics documented:

`multiway-package` *Component Models for Multi-Way Data*

Description

Fits multi-way component models via alternating least squares algorithms with optional constraints: orthogonal, non-negative, unimodal, monotonic, periodic, smooth, or structure. Fit models include Individual Differences Scaling, Parallel Factor Analysis (1 and 2), Simultaneous Component Analysis, and Tucker Factor Analysis.

Details

`indscal` fits the Individual Differences Scaling model. `parafac` fits the 3-way and 4-way Parallel Factor Analysis-1 model. `parafac2` fits the 3-way and 4-way Parallel Factor Analysis-2 model. `sca` fits the four different Simultaneous Component Analysis models. `tucker` fits the 3-way and 4-way Tucker Factor Analysis model.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Maintainer: Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics, 11*, 393-401.

Bro, R., & Kiers, H.A.L. (2003). A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics, 17*, 274-286.

Carroll, J. D., & Chang, J-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika, 35*, 283-319.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics, 16*, 1-84.

Harshman, R. A. (1972). PARAFAC2: Mathematical and technical notes. *UCLA Working Papers in Phonetics, 22*, 30-44.

Harshman, R. A., & Lundy, M. E. (1994). PARAFAC: Parallel factor analysis. *Computational Statistics and Data Analysis, 18*, 39-72.

Haughwout, S. P., LaVallee, R. A., & Castle, I-J. P. (2015). Surveillance Report #102: Apparent Per Capita Alcohol Consumption: National, State, and Regional Trends, 1977-2013. Bethesda, MD: NIAAA, Alcohol Epidemiologic Data System.

Helwig, N. E. (2013). The special sign indeterminacy of the direct-fitting Parafac2 model: Some implications, cautions, and recommendations, for Simultaneous Component Analysis. *Psychometrika, 78*, 725-739.

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803.

Kiers, H. A. L., ten Berge, J. M. F., & Bro, R. (1999). PARAFAC2-part I: A direct-fitting algorithm for the PARAFAC2 model. *Journal of Chemometrics, 13*, 275-294.

Kroonenberg, P. M., & de Leeuw, J. (1980). Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika, 45*, 69-97.

Moore, E. H. (1920). On the reciprocal of the general algebraic matrix. Bulletin of the American Mathematical Society 26, 394-395.

Nephew, T. M., Yi, H., Williams, G. D., Stinson, F. S., & Dufour, M.C., (2004). U.S. Alcohol Epidemiologic Data Reference Manual, Vol. 1, 4th ed. U.S. Apparent Consumption of Alcoholic Beverages Based on State Sales, Taxation, or Receipt Data. Bethesda, MD: NIAAA, Alcohol Epidemiologic Data System. NIH Publication No. 04-5563.

Penrose, R. (1950). A generalized inverse for matrices. Mathematical Proceedings of the Cambridge Philosophical Society 51, 406-413.

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441.

Timmerman, M. E., & Kiers, H. A. L. (2003). Four simultaneous component models for the analysis of multivariate time series from more than one subject to model intraindividual and interindividual differences. *Psychometrika, 68*, 105-121.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31*, 279-311.

Examples

```
# See examples for indscal, parafac, parafac2, sca, and tucker
```

---

congru *Tucker's Congruence Coefficient*

Description

Calculates Tucker's congruence coefficient (uncentered correlation) between `x` and `y` if these are vectors. If `x` and `y` are matrices then the congruence between the columns of `x` and `y` are computed.

Usage

```
congru(x, y = NULL)
```

Arguments

x	Numeric vector, matrix or data frame.
y	NULL (default) or a vector, matrix or data frame with compatible dimensions to x. The default is equivalent to y = x (but more efficient).

Details

Tucker's congruence coefficient is defined as

$$r = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}}$$

where x_i and y_i denote the i-th elements of x and y.

Value

Returns a scalar or matrix with congruence coefficient(s).

Note

If x is a vector, you must also enter y.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Tucker, L.R. (1951). *A method for synthesis of factor analysis studies* (Personnel Research Section Report No. 984). Washington, DC: Department of the Army.

Examples

```
##########   EXAMPLE 1   ##########

set.seed(1)
A <- rnorm(100)
B <- rnorm(100)
C <- A*5
D <- A*(-0.5)
congru(A,B)
congru(A,C)
congru(A,D)



##########   EXAMPLE 2   ##########

set.seed(1)
A <- cbind(rnorm(20),rnorm(20))
B <- cbind(A[,1]*-0.5,rnorm(20))
```

```
congru(A)
congru(A,B)
```

`const.control` *Auxiliary for Controlling Multi-Way Constraints*

Description

Auxiliary function for controlling the `const` argument of the `parafac` and `parafac2` functions. Applicable when using constraints 3 (unimodal), 4 (monotonic), 5 (periodic), and/or 6 (smooth).

Usage

```
const.control(const, df = NULL, degree = NULL, nonneg = NULL)
```

Arguments

`const`	Constraints for each mode. Vector of length 3 or 4 with entries: 0 = unconstrained (default), 1 = orthogonal, 2 = non-negative, 3 = unimodal, 4 = monotonic, 5 = periodic, 6 = smooth.
`df`	Integer vector of length 3 or 4 giving the degrees of freedom to use for the spline basis in each mode. Can also input a single number giving the common degrees of freedom to use for each mode. Defaults to 7 degrees of freedom for each applicable mode.
`degree`	Integer vector of length 3 or 4 giving the polynomial degree to use for the spline basis in each mode. Can also input a single number giving the common polynomial degree to use for each mode. Defaults to degree 3 (cubic) polynomials for each applicable mode.
`nonneg`	Logical vector of length 3 or 4 indicating whether the weights in each mode should be constrained to be non-negative. Can also input a single logical giving the common non-negativity constraints to use for each mode. Defaults to `FALSE` for each applicable mode.

Details

The `parafac` and `parafac2` functions pass the input `control` to this function to determine the fitting options when using constraints 3-6.

Value

Returns a list with elements: `const`, `df`, `degree`, and `nonneg`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE   ##########

# create random data array with Parafac structure
set.seed(4)
mydim <- c(30,10,8,10)
nf <- 4
aseq <- seq(-3,3,length=mydim[1])
Amat <- cbind(dnorm(aseq), dchisq(aseq+3.1, df=3),
              dt(aseq-2, df=4), dgamma(aseq+3.1, shape=3, rate=1))
Bmat <- svd(matrix(runif(mydim[2]*nf),mydim[2],nf))$u
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Dmat <- matrix(runif(mydim[4]*nf),mydim[4],nf)
Xmat <- array(tcrossprod(Amat,krprod(Dmat,krprod(Cmat,Bmat))),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Parafac model (unimodal A, orthogonal B, non-negative C, non-negative D)
pfac <- parafac(X,nfac=nf,nstart=1,const=c(3,1,2,2))
pfac

# same as before, but add some options to the unimodality contraints...
# fewer knots (df=5), quadratic splines (degree=2), and enforce non-negativity
cc <- const.control(c(3,1,2,2), df=5, degree=2, nonneg=TRUE)
pfac <- parafac(X,nfac=nf,nstart=1,const=c(3,1,2,2),control=cc)
pfac
```

`corcondia` *Core Consistency Diagnostic*

Description

Calculates Bro and Kiers's core consistency diagnostic (CORCONDIA) for a fit `parafac` or `parafac2` model. For Parafac2, the diagnostic is calculated after transforming the data.

Usage

```
corcondia(X, object, divisor=c("nfac","core"))
```

Arguments

`X`	Three-way data array with `dim=c(I,J,K)` or four-way data array with `dim=c(I,J,K,L)`. Can also input a list of two-way or three-way arrays (for Parafac2).
`object`	Object of class "parafac" (output from `parafac`) or class "parafac2" (output from `parafac2`).
`divisor`	Divide by number of factors (default) or core sum of squares.

Details

The core consistency diagnostic is defined as

```
100 * ( 1 - sum( (G-S)^2 ) / divisor )
```

where `G` is the least squares estimate of the Tucker core array, `S` is a super-diagonal core array, and `divisor` is the sum of squares of either `S` ("nfac") or `G` ("core"). A value of 100 indiciates a perfect multilinear structure, and smaller values indicate greater violations of multilinear structure.

Value

Returns CORCONDIA value.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & Kiers, H.A.L. (2003). A new efficient method for determining the number of components in PARAFAC models. *Journal of Chemometrics, 17*, 274-286.

Examples

```
##########   EXAMPLE   ##########

# create random data array with Parafac structure
set.seed(3)
mydim <- c(50,20,5)
nf <- 2
Amat <- matrix(rnorm(mydim[1]*nf),mydim[1],nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- array(tcrossprod(Amat,krprod(Cmat,Bmat)),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Parafac model (1-4 factors)
pfac1 <- parafac(X,nfac=1,nstart=1)
pfac2 <- parafac(X,nfac=2,nstart=1)
pfac3 <- parafac(X,nfac=3,nstart=1)
pfac4 <- parafac(X,nfac=4,nstart=1)

# check corcondia
corcondia(X, pfac1)
corcondia(X, pfac2)
corcondia(X, pfac3)
corcondia(X, pfac4)
```

`fitted` *Extract Multi-Way Fitted Values*

Description

Calculates fitted array (or list of arrays) from a `multiway` object.

Usage

