Package 'AlignLV'

October 4, 2024

Title Multiple Group Item Response Theory Alignment Helpers for 'lavaan' and 'mirt'

Version 0.1.0.0

Description Allows for multiple group item response theory alignment a la 'Mplus' to be applied to lists of single-group models estimated in 'lavaan' or 'mirt'. Allows item sets that are overlapping but not identical, facilitating alignment in secondary data analysis where not all items may be shared across assessments.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports abind, mirt, lavaan, magrittr, purrr, stats, dplyr, tibble, tidyr, rlang

Suggests doRNG, doParallel, foreach, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Maxwell Mansolf [aut, cre] (<https://orcid.org/0000-0001-6861-8657>)

Maintainer Maxwell Mansolf <maxwell.mansolf@northwestern.edu>

Repository CRAN

Date/Publication 2024-10-04 10:00:08 UTC

Contents

`align.optim` *Runs alignment optimizer*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
align.optim(
  stacked,
  n,
  estimator,
  nstarts = 50,
  ncores = 3,
  hyper.first,
  center.means,
  eps.alignment,
  clf.ignore.quantile,
  verbose
)
```

Arguments

`stacked`	Stacked parameter estimates from `stackEstimates`
`n`	Sample size in each group
`estimator`	See `Alignment` documentation.
`nstarts`	Number of starting values for alignment; default is 10
`ncores`	See `Alignment` documentation.
`hyper.first`	See `Alignment` documentation.
`center.means`	See `Alignment` documentation.
`eps.alignment`	See `Alignment` documentation.
`clf.ignore.quantile`	See `Alignment` documentation.
`verbose`	See `Alignment` documentation.

Details

See example for `Alignment` for examples

Value

A list of results from multiple runs of the alignment optimizer:

- mv Means and variances from each alignment run.
- parout A table of outputs from link[stats]{optim} containing the function values, convergence information, and resulting estimates of means and variances from each run.
- nFailedRuns The number of runs that failed to complete. An error is returned if no runs fail.

Alignment *Multiple-Group Factor Analysis Alignment from* mirt *or* lavaan

---

Description

Performs alignment (https://www.statmodel.com/Alignment.shtml) using single-group models estimated in mirt or lavaan.

Usage

