Package 'multiway'

May 9, 2026

Type Package

Title Component Models for Multi-Way Data

Version 1.0-7

Date 2025-04-15

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

Depends CMLS, parallel

Description Fits multi-way component models via alternating least squares algorithms with optional constraints. Fit models include N-way Canonical Polyadic Decomposition, Individual Differences Scaling, Multiway Covariates Regression, Parallel Factor Analysis (1 and 2), Simultaneous Component Analysis, and Tucker Factor Analysis.

License GPL (>= 2)

NeedsCompilation no

Author Nathaniel E. Helwig [aut, cre]

Repository CRAN

Date/Publication 2025-04-15 16:40:02 UTC

Contents

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

Description

Fits multi-way component models via alternating least squares algorithms with optional constraints. Fit models include N-way Canonical Polyadic Decomposition, Individual Differences Scaling, Multiway Covariates Regression, Parallel Factor Analysis (1 and 2), Simultaneous Component Analysis, and Tucker Factor Analysis.

Details

The DESCRIPTION file:

Package:	multiway
Type:	Package
Title:	Component Models for Multi-Way Data
Version:	1.0-7
Date:	2025-04-15
Authors@R:	person(given = c("Nathaniel", "E."), family = "Helwig", role = c("aut", "cre"), email = "helwig@umn.edu")
Maintainer:	Nathaniel E. Helwig <helwig@umn.edu>
Depends:	CMLS, parallel
Description:	Fits multi-way component models via alternating least squares algorithms with optional constraints. Fit models
License:	GPL (>=2)
Author:	Nathaniel E. Helwig [aut, cre]

Index of help topics:

```
USalcohol               United States Alcohol Consumption Data
                        (1970-2013)
congru                  Tucker's Congruence Coefficient
const.control           Auxiliary for Controlling Multi-Way Constraints
corcondia               Core Consistency Diagnostic
```

cpd	N-way Canonical Polyadic Decomposition
fitted.cpd	Extract Multi-Way Fitted Values
fnnls	Fast Non-Negative Least Squares
indscal	Individual Differences Scaling
krprod	Khatri-Rao Product
mcr	Multiway Covariates Regression
meansq	Mean Square of Given Object
mpinv	Moore-Penrose Pseudoinverse
multiway-package	Component Models for Multi-Way Data
ncenter	Center n-th Dimension of Array
nscale	Scale n-th Dimension of Array
parafac	Parallel Factor Analysis-1
parafac2	Parallel Factor Analysis-2
print.cpd	Print Multi-Way Model Results
reorder.cpd	Reorder Multi-Way Factors
rescale	Rescales Multi-Way Factors
resign	Resigns Multi-Way Factors
sca	Simultaneous Component Analysis
smpower	Symmetric Matrix Power
sumsq	Sum-of-Squares of Given Object
tucker	Tucker Factor Analysis

`cpd` computes the N-way Canonical Polyadic Decomposition of a tensor. `indscal` fits the Individual Differences Scaling model. `mcr` fits the Multiway Covariates Regression 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 [aut, cre]

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. doi:10.1002/cem.801

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. doi:10.1007/BF02310791

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. doi:10.1016/01679473(94)901325

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. doi:10.1007/S1133601393317

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803. doi:10.1002/bimj.201600045

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics, 6*, 164-189. doi:10.1002/sapm192761164

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. doi:10.1002/(SICI)1099-128X(199905/08)13:3/4<275::AIDCEM543>3.0.CO;2B

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. doi:10.1007/BF02293599

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. doi:10.1017/S0305004100030401

Ramsay, J. O. (1988). Monotone regression splines in action. *Statistical Science, 3*, 425-441. doi:10.1214/ss/1177012761

Smilde, A. K., & Kiers, H. A. L. (1999). Multiway covariates regression models. *Journal of Chemometrics, 13*, 31-48. doi:10.1002/(SICI)1099128X(199901/02)13:1<31::AIDCEM528>3.0.CO;2P

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. doi:10.1007/BF02296656

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31*, 279-311. doi:10.1007/BF02289464

See Also

CMLS ~~

Examples

```
# See examples for...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
```

```
#   tucker (Tucker Factor Analysis)
```

`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 `mcr`, `parafac`, and `parafac2` functions. Applicable when using smoothness constraints.

Usage

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

Arguments

`const`	Character vector of length 3 or 4 giving the constraints for each mode. See `const` for the 24 available options.
`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.

`intercept` Logical vector of length 3 or 4 indicating whether the spline basis should contain an intercept. Can also input a single logical giving the common intercept indicator to use for each mode. Defaults to `TRUE` for each applicable mode.

Details

The `mcr`, `parafac`, and `parafac2` functions pass the input `const` to this function to determine the fitting options when using smoothness constraints.

The `const` function (from **CMLS** package) describes the available constraint options.

