Package 'umx'

April 7, 2017

Version 1.7.5

Date 2017-04-01

Title Structural Equation Modelling in R with 'OpenMx'

Maintainer Timothy C Bates <timothy.c.bates@gmail.com>

License GPL-3

URL http://github.com/tbates/umx

Description Quickly create, run, and report structural equation and twin models. See '?umx' to learn more.

Depends R (>= 3.2.0), OpenMx (>= 2.5.0)

Imports DiagrammeR, knitr, MASS, Matrix, methods, mvtnorm, numDeriv, polycor, R2HTML, RCurl, sfsmisc, utils

Suggests devtools, testthat, gdata, cocor

BugReports http://github.com/tbates/umx/issues

LazyData true

RoxygenNote 6.0.1

NeedsCompilation no

Author Timothy C Bates [aut, cre]

Repository CRAN

Date/Publication 2017-04-07 06:07:27 UTC

R topics documented:

`confint.MxModel` *Get confidence intervals from an MxModel*

Description

Implements confidence interval function for OpenMx models. Note: Currently requested CIs are added to existing CIs, and all are run, even if they alrady exist in the output. This might change in the future.

Usage

```
## S3 method for class 'MxModel'
confint(object, parm = c("existing", "all",
  "vector of names"), level = 0.95, run = FALSE, showErrorCodes = FALSE,
  ...)
```

Arguments

object	An mxModel, possibly already containing mxCIs that have been mxRun with intervals = TRUE))
parm	A specification of which parameters are to be given confidence intervals. Can be "existing", "all", or a vector of names.
level	The confidence level required (default = .95)
run	Whether to run the model (defaults to FALSE)
showErrorCodes	(default = FALSE)
...	Additional argument(s) for umxConfint.

Details

Unlike confint, if parm is missing, all CIs requested will be added to the model, but (because these can take time to run) by default only CIs already computed will be reported.

CIs will be run only if run is TRUE, allowing this function to be used to add CIs without automatically having to run them. If parm is empty, and run = FALSE, a message will alert you to add run = TRUE. Even a few CIs can take too long to make running the default.

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

- confint

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(from = latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
m2 = confint(m1, "all") # default: CIs added, but user prompted to set run = TRUE
m2 = confint(m2, run = TRUE) # CIs run and reported
# Add CIs for asymmetric paths in RAM model, report them, save m1 with this CI added
```

```
m1 = confint(m1, parm = "G_to_x1", run = TRUE)
# Add CIs for asymmetric paths in RAM model, report them, save m1 with mxCIs added
m1 = confint(m1, parm = "A", run = TRUE)
confint(m1, parm = "existing") # request existing CIs (none added yet...)
```

---

dl_from_dropbox *dl_from_dropbox*

Description

Download a file from Dropbox, given either the url, or the name and key

Usage

```
dl_from_dropbox(x, key = NULL)
```

Arguments

x	Either the file name, or full dropbox URL (see example below)
key	the code after s/ and before the file name in the dropbox url

Details

Improvements would include error handling...

Value

- NULL

References

- http://thebiobucket.blogspot.kr/2013/04/download-files-from-dropbox.html

See Also

Other File Functions: umx_make_sql_from_excel, umx_move_file, umx_open, umx_rename_file, umx

Examples

```
## Not run:
dl_from_dropbox("https://dl.dropboxusercontent.com/s/7kauod48r9cfhwc/tinytwinData.rda")
dl_from_dropbox("tinytwinData.rda", key = "7kauod48r9cfhwc")

## End(Not run)
```

ex9_6 *A multilevel dataset.*

Description

A dataset for use in multivariate model example.

Usage

```
data(ex9_6)
```

Format

A data frame with 1000 rows and 8 variables:

Details

y1 something

y2 something

y3 something

y4 something

x1 something

x2 something

w something

clusterID something

Source

https://www.statmodel.com/usersguide/chap9/ex9.6.html

Examples

```
data(ex9_6)
names(ex9_6)
```

`extractAIC.MxModel` *extractAIC from MxModel*

Description

Returns the AIC for an OpenMx model helper function for `logLik.MxModel` (which enables AIC(model); logLik(model); BIC(model) Original Author: brandmaier

Usage

```
## S3 method for class 'MxModel'
extractAIC(fit, scale, k, ...)
```

Arguments

`fit`	an fitted `mxModel` from which to get the AIC
`scale`	not used
`k`	not used
`...`	any other parameters (not used)

Value

- AIC value

References

- http://openmx.ssri.psu.edu/thread/931#comment-4858

See Also

- `AIC`, `umxCompare`, `logLik.MxModel`

Other Reporting functions: `RMSEA.MxModel`, `RMSEA.summary.mxmodel`, `RMSEA`, `confint.MxModel`, `loadings`, `logLik.MxModel`, `plot.MxModel`, `residuals.MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEcov`, `umxPlotACE`, `umxPlotCP`, `umxPlotGxE`, `umxPlotIP`, `umxSummary.MxModel`, `umxSummaryACE`, `umx_drop_ok`, `umx_standardize_RAM`

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
```

```
extractAIC(m1)
# -2.615998
AIC(m1)
```

`install.OpenMx` *Easily install the latest parallel/NPSOL enabled build of OpenMx.*

Description

You can install from UVa, from travis latest, from a custom url, or open the list of travis builds.

Usage

```
install.OpenMx(loc = c("UVa", "latest", "travis"), url = NULL)
```

Arguments

`loc`	Where to install from: "UVa" (the default), "latest" (travis build), or open the "travis" list of builds.
`url`	A custom URL if you have/need one (probably not)

Value

-

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Miscellaneous Functions: umx_get_options, umx_open_CRAN_page, umx_pad

Examples

```
## Not run:
install.OpenMx()

## End(Not run)
```

`loadings` *loadings Generic loadings function to extract factor loadings from exploratory or confirmatory factor analyses.*

Description

See `loadings.MxModel` to access the loadings of OpenMx EFA models.

Usage

```
loadings(x, ...)
```

Arguments

`x`	an object from which to get loadings
`...`	additional parameters

Details

Base `loadings` handles `factanal` objects.

Value

- matrix of loadings

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Reporting functions: `RMSEA.MxModel`, `RMSEA.summary.mxmodel`, `RMSEA`, `confint.MxModel`, `extractAIC.MxModel`, `logLik.MxModel`, `plot.MxModel`, `residuals.MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEcov`, `umxPlotACE`, `umxPlotCP`, `umxPlotGxE`, `umxPlotIP`, `umxSummary.MxModel`, `umxSummaryACE`, `umx_drop_ok`, `umx_standardize_RAM`

loadings.MxModel *loadings.MxModel*

Description

loadings extracts the factor loadings from an OpenMx EFA (factor analysis) model. It behaves equivalently to stats::loadings in returning the loadings from an EFA (factor analysis), but doesn't store the rotation matrix.

Usage

```
## S3 method for class 'MxModel'
loadings(x, ...)
```

Arguments

x	A RAM model to get which to get loadings
...	Other parameters (currently unused)

Value

- loadings matrix

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- factanal, loadings

Other Reporting Functions: umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx_time, umx

Examples

```
myVars <- c("mpg", "disp", "hp", "wt", "qsec")
m1 = umxEFA(name = "test", factors = 2, data = mtcars[, myVars])
loadings(m1)
```

logLik.MxModel *logLik.MxModel*

Description

Returns the log likelihood for an OpenMx model. This helper also enables AIC(model); BIC(model).

Usage

```
## S3 method for class 'MxModel'
logLik(object, ...)
```

Arguments

object	the mxModel from which to get the log likelihood
...	Optional parameters

Details

hat-tip Andreas Brandmaier

Value

- the log likelihood

References

- http://openmx.ssri.psu.edu/thread/931#comment-4858

See Also

- AIC, umxCompare

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
```

```
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
logLik(m1)
AIC(m1)
```

`plot.MxModel` *Create a graphical figure from an MxModel*

Description

Create graphical path diagrams from your OpenMx models!

Usage

```
## S3 method for class 'MxModel'
plot(x = NA, std = FALSE, digits = 2, file = "name",
  pathLabels = c("none", "labels", "both"), fixed = TRUE, means = TRUE,
  resid = c("circle", "line", "none"), showMeans = "deprecated",
  showFixed = "deprecated", ...)
```

Arguments

`x`	An `mxModel` from which to make a path diagram
`std`	Whether to standardize the model (default = FALSE).
`digits`	The number of decimal places to add to the path coefficients
`file`	The name of the dot file to write: NA = none; "name" = use the name of the model
`pathLabels`	Whether to show labels on the paths. both will show both the parameter and the label. ("both", "none" or "labels")
`fixed`	Whether to show fixed paths (defaults to TRUE)
`means`	Whether to show means or not (default = TRUE)
`resid`	How to show residuals and variances default is "circle". Options are "line" & "none"
`showMeans`	Deprecated: just use 'means = TRUE'
`showFixed`	Deprecated: just use 'fixed = TRUE'
`...`	Optional parameters

Details

plot() produces SEM diagrams in graphviz format, and relies on DiagrammeR (or a graphviz application) to create the image. The commercial application Omnigraffle is great for editing these images.

On unix and windows, plot() will create a pdf and open it in your default pdf reader.

If you use umx_set_plot_format("graphviz"), files will open in with a graphviz helper.

Note: DiagrammeR is supported out of the box. If you use graphviz, we try and use that app, but YOU HAVE TO INSTALL IT! On OS X we try and open an app: you may need to associate the '.gv' extension with the graphviz app. Find the .gv file made by plot, get info (cmd-I), then choose "open with", select Graphviz.app (or OmniGraffle professional), then set "change all".

References

- http://www.github.com/tbates/umx, https://en.wikipedia.org/wiki/DOT_(graph_description_language)

See Also

- umx_set_plot_format, plot.MxModel, umxPlotACE, umxPlotCP, umxPlotIP, umxPlotGxE,

Other Core Modelling Functions: umxDiagnose, umxLatent, umxMatrix, umxPath, umxRAM, umxReduce, umxRun, umx

Other Plotting functions: umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Other Twin Modeling Functions: umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
plot(m1)
```

qm *qm*

Description

Quickmatrix function

Usage

```
qm(..., rowMarker = "|")
```

Arguments

...	the components of your matrix
rowMarker	mark the end of each row

Value

- matrix

References

http://www.sumsar.net/blog/2014/03/a-hack-to-create-matrices-in-R-matlab-style

See Also

Other Utility Functions: umx_find_object, umx_grep, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
# simple example
qm(0, 1 |
   2, NA)
## Not run:
# clever example
M1 = M2 = diag(2)
qm(M1,c(4,5) | c(1,2),M2 | t(1:3))

## End(Not run)
```

reliability *reliability*

Description

Compute and report Coefficient alpha (extracted from Rcmdr to avoid its dependencies)

Usage

```
reliability(S)
```

Arguments

S	A square, symmetric, numeric covariance matrix

Value

-

References

- https://cran.r-project.org/package=Rcmdr

See Also

Other Stats Functions: umxCov2cor, umx_cor, umx_means, umx

Examples

```
# treat vehicle aspects as items of a test
reliability(cov(mtcars))
```

residuals.MxModel *Get residuals from an MxModel*

Description

Return the residuals from an OpenMx RAM model

Usage

```
## S3 method for class 'MxModel'
residuals(object, digits = 2, suppress = NULL, ...)
```

Arguments

object	An fitted mxModel from which to get residuals
digits	round to how many digits (default = 2)
suppress	smallest deviation to print out (default = NULL = show all)
...	Optional parameters

Value

- matrix of residuals

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("g")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
residuals(m1)
residuals(m1, digits = 3)
residuals(m1, digits = 3, suppress = .005)
# residuals are returned as an invisible object you can capture in a variable
a = residuals(m1); a
```

RMSEA *Generic RMSEA function*

Description

See RMSEA.MxModel to access the RMSEA of MxModels

Usage

```
RMSEA(x, ci.lower, ci.upper, digits)
```

Arguments

x	an object from which to get the RMSEA
ci.lower	the lower CI to compute
ci.upper	the upper CI to compute
digits	digits to show

Value

- RMSEA object containing value (and perhaps a CI)

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

RMSEA.MxModel *RMSEA function for MxModels*

Description

Compute the confidence interval on RMSEA

Usage

```
## S3 method for class 'MxModel'
RMSEA(x, ci.lower = 0.05, ci.upper = 0.95, digits = 3)
```

Arguments

x	an mxModel from which to get RMSEA
ci.lower	the lower CI to compute
ci.upper	the upper CI to compute
digits	digits to show (defaults to 3)

Value

- object containing the RMSEA and lower and upper bounds

References

- https://github.com/simsem/semTools/wiki/Functions, https://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
RMSEA(m1)
```

RMSEA.summary.mxmodel *RMSEA function for MxModels*

Description

Compute the confidence interval on RMSEA

Usage

```
## S3 method for class 'summary.mxmodel'
RMSEA(x, ci.lower = 0.05, ci.upper = 0.95,
  digits = 3)
```

Arguments

x	an mxModel summary from which to get RMSEA
ci.lower	the lower CI to compute
ci.upper	the upper CI to compute
digits	digits to show (defaults to 3)

Value

- object containing the RMSEA and lower and upper bounds

References

- https://github.com/simsem/semTools/wiki/Functions, https://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
RMSEA(m1)
```

---

umx *Helper Functions for Structural Equation Modelling in OpenMx*

Description

umx allows you to more easily build, run, modify, and report models using OpenMx with code. The core functions are linked below under **See Also**

Details

The functions are organized into families: Have a read of these below, click to explore.

All the functions have explanatory examples, so use the help, even if you think it won't help :-) Have a look, for example at umxRAM

Introductory working examples are below. You can run all demos with demo(umx) When I have a vignette, it will be: vignette("umx", package = "umx")

There is a helpful blog at http://tbates.github.io

If you want the bleeding-edge version:

devtools::install_github("tbates/umx")

References

- http://www.github.com/tbates/umx

See Also

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxMatrix, umxPath, umxRAM, umxReduce, umxRun

Other Super-easy helpers: umxEFA, umxTwoStage

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx_time

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxFixAll, umxMI, umxSetParameters, umxUnexplainedCausalNexus

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_print, umx_rename

Other Stats Functions: reliability, umxCov2cor, umx_cor, umx_means

Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_move_file, umx_open, umx_rename_file

Other zAdvanced Helpers: umx_standardize_ACEcov, umx_standardize_ACE, umx_standardize_CP, umx_standardize_IP

Examples

```
require("umx")
data(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = nrow(demoOneFactor))
latents = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = myData,
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents  , fixedAt=1)
)

# umx added informative labels, created starting values,
# Ran you model (if autoRun is on), and displayed a brief summary
```

```
# including a comparison if you modified a model...!

# Let's get some journal-ready fit information

umxSummary(m1)
umxSummary(m1, show = "std") #also display parameter estimates
# You can get the coefficients of an MxModel with coef(), just like for lm etc.
coef(m1)

# ==================
# = Model updating =
# ==================
# Can we set the loading of X5 on G to zero?
m2 = omxSetParameters(m1, labels = "G_to_x1", values = 0, free = FALSE, name = "no_g_on_X5")
m2 = mxRun(m2)
# Compare the two models
umxCompare(m1, m2)

# Use umxModify to do the same thing in 1-line
m2 = umxModify(m1, "G_to_x1", name = "no_effect_of_g_on_X5", comparison = TRUE)

# ========================
# = Confidence intervals =
# ========================

# umxSummary() will show these, but you can also use the confint() function
confint(m1, parm = 'all', run = TRUE) # lots more to learn about ?confint.MxModel

# And make a Figure in dot (.gv) format!
plot(m1, std = TRUE)

# If you just want the .dot code returned set file = NA
plot(m1, std = TRUE, file = NA)
```

`umx-deprecated` *Deprecated. May already stop() code and ask to be updated. May be dropped entirely in future.*

Description

umxSaturated should be replaced with `mxRefModels`

umx_grep_labels should be replaced with `umx_grep`

grepSPSS_labels should be replaced with `umx_grep`

umxStart should be replaced with `umxValues`

umxTryHard is deprecated: use `umxRun` instead

genEpi_Jiggle is deprecated: use `umxJiggle` instead

umxLabels Is deprecated: use `umxLabel` instead

umxLabels Is deprecated: use umxLabel instead

umxPath is deprecated: Use mxPath and umxLabel instead

umxReportFit is deprecated: use umxSummary instead

umxGetLabels is deprecated: use umxGetParameters instead

stringToMxAlgebra is deprecated: please use umx_string_to_algebra instead

genEpi_EvalQuote is deprecated: please use umxEval instead

umxReportCIs is deprecated: please use umxCI instead

hasSquareBrackets is deprecated: please use umx_has_square_brackets instead

xmuHasSquareBrackets is deprecated: please use umx_has_square_brackets instead

replace umxReportFit with umxSummary

Replace umxGraph_RAM with plot

Replace tryHard with mxTryHard

Replace genEpi_ReRun with umxModify

Replace mxStart with umxValues

Replace umxLabeler with umxLabel

Replace standardizeRAM with umx_standardize_RAM

Replace genEpi_equate with umxEquate

Replace genEpi_Path with umxPath

Replace genEpiCompare with umxCompare

Replace mxLatent with umxLatent

Change col.as.numeric to umx_as_numeric

Change cor.prob to umx_cor

Change umx_u_APA_pval to umx_APA_pval

Arguments

...	the old function's parameters (now stripped out to avoid telling people how to do it the wrong way :-)

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other umx deprecated: umx_get_cores, umx_get_optimizer

umxACE *umxACE: Build and run a 2-group Cholesky (uni- or multi-variate)*

Description

A common task in twin modelling involves using the genetic and environmental differences between large numbers of pairs of mono-zygotic (MZ) and di-zygotic (DZ) twins reared together to model the genetic and environmental structure of one, or, typically, several phenotypes (measured behaviors). umxACE supports modeling with the ACE Cholesky model, a core model in behavior genetics (Cardon and Neale, 1996).