```
## S3 method for class 'indscal'
fitted(object,...)
## S3 method for class 'parafac'
fitted(object,...)
## S3 method for class 'parafac2'
fitted(object,simplify=TRUE,...)
## S3 method for class 'sca'
fitted(object,...)
## S3 method for class 'tucker'
fitted(object,...)
```

Arguments

`object`	Object of class "indscal" (output from `indscal`), class "parafac" (output from `parafac`), class "parafac2" (output from `parafac2`), class "sca" (output from `sca`), or class "tucker" (output from `tucker`).
`simplify`	For "parafac2", setting `simplify=FALSE` will always return a list of fitted arrays. Default of `simplify=TRUE` returns a fitted array if all levels of the nesting mode have the same number of observations (and a list of fitted arrays otherwise).
`...`	Ignored.

Details

See `indscal`, `parafac`, `parafac2`, `sca`, and `tucker` for more details.

Value

"indscal" objects: 3-way array.

"parafac" objects: 3-way or 4-way array.

"parafac2" objects: 3-way or 4-way array (if possible and simplify=TRUE); otherwise list of 2-way or 3-way arrays.

"sca" objects: list of 2-way arrays.

"tucker" objects: 3-way or 4-way array.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
### see examples for indscal, parafac, parafac2, sca, and tucker
```

`fnnls` *Fast Non-Negative Least Squares*

Description

Finds the vector `b` minimizing

```
sum( (y - X%*%b)^2 )
```

subject to `b[j] >= 0` for all `j`.

Usage

```
fnnls(XtX,Xty,ntol=NULL)
```

Arguments

`XtX`	Crossproduct matrix `crossprod(X)` of dimension p-by-p.
`Xty`	Crossproduct vector `crossprod(X,y)` of length p-by-1.
`ntol`	Tolerance for non-negativity.

Value

The vector `b` such that `b[j] >= 0` for all `j`.

Note

Default non-negativity tolerance: `ntol=10*(.Machine$double.eps)*max(colSums(abs(XtX)))*p`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics, 11*, 393-401.

Examples

```
##########   EXAMPLE 1   ##########
X <- matrix(1:100,50,2)
y <- matrix(101:150,50,1)
beta <- solve(crossprod(X))%*%crossprod(X,y)
beta
beta <- fnnls(crossprod(X),crossprod(X,y))
beta


##########   EXAMPLE 2   ##########
X <- cbind(-(1:50),51:100)
y <- matrix(101:150,50,1)
beta <- solve(crossprod(X))%*%crossprod(X,y)
beta
beta <- fnnls(crossprod(X),crossprod(X,y))
beta


##########   EXAMPLE 3   ##########
X <- matrix(rnorm(400),100,4)
btrue <- c(1,2,0,7)
y <- X%*%btrue + rnorm(100)
fnnls(crossprod(X),crossprod(X,y))


##########   EXAMPLE 4   ##########
X <- matrix(rnorm(2000),100,20)
btrue <- runif(20)
y <- X%*%btrue + rnorm(100)
beta <- fnnls(crossprod(X),crossprod(X,y))
crossprod(btrue-beta)/20
```

`indscal` *Individual Differences Scaling*

Description

Given a 3-way array `X = array(x,dim=c(J,J,K))` with `X[,,k]` denoting the k-th subject's dissimilarity matrix rating `J` objects, the INDSCAL model can be written as

```
Z[i,j,k] = sum B[i,r]*B[j,r]*C[k,r] + E[i,j,k]
```

where `Z` is the array of scalar products obtained from `X`, `B = matrix(b,J,R)` are the object weights, `C = matrix(c,K,R)` are the non-negative subject weights, and `E = array(e,dim=c(J,J,K))` is the 3-way residual array. The summation is for `r = seq(1,R)`.

Weight matrices are estimated using an alternating least squares algorithm with optional constraints.

Usage

```
indscal(X,nfac,nstart=10,const=NULL,maxit=500,
        type=c("dissimilarity","similarity"),
        ctol=1e-4,parallel=FALSE,cl=NULL,
        output=c("best","all"))
```

Arguments

X	Three-way data array with dim=c(J,J,K) where X[,,k] is dissimilarity matrix. Can also input a list of (dis)similarity matrices or objects output by dist.
nfac	Number of factors.
nstart	Number of random starts.
const	Constraints for Modes B and C. See Note.
maxit	Maximum number of iterations.
type	Character indicating if X contains dissimilarity data (default) or similarity data.
ctol	Convergence tolerance.
parallel	Logical indicating if parLapply should be used. See Examples.
cl	Cluster created by makeCluster. Only used when parallel=TRUE.
output	Output the best solution (default) or output all nstart solutions.

Value

If output="best", returns an object of class "indscal" with the following elements:

B	Mode B weight matrix.
C	Mode C weight matrix.
SSE	Sum of Squared Errors.
Rsq	R-squared value.
GCV	Generalized Cross-Validation.
edf	Effective degrees of freedom.
iter	Number of iterations.
cflag	Convergence flag.
const	See argument const.

Otherwise returns a list of length nstart where each element is an object of class "indscal".

Warnings

The ALS algorithm can perform poorly if the number of factors nfac is set too large.

Default is unconstrained ALS update, which may produce negative (invalid) Mode C weights. Use const=c(0,2) to force non-negativity via fnnls.

Note

Default use is 10 random strarts (`nstart=10`) with 500 maximum iterations of the ALS algorithm for each start (`maxit=500`) using a convergence tolerance of 1e-4 (`ctol=1e-4`). The algorithm is determined to have converged once the change in R^2 is less than or equal to `ctol`.

Input `const` should be a two element integer vector giving constraints for Modes B and C. There are four possible options:

`const=c(0,0)`	Unconstrained update for Modes B and C
`const=c(0,2)`	Unconstrained Mode B with non-negative Mode C
`const=c(1,0)`	Orthogonal Mode B with unconstrained Mode C
`const=c(1,2)`	Orthogonal Mode B with non-negative Mode C

Default is unconstrained update for all modes, i.e., `const=c(0,0)`.

Output `cflag` gives convergence information: `cflag=0` if ALS algorithm converged normally, `cflag=1` if maximum iteration limit was reached before convergence, and `cflag=2` if ALS algorithm terminated abnormally due to problem with non-negativity constraints.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics, 11*, 393-401.

Carroll, J. D., & Chang, J-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition. *Psychometrika, 35*, 283-319.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics, 16*, 1-84.

Harshman, R. A., & Lundy, M. E. (1994). PARAFAC: Parallel factor analysis. *Computational Statistics and Data Analysis, 18*, 39-72.

Examples

```
##########   array example   ##########

# create random data array with INDSCAL structure
set.seed(3)
mydim <- c(50,5,10)
nf <- 2
X <- array(0,dim=c(rep(mydim[2],2),mydim[3]))
for(k in 1:mydim[3]) {
  X[,,k] <- as.matrix(dist(t(matrix(rnorm(prod(mydim[1:2])),mydim[1],mydim[2]))))
}

# fit INDSCAL model (unconstrained)
imod <- indscal(X,nfac=nf,nstart=1)
```

```
imod

# check solution
Xhat <- fitted(imod)
sum((array(apply(X,3,ed2sp),dim=dim(X))-Xhat)^2)
imod$SSE

# reorder and resign factors
imod$B[1:4,]
imod <- reorder(imod, 2:1)
imod$B[1:4,]
imod <- resign(imod, newsign=c(1,-1))
imod$B[1:4,]
sum((array(apply(X,3,ed2sp),dim=dim(X))-Xhat)^2)
imod$SSE

# rescale factors
colSums(imod$B^2)
colSums(imod$C^2)
imod <- rescale(imod, mode="C")
colSums(imod$B^2)
colSums(imod$C^2)
sum((array(apply(X,3,ed2sp),dim=dim(X))-Xhat)^2)
imod$SSE


##########   list example   ##########

# create random data array with INDSCAL structure
set.seed(4)
mydim <- c(100,8,20)
nf <- 3
X <- vector("list",mydim[3])
for(k in 1:mydim[3]) {
  X[[k]] <- dist(t(matrix(rnorm(prod(mydim[1:2])),mydim[1],mydim[2])))
}

# fit INDSCAL model (orthogonal B, non-negative C)
imod <- indscal(X,nfac=nf,nstart=1,const=c(1,2))
imod

# check solution
Xhat <- fitted(imod)
sum((array(unlist(lapply(X,ed2sp)),dim=mydim[c(2,2,3)])-Xhat)^2)
imod$SSE
crossprod(imod$B)


## Not run:

##########   parallel computation   ##########

# create random data array with INDSCAL structure
```

```
set.seed(3)
mydim <- c(50,5,10)
nf <- 2
X <- array(0,dim=c(rep(mydim[2],2),mydim[3]))
for(k in 1:mydim[3]) {
  X[,,k] <- as.matrix(dist(t(matrix(rnorm(prod(mydim[1:2])),mydim[1],mydim[2]))))
}

# fit INDSCAL model (10 random starts -- sequential computation)
set.seed(1)
system.time({imod <- indscal(X,nfac=nf)})
imod

# fit INDSCAL model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({imod <- indscal(X,nfac=nf,parallel=TRUE,cl=cl)})
imod
stopCluster(cl)

## End(Not run)
```

---

krprod *Khatri-Rao Product*

Description

Given `X` (n-by-p) and `Y` (m-by-p), the Khatri-Rao product `Z = krprod(X,Y)` is defined as

```
Z[,j] = kronecker(X[,j],Y[,j])
```

which is the mn-by-p matrix containing Kronecker products of corresponding columns of `X` and `Y`.