Value

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

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.out = 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), nrow = mydim[2], ncol = nf), nv = 0)$u
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Cstruc <- Cmat > 0.5
Cmat <- Cmat * Cstruc
Dmat <- matrix(runif(mydim[4]*nf), nrow = mydim[4], ncol = nf)
Xmat <- tcrossprod(Amat, krprod(Dmat, krprod(Cmat, Bmat)))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + Emat

# fit Parafac model (unimodal and smooth A, orthogonal B,
#                    non-negative and structured C, non-negative D)
set.seed(123)
pfac <- parafac(X, nfac = nf, nstart = 1, Cstruc = Cstruc,
                const = c("unismo", "orthog", "nonneg", "nonneg"))
pfac

# same as before, but add some options to the unimodality contraints...
# more knots (df=10), quadratic splines (degree=2), and enforce non-negativity
cvec <- c("unsmno", "orthog", "nonneg", "nonneg")
ctrl <- const.control(cvec, df = 10, degree = 2)
set.seed(123)
```

```
pfac <- parafac(X, nfac = nf, nstart = 1, Cstruc = Cstruc,
                const = cvec, control = ctrl)
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. doi:10.1002/cem.801

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, ssnew = 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)
```

---

cpd *N-way Canonical Polyadic Decomposition*

Description

Fits Frank L. Hitchcock's Canonical Polyadic Decomposition (CPD) to N-way data arrays. Parameters are estimated via alternating least squares.

Usage

```
cpd(X, nfac, nstart = 10, maxit = 500,
    ctol = 1e-4, parallel = FALSE, cl = NULL,
    output = "best", verbose = TRUE)
```

Arguments

X	N-way data array. Missing data are allowed (see Note).
nfac	Number of factors.
nstart	Number of random starts.
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	If TRUE, fitting progress is printed via txtProgressBar. Ignored if parallel=TRUE.

Details

This is an N-way extension of the parafac function without constraints. The form of the CPD for 3-way and 4-way data is given in the documentation for the parafac function. For N > 4, the CPD has the form

```
X[i.1, ..., i.N] = sum A.1[i.1,r] * ... * A.N[i.N,r] + E[i.1, ..., i.N]
```

where A.n are the n-th mode's weights for n = 1, ..., N, and E is the N-way residual array. The summation is for r = seq(1,R).

Value

A	List of length N containing the weights for each mode.
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. See Note.

Warnings

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

Note

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 algorithm.

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

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. doi:10.1016/01679473(94)901325

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics, 6*, 164-189. doi:10.1002/sapm192761164

See Also

The `parafac` function provides a more flexible implemention for 3-way and 4-way arrays.

The `fitted.cpd` function creates the model-implied fitted values from a fit `"cpd"` object.

The `resign.cpd` function can be used to resign factors from a fit `"cpd"` object.

The `rescale.cpd` function can be used to rescale factors from a fit `"cpd"` object.

The `reorder.cpd` function can be used to reorder factors from a fit `"cpd"` object.

Examples

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

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

# fit CPD model
set.seed(0)
cano <- cpd(X, nfac = nf, nstart = 1)
cano

# fit Parafac model
set.seed(0)
pfac <- parafac(X, nfac = nf, nstart = 1)
pfac


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

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

```
nf <- 4
aseq <- seq(-3, 3, length.out = 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), nrow = mydim[2], ncol = nf), nv = 0)$u
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Cstruc <- Cmat > 0.5
Cmat <- Cmat * Cstruc
Dmat <- matrix(runif(mydim[4]*nf), nrow = mydim[4], ncol = nf)
Xmat <- tcrossprod(Amat, krprod(Dmat, krprod(Cmat, Bmat)))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + Emat

# fit CPD model
set.seed(0)
cano <- cpd(X, nfac = nf, nstart = 1)
cano

# fit Parafac model
set.seed(0)
pfac <- parafac(X, nfac = nf, nstart = 1)
pfac


##########   5-way example   ##########

# create random data array with CPD/Parafac structure
set.seed(5)
mydim <- c(5, 6, 7, 8, 9)
nmode <- length(mydim)
nf <- 3
Amat <- vector("list", nmode)
for(n in 1:nmode) {
  Amat[[n]] <- matrix(rnorm(mydim[n] * nf), mydim[n], nf)
}
Zmat <- krprod(Amat[[3]], Amat[[2]])
for(n in 4:5) Zmat <- krprod(Amat[[n]], Zmat)
Xmat <- tcrossprod(Amat[[1]], Zmat)
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + Emat

# fit CPD model
set.seed(0)
cano <- cpd(X, nfac = nf, nstart = 1)
cano
```

`fitted` *Extract Multi-Way Fitted Values*

Description

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

Usage

```
## S3 method for class 'cpd'
fitted(object, ...)
## S3 method for class 'indscal'
fitted(object, ...)
## S3 method for class 'mcr'
fitted(object, type = c("X", "Y"), ...)
## 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 "cpd" (output from `cpd`), "indscal" (output from `indscal`), class "mcr" (output from `mcr`), 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).
`type`	For "mcr", setting `type = "X"` returns the fitted predictor array (default), whereas setting `type = "Y"` retuns the fitted response array.
`...`	Ignored.

Details

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

Value

"cpd" objects: N-way array.

"indscal" objects: 3-way array.

"mcr" objects: 3-way (X) or 2-way (Y) 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...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
#   tucker (Tucker Factor Analysis)
```

---

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

Fits Carroll and Chang's Individual Differences Scaling (INDSCAL) model to 3-way dissimilarity or similarity data. Parameters are estimated via alternating least squares with optional constraints.

Usage