Usage

```
umxACE(name = "ACE", selDVs, selCovs = NULL, dzData, mzData,
  suffix = NULL, dzAr = 0.5, dzCr = 1, addStd = TRUE, addCI = TRUE,
  numObsDZ = NULL, numObsMZ = NULL, boundDiag = NULL, weightVar = NULL,
  equateMeans = TRUE, bVector = FALSE, thresholds = c("deviationBased",
  "WLS"), autoRun = getOption("umx_auto_run"), sep = NULL)
```

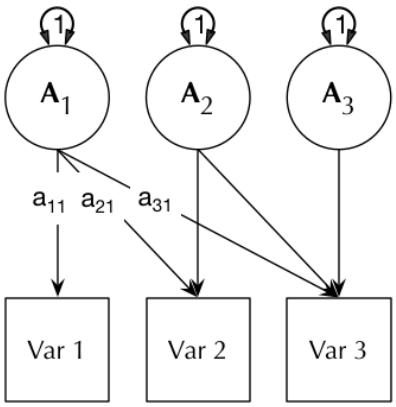
Arguments

name	The name of the model (defaults to"ACE")
selDVs	The variables to include from the data
selCovs	(optional) covariates to include from the data (do not include suffix in names)
dzData	The DZ dataframe
mzData	The MZ dataframe
suffix	The suffix for twin 1 and twin 2, often "_T". If set, simplifies SelDVs, i.e., just "dep" not c("dep_T1", "dep_T2")
dzAr	The DZ genetic correlation (defaults to .5, vary to examine assortative mating)
dzCr	The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model)
addStd	Whether to add the algebras to compute a std model (defaults to TRUE)
addCI	Whether to add intervals to compute CIs (defaults to TRUE)
numObsDZ	= Number of DZ twins: Set this if you input covariance data
numObsMZ	= Number of MZ twins: Set this if you input covariance data
boundDiag	= (optional) numeric lbound for diagonal of the a, c, and e matrices, e.g. 0
weightVar	= If provided, a vector objective will be used to weight the data. (default = NULL)
equateMeans	Whether to equate the means across twins (defaults to TRUE)
bVector	Whether to compute row-wise likelihoods (defaults to FALSE)
thresholds	How to implement ordinal thresholds c("deviationBased", "WLS")
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)
sep	allowed as a synonym for "suffix"

Details

This model decomposes phenotypic variance into Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D). Scroll down to details for how to use the function, a figure and multiple examples.

The Cholesky or lower-triangle decomposition allows a model which is both sure to be solvable, and also to account for all the variance (with some restrictions) in the data. This model creates as many latent A C and E variables as there are phenotypes, and, moving from left to right, decomposes the variance in each component into successively restricted factors. The following figure shows how the ACE model appears as a path diagram:



3x3 'a' matrix

	1	2	3
1	a_r1c1	a_r1c2	a_r1c3
2	a_r2c1	a_r2c2	a_r2c3
3	a_r3c1	a_r3c2	a_r3c3

Data Input The function flexibly accepts raw data, and also summary covariance data (in which case the user must also supple numbers of observations for the two input data sets).

Ordinal Data In an important capability, the model transparently handles ordinal (binary or multi-level ordered factor data) inputs, and can handle mixtures of continuous, binary, and ordinal data in any combination. An experimental feature is under development to allow Tobit modelling.

The function also supports weighting of individual data rows. In this case, the model is estimated for each row individually, then each row likelihood is multiplied by its weight, and these weighted likelyhoods summed to form the model-likelihood, which is to be minimised. This feature is used in the non-linear GxE model functions.

Additional features The umxACE function supports varying the DZ genetic association (defaulting to .5) to allow exploring assortative mating effects, as well as varying the DZ "C" factor from 1 (the default for modelling family-level effects shared 100 to .25 to model dominance effects.

note: Only one of C or D may be estimated simultaneously. This restriction reflects the lack of degrees of freedom to simultaneously model C and D with only MZ and DZ twin pairs ref?.

Value

- `mxModel` of subclass mxModel.ACE

References

- http://www.github.com/tbates/umx

See Also

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
# ============================
# = How heritable is height? =
# ============================
require(umx)
data(twinData) # ?twinData from Australian twins.
# Pick the variables
selDVs = c("ht1", "ht2")
mzData <- twinData[twinData$zygosity %in% "MZFF", ]
dzData <- twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData) # -2ll= 9659
umxSummary(m1, std = FALSE) # unstandardized
# tip: with report = "html", umxSummary can print the table to your browser!
plot(m1)

# ========================================================
# = Evidence for dominance ? (DZ correlation set to .25) =
# ========================================================
m2 = umxACE("ADE", selDVs = selDVs, dzData = dzData, mzData = mzData, dzCr = .25)
umxCompare(m2, m1) # ADE is better
umxSummary(m2, comparison = m1) # nb: though this is ADE, columns are labeled ACE

# ==============================
# = Univariate model of weight =
# ==============================

# Things to note:
# 1. This variable has a large variance, but umx picks good starts.
# 2. umxACE can figure out variable names: provide "sep" and "wt" -> "wt1" "wt2"
# 3. umxACE picks the variables it needs from the data.
# 4. You can use boundDiag to lbound a, c, and e at 0 (prevents mirror-solutions).
m1 = umxACE(selDVs = "wt", dzData = dzData, mzData = mzData, sep = "", boundDiag = 0)
# We can modify this model, dropping shared environment, and see a comparison
m2 = umxModify(m1, update = "c_r1c1", comparison = TRUE)
# ====================================
# = Bivariate height and weight model =
# ====================================
data(twinData)
mzData <- twinData[twinData$zygosity %in% c("MZFF", "MZMM"),]
dzData <- twinData[twinData$zygosity %in% c("DZFF", "DZMM", "DZOS"), ]
mzData <- mzData[1:80,] # quicker run to keep CRAN happy
dzData <- dzData[1:80,]
selDVs = c("ht", "wt") # umx will add suffix (in this case "") + "1" or '2'
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = '')
umxSummary(m1)
```

```
# ========================================================
# = Well done! Now you can make modify twin models in umx =
# ========================================================


# ===================
# = Ordinal example =
# ===================
require(umx)
data(twinData)
# Cut bmi column to form ordinal obesity variables
ordDVs = c("obese1", "obese2")
selDVs = c("obese")
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints <- quantile(twinData[, "bmi1"], probs = c(.5, .2), na.rm = TRUE)
twinData$obese1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obese2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
twinData[, ordDVs] <- mxFactor(twinData[, ordDVs], levels = obesityLevels)
mzData <- twinData[twinData$zyg == 1, umx_paste_names(selDVs, "", 1:2)]
dzData <- twinData[twinData$zyg == 3, umx_paste_names(selDVs, "", 1:2)]
mzData <- mzData[1:80,] # just top 80 pairs to run fast
dzData <- dzData[1:80,]
str(mzData) # make sure mz, dz, and t1 and t2 have the same levels!
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = '')
umxSummary(m1)

# ============================================
# = Bivariate continuous and ordinal example =
# ============================================
data(twinData)
selDVs = c("wt", "obese")
# Cut bmi column to form ordinal obesity variables
ordDVs = c("obese1", "obese2")
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints <- quantile(twinData[, "bmi1"], probs = c(.5, .2), na.rm = TRUE)
twinData$obese1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obese2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
twinData[, ordDVs] <- mxFactor(twinData[, ordDVs], levels = obesityLevels)
mzData <- twinData[twinData$zyg == 1,] # umxACE can trim out unused variables on its own
dzData <- twinData[twinData$zyg == 3,]
mzData <- mzData[1:80,] # just top 80 so example runs in a couple of secs
dzData <- dzData[1:80,]
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = '')
plot(m1)

# ======================================
# = Mixed continuous and binary example =
# ======================================
require(umx)
data(twinData)
```

```
# Cut to form category of 20% obese subjects
# and make into mxFactors (ensure ordered is TRUE, and require levels)
cutPoints <- quantile(twinData[, "bmi1"], probs = .2, na.rm = TRUE)
obesityLevels = c('normal', 'obese')
twinData$obese1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obese2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
ordDVs = c("obese1", "obese2")
twinData[, ordDVs] <- mxFactor(twinData[, ordDVs], levels = obesityLevels)

selDVs = c("wt", "obese")
mzData <- twinData[twinData$zygosity %in% "MZFF", umx_paste_names(selDVs, "", 1:2)]
dzData <- twinData[twinData$zygosity %in% "DZFF", umx_paste_names(selDVs, "", 1:2)]
## Not run:
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = '')
umxSummary(m1)

## End(Not run)

# ==================================
# Example with covariance data only =
# ==================================

require(umx)
data(twinData)
selDVs = c("wt1", "wt2")
mz = cov(twinData[twinData$zyg == 1, selDVs], use = "complete")
dz = cov(twinData[twinData$zyg == 3, selDVs], use = "complete")
m1 = umxACE(selDVs = selDVs, dzData = dz, mzData = mz, numObsDZ=569, numObsMZ=351)
umxSummary(m1)
plot(m1)
```

---

umxACEcov *umxACEcov: Build and run a Cholesky with covariates in the covariance*

Description

Make a 2-group ACE Cholesky Twin model with covariates modeled (see Details below)

Usage

```
umxACEcov(name = "ACEcov", selDVs, selCovs, dzData, mzData, suffix = NULL,
  dzAr = 0.5, dzCr = 1, addStd = TRUE, addCI = TRUE, boundDiag = NULL,
  equateMeans = TRUE, bVector = FALSE, thresholds = c("deviationBased",
  "left_censored"), autoRun = getOption("umx_auto_run"))
```

Arguments

name	The name of the model (defaults to"ACE")

selDVs	The variables to include from the data (do not include suffixes)
selCovs	The covariates to include from the data (do not include suffixes)
dzData	The DZ dataframe
mzData	The MZ dataframe
suffix	suffix for twin 1 and twin 2, often "_T" Used to expand selDVs into full column names, i,e "dep" –> c("dep_T1", "dep_T2")
dzAr	The DZ genetic correlation (defaults to .5, vary to examine assortative mating)
dzCr	The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model)
addStd	Whether to add the algebras to compute a std model (defaults to TRUE)
addCI	Whether to add intervals to compute CIs (defaults to TRUE)
boundDiag	= Whether to bound the diagonal of the a, c, and e matrices
equateMeans	Whether to equate the means across twins (defaults to TRUE)
bVector	Whether to compute row-wise likelihoods (defaults to FALSE)
thresholds	How to implement ordinal thresholds: c("deviationBased", "left_censored")
autoRun	Whether to run the model and return it, or just return it

Details

umxACEcov supplements the umxACE Cholesky model with covariates.

Value

- mxModel of subclass mxModel.ACEcov

References

- Neale, M. C., & Martin, N. G. (1989). The effects of age, sex, and genotype on self-report drunkenness following a challenge dose of alcohol. Behavior Genetics, 19, 63-78. doi:10.1007/BF01065884

Schwabe, I., Boomsma, D. I., Zeeuw, E. L., & Berg, S. M. (2015). A New Approach to Handle Missing Covariate Data in Twin Research : With an Application to Educational Achievement Data. Behavior Genetics. doi:10.1007/s10519-015-9771-1

See Also

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
# replicate age to age1 & age2
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi") # Set the DV
selCovs = c("age") # Set the IV
```

```
selVars = umx_paste_names(selDVs, covNames = selCovs, sep = "", suffixes = 1:2)
# 80 rows so example runs fast
mzData = subset(twinData, zygosity == "MZFF", selVars)[1:80, ]
dzData = subset(twinData, zygosity == "DZFF", selVars)[1:80, ]
m1 = umxACEcov(selDVs = selDVs, selCovs = selCovs,
   dzData = dzData, mzData = mzData, suffix = "", autoRun = TRUE
)
umxSummary(m1)
plot(m1)
# ====================
# = A bivariate test =
# ====================
selDVs  = c("ht", "wt") # Set the DV
selCovs = c("age") # Set the IV
selVars = umx_paste_names(selDVs, covNames = selCovs, sep = "", suffixes = 1:2)
# 80 rows so example runs fast enough on CRAN
mzData = subset(twinData, zygosity == "MZFF", selVars)[1:80, ]
dzData = subset(twinData, zygosity == "DZFF", selVars)[1:80, ]
m1 = umxACEcov(selDVs = selDVs, selCovs = selCovs,
   dzData = dzData, mzData = mzData, suffix = "", autoRun = TRUE
)


# Univariate bmi without covariate of age for comparison
m2     = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix="")
x      = umx_residualize("bmi", "age", suffixes=1:2, twinData)
mzData = subset(x, zygosity == "MZFF", selVars)[1:80, ]
dzData = subset(x, zygosity == "DZFF", selVars)[1:80, ]
m3     = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix="")
```

---

umxACESexLim *umxACESexLim: Build and run a sex-limitaiton twin model (not working yet)*

Description

Cholesky style sex-limitation model.

Usage

```
umxACESexLim(name = "ACE_sexlim", selDVs, mzmData, dzmData, mzfData, dzfData,
  dzoData, suffix = NULL, autoRun = getOption("umx_auto_run"))
```

Arguments

name	The name of the model (defaults to "ACE_sexlim")
selDVs	The variables to include. If you provide a suffix, you can use just the base names.

mzmData	The MZ male dataframe
dzmData	The DZ male dataframe
mzfData	The DZ female dataframe
dzfData	The DZ female dataframe
dzoData	The DZ opposite-sex dataframe. (be sure and get in right order)
suffix	The suffix for twin 1 and twin 2, often "_T". If set, you can omit suffixes in SelDVs, i.e., just "dep" not c("dep_T1", "dep_T2")
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)

Details

This is a multi-variate capable Quantitative & Qualitative Sex-Limitation script using ACE Cholesky modeling. It implements a correlation approach to ensure that order of variables does NOT affect ability of model to account for DZOS data.

Restrictions include the assumtion that twin means and variances can be equated across birth order within zygosity groups.

Note: Qualitative sex differences are differences in the latent A, C, or E latent variables Note: Quantitative sex differences are differences in the path loadings from A, C, or E to the measured variables

Value

- ACE sexlim model

References

- Neale et al., (2006). Multivariate genetic analysis of sex-lim and GxE interaction, Twin Research & Human Genetics., https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Twin Modeling Functions: plot.MxModel, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
## Not run:
# Load Libraries
require(umx);
# ========================
# = Load and Process Data =
# ========================
data('us_skinfold_data')
# rescale vars
us_skinfold_data[,c('bic_T1', 'bic_T2')] <- us_skinfold_data[,c('bic_T1', 'bic_T2')]/3.4
us_skinfold_data[,c('tri_T1', 'tri_T2')] <- us_skinfold_data[,c('tri_T1', 'tri_T2')]/3
```

```
us_skinfold_data[,c('caf_T1', 'caf_T2')] <- us_skinfold_data[,c('caf_T1', 'caf_T2')]/3
us_skinfold_data[,c('ssc_T1', 'ssc_T2')] <- us_skinfold_data[,c('ssc_T1', 'ssc_T2')]/5
us_skinfold_data[,c('sil_T1', 'sil_T2')] <- us_skinfold_data[,c('sil_T1', 'sil_T2')]/5

# Select Variables for Analysis
varList = c('ssc','sil','caf','tri','bic')
selVars = umx_paste_names(varList, "_T", 1:2)

# Data objects for Multiple Groups
mzmData = subset(us_skinfold_data, zyg == 1, selVars)
dzmData = subset(us_skinfold_data, zyg == 3, selVars)
mzfData = subset(us_skinfold_data, zyg == 2, selVars)
dzfData = subset(us_skinfold_data, zyg == 4, selVars)
dzoData = subset(us_skinfold_data, zyg == 5, selVars)

m1 = umxACESexLim(selDVs = varList, suffix = "_T",
       mzmData = mzmData, dzmData = dzmData,
       mzfData = mzfData, dzfData = dzfData,
       dzoData = dzoData)
m1 = mxRun(m1)
# ================================================
# = Test switching specific a from Males to females =
# ================================================
m2 = umxSetParameters(m1, labels = "asm_.*", free = FALSE, values = 0, regex = TRUE)
m2 = umxSetParameters(m1, labels = "asf_.*", free = TRUE , values = 0, regex = TRUE)
m2 = mxRun(m2)
summary(m2)
umxCompare(m2, m1)
# does fit move on repeated execution?
# for (i in 1:4) { m2 <- mxRun(m2); print(m2 $output$mi) }

## End(Not run)
```

---

umxAdd1 *umxAdd1*

Description

Add each of a set of paths you provide to the model, returning a table of theire effect on fit

Usage

```
umxAdd1(model, pathList1 = NULL, pathList2 = NULL, arrows = 2, maxP = 1)
```

Arguments

model	an mxModel to alter
pathList1	a list of variables to generate a set of paths
pathList2	an optional second list: IF set paths will be from pathList1 to members of this list

arrows	Make paths with one or two arrows
maxP	The threshold for returning values (defaults to p==1 - all values)

Value

a table of fit changes

References

- http://www.github.com/tbates/umx

See Also

Other Modify or Compare Models: umxDrop1, umxEquate, umxFixAll, umxMI, umxSetParameters, umxUnexplainedCausalNexus, umx

Examples

```
## Not run:
model = umxAdd1(model)

## End(Not run)
```

---

umxAPA *umxAPA*

Description

This function creates object summaries used in reporting models, effects, and summarizing data.

1. Given an lm, will return a formated effect, including 95% CI in square brackets, for one of the effects (specified by name in se). e.g.: umxAPA(m1, "wt") yields:

β = -5.344 [-6.486, -4.203], p< 0.001

2. Given a dataframe, summaryAPA will return a table of correlations, with the mean and SD of each variable as the last row. So, `umxAPA(mtcars[,c("cyl", "wt", "mpg", )])` yields a table of correlations, means and SDs thus:

	cyl	wt	mpg
cyl	1	0.78	-0.85
wt	0.78	1	-0.87
mpg	-0.85	-0.87	1
mean_sd	6.19 (1.79)	3.22 (0.98)	20.09 (6.03)

3. Given obj and se, umxAPA returns a CI based on 1.96 times the se.

4. Given only a number as obj will be treated as a p-value as returned in APA format.

Usage

```
umxAPA(obj, se = NULL, std = FALSE, digits = 2, use = "complete",
  min = 0.001, addComparison = NA, report = c("table", "html"),
  lower = TRUE)
```

Arguments

obj	Either a model (lm), a beta-value, or a data.frame
se	If b is a model, then name of the parameter of interest, else the SE (standard-error)
std	If obj is an lm, whether to re-run the model on standardized data and report std betas
digits	Round numbers to how many values
use	If obj is a data.frame, how to handle NA (default = "complete")
min	= .001 for a p-value, the smallest value to report numerically
addComparison	for a p-value, whether to add "</=" default (NA) adds "<" if necessary
report	what to return (default = markdown table). Use "html" to open a web page table
lower	whether to report on the lower triangle of correlations for a data.frame (Default = TRUE)

Value

- string

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Reporting Functions: loadings.MxModel, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx_time, umx

Examples

```
# Generate a formatted string describing a regression:
umxAPA(lm(mpg ~ wt + disp, mtcars))
umxAPA(lm(mpg ~ wt + disp, mtcars), "disp")
# Generate a summary table of correlations + Mean and SD:
umxAPA(mtcars[,1:3])
umxAPA(mtcars[,1:3], digits = 3)
umxAPA(mtcars[,1:3], lower = FALSE)
umxAPA(mtcars[,1:3], report = "html")
# Generate a CI string based on effect and se
umxAPA(.4, .3)
# format p-value
umxAPA(.0182613)
umxAPA(.000182613)
```

umxCF_SexLim *umxCF_SexLim*

Description

Build a multivariate twin analysis with sex limitation based on a correlated factors model. This allows Quantitative & Qualitative Sex-Limitation. The correlation approach ensures that variable order does NOT affect ability of model to account for DZOS data. Restrictions: Assumes means and variances can be equated across birth order within zygosity groups

Usage

