Package 'GPArotation'

August 22, 2026

Version 2026.8-2

Title Gradient Projection Factor Rotation

Depends R (>= 3.5.0)

Description Gradient projection algorithms for orthogonal and oblique rotation of factor loadings matrices in factor analysis. Implements a comprehensive set of rotation criteria including quartimax, quartimin, oblimin, geomin, simplimax, the Crawford-Ferguson family, and target rotation, among others. Supports multiple random starts. For details see Bernaards and Jennrich (2005) <doi:10.1177/0013164404272507>.

License GPL (>= 2)

URL https://cran.r-project.org/package=GPArotation

Imports stats,grDevices,graphics, utils

VignetteBuilder utils

LazyData yes

NeedsCompilation no

Author Coen Bernaards [aut, cre],
Paul Gilbert [aut],
Robert Jennrich [aut]

Maintainer Coen Bernaards <cab.gparotation@gmail.com>

Repository CRAN

Date/Publication 2026-08-22 15:40:09 UTC

Contents

00.GPArotation 2
CCAI 7
echelon 10
eiv 12
GPA 14
GriffithMulaik 21
Harman 24

lp 25
plot.GPArotation 27
plot2fOrthComparison 28
print.GPArotation 30
Random.Start 32
rotations 33
Thurstone 40
vgQ 41
WansbeekMeijer 43

Index **45**

`00.GPArotation` *Gradient Projection Algorithms for Factor Rotation*

Description

GPA Rotation for Factor Analysis

The GPArotation package contains functions for the rotation of factor loadings matrices. The functions implement Gradient Projection (GP) algorithms for orthogonal and oblique rotation. Additionally, a number of rotation criteria are provided. The GP algorithms minimize the rotation criterion function and provide the corresponding rotation matrix. For oblique rotation, the covariance/correlation matrix of the factors is also provided. The rotation criteria implemented in this package are described in Bernaards and Jennrich (2005) in addition to a number of others. Theory of the GP algorithm is described in Jennrich (2001, 2002).

Vignettes are provided covering general usage, local minima diagnostics, bifactor rotation and reliability, and a visual walkthrough of the gradient projection algorithm. Access them via `browseVignettes("GPArotation")`.

Package:	GPArotation
Depends:	R (>= 3.5.0)
License:	GPL Version 2.

Index of functions:

Rotations using gradient projection algorithms

`oblimin`	Oblimin rotation
`quartimin`	Quartimin rotation
`targetT`	Orthogonal target rotation
`targetQ`	Oblique target rotation
`pstT`	Orthogonal partially specified target rotation
`pstQ`	Oblique partially specified target rotation
`oblimax`	Oblimax rotation
`entropy`	Minimum entropy rotation
`quartimax`	Quartimax rotation

Varimax	Varimax rotation
simplimax	Simplimax rotation
bentlerT	Orthogonal Bentler invariant pattern simplicity rotation
bentlerQ	Oblique Bentler invariant pattern simplicity rotation
tandemI	Tandem criteria principle I rotation
tandemII	Tandem criteria principle II rotation
geominT	Orthogonal Geomin rotation
geominQ	Oblique Geomin rotation
bigeominT	Orthogonal Bi-Geomin rotation
bigeominQ	Oblique Bi-Geomin rotation
cfT	Orthogonal Crawford-Ferguson family rotation
cfQ	Oblique Crawford-Ferguson family rotation
equamax	Equamax rotation
parsimax	Parsimax rotation
infomaxT	Orthogonal Infomax rotation
infomaxQ	Oblique Infomax rotation
mccammon	McCammon minimum entropy ratio rotation
binormamin	Binormamin rotation
varimin	Varimin rotation
bifactorT	Orthogonal bifactor rotation
bifactorQ	Oblique bifactor rotation
lpT	Orthogonal L^p rotation
lpQ	Oblique L^p rotation

Core gradient projection algorithms

GPForth	Orthogonal rotation function
GPFoblq	Oblique rotation function

Random-start wrappers and internal engine

GPFRSorth	Random-start wrapper for orthogonal rotation
GPFRSoblq	Random-start wrapper for oblique rotation

2D orthogonal trajectory plot

plot2fOrthComparison	2D trajectory plot for 2 factor orthogonal rotation

S3 methods

print.GPArotation	Print rotated solution with simple structure measures
summary.GPArotation	Summary with pattern, structure, and diagnostics
plot.GPArotation	Plot: salient, pairs, profile, vector, target, heatmap, trajectory, diagnostics, residuals
residuals.GPArotation	Observed minus modeled residual of correlation matrix
update	update [The R Stats Package]

Data sets

Harman8	Harman's 8 physical variables; centroid loadings
NetherlandsTV	Wansbeek and Meijer Netherlands TV viewership; correlation matrix
box20	Thurstone's 20 box variables; unrotated factor loadings
box26	Thurstone's 26 box variables; unrotated factor loadings
CCAI	CCAI Climate-Friendly Purchasing Choices domain; correlation matrix, pattern matrix, and factor interco
GriffithMulaik	Griffith and Mulaik interpersonal personality traits; 24-variable correlation matrix

Other rotations not using gradient projection algorithms

eiv	Errors-in-variables rotation
echelon	Echelon rotation
varimax	varimax [The R Stats Package]
promax	promax [The R Stats Package]

Legacy gradient projection algorithms (code unchanged since 2008)

GPForth.legacy	Orthogonal rotation, original implementation (not exported)
GPFoblq.legacy	Oblique rotation, original implementation (not exported)

Utility functions

Random.Start	Random starting matrix for factor rotation
GPForth.lp	Single-start L^p orthogonal rotation
GPFoblq.lp	Single-start L^p oblique rotation

Utility functions (not exported)

.GPA_RS_engine	Internal random-start engine
.sortGPALoadings	Sort and sign-correct factors

NormalizingWeight	Normalizing weights utility
calc_AUC	AUC simple structure measure
calc_FSI	Factor Simplicity Index
calc_simplicity	Hoffman, Gini, Bentler simplicity indices
calc_hyperplane	Hyperplane count
calc_fitstats	ML fit statistics: RMSEA, SRMR
plot_trajectory	Helper function for plot2fOrthComparison
plot_gpa_diagnostics	Helper function for plot2fOrthComparison
plot_algorithm_comparison	Helper function for plot2fOrthComparison
.reconstruct_trajectory	Helper function for plot2fOrthComparison
plotRotationLandscape	Helper function for plot2fOrthComparison
plot_landscape_trajectory	Helper function for plot2fOrthComparison

Value, gradient, rotation criterion functions (not exported)

vgQ.oblimin	Oblimin
vgQ.quartimin	Quartimin
vgQ.target	Target
vgQ.pst	Partially specified target
vgQ.oblimax	Oblimax
vgQ.entropy	Minimum entropy
vgQ.quartimax	Quartimax
vgQ.varimax	Varimax
vgQ.simplimax	Simplimax
vgQ.bentler	Bentler invariant pattern simplicity
vgQ.tandemI	Tandem criteria principle I
vgQ.tandemII	Tandem criteria principle II
vgQ.geomin	Geomin
vgQ.bigeomin	Bi-Geomin
vgQ.cf	Crawford-Ferguson family
vgQ.infomax	Infomax
vgQ.mccammon	McCammon minimum entropy ratio
vgQ.varimin	Varimin
vgQ.bifactor	Bifactor
vgQ.lp.wls	Weighted least squares for L^p rotation

Vignettes

GPA1guide	Gradient Projection Factor Rotation (main guide)
GPA2local	Assessing Local Minima in Factor Rotation
GPA3bifactor	Bifactor Rotation and Reliability Coefficients
GPA4fitstats	Factor Model Fit and Simple Structure Diagnostics

Author(s)

Coen A. Bernaards and Robert I. Jennrich with some R modifications by Paul Gilbert.

References

The software reference is:

Bernaards, C.A. and Jennrich, R.I. (2005). Gradient projection algorithms and software for arbitrary rotation criteria in factor analysis. *Educational and Psychological Measurement*, **65**, 676–696. doi:10.1177/0013164404272507

Theory of gradient projection algorithms:

Jennrich, R.I. (2001). A simple general procedure for orthogonal rotation. *Psychometrika*, **66**, 289–306. doi:10.1007/BF02294840

Jennrich, R.I. (2002). A simple general method for oblique rotation. *Psychometrika*, **67**, 7–19. doi:10.1007/BF02294706

A clear and accessible introduction to gradient projection algorithms for factor rotation is provided in:

Mansolf, M. and Reise, S.P. (2016). Exploratory bifactor analysis: The Schmid-Leiman orthogonalization and Jennrich-Bentler analytic rotations. *Multivariate Behavioral Research*, **51**(5), 698–717. doi:10.1080/00273171.2016.1215898

Barzilai-Borwein step size:

Barzilai, J. and Borwein, J.M. (1988). Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, **8**, 141–148. doi:10.1093/imanum/8.1.141

Cayley transform retraction:

Wen, Z. and Yin, W. (2013). A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, **142**, 397–434. doi:10.1007/s1010701205841

Non-monotone line search:

Grippo, L., Lampariello, F., and Lucidi, S. (1986). A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, **23**(4), 707–716. doi:10.1137/0723046

Zhang, H. and Hager, W.W. (2004). A nonmonotone line search technique and its application to unconstrained optimization. *SIAM Journal on Optimization*, **14**(4), 1043–1056. doi:10.1137/S1052623403428208

Simple structure measures:

Liu, X., Wallin, G., Chen, Y., and Moustaki, I. (2023). Rotation to sparse loadings using L^p losses and related inference problems. *Psychometrika*, **88**(2), 527–553. doi:10.1007/s1133602309911y

Lorenzo-Seva, U. (2003). A factor simplicity index. *Psychometrika*, **68**(1), 49–60. doi:10.1007/BF02296652

See Also

GPFRSorth, GPFRSoblq, rotations, vgQ, simple_structure, plot.GPArotation browseVignettes("GPArotation")

CCAI *The CCAI Climate-Friendly Purchasing Choices Domain*

Description

Correlation matrix, factor pattern matrix, and factor intercorrelations for the Climate Change Action Inventory (CCAI) (Barchard et al., 2021) Climate-Friendly Purchasing Choices domain, analyzed in Bi and Barchard (2024). The scale measures the frequency with which individuals make purchasing choices aimed at reducing climate change. Data were collected from 500 United States MTurk workers. After 15 climate change deniers and 24 multivariate outliers were removed, 461 participants remained. Climate change deniers were excluded because the scale measures behaviors intended to reduce climate change — including individuals who do not believe climate change exists would be inconsistent with the measure's intent.

The Climate Change Action Inventory contains eight domains; the Climate-Friendly Purchasing Choices domain analyzed here is one of them.

Usage

```
data(CCAI, package = "GPArotation")
```

Format

Three objects are loaded by `data(CCAI)`:

- `CCAI_R`: a 14×14 numeric correlation matrix. Row and column names are item labels `CCAI1` through `CCAI14`, ordered by factor to match `CCAI_pattern`.
- `CCAI_pattern`: a 14×3 numeric matrix of factor pattern coefficients. Row names are item labels `CCAI1` through `CCAI14`, ordered by factor to facilitate interpretation. Column names are factor labels `SustainableOptions`, `CollectiveAction`, and `AvoidBuyingNew`.
- `CCAI_Phi`: a 3×3 numeric matrix of factor intercorrelations. Row and column names are the three factor labels.

Details

The CCAI Climate-Friendly Purchasing Choices domain consists of 14 items. Items used a nine-point frequency scale ranging from 1 (less than once a year) to 9 (at least 14 times a week). For full details on study design, data collection, analysis, and interpretation see Bi and Barchard (2024).

`CCAI_R` is the observed 14×14 correlation matrix for the 14 items of the Climate-Friendly Purchasing Choices domain. The correlation matrix was kindly provided by the authors and has not been published separately.

`CCAI_pattern` is a 14×3 factor pattern matrix from principal components extraction followed by direct oblimin rotation, reported in Table 2 of Bi and Barchard (2024) (the Table is labeled "Factor Structure" in the paper but contains pattern coefficients). The three factors are: Choosing Sustainable Options (F1), Supporting Collective Action (F2), and Avoiding Buying New (F3).