```
indscal(X, nfac, nstart = 10, const = NULL, control = NULL,
        type = c("dissimilarity", "similarity"),
        Bfixed = NULL, Bstart = NULL, Bstruc = NULL, Bmodes = NULL,
        Cfixed = NULL, Cstart = NULL, Cstruc = NULL, Cmodes = NULL,
        maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
        output = c("best", "all"), verbose = TRUE, backfit = FALSE)
```

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	Character vector of length 2 giving the constraints for modes B and C (defaults to unconstrained for B and non-negative for C). See const for the 24 available options. Constraints for Mode C weights are limited to one of the 8 possible non-negative options.
control	List of parameters controlling options for smoothness constraints. This is passed to const.control, which describes the available options.
type	Character indicating if X contains dissimilarity data (default) or similarity data.
Bfixed	Used to fit model with fixed Mode B weights.
Bstart	Starting Mode B weights. Default uses random weights.
Bstruc	Structure constraints for Mode B weights. See Note.
Bmodes	Mode ranges for Mode B weights (for unimodality constraints). See Note.
Cfixed	Used to fit model with fixed Mode C weights.
Cstart	Starting Mode C weights. Default uses random weights.
Cstruc	Structure constraints for Mode C weights. See Note.
Cmodes	Mode ranges for Mode C weights (for unimodality constraints). See Note.
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	If TRUE, fitting progress is printed via txtProgressBar. Ignored if parallel=TRUE.
backfit	Should backfitting algorithm be used for cmls?

Details

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.

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. See Note.
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 "indscal".

Warnings

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

Note

Structure constraints should be specified with a matrix of logicals (TRUE/FALSE), such that FALSE elements indicate a weight should be constrained to be zero. Default uses unstructured weights, i.e., a matrix of all TRUE values.

When using unimodal constraints, the `*modes` inputs can be used to specify the mode search range for each factor. These inputs should be matrices with dimension `c(2,nfac)` where the first row gives the minimum mode value and the second row gives the maximum mode value (with respect to the indicies of the given corresponding matrix).

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

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. doi:10.1007/BF02310791

See Also

The `fitted.indscal` function creates the model-implied fitted values from a fit `"indscal"` object.

The `resign.indscal` function can be used to resign factors from a fit `"indscal"` object.

The `rescale.indscal` function can be used to rescale factors from a fit `"indscal"` object.

The `reorder.indscal` function can be used to reorder factors from a fit `"indscal"` object.

The `cmls` function (from **CMLS** package) is called as a part of the alternating least squares algorithm.

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
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("orthog", "nonneg"))
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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
clusterSetRNGStream(cl, 1)
system.time({imod <- indscal(X, nfac = nf, parallel = TRUE, cl = cl)})
imod
stopCluster(cl)

## End(Not run)
```

`krprod` *Khatri-Rao Product*

Description

Calculates the Khatri-Rao product (i.e., columnwise Kronecker product) between two matrices with the same number of columns.

Usage

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

Arguments

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

Details

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`.

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)
```

---

mcr *Multiway Covariates Regression*

Description

Fits Smilde and Kiers's Multiway Covariates Regression (MCR) model to connect a 3-way predictor array and a 2-way response array that share a common mode. Parameters are estimated via alternating least squares with optional constraints.

Usage

```
mcr(X, Y, nfac = 1, alpha = 0.5, nstart = 10,
    model = c("parafac", "parafac2", "tucker"),
    const = NULL, control = NULL, weights = NULL,
    Afixed = NULL, Bfixed = NULL, Cfixed = NULL, Dfixed = NULL,
    Astart = NULL, Bstart = NULL, Cstart = NULL, Dstart = NULL,
    Astruc = NULL, Bstruc = NULL, Cstruc = NULL, Dstruc = NULL,
    Amodes = NULL, Bmodes = NULL, Cmodes = NULL, Dmodes = NULL,
    maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
    output = c("best", "all"), verbose = TRUE, backfit = FALSE)
```

Arguments

X	Three-way predictor array with dim = c(I,J,K).
Y	Two-way response array with dim = c(K,L).
nfac	Number of factors.
alpha	Tuning parameter between 0 and 1.
nstart	Number of random starts.
model	Model for X. Defaults to "parafac".
const	Character vector of length 4 giving the constraints for A, B, C, and D (defaults to unconstrained). See const for the 24 available options. Ignored if model = "tucker".
control	List of parameters controlling options for smoothness constraints. This is passed to const.control, which describes the available options.
weights	Vector of length K giving non-negative weights for fitting via weighted least squares. Defaults to vector of ones.
Afixed	Used to fit model with fixed Mode A weights.
Bfixed	Used to fit model with fixed Mode B weights.
Cfixed	Used to fit model with fixed Mode C weights.
Dfixed	Used to fit model with fixed Mode D weights.
Astart	Starting Mode A weights. Default uses random weights.
Bstart	Starting Mode B weights. Default uses random weights.
Cstart	Starting Mode C weights. Default uses random weights.
Dstart	Starting Mode D weights. Default uses random weights.
Astruc	Structure constraints for Mode A weights. See Note.
Bstruc	Structure constraints for Mode B weights. See Note.
Cstruc	Structure constraints for Mode C weights. Ignored.
Dstruc	Structure constraints for Mode D weights. See Note.
Amodes	Mode ranges for Mode A weights (for unimodality constraints). See Note.
Bmodes	Mode ranges for Mode B weights (for unimodality constraints). See Note.
Cmodes	Mode ranges for Mode C weights (for unimodality constraints). Ignored.
Dmodes	Mode ranges for Mode D weights (for unimodality constraints). See Note.
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	If TRUE, fitting progress is printed via txtProgressBar. Ignored if parallel=TRUE.
backfit	Should backfitting algorithm be used for cmls?

Details

Given a predictor array `X = array(x, dim=c(I,J,K))` and a response matrix `Y = matrix(y, nrow=K, ncol=L)`, the multiway covariates regression (MCR) model assumes a tensor model for `X` and a bilinear model for `Y`, which are linked through a common `C` weight matrix. For example, using the Parafac model for `X`, the MCR model has the form