Usage

```
krprod(X,Y)
```

Arguments

`X`	Matrix of order n-by-p.
`Y`	Matrix of order m-by-p.

Value

The mn-by-p matrix of columnwise Kronecker products.

Note

X and Y must have the same number of columns.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE 1   ##########
X <- matrix(1,4,2)
Y <- matrix(1:4,2,2)
krprod(X,Y)


##########   EXAMPLE 2   ##########
X <- matrix(1:2,4,2)
Y <- matrix(1:4,2,2)
krprod(X,Y)


##########   EXAMPLE 3   ##########
X <- matrix(1:2,4,2,byrow=TRUE)
Y <- matrix(1:4,2,2)
krprod(X,Y)
```

---

mpinv *Moore-Penrose Pseudoinverse*

Description

Calculates the Moore-Penrose pseudoinverse of the input matrix using a truncated singular value decomposition.

Usage

```
mpinv(X, tol = NULL)
```

Arguments

X	Real-valued matrix.
tol	Stability tolerance for singular values.

Value

Returns pseudoinverse of X.

Note

Default tolerance is `tol = max(dim(X)) * .Machine$double.eps`.

Note

Basically returns `Y$v %*% diag(1/Y$d) %*% t(Y$u)` where `Y = svd(X)`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Moore, E.H. (1920). On the reciprocal of the general algebraic matrix. Bulletin of the American Mathematical Society 26, 394-395.

Penrose, R. (1950). A generalized inverse for matrices. Mathematical Proceedings of the Cambridge Philosophical Society 51, 406-413.

Examples

```
##########   EXAMPLE   ##########

set.seed(1)
X <- matrix(rnorm(2000),100,20)
Xi <- mpinv(X)
sum( ( X - X %*% Xi %*% X )^2 )
sum( ( Xi - Xi %*% X %*% Xi )^2 )
isSymmetric(X %*% Xi)
isSymmetric(Xi %*% X)
```

---

ncenter *Center n-th Dimension of Array*

Description

Fiber-center across the levels of the specified mode. Can input 2-way, 3-way, and 4-way arrays, or input a list containing array elements.

With `X` a matrix (I-by-J) there are two options:

```
mode=1:       x[i,j] - mean(x[,j])
mode=2:       x[i,j] - mean(x[i,])
```

With `X` a 3-way array (I-by-J-by-K) there are three options:

```
mode=1:       x[i,j,k] - mean(x[,j,k])
```

```
mode=2:       x[i,j,k] - mean(x[i,,k])
mode=3:       x[i,j,k] - mean(x[i,j,])
```

With X a 4-way array (I-by-J-by-K-by-L) there are four options:

```
mode=1:       x[i,j,k,l] - mean(x[,j,k,l])
mode=2:       x[i,j,k,l] - mean(x[i,,k,l])
mode=3:       x[i,j,k,l] - mean(x[i,j,,l])
mode=4:       x[i,j,k,l] - mean(x[i,j,k,])
```

Usage

```
ncenter(X,mode=1)
```

Arguments

X	Array (2-way, 3-way, or 4-way) or a list containing array elements.
mode	Mode to center across.

Value

Returns centered version of X.

Note

When entering a list with array elements, each element must be an array (2-way, 3-way, or 4-way) that contains the specified mode to center across.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE 1   ##########
X <- matrix(rnorm(2000),100,20)
Xc <- ncenter(X)          # center across rows
sum(colSums(Xc))
Xc <- ncenter(Xc,mode=2) # recenter across columns
sum(colSums(Xc))
sum(rowSums(Xc))


##########   EXAMPLE 2   ##########
X <- array(rnorm(20000),dim=c(100,20,10))
Xc <- ncenter(X,mode=2)   # center across columns
sum(rowSums(Xc))
```

```
##########   EXAMPLE 3   ##########
X <- array(rnorm(100000),dim=c(100,20,10,5))
Xc <- ncenter(X,mode=4)   # center across 4-th mode
sum(rowSums(Xc))


##########   EXAMPLE 4   ##########
X <- replicate(5,array(rnorm(20000),dim=c(100,20,10)),simplify=FALSE)
Xc <- ncenter(X)
sum(colSums(Xc[[1]]))
```

---

nscale *Scale n-th Dimension of Array*

Description

Slab-scale within each level of the specified mode. Can input 2-way, 3-way, and 4-way arrays, or input a list containing array elements (see Note).

With X a matrix (I-by-J) there are two options:

```
mode=1:       x[i,j]*sqrt(ssnew/sumsq(x[i,]))
mode=2:       x[i,j]*sqrt(ssnew/sumsq(x[,j]))
```

With X a 3-way array (I-by-J-by-K) there are three options:

```
mode=1:       x[i,j,k]*sqrt(ssnew/sumsq(x[i,,]))
mode=2:       x[i,j,k]*sqrt(ssnew/sumsq(x[,j,]))
mode=3:       x[i,j,k]*sqrt(ssnew/sumsq(x[,,k]))
```

With X a 4-way array (I-by-J-by-K-by-L) there are four options:

```
mode=1:       x[i,j,k,l]*sqrt(ssnew/sumsq(x[i,,,]))
mode=2:       x[i,j,k,l]*sqrt(ssnew/sumsq(x[,j,,]))
mode=3:       x[i,j,k,l]*sqrt(ssnew/sumsq(x[,,k,]))
mode=4:       x[i,j,k,l]*sqrt(ssnew/sumsq(x[,,,l]))
```

Usage

```
nscale(X,mode=1,ssnew=1)
nrescale(X,mode=1,ssnew=1)
```

Arguments

X	Array (2-way, 3-way, or 4-way) or a list containing array elements.
mode	Mode to scale within (set mode=0 to scale across all modes).
ssnew	Desired sum-of-squares for each level of scaled mode.

Value

Returns scaled version of X.

Note

When entering a list with array elements, each element must be a 2-way or 3-way array. The list elements are treated as the 3rd mode (for list of 2-way arrays) or the 4th mode (for list of 3-way arrays) in the formulas provided in the Description.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE 1   ##########
X <- matrix(rnorm(2000),100,20)
Xr <- nscale(X,mode=2)          # scale columns to ssnew=1
colSums(Xr^2)
Xr <- nscale(X,mode=2,ssnew=2)  # scale columns to ssnew=2
colSums(Xr^2)


##########   EXAMPLE 2   ##########
Xold <- X <- matrix(rnorm(400),20,20)
iter <- 0
chk <- 1
# iterative rescaling of modes 1 and 2
while(iter<500 & chk>=10^-9){
  Xr <- nscale(Xold,mode=1)
  Xr <- nscale(Xr,mode=2)
  chk <- sum((Xold-Xr)^2)
  Xold <- Xr
  iter <- iter + 1
}
iter
rowSums(Xr^2)
colSums(Xr^2)


##########   EXAMPLE 3   ##########
X <- array(rnorm(20000),dim=c(100,20,10))
Xc <- nscale(X,mode=2)   # scale within columns
rowSums(aperm(Xc,perm=c(2,1,3))^2)
```

```
##########   EXAMPLE 4   ##########
X <- array(rnorm(100000),dim=c(100,20,10,5))
Xc <- nscale(X,mode=4)   # scale across 4-th mode
rowSums(aperm(Xc,perm=c(4,1,2,3))^2)


##########   EXAMPLE 5   ##########
X <- replicate(5,array(rnorm(20000),dim=c(100,20,10)),simplify=FALSE)
Xc <- nscale(X,ssnew=(20*10*5))   # mean square of 1
rowSums(sapply(Xc, function(x) rowSums(x^2))) / (20*10*5)
```

`parafac` *Parallel Factor Analysis-1*

Description

Given a 3-way array `X = array(x,dim=c(I,J,K))`, the 3-way Parafac model can be written as

```
X[i,j,k] = sum A[i,r]*B[j,r]*C[k,r] + E[i,j,k]
```

where `A = matrix(a,I,R)` are the Mode A (first mode) weights, `B = matrix(b,J,R)` are the Mode B (second mode) weights, `C = matrix(c,K,R)` are the Mode C (third mode) weights, and `E = array(e,dim=c(I,J,K))` is the 3-way residual array. The summation is for `r = seq(1,R)`.

Given a 4-way array `X = array(x,dim=c(I,J,K,L))`, the 4-way Parafac model can be written as

```
X[i,j,k,l] = sum A[i,r]*B[j,r]*C[k,r]*D[l,r] + E[i,j,k,l]
```

where `D = matrix(d,L,R)` are the Mode D (fourth mode) weights, `E = array(e,dim=c(I,J,K,L))` is the 4-way residual array, and the other terms can be interprered as previously described.

Weight matrices are estimated using an alternating least squares algorithm with optional constraints.

Usage