Item	Description	F1	F2	F3
CCAI8	Choose products with less impact on climate change	.95	.00	-.02
CCAI6	Choose products with less packaging	.91	-.04	.01
CCAI7	Choose products made locally	.89	-.09	.07
CCAI11	Encourage others to choose climate-friendly products	.56	.40	.05
CCAI12	Problem solve to reduce impact of purchases	.52	.44	.04
CCAI10	Encourage others to buy less	.41	.44	.12
CCAI14	Give time/money to orgs reducing purchase impact	-.01	.97	-.02
CCAI13	Give time/money to orgs reducing purchases	.02	.93	.01
CCAI5	Donate to charity rather than buying a gift	-.02	.78	.20
CCAI2	Use borrowed/rented/digital rather than buying	-.02	-.18	.90
CCAI4	Buy used rather than new	.00	.15	.76
CCAI1	Repair rather than buying replacements	.03	.06	.76
CCAI3	Use borrowed/rented tools rather than buying	.10	.21	.62
CCAI9	Donate or sell old possessions	.22	.27	.46

`CCAI_Phi` is a 3×3 factor intercorrelation matrix. All three intercorrelations exceed 0.50.

	F1	F2	F3
Choosing Sustainable Options	1.00	0.59	0.59
Supporting Collective Action	0.59	1.00	0.53
Avoiding Buying New	0.59	0.53	1.00

The Tandem Paradigm on Correlated Data: The `CCAI` Purchasing Choices domain illustrates the Tandem criteria on highly correlated factors. Because the underlying dimensions are strongly intercorrelated ($\phi_{ij} > 0.50$), **TandemI** pulls shared variance into a single dominant general factor (approximately 59 percent of total variance), alerting the researcher to a unified behavioral core.

TandemII forces an orthogonal simple structure, but the 90-degree constraint creates an artificial compromise when true dimensions are correlated — smearing loadings across factors. Comparing TandemII to oblique **Oblimin** using the package summary diagnostics (`calc_AUC`, `calc_hyperplane`) provides empirical evidence that an oblique structure is conceptually superior for these data.

See `vignette("GPA3bifactor", package = "GPArotation")` for a bifactor analysis of these data.

Note

The raw data are publicly available on the Open Science Framework. As the data are subject to a license, users should consult the OSF page for terms of use before using the raw data directly: https://osf.io/h38yb/overview.

References

Barchard, K.A., Okagawa, K., Hoffman, C.K., and Odents, O. (2021). *Climate Change Action Inventory.* Unpublished psychological test. Available from kim.barchard@unlv.edu.

Bi, Y. and Barchard, K.A. (2024). Purchasing choices that reduce climate change: An exploratory factor analysis. *Spectra Undergraduate Research Journal*, **3**(2), 8–14. doi:10.9741/27667227.1028.

See Also

rotations, bifactorT, eigen, factanal, Harman, Thurstone, WansbeekMeijer, GriffithMulaik

Examples

```
data(CCAI, package = "GPArotation")

# Observed correlation matrix
round(CCAI_R, 2)

# Published pattern matrix and factor intercorrelations
round(CCAI_pattern, 2)
round(CCAI_Phi, 2)

# Reproduce published analysis: PCA extraction via eigendecomposition
# followed by oblimin rotation. No additional packages required.
# Gives the same result as psych::principal used by Bi and Barchard (2024).
ev          <- eigen(CCAI_R)
k           <- 3
L_unrotated <- ev$vectors[, 1:k] %*% diag(sqrt(ev$values[1:k]))
rownames(L_unrotated) <- colnames(CCAI_R)
res.ob <- oblimin(L_unrotated, randomStarts = 100)

# Sort and extract loadings for comparison
res_sorted <- GPArotation:::.sortGPALoadings(res.ob)
L_repro    <- unclass(res_sorted$loadings)

# Compare reproduced vs published side by side, alternating by factor
comparison <- cbind(round(L_repro[, 1], 2), round(CCAI_pattern[, 1], 2),
                    round(L_repro[, 2], 2), round(CCAI_pattern[, 2], 2),
                    round(L_repro[, 3], 2), round(CCAI_pattern[, 3], 2))
colnames(comparison) <- c("F1.repro", "F1.pub",
                          "F2.repro", "F2.pub",
                          "F3.repro", "F3.pub")
print(comparison)

# --- Tandem criteria ---
# Stage 1: TandemI reveals factor redundancy structure.
# One dominant general factor (SS = 8.3, 59% variance) suggests
# highly correlated underlying constructs.
res.t1 <- tandemI(L_unrotated, randomStarts = 100)
print(res.t1)

# Stage 2: TandemII for simple structure (orthogonal constraint).
res.t2 <- tandemII(L_unrotated, randomStarts = 100)
summary(res.t2)

# For heavily correlated constructs, oblique rotation achieves
# substantially cleaner simple structure than TandemII.
# Compare AUC and hyperplane counts across methods.
cat("           AUC    Hyperplane\n")
cat("TandemII  ",
```

```
    round(GPArotation:::calc_AUC(res.t2)$AUC_mean, 3), "  ",
    sum(abs(unclass(res.t2$loadings)) < 0.1), "loadings\n")
cat("Oblimin   ",
    round(GPArotation:::calc_AUC(res.ob)$AUC_mean, 3), "  ",
    sum(abs(unclass(res.ob$loadings)) < 0.1), "loadings\n")

# Orthogonal bifactor rotation using MLE extraction
fa.un <- factanal(factors = 3, covmat = CCAI_R, n.obs = 461,
                  rotation = "none")
bif <- bifactorT(fa.un)
print(bif, sortLoadings = FALSE, digits = 3, cutoff = 0.05)
```

`echelon` *Echelon Rotation*

Description

Rotates a factor loading matrix to an echelon parameterization.

Usage

```
echelon(A, reference = seq(ncol(A)), ...)
```

Arguments

`A`	a factor loading matrix.
`reference`	integer vector indicating which rows of the loading matrix are used to determine the rotation transformation. Default uses the first k rows. If the submatrix indicated by `reference` is singular, a different choice of rows must be supplied.
`...`	additional arguments discarded.

Details

The loading matrix is rotated so that the k rows indicated by `reference` form the Cholesky factorization given by `t(chol(L[reference,] %*% t(L[reference,])))`. This defines the rotation transformation, which is then applied to all rows to give the rotated loading matrix.

The optimization is not iterative and does not use the gradient projection algorithm. The function can be used directly or passed to factor analysis functions like `factanal` via the `rotation` argument.

This parameterization has several useful properties:

1. It can be useful for comparison with published results in this parameterization.
2. Standard errors are more straightforward to compute because the solution corresponds to an unconstrained optimization.
3. Models with k and $k + 1$ factors are nested, making it straightforward to test the k-factor model versus the $(k + 1)$-factor model. In particular, the Wald test and LM test can be used in addition to the LR test. The test of a k-factor model versus a $(k + 1)$-factor model is a joint test of whether all free parameters (loadings) in the $(k + 1)$st column are zero.

4. For some purposes, only the subspace spanned by the factors matters, not the specific parameterization within this subspace.
5. Back-predicted indicators (the explained portion of the indicators) do not depend on the rotation method. Combined with the greater ease of obtaining correct standard errors, this allows easier and more accurate prediction standard errors.
6. This parameterization and its standard errors can be used to detect identification problems (McDonald, 1999, pp. 181–182).

One use of echelon rotation is obtaining good starting values for subsequent rotation, though it may seem counterintuitive to rotate towards this solution afterwards.

Value

A `GPArotation` object which is a list with elements:

`loadings`	The rotated loadings matrix.
`Th`	The rotation matrix.
`method`	A string indicating the rotation method (`"echelon"`).
`orthogonal`	Always `TRUE` (an orthogonal solution is assumed; Φ is the identity matrix).
`convergence`	Always `TRUE` (the optimization is not iterative).

Author(s)

Erik Meijer and Paul Gilbert.

References

McDonald, R.P. (1999). *Test Theory: A Unified Treatment*. Erlbaum.

Wansbeek, T. and Meijer, E. (2000). *Measurement Error and Latent Variables in Econometrics*. North-Holland.

See Also

`eiv`, `rotations`, `GPForth`, `GPFoblq`

Examples

```
data("WansbeekMeijer", package = "GPArotation")
fa.un <- factanal(factors = 2, covmat = NetherlandsTV,
                  rotation = "none", n.obs = 2154)

# Echelon rotation - first k rows as reference (default)
fa.ech <- echelon(fa.un)
print(fa.ech)

# Equivalent via factanal rotation argument
fa.ech2 <- factanal(factors = 2, covmat = NetherlandsTV,
                    rotation = "echelon")
```

```
# Echelon rotation with a different reference set
fa.ech3 <- echelon(fa.un, reference = 6:7)
print(fa.ech3)
```

`eiv` *Errors-in-Variables Rotation*

Description

Rotates a factor loading matrix to an errors-in-variables representation.

Usage

```
eiv(A, identity = seq(ncol(A)), ...)
```

Arguments

`A`	a factor loading matrix.
`identity`	integer vector indicating which rows of the loading matrix should form an identity matrix. Default uses the first k rows. If inverting the submatrix indicated by `identity` fails, a different choice of rows must be supplied.
`...`	additional arguments discarded.

Details

The loading matrix is rotated so that the k rows indicated by `identity` form an identity matrix, with the remaining $M - k$ rows as free parameters. Φ is also free.

The optimization is not iterative and does not use the gradient projection algorithm. The function can be used directly or passed to factor analysis functions like `factanal` via the `rotation` argument.

Viewed as a rotation method it is oblique, with an explicit solution. Given an initial loadings matrix L partitioned as $L = (L_1^T, L_2^T)^T$, the rotated loadings matrix is $(I, (L_2 L_1^{-1})^T)^T$ and $\Phi = L_1 L_1^T$, where I is the $k \times k$ identity matrix. It is assumed that $\Phi = I$ for the initial loadings matrix.

Not all authors consider this representation to be a rotation in the strict sense.

This parameterization has several useful properties:

1. It can be useful for comparison with published results in this parameterization.
2. Standard errors are more straightforward to compute because the solution corresponds to an unconstrained optimization.
3. One may have prior knowledge about which reference variables load on only one factor without imposing restrictive constraints on other loadings — in this sense it has similarities to CFA.
4. For some purposes, only the subspace spanned by the factors matters, not the specific parameterization within this subspace.

5. Back-predicted indicators (the explained portion of the indicators) do not depend on the rotation method. Combined with the greater ease of obtaining correct standard errors, this allows easier and more accurate prediction standard errors.

One use of this parameterization is obtaining good starting values for subsequent rotation, though it may seem counterintuitive to rotate towards this solution afterwards.

Note that the rotated loadings for non-identity rows may be large in magnitude and should not be interpreted as standard factor loadings. Φ is the factor covariance matrix $L_1 L_1^T$ and is not a correlation matrix — diagonal values are not constrained to 1.

Value

A `GPArotation` object which is a list with elements:

`loadings`	The rotated loadings matrix.
`Th`	The rotation matrix.
`method`	A string indicating the rotation method (`"eiv"`).
`orthogonal`	Always `FALSE` (oblique rotation).
`convergence`	Always `TRUE` (the optimization is not iterative).
`Phi`	The covariance matrix of the rotated factors.

Author(s)

Erik Meijer and Paul Gilbert.

References

Hägglund, G. (1982). Factor analysis by instrumental variables methods. *Psychometrika*, **47**, 209–222. doi:10.1007/BF02296276

Lewin-Koh, S.C. and Amemiya, Y. (2003) Heteroscedastic factor analysis. *Biometrika*, **90**(1), 85–97. doi:10.1093/biomet/90.1.85

Wansbeek, T. and Meijer, E. (2000). *Measurement Error and Latent Variables in Econometrics*. North-Holland.