```
Alignment(
  fitList,
  estimator,
  SE = FALSE,
  eps.alignment = 0.01,
  clf.ignore.quantile = 0.1,
  bifactor.marginal = FALSE,
  hyper.first = "variances",
  center.means = TRUE,
  nstarts = 10,
  ncores = 1,
  verbose = TRUE
)
```

Arguments

fitList	A list of fitted model objects. Currently only works for single-group, unidimensional or bifactor models with no covariates estimated in mirt or lavaan.
estimator	The model type used, either 'mirt.grm' for the graded response model estimated in mirt or 'lavaan.ordered' for the categorical factor analysis model applied by lavaan when the ordered input includes all variables in the model.
SE	Whether to also return standard errors from parameter estimates after alignment. SE's are transformed using the delta method from those provided in the original model objects, which must (for mirt), have been fitted with standard errors estimated (SE=TRUE).

`eps.alignment` A numeric scalar for the alignment simplicity function, given by (Asparouhov & Muthén, 2014, *Structural Equation Modeling*):

$$\sqrt{\sqrt{x^2 + \epsilon}}$$

where x is the difference between corresponding estimates in each pair of aligned models. Lower values may cause numerical instability; default 0.01

`clf.ignore.quantile`
Another protection from numerical instability; CLF values less than the `clf.ignore.quantile` of the full set of CLF values are ignored when calculating the complexity function at each step. Default 0.1 for removing the lowest 10% of CLF values.

`bifactor.marginal`
A logical scalar indicating whether, for bifactor models, alignment should take place on the marginal, rather than conditional, metric for slopes (Ip, 2010, *Applied Psychological Measurement*).

`hyper.first` A string scalar denoting which hyperparameter to align first. Asparouhov & Muthén (2014) align all parameters simultaneously (`'no'`); `'variances'` (default) performs a two-step process, first aligning variances, then aligning means conditional on variance estimates from the first step. `'means'` does the reverse.

`center.means` A logical scalar. Alignment fixes the first group's mean to zero to estimate the others. If `center.means` is `TRUE` (default), aligned means and models are returned after subtracting the weighted mean `weighted.mean` from all mean estimates, yielding a (weighted) grand mean of zero. Variances are automatically rescaled such that their weighted product (i.e., log of weighted mean of `e^(variance)`) is 1.

`nstarts` Number of starting values for alignment; default is 10

`ncores` Number of processor cores to distribute alignment starts across; on systems that support multicore processing, using additional cores can speed up the alignment step by roughly a factor of the number of cores. Defaults to 1 for no parallel processing. Requires the `doRNG` package and defaults to sequential processing if not installed.

`verbose` Whether stuff gets printed to the console. May help with debugging.

Details

Currently, no automated process provides statistical tests for DIF. Instead, I recommend interpreting the DIF impact directly by comparing scores obtained from a single-group model combining all groups, and the multiple models produced by `Alignment`. If standard errors are requested from `getEstimates.mirt`, or `getEstimates.lavaan`, and then the corresponding `transformEstimates.mirt.grm` or `transformEstimates.lavaan.ordered` is applied, SE's after alignment can be obtained and used for multiple comparison testing, but this is not yet automated. Alternatively, consider re-fitting models with means and variances fixed to those obtained from alignment to obtain these standard errors. In the latter case, especially when priors are used as in `mirt`, your estimates may not match those from `Alignment` exactly.

For `lavaan`, the metric for alignment must be the "theta" parameterization, which is not the default, in order to properly search for latent means and variances, because only then do the transformations

apply. My current thinking: under the delta parameterization, the transformed estimates (calculate delta, incorporate it into parameters, then transform parameters, BUT don't reverse the delta transformation) do NOT yield an equivalent model, but DO yield a model that can be compared across groups. In order to get an equivalent model, you also need to reverse the delta transformation at the end. To account for this, if the the extra argument toCompare should be turned on `TRUE` if transformed parameters are to be compared for equivalence across groups. Turning it off results in NOT applying the reverse of the delta transformation at the end. This currently is fixed to TRUE and cannot be modified, but you can access `transformEstimates.lavaan.ordered` directly if you want to play around.

If `parallel==TRUE`, a parallel backend with the `doParallel` package leverages multi-core processing if the number of cores specified in `ncores` is greater than one. Uses `%dorng%` to pass the R session's seed to the alignment optimizer, such that you can replicate random starts with `set.seed` (see example).

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A `list` with the following elements:

- `fit` A `list` of fitted objects of type `mirt` or `lavaan`, depending on the estimator, where models were re-estimated with means and variances set to those obtained from alignment.
- `est.og` A nested `list` of parameter estimates and standard errors provided to the alignment optimizer from the provided models. Each element corresponds to a provided model, and each element thereof corresponds to a parameter name (e.g., `a` and `d` parameters from `mirt.grm`) and contains a matrix of the corresponding estimates.
- `est` The estimates from `est.og`, transformed after alignment using the obtained mean and variance estimates therefrom.
- `hypers` A `list` of two-element numeric vectors, where `mean` gives the estimated mean from alignment in the corresponding group, and `var` the estimated variance.
- `parout` Optimizer output for the alignment step, used to examine convergence. Contains the following columns:
  - `f` The final complexity function value from alignment.
  - `convergence` The `convergence` output from `optim`
  - `M.2` to `M.`(number of groups minus 2) The estimated means from alignment
  - `V.2` to `M.`(number of groups minus 2) The estimated variances from alignment

Examples

```
#load data
library(mirt)
library(lavaan)
library(purrr)
library(tibble)
library(magrittr)
dat=expand.table(Bock1997)
#fit configural models
```

```
fit.mirt=mirt(dat,1,SE=TRUE)
fit.lavaan=cfa(model='G =~ Item.1+Item.2+Item.3',data=dat,
               ordered=c('Item.1','Item.2','Item.3'),
               std.lv=TRUE,parameterization='delta')
(fit.lavaan@ParTable)%>%tibble::as_tibble()%>%print(n=Inf)
#test stuff
tab=fit.lavaan@ParTable
tab$start[23]=3
tab$est[23]=3
fit.lavaan2=lavaan(tab,data=fit.lavaan@Data)

#get estimates
est.mirt=getEstimates.mirt(fit.mirt,SE=TRUE,bifactor.marginal=FALSE)
est.lavaan=getEstimates.lavaan(fit.lavaan,SE=TRUE)

#test transformations
newMean=10
newVar=2
test.mirt=transformEstimates.mirt.grm(newMean,newVar,est.mirt)
test.lavaan=transformEstimates.lavaan.ordered(
              newMean,newVar,est.lavaan,toCompare=TRUE)
#load and test equivalence
tfit.mirt=loadEstimates.mirt.grm(fit.mirt,newMean,newVar,newpars=test.mirt,
                                 verbose=TRUE)
test.mirt=mirt::coef(fit.mirt)
test.mirt
tfit.lavaan=loadEstimates.lavaan.ordered(
              fit.lavaan,newMean,newVar,newpars=test.lavaan,
              verbose=TRUE)
tfit.lavaan@ParTable%>%tibble::as_tibble()%>%print(n=Inf)
test.lavaan