```
umxCF_SexLim(name = "ACE_sexlim", selDVs, mzmData, dzmData, mzfData, dzfData,
  dzoData, C_or_A = "A", suffix = NA)
```

Arguments

name	The name of the model (Default = "CF_sexlim")
selDVs	BASE NAMES of the variables in the analysis. You MUST provide suffixes.
mzmData	Dataframe containing the MZ male data
dzmData	Dataframe containing the DZ male data
mzfData	Dataframe containing the MZ female data
dzfData	Dataframe containing the DZ female data
dzoData	Dataframe containing the DZ opposite-sex data (be sure and get in right order)
C_or_A	Whether to model sex-limitation on C or on A. (Defaults to "A")
suffix	Suffix used for twin variable naming. Allows using just the base names in selVars

Value

- CF SexLim model

References

- Neale et al. (2006). Multivariate genetic analysis of sex-lim and GxE interaction. *Twin Research & Human Genetics*, **9**, pp. 481–489.

See Also

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
## Not run:
# Load Libraries
require(umx)
# Create Functions to Assign Labels
laLower <- function(la,nVar) {
paste(la,rev(nVar+1-sequence(1:nVar)),rep(1:nVar,nVar:1),sep="_")
}
laSdiag <- function(la,nVar) {
paste(la,rev(nVar+1-sequence(1:(nVar-1))),rep(1:(nVar-1),(nVar-1):1),sep="_")
}
laFull  <- function(la,nVar) {
paste(la,1:nVar,rep(1:nVar,each=nVar),sep="_")
}
laDiag  <- function(la,nVar) {
paste(la,1:nVar,1:nVar,sep="_")
}
laSymm  <- function(la,nVar) {
paste(la,rev(nVar+1-sequence(1:nVar)),rep(1:nVar,nVar:1),sep="_")
}
# =========================
# = Load and Process Data =
# =========================
data("us_skinfold_data")
# rescale vars
us_skinfold_data[,c('bic_T1', 'bic_T2')] <- us_skinfold_data[,c('bic_T1', 'bic_T2')]/3.4
us_skinfold_data[,c('tri_T1', 'tri_T2')] <- us_skinfold_data[,c('tri_T1', 'tri_T2')]/3
us_skinfold_data[,c('caf_T1', 'caf_T2')] <- us_skinfold_data[,c('caf_T1', 'caf_T2')]/3
us_skinfold_data[,c('ssc_T1', 'ssc_T2')] <- us_skinfold_data[,c('ssc_T1', 'ssc_T2')]/5
us_skinfold_data[,c('sil_T1', 'sil_T2')] <- us_skinfold_data[,c('sil_T1', 'sil_T2')]/5
# describe(us_skinfold_data, skew = FALSE)

# Select Variables for Analysis
varList = c('ssc','sil','caf','tri','bic')
selVars = umx_paste_names(varList, "_T", 1:2)
nVar = length(selVars)

# Data objects for Multiple Groups
mzmData = subset(us_skinfold_data, zyg == 1, selVars)
dzmData = subset(us_skinfold_data, zyg == 3, selVars)
mzfData = subset(us_skinfold_data, zyg == 2, selVars)
dzfData = subset(us_skinfold_data, zyg == 4, selVars)
dzoData = subset(us_skinfold_data, zyg == 5, selVars)

m1 = umxACESexLim(selDVs = varList, suffix = "_T",
       mzmData = mzmData, dzmData = dzmData,
       mzfData = mzfData, dzfData = dzfData,
       dzoData = dzoData)
m1 = mxRun(m1)
summary(m1)

# ==============================
```

```
# = 1 Nonscalar Sex Limitation  =
# ================================
# Quantitative Sex Differences & Qualitative Sex Differences for A
# Male and female paths, plus male and female Ra, Rc and Re between variables
# Male-Female correlations in DZO group between
# A factors Rao FREE, Rc constrained across male/female and opp-sex
# ================================================
# = Test switching specific a from Males to females =
# ================================================

m2 = umxSetParameters(m1, labels = "asm_.*", free = FALSE, values = 0, regex = TRUE)
m2 = umxSetParameters(m1, labels = "asf_.*", free = TRUE , values = 0, regex = TRUE)
m2 = mxRun(m2)
summary(m2)
mxCompare(m2, m1)
# ================================
# = 2 Nonscalar Sex Limitation  =
# ================================
# Quantitative Sex Differences & Qualitative Sex Differences for C
# Male and female paths, plus male and female Ra, Rc and Re between variables
# Male-Female correlations in DZO group between C
# factors Rco FREE, Ra constrained across male/female and oppsex

# -------|---------|---------|---------|---------|---------|---------|---------|---------|-----|
# 3 Scalar Sex Limitation
# Quantitative Sex Differences but NO Qualitative Sex Differences
# Male and female paths, but one set of Ra, Rc and Re between variables (same for male & female)
# ---------------------------------------------------------------------------------------------|

# ================================
# ================================
frODiag   <- c(rep(c(FALSE,rep(TRUE,nVar)),nVar-1),FALSE)
svODiag   <- c(rep(c(1,rep(.4,nVar)),nVar-1),1)
m3 = umxSetParameters(m2, labels = "asm_.*", free = FALSE, values = 0, regex = TRUE)
pathRam = mxMatrix(name="Ram", "Stand", nrow= nVar, free = TRUE, values = .4,
label = laSdiag("ra", nVar), lbound = -1, ubound = 1)
pathRaf = mxMatrix(name = "Raf", "Stand", nrow = nVar, free = TRUE, values = .4,
label=laSdiag("ra", nVar), lbound = -1, ubound = 1)
pathRcm = mxMatrix(name="Rcm", "Stand", nrow = nVar, free = TRUE, values = .4,
label=laSdiag("rc", nVar), lbound = -1, ubound = 1)
pathRcf = mxMatrix(name="Rcf", "Stand", nrow = nVar, free = TRUE, values = .4,
label=laSdiag("rc", nVar), lbound = -1, ubound = 1)
pathRem = mxMatrix(name="Rem", "Stand", nrow= nVar, free=TRUE, values = .4,
label = laSdiag("re", nVar), lbound = -1, ubound = 1)
pathRef = mxMatrix(name="Ref", "Stand", nrow= nVar, free=TRUE, values = .4,
label = laSdiag("re", nVar), lbound = -1, ubound = 1)
corRao  = mxMatrix(name="Rao", "Symm" , nrow= nVar, free = frODiag, values = svODiag,
         label = laSymm("ra", nVar), lbound = -1, ubound = 1)
corRco  = mxMatrix(name="Rco", "Symm" , nrow= nVar, free = frODiag, values = svODiag,
         label = laSymm("rc", nVar), lbound = -1, ubound = 1)

# m3 <- makeModel("HetCfAce")
# m3 <- mxRun(m3)
```

```
# summary(m3)
# round(m3$VarsZm$result,4); round(m3$CorsZm$result,4)
# round(m3$VarsZf$result,4); round(m3$CorsZf$result,4)
# mxCompare(HetCfAceRgFit, m3)

# ===================
# = 4 Homogeneity
# = NO Quantitative Sex Differences AND NO Qualitative Sex Differences
# = Same paths for males and females
# ===================

# ====================================
# = Equate [ace]m and [ace]f matrices =
# ====================================

pathAm = mxMatrix(name="am", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("a", nVar))
pathCm = mxMatrix(name="cm", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("c", nVar))
pathEm = mxMatrix(name="em", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("e", nVar))
pathAf = mxMatrix(name="af", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("a", nVar))
pathCf = mxMatrix(name="cf", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("c", nVar))
pathEf = mxMatrix(name="ef", "Diag", nrow = nVar, free = TRUE, values = .5,
label = laDiag("e", nVar))

# m4 <- makeModel("HomCfAce")
# m4 <- mxRun(m4)
# summary(m4)
# round(m4$VarsZm$result,4); round(m4$CorsZm$result,4)
# round(m4$VarsZf$result,4); round(m4$CorsZf$result,4)
# mxCompare(m3, m4)

# ============================================
# = Generate Output Table of all Nested Models =
# ============================================

# mxCompare(HetCfAceRgFit, c(HetCfAceRcFit, m3, m4))

# rbind(
#  mxCompare(HetCfAceRgFit, HetCfAceRcFit),
#   mxCompare(HetCfAceRcFit, m3)[2,],
#   mxCompare(m3, m4)[2,]
# )

## End(Not run)
```

---

umxCI *umxCI*

Description

umxCI adds mxCI() calls for all free parameters in a model, runs the CIs, and reports a neat summary.

Usage

```
umxCI(model = NULL, which = c("ALL", NA, "list of your making"),
  remove = FALSE, run = c("no", "yes", "if necessary"),
  showErrorCodes = TRUE)
```

Arguments

model	The mxModel you wish to report mxCIs on
which	What CIs to add: c("ALL", NA, "list of your making")
remove	= FALSE (if set, removes existing specified CIs from the model)
run	Whether or not to compute the CIs. Valid values = "no" (default), "yes", "if necessary".
showErrorCodes	Whether to show errors (default == TRUE)

Details

This function also reports any problems computing a CI. The codes are standard OpenMx errors and warnings

- 1: The final iterate satisfies the optimality conditions to the accuracy requested, but the sequence of iterates has not yet converged. NPSOL was terminated because no further improvement could be made in the merit function (Mx status GREEN)
- 2: The linear constraints and bounds could not be satisfied. The problem has no feasible solution.
- 3: The nonlinear constraints and bounds could not be satisfied. The problem may have no feasible solution.
- 4: The major iteration limit was reached (Mx status BLUE).
- 6: The model does not satisfy the first-order optimality conditions to the required accuracy, and no improved point for the merit function could be found during the final linesearch (Mx status RED)
- 7: The function derivates returned by funcon or funobj appear to be incorrect.
- 9: An input parameter was invalid

If runCIs is FALSE, the function simply adds CIs to be computed and returns the model.

Value

- mxModel

References

- http://www.github.com/tbates/umx/

See Also

- mxCI, umxLabel, umxRun

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
m1$intervals # none yet list()
m1 = umxCI(m1)
m1$intervals # $G_to_x1
m1 = umxCI(m1, remove = TRUE) # Add CIs for all free parameters, and return model
## Not run:
umxCI(model, run = "yes") # force update of CIs
# Don't force update of CIs, but if they were just added, then calculate them
umxCI(model, run = "if necessary")

## End(Not run)
```

---

umxCI_boot *umxCI_boot*

Description

Compute boot-strapped Confidence Intervals for parameters in an mxModel The function creates a sampling distribution for parameters by repeatedly drawing samples with replacement from your data and then computing the statistic for each redrawn sample.

Usage

```
umxCI_boot(model, rawData = NULL, type = c("par.expected", "par.observed",
  "empirical"), std = TRUE, rep = 1000, conf = 95, dat = FALSE,
  digits = 3)
```

Arguments

model	is an optimized mxModel
rawData	is the raw data matrix used to estimate model
type	is the kind of bootstrap you want to run. "par.expected" and "par.observed" use parametric Monte Carlo bootstrapping based on your expected and observed covariance matrices, respectively. "empirical" uses empirical bootstrapping based on rawData.
std	specifies whether you want CIs for unstandardized or standardized parameters (default: std = TRUE)
rep	is the number of bootstrap samples to compute (default = 1000).
conf	is the confidence value (default = 95)
dat	specifies whether you want to store the bootstrapped data in the output (useful for multiple analyses, such as mediation analysis)
digits	rounding precision

Value

- expected covariance matrix

References

- http://openmx.ssri.psu.edu/thread/2598 Original written by http://openmx.ssri.psu.edu/users/bwiernik

See Also

- umxExpMeans, umxExpCov

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
```

```
umxCI_boot(m1, type = "par.expected")

## End(Not run)
```

---

umxCompare *umxCompare*

Description

umxCompare compares two or more mxModels. It has several nice features: 1. It supports direct control of rounding, and reports p-values rounded to APA style. 2. It reports the table in your preferred markdown format (relies on knitr) 3. The table columns are arranged in a method suitable for easy comparison for readers. 4. By default, it also reports the output as an ENglish sentence suitable for a paper. 5. It can open tabular output in a browser (report = "html")

Usage

```
umxCompare(base = NULL, comparison = NULL, all = TRUE, digits = 3,
  report = c("2", "1", "html"), file = "tmp.html")
```

Arguments

base	The base mxModel for comparison
comparison	The model (or list of models) which will be compared for fit with the base model (can be empty)
all	Whether to make all possible comparisons if there is more than one base model (defaults to T)
digits	rounding for p etc.
report	Optionally (report = "2") add sentences for inclusion in a paper, or ("html") create a web table and open your default browser. (handy for getting tables into Word, and other text systems!)
file	file to write html too if report=3 (defaults to "tmp.html")

Details

note: If you leave comparison blank, it will just give fit info for the base model

References

- http://www.github.com/tbates/umx/

See Also

- mxCompare, umxSummary, umxRAM,

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
m2 = umxModify(m1, update = "G_to_x2", name = "drop_path_2_x2")
umxCompare(m1, m2)
mxCompare(m1, m2) # what OpenMx gives by default
umxCompare(m1, m2, report = "2") # Add English-sentence descriptions
## Not run:
umxCompare(m1, m2, report = "html") # Open table in browser

## End(Not run)
m3 = umxModify(m2, update = "G_to_x3", name = "drop_path_2_x2_and_3")
umxCompare(m1, c(m2, m3))
umxCompare(c(m1, m2), c(m2, m3), all = TRUE)
```

---

umxCov2cor *umxCov2cor*

Description

Version of cov2cor that forces upper and lower triangles to be identical (rather than nearly identical)

Usage

```
umxCov2cor(x)
```

Arguments

x	something that cov2cor can work on (matrix, df, etc.)

Value

- a correlation matrix

References

- http://www.github.com/tbates/umx

See Also

Other Stats Functions: reliability, umx_cor, umx_means, umx

Examples

```
umxCov2cor(cov(mtcars))
```

---

umxCovData *umxCovData*

Description

convert a dataframe in an mxData object of type "cov"

Usage

```
umxCovData(df, columns = NA, use = c("complete.obs", "everything",
  "all.obs", "na.or.complete", "pairwise.complete.obs"))
```

Arguments

df	the dataframe to covert to a covariance matrix
columns	= manifests
use	= Default is "complete.obs"

Value

- mxModel

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Data Functions: umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
umxCovData(mtcars, c("mpg", "hp"))
```

umxCP *umxCP: Build and run a Common pathway twin model*

Description

Make a 2-group Common Pathway twin model (Common-factor common-pathway multivariate model).

Usage

```
umxCP(name = "CP", selDVs, dzData, mzData, suffix = NULL, nFac = 1,
  freeLowerA = FALSE, freeLowerC = FALSE, freeLowerE = FALSE,
  correlatedA = FALSE, equateMeans = T, dzAr = 0.5, dzCr = 1,
  addStd = T, addCI = T, numObsDZ = NULL, numObsMZ = NULL,
  autoRun = getOption("umx_auto_run"), sep = NULL)
```

Arguments

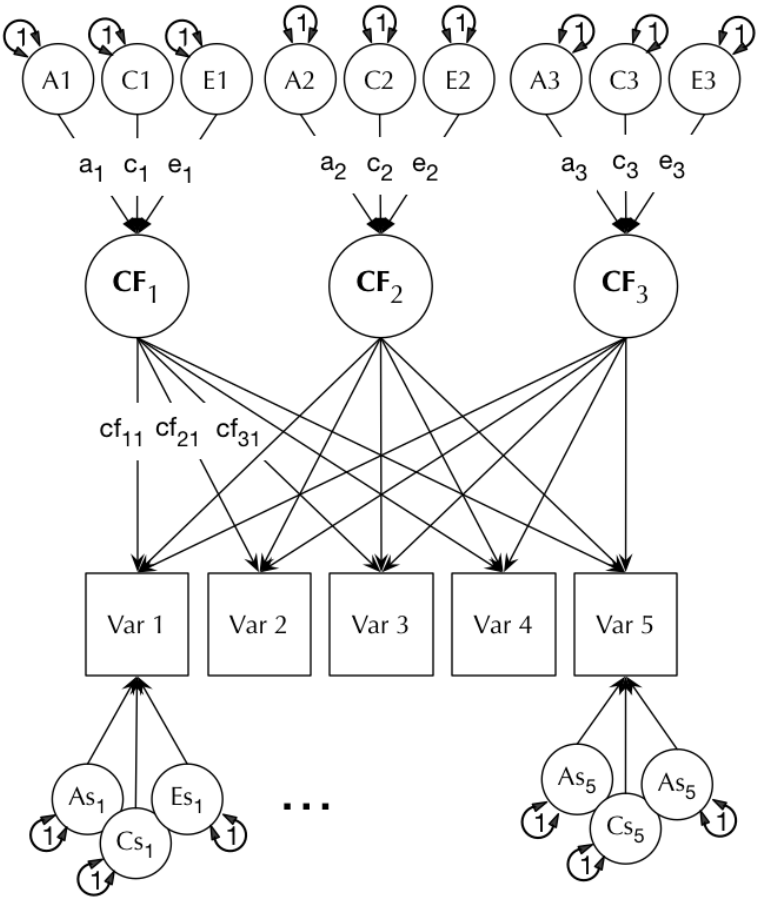
name	The name of the model (defaults to "CP")
selDVs	The variables to include
dzData	The DZ dataframe
mzData	The MZ dataframe
suffix	The suffix for twin 1 and twin 2, often "_T". If set, you can omit suffixes in SelDVs, i.e., just "dep" not c("dep_T1", "dep_T2")
nFac	How many common factors (default = 1)
freeLowerA	Whether to leave the lower triangle of A free (default = F)
freeLowerC	Whether to leave the lower triangle of C free (default = F)
freeLowerE	Whether to leave the lower triangle of E free (default = F)
correlatedA	?? (default = FALSE)
equateMeans	Whether to equate the means across twins (defaults to T)
dzAr	The DZ genetic correlation (defaults to .5, vary to examine assortative mating)
dzCr	The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model)
addStd	Whether to add the algebras to compute a std model (defaults to TRUE)
addCI	Whether to add the interval requests for CIs (defaults to TRUE)
numObsDZ	= not yet implemented: Ordinal Number of DZ twins: Set this if you input covariance data
numObsMZ	= not yet implemented: Ordinal Number of MZ twins: Set this if you input covariance data
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)
sep	allowed as a synonym for "suffix"

Details

The common-pathway model provides a powerful tool for theory-based decomposition of genetic and environmental differences.

umxCP supports this with pairs of mono-zygotic (MZ) and di-zygotic (DZ) twins reared together to model the genetic and environmental structure of multiple phenotypes (measured behaviors).