See Also

`echelon`, `rotations`, `GPForth`, `GPFoblq`

Examples

```
data("WansbeekMeijer", package = "GPArotation")
fa.un <- factanal(factors = 2, covmat = NetherlandsTV,
                  rotation = "none")

# Eiv rotation: items 1 and 2 define the identity
fa.eiv <- eiv(fa.un)

# Equivalent via factanal rotation argument
```

```
fa.eiv2 <- factanal(factors = 2, covmat = NetherlandsTV,
                    rotation = "eiv")

# Rotated loadings
print(fa.eiv)

# Eiv rotation with a different identity set (items 6 and 7)
fa.eiv3 <- eiv(fa.un, identity = 6:7)
print(fa.eiv3)
```

GPA *Core Algorithms and Random-Start Wrappers*

Description

Gradient projection rotation optimization routines for orthogonal and oblique factor rotation. These functions can be used directly to rotate a loadings matrix, or indirectly through a rotation objective passed to a factor estimation routine such as `factanal`.

Usage

```
GPForth(A, Tmat=diag(ncol(A)), normalize=FALSE, eps=1e-5, maxit=2000,
   method="varimax", methodArgs=NULL, algorithm = "bb", fwindow = 10)
GPFoblq(A, Tmat=diag(ncol(A)), normalize=FALSE, eps=1e-5, maxit=2000,
   method="quartimin", methodArgs=NULL, algorithm = "bb", fwindow = 10)

GPFRSorth(A, Tmat=NULL, normalize=FALSE, eps=1e-5, maxit=2000, method="varimax",
    methodArgs=NULL, randomStarts=0, algorithm = "bb", fwindow = 10, ...)
GPFRSoblq(A, Tmat=NULL, normalize=FALSE, eps=1e-5, maxit=2000, method="quartimin",
    methodArgs=NULL, randomStarts=0, algorithm = "bb", fwindow = 10, ...)
```

Arguments

`A`	initial factor loadings matrix for which the rotation criterion is to be optimized.
`Tmat`	initial rotation matrix.
`normalize`	see details.
`eps`	convergence is assumed when the norm of the gradient is smaller than `eps`.
`maxit`	maximum number of iterations allowed in the main loop.
`method`	rotation objective criterion.
`methodArgs`	a list of additional arguments passed to the rotation objective.
`randomStarts`	number of random starts (`GPFRSorth` and `GPFRSoblq` only).
`algorithm`	Line-search strategy used to calculate step size alpha.
`fwindow`	integer. Number of previous criterion values used in the non-monotone line search.
`...`	additional arguments passed to `GPForth` or `GPFoblq`, such as `eps` and `maxit` when calling via `GPFRSorth` or `GPFRSoblq`.

Details

The GPFRSorth and GPFRSoblq functions serve as the primary user interfaces for orthogonal and oblique rotations, respectively. They act as wrappers for the core GP algorithms (GPForth for orthogonal rotation and GPFoblq for oblique rotation), extending them with the ability to perform multiple random starts. Any additional arguments provided to these wrappers are passed directly down to the underlying GP algorithms. While the wrappers are generally recommended, the core functions GPForth and GPFoblq can also be invoked directly.

All of these functions require an initial loadings matrix, A, which fixes the equivalence class over which the optimization is performed. This matrix must be the solution to an orthogonal factor analysis problem, such as one obtained from factanal or another factor estimation routine.

Mathematically, a general rotation of a matrix A is defined as A %*% solve(t(Th)). In the case of orthogonal rotation, the initial rotation matrix Tmat is orthonormal, which simplifies the rotation formula to A %*% Th. In all scenarios, the final rotation matrix Th is computed by the GP rotation algorithm.

An accessible introduction to gradient projection algorithms for factor rotation is provided in Mansolf and Reise (2016).

The normalize **argument:** The normalize argument specifies whether and how the loadings matrix should be normalized prior to rotation, and subsequently denormalized after rotation.

- If FALSE (the default), no normalization is performed.
- If TRUE, Kaiser normalization is applied so that the squared row entries of the normalized matrix A sum to 1.0. This procedure is sometimes referred to as Horst normalization.
- If provided as a **vector** (which must have a length equal to the number of indicators, i.e., the number of rows in A), the columns of A are divided by this vector before rotation and multiplied by it afterward.
- If provided as a **function**, it can be used to apply a custom normalization scheme. The function must take A as an argument and return a vector, which is then applied in the same manner as the vector input described above. See NormalizingWeight for an example implementing Cureton-Mulaik normalization.

For a detailed investigation into how normalization affects factor rotations, including its potential impact on the qualitative interpretation of loadings, see Nguyen and Waller (2022).

The method **argument:** The method argument takes a string specifying the rotation objective function. By default, oblique rotations use "quartimin", while orthogonal rotations default to "varimax". The package supports a comprehensive suite of rotation objectives: "oblimin", "quartimin", "target", "pst", "oblimax", "entropy", "quartimax", "Varimax", "simplimax", "bentler", "tandemI", "tandemII", "geomin", "cf", "infomax", "mccammon", "bifactor", "lp", and "varimin".

Internally, this string is prefixed with "vgQ." to invoke the actual calculation function (see vgQ for underlying mathematical details). It is important to note that several rotation criteria — specifically "oblimin", "target", "pst", "simplimax", "geomin", "cf", and "lp" — require one or more supplementary arguments. These additional arguments can be seamlessly passed via the methodArgs list in the wrapper functions. Default values and direct usage examples for these arguments can be found in the rotations documentation.

The randomStarts **argument:** Because factor rotation criteria frequently suffer from local minima, trying multiple starting configurations can help identify a superior solution. The randomStarts

argument, available exclusively in the GPFRSorth and GPFRSoblq wrappers, facilitates this robust search approach.

- By default, randomStarts = 0, which defaults to using the identity matrix as the initial rotation matrix Tmat. The initial rotation matrix Tmat can also be set by the user.
- Setting randomStarts = 1 initializes Tmat with a single random matrix.
- Setting randomStarts > 1 attempts multiple random starts and returns the rotated loadings matrix that achieved the lowest criterion value f across all attempts. Note that this returned solution is technically still a local minimum, and is not guaranteed to be the global minimum. Users are encouraged to review the random start diagnostics detailed in the package examples.

Under the hood, an internal, unexported engine named .GPA_RS_engine safely manages the random start loop, tracks convergence diagnostics, and handles factor correlation matrix naming.

While the core algorithms GPForth and GPFoblq do not support the randomStarts argument directly, users can manually supply a single random initial rotation matrix to them using Tmat = Random.Start(ncol(A)).

The algorithm **argument:** The algorithm argument controls how the step size α is initialized at each iteration of the gradient projection loop. Three options are available:

"legacy" The original Bernaards and Jennrich (2005) heuristic: α is doubled at the start of each iteration and halved until the Armijo sufficient decrease condition is satisfied. Simple and reliable, particularly for smooth orthogonal criteria. This is the default for all orthogonal rotations and for target and PST rotations.

"bb" Barzilai-Borwein (BB) step size initialization. Uses the ratio of successive changes in the rotation matrix and projected gradient to estimate a good starting step size automatically:

$$\alpha_{BB} = \frac{\|\Delta T\|^2}{|\langle \Delta T, \Delta G_p \rangle|}$$

where $\Delta T = T_k - T_{k-1}$ and $\Delta G_p = G_{p,k} - G_{p,k-1}$ are the changes in the rotation matrix and projected gradient between consecutive iterations. The BB estimate is then used as the starting point for the Armijo line search. Benchmarking shows meaningful speedups for oblique rotation criteria on larger matrices (50 variables or more) with many random starts, particularly for non-smooth criteria such as simplimax. This is the default for all oblique rotations except target and PST.

"cayley" The Cayley transform retraction for orthogonal rotations (Wen and Yin, 2013). Instead of projecting back onto the orthogonal manifold via SVD, the Cayley transform produces an exactly orthogonal update in closed form:

$$T_{k+1} = \left(I + \frac{\alpha}{2} W\right)^{-1} \left(I - \frac{\alpha}{2} W\right) T_k$$

where $W = G_p T_k^T - T_k G_p^T$ is a skew-symmetric matrix derived from the projected gradient. The Cayley transform avoids the SVD decomposition at each iteration and can be faster for larger orthogonal problems. Available for orthogonal rotations only.

- algorithm = "bb" with fwindow = 10 is the default for all rotation criteria.
- algorithm = "cayley" is available for orthogonal rotations and can be specified explicitly by the user.

- `algorithm = "legacy"` is available for exact reproducibility of pre-2027 results.

Users can override the default by passing `algorithm` explicitly to any rotation function or directly to `GPForth` and `GPFoblq`.

The `fwindow` argument: The `fwindow` argument controls the width of the non-monotone line search window (Zhang and Hager, 2004). At each iteration, the Armijo sufficient decrease condition is evaluated against the maximum criterion value over the last `fwindow` iterations rather than just the immediately preceding value:

$$f(T_{k+1}) < \max_{j \in W_k} f(T_j) - 0.5\, s^2\, \alpha$$

where W_k is the window of the last `fwindow` iterates and s is the projected gradient norm.

- `fwindow = 1` (default for `"legacy"`) reduces to the standard monotone Armijo condition — the criterion must decrease at every iteration.
- `fwindow > 1` allows temporary increases in the criterion value, which can help the algorithm escape flat regions and shallow local minima. This is particularly beneficial in combination with `algorithm = "bb"`.
- `fwindow = 10` is the default for `algorithm = "bb"` and `algorithm = "cayley"`.

Benchmarking across matrix sizes and rotation criteria suggests that `fwindow = 10` with `algorithm = "bb"` provides the best balance of speed and solution quality for oblique rotations on larger models. For orthogonal rotations with `algorithm = "legacy"`, `fwindow = 1` is recommended as larger windows can slow convergence.

Legacy functions: The original implementations authored by Bernaards and Jennrich (2005) have been retained as `GPForth.legacy` and `GPFoblq.legacy`. These functions are kept purely for historical reference and backward compatibility for reproducibility. They are not exported into the package namespace, meaning they must be explicitly invoked using the triple-colon operator:

```
GPArotation:::GPForth.legacy(A, method = "varimax")
GPArotation:::GPFoblq.legacy(A, method = "quartimin")
```

The results generated by these legacy functions should be numerically identical to those produced by the current implementations when `algorithm = "legacy"` and `fwindow = 1` are used.

Value

A `GPArotation` object which is a list with elements:

`loadings`	The rotated loadings matrix, one column per factor. If random starts were requested, this is the solution with the lowest criterion value.
`Th`	The rotation matrix, satisfying `loadings %*% t(Th) = A` for orthogonal rotation and `loadings = A %*% solve(t(Th))` for oblique rotation.
`Table`	A matrix recording the iteration history: iteration number, criterion value, log10 of the gradient norm, and step size (alpha).
`method`	A string indicating the rotation criterion.
`orthogonal`	A logical indicating if the rotation is orthogonal.
`convergence`	A logical indicating if convergence was obtained.

`Phi`	`t(Th) %*% Th`, the covariance matrix of the rotated factors. Omitted (`NULL`) for orthogonal rotations.
`Gq`	The gradient of the criterion at the rotated loadings.
`randStartChar`	A named vector summarising random start results: `randomStarts`, `Converged`, `atMinimum`, `localMins`. Only present when `randomStarts > 1`.