```
parafac(X, nfac, nstart = 10, const = NULL, control = NULL,
        Bfixed = NULL, Cfixed = NULL, Dfixed = NULL,
        Bstart = NULL, Cstart = NULL, Dstart = NULL,
        Bstruc = NULL, Cstruc = NULL, Dstruc = NULL,
        maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
        output = c("best", "all"), verbose = FALSE)
```

Arguments

`X`	Three-way data array with `dim=c(I,J,K)` or four-way data array with `dim=c(I,J,K,L)`. Missing data are allowed (see Note).

nfac	Number of factors.
nstart	Number of random starts.
const	Constraints for each mode. Vector of length 3 or 4 with entries: 0 = unconstrained (default), 1 = orthogonal, 2 = non-negative, 3 = unimodal, 4 = monotonic, 5 = periodic, 6 = smooth. Use control argument to adjust options for constraints 3-6.
control	List of parameters controlling options for constraints 3-6. This is passed to const.control, which describes the available options.
Bfixed	Fixed Mode B weights. Only used to fit model with fixed Mode B weights.
Cfixed	Fixed Mode C weights. Only used to fit model with fixed Mode C weights.
Dfixed	Fixed Mode D weights. Only used to fit model with fixed Mode D weights.
Bstart	Starting Mode B weights for ALS algorithm. Default uses random weights.
Cstart	Starting Mode C weights for ALS algorithm. Default uses random weights.
Dstart	Starting Mode D weights for ALS algorithm. Default uses random weights.
Bstruc	Structure constraints for Mode B weights. Default uses unstructured weights.
Cstruc	Structure constraints for Mode C weights. Default uses unstructured weights.
Dstruc	Structure constraints for Mode D weights. Default uses unstructured weights.
maxit	Maximum number of iterations.
ctol	Convergence tolerance (R^2 change).
parallel	Logical indicating if parLapply should be used. See Examples.
cl	Cluster created by makeCluster. Only used when parallel=TRUE.
output	Output the best solution (default) or output all nstart solutions.
verbose	Logical indicating if extra information on progress should be reported. Ignored if parallel=TRUE.

Value

If output="best", returns an object of class "parafac" with the following elements:

A	Mode A weight matrix.
B	Mode B weight matrix.
C	Mode C weight matrix.
D	Mode D weight matrix.
SSE	Sum of Squared Errors.
Rsq	R-squared value.
GCV	Generalized Cross-Validation.
edf	Effective degrees of freedom.
iter	Number of iterations.
cflag	Convergence flag.
const	See argument const.

control	See argument control.
fixed	Logical vector indicating whether 'fixed' weights were used for each mode.
struc	Logical vector indicating whether 'struc' constraints were used for each mode.

Otherwise returns a list of length nstart where each element is an object of class "parafac".

Warnings

The ALS algorithm can perform poorly if the number of factors nfac is set too large.

Non-negativity constraints can be sensitive to local optima.

Non-negativity constraints can result in slower performance.

Structure constraints for override constraints in const input.

Note

Default use is 10 random strarts (nstart=10) with 500 maximum iterations of the ALS algorithm for each start (maxit=500) using a convergence tolerance of 1e-4 (ctol=1e-4). The algorithm is determined to have converged once the change in R^2 is less than or equal to ctol.

Output cflag gives convergence information: cflag=0 if ALS algorithm converged normally, cflag=1 if maximum iteration limit was reached before convergence, and cflag=2 if ALS algorithm terminated abnormally due to a problem with the constraints.

Constraints 3 (unimodality) and 4 (monotonicity) are implemented using I-splines, and constraints 5 (periodicity) and 6 (smoothness) are implemented using M-splines (see Ramsay, 1988).

Missing data should be specified as NA values in the input X. The missing data are randomly initialized and then iteratively imputed as a part of the ALS algorithm.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics, 11*, 393-401.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics, 16*, 1-84.

Harshman, R. A., & Lundy, M. E. (1994). PARAFAC: Parallel factor analysis. *Computational Statistics and Data Analysis, 18*, 39-72.

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803.

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441.

Examples

```
##########   3-way example   ##########

# create random data array with Parafac structure
set.seed(3)
mydim <- c(50,20,5)
nf <- 3
Amat <- matrix(rnorm(mydim[1]*nf),mydim[1],nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- array(tcrossprod(Amat,krprod(Cmat,Bmat)),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Parafac model (unconstrained)
pfac <- parafac(X,nfac=nf,nstart=1)
pfac

# fit Parafac model (non-negativity on Modes B and C)
pfacNN <- parafac(X,nfac=nf,nstart=1,const=c(0,2,2))
pfacNN

# check solution
Xhat <- fitted(pfac)
sum((Xmat-Xhat)^2)/prod(mydim)

# reorder and resign factors
pfac$B[1:4,]
pfac <- reorder(pfac, c(3,1,2))
pfac$B[1:4,]
pfac <- resign(pfac, mode="B")
pfac$B[1:4,]
Xhat <- fitted(pfac)
sum((Xmat-Xhat)^2)/prod(mydim)

# rescale factors
colSums(pfac$B^2)
colSums(pfac$C^2)
pfac <- rescale(pfac, mode="C", absorb="B")
colSums(pfac$B^2)
colSums(pfac$C^2)
Xhat <- fitted(pfac)
sum((Xmat-Xhat)^2)/prod(mydim)


##########   4-way example   ##########

# create random data array with Parafac structure
set.seed(4)
mydim <- c(30,10,8,10)
nf <- 4
```

```
aseq <- seq(-3,3,length=mydim[1])
Amat <- cbind(dnorm(aseq), dchisq(aseq+3.1, df=3),
              dt(aseq-2, df=4), dgamma(aseq+3.1, shape=3, rate=1))
Bmat <- svd(matrix(runif(mydim[2]*nf),mydim[2],nf))$u
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Dmat <- matrix(runif(mydim[4]*nf),mydim[4],nf)
Xmat <- array(tcrossprod(Amat,krprod(Dmat,krprod(Cmat,Bmat))),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Parafac model (unimodal A, orthogonal B, non-negative C, non-negative D)
pfac <- parafac(X,nfac=nf,nstart=1,const=c(3,1,2,2))
pfac

# check solution
Xhat <- fitted(pfac)
sum((Xmat-Xhat)^2)/prod(mydim)
congru(Amat, pfac$A)
crossprod(pfac$B)
pfac$C


## Not run:

##########   parallel computation   ##########

# create random data array with Parafac structure
set.seed(3)
mydim <- c(50,20,5)
nf <- 3
Amat <- matrix(rnorm(mydim[1]*nf),mydim[1],nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- array(tcrossprod(Amat,krprod(Cmat,Bmat)),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Parafac model (10 random starts -- sequential computation)
set.seed(1)
system.time({pfac <- parafac(X,nfac=nf)})
pfac

# fit Parafac model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({pfac <- parafac(X,nfac=nf,parallel=TRUE,cl=cl)})
pfac
stopCluster(cl)

## End(Not run)
```

parafac2 *Parallel Factor Analysis-2*

Description

Given a list of matrices `X[[k]] = matrix(xk,I[k],J)` for `k = seq(1,K)`, the 3-way Parafac2 model (with Mode A nested in Mode C) can be written as

```
X[[k]] = tcrossprod(A[[k]]%*%diag(C[k,]),B) + E[[k]]
                        subject to crossprod(A[[k]]) = Phi
```

where `A[[k]] = matrix(ak,I[k],R)` are the Mode A (first mode) weights for the k-th level of Mode C (third mode), `Phi` is the common crossproduct matrix shared by all `K` levels of Mode C, `B = matrix(b,J,R)` are the Mode B (second mode) weights, `C = matrix(c,K,R)` are the Mode C (third mode) weights, and `E[[k]] = matrix(ek,I[k],J)` is the residual matrix corresponding to k-th level of Mode C.

Given a list of arrays `X[[l]] = array(xl,dim=c(I[l],J,K))` for `l = seq(1,L)`, the 4-way Parafac2 model (with Mode A nested in Mode D) can be written as

```
X[[l]][,,k] = tcrossprod(A[[l]]%*%diag(D[l,]*C[k,]),B) + E[[k]]
                                    subject to crossprod(A[[l]]) = Phi
```

`A[[l]] = matrix(al,I[l],R)` are the Mode A (first mode) weights for the `l`-th level of Mode D (fourth mode), `Phi` is the common crossproduct matrix shared by all `L` levels of Mode D, `D = matrix(d,L,R)` are the Mode D (fourth mode) weights, and `E[[l]] = matrix(el,I[l],J,K)` is the residual array corresponding to `l`-th level of Mode D.

Weight matrices are estimated using an alternating least squares algorithm with optional constraints.

Usage