Common-pathway path diagram:



As can be seen, each phenotype also by default has A, C, and E influences specific to that phenotye.

Like the `umxACE` model, the CP model decomposes phenotypic variance into Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

Unlike the Cholesky, these factors do not act directly on the phenotype. Instead latent A, C, and E influences impact on one or more latent factors which in turn account for variance in the phenotypes (see Figure below).

Data Input Currently, the umxCP function accepts only raw data. This may change in future versions.

Ordinal Data In an important capability, the model transparently handles ordinal (binary or multi-level ordered factor data) inputs, and can handle mixtures of continuous, binary, and ordinal data in

any combination.

Additional features The umxCP function supports varying the DZ genetic association (defaulting to .5) to allow exploring assortative mating effects, as well as varying the DZ "C" factor from 1 (the default for modelling family-level effects shared 100 to .25 to model dominance effects.

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

- umxACE() for more examples of twin modeling, plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
zygList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = zygList)
twinData$wt1 = twinData$wt1/10 # help CSOLNP by putting wt on a similar scale to ht
twinData$wt2 = twinData$wt2/10 # help CSOLNP by putting wt on a similar scale to ht
selDVs = c("ht", "wt")
mzData <- subset(twinData, ZYG == "MZFF", umx_paste_names(selDVs, "", 1:2))
dzData <- subset(twinData, ZYG == "DZFF", umx_paste_names(selDVs, "", 1:2))
umx_set_optimizer("SLSQP") #preferably NPSOL: CSOLNP needs setup to run this model.
m1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = "")
umxSummary(m1)
umxGetParameters(m1, "^c", free = TRUE)
m2 = umxModify(m1, update = "(cs_.*$)|(c_cp_)", regex = TRUE, name = "dropC")
umxSummaryCP(m2, comparison = m1, file = NA)
umxCompare(m1, m2)
```

---

umxDiagnose *mxDiagnostic*

Description

Diagnose problems in a model - this is a work in progress.

Usage

```
umxDiagnose(model, tryHard = FALSE, diagonalizeExpCov = FALSE)
```

Arguments

model	an mxModel to diagnose
tryHard	whether I should try and fix it? (defaults to FALSE)
diagonalizeExpCov	Whether to diagonalize the ExpCov

Value

- helpful messages and perhaps a modified model

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Core Modelling Functions: plot.MxModel, umxLatent, umxMatrix, umxPath, umxRAM, umxReduce, umxRun, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
m1 <- umxRAM("OneFactor", data = myData,
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
m1 = mxRun(m1)
umxSummary(m1, show = "std")
umxDiagnose(m1)
```

---

umxDrop1 *umxDrop1: Unfinished function to mimic drop1 in OpenMx*

Description

Drops each free parameter (selected via regex), returning an mxCompare table comparing the effects. A great way to quickly determine which of several parameters can be dropped without excessive cost

Usage

```
umxDrop1(model, regex = NULL, maxP = 1)
```

Arguments

model	An mxModel to drop parameters from
regex	A string to select parameters to drop. leave empty to try all. This is regular expression enabled. i.e., "^a_" will drop parameters beginning with "a_"
maxP	The threshold for returning values (defaults to p==1 - all values)

Value

a table of model comparisons

References

- http://www.github.com/tbates/umx

See Also

Other Modify or Compare Models: umxAdd1, umxEquate, umxFixAll, umxMI, umxSetParameters, umxUnexplainedCausalNexus, umx

Examples

```
## Not run:
umxDrop1(fit3) # try dropping each free parameters (default)
# drop "a_r1c1" and "a_r1c2" and see which matters more.
umxDrop1(model, regex="a_r1c1|a_r1c2")

## End(Not run)
```

---

umxEFA *umxEFA*

Description

Perform full-information maximum-likelihood factor analysis on a data matrix. as in factanal, you need only specify the number of factors and offer up some manifest data, e.g:

umxEFA(factors = 2, data = mtcars)

Usage

```
umxEFA(x = NULL, factors = NULL, data = NULL, covmat = NULL,
  n.obs = NULL, scores = c("none", "ML", "WeightedML", "Regression"),
  rotation = c("varimax", "promax", "none"), name = "efa", digits = 2,
  report = c("1", "table", "html"))
```

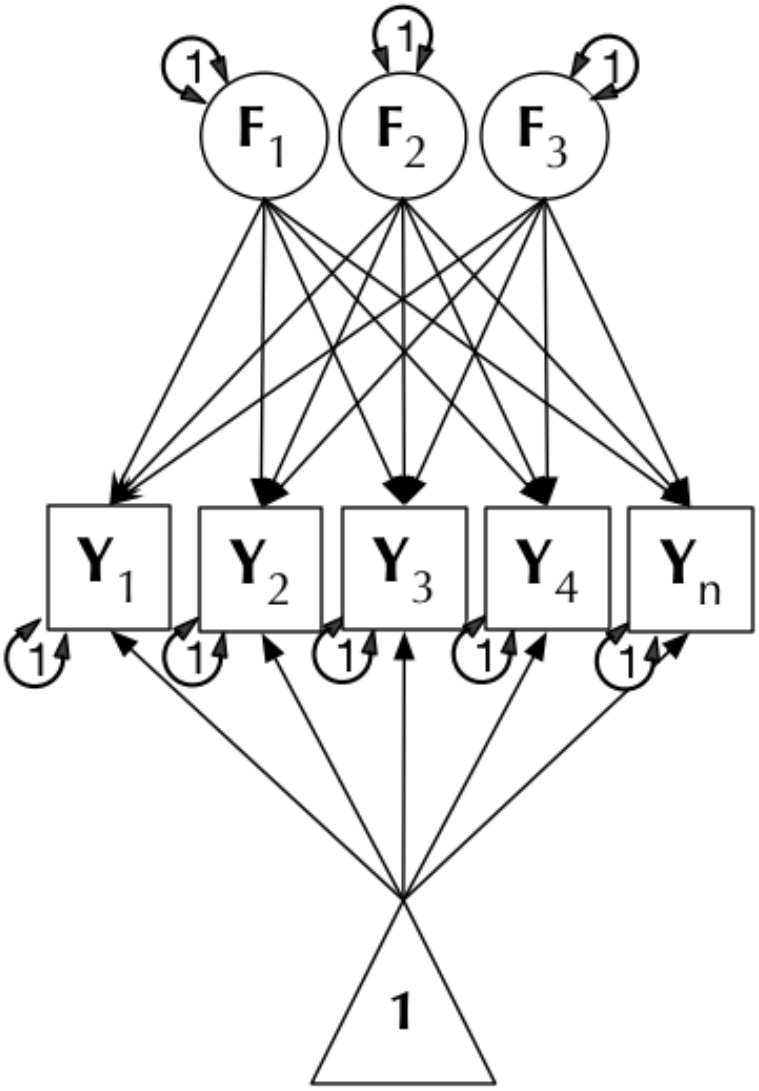
Arguments

x	Either 1: data, 2: A formula (not implemented yet), 3: A collection of variable names, or 4: A name for the model.
factors	Either number of factors to request or a vector of factor names.
data	A dataframe of manifest columns you are modeling
covmat	Covariance matrix of data you are modeling (not implemented)
n.obs	Number of observations in covmat (if provided, default = NA)
scores	Type of scores to produce, if any. The default is none, "Regression" gives Thompson's scores. Other options are 'ML', 'WeightedML', Partial matching allows these names to be abbreviated.
rotation	A rotation to perform on the loadings (default = "varimax" (orthogonal))
name	A name for your model
digits	rounding (default = 2)
report	What to report

Details

Equivalently, you can also give a list of factor names:

umxEFA(factors = c("g", "v"), data = mtcars)



notes: In an EFA, all items may load on all factors. For identification we need m^2 degrees of freedom. We get m * (m+1)/2 from fixing factor variances to 1 and covariances to 0. We get another m(m-1)/2 degrees of freemdom by fixing the upper-right hand corner of the factor loadings component of the A matrix. The manifest variances are also lbounded at 0.

EFA reports standardized loadings: to do this, we scale the data.

Bear in mind that factor scores are indeterminate and can be rotated.

This is very much early days. Adding "scores" in response to demand.

todo: detect ordinal items and switch to UWLS

Value

- EFA `mxModel`

References

- http://github.com/tbates/umx

See Also

- factanal, mxFactorScores

Other Super-easy helpers: umxTwoStage, umx

Examples

```
myVars <- c("mpg", "disp", "hp", "wt", "qsec")
m1 = umxEFA(mtcars[, myVars], factors =   2, rotation = "promax")
loadings(m1)
m2 = factanal(~ mpg + disp + hp + wt + qsec, factors = 2, rotation = "promax", data = mtcars)
loadings(m2)
## Not run:
plot(m2)
m1 = umxEFA(myVars, factors = 2, data = mtcars, rotation = "promax")
m1 = umxEFA(name = "named", factors = "g", data = mtcars[, myVars])
m1 = umxEFA(name = "by_number", factors = 2, rotation = "promax", data = mtcars[, myVars])
m1 = umxEFA(name = "score", factors = "g", data = mtcars[, myVars], scores= "Regression")

## End(Not run)
```

---

umxEquate *umxEquate: Equate two or more paths*

Description

In addition to dropping or adding parameters, a second common task in modelling is to equate parameters. umx provides a convenience function to equate parameters by setting one or more parameters (the "slave" set) equal to one or more "master" parameters. These parameters are picked out via their labels, and setting two or more parameters to have the same value is accomplished by setting the slave(s) to have the same label(s) as the master parameters, thus constraining them to take the same value during model fitting.

Usage

```
umxEquate(model, master, slave, free = c(TRUE, FALSE, NA), verbose = TRUE,
  name = NULL, autoRun = getOption("umx_auto_run"), comparison = TRUE)
```

Arguments

model	An mxModel within which to equate parameters
master	A list of "master" labels to which slave labels will be equated
slave	A list of slave labels which will be updated to match master labels, thus equating the parameters

free	Should parameter(s) initally be free? (default = TRUE)
verbose	Whether to give verbose feedback (default = TRUE)
name	name for the returned model (optional: Leave empty to leave name unchanged)
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)
comparison	Compare the new model to the old (if updating an existing model: default = TRUE)

Details

note: In addition to using this method to equating parameters, you can also equate one parameter to another by setting its label to the "square bracket" address of the master, e.g. "a[r,c]".

Tip: To find labels of free parameters use umxGetParameters with free = T Tip: To find labels by name, use the regex parameter of umxGetParameters

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

Other Modify or Compare Models: umxAdd1, umxDrop1, umxFixAll, umxMI, umxSetParameters, umxUnexplainedCausalNexus, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
m2 = umxEquate(m1, master = "G_to_x1", slave = "G_to_x2", name = "Equate x1 and x2 loadings")
m2 = mxRun(m2) # have to run the model again...
umxCompare(m1, m2) # not good :-)
```

umxEval *umxEval*

Description

Takes an expression as a string, and evaluates it as an expression in model, optionally computing the result. # TODO Currently broken...

Usage

```
umxEval(expstring, model, compute = FALSE, show = FALSE)
```

Arguments

expstring	an expression string, i.e, "a + b"
model	an mxModel to evaluate in
compute	Whether to compute the result or not (default = FALSE)
show	Whether to show??? (default = FALSE)

Value

- an openmx algebra (formula)

References

- http://www.github.com/tbates/umx

See Also

Other Misc: umx_APA_model_CI, umx_add_variances, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
m1 = mxModel("fit",
mxMatrix("Full", nrow = 1, ncol = 1, free = TRUE, values = 1, name = "a"),
mxMatrix("Full", nrow = 1, ncol = 1, free = TRUE, values = 2, name = "b"),
mxAlgebra(a %*% b, name = "ab"),
mxConstraint(ab == 35, name = "maxHours"),
mxAlgebraObjective(algebra = "ab", numObs= NA, numStats = NA)
)
m1 = mxRun(m1)
mxEval(list(ab = ab), m1)
```

umxExpCov *Get the expected vcov matrix*

Description

Extract the expected covariance matrix from an mxModel

Usage

```
umxExpCov(object, latents = FALSE, manifests = TRUE, digits = NULL, ...)
```

Arguments

object	an mxModel to get the covariance matrix from
latents	Whether to select the latent variables (defaults to TRUE)
manifests	Whether to select the manifest variables (defaults to TRUE)
digits	precision of reporting. NULL (Default) = no rounding.
...	extra parameters (to match vcov)

Value

- expected covariance matrix

References

- http://openmx.ssri.psu.edu/thread/2598 Original written by http://openmx.ssri.psu.edu/users/bwiernik

See Also

- umxRun, umxCI_boot

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
```

```
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
vcov(m1)
umxExpCov(m1, digits = 3)
```

---

umxExpMeans *umxExpMean*

Description

Extract the expected means matrix from an mxModel

Usage

```
umxExpMeans(model, manifests = TRUE, latents = NULL, digits = NULL)
```

Arguments

model	an mxModel to get the means from
manifests	Whether to select the manifest variables (defaults to TRUE)
latents	Whether to select the latent variables (defaults to TRUE)
digits	precision of reporting. Default (NULL) will not round at all.

Value

- expected means

References

- http://openmx.ssri.psu.edu/thread/2598

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
```

```
mxPath(from = manifests, arrows = 2),
mxPath(from = "one", to = manifests),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(demoOneFactor[1:100,], type = "raw")
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umxExpMeans(model = m1)
umxExpMeans(m1, digits = 3)
```

---

umxFactor *umxFactor*

Description

A convenient version of mxFactor supporting the common case in which the factor levels are those in the variable.

Usage

```
umxFactor(x = character(), levels = NULL, labels = levels, exclude = NA,
  ordered = TRUE, collapse = FALSE, verbose = FALSE, sep = NA)
```

Arguments

x	A variable to recode as an mxFactor (see mxFactor)
levels	(default NULL). Like factor but UNLIKE mxFactor, unique values will be used if levels not specified.
labels	= levels (see mxFactor)
exclude	= NA (see mxFactor)
ordered	= TRUE By default return an ordered mxFactor
collapse	= FALSE (see mxFactor)
verbose	Whether to tell user about such things as coercing to factor
sep	If twin data are being used, the string that separates the base from twin index # will try and ensure factor levels same across all twins.

Value

- mxFactor

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umxFactanal

Other Data Functions: umxCovData, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
umxFactor(letters)
umxFactor(letters, verbose = TRUE) # report coercions
umxFactor(letters, ordered = FALSE) # non-ordered factor like factor(x)
# Dataframe example:
x = umx_factor(mtcars[,c("cyl", "am")], ordered = FALSE); str(x)
# =================
# = Twin example: =
# =================
data(twinData)
tmp = twinData[, c("bmi1", "bmi2")]
tmp$bmi1[tmp$bmi1 <= 22] = 22
tmp$bmi2[tmp$bmi2 <= 22] = 22
# remember to factor _before_ breaking into MZ and DZ groups
x = umxFactor(tmp, sep = ""); str(x)
xmu_check_levels_identical(x, "bmi", sep="")

# Simple example to check behavior
x = round(10 * rnorm(1000, mean = -.2))
y = round(5 * rnorm(1000))
x[x < 0] = 0; y[y < 0] = 0
jnk = umxFactor(x); str(jnk)
df  = data.frame(x = x, y = y)
jnk = umxFactor(df); str(jnk)
```

---

umxFitIndices *umxFitIndices*

Description

A list of fit indices. Originated in this thread: http://openmx.ssri.psu.edu/thread/765 note: This is not a full-fat fit reporter. It is not robust across multi-group designs, definition variables. It is primarily designed to add less-often reported fit indices for RAM models where reviewer 2 wants something other than CFA/TLI/RMSEA :-).

Usage

```
umxFitIndices(model, refModels = mxRefModels(model, run = TRUE))
```

Arguments

model	The mxModel for which you want fit indices.
refModels	Independence and saturated models. default mxRefModels(model, run = TRUE)

Details

Fit information reported includes: N, deviance, N.parms, Chi, df, p.Chi, Chi.df, AICchi, AICdev, BCCchi, BCCdev, BICchi, BICdev, CAICchi, CAICdev, RMSEA, SRMR, RMR, SMAR, MAR, SMAR.nodiag, MAR.nodiag, GFI, AGFI, PGFI, NFI, RFI, IFI, NNFI.TLI, CFI, PRATIO, PNFI, PCFI, NCP, ECVIchi, ECVIdev, MECVIchi, MECVIdev, MFI, GH

Want more? File a report at github

Value

Table of fit statistics

References

-

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor",
data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
umxFitIndices(m1)
# And with raw data
m1 <- umxRAM("m1", data = demoOneFactor,
umxPath(latents, to = manifests),
umxPath(v.m. = manifests),
umxPath(v1m0 = latents)
)
umxFitIndices(m1)
umxAPA(umxFitIndices(m1), digits = 3)
```

umxFixAll *umxFixAll: Fix all free parameters*

Description

Fix all free parameters in a model using omxGetParameters()

Usage

```
umxFixAll(model, name = "_fixed", run = FALSE, verbose = FALSE)
```

Arguments

model	an mxModel within which to fix free parameters
name	optional new name for the model. if you begin with a _ it will be made a suffix
run	whether to fix and re-run the model, or just return it (defaults to FALSE)
verbose	whether to mention how many paths were fixed (default is FALSE)

Value

- the fixed mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxMI, umxSetParameters, umxUnexplainedCausalNexus, umx

Examples

```
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("OneFactor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
m2 = umxFixAll(m1, run = TRUE, verbose = TRUE)
mxCompare(m1, m2)
```

umxGetParameters *umxGetParameters: Get parameters from a model*

Description

Get the parameter labels from a model. Like omxGetParameters, but supercharged with regular expressions for more power and ease!