Author(s)

Coen A. Bernaards and Robert I. Jennrich with some R modifications by Paul Gilbert

References

Barzilai, J. and Borwein, J.M. (1988) Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, **8**, 141–148. doi:10.1093/imanum/8.1.141

Bernaards, C.A. and Jennrich, R.I. (2005) Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis. *Educational and Psychological Measurement*, **65**, 676–696. doi:10.1177/0013164404272507

Jennrich, R.I. (2001) A simple general procedure for orthogonal rotation. *Psychometrika*, **66**, 289–306. doi:10.1007/BF02294840

Jennrich, R.I. (2002) A simple general method for oblique rotation. *Psychometrika*, **67**, 7–19. doi:10.1007/BF02294706

Mansolf, M. and Reise, S.P. (2016) Exploratory Bifactor Analysis: The Schmid-Leiman Orthogonalization and Jennrich-Bentler Analytic Rotations. *Multivariate Behavioral Research*, **51**(5), 698–717. doi:10.1080/00273171.2016.1215898

Nguyen, H.V. and Waller, N.G. (2023) Local minima and factor rotations in exploratory factor analysis. *Psychological Methods*, **28**(5), 1122–1141. doi:10.1037/met0000467

Wen, Z. and Yin, W. (2013) A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, **142**, 397–434. doi:10.1007/s1010701205841

Zhang, H. and Hager, W.W. (2004) A nonmonotone line search technique and its application to unconstrained optimization. *SIAM Journal on Optimization*, **14**(4), 1043–1056. doi:10.1137/S1052623403428208

See Also

`rotations`, `Random.Start`, `factanal`, `Harman8`, `box26`, `CCAI`, `NetherlandsTV`

Examples

```
# --- Basic rotation calls ---
data(Harman, package = "GPArotation")          # 8 physical variables
quartimax(Harman8)                             # direct rotation call
GPFRSorth(Harman8, method = "quartimax")       # equivalent via wrapper
GPFRSoblq(Harman8, method = "quartimin", normalize = TRUE)
loadings(quartimin(Harman8, normalize = TRUE))  # extract loadings directly

# --- Passing criterion arguments via methodArgs ---
# Crawford-Ferguson family: kappa selects the criterion.
```

```
# For box26: p = 26 variables, m = 3 factors.
# Equamax:  kappa = m / (2 * p) = 3 / 52
# Parsimax: kappa = (m - 1) / (p + m - 2) = 2 / 27
data(Thurstone, package = "GPArotation") # 26 variable box problem
GPFRSoblq(box26, method = "cf", methodArgs = list(kappa = 3/52))  # Equamax
GPFRSoblq(box26, method = "cf", methodArgs = list(kappa = 2/27))  # Parsimax

# --- Two-step vs single-step factanal for oblique rotation ---
#
# The recommended approach for oblique rotation is the two-step procedure:
# (1) obtain unrotated loadings from factanal, then
# (2) rotate separately using GPArotation.
# This gives full control over the rotation, including random starts.
#
# Prior to R 4.5.1, the single-step approach (rotation inside factanal)
# had a bug in factor reordering after oblique rotation.
# This was fixed by the R core team in R 4.5.1.

data("WansbeekMeijer", package = "GPArotation")

# Step 1: unrotated 3-factor solution
fa.unrotated <- factanal(factors = 3, covmat = NetherlandsTV,
                         normalize = TRUE, rotation = "none")

# Step 2: oblique Crawford-Ferguson rotation with kappa = 0.3
# (non-standard kappa, not corresponding to any named special case)
set.seed(44)
fa.cf <- cfQ(fa.unrotated, kappa = 0.3, normalize = TRUE, randomStarts = 100)
fa.cf

# Single-step via factanal - correct in R >= 4.5.1
if (getRversion() >= "4.5.1") {
  set.seed(44)
  fa.factanal <- factanal(factors = 3, covmat = NetherlandsTV, rotation = "cfQ",
    control = list(rotate = list(normalize = TRUE, kappa = 0.3, randomStarts = 100)))
  # The two approaches should agree after sorting
  fa.sorted <- print(fa.cf, sortLoadings = TRUE)
  cat("Maximum difference in loadings between two-step and single-step:\n")
  print(max(abs(abs(fa.sorted$loadings) - abs(fa.factanal$loadings))))
} else {
  cat("Single-step factanal oblique rotation requires R >= 4.5.1.\n")
  cat("Use the two-step procedure above for correct results.\n")
}

# --- Displaying rotation output ---
origdigits <- options("digits")
data("CCAI", package = "GPArotation")
fa.unrotated <- factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "none")
res <- oblimin(fa.unrotated, gam = -0.5, randomStarts = 20)
                                  # gam = -0.5: more orthogonal than quartimin
res                               # default print
print(res)                        # equivalent to above
print(res, Table = TRUE)          # include iteration table
```

```
print(res, rotateMat = TRUE)      # include rotating matrix
print(res, digits = 2)            # rounded to 2 decimal places
summary(res)                     # pattern and structure matrices for oblique rotation
summary(res, Structure = FALSE)   # pattern matrix only
options(digits = origdigits$digits)


# --- Random start diagnostics ---
# When randomStarts > 1, the output includes randStartChar which summarizes
# the random start results:
#   randomStarts : number of random starts attempted
#   Converged    : number of starts that converged
#   atMinimum    : number of starts at the same lowest minimum
#   localMins    : number of distinct local minima found
data(Thurstone, package = "GPArotation")
res <- GPFRSoblq(box26, method = "geomin", normalize = TRUE, randomStarts = 50)
res$randStartChar

# --- Factor ordering ---
# Raw GPArotation output is unsorted — factors may appear in any order
# depending on the starting matrix. Use print() to obtain sorted loadings.
# Once sorted, repeated calls to print() are stable.
set.seed(334)
xusl <- quartimin(Harman8, normalize = TRUE, randomStarts = 100)

loadings(xusl)                                      # unsorted raw output
max(abs(print(xusl)$loadings - xusl$loadings)) == 0  # FALSE: print() reorders
xsl <- print(xusl)                                   # capture sorted result
max(abs(print(xsl)$loadings - xsl$loadings)) == 0    # TRUE: already sorted

# --- Normalization ---
# Kaiser normalization
data("CCAI", package = "GPArotation")
fa.unrotated <- factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "none")
oblimin(fa.unrotated, normalize = TRUE, randomStarts = 100)
  data("CCAI", package = "GPArotation")
res.u <- factanal(covmat = CCAI_R, factors = 4, rotation = "none",
                  n.obs = 461)

# TandemI --- non-smooth orthogonal, BB and Cayley both faster
res.t1l <- tandemI(res.u, algorithm = "legacy", fwindow =  1, maxit = 10000)
res.t1b <- tandemI(res.u, algorithm = "bb",     fwindow = 10)
res.t1c <- tandemI(res.u, algorithm = "cayley", fwindow = 10)
cat("TandemI   legacy:", nrow(res.t1l$Table) - 1, "iterations\n")
cat("TandemI   bb:    ", nrow(res.t1b$Table) - 1, "iterations",
    " max diff:", format(max(abs(res.t1l$loadings - res.t1b$loadings)),
                         scientific = TRUE, digits = 2), "\n")
cat("TandemI   cayley:", nrow(res.t1c$Table) - 1, "iterations",
    " max diff:", format(max(abs(res.t1l$loadings - res.t1c$loadings)),
                         scientific = TRUE, digits = 2), "\n")

# Entropy --- orthogonal, BB and Cayley both faster
res.t2l <- entropy(res.u, algorithm = "legacy", fwindow =  1, maxit = 10000)
```

```
res.t2b <- entropy(res.u, algorithm = "bb",     fwindow = 10)
res.t2c <- entropy(res.u, algorithm = "cayley", fwindow = 10)
cat("Entropy   legacy:", nrow(res.t2l$Table) - 1, "iterations\n")
cat("Entropy   bb:    ", nrow(res.t2b$Table) - 1, "iterations",
    " max diff:", format(max(abs(res.t2l$loadings - res.t2b$loadings)),
                         scientific = TRUE, digits = 2), "\n")
cat("Entropy   cayley:", nrow(res.t2c$Table) - 1, "iterations",
    " max diff:", format(max(abs(res.t2l$loadings - res.t2c$loadings)),
                         scientific = TRUE, digits = 2), "\n")

# Oblimin --- smooth oblique, BB faster, same solution
res.ol <- oblimin(res.u, algorithm = "legacy", fwindow =  1, maxit = 10000)
res.ob <- oblimin(res.u, algorithm = "bb",     fwindow = 10)
cat("Oblimin   legacy:", nrow(res.ol$Table) - 1, "iterations\n")
cat("Oblimin   bb:    ", nrow(res.ob$Table) - 1, "iterations",
    " max diff:", format(max(abs(res.ol$loadings - res.ob$loadings)),
                         scientific = TRUE, digits = 2), "\n")

# Simplimax --- non-smooth oblique with many local minima.
# BB finds a substantially better solution in fewer iterations.
# Difference in loadings reflects different local minima, not noise.
res.sl <- simplimax(res.u, algorithm = "legacy", fwindow =  1, maxit = 10000)
res.sb <- simplimax(res.u, algorithm = "bb",     fwindow = 10)
cat("Simplimax legacy:", nrow(res.sl$Table) - 1, "iterations",
    " f =", round(res.sl$Table[nrow(res.sl$Table), 2], 6), "\n")
cat("Simplimax bb:    ", nrow(res.sb$Table) - 1, "iterations",
    " f =", round(res.sb$Table[nrow(res.sb$Table), 2], 6), "\n")
cat("BB found a lower criterion value --- better solution quality.\n")
```

GriffithMulaik *Griffith and Mulaik Interpersonal Personality Traits*

Description

Correlation matrix for 24 bipolar seven-point trait-adjective scales representing six hypothetical factors from the interpersonal domain of personality. Data from an unpublished senior thesis conducted under the supervision of Stanley Mulaik at the School of Psychology, Georgia Institute of Technology, 1998.

Usage

```
data(GriffithMulaik, package = "GPArotation")
```

Format

A 24×24 numeric correlation matrix with `n.obs` = 523. Row and column names are abbreviated variable labels for 24 interpersonal trait scales: ALLOW, OUTGO, NONTALK, PERMISS, FAIR, UNFRIEND, FORCELSS, RESTRAIN, NONDOMIN, UNLOVING, NONCNFRM, COMPLINT, IMPARTL, LEADER, UNKIND, UNCNVENT, DIRECTNG, PROHIBTV, UNGREGAR, JUST, NEUTRAL, SOCIABLE, UNAFFECT, REBELLOS.

Details

Participants were 523 Georgia Tech introductory psychology students who each rated three stereotypical persons selected at random from a large list of stereotypes on 52 bipolar seven-point trait-adjective scales. The 24 variables analyzed here represent six hypothetical factors from the interpersonal domain of personality: four synonym trait items per factor.

The study was intended to test Mulaik's theory of personality trait factors. Six simple structure factors were obtained as expected, providing tentative support for the theory.

The correlation matrix is reproduced as Table 8.5 in Mulaik (2018). The data are used in that chapter to illustrate maximum likelihood factor analysis with a reduced correlation matrix scree plot.

The six-factor oblimin solution obtained by `GPArotation` closely replicates the Direct Oblimin solution reported in Mulaik (2018) Table 8.6, with minor differences attributable to factor ordering and sign conventions.

References

Griffith, D. and Mulaik, S.A. (1998). *Personality trait factors from the interpersonal domain.* Poster presented at the Society for Multivariate Experimental Psychology (SMEP).

Mulaik, S.A. (2018). Fundamentals of common factor analysis. In P. Irwing, T. Booth, and D.J. Hughes (Eds.), *The Wiley Handbook of Psychometric Testing* (pp. 211–252). Wiley.