#now on stacked estimates
estList=list(est.mirt%>%purrr::imap(function(x,n){
  rownames(x)[2]=paste0(rownames(x)[2],'_ho')
  if(!n%in%c('a','se.a'))colnames(x)[2]=paste0(colnames(x)[2],'_ho')
  x
}),est.mirt%>%purrr::imap(function(x,n){
  rownames(x)[1]=paste0(rownames(x)[1],'_hi')
  if(!n%in%c('a','se.a'))colnames(x)[1]=paste0(colnames(x)[1],'_hi')
  x
}))
stack=stackEstimates(estList)
test.stack=transformEstimates.mirt.grm(c(0,0),c(1,1),stack)
sf.stack=SF.mplus3D(c(0,1),stack,combn(1:2,2),c(100,200),'mirt.grm',
                                       eps.alignment=0.01,
                                       clf.ignore.quantile=0.1)
test.stack2=transformEstimates.mirt.grm(c(0,1),c(1,1/2),stack)

#try align?
#lavaan
set.seed(0)
sim.base=list(simdata(a=as.numeric(est.mirt$a),d=est.mirt$d,N=5000,
```

```
                    itemtype='graded',sigma=matrix(1),mu=0),
              simdata(a=as.numeric(est.mirt$a),d=est.mirt$d,N=5000,
                    itemtype='graded',sigma=matrix(2),mu=1))
fit.base=sim.base%>%purrr::map(~cfa(model="G =~ Item_1 + Item_2 + Item_3",
                            data=as.data.frame(.),
                            ordered=paste0('Item_',1:3),std.lv=TRUE,
                            parameterization='delta'))
fit.base%>%purrr::map(lavInspect,'est')%>%purrr::transpose()
est.base=purrr::map(fit.base,getEstimates.lavaan,SE=TRUE)
#not run: using parallel processes with ncores=3
set.seed(1)
# align.stack=align.optim(stackEstimates(est.base),c(100,200),nstarts=3,
#                         hyper.first='variances',ncores=3,
#                         eps.alignment=0.01,clf.ignore.quantile=0.1,
#                        estimator='lavaan.ordered',center.means=FALSE,
#                        verbose=TRUE)
# #same seed
# set.seed(1)
# align.stack=align.optim(stackEstimates(est.base),c(100,200),nstarts=3,
#                         hyper.first='variances',ncores=3,
#                         eps.alignment=0.01,clf.ignore.quantile=0.1,
#                         estimator='lavaan.ordered',center.means=FALSE,
#                         verbose=TRUE)
#sequential
align.stack=align.optim(stackEstimates(est.base),c(100,200),nstarts=3,
                       hyper.first='variances',ncores=1,
                       eps.alignment=0.01,clf.ignore.quantile=0.1,
                       estimator='lavaan.ordered',center.means=FALSE,
                       verbose=TRUE)
align.stack
fit.align=Alignment(fit.base,'lavaan.ordered',center.means=FALSE,SE=TRUE,
            verbose=TRUE)

#mirt
fit.base2=list()
for(i in 1:length(sim.base)){
  fit.base2[[i]]=mirt(sim.base[[i]],1,'graded',SE=TRUE)
}
est.base2=purrr::map(fit.base2,getEstimates.mirt,SE=TRUE,
bifactor.marginal=FALSE)
#not run: using parallel processes with ncores=3
# align.stack2=align.optim(stackEstimates(est.base2),c(100,200),nstarts=3,
#                          hyper.first='variances',ncores=3,
#                          eps.alignment=0.01,clf.ignore.quantile=0.1,
#                          estimator='mirt.grm',center.means=FALSE)
align.stack2=align.optim(stackEstimates(est.base2),c(100,200),nstarts=3,
                        hyper.first='variances',ncores=1,
                        eps.alignment=0.01,clf.ignore.quantile=0.1,
                        estimator='mirt.grm',center.means=FALSE,
                        verbose=TRUE)
align.stack2
fit.align2=Alignment(fit.base2,'mirt.grm',center.means=FALSE,SE=TRUE)
```

```
#did it work?
fit.align$hypers
fit.align2$hypers
fit.align$est%>%purrr::transpose()%>%purrr::map(~mean(.[[1]]-.[[2]]))
fit.align2$est%>%purrr::transpose()%>%purrr::map(~mean(.[[1]]-.[[2]]))
fit.align$fit
fit.align2$fit
(fit.align$fit%>%purrr::map(~.@ParTable%>%
tibble::as_tibble()%>%dplyr::filter(free!=0))%>%
  purrr::transpose())[c('start','est')]%>%purrr::map(~mean(.[[1]]-.[[2]]))
(fit.align2$fit%>%purrr::map(coef)%>%
    purrr::transpose())[paste0('Item_',1:3)]%>%
    purrr::map(~mean(.[[1]]-.[[2]]))
#appears so!
```

`getEstimates.lavaan` *Prepare* `lavaan` *estimates for alignment*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
getEstimates.lavaan(fit, SE = TRUE)
```

Arguments

`fit`	A `lavaan` object compatible with `Alignment`
`SE`	logical; whether to also obtain standard errors.