Usage

```
umxGetParameters(inputTarget, regex = NA, free = NA, verbose = FALSE)

parameters(inputTarget, regex = NA, free = NA, verbose = FALSE)
```

Arguments

inputTarget	An object to get parameters from: could be a RAM mxModel
regex	A regular expression to filter the labels defaults to NA - just returns all labels)
free	A Boolean determining whether to return only free parameters.
verbose	How much feedback to give

References

- http://www.github.com/tbates/umx

See Also

Other Reporting Functions: loadings.MxModel, umxAPA, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx_time, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1)
umxGetParameters(m1)
m1 = umxRun(m1, setLabels = TRUE)
umxGetParameters(m1)
umxGetParameters(m1, free = TRUE) # only the free parameter
```

```
umxGetParameters(m1, free = FALSE) # only parameters which are fixed
## Not run:
# Complex regex patterns
umxGetParameters(m2, regex = "S_r_[0-9]c_6", free = TRUE) # Column 6 of matrix "as"

## End(Not run)
```

---

umxGxE *umxGxE: Implements ACE models with moderation of paths, e.g. by SES.*

Description

Make a 2-group GxE (moderated ACE) model (Purcell, 2002). GxE interaction studies test the hypothesis that the strength of genetic (or environmental) influence varies parametrically (usuaally linear effects on path estimates) across levels of environment. umxGxE allows detecting, testing, and visualizing G xE (or C or E x E) interaction forms.

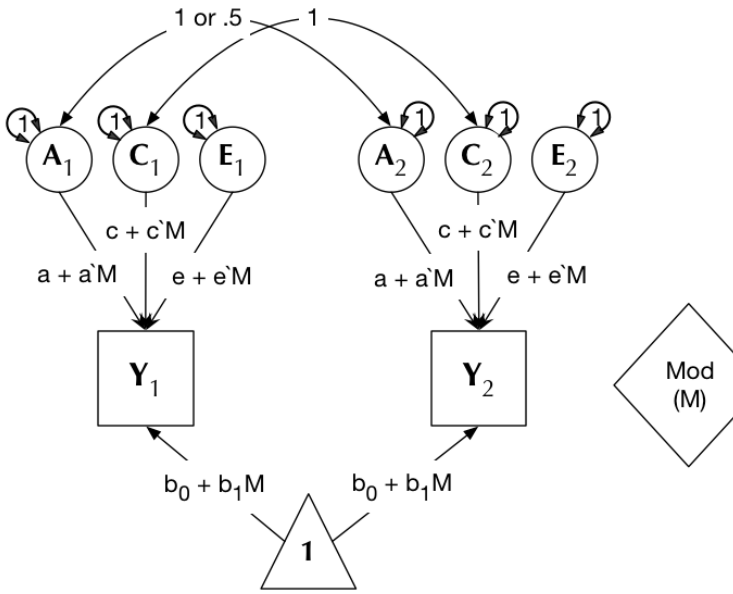
Usage

```
umxGxE(name = "G_by_E", selDVs, selDefs, dzData, mzData, suffix = NULL,
  lboundACE = NA, lboundM = NA, dropMissingDef = FALSE,
  autoRun = getOption("umx_auto_run"))
```

Arguments

name	The name of the model (defaults to "G_by_E")
selDVs	The dependent variable (e.g. IQ)
selDefs	The definition variable (e.g. socio economic status)
dzData	The DZ dataframe containing the Twin 1 and Twin 2 DV and moderator (4 columns)
mzData	The MZ dataframe containing the Twin 1 and Twin 2 DV and moderator (4 columns)
suffix	Expand variable base names, i.e., "_T" makes var -> var_T1 and var_T2
lboundACE	= numeric: If !is.na, then lbound the main effects at this value (default = NA)
lboundM	= numeric: If !is.na, then lbound the moderators at this value (default = NA)
dropMissingDef	Whether to automatically drop missing def var rows for the user (gives a warning) default = FALSE
autoRun	Whether to run the model, and return that (default), or just to create it and return without running.

Details



The following figure the GxE model as a path diagram:

Value

- GxE `mxModel`

References

- Purcell, S. (2002). Variance components models for gene-environment interaction in twin analysis.
Twin Research, **6**, 554-571. Retrieved from https://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&do

See Also

- `plot()` and `umxSummary()` work for IP, CP, GxE, SAT, and ACE models.

Other Twin Modeling Functions: `plot.MxModel`, `umxACESexLim`, `umxACEcov`, `umxACE`, `umxCF_SexLim`, `umxCP`, `umxGxE_window`, `umxIP`, `umxPlotACEcov`, `umxPlotCP`, `umxPlotGxE`, `umxPlotIP`, `umxSummaryACEcov`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxE`, `umxSummaryIP`, `umx`

Examples

```
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDVs, selDefs)
mzData  = subset(twinData, zyg == 1, selVars)[1:80,]
dzData  = subset(twinData, zyg == 3, selVars)[1:80,]
m1 = umxGxE(selDVs = selDVs, selDefs = selDefs,
dzData = dzData, mzData = mzData, dropMissing = TRUE)
```

```
# Plot Moderation
umxSummaryGxE(m1)
umxSummary(m1, location = "topright")
umxSummary(m1, separateGraphs = FALSE)
m2 = umxModify(m1, "am_.*", regex=TRUE, comparison = TRUE)
```

---

umxGxE_window *umxGxE_window*

Description

Make a 2-group GxE (moderated ACE) model using LOSEM. In GxE interaction studies, typically, the hypothesis that the strength of genetic influence varies parametrically (usuaally linear effects on path estimates) across levels of environment. Of course, the function linking genetic influence and context is not necessarily linear, but may react more steeply at the extremes, or take other, unknown forms. To avoid obscuring the underlying shape of the interaction effect, local structural equation modeling (LOSEM) may be used, and GxE_window implements this. LOSEM is a non-parametric, estimating latent interaction effects across the range of a measured moderator using a windowing function which is walked along the context dimension, and which weights subjects near the centrre of the window highly relative to subjects far above or below the window centre. This allows detecting and visualizing arbitrary GxE (or CxE or ExE) interaction forms.

Usage

```
umxGxE_window(selDVs = NULL, moderator = NULL, mzData = mzData,
  dzData = dzData, suffix = NA, weightCov = FALSE, target = NULL,
  width = 1, plotWindow = FALSE, return = c("estimates", "last_model"))
```

Arguments

selDVs	The dependent variables for T1 and T2, e.g. c("bmi_T1", "bmi_T2")
moderator	The name of the moderator variable in the dataset e.g. "age", "SES" etc.
mzData	Dataframe containing the DV and moderator for MZ twins
dzData	Dataframe containing the DV and moderator for DZ twins
suffix	(optional) separator, e.g. "_T" which will be used expand base names into full variable names: e.g.: 'bmi' –> c("bmi_T1", "bmi_T2")
weightCov	Whether to use cov.wt matrices or FIML default = FALSE, i.e., FIML
target	A user-selected list of moderator values to test (default = NULL = explore the full range)
width	An option to widen or narrow the window from its default (of 1)
plotWindow	whether to plot what the window looks like
return	whether to return the last model (useful for specifiedTargets) or the list of estimates (default = "estimates")

Value

- Table of estimates of ACE along the moderator

References

- Hildebrandt, A., Wilhelm, O, & Robitzsch, A. (2009) Complementary and competing factor analytic approaches for the investigation of measurement invariance. *Review of Psychology*, **16**, 87–107.

Briley, D.A., Harden, K.P., Bates, T.C., Tucker-Drob, E.M. (2015). Nonparametric Estimates of Gene x Environment Interaction Using Local Structural Equation Modeling. *Behavior Genetics*.

See Also

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
library(umx);
# ==============================
# = 1. Open and clean the data =
# ==============================
# umxGxE_window takes a dataframe consisting of a moderator and two DV columns: one for each twin
mod = "age" # The name of the moderator column in the dataset
selDVs = c("bmi1", "bmi2") # The DV for twin 1 and twin 2
data(twinData) # Dataset of Australian twins, built into OpenMx
# The twinData consist of two cohorts. First we label them
# TODO: Q for OpenMx team: can I add a cohort column to this dataset?
twinData$cohort = 1; twinData$cohort[twinData$zyg %in% 6:10] = 2
twinData$zyg[twinData$cohort == 2] = twinData$zyg[twinData$cohort == 2]-5
# And set a plain-English label
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
# The model also assumes two groups: MZ and DZ. Moderator can't be missing
# Delete missing moderator rows
twinData = twinData[!is.na(twinData[mod]),]
mzData = subset(twinData, ZYG == "MZFF", c(selDVs, mod))
dzData = subset(twinData, ZYG == "DZFF", c(selDVs, mod))

# ========================
# = 2. Run the analyses! =
# ========================
# Run and plot for specified windows (in this case just 1927)
umxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData,
target = 40, plotWindow = TRUE)

## Not run:
# Run with FIML (default) uses all information
umxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData);
```

```
# Run creating weighted covariance matrices (excludes missing data)
umxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData,
weightCov = TRUE);

## End(Not run)
```

---

umxHetCor *umxHetCor*

Description

umxHetCor Helper to return just the correlations from John Fox's polycor::hetcor function

Usage

```
umxHetCor(data, ML = FALSE, use = c("pairwise.complete.obs",
  "complete.obs"), treatAllAsFactor = FALSE, verbose = FALSE)
```

Arguments

data	A data.frame of columns for which to compute heterochoric correlations
ML	Whether to use Maximum likelihood computation of correlations (default = FALSE)
use	How to handle missing data: "complete.obs" (Default), "pairwise.complete.obs"
treatAllAsFactor	Whether to treat all columns as factors, whether they are or not.
verbose	How much to tell the user about what was done.

Value

- A matrix of correlations

References

-

See Also

Other Data Functions: umxCovData, umxFactor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

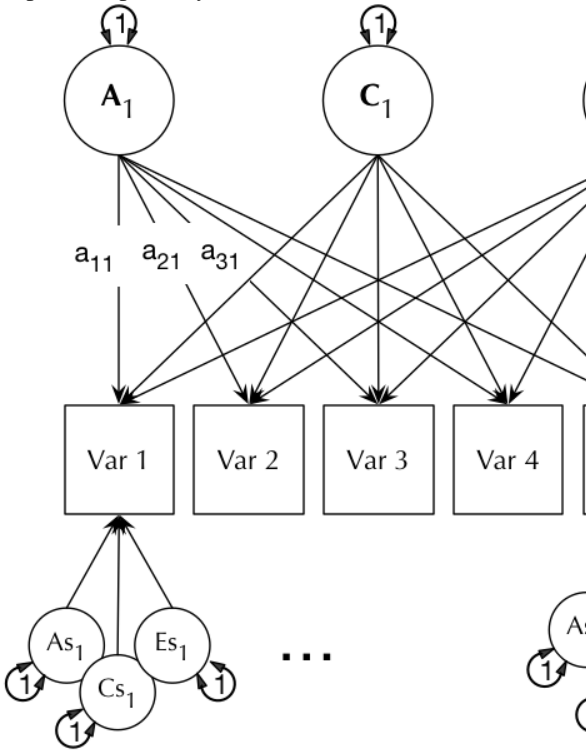
Examples

```
umxHetCor(mtcars[,c("mpg", "am")])
umxHetCor(mtcars[,c("mpg", "am")], treatAllAsFactor = FALSE, verbose = TRUE)
```

umxIP *umxIP: Build and run an Independent pathway twin model*

Description

Make a 2-group Independent Pathway twin model (Common-factor independent-pathway multivariate model) The following figure shows the IP model diagramatically:



Usage

```
umxIP(name = "IP", selDVs, dzData, mzData, suffix = NULL, nFac = 1,
  freeLowerA = FALSE, freeLowerC = FALSE, freeLowerE = FALSE,
  equateMeans = TRUE, dzAr = 0.5, dzCr = 1, correlatedA = FALSE,
  addStd = TRUE, addCI = TRUE, numObsDZ = NULL, numObsMZ = NULL,
  autoRun = getOption("umx_auto_run"), sep = NULL)
```

Arguments

name	The name of the model (defaults to "IP")
selDVs	The variables to include
dzData	The DZ dataframe
mzData	The MZ dataframe

suffix	The suffix for twin 1 and twin 2, often "_T". If set, you can omit suffixes in SelDVs, i.e., just "dep" not c("dep_T1", "dep_T2")
nFac	How many common factors (default = 1)
freeLowerA	Whether to leave the lower triangle of A free (default = F)
freeLowerC	Whether to leave the lower triangle of C free (default = F)
freeLowerE	Whether to leave the lower triangle of E free (default = F)
equateMeans	Whether to equate the means across twins (defaults to T)
dzAr	The DZ genetic correlation (defaults to .5, vary to examine assortative mating)
dzCr	The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model)
correlatedA	Whether factors are allowed to correlate (not implemented yet: FALSE)
addStd	Whether to add the algebras to compute a std model (defaults to TRUE)
addCI	Whether to add the interval requests for CIs (defaults to TRUE)
numObsDZ	= todo: implement ordinal Number of DZ twins: Set this if you input covariance data
numObsMZ	= todo: implement ordinal Number of MZ twins: Set this if you input covariance data
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)
sep	allowed as a synonym for "suffix"

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

- plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
zygList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = zygList)
mzData <- subset(twinData, ZYG == "MZFF")
dzData <- subset(twinData, ZYG == "DZFF")
selDVs = c("ht", "wt") # These will be expanded into "ht1" "ht2"
m1 = umxIP(selDVs = selDVs, suffix = "", dzData = dzData, mzData = mzData)
umxSummary(m1)
plot(m1)
```

umxJiggle *umxJiggle*

Description

umxJiggle takes values in a matrix and jiggles them

Usage

```
umxJiggle(matrixIn, mean = 0, sd = 0.1, dontTouch = 0)
```

Arguments

matrixIn	an mxMatrix to jiggle the values of
mean	the mean value to add to each value
sd	the sd of the jiggle noise
dontTouch	A value, which, if found, will be left as-is (defaults to 0)

Value

- mxMatrix

References

- http://www.github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx

Examples

```
## Not run:
mat1 = umxJiggle(mat1)

## End(Not run)
```

umxLabel *umxLabel: Add labels to a RAM model, matrix, or path*

Description

umxLabel adds labels to things, be it an: mxModel (RAM or matrix based), an mxPath, or an mxMatrix This is a core function in umx: Adding labels to paths opens the door to umxEquate, as well as omxSetParameters

Usage

```
umxLabel(obj, suffix = "", baseName = NA, setfree = FALSE, drop = 0,
  labelFixedCells = TRUE, jiggle = NA, boundDiag = NA, verbose = FALSE,
  overRideExisting = FALSE, name = NULL)
```

Arguments

obj	An mxModel (RAM or matrix based), mxPath, or mxMatrix
suffix	String to append to each label (might be used to distinguish, say male and female submodels in a model)
baseName	String to prepend to labels. Defaults to NA ("")
setfree	Whether to label only the free paths (defaults to FALSE)
drop	The value to fix "drop" paths to (defaults to 0)
labelFixedCells	= TRUE
jiggle	How much to jiggle values in a matrix or list of path values
boundDiag	Whether to bound the diagonal of a matrix
verbose	How much feedback to give the user (default = FALSE)
overRideExisting	= FALSE
name	Optional new name if given a model. Default (NULL) does not rename model.

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umxJiggle, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx

Examples

```
# ============================================================
# = Show how OpenMx models are not labels, and then add labels =
# ============================================================
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umxGetParameters(m1) # Default "matrix address" labels, i.e "One Factor.S[2,2]"
m1 = umxLabel(m1)
umxGetParameters(m1, free = TRUE) # Informative labels: "G_to_x1", "x4_with_x4", etc.
# ======================================================================
# = Create a new model, with suffixes added to paths, and model renamed =
# ======================================================================
m2 = umxLabel(m1, suffix= "_male", overRideExisting= TRUE, name = "male_model")
umxGetParameters(m2, free = TRUE) # suffixes added

# ============================
# = Example Labeling a matrix =
# ============================
a = umxLabel(mxMatrix(name = "a", "Full", 3, 3, values = 1:9))
a$labels
# note: labels with "data." in the name are left untouched!
a = mxMatrix(name = "a", "Full", 1,3, labels = c("data.a", "test", NA))
umxLabel(a, verbose = TRUE)
umxLabel(a, verbose = TRUE, overRideExisting = FALSE)
umxLabel(a, verbose = TRUE, overRideExisting = TRUE)
```

`umxLatent` *umxLatent: Helper to ease making formative and reflective latent variables*

Description

Helper to ease the creation of latent variables including formative and reflective variables (see below) For formative variables, the manifests define (form) the latent. This function takes care of intercorrelating manifests for formatives, and fixing variances correctly

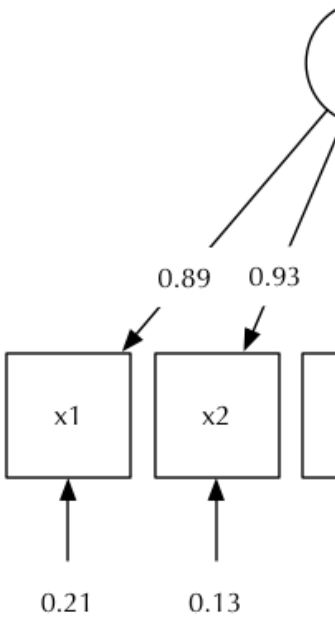
Usage

```
umxLatent(latent = NULL, formedBy = NULL, forms = NULL, data = NULL,
  type = NULL, name = NULL, labelSuffix = "", verbose = TRUE)
```

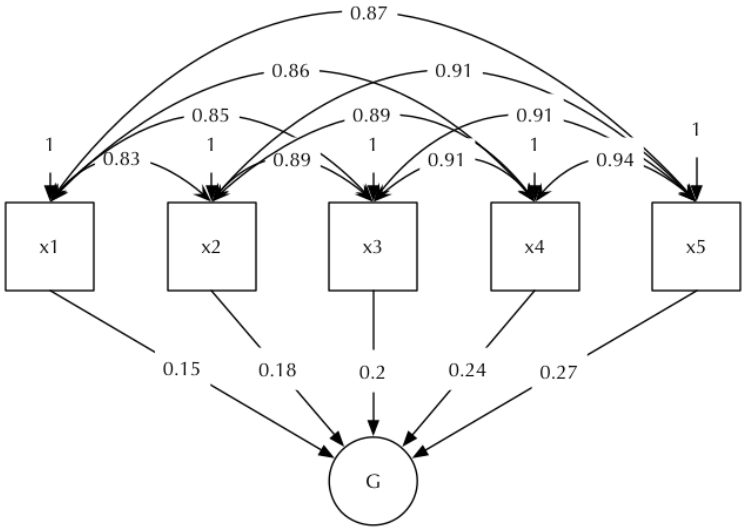
Arguments

latent	the name of the latent variable (string)
formedBy	the list of variables forming this latent
forms	the list of variables which this latent forms (leave blank for formedBy)
data	the dataframe being used in this model
type	= \"exogenous|endogenous\"
name	A name for the path NULL
labelSuffix	a suffix string to append to each label
verbose	Default is TRUE as this function does quite a lot

Details



The following figures show how a reflective and a formative variable look as path diagrams:



formative

Value

- path list

References

- http://www.github.com/tbates/umx

See Also

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxMatrix, umxPath, umxRAM, umxReduce, umxRun, umx

Examples