See Also

`rotations`, `Harman`, `Thurstone`, `CCAI`

Examples

```
data(GriffithMulaik, package = "GPArotation")

## Scree plot comparing raw versus reduced correlation matrix
## (Mulaik, 2018, recommends reduced matrix for ML factor analysis)
fa.un <- factanal(factors = 6, covmat = GriffithMulaik,
                  n.obs = 523, rotation = "none")
res.un <- quartimax(fa.un)

## Six factor oblimin rotation
res.ob <- oblimin(fa.un, randomStarts = 100)
summary(res.ob)

## Compare simple structure across rotation methods
res.vm <- Varimax(fa.un)
res.gm <- geominQ(fa.un, randomStarts = 100)

cat(sprintf("%-12s  AUC = %.3f  Hyperplane = %.1f pct\n",
            "Varimax",  GPArotation:::calc_AUC(res.vm)$AUC_mean,
            GPArotation:::calc_hyperplane(res.vm)$HP_pct))
cat(sprintf("%-12s  AUC = %.3f  Hyperplane = %.1f pct\n",
            "Oblimin",  GPArotation:::calc_AUC(res.ob)$AUC_mean,
            GPArotation:::calc_hyperplane(res.ob)$HP_pct))
cat(sprintf("%-12s  AUC = %.3f  Hyperplane = %.1f pct\n",
```

```
            "GeominQ",  GPArotation:::calc_AUC(res.gm)$AUC_mean,
            GPArotation:::calc_hyperplane(res.gm)$HP_pct))


## --- Theory-guided rotation via partially specified target ---
## Mulaik (2018) Table 8.7 specifies a CFA hypothesis matrix:
## * = free parameter (NA), o = constrained to zero (0)
GM_target <- matrix(c(
## F1  F2  F3  F4  F5  F6
  NA,  0,  0,  0, NA,  0,  ## ALLOW
  NA, NA,  0,  0, NA,  0,  ## OUTGO
   0, NA,  0,  0, NA,  0,  ## NONTALK
  NA,  0,  0,  0,  0,  0,  ## PERMISS
   0,  0, NA, NA,  0,  0,  ## FAIR
  NA, NA, NA, NA,  0,  0,  ## UNFRIEND
  NA, NA,  0, NA, NA,  0,  ## FORCELSS
  NA, NA,  0,  0, NA,  0,  ## RESTRAIN
  NA, NA,  0, NA, NA,  0,  ## NONDOMIN
   0,  0,  0, NA,  0,  0,  ## UNLOVING
   0,  0,  0,  0,  0, NA,  ## NONCNFRM
  NA,  0,  0,  0,  0, NA,  ## COMPLINT
   0, NA, NA,  0, NA,  0,  ## IMPARTL
   0,  0,  0,  0, NA,  0,  ## LEADER
   0, NA, NA, NA,  0,  0,  ## UNKIND
   0,  0,  0,  0,  0, NA,  ## UNCNVENT
   0,  0,  0, NA, NA,  0,  ## DIRECTNG
  NA,  0,  0,  0, NA, NA,  ## PROHIBTV
   0, NA,  0,  0,  0,  0,  ## UNGREGAR
   0,  0, NA, NA,  0,  0,  ## JUST
  NA, NA, NA, NA, NA,  0,  ## NEUTRAL
  NA, NA,  0, NA, NA, NA,  ## SOCIABLE
   0, NA,  0, NA,  0,  0,  ## UNAFFECT
   0, NA,  0, NA,  0, NA   ## REBELLOS
), nrow = 24, ncol = 6, byrow = TRUE)
rownames(GM_target) <- rownames(GriffithMulaik)
colnames(GM_target) <- paste0("F", 1:6)

## W: 1 = constrained to zero, 0 = free
GM_target_pst <- ifelse(is.na(GM_target), 0, GM_target)
GM_W          <- ifelse(is.na(GM_target), 0, 1)

## Oblique PST closely replicates Mulaik (2018) Table 8.8 CFA solution
res.pstQ <- pstQ(fa.un, Target = GM_target_pst, W = GM_W)
print(res.pstQ)

## Compare data-driven vs theory-guided rotation
cat(sprintf("%-10s  AUC = %.3f  Hyperplane = %.1f pct\n",
            "PST-Q",   GPArotation:::calc_AUC(res.pstQ)$AUC_mean,
            GPArotation:::calc_hyperplane(res.pstQ)$HP_pct))
cat(sprintf("%-10s  AUC = %.3f  Hyperplane = %.1f pct\n",
            "Oblimin", GPArotation:::calc_AUC(res.ob)$AUC_mean,
            GPArotation:::calc_hyperplane(res.ob)$HP_pct))
```

Harman *Harman's Eight Physical Variables*

Description

Unrotated factor loading matrix extracted by the centroid method for Harman's 8 physical variables, a classic dataset in factor analysis.

Usage

```
data(Harman)
```

Format

`Harman8`: a 8×2 numeric matrix of unrotated centroid factor loadings. Row names are variable names; column names are `CF1` and `CF2`.

Details

`Harman8` is a 8×2 matrix of unrotated factor loadings extracted by the centroid method (Table 14.3 in Harman, 1976) from the correlation matrix of 8 physical measurements taken on 305 girls (Table 2.3 in Harman, 1976). The centroid method was a standard extraction procedure before iterative methods became computationally feasible. The two columns represent unrotated centroid factors (`CF1` and `CF2`). Row names are the 8 variable names.

The covariance matrix for these 8 physical variables is available in base R (see `Harman23.cor`).

Source

Harman, H.H. (1976). *Modern Factor Analysis* (3rd ed.). University of Chicago Press.

See Also

`Harman23.cor`, `GPForth`, `rotations`, `Thurstone`, `WansbeekMeijer`, `CCAI`, `GriffithMulaik`

Examples

```
data(Harman, package = "GPArotation")

# Centroid unrotated loadings from Harman (1976) Table 14.3
print(Harman8)

# Rotate the centroid solution
quartimax(Harman8)
oblimin(Harman8, randomStarts = 100)
```

lp *L^p Rotation*

Description

Performs L^p rotation to obtain sparse factor loadings.

Usage

```
GPForth.lp(A, Tmat = diag(ncol(A)), p = 1, normalize = FALSE, eps = 1e-05,
           maxit = 10000, gpaiter = 5)
GPFoblq.lp(A, Tmat = diag(ncol(A)), p = 1, normalize = FALSE, eps = 1e-05,
           maxit = 10000, gpaiter = 5)
```

Arguments

A	Initial factor loadings matrix to be rotated.
Tmat	Initial rotation matrix.
p	Component-wise L^p where $0 < p \leq 1$.
normalize	Not recommended for L^p rotation.
eps	Convergence is assumed when the norm of the gradient is smaller than eps.
maxit	Maximum number of iterations allowed in the main loop.
gpaiter	Maximum iterations for the inner GPA rotation loop. The goal is to decrease the objective value, not fully optimize the inner loop. Warnings may appear but can be ignored if the main loop converges.

Details

These functions optimize an L^p rotation objective where `0 < p <= 1`. A smaller value of p promotes greater sparsity in the loading matrix but increases computational difficulty. For guidance on choosing p, see the Concluding Remarks in Liu et al. (2023).

The user-facing wrapper functions lpT and lpQ provide random start functionality on top of GPForth.lp and GPFoblq.lp respectively, analogous to how GPFRSorth and GPFRSoblq wrap GPForth and GPFoblq for standard rotation criteria. For most users lpT and lpQ are the recommended entry points.

Since the L^p objective is nonsmooth, a different optimization method is required compared to smooth rotation criteria. GPForth.lp and GPFoblq.lp replace GPForth and GPFoblq for orthogonal and oblique L^p rotations, respectively.

The optimization uses an iterative reweighted least squares (IRLS) approach. The nonsmooth objective is approximated by a smooth weighted least squares function in the outer loop, which is then optimized using GPA in the inner loop (gpaiter controls the maximum inner iterations).

Normalization is not recommended for L^p rotation and may produce unexpected results.

Value

A `GPArotation` object, which is a list containing:

`loadings`	Rotated loadings matrix, with one column per factor.
`Th`	Rotation matrix, satisfying `loadings %*% t(Th) = A`.
`Table`	Data frame recording iteration details: iteration count, objective value, and elapsed time.
`method`	String indicating the rotation objective function.
`orthogonal`	Logical indicating whether the rotation is orthogonal.
`convergence`	Logical indicating whether convergence was achieved. Convergence is assessed element-wise using `eps` as tolerance.
`Phi`	Covariance matrix of rotated factors, `t(Th) %*% Th`.

Author(s)

Xinyi Liu, with minor modifications for GPArotation by C. Bernaards.

References

Liu, X., Wallin, G., Chen, Y., and Moustaki, I. (2023). Rotation to sparse loadings using L^p losses and related inference problems. *Psychometrika*, **88**(2), 527–553. doi:10.1007/s1133602309911y

See Also

`lpT`, `lpQ`, `vgQ.lp.wls`

Examples

```
data("WansbeekMeijer", package = "GPArotation")
fa.unrotated <- factanal(factors = 2, covmat = NetherlandsTV, rotation = "none")
options(warn = -1)

# Orthogonal rotation - single start
fa.lpT1 <- GPForth.lp(loadings(fa.unrotated), p = 1)

# Orthogonal rotation - 10 random starts
fa.lpT <- lpT(loadings(fa.unrotated), Tmat = Random.Start(2), p = 1,
              randomStarts = 10)
print(fa.lpT, digits = 5, sortLoadings = FALSE, Table = TRUE, rotateMat = TRUE)

# Oblique rotation - single start
fa.lpQ1 <- GPFoblq.lp(loadings(fa.unrotated), p = 1)

# Oblique rotation - 10 random starts
fa.lpQ <- lpQ(loadings(fa.unrotated), p = 1, randomStarts = 10)
summary(fa.lpQ, Structure = TRUE)

# Compare Lp (p=1), Lp (p=0.5), and Geomin oblique rotations
set.seed(1020)
```

```
fa.lpQ1   <- lpQ(loadings(fa.unrotated), p = 1,   randomStarts = 10)
fa.lpQ0.5 <- lpQ(loadings(fa.unrotated), p = 0.5, randomStarts = 10)
fa.geo    <- geominQ(loadings(fa.unrotated),       randomStarts = 10)

# With factor ordering using internal sortGPALoadings helper
res <- round(cbind(GPArotation:::.sortGPALoadings(fa.lpQ1)$loadings,
                   GPArotation:::.sortGPALoadings(fa.lpQ0.5)$loadings,
                   GPArotation:::.sortGPALoadings(fa.geo)$loadings), 3)
print(c("oblique --  Lp p=1        Lp p=0.5       Geomin"))
print(res)

# Without factor ordering
res <- round(cbind(fa.lpQ1$loadings, fa.lpQ0.5$loadings, fa.geo$loadings), 3)
print(c("oblique --  Lp p=1        Lp p=0.5       Geomin"))
print(res)

options(warn = 0)
```

`plot.GPArotation` *Plot Method for GPArotation Objects*

Description

Produces visual summaries of rotated factor matrices, including an APA-style salient table-style, a heatmap, a tripartite pairs plot of factor geometry, a macro-level profile bar chart, a target rotation schema, a rotation trajectory view, or a set of diagnostics.

Usage

```
  ## S3 method for class 'GPArotation'
plot(x, type = c("salient", "pairs", "profile", "vector",
     "target", "heatmap", "trajectory", "diagnostics", "residuals"),
     hide_hyperplane = FALSE, salient = 0.40, cutoff = .1, R = NULL,
     factors = c(1,2), items = NULL, main = NULL, ...)
```

Arguments

`x`	An object of class `"GPArotation"`.
`type`	A character string indicating the type of plot. Must be one of `"salient"`, `"pairs"`, or `"profile"`, `"vector"`, `"target"`, `"heatmap"`, `"trajectory"`, `"diagnostics"`, `residuals`.
`hide_hyperplane`	Boolean. Hide values below cutoff. Default is FALSE.
`salient`	Numeric. Absolute values equal to or above this are highlighted as salient. Default is 0.40.

cutoff	Numeric. Absolute values strictly below this are considered noise/hyperplane. Default is 0.1.
R	Correlation matrix. If not provided taken from GPArotation object. Default is NULL.
factors	Two factors used in trajectory plot. Default = c(1,2).
items	items plotted in the trajectory plot. All items are plotted by default. Default = NULL.
main	main title where applicable. Default = NULL.
...	Additional graphical parameters passed to base plotting functions.

Value

Invisibly returns the original x object.

Author(s)

Your Name

Examples

```
  ## Not run:
    # Assuming 'res' is a fitted GPArotation object:
    plot(res, type = "salient")
    plot(res, type = "pairs")
    plot(res, type = "profile")

## End(Not run)
```

---

plot2fOrthComparison *Algorithm Comparison for Two-Factor Orthogonal Rotation*

Description

Produces a 2×3 panel figure comparing the "legacy", "bb", and "cayley" optimization algorithms on a two-factor orthogonal rotation. The top row shows the criterion landscape and the algorithm trajectory for each algorithm; the bottom row shows the corresponding factor migration paths. An optional endgame inset zooms into the convergence region near the global minimum.