```
parafac2(X, nfac, nstart = 10, const = NULL, control = NULL,
         Gfixed = NULL, Bfixed = NULL, Cfixed = NULL, Dfixed = NULL,
         Gstart = NULL, Bstart = NULL, Cstart = NULL, Dstart = NULL,
         Gstruc = NULL, Bstruc = NULL, Cstruc = NULL, Dstruc = NULL,
         maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
         output = c("best", "all"), verbose = FALSE)
```

Arguments

`X`	For 3-way Parafac2: list of length `K` where k-th element is `I[k]`-by-`J` matrix or three-way data array with `dim=c(I,J,K)`. For 4-way Parafac2: list of

	length L where l-th element is I[l]-by-J-by-K array or four-way data array with dim=c(I,J,K,L). Missing data are allowed (see Note).
nfac	Number of factors.
nstart	Number of random starts.
const	Constraints for each mode. Vector of length 3 or 4 with entries: 0 = unconstrained (default), 1 = orthogonal, 2 = non-negative, 3 = unimodal, 4 = monotonic, 5 = periodic, 6 = smooth. Use control argument to adjust options for constraints 3-6. Note: constraints 2-4 cannot be applied on Mode A.
control	List of parameters controlling options for constraints 3-6. This is passed to const.control, which describes the available options.
Gfixed	Fixed Mode A crossproducts (crossprod(Gfixed)=Phi). Only used to fit model with fixed Phi matrix.
Bfixed	Fixed Mode B weights. Only used to fit model with fixed Mode B weights.
Cfixed	Fixed Mode C weights. Only used to fit model with fixed Mode C weights.
Dfixed	Fixed Mode D weights. Only used to fit model with fixed Mode D weights.
Gstart	Starting Mode A crossproduct matrix for ALS algorithm (crossprod(Gstart)=Phi). Default uses random weights.
Bstart	Starting Mode B weights for ALS algorithm. Default uses random weights.
Cstart	Starting Mode C weights for ALS algorithm. Default uses random weights.
Dstart	Starting Mode D weights for ALS algorithm. Default uses random weights.
Gstruc	Structure constraints for Mode A crossproduct matrix (crossprod(Gstruc) = Phi structure). Default uses unstructured crossproducts.
Bstruc	Structure constraints for Mode B weights. Default uses unstructured weights.
Cstruc	Structure constraints for Mode C weights. Default uses unstructured weights.
Dstruc	Structure constraints for Mode D weights. Default uses unstructured weights.
maxit	Maximum number of iterations.
ctol	Convergence tolerance.
parallel	Logical indicating if parLapply should be used. See Examples.
cl	Cluster created by makeCluster. Only used when parallel=TRUE.
output	Output the best solution (default) or output all nstart solutions.
verbose	Logical indicating if extra information on progress should be reported. Ignored if parallel=TRUE.

Value

If output="best", returns an object of class "parafac2" with the following elements:

A	List of Mode A weight matrices.
B	Mode B weight matrix.
C	Mode C weight matrix.
D	Mode D weight matrix.

Phi	Mode A crossproduct matrix.
SSE	Sum of Squared Errors.
Rsq	R-squared value.
GCV	Generalized Cross-Validation.
edf	Effective degrees of freedom.
iter	Number of iterations.
cflag	Convergence flag.
const	See argument const.
control	See argument control.
fixed	Logical vector indicating whether 'fixed' weights were used for each mode.
struc	Logical vector indicating whether 'struc' constraints were used for each mode.

Otherwise returns a list of length nstart where each element is an object of class "parafac2".

Warnings

The ALS algorithm can perform poorly if the number of factors nfac is set too large.

Non-negativity constraints can be sensitive to local optima.

Non-negativity constraints can result in slower performance.

Structure constraints for override constraints in const input.

Note

Default use is 10 random strarts (nstart=10) with 500 maximum iterations of the ALS algorithm for each start (maxit=500) using a convergence tolerance of 1e-4 (ctol=1e-4). The algorithm is determined to have converged once the change in R^2 is less than or equal to ctol.

Output cflag gives convergence information: cflag=0 if ALS algorithm converged normally, cflag=1 if maximum iteration limit was reached before convergence, and cflag=2 if ALS algorithm terminated abnormally due to a problem with the constraints.

Constraints 3 (unimodality) and 4 (monotonicity) are implemented using I-splines, and constraints 5 (periodicity) and 6 (smoothness) are implemented using M-splines (see Ramsay, 1988).

Missing data should be specified as NA values in the input X. The missing data are randomly initialized and then iteratively imputed as a part of the ALS algorithm.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Bro, R., & De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics, 11*, 393-401.

Harshman, R. A. (1972). PARAFAC2: Mathematical and technical notes. *UCLA Working Papers in Phonetics, 22*, 30-44.

Helwig, N. E. (2013). The special sign indeterminacy of the direct-fitting Parafac2 model: Some implications, cautions, and recommendations, for Simultaneous Component Analysis. *Psychometrika, 78*, 725-739.

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803.

Kiers, H. A. L., ten Berge, J. M. F., & Bro, R. (1999). PARAFAC2-part I: A direct-fitting algorithm for the PARAFAC2 model. *Journal of Chemometrics, 13*, 275-294.

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441.

Examples

```
##########   3-way example   ##########

# create random data list with Parafac2 structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- matrix(rnorm(nf^2),nf,nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- Emat <- Hmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Hmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Hmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit Parafac2 model (unconstrained)
pfac <- parafac2(X,nfac=nf,nstart=1)
pfac

# check solution
Xhat <- fitted(pfac)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])
crossprod(pfac$A[[1]])
crossprod(pfac$A[[2]])
pfac$Phi

# reorder and resign factors
pfac$B[1:4,]
pfac <- reorder(pfac, 2:1)
pfac$B[1:4,]
pfac <- resign(pfac, mode="B")
pfac$B[1:4,]
Xhat <- fitted(pfac)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])
```

```
# rescale factors
colSums(pfac$B^2)
colSums(pfac$C^2)
pfac <- rescale(pfac, mode="C", absorb="B")
colSums(pfac$B^2)
colSums(pfac$C^2)
Xhat <- fitted(pfac)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])


##########   4-way example   ##########

# create random data list with Parafac2 structure
set.seed(4)
mydim <- c(NA,10,20,5)
nf <- 3
nk <- sample(c(50,100,200),mydim[4],replace=TRUE)
Gmat <- matrix(rnorm(nf^2),nf,nf)
Bmat <- scale(matrix(rnorm(mydim[2]*nf),mydim[2],nf), center=FALSE)
cseq <- seq(-3, 3, length=mydim[3])
Cmat <- cbind(pnorm(cseq), pgamma(cseq+3.1, shape=1, rate=1)*(3/4), pt(cseq-2, df=4)*2)
Dmat <- scale(matrix(runif(mydim[4]*nf)*2,mydim[4],nf), center=FALSE)
Xmat <- Emat <- Hmat <- vector("list",mydim[4])
for(k in 1:mydim[4]){
  aseq <- seq(-3,3,length=nk[k])
  Hmat[[k]] <- svd(cbind(sin(aseq), sin(abs(aseq)), exp(-aseq^2)),nv=0)$u
  Xmat[[k]] <- array(tcrossprod(Hmat[[k]]%*%Gmat%*%diag(Dmat[k,]),
                                krprod(Cmat,Bmat)),dim=c(nk[k],mydim[2],mydim[3]))
  Emat[[k]] <- array(rnorm(nk[k]*mydim[2]*mydim[3]),dim=c(nk[k],mydim[2],mydim[3]))
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit Parafac model (smooth A, unconstrained B, monotonic C, non-negative D)
pfac <- parafac2(X,nfac=nf,nstart=1,const=c(6,0,4,2))
pfac

# check solution
Xhat <- fitted(pfac)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2]*mydim[3])
crossprod(pfac$A[[1]])
crossprod(pfac$A[[2]])
pfac$Phi


## Not run:

##########   parallel computation   ##########

# create random data list with Parafac2 structure
```

```
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- matrix(rnorm(nf^2),nf,nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- Emat <- Hmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Hmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Hmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit Parafac2 model (10 random starts -- sequential computation)
set.seed(1)
system.time({pfac <- parafac2(X,nfac=nf)})
pfac

# fit Parafac2 model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({pfac <- parafac2(X,nfac=nf,parallel=TRUE,cl=cl)})
pfac
stopCluster(cl)

## End(Not run)
```

`print` *Print Multi-Way Model*

Description

Prints constraint, fit, and convergence details for a fit `multiway` model.

Usage

```
## S3 method for class 'indscal'
print(x,...)
## S3 method for class 'parafac'
print(x,...)
## S3 method for class 'parafac2'
print(x,...)
## S3 method for class 'sca'
print(x,...)
## S3 method for class 'tucker'
print(x,...)
```

Arguments

x	Object of class "indscal" (output from indscal), class "parafac" (output from parafac), class "parafac2" (output from parafac2), class "sca" (output from sca), or class "tucker" (output from tucker).
...	Ignored.

Details

See indscal, parafac, parafac2, sca, and tucker for examples.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
### see examples for indscal, parafac, parafac2, sca, and tucker
```

---

reorder *Reorder Multi-Way Factors*

Description

Reorders factors from a multiway object.

Usage

```
## S3 method for class 'indscal'
reorder(x, neworder, ...)
## S3 method for class 'parafac'
reorder(x, neworder, ...)
## S3 method for class 'parafac2'
reorder(x, neworder, ...)
## S3 method for class 'sca'
reorder(x, neworder, ...)
## S3 method for class 'tucker'
reorder(x, neworder, mode="A", ...)
```

Arguments

x	Object of class "indscal" (output from indscal), class "parafac" (output from parafac), class "parafac2" (output from parafac2), class "sca" (output from sca), or class "tucker" (output from tucker).
neworder	Vector specifying the new factor ordering. Must be a permutation of the integers 1 to nfac.

mode	Character indicating which mode to reorder (only for tucker models). For 3-way Tucker options include "A", "B", and "C". For 4-way Tucker, options are "A", "B", "C", and "D".
...	Ignored.

Details

See indscal, parafac, parafac2, sca, and tucker for more details.