```
## Not run:
library(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor) # x1-5
theData = cov(demoOneFactor)
m1 = mxModel("reflective", type = "RAM",
manifestVars = manifests,
latentVars   = latents,
# Factor loadings
umxLatent("G", forms = manifests, type = "exogenous", data = theData),
mxData(theData, type = "cov", numObs = nrow(demoOneFactor))
)
m1 = umxRun(m1, setValues = TRUE, setLabels = TRUE); umxSummary(m1, show="std")
plot(m1)

m2 = mxModel("formative", type = "RAM",
manifestVars = manifests,
latentVars   = latents,
# Factor loadings
umxLatent("G", formedBy = manifests, data = theData),
mxData(theData, type = "cov", numObs = nrow(demoOneFactor))
)
m2 = umxRun(m2, setValues = TRUE, setLabels = TRUE);
umxSummary(m2, show = "std")
plot(m2)

## End(Not run)
```

---

umxMatrix *umxMatrix: labeled mxMatrix, with name in the first place*

Description

umxMatrix is a wrapper for mxMatrix which labels cells buy default, and has the name parameter first in order.

Usage

```
umxMatrix(name = NA, type = "Full", nrow = NA, ncol = NA,
  free = FALSE, values = NA, labels = TRUE, lbound = NA, ubound = NA,
  byrow = getOption("mxByrow"), dimnames = NA,
  condenseSlots = getOption("mxCondenseMatrixSlots"), ...,
  joinKey = as.character(NA), joinModel = as.character(NA), jiggle = NA)
```

Arguments

name	The name of the matrix (Default = NA). Note the different order compared to mxMatrix!
type	The type of the matrix (Default = "Full")
nrow	Number of rows in the matrix: Must be set
ncol	Number of columns in the matrix: Must be set
free	Whether cells are free (Defaul FALSE)
values	The values of the matrix (Default NA)
labels	Either whether to label the matrix (default TRUE), OR a vector of labels to apply.
lbound	Lower bounds on cells (Defaults to NA)
ubound	Upper bounds on cells (Defaults to NA)
byrow	Whether to fill the matrix down columns or across rows first (Default = getOption('mxByrow')
dimnames	NA
condenseSlots	Whether to save memory by NULLing out unused matrix elements, like labels, ubnound etc. Default = getOption('mxCondenseMatrixSlots')
...	Additional parameters (!! not currently supported by umxMatrix)
joinKey	See mxMatrix documentation: Defaults to as.character(NA)
joinModel	See mxMatrix documentation: Defaults to as.character(NA)
jiggle	= NA passed to umxLabel to jiggle start values (default does nothing)

Value

- mxMatrix

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umxLabel

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxPath, umxRAM, umxReduce, umxRun, umx

Examples

```
umxMatrix("test", "Full", 1, 1)
```

umxMI *umxMI*

Description

Report modifications which would improve fit. Notes: 1. Runs much fast with full = FALSE (but this doesn't allow the model to re-fit around the newly- freed parameter). 2. Compared to mxMI, this function returns top changes, and also suppresses the run message. 3. Finally, of course: see the requirements for (legitimate) post-hoc modeling in mxMI You are almost certainly doing better science when testing competing models rather than modifying a model to fit.

Usage

```
umxMI(model = NA, matrices = NA, full = TRUE, numInd = NA,
  typeToShow = "both", decreasing = TRUE)
```

Arguments

model	An mxModel for which to report modification indices
matrices	which matrices to test. The default (NA) will test A & S for RAM models
full	Change in fit allowing all parameters to move. If FALSE only the parameter under test can move.
numInd	How many modifications to report. Use -1 for all. Default (NA) will report all over 6.63 (p = .01)
typeToShow	Whether to shown additions or deletions (default = "both")
decreasing	How to sort (default = TRUE, decreasing)

References

- http://www.github.com/tbates/umx

See Also

- mxMI

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxFixAll, umxSetParameters, umxUnexplainedCausalNexus, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)[1:3]
df = mxData(cov(demoOneFactor[,manifests]), type = "cov", numObs = 500)
m1 <- umxRAM("One Factor", data = df,
umxPath(latents, to = manifests),
```

```
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
umxMI(m1, full=FALSE)
```

---

umxModel *Catches users typing umxModel instead of umxRAM*

Description

Catches a common typo, moving from mxModel to umx.

Usage

```
umxModel(...)
```

Arguments

...	anything. We're just going to throw an error.

Value

-

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umxRAM, mxModel

Other xmu internal not for end user: xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
## Not run:
umxModel()

## End(Not run)
```

umxModify *umxModify: Add, set, or drop model paths by label.*

Description

umxModify allows you to modify, re-run and summarize an mxModel, all in one line of script. You can add paths, or other model elements, set paths or drop them. As an example, this one-liner drops a path labelled "Cs", and returns the updated model:

Usage

```
umxModify(lastFit, update = NULL, regex = FALSE, free = FALSE,
  value = 0, freeToStart = NA, name = NULL, verbose = FALSE,
  intervals = FALSE, comparison = FALSE, autoRun = TRUE,
  dropList = "deprecated")
```

Arguments

lastFit	The mxModel you wish to update and run.
update	What to update before re-running. Can be a list of labels, a regular expression (set regex = TRUE) or an object such as mxCI etc.
regex	Whether or not update is a regular expression (defaults to FALSE). If you provide a string, it over-rides the contents of update, and sets regex to TRUE.
free	The state to set "free" to for the parameters whose labels you specify (defaults to free = FALSE, i.e., fixed)
value	The value to set the parameters whose labels you specify too (defaults to 0)
freeToStart	Whether to update parameters based on their current free-state. free = c(TRUE, FALSE, NA), (defaults to NA - i.e, not checked)
name	The name for the new model
verbose	How much feedback to give
intervals	Whether to run confidence intervals (see mxRun)
comparison	Whether to run umxCompare() after umxRun
autoRun	Whether to run the modified model before returning it (default), or just to modify and return without running.
dropList	(deprecated: use 'update' instead.

Details

```
fit2 = umxModify(fit1, update = "Cs", name = "newModelName", comparison = TRUE)
```

Regular expressions are a powerful feature: they let you drop collections of paths by matching patterns fit2 = umxModify(fit1, regex = "C[sr]", name = "drop_Cs_andCr", comparison = TRUE)

If you are just starting out, you might find it easier to be more explicit. Like this:

fit2 = omxSetParameters(fit1, labels = "Cs", values = 0, free = FALSE, name = "newModelName") fit2 = mxRun(fit2) summary(fit2)

Value

- mxModel

References

- http://github.com/tbates/umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
umx_set_optimizer("SLSQP")

m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
m2 = umxModify(m1, update = "G_to_x1", name = "drop_X1")
umxSummary(m2); umxCompare(m1, m2)
# 1-line version including comparison
m2 = umxModify(m1, update = "G_to_x1", name = "drop_X1", comparison = TRUE)
# use regular expression to drop multiple paths: e.g. G to x3, x4, x5
m2 = umxModify(m1, update = "^G_to_x[3-5]", regex = TRUE, name = "no_G_to_x3_5", comp = TRUE)
# Same, but shorter
m2 = umxModify(m1, regex  = "^G_to_x[3-5]", name = "no_G_to_x3_5")
# Same, but don't autoRun
m2 = umxModify(m1, regex  = "^G_to_x[3-5]", name = "no_G_to_x3_5", autoRun = FALSE)
m2 = umxModify(m1, update = "G_to_x1", value = .2, name = "fix_G_x1_at_point2", comp = TRUE)
m3 = umxModify(m2, update = "G_to_x1", free = TRUE, name = "free_G_x1_again", comparison = TRUE)
```

---

umxPadAndPruneForDefVars

umxPadAndPruneForDefVars

Description

Replaces NAs in definition slots with the mean for that variable ONLY where all data are missing for that twin

Usage

```
umxPadAndPruneForDefVars(df, varNames, defNames, suffixes, highDefValue = 99,
  rm = c("drop_missing_def", "pad_with_mean"))
```

Arguments

df	the dataframe to process
varNames	list of names of the variables being analysed
defNames	list of covariates
suffixes	suffixes that map names on columns in df (i.e., c("T1", "T2"))
highDefValue	What to replace missing definition variables (covariates) with. Default = 99
rm	= how to handle missing values in the varNames. Default is "drop_missing_def", "pad_with_mean")

Value

- dataframes

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
## Not run:
df = umxPadAndPruneForDefVars(df, "E", "age", c("_T1", "_T2"))

## End(Not run)
```

---

umxPath *umxPath: Easier (but powerful) specification of RAM paths.*

Description

The goal of this function is to enable quck-to-write, quick-to-read, flexible path descriptions for RAM models in OpenMx.

It introduces the following new words to our vocabulary for describing paths: **with**, **var**, **cov**, **means**, **v1m0**, **v.m0**, **v.m.**, **fixedAt**, **freeAt**, **firstAt**, **unique.bivariate**, **unique.pairs**, **Cholesky**, **defn**, **forms**.

The new preposition "with" means you no-longer need set arrows = 2 on covariances. Instead, you can say:

`umxPath(A, with = B)` instead of `mxPath(from = A, to = B, arrows = 2)`.

Specify a variance for A with

`umxPath(var = "A")`.

This is equivalent to `mxPath(from = "A", to = "A", arrows = 2)`.

Of course you can use vectors anywhere:

`umxPath(var = c('N','E', 'O'))`

To specify a mean, you just say

`umxPath(mean = "A")`, which is equivalent to `mxPath(from = "one", to = "A")`.

To fix a path at a value, instead of to `mxPath(from = A, to = A, arrows = 2, free = FALSE, values = 1)` you can say:

`umxPath(var = "A", fixedAt = 1)` .

The common task of creating a variable with variance fixed at 1 and mean at 0 is done thus:

`umxPath(v1m0 = "A")`

For free variance and means use:

`umxPath(v.m. = "A")`

umxPath exposes "unique.bivariate" so you don't have to remember how to fill in connect = in mxPath (you can still use connect if you wish).

So, to create paths A<->B, B<->C, and A<->C, you would say:

`umxPath(unique.bivariate = c('A',"B","C"))`

Setting up a latent trait, you can fix the loading of the first path with

`mxPath(A, to = c(B,C,D), fixFirst = TRUE)`

This is equivalent to `mxPath(from = A, to = c(B,C,D), free = c(F, T, T), values = c(1, .5, .4))`.

A new feature is the ability to create Cholesky-pattern connections:

`umxPath(Cholesky = c("A1", "A2"), to c("var1", "var2"))`

Finally, a feature, not implemented in this release, but intended for the future is John Fox "sem"-package style notation,

i.e., "A -> B; X <-> B; "

Usage

```
umxPath(from = NULL, to = NULL, with = NULL, var = NULL, cov = NULL,
  unique.bivariate = NULL, unique.pairs = NULL, forms = NULL,
  Cholesky = NULL, defn = NULL, means = NULL, v1m0 = NULL,
  v.m. = NULL, v0m0 = NULL, v.m0 = NULL, fixedAt = NULL,
  freeAt = NULL, firstAt = NULL, connect = c("single", "all.pairs",
  "all.bivariate", "unique.pairs", "unique.bivariate"), arrows = 1,
  free = TRUE, values = NA, labels = NA, lbound = NA, ubound = NA,
  hasMeans = NULL)
```

Arguments

from	either a source variable e.g "A" or c("A","B"), OR a sem-style path description, e.g. "A-> B" or "C <> B"
to	one or more target variables for one-headed paths, e.g "A" or c("A","B")

with	same as "to = vars, arrows = 2". nb: from, to= and var= must be left empty (their default)
var	equivalent to setting "from = vars, arrows = 2". nb: from, to, and with must be left empty (their default)
cov	equivalent to setting "from = X, to = Y, arrows = 2". nb: from, to, and with must be left empty (their default)
unique.bivariate	equivalent to setting "connect = "unique.bivariate", arrows = 2". nb: from, to, and with must be left empty (their default)
unique.pairs	equivalent to setting "connect = "unique.pairs", arrows = 2" (don't use from, to, or with)
forms	Paired with from, this will build a formative variable. from vars form the latent. Latent variance is fixed at 0. Loading of path 1 is fixed at 1. unique.bivariate among froms.
Cholesky	Treat **Cholesky** vars as latent and **to** as measured, and connect as in an ACE model.
defn	latent variable, var@0 mean fixed, with label-based as data source
means	equivalent to "from = 'one', to = x. nb: from, to, with and var must be left empty (their default).
v1m0	variance of 1 and mean of zero in one call.
v.m.	variance and mean, both free.
v0m0	variance and mean, both fixed at zero.
v.m0	variance free, mean fixed at zero.
fixedAt	Equivalent to setting "free = FALSE, values = x" nb: free and values must be left empty (their default)
freeAt	Equivalent to setting "free = TRUE, values = x" nb: free and values must be left empty (their default)
firstAt	first value is fixed at this (values passed to free are ignored: warning if not a single TRUE)
connect	as in mxPath - nb: Only used when using from and to
arrows	as in mxPath - nb: Only used when using from and to
free	whether the value is free to be optimised
values	default value list
labels	labels for each path
lbound	lower bounds for each path value
ubound	upper bounds for each path value
hasMeans	Used in 'forms' case to know whether the data have means or not.

Details

This function returns a standard mxPath, but gives new options for specifying the path. In addition to the normal "from" and "to", it adds specialised parameters for variances (var), two headed paths (with) and means (mean). There are also new terms to describe fixing values: "fixedAt" and "fixFirst".

Finally, (in future) it will allow sem-style "A->B" string specification.

Value

- 1 or more mxPaths

References

- http://tbates.github.io

See Also

- mxPath

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxMatrix, umxRAM, umxReduce, umxRun, umx

Examples

```
require(umx)
# Some examples of paths with umxPath
umxPath("A", to = "B") # One-headed path from A to B
umxPath("A", to = "B", fixedAt = 1) # same, with value fixed @1
umxPath("A", to = c("B", "C"), fixedAt = 1:2) # same, with more than 1 value
umxPath("A", to = LETTERS[2:4], firstAt = 1) # Fix only the first path, others free
umxPath(var = "A") # Give a variance to A
umxPath(var = "A", fixedAt = 1) # Give a variance, fixed at 1
umxPath(var = LETTERS[1:5], fixedAt = 1)
umxPath(means = c("A","B")) # Create a means model for A: from = "one", to = "A"
umxPath(v1m0 = "A") # Give "A" variance and a mean, fixed at 1 and 0 respectively
umxPath(v.m. = "A") # Give "A" variance and a mean, leaving both free.
umxPath("A", with = "B") # using with: same as "to = B, arrows = 2"
umxPath("A", with = "B", fixedAt = .5) # 2-head path fixed at .5
umxPath("A", with = c("B", "C"), firstAt = 1) # first covariance fixed at 1
umxPath(cov = c("A", "B"))  # Covariance A <-> B
umxPath(unique.bivariate = letters[1:4]) # bivariate paths a<->b, a<->c, a<->d, b<->c etc.
umxPath(unique.pairs = letters[1:4]) # bivariate paths a<->b, a<->c, a<->d, b<->c etc.
umxPath(Cholesky = c("A1","A2"), to = c("m1", "m2")) # Cholesky
# A worked example
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
m1 <- umxRAM("One Factor", data = myData,
umxPath(latents, to = manifests),
umxPath(var = manifests),
```

```
umxPath(var = latents, fixedAt = 1.0)
)
umxSummary(m1, show = "std")

# ====================
# = Cholesky example =
# ====================
latents   = paste0("A", 1:3)
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
m1 <- umxRAM("Chol", data = myData,
umxPath(Cholesky = latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
umxSummary(m1, show = "std")

# The following NOT YET implemented!!
# umxPath("A <-> B") # same path as above using a string
# umxPath("A -> B") # one-headed arrow with string syntax
# umxPath("A <> B; A <-- B") # This is ok too
# umxPath("A -> B; B>C; C --> D") # two paths. white space and hyphens not needed
# # manifests is a reserved word, as is latents.
# # It allows the string syntax to use the manifestVars variable
# umxPath("A -> manifests")
```

---

umxPlotACE *umxPlotACE*

Description

Make a graphical display of an ACE model

Usage

```
umxPlotACE(x = NA, file = "name", digits = 2, means = FALSE,
  std = TRUE, showMeans = "deprecated", showStd = "deprecated", ...)
```

Arguments

x	mxModel to plot (created by umxACE in order to inherit the MxModel.ACE class)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
digits	How many decimals to include in path loadings (default is 2)
means	Whether to show means paths (default is FALSE)
std	Whether to standardize the model (default is TRUE)
showMeans	DEPRECATED: just use 'means = ' for simplicity of typing.

showStd	DEPRECATED: just use 'std = '
...	Additional (optional) parameters

Value

- optionally return the dot code

References

- http://www.github.com/tbates/umx

See Also

Other Plotting functions: plot.MxModel, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(twinData)
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
selDVs = c("bmi1", "bmi2")
mzData <- subset(twinData, ZYG == "MZFF", selDVs)
dzData <- subset(twinData, ZYG == "DZFF", selDVs)
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
plot(m1)
plot(m1, std = FALSE) # don't standardize
```

---

umxPlotACEcov *umxPlotACEcov*

Description

Make a graphical display of an ACE model with covariates.