Details

See example for `Alignment` for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A `list` of estimates in a format amenable to subsequent alignment

getEstimates.mirt *Prepare* mirt *estimates for alignment*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
getEstimates.mirt(fit, SE = FALSE, bifactor.marginal = FALSE)
```

Arguments

fit	A mirt object compatible with Alignment
SE	logical; whether to also obtain standard errors.
bifactor.marginal	See Alignment documentation.

Details

See example for Alignment for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A list of estimates in a format amenable to subsequent alignment

loadEstimates.lavaan.ordered *Estimate* lavaan *models using aligned parameter estimates*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
loadEstimates.lavaan.ordered(
  fit,
  align.mean,
  align.variance,
  newpars,
  do.fit = TRUE,
  verbose = TRUE
)
```

Arguments

fit	A mirt object compatible with Alignment
align.mean	Mean to transform model to.
align.variance	Variance to transform model to.
newpars	New (transformed) estimates to load into model object.
do.fit	Whether to re-fit the model after loading and fixing estimates.
verbose	See Alignment documentation.

Details

See example for Alignment for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A lavaan object, based on fit but with modified parameters.

---

loadEstimates.mirt.grm

Estimate mirt *models using aligned parameter estimates*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
loadEstimates.mirt.grm(
  fit,
  align.mean,
  align.variance,
  newpars,
  do.fit = TRUE,
  verbose = TRUE
)
```

Arguments

fit	A mirt object compatible with Alignment
align.mean	Mean to transform model to.
align.variance	Variance to transform model to.
newpars	New (transformed) estimates to load into model object.
do.fit	Whether to re-fit the model after loading and fixing estimates.
verbose	See Alignment documentation.

Details

See example for Alignment for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A mirt, object based on fit but with modified parameters.

SF.mplus3D *Simplicity function for alignment*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
SF.mplus3D(
  pars,
  est,
  comb,
  nobs,
  estimator,
  eps.alignment,
  clf.ignore.quantile,
  hyper = "all",
  otherHyper = NULL
)
```

Arguments

pars	Hyperparameters to feed into optimizer
est	Estimates to transform, from getEstimates.mirt or getEstimates.lavaan
comb	All combinations of groups from combn
nobs	Sample size in each group
estimator	See Alignment documentation.
eps.alignment	See Alignment documentation.
clf.ignore.quantile	See Alignment documentation.
hyper	Hyperparameter to calculate simplicity function for; see Alignment documentation.
otherHyper	Non-included hyperparameter

Details

See example for `Alignment` for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

A value of the simplicity function from Asparouhuv & Muthen, 2014.

`stackEstimates` *Stack estimates for optimization*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
stackEstimates(estList)
```

Arguments

`estList`	List of estimates from `getEstimates.lavaan` or `getEstimates.mirt` to stack to feed into `SF.mplus3D`

Details

See example for `Alignment` for examples

Value

A set of estimates prepared for efficient use with `SF.mplus3D`

`transformEstimates.lavaan.ordered`
Transform `lavaan` *estimates using aligned estimates of latent mean and variance*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
transformEstimates.lavaan.ordered(
  align.mean,
  align.variance,
  est,
  toCompare = FALSE
)
```

Arguments

align.mean	Mean to transform model to.
align.variance	Variance to transform model to.
est	Estimates to transform, from getEstimates.lavaan
toCompare	Accounts for discrepancies between delta and theta parameterizations; see Alignment documentation.

Details

See example for Alignment for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

Estimates in the same structure as from getEstimates.lavaan, but transformed from (assumed) mean 0 and variance 1 to the metric specified by align.mean and align.variance.

---

transformEstimates.mirt.grm

Transform mirt *estimates using aligned estimates of latent mean and variance*

Description

Not generally intended to be used on its own, but exported anyway for didactic purposes.

Usage

```
transformEstimates.mirt.grm(align.mean, align.variance, est)
```

Arguments

align.mean	Mean to transform model to.
align.variance	Variance to transform model to.
est	Estimates to transform, from getEstimates.mirt

Details

See example for `Alignment` for examples

This program was designed based on the published work of Asparouhov & Muthen, and was not intended to match Mplus exactly, and may not.

Value

Estimates in the same structure as from `getEstimates.mirt`, but transformed from (assumed) mean 0 and variance 1 to the metric specified by `align.mean` and `align.variance`.

Index