Value

Same as input.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
### see examples for indscal, parafac, parafac2, sca, and tucker
```

---

rescale *Rescales Multi-Way Factors*

Description

Rescales factors from a multiway object.

Usage

```
## S3 method for class 'indscal'
rescale(x, mode="B", newscale=1, ...)
## S3 method for class 'parafac'
rescale(x, mode="A", newscale=1, absorb="C", ...)
## S3 method for class 'parafac2'
rescale(x, mode="A", newscale=1, absorb="C", ...)
## S3 method for class 'sca'
rescale(x, mode="B", newscale=1, ...)
## S3 method for class 'tucker'
rescale(x, mode="A", newscale=1, ...)
```

Arguments

x	Object of class "indscal" (output from indscal), class "parafac" (output from parafac), class "parafac2" (output from parafac2), class "sca" (output from sca), or class "tucker" (output from tucker).
mode	Character indicating which mode to rescale.
newscale	Desired root mean-square for each column of rescaled mode. Can input a scalar or a vector with length equal to the number of factors for the given mode.
absorb	Character indicating which mode should absorb the inverse of the rescalings applied to mode (cannot be equal to mode).
...	Ignored.

Details

See indscal, parafac, parafac2, sca, and tucker for more details.

Value

Same as input.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2013). The special sign indeterminacy of the direct-fitting Parafac2 model: Some implications, cautions, and recommendations, for Simultaneous Component Analysis. *Psychometrika, 78*, 725-739.

Examples

```
### see examples for indscal, parafac, parafac2, sca, and tucker
```

---

resign *Resigns Multi-Way Factors*

Description

Resigns factors from a multiway object.

Usage

```
## S3 method for class 'indscal'
resign(x, mode="B", newsign=1, ...)
## S3 method for class 'parafac'
resign(x, mode="A", newsign=1, absorb="C", ...)
## S3 method for class 'parafac2'
resign(x, mode="A", newsign=1, absorb="C", method="pearson", ...)
## S3 method for class 'sca'
resign(x, mode="B", newsign=1, ...)
## S3 method for class 'tucker'
resign(x, mode="A",newsign=1, ...)
```

Arguments

x	Object of class "indscal" (output from indscal), class "parafac" (output from parafac), class "parafac2" (output from parafac2), class "sca" (output from sca), or class "tucker" (output from tucker).
mode	Character indicating which mode to resign.
newsign	Desired sign for each column of resigned mode. Can input a scalar or a vector with length equal to the number of factors for the given mode. If x is of class "parafac2" and mode="A" you can input a list of covariates (see Details).
absorb	Character indicating which mode should absorb the inverse of the rescalings applied to mode (cannot be equal to mode).
method	Correlation method to use if newsign is a list input (see Details).
...	Ignored.

Details

If x is of class "parafac2" and mode="A", the input newsign can be a list where each element contains a covariate vector for resigning Mode A. You need length(newsign[[k]]) = nrow(x$A[[k]]) for all k when newsign is a list. In this case, the resigning is implemented according to the sign of cor(newsign[[k]], x$A[[k]][,1], method). See Helwig (2013) for details.

See indscal, parafac, parafac2, sca, and tucker for more details.

Value

Same as input.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2013). The special sign indeterminacy of the direct-fitting Parafac2 model: Some implications, cautions, and recommendations, for Simultaneous Component Analysis. *Psychometrika, 78*, 725-739.

Examples

```
### see examples for indscal, parafac, parafac2, sca, and tucker
```

---

sca *Simultaneous Component Analysis*

Description

Given a list of matrices `X[[k]] = matrix(xk,I[k],J)` for `k = seq(1,K)`, the SCA model is

$$X[[k]] = tcrossprod(D[[k]],B) + E[[k]]$$

where `D[[k]] = matrix(dk,I[k],R)` are the Mode A (first mode) weights for the k-th level of Mode C (third mode), `B = matrix(b,J,R)` are the Mode B (second mode) weights, and `E[[k]] = matrix(ek,I[k],J)` is the residual matrix corresponding to k-th level of Mode C.

There are four different versions of the SCA model: SCA with invariant pattern (SCA-P), SCA with Parafac2 constraints (SCA-PF2), SCA with INDSCAL constraints (SCA-IND), and SCA with equal average crossproducts (SCA-ECP). These four models differ with respect to the assumed crossproduct structure of the `D[[k]]` weights:

SCA-P:	`crossprod(D[[k]])/I[k] = Phi[[k]]`
SCA-PF2:	`crossprod(D[[k]])/I[k] = diag(C[k,])%*%Phi%*%diag(C[k,])`
SCA-IND:	`crossprod(D[[k]])/I[k] = diag(C[k,]*C[k,])`
SCA-ECP:	`crossprod(D[[k]])/I[k] = Phi`

where `Phi[[k]]` is specific to the k-th level of Mode C, `Phi` is common to all `K` levels of Mode C, and `C = matrix(c,K,R)` are the Mode C (third mode) weights. This function estimates the weight matrices `D[[k]]` and `B` (and `C` if applicable) using alternating least squares.

Usage

```
sca(X,nfac,nstart=10,maxit=500,
    type=c("sca-p","sca-pf2","sca-ind","sca-ecp"),
    rotation=c("none","varimax","promax"),
    ctol=1e-4,parallel=FALSE,cl=NULL)
```

Arguments

`X`	List of length `K` where the k-th element contains the `I[k]`-by-`J` data matrix `X[[k]]`. If `I[k]=I[1]` for all k, can input 3-way data array with `dim=c(I,J,K)`.
`nfac`	Number of factors.
`nstart`	Number of random starts.

maxit	Maximum number of iterations.
type	Type of SCA model to fit.
rotation	Rotation to use for type="sca-p" or type="sca-ecp".
ctol	Convergence tolerance.
parallel	Logical indicating if parLapply should be used. See Examples.
cl	Cluster created by makeCluster. Only used when parallel=TRUE.

Value

D	List of length K where k-th element contains D[[k]].
B	Mode B weight matrix.
C	Mode C weight matrix.
Phi	Mode A common crossproduct matrix (if type!="sca-p").
SSE	Sum of Squared Errors.
Rsq	R-squared value.
GCV	Generalized Cross-Validation.
edf	Effective degrees of freedom.
iter	Number of iterations.
cflag	Convergence flag.
type	Same as input type.
rotation	Same as input rotation.

Warnings

The ALS algorithm can perform poorly if the number of factors nfac is set too large.

Computational Details

The least squares SCA-P solution can be obtained from the singular value decomposition of the stacked matrix rbind(X[[1]],...,X[[K]]).

The least squares SCA-PF2 solution can be obtained using the uncontrained Parafac2 ALS algorithm (see parafac2).

The least squares SCA-IND solution can be obtained using the Parafac2 ALS algorithm with orthogonality constraints on Mode A.

The least squares SCA-ECP solution can be obtained using the Parafac2 ALS algorithm with orthogonality constraints on Mode A and the Mode C weights fixed at C[k,] = rep(I[k]^0.5,R).

Note

Default use is 10 random strarts (nstart=10) with 500 maximum iterations of the ALS algorithm for each start (maxit=500) using a convergence tolerance of 1e-4 (ctol=1e-4). The algorithm is determined to have converged once the change in R^2 is less than or equal to ctol.

Output cflag gives convergence information: cflag=0 if ALS algorithm converged normally, cflag=1 if maximum iteration limit was reached before convergence, and cflag=2 if ALS algorithm terminated abnormally due to problem with non-negativity constraints.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Helwig, N. E. (2013). The special sign indeterminacy of the direct-fitting Parafac2 model: Some implications, cautions, and recommendations, for Simultaneous Component Analysis. *Psychometrika, 78*, 725-739.

Timmerman, M. E., & Kiers, H. A. L. (2003). Four simultaneous component models for the analysis of multivariate time series from more than one subject to model intraindividual and interindividual differences. *Psychometrika, 68*, 105-121.

Examples

```
##########   sca-p   ##########

# create random data list with SCA-P structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Dmat <- matrix(rnorm(sum(nk)*nf),sum(nk),nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Dmats <- vector("list",mydim[3])
Xmat <- Emat <- vector("list",mydim[3])
dfc <- 0
for(k in 1:mydim[3]){
  dinds <- 1:nk[k] + dfc
  Dmats[[k]] <- Dmat[dinds,]
  dfc <- dfc + nk[k]
  Xmat[[k]] <- tcrossprod(Dmats[[k]],Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
rm(Dmat)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit SCA-P model (no rotation)
scamod <- sca(X,nfac=nf,nstart=1)
scamod

# check solution
crossprod(scamod$D[[1]] %*% diag(scamod$C[1,]^-1) ) / nk[1]
crossprod(scamod$D[[5]] %*% diag(scamod$C[5,]^-1) ) / nk[5]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# reorder and resign factors
scamod$B[1:4,]
scamod <- reorder(scamod, 2:1)
```

```
scamod$B[1:4,]
scamod <- resign(scamod, mode="B", newsign=c(1,-1))
scamod$B[1:4,]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# rescale factors
colSums(scamod$B^2)
colSums(scamod$C^2)
scamod <- rescale(scamod, mode="C")
colSums(scamod$B^2)
colSums(scamod$C^2)
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])


##########   sca-pf2   ##########

# create random data list with SCA-PF2 (Parafac2) structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- 10*matrix(rnorm(nf^2),nf,nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- Emat <- Fmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Fmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Fmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit SCA-PF2 model
scamod <- sca(X,nfac=nf,nstart=1,type="sca-pf2")
scamod