Usage

```
umxPlotACEcov(x = NA, file = "name", digits = 2, means = FALSE,
  std = TRUE, showMeans = "deprecated", ...)
```

Arguments

x	mxModel to plot (created by umxACE in order to inherit the MxModel.ACE class)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
digits	How many decimals to include in path loadings (default is 2)
means	Whether to show means paths (default is FALSE)
std	Whether to standardize the model (default is TRUE)
showMeans	DEPRECATED use "means" instead
...	Additional (optional) parameters

Value

- optionally return the dot code

References

- http://tbates.github.io

See Also

Other Plotting functions: plot.MxModel, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
# BMI ?twinData from Australian twins.
# Cohort 1 Zygosity 1 == MZ females 3 == DZ females
data(twinData)
# Pick the variables. We will use base names (i.e., "bmi") and set suffix.
selDVs  = c("bmi")
selCovs = c("age")
selVars = umx_paste_names(c(selDVs, selCovs), sep = "", suffixes= 1:2)
# just top few pairs so example runs quickly
mzData = subset(twinData, zyg == 1, selVars)[1:100, ]
dzData = subset(twinData, zyg == 3, selVars)[1:100, ]
# TODO update for new dataset variable zygosity
# mzData = subset(twinData, zygosity == "MZFF", selVars)[1:200, ]
# dzData = subset(twinData, zygosity == "DZFF", selVars)[1:200, ]
m1 = umxACEcov(selDVs = selDVs, selCovs = selCovs, dzData = dzData, mzData = mzData,
 suffix = "", autoRun = TRUE)
```

```
plot(m1)
plot(m1, std = FALSE) # don't standardize
```

---

umxPlotCP *umxPlotCP*

Description

Draw a graphical figure for a Common Pathway model

Usage

```
umxPlotCP(x = NA, file = "name", digits = 2, means = FALSE,
  std = TRUE, showMeans = "deprecated", ...)
```

Arguments

x	The Common Pathway mxModel to display graphically
file	The name of the dot file to write: NA = none; "name" = use the name of the model
digits	How many decimals to include in path loadings (defaults to 2)
means	Whether to show means paths (defaults to FALSE)
std	Whether to standardize the model (defaults to TRUE)
showMeans	deprecated: replace with just 'means'
...	Optional additional parameters

Value

- Optionally return the dot code

References

- http://tbates.github.io

See Also

- plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.

Other Plotting functions: plot.MxModel, umxPlotACEcov, umxPlotACE, umxPlotGxE, umxPlotIP

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
## Not run:
plot(yourCP_Model) # no need to remember a special name: plot works fine!

## End(Not run)
```

---

umxPlotGxE *umxPlotGxE*

Description

Plot GxE results (univariate environmental moderation of ACE components)

Usage

```
umxPlotGxE(x, xlab = NA, location = "topleft", separateGraphs = FALSE,
  ...)
```

Arguments

x	A fitted umxGxE model to plot
xlab	String to use for the x label (default = NA, which will use the variable name)
location	Where to plot the legend (default = "topleft") see ?legend for alternatives like bottomright
separateGraphs	(default = FALSE)
...	Optional additional parameters

Value

-

References

- http://tbates.github.io

See Also

- plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.
- umxGxE

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Other Plotting functions: plot.MxModel, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotIP

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDVs, selDefs)
mzData  = subset(twinData, zyg == 1, selVars)
dzData  = subset(twinData, zyg == 3, selVars)
m1 = umxGxE(selDVs = selDVs, selDefs = selDefs,
  dzData = dzData, mzData = mzData, dropMissing = TRUE)
plot(m1)
umxPlotGxE(x = m1, xlab = "SES", separateGraphs = TRUE, location = "topleft")
```

---

umxPlotIP *umxPlotIP*

Description

Make a graphical display of an Independent Pathway model

Usage

```
umxPlotIP(x = NA, file = "name", digits = 2, means = FALSE,
  std = TRUE, showMeans = "deprecated", ...)
```

Arguments

x	The umxIP model to plot
file	The name of the dot file to write: NA = none; "name" = use the name of the model
digits	How many decimals to include in path loadings (defaults to 2)
means	Whether to show means paths (defaults to FALSE)
std	whether to standardize the model (defaults to TRUE)
showMeans	Deprecated: replace with just 'means' for simplicity.
...	Optional additional parameters

Value

- optionally return the dot code

References

- http://tbates.github.io

See Also

- plot(), umxSummary()

Other Plotting functions: plot.MxModel, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxSummary.MxModel, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
## Not run:
plot(model)
umxPlotIP(model, file = NA)

## End(Not run)
```

---

umxRAM *Easy-to-use RAM model maker.*

Description

umxRAM expedites creation of RAM models, still without doing invisible things to the model.

Usage

```
umxRAM(model = NA, ..., data = NULL, name = NA, comparison = TRUE,
  setValues = TRUE, suffix = "", independent = NA,
  remove_unused_manifests = TRUE, showEstimates = c("none", "raw", "std",
  "both", "list of column names"), refModels = NULL,
  thresholds = c("deviationBased", "direct", "ignore", "left_censored"),
  autoRun = getOption("umx_auto_run"))
```

Arguments

model	A model to update (or set to string to use as name for new model)
...	mx or umxPaths, mxThreshold objects, etc.
data	data for the model. Can be an mxData or a data.frame
name	A friendly name for the model
comparison	Compare the new model to the old (if updating an existing model: default = TRUE)
setValues	Whether to generate likely good start values (Defaults to TRUE)

suffix	String to append to each label (useful if model will be used in a multi-group model)
independent	Whether the model is independent (default = NA)
remove_unused_manifests	Whether to remove variables in the data to which no path makes reference (defaults to TRUE)
showEstimates	Whether to show estimates. Defaults to no (alternatives = "raw", "std", etc.)
refModels	pass in reference models if available. Use FALSE to supress computing these if not provided.
thresholds	Whether to use deviation-based threshold modeling for ordinal data (if any is detected), direct, or do nothing.
autoRun	Whether to mxRun the model (default TRUE: the estimated model will be returned)

Details

Like mxModel, you list the theoretical causal paths. Unlike mxModel:

1. You don't need to set type = "RAM"
2. You don't need to list manifestVars (they are detected from path usage)
3. You don't need to list latentVars (detected as anything in paths but not in mxData)
4. You add data like you do in lm, with **data =**
5. with umxPath you can use powerful verbs like **var =**
6. You don't need to add labels: paths are automatically labelled "a_to_b" etc.
7. You don't need to set start values, they will be done for you.
8. You don't need to mxRun the model: it will run automatically, and print a summary

umxRAM is like lm, ggplot2 etc: you give the data in a data = parameter A common error is to include data in the main list, a bit like saying lm(y ~ x + df) instead of lm(y ~ x, data = dd).

nb: unlike mxModel, umxRAM needs data at build time.

If you are at the "sketching" stage of theory consideration, umxRAM supports a simple vector of manifest names to work with.

Comparison with other software

Some software has massive behind-the-scenes defaulting and path addition. I've played with similar features (like auto-creating error and exogenous variances using `endog.variances = TRUE` and `exog.variances = TRUE`). Also identification helpers like `fix = "latents"` and `fix = "firstLoadings"`.

To be honest, these are not only more trouble than they are worth, they encourage errors and poor modelling. I suggest user learn the handful of umxPath short cuts and stay clean and explicit!

Value

- mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

umxPath, umxSummary, plot

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxMatrix, umxPath, umxReduce, umxRun, umx

Examples

```
# ============================
# = Here's a simple example =
# ============================
m1 = umxRAM("tim", data = mtcars,
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(v.m. = c("wt", "disp", "mpg"))
)

# ====================================
# = A cov model, with steps laid out =
# ====================================

# 1. For convenience, list up the manifests you will be using
selVars = c("mpg", "wt", "disp")

# 2. Create an mxData object
myCov = mxData(cov(mtcars[,selVars]), type = "cov", numObs = nrow(mtcars) )

# 3. Create the model (see ?umxPath for more nifty options)
m1 = umxRAM("tim", data = myCov,
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(var = selVars)
)

# 4. Use umxSummary to get standardized parameters, CIs etc.
umxSummary(m1, show = "std")

# 5. Display path diagram
plot(m1)
plot(m1, resid = "line")

# ================================================================================
# = For playing around, umxRAM supports a vector of manifest names in place of data =
# ================================================================================

m1 = umxRAM("play", data = paste0("x", 1:4),
umxPath("g", to = paste0("x", 1:4)),
```

```
umxPath(var = paste0("x", 1:4)),
umxPath(v1m0 = "g")
)

# ================================================
# = This is an example of using your own labels:  =
#   umxRAM will not over-ride them                =
# ================================================
m1 = umxRAM("tim", data = myCov,
umxPath(c("wt", "disp"), to = "mpg"),
umxPath(cov = c("wt", "disp"), labels = "b1"),
umxPath(var = c("wt", "disp", "mpg"))
)
omxCheckEquals(m1$S$labels["disp", "wt"], "b1") # label preserved
m1$S$labels
#      mpg             wt            disp
# mpg  "mpg_with_mpg"  "mpg_with_wt" "disp_with_mpg"
# wt   "mpg_with_wt"   "wt_with_wt"  "b1"
# disp "disp_with_mpg" "b1"          "disp_with_disp"
```

---

umxRAM2Ordinal *umxRAM2Ordinal*

Description

umxRAM2Ordinal: Convert a RAM model whose data contain ordinal vars to a threshold-based model

Usage

```
umxRAM2Ordinal(model, verbose = T, thresholds = c("deviationBased",
  "direct", "ignore", "left_censored"), name = NULL, showEstimates = TRUE,
  refModels = NULL, autoRun = getOption("umx_auto_run"))
```

Arguments

model	An RAM model to add thresholds too.
verbose	Tell the user what was added and why (Default = TRUE)
thresholds	How to implement thresholds: c("deviationBased", "direct", "ignore", "left_censored")
name	= A new name for the modified model (NULL means leave it as it)
showEstimates	= Whether to show estimates in the summary (if autorunning) TRUE
refModels	pass in reference models if available. Use FALSE to supress computing these if not provided.
autoRun	= whether to run the model before returning it: defaults to getOption("umx_auto_run"))

Value

- mxModel

See Also

- umxRAM

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx

Examples

```
## Not run:
m1 = umxRAM2Ordinal(model)

## End(Not run)
```

---

umxReduce *umxReduce*

Description

Reduce a model - this is a work in progress.

Usage

```
umxReduce(m1, report = "html", baseFileName = "tmp")
```

Arguments

m1	an mxModel to reduce
report	How to report the results. "html" = open in browser
baseFileName	file (I add the html) to use when report = "html" (defaults to "tmp")

Value

-

References

- http://tbates.github.io

See Also

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxMatrix, umxPath, umxRAM, umxRun, umx

Examples

```
## Not run:
model = umxReduce(model)

## End(Not run)
```

umxRun *umxRun: Run an mxModel*

Description

umxRun is a version of mxRun which can run also set start values, labels, and run multiple times It can also calculate the saturated and independence likelihoods necessary for most fit indices.

Usage

```
umxRun(model, n = 1, calc_SE = TRUE, calc_sat = TRUE, setValues = FALSE,
  setLabels = FALSE, intervals = FALSE, comparison = NULL,
  setStarts = NULL)
```

Arguments

model	The mxModel you wish to run.
n	The maximum number of times you want to run the model trying to get a code green run (defaults to 1)
calc_SE	Whether to calculate standard errors (not used when n = 1) for the summary (they are not very accurate, so if you use mxCI or umxCI, you can turn this off)
calc_sat	Whether to calculate the saturated and independence models (for raw mxData mxModels) (defaults to TRUE - why would you want anything else?)
setValues	Whether to set the starting values of free parameters (defaults to F)
setLabels	Whether to set the labels (defaults to F)
intervals	Whether to run mxCI confindence intervals (defaults to F)
comparison	Whether to run umxCompare() after umxRun
setStarts	Deprecated way to setValues

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

Other Core Modelling Functions: plot.MxModel, umxDiagnose, umxLatent, umxMatrix, umxPath, umxRAM, umxReduce, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1) # just run: will create saturated model if needed
m1 = umxRun(m1, setValues = TRUE, setLabels = TRUE) # set start values and label all parameters
umxSummary(m1, show = "std")
m1 = mxModel(m1, mxCI("G_to_x1")) # add one CI
m1 = mxRun(m1, intervals = TRUE)
residuals(m1, run = TRUE) # get CIs on all free parameters
confint(m1, run = TRUE) # get CIs on all free parameters
m1 = umxRun(m1, n = 10) # re-run up to 10 times if not green on first run
```

---

umxSetParameters *umxSetParameters: Set parameters in an mxModel*

Description

Free or fix parameters in an mxModel. This allows similar actions that update enables for lm models. Updating can create duplicate labels, so this function also calls omxAssignFirstParameters to equate the start values for parameters which now have identical labels.

Usage

```
umxSetParameters(model, labels, free = NULL, values = NULL,
  newlabels = NULL, lbound = NULL, ubound = NULL, indep = FALSE,
  strict = TRUE, name = NULL, regex = FALSE, test = FALSE)
```

Arguments

model	an mxModel to WITH
labels	= labels to find
free	= new value for free
values	= new values
newlabels	= newlabels
lbound	= value for lbound
ubound	= value for ubound
indep	= whether to look in indep models

strict	whether to complain if labels not found
name	= new name for the returned model
regex	Is labels a regular expression (defaults to FALSE)
test	just show what you would do? (defaults to FALSE)

Details

It also supports regular expressions to select labels. In this respect, it is similar to umxModify without running the model.

Value

- mxModel

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umxModify, umxLabel

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxFixAll, umxMI, umxUnexplainedCausalNexus, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(demoOneFactor[1:80,], type = "raw"),
umxPath(from = latents, to = manifests),
umxPath(v.m. = manifests),
umxPath(v1m0 = latents)
)
parameters(m1, free=TRUE)
m2 = umxSetParameters(m1, "G_to_x1", newlabels= "G_to_x2", test = FALSE)
m2 = umxSetParameters(m1, "^", newlabels= "m1_", regex = TRUE, test = TRUE)
```

---

umxSummary *umxSummary.default*

Description

Report the fit of a OpenMx model or specialized model class (such as ACE, CP etc.) in a compact form suitable for reporting in a journal.

See documentation for RAM models summary here: umxSummary.MxModel.

View documentation on the ACE model subclass here: umxSummary.MxModel.ACE.

View documentation on the IP model subclass here: umxSummary.MxModel.IP.

View documentation on the CP model subclass here: umxSummary.MxModel.CP.

View documentation on the GxE model subclass here: umxSummary.MxModel.GxE.

Usage

```
umxSummary(model, ...)
```

Arguments

model	The mxModel whose fit will be reported
...	Other parameters to control model summary

See Also

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx_time, umx

---

umxSummary.MxModel *Shows a compact, publication-style, summary of a RAM model*

Description

Report the fit of a model in a compact form suitable for a journal. Emits a "warning" when model fit is worse than accepted criterion (TLI >= .95 and RMSEA <= .06; (Hu & Bentler, 1999; Yu, 2002).

Usage

```
## S3 method for class 'MxModel'
umxSummary(model, refModels = NULL,
  showEstimates = c("raw", "std", "none", "both"), digits = 2,
  report = c("1", "table", "html"), filter = c("ALL", "NS", "SIG"),
  SE = TRUE, RMSEA_CI = FALSE, matrixAddresses = FALSE, std = NULL, ...)
```

Arguments

model	The mxModel whose fit will be reported
refModels	Saturated models if needed for fit indices (see example below: If NULL will be competed on demand. If FALSE will not be computed. Only needed for raw data.
showEstimates	What estimates to show. By default, the raw estimates are shown (Options = c("raw", "std", "none", "both").
digits	How many decimal places to report (default = 2)
report	If "html", then show results in browser ("1", "table", "html")
filter	whether to show significant paths (SIG) or NS paths (NS) or all paths (ALL)
SE	Whether to compute SEs... defaults to TRUE. In rare cases, you might need to turn off to avoid errors.
RMSEA_CI	Whether to compute the CI on RMSEA (Defaults to FALSE)
matrixAddresses	Whether to show "matrix address" columns (Default = FALSE)
std	deprecated: use show = "std" instead!
...	Other parameters to control model summary

Details

Note: For some (multi-group) models, you will need to fall back on summary

CIs and Identification This function uses the standard errors reported by OpenMx to produce the CIs you see in umxSummary These are used to derive confidence intervals based on the formula 95

Sometimes they appear NA. This often indicates a model which is not identified (seehttp://davidakenny.net/cm/identify.htm). This can include empirical under-identification - for instance two factors that are essentially identical in structure. use mxCheckIdentification to check identification.

One or more paths estimated at or close to zero suggests that fixing one or two of these to zero may fix the standard error calculation, and alleviate the need to estimate likelihood-based or bootstrap CIs

If factor loadings can flip sign and provide identical fit, this creates another form of under-identification and can break confidence interval estimation. Fixing a factor loading to 1 and estimating factor variances can help here.

Value

- parameterTable returned invisibly, if estimates requested

References

- Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. Structural Equation Modeling, 6, 1-55.

- Yu, C.Y. (2002). Evaluating cutoff criteria of model fit indices for latent variable models with binary and continuous outcomes. University of California, Los Angeles, Los Angeles. Retrieved from http://www.statmodel.com/download/Yudissertation.pdf

http://tbates.github.io

See Also

- umxRun

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACE, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor",
data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
umxSummary(m1, showEstimates = "std")
# output as latex
umx_set_table_format("latex")
umxSummary(m1, showEstimates = "std")
umx_set_table_format("markdown")
# output as raw
umxSummary(m1, show = "raw")
m1 <- mxModel(m1,
  mxData(demoOneFactor[1:100,], type = "raw"),
  umxPath(mean = manifests),
  umxPath(mean = latents, fixedAt = 0)
)
m1 <- mxRun(m1)
umxSummary(m1, showEstimates = "std", filter = "NS")
```

---

umxSummaryACE *umxSummaryACE*

Description

Summarise a Cholesky model returned by \codeumxACE.

Usage

```
umxSummaryACE(model, digits = 2, file = getOption("umx_auto_plot"),
  comparison = NULL, std = TRUE, showRg = FALSE, CIs = TRUE,
  report = c("1", "2", "html"), returnStd = FALSE, extended = FALSE,
  zero.print = ".", ...)
```

Arguments

model	an mxModel to summarize
digits	round to how many digits (default = 2)
file	The name of the dot file to write: "name" = use the name of the model. Defaults to NA = do not create plot output
comparison	you can run mxCompare on a comparison model (NULL)
std	Whether to standardize the output (default = TRUE)
showRg	= whether to show the genetic correlations (FALSE)
CIs	Whether to show Confidence intervals if they exist (T)
report	If "html", then open an html table of the results
returnStd	Whether to return the standardized form of the model (default = FALSE)
extended	how much to report (FALSE)
zero.print	How to show zeros (".")
...	Other parameters to control model summary

Value

- optional mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

- umxACE

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umx_drop_ok, umx_standardize_RAM

Examples

```
require(umx)
data(twinData)
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
selDVs = c("bmi1", "bmi2")
mzData <- subset(twinData, ZYG == "MZFF", selDVs)
dzData <- subset(twinData, ZYG == "DZFF", selDVs)
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
m1 = umxRun(m1)
umxSummaryACE(m1)
```

```
## Not run:
umxSummaryACE(m1, file = NA);
umxSummaryACE(m1, file = "name", std = TRUE)
stdFit = umxSummaryACE(m1, returnStd = TRUE);

## End(Not run)
```

---

umxSummaryACEcov *umxSummaryACEcov*

Description

Summarise a Cholesky model as returned by umxACEcov

Usage