```
X[i,j,k] = sum A[i,r]*B[j,r]*C[k,r] + Ex[i,j,k]
```
and
```
Y[k,l] = sum C[k,r]*D[l,r] + Ey[k,l]
```

Parameter matrices are estimated by minimizing the loss function

```
LOSS = alpha * (SSE.X / SSX) + (1 - alpha) * (SSE.Y / SSY)
```

where

```
SSE.X = sum((X - Xhat)^2)
SSE.Y = sum((Y - Yhat)^2)
SSX = sum(X^2)
SSY = sum(Y^2)
```

When `weights` are input, `SSE.X`, `SSE.Y`, `SSX`, and `SSY` are replaced by the corresponding weighted versions.

Value

`A`	Predictor A weight matrix.
`B`	Predictor B weight matrix.
`C`	Common C weight matrix.
`D`	Response D weight matrix.
`W`	Coefficients. See Note.
`LOSS`	Value of `LOSS` function.
`SSE`	Sum of Squared Errors for `X` and `Y`.
`Rsq`	R-squared value for `X` and `Y`.
`iter`	Number of iterations.
`cflag`	Convergence flag. See Note.
`model`	See argument `model`.
`const`	See argument `const`.
`control`	See argument `control`.
`weights`	See argument `weights`.
`alpha`	See argument `alpha`.

fixed	Logical vector indicating whether 'fixed' weights were used for each matrix.
struc	Logical vector indicating whether 'struc' constraints were used for each matrix.
Phi	Mode A crossproduct matrix. Only if model = "parafac2".
G	Core array. Only if model = "tucker".

Note

When model = "parafac2", the arguments Afixed, Astart, and Astruc are treated as the arguments Gfixed, Gstart, and Gstruc from the parafac2 function.

Structure constraints should be specified with a matrix of logicals (TRUE/FALSE), such that FALSE elements indicate a weight should be constrained to be zero. Default uses unstructured weights, i.e., a matrix of all TRUE values. Structure constraints are ignored if model = "tucker".

When using unimodal constraints, the *modes inputs can be used to specify the mode search range for each factor. These inputs should be matrices with dimension c(2,nfac) where the first row gives the minimum mode value and the second row gives the maximum mode value (with respect to the indicies of the corresponding weight matrix).

C = Xc %*% W where Xc = matrix(aperm(X,c(3,1,2)),K)

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

Smilde, A. K., & Kiers, H. A. L. (1999). Multiway covariates regression models, *Journal of Chemometrics, 13*, 31-48. doi:10.1002/(SICI)1099128X(199901/02)13:1<31::AIDCEM528>3.0.CO;2P

See Also

The fitted.mcr function creates the model-implied fitted values from a fit "mcr" object.

The resign.mcr function can be used to resign factors from a fit "mcr" object.

The rescale.mcr function can be used to rescale factors from a fit "mcr" object.

The reorder.mcr function can be used to reorder factors from a fit "mcr" object.

The cmls function (from **CMLS** package) is called as a part of the alternating least squares algorithm.

See parafac, parafac2, and tucker for more information about the Parafac, Parafac2, and Tucker models.

Examples

```
##########   multiway covariates regression   ##########

# create random data array with Parafac structure
set.seed(3)
mydim <- c(10, 20, 100)
nf <- 2
Amat <- matrix(rnorm(mydim[1]*nf), mydim[1], nf)
Bmat <- matrix(rnorm(mydim[2]*nf), mydim[2], nf)
Cmat <- matrix(rnorm(mydim[3]*nf), mydim[3], nf)
Xmat <- tcrossprod(Amat, krprod(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
EX <- array(rnorm(prod(mydim)), dim = mydim)
EX <- nscale(EX, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + EX

# create response array
ydim <- c(mydim[3], 4)
Dmat <- matrix(rnorm(ydim[2]*nf), ydim[2], nf)
Ymat <- tcrossprod(Cmat, Dmat)
EY <- array(rnorm(prod(ydim)), dim = ydim)
EY <- nscale(EY, 0, ssnew = sumsq(Ymat))   # SNR = 1
Y <- Ymat + EY

# fit MCR model
mcr(X, Y, nfac = nf, nstart = 1)
mcr(X, Y, nfac = nf, nstart = 1, model = "parafac2")
mcr(X, Y, nfac = nf, nstart = 1, model = "tucker")



## Not run:

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

# create random data array with Parafac structure
set.seed(3)
mydim <- c(10, 20, 100)
nf <- 2
Amat <- matrix(rnorm(mydim[1]*nf), mydim[1], nf)
Bmat <- matrix(rnorm(mydim[2]*nf), mydim[2], nf)
Cmat <- matrix(rnorm(mydim[3]*nf), mydim[3], nf)
Xmat <- tcrossprod(Amat, krprod(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
EX <- array(rnorm(prod(mydim)), dim = mydim)
EX <- nscale(EX, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + EX

# create response array
ydim <- c(mydim[3], 4)
Dmat <- matrix(rnorm(ydim[2]*nf), ydim[2], nf)
Ymat <- tcrossprod(Cmat, Dmat)
```

```
EY <- array(rnorm(prod(ydim)), dim = ydim)
EY <- nscale(EY, 0, ssnew = sumsq(Ymat))   # SNR = 1
Y <- Ymat + EY

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

# fit MCR-Parafac model (10 random starts -- parallel computation)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl, library(multiway))
clusterSetRNGStream(cl, 1)
system.time({mod <- mcr(X, Y, nfac = nf, parallel = TRUE, cl = cl)})
mod
stopCluster(cl)


## End(Not run)
```

`meansq` *Mean Square of Given Object*

Description

Calculates the mean square of `X`.

Usage

```
meansq(X, na.rm = FALSE)
```

Arguments

`X`	Numeric scalar, vector, list, matrix, or array.
`na.rm`	logical. Should missing values (including `NaN`) be removed?

Value

Mean square of `X`.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

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


##########   EXAMPLE 2   ##########
X <- 1:10
meansq(X)


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


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


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

`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.

Details

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

Value

Returns pseudoinverse of `X`.

Note

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

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. doi:10.1017/S0305004100030401

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.

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.

Details

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,])
```

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).