# check solution
scamod$Phi
crossprod(scamod$D[[1]] %*% diag(scamod$C[1,]^-1) ) / nk[1]
crossprod(scamod$D[[5]] %*% diag(scamod$C[5,]^-1) ) / nk[5]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# reorder and resign factors
scamod$B[1:4,]
scamod <- reorder(scamod, 2:1)
scamod$B[1:4,]
```

```
scamod <- resign(scamod, mode="B", newsign=c(1,-1))
scamod$B[1:4,]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# rescale factors
colSums(scamod$B^2)
colSums(scamod$C^2)
scamod <- rescale(scamod, mode="C")
colSums(scamod$B^2)
colSums(scamod$C^2)
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])


##########   sca-ind   ##########

# create random data list with SCA-IND structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- diag(nf)  # SCA-IND is Parafac2 with Gmat=identity
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- 10*matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- Emat <- Fmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Fmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Fmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit SCA-IND model
scamod <- sca(X,nfac=nf,nstart=1,type="sca-ind")
scamod

# check solution
scamod$Phi
crossprod(scamod$D[[1]] %*% diag(scamod$C[1,]^-1) ) / nk[1]
crossprod(scamod$D[[5]] %*% diag(scamod$C[5,]^-1) ) / nk[5]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# reorder and resign factors
scamod$B[1:4,]
scamod <- reorder(scamod, 2:1)
scamod$B[1:4,]
scamod <- resign(scamod, mode="B", newsign=c(1,-1))
```

```
scamod$B[1:4,]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# rescale factors
colSums(scamod$B^2)
colSums(scamod$C^2)
scamod <- rescale(scamod, mode="C")
colSums(scamod$B^2)
colSums(scamod$C^2)
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])


##########   sca-ecp   ##########

# create random data list with SCA-ECP structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- diag(nf)
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- matrix(sqrt(nk),mydim[3],nf)
Xmat <- Emat <- Fmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Fmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Fmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit SCA-ECP model
scamod <- sca(X,nfac=nf,nstart=1,type="sca-ecp")
scamod

# check solution
scamod$Phi
crossprod(scamod$D[[1]] %*% diag(scamod$C[1,]^-1) ) / nk[1]
crossprod(scamod$D[[5]] %*% diag(scamod$C[5,]^-1) ) / nk[5]
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# reorder and resign factors
scamod$B[1:4,]
scamod <- reorder(scamod, 2:1)
scamod$B[1:4,]
scamod <- resign(scamod, mode="B", newsign=c(-1,1))
scamod$B[1:4,]
```

```
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])

# rescale factors
colSums(scamod$B^2)
colSums(scamod$C^2)
scamod <- rescale(scamod, mode="B")
colSums(scamod$B^2)
colSums(scamod$C^2)
Xhat <- fitted(scamod)
sse <- sumsq(mapply("-",Xmat,Xhat))
sse/(sum(nk)*mydim[2])


## Not run:

##########   parallel computation   ##########

# create random data list with SCA-IND structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- sample(c(50,100,200),mydim[3],replace=TRUE)
Gmat <- diag(nf)  # SCA-IND is Parafac2 with Gmat=identity
Bmat <- matrix(runif(mydim[2]*nf),mydim[2],nf)
Cmat <- 10*matrix(runif(mydim[3]*nf),mydim[3],nf)
Xmat <- Emat <- Fmat <- vector("list",mydim[3])
for(k in 1:mydim[3]){
  Fmat[[k]] <- svd(matrix(rnorm(nk[k]*nf),nk[k],nf),nv=0)$u
  Xmat[[k]] <- tcrossprod(Fmat[[k]]%*%Gmat%*%diag(Cmat[k,]),Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]),nk[k],mydim[2])
}
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- mapply("+",Xmat,Emat)

# fit SCA-PF2 model (10 random starts -- sequential computation)
set.seed(1)
system.time({scamod <- sca(X,nfac=nf,type="sca-pf2")})
scamod

# fit SCA-PF2 model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({scamod <- sca(X,nfac=nf,type="sca-pf2",parallel=TRUE,cl=cl)})
scamod
stopCluster(cl)

# fit SCA-IND model (10 random starts -- sequential computation)
set.seed(1)
system.time({scamod <- sca(X,nfac=nf,type="sca-ind")})
scamod
```

```
# fit SCA-IND model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({scamod <- sca(X,nfac=nf,type="sca-ind",parallel=TRUE,cl=cl)})
scamod
stopCluster(cl)

# fit SCA-ECP model (10 random starts -- sequential computation)
set.seed(1)
system.time({scamod <- sca(X,nfac=nf,type="sca-ecp")})
scamod

# fit SCA-ECP model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({scamod <- sca(X,nfac=nf,type="sca-ecp",parallel=TRUE,cl=cl)})
scamod
stopCluster(cl)

## End(Not run)
```

`smpower` *Symmetric Matrix Power*

Description

Raise symmetric matrix to specified power. Default calculates symmetric square root.

Usage

```
smpower(X, power = 0.5, tol = NULL)
```

Arguments

`X`	Symmetric real-valued matrix.
`power`	Power to apply to eigenvalues of `X`.
`tol`	Stability tolerance for eigenvalues.

Value

Returns `X` raised to specified power.

Note

Default tolerance is `tol = max(dim(X)) * .Machine$double.eps`.

Note

Basically returns `tcrossprod(Y$vec%*%diag(Y$val^power),Y$vec)` where `Y = eigen(X,symmetric=TRUE)`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE   ##########

X <- crossprod(matrix(rnorm(2000),100,20))
Xsqrt <- smpower(X)         # square root
Xinv <- smpower(X,-1)       # inverse
Xisqrt <- smpower(X,-0.5)   # inverse square root
```

---

sumsq *Sum-of-Squares of Given Object*

Description

Calculates the sum-of-squares of `X`.

Usage

```
sumsq(X)
```

Arguments

`X`	Numeric scalar, vector, list, matrix, or array.

Value

Sum-of-squares of `X`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
##########   EXAMPLE 1   ##########
X <- 10
sumsq(X)


##########   EXAMPLE 2   ##########
X <- 1:10
```

```
sumsq(X)


##########   EXAMPLE 3   ##########
X <- matrix(1:10,5,2)
sumsq(X)


##########   EXAMPLE 4   ##########
X <- array(matrix(1:10,5,2),dim=c(5,2,2))
sumsq(X)


##########   EXAMPLE 5   ##########
X <- vector("list",5)
for(k in 1:5){ X[[k]] <- matrix(1:10,5,2) }
sumsq(X)
```

`tucker` *Tucker Factor Analysis*

Description

Given a 3-way array `X = array(x,dim=c(I,J,K))`, the 3-way Tucker model can be written as

```
X[i,j,k] = sum sum sum A[i,p]*B[j,q]*C[k,r]*G[p,q,r] + E[i,j,k]
```

where `A = matrix(a,I,P)` are the Mode A (first mode) weights, `B = matrix(b,J,Q)` are the Mode B (second mode) weights, `C = matrix(c,K,R)` are the Mode C (third mode) weights, `G = array(g,dim=c(P,Q,R))` is the 3-way core array, and `E = array(e,dim=c(I,J,K))` is the 3-way residual array. The summations are for `p = seq(1,P)`, `q = seq(1,Q)`, and `r = seq(1,R)`.

Given a 4-way array `X = array(x,dim=c(I,J,K,L))`, the 4-way Tucker model can be written as

```
X[i,j,k,l] = sum sum sum sum A[i,p]*B[j,q]*C[k,r]*D[l,s]*G[p,q,r,s] + E[i,j,k,l]
```

where `D = matrix(d,L,S)` are the Mode D (fourth mode) weights, `G = array(g,dim=c(P,Q,R,S))` is the 4-way residual array, `E = array(e,dim=c(I,J,K,L))` is the 4-way residual array, and the other terms can be interprered as previously described.

Weight matrices are estimated using an alternating least squares algorithm.

Usage

```
tucker(X, nfac, nstart = 10, Afixed = NULL,
       Bfixed = NULL, Cfixed = NULL, Dfixed = NULL,
       Bstart = NULL, Cstart = NULL, Dstart = NULL,
       maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
```