Usage

```
plot2fOrthComparison(A, method = "quartimax", items = NULL,
                     nang = 6000, maxit = 500, magnify = FALSE,
                     inset_pos = "auto", randomStart = FALSE, ...)
```

Arguments

A	An unrotated loading matrix with exactly two columns, or a factanal object.
method	Character string naming the rotation criterion. Must be an orthogonal criterion available in **GPArotation**, for example "quartimax", "Varimax", or "geominT". Default is "quartimax".
items	Integer vector specifying which variables to include in the factor migration path plots. Default NULL includes all variables.
nang	Number of angles at which to evaluate the criterion landscape. Default 6000.
maxit	Maximum number of iterations for each algorithm. Default 500.
magnify	Logical. If TRUE, an endgame inset is added to each landscape panel, zooming into the convergence region near the global minimum. Default FALSE.
inset_pos	Placement of the inset. Options are "topleft", "bottomleft", and the default = "auto"
randomStart	Initiate with random start matrix. Default = FALSE
...	Additional arguments passed to the rotation function, for example normalize = TRUE.

Value

Invisibly returns a list with components legacy, bb, and cayley, each a "GPArotation" object from the corresponding algorithm run.

Note

This function requires a two-factor loading matrix. The criterion landscape visualization is only defined for two-factor orthogonal rotations, where all rotation matrices are parameterized by a single angle $\theta \in [0, 2\pi)$.

See Also

plot.GPArotation, GPForth, rotations

Examples

```
data("Harman", package = "GPArotation")
plot2fOrthComparison(Harman8, method = "quartimax", magnify = TRUE)

plot2fOrthComparison(Harman8, method = "tandemII", magnify = FALSE)
```

print.GPArotation *Print and Summary Methods for GPArotation Class Objects*

Description

Print and summary methods for objects returned by GPFRSorth, GPFRSoblq, GPForth, or GPFoblq. Both print.GPArotation and summary.GPArotation apply consistent factor sorting and sign correction via the internal helper .sortGPALoadings when sortLoadings = TRUE. Factors are ordered by descending variance explained and signs are adjusted so that the sum of loadings per factor is positive. This convention matches that used by factanal. For oblique rotations, summary.GPArotation displays both the pattern matrix (regression coefficients of items on factors, controlling for factor intercorrelations) and the structure matrix (loadings %*% Phi, correlations between items and factors) when Structure = TRUE. The two matrices coincide for orthogonal rotations where $\Phi = I$. Output includes contributions of factors via SS loadings (sum of squared loadings); see Harman (1976), sections 2.4 and 12.4. For orthogonal rotations, Proportion Var and Cumulative Var are also shown. For oblique rotations these are suppressed as they are not meaningful when factors are correlated. Simple structure quality is reported via two criterion-free per-factor measures: AUC (Liu and Moustaki, 2024) and FSI (Lorenzo-Seva, 2003). Higher values indicate cleaner simple structure. These measures are shown for both orthogonal and oblique rotations and can be used to compare simple structure quality across rotation methods.

Usage

```
    ## S3 method for class 'GPArotation'
print(x, digits = 3, sortLoadings = TRUE, cutoff = 0.1,
                   rotateMat = FALSE, Table = FALSE, ...)
    ## S3 method for class 'GPArotation'
summary(object, digits = 3, Structure = TRUE, ...)
    ## S3 method for class 'summary.GPArotation'
print(x, cutoff = 0.1, ...)
```

Arguments

x	a GPArotation or summary.GPArotation object to print.
object	a GPArotation object to summarize.
digits	precision of printed numbers.
sortLoadings	logical; if TRUE (default) factors are sorted by descending variance explained and factor signs are adjusted so that the sum of loadings per factor is positive. Adapted from factanal sorting conventions. Use sortLoadings = FALSE to display the raw unsorted solution, for example when the factor order is meaningful as in bifactor rotation.
cutoff	Numeric. Absolute values strictly below this are considered noise/hyperplane. Default is 0.1.
rotateMat	logical; if TRUE the rotation matrix is displayed.
Table	logical; if TRUE the iteration table is displayed.

`Structure`	logical; if `TRUE` (default) the structure matrix (`loadings %*% Phi`) is displayed for oblique rotations in `summary`.
`...`	further arguments passed to other methods.

Details

Factor sorting and sign correction are applied consistently in both `print` and `summary` via the internal function `.sortGPALoadings`, adapted from `factanal` sorting conventions (R Core Team). This ensures that the pattern matrix shown by `summary` is consistent with the loadings shown by `print`.

The `digits` argument controls the number of decimal places shown in the loadings, structure matrix, and Phi.

Display options such as `cutoff` are controlled at print time: `print(summary(x), cutoff = 0.4)`, consistent with standard R convention.

Accessing the rotated loading matrix: The rotated loading matrix can be accessed directly via `x\$loadings` for use in further computations. The formatted output with factor sorting, AUC, FSI, and SS loadings is produced by `print(x)` or simply by typing the object name at the console. The `loadings` function from the **stats** package can also be used but returns the unsorted matrix without simple structure measures.

Convergence and algorithm information: The print header reports convergence status, the algorithm used (`algorithm`), and the non-monotone line search window (`fwindow`). If convergence was not obtained, a diagnostic suggestion is provided based on the algorithm currently in use. See `GPForth` and `GPFoblq` for details on algorithm options.

For examples see `GPFRSorth` and the package vignettes: `vignette("GPA1guide", package = "GPArotation")`.

Value

`print.GPArotation` returns the sorted `GPArotation` object invisibly when `sortLoadings = TRUE`, or the unsorted object when `sortLoadings = FALSE`. `summary.GPArotation` returns a `summary.GPArotation` object with sorted loadings and, for oblique rotations, the structure matrix. `print.summary.GPArotation` returns the object invisibly.

References

Harman, H.H. (1976) *Modern Factor Analysis*, 3rd ed. University of Chicago Press.

Liu, X., Wallin, G., Chen, Y., and Moustaki, I. (2023). Rotation to sparse loadings using L^p losses and related inference problems. *Psychometrika*, **88**(2), 527–553. doi:10.1007/s1133602309911y

Lorenzo-Seva, U. (2003) A factor simplicity index. *Psychometrika*, **68**(1), 49–60. doi:10.1007/BF02296652

See Also

`GPFRSorth`, `GPForth`, `factanal`, `summary`

Examples

```
data(Harman, package = "GPArotation")
res <- oblimin(Harman8, normalize = TRUE, randomStarts = 100)

# Print sorted loadings (default)
print(res)

# Print unsorted loadings
print(res, sortLoadings = FALSE)

# Summary with pattern and structure matrices
summary(res, Structure = TRUE)

# Summary without structure matrix
summary(res, Structure = FALSE)

# Print with iteration table
print(res, Table = TRUE)
```

Random.Start *Random Starting Matrix for Factor Rotation*

Description

Generates a random orthogonal matrix for use as an initial rotation matrix (`Tmat`) in gradient projection rotation functions.

Usage

```
Random.Start(k = 2L)
```

Arguments

k	a positive integer specifying the dimension of the square orthogonal matrix to generate (default 2).

Details

The function generates a random orthogonal matrix using QR decomposition of a matrix of standard normal variates, with a sign correction applied to the diagonal of R to ensure uniform sampling from the Haar measure. This follows the approach of Stewart (1980) and Mezzadri (2007).

The naive approach of `qr.Q(qr(matrix(rnorm(k*k), k)))` does not guarantee uniform sampling from the Haar measure. The sign correction `Q %*% diag(sign(diag(R)))` is required to achieve this. This was updated in GPArotation 2024.2-1 following a suggestion by Yves Rosseel.

For oblique rotation a random starting transformation matrix can be generated by normalizing the columns of a random matrix: `X %*% diag(1/sqrt(diag(crossprod(X))))` where `X <- matrix(rnorm(k*k), k)`.

Value

A $k \times k$ orthogonal matrix drawn uniformly from the Haar measure on the orthogonal group O(k). Columns have unit length and are mutually orthogonal.

Author(s)

Coen A. Bernaards and Robert I. Jennrich with some R modifications by Paul Gilbert. Updated following a suggestion by Yves Rosseel.

References

Stewart, G.W. (1980). The efficient generation of random orthogonal matrices with an application to condition estimators. *SIAM Journal on Numerical Analysis*, **17**(3), 403–409. doi:10.1137/0717034

Mezzadri, F. (2007) How to generate random matrices from the classical compact groups. *Notices of the American Mathematical Society*, **54**(5), 592–604. arXiv:math-ph/0609050

See Also

GPFRSorth, GPFRSoblq, GPForth, GPFoblq, rotations

Examples

```
# Generate a 5 x 5 random orthogonal matrix
Random.Start(5)

# Generate a random orthogonal start matrix
Q <- Random.Start(4)

# Verify orthogonality: t(Q) %*% Q should equal the identity matrix
stopifnot(isTRUE(all.equal(t(Q) %*% Q, diag(4), tolerance = 1e-10)))

# Use as starting matrix for rotation
data("Thurstone", package = "GPArotation")
simplimax(box26, Tmat = Random.Start(3))
```

---

rotations *Rotations Functions Using Gradient Projection Algorithms*

Description

Optimize factor loading rotation objective.

Usage

```
oblimin(A, Tmat = NULL, gam=0, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
quartimin(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
targetT(A, Tmat = NULL, Target = NULL, normalize = FALSE, randomStarts = 0,
    algorithm = "bb", fwindow = 10, ...)
targetQ(A, Tmat = NULL, Target = NULL, normalize = FALSE, randomStarts = 0,
    algorithm = "bb", fwindow = 10, ...)
pstT(A, Tmat = NULL, W = NULL, Target = NULL, normalize = FALSE,
    randomStarts = 0, algorithm = "bb", fwindow = 10, ...)
pstQ(A, Tmat = NULL, W = NULL, Target = NULL, normalize = FALSE,
    randomStarts = 0, algorithm = "bb", fwindow = 10, ...)
oblimax(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
entropy(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
quartimax(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
Varimax(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
simplimax(A, Tmat = NULL, k = NULL, normalize=FALSE,
    randomStarts=0, algorithm = "bb", fwindow = 10,...)
bentlerT(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
bentlerQ(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
tandemI(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
tandemII(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
geominT(A, Tmat = NULL, delta=0.01, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
geominQ(A, Tmat = NULL, delta=0.01, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
bigeominT(A, Tmat = NULL, delta=0.01, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
bigeominQ(A, Tmat = NULL, delta=0.01, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
cfT(A, Tmat = NULL, kappa=0, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
cfQ(A, Tmat = NULL, kappa=0, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
equamax(A, Tmat = NULL, kappa=NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
parsimax(A, Tmat = NULL, kappa=NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
infomaxT(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
```

```
    algorithm = "bb", fwindow = 10, ...)
infomaxQ(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
mccammon(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
binormamin(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
varimin(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
bifactorT(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
bifactorQ(A, Tmat = NULL, normalize=FALSE, randomStarts=0,
    algorithm = "bb", fwindow = 10, ...)
lpT(A, Tmat=diag(ncol(A)), p=1, normalize=FALSE, eps=1e-05, maxit=2000,
        randomStarts=0, gpaiter=5)
lpQ(A, Tmat=diag(ncol(A)), p=1, normalize=FALSE, eps=1e-05, maxit=2000,
    randomStarts=0, gpaiter=5)
```

Arguments

A	an initial loadings matrix to be rotated.
Tmat	initial rotation matrix.
gam	Obliqueness parameter (γ in the mathematical definition). 0=Quartimin, .5=Bi-quartimin, 1=Covarimin.
Target	rotation target for objective calculation.
W	weighting of each element in target.
k	number of close to zero loadings.
delta	constant added to Λ^2 in the objective calculation.
kappa	see details.
normalize	parameter passed to optimization routine (GPForth or GPFoblq).
eps	convergence tolerance passed to GPForth or GPFoblq via .... Convergence is assumed when the norm of the gradient is smaller than eps. Default is 1e-5.
maxit	maximum number of iterations passed to GPForth or GPFoblq via .... Default is 2000.
randomStarts	parameter passed to optimization routine (GPFRSorth or GPFRSoblq).
algorithm	Line-search strategy used to calculate step size alpha.
fwindow	integer. Number of previous criterion values used in the non-monotone line search.
p	Component-wise L^p, where $0 < p =< 1$.
gpaiter	Maximum iterations for GPA rotation loop in L^p rotation.
...	additional arguments passed to GPForth or GPFoblq, including eps and maxit.

Details

These functions optimize a rotation objective. They can be used directly or the function name can be passed to factor analysis functions like `factanal`. Several of the function names end in T or Q, which indicates if they are orthogonal or oblique rotations (using `GPFRSorth` or `GPFRSoblq` respectively). The gradient projection algorithms are described in Bernaards and Jennrich (2005).