Usage

```
nscale(X, mode = 1, ssnew = NULL, newscale = 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.
`newscale`	Desired root-mean-square for each level of scaled mode. Ignored if `ssnew` is supplied.

Details

Default (as of ver 1.0-5) uses `newscale` argument...

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

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

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

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

If argument ssnew is provided...

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]))
```

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), nrow = 100, ncol = 20)
Xr <- nscale(X, mode = 2)               # scale columns to newscale=1
sqrt(colMeans(Xr^2))
Xr <- nscale(X, mode = 2, newscale = 2)  # scale columns to newscale=2
sqrt(colMeans(Xr^2))


##########   EXAMPLE 2   ##########
Xold <- X <- matrix(rnorm(400), nrow = 20, ncol = 20)
iter <- 0
chk <- 1
# iterative scaling 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
sqrt(rowMeans(Xr^2))
sqrt(colMeans(Xr^2))


##########   EXAMPLE 3   ##########
X <- array(rnorm(20000), dim = c(100,20,10))
Xc <- nscale(X, mode = 2)   # scale within columns
sqrt(rowMeans(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
sqrt(rowMeans(aperm(Xc, perm = c(4,1,2,3))^2))


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

parafac *Parallel Factor Analysis-1*

Description

Fits Richard A. Harshman's Parallel Factors (Parafac) model to 3-way or 4-way data arrays. Parameters are estimated via alternating least squares with optional constraints.

Usage

```
parafac(X, nfac, nstart = 10, const = NULL, control = NULL,
        Afixed = NULL, Bfixed = NULL, Cfixed = NULL, Dfixed = NULL,
        Astart = NULL, Bstart = NULL, Cstart = NULL, Dstart = NULL,
        Astruc = NULL, Bstruc = NULL, Cstruc = NULL, Dstruc = NULL,
        Amodes = NULL, Bmodes = NULL, Cmodes = NULL, Dmodes = NULL,
        maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
        output = c("best", "all"), verbose = TRUE, backfit = 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	Character vector of length 3 or 4 giving the constraints for each mode (defaults to unconstrained). See const for the 24 available options.
control	List of parameters controlling options for smoothness constraints. This is passed to const.control, which describes the available options.
Afixed	Used to fit model with fixed Mode A weights.
Bfixed	Used to fit model with fixed Mode B weights.
Cfixed	Used to fit model with fixed Mode C weights.
Dfixed	Used to fit model with fixed Mode D weights.
Astart	Starting Mode A weights. Default uses random weights.
Bstart	Starting Mode B weights. Default uses random weights.
Cstart	Starting Mode C weights. Default uses random weights.
Dstart	Starting Mode D weights. Default uses random weights.
Astruc	Structure constraints for Mode A weights. See Note.
Bstruc	Structure constraints for Mode B weights. See Note.
Cstruc	Structure constraints for Mode C weights. See Note.
Dstruc	Structure constraints for Mode D weights. See Note.
Amodes	Mode ranges for Mode A weights (for unimodality constraints). See Note.

Bmodes	Mode ranges for Mode B weights (for unimodality constraints). See Note.
Cmodes	Mode ranges for Mode C weights (for unimodality constraints). See Note.
Dmodes	Mode ranges for Mode D weights (for unimodality constraints). See Note.
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	If TRUE, fitting progress is printed via txtProgressBar. Ignored if parallel=TRUE.
backfit	Should backfitting algorithm be used for cmls?

Details

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.

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. See Note.

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

Note

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 algorithm.

Structure constraints should be specified with a matrix of logicals (TRUE/FALSE), such that FALSE elements indicate a weight should be constrained to be zero. Default uses unstructured weights, i.e., a matrix of all TRUE values.

When using unimodal constraints, the *modes inputs can be used to specify the mode search range for each factor. These inputs should be matrices with dimension c(2,nfac) where the first row gives the minimum mode value and the second row gives the maximum mode value (with respect to the indicies of the corresponding weight matrix).

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

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. doi:10.1016/01679473(94)901325

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803. doi:10.1002/bimj.201600045

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics, 6*, 164-189. doi:10.1002/sapm192761164

See Also

The cpd function implements an N-way extension without constraints.

The fitted.parafac function creates the model-implied fitted values from a fit "parafac" object.

The resign.parafac function can be used to resign factors from a fit "parafac" object.

The rescale.parafac function can be used to rescale factors from a fit "parafac" object.

The reorder.parafac function can be used to reorder factors from a fit "parafac" object.

The cmls function (from **CMLS** package) is called as a part of the alternating least squares algorithm.

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), nrow = mydim[1], ncol = nf)
Bmat <- matrix(runif(mydim[2]*nf), nrow = mydim[2], ncol = nf)
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Xmat <- tcrossprod(Amat, krprod(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = 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("uncons", "nonneg", "nonneg"))
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.out = 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), nrow = mydim[2], ncol = nf), nv = 0)$u
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Cstruc <- Cmat > 0.5
Cmat <- Cmat * Cstruc
Dmat <- matrix(runif(mydim[4]*nf), nrow = mydim[4], ncol = nf)
Xmat <- tcrossprod(Amat, krprod(Dmat, krprod(Cmat, Bmat)))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR = 1
X <- Xmat + Emat

# fit Parafac model (unimodal and smooth A, orthogonal B,
#                    non-negative and structured C, non-negative D)
pfac <- parafac(X, nfac = nf, nstart = 1, Cstruc = Cstruc,
                const = c("unismo", "orthog", "nonneg", "nonneg"))
pfac

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

## 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), nrow = mydim[1], ncol = nf)
Bmat <- matrix(runif(mydim[2]*nf), nrow = mydim[2], ncol = nf)
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Xmat <- tcrossprod(Amat, krprod(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = 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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl, library(multiway))
clusterSetRNGStream(cl, 1)
system.time({pfac <- parafac(X, nfac = nf, parallel = TRUE, cl = cl)})
pfac
stopCluster(cl)

## End(Not run)
```