```
output = c("best", "all"), verbose = FALSE)
```

Arguments

X	Three-way data array with dim=c(I,J,K) or four-way data array with dim=c(I,J,K,L). Missing data are allowed (see Note).
nfac	Number of factors in each mode.
nstart	Number of random starts.
Afixed	Fixed Mode A weights. Only used to fit model with fixed weights in Mode A.
Bfixed	Fixed Mode B weights. Only used to fit model with fixed weights in Mode B.
Cfixed	Fixed Mode C weights. Only used to fit model with fixed weights in Mode C.
Dfixed	Fixed Mode D weights. Only used to fit model with fixed weights in Mode D.
Bstart	Starting Mode B weights for ALS algorithm. Default uses random weights.
Cstart	Starting Mode C weights for ALS algorithm. Default uses random weights.
Dstart	Starting Mode D weights for ALS algorithm. Default uses random weights.
maxit	Maximum number of iterations.
ctol	Convergence tolerance.
parallel	Logical indicating if parLapply should be used. See Examples.
cl	Cluster created by makeCluster. Only used when parallel=TRUE.
output	Output the best solution (default) or output all nstart solutions.
verbose	Logical indicating if extra information on progress should be reported. Ignored if parallel=TRUE.

Value

If output="best", returns an object of class "tucker" with the following elements:

A	Mode A weight matrix.
B	Mode B weight matrix.
C	Mode C weight matrix.
D	Mode D weight matrix.
G	Core array.
SSE	Sum of Squared Errors.
Rsq	R-squared value.
GCV	Generalized Cross-Validation.
edf	Effective degrees of freedom.
iter	Number of iterations.
cflag	Convergence flag.

Otherwise returns a list of length nstart where each element is an object of class "tucker".

Warnings

The ALS algorithm can perform poorly if the number of factors `nfac` is set too large.

Input matrices in `Afixed`, `Bfixed`, `Cfixed`, `Dfixed`, `Bstart`, `Cstart`, and `Dstart` must be columnwise orthonormal.

Note

Default use is 10 random strarts (`nstart=10`) with 500 maximum iterations of the ALS algorithm for each start (`maxit=500`) using a convergence tolerance of 1e-4 (`ctol=1e-4`). The algorithm is determined to have converged once the change in R^2 is less than or equal to `ctol`.

Output `cflag` gives convergence information: `cflag=0` if ALS algorithm converged normally, and `cflag=1` if maximum iteration limit was reached before convergence.

Missing data should be specified as `NA` values in the input `X`. The missing data are randomly initialized and then iteratively imputed as a part of the ALS algorithm.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Kroonenberg, P. M., & de Leeuw, J. (1980). Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika, 45*, 69-97.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31*, 279-311.

Examples

```
##########   3-way example   ##########

# create random data array with Tucker structure
set.seed(3)
mydim <- c(50,20,5)
nf <- c(3,2,3)
Amat <- svd(matrix(rnorm(mydim[1]*nf[1]),mydim[1],nf[1]),nu=nf[1])$u
Bmat <- svd(matrix(rnorm(mydim[2]*nf[2]),mydim[2],nf[2]),nu=nf[2])$u
Cmat <- svd(matrix(rnorm(mydim[3]*nf[3]),mydim[3],nf[3]),nu=nf[3])$u
Gmat <- array(rnorm(prod(nf)),dim=nf)
Xmat <- array(tcrossprod(Amat%*%matrix(Gmat,nf[1],nf[2]*nf[3]),kronecker(Cmat,Bmat)),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker model
tuck <- tucker(X,nfac=nf,nstart=1)
tuck

# check solution
Xhat <- fitted(tuck)
```

```
sum((Xmat-Xhat)^2)/prod(mydim)

# reorder mode="A"
tuck$A[1:4,]
tuck$G
tuck <- reorder(tuck, neworder=c(3,1,2), mode="A")
tuck$A[1:4,]
tuck$G
Xhat <- fitted(tuck)
sum((Xmat-Xhat)^2)/prod(mydim)

# reorder mode="B"
tuck$B[1:4,]
tuck$G
tuck <- reorder(tuck, neworder=2:1, mode="B")
tuck$B[1:4,]
tuck$G
Xhat <- fitted(tuck)
sum((Xmat-Xhat)^2)/prod(mydim)

# resign mode="C"
tuck$C[1:4,]
tuck <- resign(tuck, mode="C")
tuck$C[1:4,]
Xhat <- fitted(tuck)
sum((Xmat-Xhat)^2)/prod(mydim)


##########   4-way example   ##########

# create random data array with Tucker structure
set.seed(4)
mydim <- c(30,10,8,10)
nf <- c(2,3,4,3)
Amat <- svd(matrix(rnorm(mydim[1]*nf[1]),mydim[1],nf[1]),nu=nf[1])$u
Bmat <- svd(matrix(rnorm(mydim[2]*nf[2]),mydim[2],nf[2]),nu=nf[2])$u
Cmat <- svd(matrix(rnorm(mydim[3]*nf[3]),mydim[3],nf[3]),nu=nf[3])$u
Dmat <- svd(matrix(rnorm(mydim[4]*nf[4]),mydim[4],nf[4]),nu=nf[4])$u
Gmat <- array(rnorm(prod(nf)),dim=nf)
Xmat <- array(tcrossprod(Amat%*%matrix(Gmat,nf[1],prod(nf[2:4])),
                      kronecker(Dmat,kronecker(Cmat,Bmat))),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker model
tuck <- tucker(X,nfac=nf,nstart=1)
tuck

# check solution
Xhat <- fitted(tuck)
sum((Xmat-Xhat)^2)/prod(mydim)
```

```
## Not run:

##########   parallel computation   ##########

# create random data array with Tucker structure
set.seed(3)
mydim <- c(50,20,5)
nf <- c(3,2,3)
Amat <- svd(matrix(rnorm(mydim[1]*nf[1]),mydim[1],nf[1]),nu=nf[1])$u
Bmat <- svd(matrix(rnorm(mydim[2]*nf[2]),mydim[2],nf[2]),nu=nf[2])$u
Cmat <- svd(matrix(rnorm(mydim[3]*nf[3]),mydim[3],nf[3]),nu=nf[3])$u
Gmat <- array(rnorm(prod(nf)),dim=nf)
Xmat <- array(tcrossprod(Amat%*%matrix(Gmat,nf[1],nf[2]*nf[3]),kronecker(Cmat,Bmat)),dim=mydim)
Emat <- array(rnorm(prod(mydim)),dim=mydim)
Emat <- nscale(Emat,0,sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker model (10 random starts -- sequential computation)
set.seed(1)
system.time({tuck <- tucker(X,nfac=nf)})
tuck$Rsq

# fit Tucker model (10 random starts -- parallel computation)
set.seed(1)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
system.time({tuck <- tucker(X,nfac=nf,parallel=TRUE,cl=cl)})
tuck$Rsq
stopCluster(cl)

## End(Not run)
```

`USalcohol` *United States Alcohol Consumption Data (1970-2013)*

Description

This dataset contains yearly (1970-2013) consumption data from the 50 United States and the District of Columbia for three types of alcoholic beverages: spirits, wine, and beer. The data were obtained from the National Institute on Alcohol Abuse and Alcoholism (NIAAA) Surveillance Report #102 (see below link).

Usage

```
data("USalcohol")
```

Format

A data frame with 6732 observations on the following 8 variables.

`year` integer Year (1970-2013)

`state` factor State Name (51 levels)

`region` factor Region Name (4 levels)

`type` factor Beverage Type (3 levels)

`beverage` numeric Beverage Consumed (thousands of gallons)

`ethanol` numeric Absolute Alcohol Consumed (thousands of gallons)

`pop14` numeric Population Age 14 and Older (thousands of people)

`pop21` numeric Population Age 21 and Older (thousands of people)

Details

In the data source, the population age 21 and older for Mississippi in year 1989 is reported to be 3547.839 thousand, which is incorrect. In this dataset, the miscoded population value has been replaced with the average of the corresponding 1988 population (1709 thousand) and the 1990 population (1701.527 thousand).

Source

https://pubs.niaaa.nih.gov/publications/surveillance102/pcyr19702013.txt

References

Haughwout, S. P., LaVallee, R. A., & Castle, I-J. P. (2015). Surveillance Report #102: Apparent Per Capita Alcohol Consumption: National, State, and Regional Trends, 1977-2013. Bethesda, MD: NIAAA, Alcohol Epidemiologic Data System.

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803.

Nephew, T. M., Yi, H., Williams, G. D., Stinson, F. S., & Dufour, M.C., (2004). U.S. Alcohol Epidemiologic Data Reference Manual, Vol. 1, 4th ed. U.S. Apparent Consumption of Alcoholic Beverages Based on State Sales, Taxation, or Receipt Data. Bethesda, MD: NIAAA, Alcohol Epidemiologic Data System. NIH Publication No. 04-5563.

Examples

```
# load data and print first six rows
data(USalcohol)
head(USalcohol)

# form tensor (time x variables x state)
Xbev <- with(USalcohol, tapply(beverage/pop21, list(year, type, state), c))
Xeth <- with(USalcohol, tapply(ethanol/pop21, list(year, type, state), c))
X <- array(0, dim=c(44, 6, 51))
X[, c(1,3,5) ,] <- Xbev
X[, c(2,4,6) ,] <- Xeth
```

```
dnames <- dimnames(Xbev)
dnames[[2]] <- c(paste0(dnames[[2]],".bev"), paste0(dnames[[2]],".eth"))[c(1,4,2,5,3,6)]
dimnames(X) <- dnames

# center each variable across time (within state)
Xc <- ncenter(X, mode=1)

# scale each variable to have mean square of 1 (across time and states)
Xs <- nscale(Xc, mode=2, ssnew=44*51)

# fit parafac model with 3 factors
set.seed(1)
pfac <- parafac(Xs, nfac=3, nstart=1)

# fit parafac model with functional constraints
set.seed(1)
pfacF <- parafac(Xs, nfac=3, nstart=1, const=c(6,0,0))

# fit parafac model with functional and structural constraints
Bstruc <- matrix(c(rep(c(TRUE,FALSE), c(2,4)),
                   rep(c(FALSE,TRUE,FALSE), c(2,2,2)),
                   rep(c(FALSE,TRUE), c(4,2))), nrow=6, ncol=3)
set.seed(1)
pfacFS <- parafac(Xs, nfac=3, nstart=1, const=c(6,0,0), Bstruc=Bstruc)
```

Index