`oblimin`	oblique	oblimin family; `gam` controls obliqueness
`quartimin`	oblique	oblimin with `gam = 0`
`targetT`	orthogonal	rotation towards a target matrix
`targetQ`	oblique	rotation towards a target matrix
`pstT`	orthogonal	partially specified target rotation
`pstQ`	oblique	partially specified target rotation
`oblimax`	oblique	maximizes overall kurtosis of loadings
`entropy`	orthogonal	minimizes entropy of squared loadings
`quartimax`	orthogonal	maximizes variance of squared loadings within variables
`Varimax`	orthogonal	maximizes variance of squared loadings within factors
`simplimax`	oblique	minimizes the k smallest squared loadings
`bentlerT`	orthogonal	invariant pattern simplicity
`bentlerQ`	oblique	invariant pattern simplicity
`tandemI`	orthogonal	factors share high loadings on same variables
`tandemII`	orthogonal	factors do not share high loadings on same variables
`geominT`	orthogonal	minimizes geometric mean of squared loadings
`geominQ`	oblique	minimizes geometric mean of squared loadings
`bigeominT`	orthogonal	geomin with a general factor in column 1
`bigeominQ`	oblique	geomin with a general factor in column 1
`cfT`	orthogonal	Crawford-Ferguson family; `kappa` controls complexity
`cfQ`	oblique	Crawford-Ferguson family; `kappa` controls complexity
`equamax`	orthogonal	Crawford-Ferguson with `kappa = m/(2p)`
`parsimax`	orthogonal	Crawford-Ferguson with `kappa = (m-1)/(p+m-2)`
`infomaxT`	orthogonal	infomax information criterion
`infomaxQ`	oblique	infomax information criterion
`mccammon`	orthogonal	minimizes entropy ratio across factors
`binormamin`	oblique	minimizes minimizes cross-factor covariation of squared loadings
`varimin`	orthogonal	minimizes variance of squared loadings within factors
`bifactorT`	orthogonal	bifactor; general factor in column 1
`bifactorQ`	oblique	biquartimin; general factor in column 1
`lpT`	orthogonal	L^p sparsity rotation
`lpQ`	oblique	L^p sparsity rotation

The `Varimax` implementation in the list uses the gradient projection algorithm applied to `vgQ.varimax`. This implementation is different that the `varimax` rotation defined in the `stats` package. Additionally, `varimax` does Kaiser normalization by default whereas `GPArotation::Varimax` does not.

The argument `kappa` parameterizes the family for the Crawford-Ferguson method. If `m` is the number of factors and `p` is the number of indicators then `kappa` values having special names are 0 =Quartimax, $1/p$ =Varimax, $m/(2*p)$ =Equamax, $(m-1)/(p+m-2)$ =Parsimax, 1 =Factor parsimony.

Bifactor rotations, bifactorT and bifactorQ are called bifactor and biquartimin in Jennrich and Bentler (2011). For a comparison of exploratory bifactor analysis algorithms including those implemented here, see Garcia-Garzon, Abad and Garrido (2021).

The argument `p` is needed for L^p rotation. See Lp rotation for details on the rotation method.

`update` follows standard R convention — use it to modify arguments (e.g. `randomStarts`, `normalize`) not to change the rotation criterion. To re-rotate with a different criterion, use `GPFRSorth` or `GPFRSoblq` with `attr(object, "A_unrotated")`.

Value

A `GPArotation` object which is a list with elements:

`loadings`	The rotated loadings matrix, one column per factor. If random starts were requested, this is the solution with the lowest criterion value.
`Th`	The rotation matrix, satisfying `loadings %*% t(Th) = A` for orthogonal rotation and `loadings = A %*% solve(t(Th))` for oblique rotation.
`Table`	A matrix recording the iteration history: iteration number, criterion value, log10 of the gradient norm, and step size (alpha).
`method`	A string indicating the rotation criterion.
`orthogonal`	A logical indicating if the rotation is orthogonal.
`convergence`	A logical indicating if convergence was obtained.
`Phi`	`t(Th) %*% Th`, the covariance matrix of the rotated factors. Omitted (`NULL`) for orthogonal rotations.
`Gq`	The gradient of the criterion at the rotated loadings.
`randStartChar`	A named vector summarising random start results: `randomStarts`, `Converged`, `atMinimum`, `localMins`. Only present when `randomStarts > 1`.

Author(s)

Coen A. Bernaards and Robert I. Jennrich with some R modifications by Paul Gilbert.

References

Bernaards, C.A. and Jennrich, R.I. (2005) Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis. *Educational and Psychological Measurement*, **65**, 676–696. doi:10.1177/0013164404272507

Bi, Y. and Barchard, K.A. (2024). Purchasing choices that reduce climate change: An exploratory factor analysis. *Spectra Undergraduate Research Journal*, **3**(2), 8–14. doi:10.9741/27667227.1028

Fischer, R., & Fontaine, J. (2010). Methods for investigating structural equivalence. In D. Matsumoto & F. van de Vijver (Eds.), Cross-Cultural Research Methods in Psychology (pp. 179–215). Cambridge University Press. doi:10.1017/CBO9780511779381.010

Garcia-Garzon, E., Abad, F.J. and Garrido, L.E. (2021). On omega hierarchical estimation: A comparison of exploratory bi-factor analysis algorithms. *Multivariate Behavioral Research*, **56**(1), 101–119. doi:10.1080/00273171.2020.1736977

Jennrich, R.I. and Bentler, P.M. (2011). Exploratory bi-factor analysis. *Psychometrika*, **76**(4), 537–549. doi:10.1007/s1133601192184

For references to individual rotation criteria see `vignette("GPA1guide", package = "GPArotation")`.

See Also

factanal, GPFRSorth, GPFRSoblq, vgQ, Harman8, NetherlandsTV, CCAI, box26

Examples

```
set.seed (2026)
# For extended examples see the vignettes:
# vignette("GPA1guide",    package = "GPArotation")
# vignette("GPA2local",    package = "GPArotation")
# vignette("GPA3bifactor", package = "GPArotation")

# --- Accessing rotated loadings ---
data("Harman", package = "GPArotation") # 8 physical variables
qHarman <- quartimax(Harman8)
loadings(qHarman)                               # via extractor (recommended)
qHarman$loadings                                # via direct list access
all.equal(loadings(qHarman), qHarman$loadings) # identical

# --- Rotating factanal loadings ---
data("WansbeekMeijer", package = "GPArotation") # Netherlands TV viewership
fa.unrotated <- factanal(factors = 2, covmat = NetherlandsTV,
                         normalize = TRUE, rotation = "none")
quartimax(fa.unrotated, normalize = TRUE)
geominQ(fa.unrotated, normalize = TRUE, randomStarts = 100)

# --- Passing rotation to factanal ---
# CCAI:Climate-Friendly Purchasing Choices domain of the Climate Change Action Inventory
data("CCAI", package = "GPArotation")
factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "infomaxT")
factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "infomaxT",
         control = list(rotate = list(normalize = TRUE, eps = 1e-6)))

# --- Target rotation ---
# Orthogonal target rotation of two varimax rotated matrices
# towards each other. Data from Fischer and Fontaine (2010).
# See vignette("GPA1guide", package = "GPArotation") for further analyses.
trBritain <- matrix(c(.783, -.163, .811, .202, .724, .209, .850, .064,
                      -.031, .592, -.028, .723, .388, .434, .141, .808,
                      .215, .709), byrow = TRUE, ncol = 2)

trGermany <- matrix(c(.778, -.066, .875, .081, .751, .079, .739, .092,
                      .195, .574, -.030, .807, -.135, .717, .125, .738,
                      .060, .691), byrow = TRUE, ncol = 2)
trx <- targetT(trGermany, Target = trBritain)
print(trx$loadings - trBritain, cutoff = 0, digits = 3)  # difference from target

# Plot discrepancy from target --- steelblue = below target, firebrick = above
plot(trx, type = "target", Target = trBritain)


# --- Promax via targetQ, vs. stats::promax ---
# targetQ enforces the oblique normalization throughout the fit,
```

```
# rather than correcting for it afterward as stats::promax does;
# this reaches a lower value of the fitting criterion on the
# identical target.
data("Harman", package = "GPArotation")
xv     <- unclass(loadings(Varimax(Harman8)))
target <- xv * abs(xv)^(4 - 1)
res.pq <- targetQ(Harman8, Target = target)
res.pm <- promax(Harman8)
sum((unclass(res.pq$loadings) - target)^2)          # targetQ
sum((unclass(res.pm$loadings) - target)^2)           # stats::promax


# --- Partially specified target rotation ---
# See vignette("GPA1guide", package = "GPArotation") for full context.
# Unrotated loadings matrix A and partially specified target SPA.
# NA entries in SPA are unspecified --- rotation is free there.
# Numeric entries are the target values the rotation aims towards.
A <- matrix(c(.664, .688, .492, .837, .705, .82, .661, .457, .765, .322,
              .248, .304, -0.291, -0.314, -0.377, .397, .294, .428,
              -0.075, .192, .224, .037, .155, -.104, .077, -.488, .009), ncol = 3)
SPA <- matrix(c(rep(NA, 6), .7, .0, .7, rep(0, 3), rep(NA, 7),
                0, 0, NA, 0, rep(NA, 4)), ncol = 3)
comparison <- cbind(round(A, 3), rep(NA, nrow(A)), SPA)
colnames(comparison) <- c("A.F1", "A.F2", "A.F3", "|", "T.F1", "T.F2", "T.F3")
cat("Unrotated loadings (A) and partially specified target (SPA):\n")
print(comparison, na.print = "NA")

res.t <- targetT(A, Target = SPA)
print(res.t)

# Discrepancy from target --- specified elements only
plot(res.t, type = "target", Target = SPA)

# --- Random starts and Update calls ---
# CCAI Climate-Friendly Purchasing Choices domain, 14 items, 3 oblique factors.
# High factor intercorrelations make oblimin the natural choice.
# Note: factanal uses MLE extraction; results differ somewhat from
# PCA-based extraction used in Bi and Barchard (2024).
data("CCAI", package = "GPArotation")
fa.unrotated <- factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "none")
# Single random start via Tmat
res.o <- oblimin(fa.unrotated, Tmat = Random.Start(3))
res.o <- oblimin(fa.unrotated, randomStarts = 1)  # equivalent using randomStarts = 1
res.o <- oblimin(fa.unrotated, randomStarts = 100) # multiple random starts
# Update arguments without re-specifying the full call
res.o2 <- update(res.o, randomStarts = 250)  # randomStarts = 250
res.o3 <- update(res.o, gam = -0.5) # gam = -0.5, randomStarts = 100
res.o4 <- update(res.o, normalize = TRUE) # normalize = TRUE, randomStarts = 100

# To change rotation criterion, use A_unrotated directly
A <- attr(res.o, "A_unrotated")
res.qt <- GPFRSoblq(A, method = "quartimin")
```

```
# Directly via factanal call
factanal(factors = 3, covmat = CCAI_R, n.obs = 461, rotation = "oblimin",
    control = list(rotate = list(normalize = TRUE, gam = -0.1, randomStarts = 10)))

# --- Assessing local minima ---
# For detailed investigation of local minima across all random starts
# see vignette("GPA2local", package = "GPArotation").
data(Thurstone, package = "GPArotation")
infomaxQ(box26, normalize = TRUE, randomStarts = 150)
geominQ(box26,  normalize = TRUE, randomStarts = 150)
```

`Thurstone` *Thurstone Box Data*

Description

Factor loading matrices derived from Thurstone's box data.

Usage

```
data(Thurstone)
```

Format

Two numeric matrices:

- `box26`: 26 x 3 matrix of unrotated factor loadings.
- `box20`: 20 x 3 matrix of unrotated factor loadings.

Details

Loading the data makes the following objects available:

`box26` is a 26×3 unrotated factor loading matrix for 26 variables measured on a set of boxes. It appears frequently in the rotation literature as a dataset for comparing rotation criteria, particularly for assessing local minima.