`parafac2` *Parallel Factor Analysis-2*

Description

Fits Richard A. Harshman's Parallel Factors-2 (Parafac2) model to 3-way or 4-way ragged data arrays. Parameters are estimated via alternating least squares 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,
         Gmodes = NULL, Bmodes = NULL, Cmodes = NULL, Dmodes = NULL,
         maxit = 500, ctol = 1e-4, parallel = FALSE, cl = NULL,
         output = c("best", "all"), verbose = TRUE, backfit = 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`	Character vector of length 3 or 4 giving the constraints for each mode (defaults to unconstrained). See `const` for the 24 available options.
`control`	List of parameters controlling options for smoothness constraints. This is passed to `const.control`, which describes the available options.
`Gfixed`	Used to fit model with fixed Phi matrix: `crossprod(Gfixed) = Phi`.

Bfixed	Used to fit model with fixed Mode B weights.
Cfixed	Used to fit model with fixed Mode C weights.
Dfixed	Used to fit model with fixed Mode D weights.
Gstart	Starting Mode A crossproduct matrix: `crossprod(Gstart) = Phi`. Default uses random weights.
Bstart	Starting Mode B weights. Default uses random weights.
Cstart	Starting Mode C weights. Default uses random weights.
Dstart	Starting Mode D weights. Default uses random weights.
Gstruc	Structure constraints for Mode A crossproduct matrix: `crossprod(Gstruc) = Phistruc`. See Note.
Bstruc	Structure constraints for Mode B weights. See Note.
Cstruc	Structure constraints for Mode C weights. See Note.
Dstruc	Structure constraints for Mode D weights. See Note.
Gmodes	Mode ranges for Mode A weights (for unimodality constraints). Ignored.
Bmodes	Mode ranges for Mode B weights (for unimodality constraints). See Note.
Cmodes	Mode ranges for Mode C weights (for unimodality constraints). See Note.
Dmodes	Mode ranges for Mode D weights (for unimodality constraints). See Note.
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	If `TRUE`, fitting progress is printed via `txtProgressBar`. Ignored if `parallel=TRUE`.
backfit	Should backfitting algorithm be used for `cmls`?

Details

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

$$X[[l]][,,k] = \text{tcrossprod}(A[[l]] \,\%*\%\, \text{diag}(D[l,]*C[k,]), B) + E[[l]][,,k]$$
$$\text{subject to } \text{crossprod}(A[[l]]) = \text{Phi}$$

where `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]] = array(el, dim = c(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.

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

Note

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 algorithm.

Structure constraints should be specified with a matrix of logicals (TRUE/FALSE), such that FALSE elements indicate a weight should be constrained to be zero. Default uses unstructured weights, i.e., a matrix of all TRUE values.

When using unimodal constraints, the `*modes` inputs can be used to specify the mode search range for each factor. These inputs should be matrices with dimension `c(2,nfac)` where the first row gives the minimum mode value and the second row gives the maximum mode value (with respect to the indicies of the corresponding weight matrix).

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

References

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. doi:10.1007/S1133601393317

Helwig, N. E. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. *Biometrical Journal, 59*(4), 783-803. doi:10.1002/bimj.201600045

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. doi:10.1002/(SICI)1099-128X(199905/08)13:3/4<275::AIDCEM543>3.0.CO;2B

See Also

The `fitted.parafac2` function creates the model-implied fitted values from a fit `"parafac2"` object.

The `resign.parafac2` function can be used to resign factors from a fit `"parafac2"` object.

The `rescale.parafac2` function can be used to rescale factors from a fit `"parafac2"` object.

The `reorder.parafac2` function can be used to reorder factors from a fit `"parafac2"` object.

The `cmls` function (from **CMLS** package) is called as a part of the alternating least squares algorithm.

Examples

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

# create random data list with Parafac2 structure
set.seed(3)
mydim <- c(NA, 10, 20)
nf <- 2
nk <- rep(c(50, 100, 200), length.out = mydim[3])
Gmat <- matrix(rnorm(nf^2), nrow = nf, ncol = nf)
Bmat <- matrix(runif(mydim[2]*nf), nrow = mydim[2], ncol = nf)
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Xmat <- Emat <- Amat <- vector("list", mydim[3])
```

```
for(k in 1:mydim[3]){
  Amat[[k]] <- matrix(rnorm(nk[k]*nf), nrow = nk[k], ncol = nf)
  Amat[[k]] <- svd(Amat[[k]], nv = 0)$u %*% Gmat
  Xmat[[k]] <- tcrossprod(Amat[[k]] %*% diag(Cmat[k,]), Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k]*mydim[2]), nrow = nk[k], ncol = mydim[2])
}
Emat <- nscale(Emat, 0, ssnew = 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 <- rep(c(50,100,200), length.out = mydim[4])
Gmat <- matrix(rnorm(nf^2), nrow = nf, ncol = nf)
Bmat <- scale(matrix(rnorm(mydim[2]*nf), nrow = mydim[2], ncol = 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, nrow = mydim[4], ncol = nf), center = FALSE)
Xmat <- Emat <- Amat <- vector("list",mydim[4])
for(k in 1:mydim[4]){
  aseq <- seq(-3, 3, length.out = nk[k])
  Amat[[k]] <- cbind(sin(aseq), sin(abs(aseq)), exp(-aseq^2))
  Amat[[k]] <- svd(Amat[[k]], nv = 0)$u %*% Gmat
  Xmat[[k]] <- array(tcrossprod(Amat[[k]] %*% 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, ssnew = 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("smooth", "uncons", "moninc", "nonneg"))
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 <- rep(c(50, 100, 200), length.out = mydim[3])
Gmat <- matrix(rnorm(nf^2), nrow = nf, ncol = nf)
Bmat <- matrix(runif(mydim[2]*nf), nrow = mydim[2], ncol = nf)
Cmat <- matrix(runif(mydim[3]*nf), nrow = mydim[3], ncol = nf)
Xmat <- Emat <- Hmat <- vector("list", mydim[3])
for(k in 1:mydim[3]){
  Hmat[[k]] <- svd(matrix(rnorm(nk[k] * nf), nrow = nk[k], ncol = nf), nv = 0)$u
  Xmat[[k]] <- tcrossprod(Hmat[[k]] %*% Gmat %*% diag(Cmat[k,]), Bmat)
  Emat[[k]] <- matrix(rnorm(nk[k] * mydim[2]), nrow = nk[k], mydim[2])
}
Emat <- nscale(Emat, 0, ssnew = 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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl, library(multiway))
clusterSetRNGStream(cl, 1)
system.time({pfac <- parafac2(X, nfac = nf, parallel = TRUE, cl = cl)})
pfac
stopCluster(cl)

## End(Not run)
```

---

print *Print Multi-Way Model Results*

Description

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

Usage

```
## S3 method for class 'cpd'
print(x,...)
## S3 method for class 'indscal'
print(x,...)
## S3 method for class 'mcr'
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 "cpd" (output from cpd), class "indscal" (output from indscal), class "mcr" (output from mcr), class "parafac" (output from parafac), class "parafac2" (output from parafac2), class "sca" (output from sca), or class "tucker" (output from tucker).
...	Ignored.

Details

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

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
# See examples for...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
#   tucker (Tucker Factor Analysis)
```

---

reorder *Reorder Multi-Way Factors*

Description

Reorders factors from a `multiway` object.

Usage

```
## S3 method for class 'cpd'
reorder(x, neworder, ...)
## S3 method for class 'indscal'
reorder(x, neworder, ...)
## S3 method for class 'mcr'
reorder(x, neworder, mode = "A", ...)
## 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 "cpd" (output from `cpd`), "indscal" (output from `indscal`), class "mcr" (output from `mcr`), 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 cpd, indscal, mcr, parafac, parafac2, sca, and tucker for more details.

Value

Same as input.

Author(s)

Nathaniel E. Helwig <helwig@umn.edu>

Examples

```
# See examples for...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INdividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
#   tucker (Tucker Factor Analysis)
```

---

rescale *Rescales Multi-Way Factors*

Description

Rescales factors from a multiway object.

Usage

```
## S3 method for class 'cpd'
rescale(x, mode = 1, newscale = 1, absorb = 3, ...)
## S3 method for class 'indscal'
rescale(x, mode = "B", newscale = 1, ...)
## S3 method for class 'mcr'
rescale(x, mode = "A", newscale = 1, absorb = "C", ...)
## 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 "mcr" (output from mcr), 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. For "cpd" objects, should be an integer between 1 and N.
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). For "cpd" objects, should be an integer between 1 and N.
...	Ignored.

Details

See cpd, indscal, mcr, 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. doi:10.1007/S1133601393317

Examples

```
# See examples for...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
#   tucker (Tucker Factor Analysis)
```

resign *Resigns Multi-Way Factors*

Description

Resigns factors from a multiway object.

Usage

```
## S3 method for class 'cpd'
resign(x, mode = 1, newsign = 1, absorb = 3, ...)
## S3 method for class 'indscal'
resign(x, mode = "B", newsign = 1, ...)
## S3 method for class 'mcr'
resign(x, mode = "A", newsign = 1, absorb = "C", ...)
## 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 "cpd" (output from cpd), "indscal" (output from indscal), class "mcr" (output from mcr), 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. For "cpd" objects, should be an integer between 1 and N.
newsign	Desired resigning for each column of specified 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). For "cpd" objects, should be an integer between 1 and N.
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 cpd, indscal, mcr, 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. doi:10.1007/S1133601393317

Examples

```
# See examples for...
#   cpd (Canonical Polyadic Decomposition)
#   indscal (INdividual Differences SCALing)
#   mcr (Multiway Covariates Regression)
#   parafac (Parallel Factor Analysis-1)
#   parafac2 (Parallel Factor Analysis-2)
#   sca (Simultaneous Component Analysis)
#   tucker (Tucker Factor Analysis)
```

---

sca *Simultaneous Component Analysis*

Description

Fits Timmerman and Kiers's four Simultaneous Component Analysis (SCA) models to a 3-way data array or a list of 2-way arrays with the same number of columns.