`box20` is a 20×3 unrotated factor loading matrix for a subset of 20 variables from the same box data. It is used in Jennrich (2002) to illustrate rotation criteria and local minima behavior.

Both matrices are suitable as input to `GPForth`, `GPFoblq`, and all rotation wrapper functions in `GPArotation`. For a richer collection of factor analysis datasets, see the `psych` package.

Source

Thurstone, L.L. (1947). *Multiple Factor Analysis*. University of Chicago Press.

References

Jennrich, R.I. (2002). A simple general method for oblique rotation. *Psychometrika*, **67**(1), 7–19. doi:10.1007/BF02294706

See Also

GPForth, rotations, Harman, CCAI, WansbeekMeijer, GriffithMulaik

---

vgQ *Rotation Criteria: Value and Gradient Functions*

Description

Rotation criterion functions that compute the value and gradient of the rotation objective Q. Not exported from the package NAMESPACE.

Usage

```
vgQ.oblimin(L, gam=0)
vgQ.quartimin(L)
vgQ.target(L, Target=NULL)
vgQ.pst(L, W=NULL, Target=NULL)
vgQ.oblimax(L)
vgQ.entropy(L)
vgQ.quartimax(L)
vgQ.varimax(L)
vgQ.simplimax(L, k=nrow(L) * (ncol(L) - 1))
vgQ.bentler(L)
vgQ.tandemI(L)
vgQ.tandemII(L)
vgQ.geomin(L, delta=0.01)
vgQ.bigeomin(L, delta=0.01)
vgQ.cf(L, kappa=0)
vgQ.infomax(L)
vgQ.mccammon(L)
vgQ.varimin(L)
vgQ.bifactor(L)
vgQ.lp.wls(L, W)
```

Arguments

L	a factor loading matrix.
gam	0 = Quartimin, 0.5 = Biquartimin, 1 = Covarimin.
Target	rotation target matrix for objective calculation.
W	weight matrix; weighting of each element in target rotation (vgQ.pst) or in iterative reweighted least squares (vgQ.lp.wls).
k	number of near-zero loadings to target.
delta	small constant added to Λ^2 for numerical stability in the objective calculation.
kappa	complexity weight for the Crawford-Ferguson family; see cfT for details.

Details

The vgQ.* functions are called internally by the gradient projection optimization routines and would typically not be used directly. They can be inspected using :::, for example: GPArotation:::vgQ.oblimin. The gradient projection algorithms are described in Bernaards and Jennrich (2005).

The T or Q suffix on rotation function names should be omitted for the corresponding vgQ.* function. For example, GPArotation:::vgQ.target is the criterion used by both targetT and targetQ.

vgQ.oblimin	orthogonal or oblique	oblimin family
vgQ.quartimin	oblique	oblimin with gam = 0
vgQ.target	orthogonal or oblique	target rotation
vgQ.pst	orthogonal or oblique	partially specified target rotation
vgQ.oblimax	oblique	maximizes overall kurtosis of loadings
vgQ.entropy	orthogonal	minimum entropy
vgQ.quartimax	orthogonal	maximizes variance of squared loadings within variables
vgQ.varimax	orthogonal	maximizes variance of squared loadings within factors
vgQ.simplimax	oblique	minimizes the k smallest squared loadings
vgQ.bentler	orthogonal or oblique	Bentler invariant pattern simplicity
vgQ.tandemI	orthogonal	Tandem principle I criterion
vgQ.tandemII	orthogonal	Tandem principle II criterion
vgQ.geomin	orthogonal or oblique	minimizes geometric mean of squared loadings
vgQ.bigeomin	orthogonal or oblique	geomin with a general factor in column 1
vgQ.cf	orthogonal or oblique	Crawford-Ferguson family
vgQ.infomax	orthogonal or oblique	infomax information criterion
vgQ.mccammon	orthogonal	McCammon minimum entropy ratio
vgQ.varimin	orthogonal	varimin criterion
vgQ.bifactor	orthogonal or oblique	bifactor/biquartimin rotation
vgQ.lp.wls	orthogonal or oblique	iterative reweighted least squares for L^p rotation

See rotations for details on rotation arguments.

New rotation criteria can be added by defining a function named vgQ.newmethod. The function takes L as its first argument, plus any additional arguments, and must return a list with elements f, Gq, and Method.

Gradient projection *without* derivatives can be performed using the GPArotateDF package; type vignette("GPArotateDF", package = "GPArotation") at the command line.

Value

A list with:

f	The value of the rotation criterion at L.
Gq	The gradient of the criterion at L.
Method	A string indicating the criterion name.

Author(s)

Coen A. Bernaards and Robert I. Jennrich with some R modifications by Paul Gilbert.

References

Bernaards, C.A. and Jennrich, R.I. (2005) Gradient Projection Algorithms and Software for Arbitrary Rotation Criteria in Factor Analysis. *Educational and Psychological Measurement*, **65**, 676–696. doi:10.1177/0013164404272507

See Also

rotations, GPFRSorth

Examples

```
GPArotation:::vgQ.oblimin
```

WansbeekMeijer *Netherlands Television Viewership Data*

Description

Correlation matrix for Netherlands television viewership, used as an example dataset for factor rotation. From Wansbeek and Meijer (2000), page 171.

Usage

```
data(WansbeekMeijer)
```

Format

NetherlandsTV is a list with components:

- $cov: a 7×7 numeric correlation matrix
- $n.obs: sample size (2154)

Details

NetherlandsTV is a list with components $cov (a 7×7 correlation matrix for 7 television viewership variables measured in the Netherlands) and $n.obs (sample size, 2154). Use cov2cor(NetherlandsTV\$cov) to obtain the correlation matrix, or pass NetherlandsTV directly to factanal which handles the list structure automatically. It is used throughout the GPArotation documentation and vignettes as an example dataset for oblique rotation with 2 or 3 factors.

Source

Wansbeek, T. and Meijer, E. (2000). *Measurement Error and Latent Variables in Econometrics*. North-Holland.

See Also

GPForth, rotations, Thurstone, Harman, CCAI, GriffithMulaik

Examples

```
data(WansbeekMeijer, package = "GPArotation")

# Correlation matrix
round(cov2cor(NetherlandsTV$cov), 2)

# factanal picks up n.obs automatically from the list
factanal(factors = 2, covmat = NetherlandsTV, rotation = "none")

# Two-step oblique rotation
fa.unrotated <- factanal(factors = 3, covmat = NetherlandsTV,
                         rotation = "none")
oblimin(loadings(fa.unrotated), randomStarts = 100)
```

Index

∗ **datasets**
CCAI, 7
GriffithMulaik, 21
Harman, 24
Thurstone, 40
WansbeekMeijer, 43
∗ **hplot**
plot2fOrthComparison, 28
∗ **multivariate**
echelon, 10
eiv, 12
GPA, 14
lp, 25
print.GPArotation, 30
Random.Start, 32
rotations, 33
vgQ, 41
∗ **package**
00.GPArotation, 2
∗ **rotation**
echelon, 10
eiv, 12
GPA, 14
lp, 25
print.GPArotation, 30
Random.Start, 32
rotations, 33
vgQ, 41
.GPA_RS_engine *(*GPA*)*, 14
.sortGPALoadings *(*print.GPArotation*)*, 30
00.GPArotation, 2

bentlerQ, *3*
bentlerQ *(*rotations*)*, 33
bentlerT, *3*
bentlerT *(*rotations*)*, 33
bifactorQ, *3*
bifactorQ *(*rotations*)*, 33
bifactorT, *3*, *9*
bifactorT *(*rotations*)*, 33
bigeominQ, *3*
bigeominQ *(*rotations*)*, 33
bigeominT, *3*
bigeominT *(*rotations*)*, 33
binormamin, *3*
binormamin *(*rotations*)*, 33
box20, *4*
box20 *(*Thurstone*)*, 40
box26, *4*, *18*, *38*
box26 *(*Thurstone*)*, 40

calc_AUC, *5*
calc_fitstats, *5*
calc_FSI, *5*
calc_hyperplane, *5*
calc_simplicity, *5*
CCAI, *4*, 7, *18*, *22*, *24*, *38*, *41*, *43*
CCAI_pattern *(*CCAI*)*, 7
CCAI_Phi *(*CCAI*)*, 7
CCAI_R *(*CCAI*)*, 7
cfQ, *3*
cfQ *(*rotations*)*, 33
cfT, *3*, *41*
cfT *(*rotations*)*, 33

echelon, *4*, 10, *13*
eigen, *9*
eiv, *4*, *11*, 12
entropy, *2*
entropy *(*rotations*)*, 33
equamax, *3*
equamax *(*rotations*)*, 33

factanal, *9*, *10*, *12*, *14*, *18*, *30*, *31*, *38*, *43*

geominQ, *3*
geominQ *(*rotations*)*, 33
geominT, *3*
geominT *(*rotations*)*, 33
GPA, 14

GPArotation (00.GPArotation), 2
GPArotation-package (00.GPArotation), 2
GPArotation.Intro (00.GPArotation), 2
GPFoblq, 3, 11, 13, 31, 33
GPFoblq (GPA), 14
GPFoblq.lp, 4
GPFoblq.lp (lp), 25
GPForth, 3, 11, 13, 24, 29, 31, 33, 41, 43
GPForth (GPA), 14
GPForth.lp, 4
GPForth.lp (lp), 25
GPFRSoblq, 3, 6, 25, 33, 38
GPFRSoblq (GPA), 14
GPFRSorth, 3, 6, 25, 31, 33, 38, 43
GPFRSorth (GPA), 14
GriffithMulaik, 4, 9, 21, 24, 41, 43

Harman, 9, 22, 24, 41, 43
Harman23.cor, 24
Harman8, 4, 18, 38
Harman8 (Harman), 24

infomaxQ, 3
infomaxQ (rotations), 33
infomaxT, 3
infomaxT (rotations), 33

loadings, 31
lp, 25
Lp rotation, 37
Lp rotation (lp), 25
lpQ, 3, 25, 26
lpQ (rotations), 33
lpT, 3, 25, 26
lpT (rotations), 33

mccammon, 3
mccammon (rotations), 33

NetherlandsTV, 4, 18, 38
NetherlandsTV (WansbeekMeijer), 43
NormalizingWeight, 5, 15

oblimax, 2
oblimax (rotations), 33
oblimin, 2
oblimin (rotations), 33

parsimax, 3
parsimax (rotations), 33

plot.GPArotation, 4, 6, 27, 29
plot2fOrthComparison, 3, 28
print.GPArotation, 4, 30
print.summary.GPArotation
(print.GPArotation), 30
promax, 4
pstQ, 2
pstQ (rotations), 33
pstT, 2
pstT (rotations), 33

quartimax, 2
quartimax (rotations), 33
quartimin, 2
quartimin (rotations), 33

Random.Start, 4, 16, 18, 32
rotations, 6, 9, 11, 13, 15, 18, 22, 24, 29, 33, 33, 41–43

simple_structure, 6
simplimax, 3
simplimax (rotations), 33
summary, 31
summary.GPArotation, 4
summary.GPArotation
(print.GPArotation), 30

tandemI, 3
tandemI (rotations), 33
tandemII, 3
tandemII (rotations), 33
targetQ, 2
targetQ (rotations), 33
targetT, 2
targetT (rotations), 33
Thurstone, 9, 22, 24, 40, 43

update, 4

Varimax, 3
Varimax (rotations), 33
varimax, 4
varimin, 3
varimin (rotations), 33
vgQ, 6, 15, 38, 41
vgQ.bentler, 5
vgQ.bifactor, 5
vgQ.bigeomin, 5
vgQ.cf, 5

vgQ.entropy, *5*
vgQ.geomin, *5*
vgQ.infomax, *5*
vgQ.lp.wls, *5*, *26*
vgQ.mccammon, *5*
vgQ.oblimax, *5*
vgQ.oblimin, *5*
vgQ.pst, *5*
vgQ.quartimax, *5*
vgQ.quartimin, *5*
vgQ.simplimax, *5*
vgQ.tandemI, *5*
vgQ.tandemII, *5*
vgQ.target, *5*
vgQ.varimax, *5*
vgQ.varimin, *5*

WansbeekMeijer, *9*, *24*, *41*, 43