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, verbose = TRUE)
```

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.
verbose	If TRUE, fitting progress is printed via txtProgressBar. Ignored if parallel=TRUE.

Details

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.

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. doi:10.1007/S1133601393317

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. doi:10.1007/BF02296656

Examples

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

# create random data list with SCA-P structure
set.seed(3)
mydim <- c(NA,10,20)
nf <- 2
nk <- rep(c(50,100,200), length.out = mydim[3])
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, ssnew = 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 <- rep(c(50,100,200), length.out = mydim[3])
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, ssnew = 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 <- rep(c(50,100,200), length.out = mydim[3])
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, ssnew = 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 <- rep(c(50,100,200), length.out = mydim[3])
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, ssnew = 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 <- rep(c(50,100,200), length.out = mydim[3])
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, ssnew = 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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
clusterSetRNGStream(cl, 1)
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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
clusterSetRNGStream(cl, 1)
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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
clusterSetRNGStream(cl, 1)
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.

Details

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

Value

Returns X raised to specified power.

Note

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

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, na.rm = FALSE)
```

Arguments

X	Numeric scalar, vector, list, matrix, or array.
na.rm	logical. Should missing values (including NaN) be removed?

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

Fits Ledyard R. Tucker's factor analysis model to 3-way or 4-way data arrays. Parameters are estimated via alternating least squares.

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 = TRUE)
```

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`	If `TRUE`, fitting progress is printed via `txtProgressBar`. Ignored if `parallel=TRUE`.

Details

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.

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. doi:10.1007/BF02293599

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31*, 279-311. doi:10.1007/BF02289464

Examples

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

##########   TUCKER3   ##########

# create random data array with Tucker3 structure
set.seed(3)
mydim <- c(50,20,5)
nf <- c(3,2,3)
Amat <- matrix(rnorm(mydim[1]*nf[1]), mydim[1], nf[1])
Amat <- svd(Amat, nu = nf[1], nv = 0)$u
Bmat <- matrix(rnorm(mydim[2]*nf[2]), mydim[2], nf[2])
Bmat <- svd(Bmat, nu = nf[2], nv = 0)$u
Cmat <- matrix(rnorm(mydim[3]*nf[3]), mydim[3], nf[3])
Cmat <- svd(Cmat, nu = nf[3], nv = 0)$u
Gmat <- matrix(rnorm(prod(nf)), nf[1], prod(nf[2:3]))
Xmat <- tcrossprod(Amat %*% Gmat, kronecker(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker3 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)


##########   TUCKER2   ##########

# create random data array with Tucker2 structure
set.seed(3)
mydim <- c(50, 20, 5)
nf <- c(3, 2, mydim[3])
Amat <- matrix(rnorm(mydim[1]*nf[1]), mydim[1], nf[1])
Amat <- svd(Amat, nu = nf[1], nv = 0)$u
Bmat <- matrix(rnorm(mydim[2]*nf[2]), mydim[2], nf[2])
Bmat <- svd(Bmat, nu = nf[2], nv = 0)$u
Cmat <- diag(nf[3])
Gmat <- matrix(rnorm(prod(nf)), nf[1], prod(nf[2:3]))
Xmat <- tcrossprod(Amat %*% Gmat, kronecker(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker2 model
tuck <- tucker(X, nfac = nf, nstart = 1, Cfixed = diag(nf[3]))
tuck

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

```
##########   TUCKER1   ##########

# create random data array with Tucker1 structure
set.seed(3)
mydim <- c(50, 20, 5)
nf <- c(3, mydim[2:3])
Amat <- matrix(rnorm(mydim[1]*nf[1]), mydim[1], nf[1])
Amat <- svd(Amat, nu = nf[1], nv = 0)$u
Bmat <- diag(nf[2])
Cmat <- diag(nf[3])
Gmat <- matrix(rnorm(prod(nf)), nf[1], prod(nf[2:3]))
Xmat <- tcrossprod(Amat %*% Gmat, kronecker(Cmat, Bmat))
Xmat <- array(Xmat, dim = mydim)
Emat <- array(rnorm(prod(mydim)), dim = mydim)
Emat <- nscale(Emat, 0, ssnew = sumsq(Xmat))   # SNR=1
X <- Xmat + Emat

# fit Tucker1 model
tuck <- tucker(X, nfac = nf, nstart = 1,
               Bfixed = diag(nf[2]), Cfixed = diag(nf[3]))
tuck

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

# closed-form Tucker1 solution via SVD
tsvd <- svd(matrix(X, nrow = mydim[1]), nu = nf[1], nv = nf[1])
Gmat0 <- t(tsvd$v %*% diag(tsvd$d[1:nf[1]]))
Xhat0 <- array(tsvd$u %*% Gmat0, dim = mydim)
sum((Xmat-Xhat0)^2) / prod(mydim)

# get Mode A weights and core array
tuck0 <- NULL
tuck0$A <- tsvd$u                   # A weights
tuck0$G <- array(Gmat0, dim = nf)   # core array



##########   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, ssnew = 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, ssnew = 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)
cl <- makeCluster(detectCores())
ce <- clusterEvalQ(cl,library(multiway))
clusterSetRNGStream(cl, 1)
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://www.niaaa.nih.gov/publications/surveillance-reports

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. doi:10.1002/bimj.201600045

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)

# 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("smooth", NA, NA))

# 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("smooth", NA, NA), Bstruc = Bstruc)
```

Index