```
umxSummaryACEcov(model, digits = 2, file = getOption("umx_auto_plot"),
  returnStd = FALSE, extended = FALSE, showRg = FALSE, std = TRUE,
  comparison = NULL, CIs = TRUE, zero.print = ".", report = c("1", "2",
  "html"), ...)
```

Arguments

model	a umxACEcov model to summarize
digits	round to how many digits (default = 2)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
returnStd	Whether to return the standardized form of the model (default = FALSE)
extended	how much to report (FALSE)
showRg	= whether to show the genetic correlations (FALSE)
std	= whether to show the standardized model (TRUE)
comparison	you can run mxCompare on a comparison model (NULL)
CIs	Whether to show Confidence intervals if they exist (TRUE)
zero.print	How to show zeros (".")
report	If "html", then open an html table of the results.
...	Other parameters to control model summary

Value

- optional mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

- umxACEcov

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
selDVs = c("bmi1", "bmi2")
mzData <- subset(twinData, ZYG == "MZFF", selDVs)
dzData <- subset(twinData, ZYG == "DZFF", selDVs)
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
m1 = umxRun(m1)
umxSummaryACE(m1)
## Not run:
umxSummaryACE(m1, file = NA);
umxSummaryACE(m1, file = "name", std = TRUE)
stdFit = umxSummaryACE(m1, returnStd = TRUE);

## End(Not run)
```

---

umxSummaryCP *umxSummaryCP*

Description

Summarise a Commmon Pathway model, as returned by umxCP

Usage

```
umxSummaryCP(model, digits = 2, file = umx_set_auto_plot(silent = TRUE),
  returnStd = FALSE, extended = FALSE, showRg = TRUE, comparison = NULL,
  std = TRUE, CIs = FALSE, ...)
```

Arguments

model	A fitted umxCP model to summarize
digits	round to how many digits (default = 2)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
returnStd	Whether to return the standardized form of the model (default = FALSE)
extended	how much to report (FALSE)

showRg	Whether to show the genetic correlations (FALSE) (Not implemented!)
comparison	Whether to run mxCompare on a comparison model (NULL)
std	Whether to show the standardized model (TRUE) (ignored: used extended = TRUE to get unstandardized)
CIs	Confidence intervals (F)
...	Optional additional parameters

Value

- optional mxModel

References

- http://www.github.com/tbates/umx, http://tbates.github.io

See Also

- umxCP(), plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryGxE, umxSummaryIP, umx

Examples

```
require(umx)
data(twinData)
zygList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = zygList)
twinData$wt1 = twinData$wt1/10 # help CSOLNP by putting wt on a similar scale to ht
twinData$wt2 = twinData$wt2/10 # help CSOLNP by putting wt on a similar scale to ht
selDVs = c("ht", "wt")
mzData <- subset(twinData, ZYG == "MZFF", umx_paste_names(selDVs, "", 1:2))
dzData <- subset(twinData, ZYG == "DZFF", umx_paste_names(selDVs, "", 1:2))
m1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = "")
umxSummaryCP(m1, file = NA) # suppress plot creation with file
umxSummary(m1, file = NA) # generic summary is the same
stdFit = umxSummaryCP(m1, digits = 2, file = NA, returnStd = TRUE,
extended = FALSE, showRg = TRUE, std = TRUE, CIs = TRUE);
umxSummaryCP(m1, ext = TRUE, file = "name")
umxSummaryCP(m1, file = "Figure 3", std = TRUE)
```

umxSummaryGxE *umxSummaryGxE*

Description

Summarise a Moderation model, as returned by umxGxE

Usage

```
umxSummaryGxE(model = NULL, digits = 2, xlab = NA, location = "topleft",
  separateGraphs = FALSE, file = getOption("umx_auto_plot"),
  returnStd = NULL, std = NULL, reduce = FALSE, CIs = NULL,
  report = c("1", "2", "html"), ...)
```

Arguments

model	A fitted umxGxE model to summarize
digits	round to how many digits (default = 2)
xlab	label for the x-axis of plot
location	default = "topleft"
separateGraphs	default = F
file	The name of the dot file to write: NA = none; "name" = use the name of the model
returnStd	Whether to return the standardized form of the model (default = FALSE)
std	Whether to show the standardized model (not implemented! TRUE)
reduce	Whether run and tabulate a complete model reduction...(Defaults to FALSE)
CIs	Confidence intervals (FALSE)
report	"1" = regular, "2" = add descriptive sentences; "html" = open a browser and copyable tables
...	Optional additional parameters

Value

- optional mxModel

References

- https://github.com/tbates/umx, http://tbates.github.io

See Also

- umxGxE(), plot(), umxSummary() work for IP, CP, GxE, and ACE models.

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryIP, umx

Examples

```
# The total sample has been subdivided into a young cohort,
# aged 18-30 years, and an older cohort aged 31 and above.
# Cohort 1 Zygosity is coded as follows 1 == MZ females 2 == MZ males
# 3 == DZ females 4 == DZ males 5 == DZ opposite sex pairs
require(umx)
data(twinData)
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDVs, selDefs)
mzData  = subset(twinData, ZYG == "MZFF", selVars)
dzData  = subset(twinData, ZYG == "DZMM", selVars)
# Exclude cases with missing Def
mzData <- mzData[!is.na(mzData[selDefs[1]]) & !is.na(mzData[selDefs[2]]),]
dzData <- dzData[!is.na(dzData[selDefs[1]]) & !is.na(dzData[selDefs[2]]),]
m1 = umxGxE(selDVs = selDVs, selDefs = selDefs, dzData = dzData, mzData = mzData)
m1 = umxRun(m1)
# Plot Moderation
umxSummaryGxE(m1)
umxSummaryGxE(m1, location = "topright")
umxSummaryGxE(m1, separateGraphs = FALSE)
```

---

umxSummaryIP *umxSummaryIP*

Description

Summarise a Independent Pathway model, as returned by umxIP

Usage

```
umxSummaryIP(model, digits = 2, file = getOption("umx_auto_plot"),
  returnStd = FALSE, std = TRUE, showRg = TRUE, comparison = NULL,
  CIs = FALSE, ...)
```

Arguments

model	A fitted umxIP model to summarize
digits	round to how many digits (default = 2)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
returnStd	Whether to return the standardized form of the model (default = F)
std	= Whether to show the standardized model (TRUE)
showRg	= whether to show the genetic correlations (F)

comparison	Whether to run mxCompare on a comparison model (NULL)
CIs	Confidence intervals (F)
...	Optional additional parameters

Value

- optional mxModel

References

- http://github.com/tbates/umx, http://tbates.github.io

See Also

- umxIP(), plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.

Other Twin Modeling Functions: plot.MxModel, umxACESexLim, umxACEcov, umxACE, umxCF_SexLim, umxCP, umxGxE_window, umxGxE, umxIP, umxPlotACEcov, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxE, umx

Examples

```
require(umx)
data(twinData)
labList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = labList)
selDVs = c("ht1", "wt1", "ht2", "wt2")
mzData <- subset(twinData, ZYG == "MZFF")
dzData <- subset(twinData, ZYG == "DZFF")
m1 = umxIP(selDVs = selDVs, dzData = dzData, mzData = mzData)
m1 = umxRun(m1)
umxSummaryIP(m1)
plot(m1)
## Not run:
umxSummaryIP(m1, digits = 2, file = "Figure3", showRg = FALSE, CIs = TRUE);

## End(Not run)
```

---

umxThresholdMatrix *umxThresholdMatrix: Helper makes a complete threshold matrix needed in ordinal models.*

Description

High-level helper for ordinal modeling. Creates, labels, and sets smart-starts for this complex matrix. Big time saver!

Usage

```
umxThresholdMatrix(df, sep = NA, method = c("auto", "Mehta", "allFree"),
  thresholds = c("deviationBased", "direct", "ignore", "left_censored"),
  threshMatName = "threshMat", l_u_bound = c(NA, NA), droplevels = FALSE,
  suffixes = "deprecated", verbose = FALSE)
```

Arguments

df	the data being modelled (to allow access to the factor levels and quantiles within these for each variable)
sep	(optional) string for wide (twin) data, separating the base name from a numeric suffix (e.g. "_T")
method	How to set the thresholds: auto (the default), Mehta, which fixes the first two (auto chooses this for ordinal) or "allFree" (auto chooses this for binary)
thresholds	How to implement thresholds: "deviationBased" (default), "direct", "ignore", "left_censored"
threshMatName	name of the matrix which is returned. Defaults to "threshMat" - best not to change it.
l_u_bound	c(NA, NA) by default, you can use this to bound the thresholds. Careful you don't set bounds too close if you do.
droplevels	Whether to drop levels with no observed data (defaults to FALSE)
suffixes	deprecated. Instead of c("T1", "T2"), set sep (see above)
verbose	(defaults to FALSE))

Details

When modeling ordinal data (sex, low-med-hi, depressed/normal, not at all, rarely, often, always), a useful conceptual strategy to handle expectations is to build a standard-normal model (i.e., a latent model with zero-means, and unit (1.0) variances), and then to threshold this normal distribution to generate the observed data. Thus an observation of "depressed" is modeled as a high score on the latent normally distributed trait, with thresholds set so that only scores above this threshold (1-minus the number of categories).

For **deviation methods**, it returns a list of lowerOnes_for_thresh, deviations_for_thresh & thresholdsAlgebra (named threshMatName)

For **direct**, it returns a thresholdsMatrix (named threshMatName)

Value

- thresholds matrix

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxValues, umx_fix_first_loadings, umx_fix_latents, umx

Examples

```
x = data.frame(ordered(rbinom(100,1,.5))); names(x) <- c("x")
umxThresholdMatrix(x)
x = cut(rnorm(100), breaks = c(-Inf,.2,.5, .7, Inf)); levels(x) = 1:5
x = data.frame(ordered(x)); names(x) <- c("x")
tmp = umxThresholdMatrix(x)

# ==================
# = Binary example =
# ==================
require(umx)
if(!packageVersion("OpenMx") > 2.5){message("Update OpenMx to get a new version of twinData")}
data(twinData)
# ================================================================
# = Create a series of binary and ordinal columns to work with =
# ================================================================
# Cut to form category of 80 % obese subjects
selDVs = c("obese1", "obese2")
obesityLevels = c('normal', 'obese')
cutPoints <- quantile(twinData[, "bmi1"], probs = .2, na.rm = TRUE)
twinData$obese1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obese2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Step 2: Make the ordinal variables into mxFactors
# this ensures ordered= TRUE + requires user to confirm levels
twinData[, selDVs] <- mxFactor(twinData[, selDVs], levels = obesityLevels)

# Repeat for three-level weight variable
selDVs = c("obeseTri1", "obeseTri2")
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints <- quantile(twinData[, "bmi1"], probs = c(.4, .7), na.rm = TRUE)
twinData$obeseTri1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obeseTri2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData[, selDVs] <- mxFactor(twinData[, selDVs], levels = obesityLevels)

selDVs = c("obeseQuad1", "obeseQuad2")
obesityLevels = c('underWeight', 'normal', 'overweight', 'obese')
cutPoints <- quantile(twinData[, "bmi1"], probs = c(.25, .4, .7), na.rm = TRUE)
twinData$obeseQuad1 <- cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obeseQuad2 <- cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData[, selDVs] <- mxFactor(twinData[, selDVs], levels = obesityLevels)

# Example 1
selDVs = c("obese1", "obese2")
tmp = umxThresholdMatrix(twinData[,selDVs], sep = "", verbose = TRUE) # informative messages

# ====================================
# = Ordinal (n categories > 2) example =
```

```
# ====================================
selDVs = c("obeseTri1", "obeseTri2")
tmp = umxThresholdMatrix(twinData[,selDVs], sep = "", verbose = TRUE)

# ========================================================
# = Mix of all three kinds example (and a 4-level trait) =
# ========================================================

selDVs = umx_paste_names(c("bmi", "obese", "obeseTri", "obeseQuad"), "", 1:2)
tmp = umxThresholdMatrix(twinData[,selDVs], sep = "", verbose = TRUE)
str(tmp)

# ===================
# = "left_censored" =
# ===================

x = round(10 * rnorm(1000, mean = -.2))
y = round(5 * rnorm(1000))
x[x < 0] = 0; y[y < 0] = 0
df  = data.frame(x1 = x, x2 = y)
df  = umxFactor(df); #str(df)
table(df)
tmp = umxThresholdMatrix(df, thresholds = "left_censored"); class(tmp)
any(tmp$free) # all fixed.
tmp$labels
df  = umxFactor(data.frame(x_T1 = x, x_T2 = y), sep="_T"); #str(df)
tmp = umxThresholdMatrix(df, sep="_T", thresholds = "left_censored"); class(tmp)
any(tmp$free) # all fixed.
tmp$labels
```

---

umxTwoStage *umxTwoStage*

Description

umxTwoStage implements 2-stage least squares regression in Structural Equation Modeling. For ease of learning, the function is modeled closely on the tsls.

Usage

```
umxTwoStage(formula, instruments, data, subset, weights, contrasts = NULL,
  name = "tsls", ...)
```

Arguments

formula	for structural equation to be estimated; a regression constant is implied if not explicitly omitted.
instruments	one-sided formula specifying instrumental variables.
data	data.frame containing the variables in the model.

subset	[optional] vector specifying a subset of observations to be used in fitting the model.
weights	[optional] vector of weights to be used in the fitting process; if specified should be a non-negative numeric vector with one entry for each observation, to be used to compute weighted 2SLS estimates.
contrasts	an optional list. See the contrasts.arg argument of model.matrix.default.
name	for the model (defaults to "tsls")
...	arguments to be passed down.

Details

The example is a Mendelian Randomization https://en.wikipedia.org/wiki/Mendelian_randomization analysis to show the value of two-stage.

Value

-

References

- Fox, J. (1979) Simultaneous equation models and two-stage least-squares. In Schuessler, K. F. (ed.) *Sociological Methodology*, Jossey-Bass., Greene, W. H. (1993) *Econometric Analysis*, Second Edition, Macmillan.

See Also

- umxRAM, tsls

Other Super-easy helpers: umxEFA, umx

Examples

```
library(umx)


# ====================================
# = Mendelian randomization analysis =
# ====================================

# Note: in practice: many more subjects are desireable - this just to let example run fast
df = umx_make_MR_data(1000)
m1 = umxTwoStage(Y ~ X, instruments = ~ qtl, data = df)
coef(m1)
plot(m1)

# Errant analysis using ordinary least squares regression (WARNING this result is CONFOUNDED!!)
m1 = lm(Y ~ X    , data = df); coef(m1) # incorrect .35 effect of X on Y
m1 = lm(Y ~ X + U, data = df); coef(m1) # Controlling U reveals the true 0.1 beta weight
#
#
## Not run:
```

```
df = umx_make_MR_data(1e5)
m1 = umxTwoStage(Y ~ X, instruments = ~ qtl, data = df)

# ======================
# = now with sem::tsls =
# ======================
# library(sem) # will require you to install X11
m2 = sem::tsls(formula = Y ~ X, instruments = ~ qtl, data = df)
coef(m1)
coef(m2)
m3 = tsls(formula = Y ~ X, instruments = ~ qtl, data = (df[1,"qtl"] = NA))

## End(Not run)
```

---

umxUnexplainedCausalNexus *umxUnexplainedCausalNexus*

Description

umxUnexplainedCausalNexus report the effect of a change (delta) in a variable (from) on an output (to)

Usage

```
umxUnexplainedCausalNexus(from, delta, to, model)
```

Arguments

from	A variable in the model that you want to imput the effect of a change
delta	A the amount to simulate changing \"from\" by.
to	The dependent variable that you want to watch changing
model	The model containing from and to

References

- http://www.github.com/tbates/umx/

See Also

- umxRun, mxCompare

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxFixAll, umxMI, umxSetParameters, umx

Examples

```
## Not run:
umxUnexplainedCausalNexus(from="yrsEd", delta = .5, to = "income35", model)

## End(Not run)
```

---

umxValues *umxValues: Set values in RAM model, matrix, or path*

Description

For models to be estimated, it is essential that path values start at credible values. umxValues takes on that task for you. umxValues can set start values for the free parameters in both RAM and Matrix mxModels. It can also take an mxMatrix as input. It tries to be smart in guessing starts from the values in your data and the model type. note: If you give it a numeric input, it will use obj as the mean, return a list of length n, with sd = sd

Usage

```
umxValues(obj = NA, sd = NA, n = 1, onlyTouchZeros = FALSE)
```

Arguments

obj	The RAM or matrix mxModel, or mxMatrix that you want to set start values for.
sd	Optional Standard Deviation for start values
n	Optional Mean for start values
onlyTouchZeros	Don't alter parameters that appear to have already been started (useful for speeding umxModify)

Value

- mxModel with updated start values

References

- http://www.github.com/tbates/umx, https://tbates.github.io

See Also

- Core functions:

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umx_fix_first_loadings, umx_fix_latents, umx

Examples

```
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
mxEval(S, m1) # default variances are 0
m1 = umxValues(m1)
mxEval(S, m1) # plausible variances
umx_print(mxEval(S,m1), 3, zero.print = ".") # plausible variances
umxValues(14, sd = 1, n = 10) # Return vector of length 10, with mean 14 and sd 1
# todo: handle complex guided matrix value starts...
```

---

umxVersion *Print the version of umx, along with detail from OpenMx and general system info.*

Description

umxVersion returns the version information for umx, and for OpenMx and R. essential for bug-reports.

Usage

```
umxVersion(model = NULL, verbose = TRUE, return = "umx")
```

Arguments

model	Optional to show optimizer in this model
verbose	= TRUE
return	Which package (umx or openMx to return version on

Value

- mxModel

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- packageVersion

Other Miscellaneous Utility Functions: umx_make, umx_write_to_clipboard

Examples

```
x = umxVersion(); x
```

---

umx_add_variances *umx_add_variances*

Description

Convenience function to save the user specifying mxPaths adding variance to each variable

Usage

```
umx_add_variances(model, add.to, values = NULL, free = NULL)
```

Arguments

model	an mxModel to add variances to
add.to	= List of variables to create variance for
values	= List of values (default = NULL)
free	= List of variables to create variance for (default = NULL)

Value

- mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Misc: umxEval, umx_APA_model_CI, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
require(umx)
data(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
 manifestVars = names(demoOneFactor),
 latentVars = "g",
mxPath(from = "g", to = names(demoOneFactor), values= .1),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umx_show(m1, matrices = "S") # variables lack variance :-(
m1 = umx_add_variances(m1, add.to = names(demoOneFactor))
m1 = umx_add_variances(m1, add.to = "g", FALSE, 1)
umx_show(m1, matrices = "S")
# Note: latent g has been treated like the manifests...
# umxFixLatents() will take care of this for you...
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umxSummary(m1)
```

---

umx_aggregate *umx_aggregate*

Description

R's built-in aggregate function is extremely useful and powerful, allowing xtabs based on a formula. umx_aggregate just tries to make using it a bit easier. In particular, it has some handy base functions that simplify the task of summarising data aggregating over some grouping factor. A common use is preparing summary tables.

Usage

```
umx_aggregate(formula = DV ~ condition, data = NA, what = c("mean_sd",
  "n"), digits = 2, kable = TRUE)
```

Arguments

formula	The aggregation formula. e.g., DV ~ condition
data	frame to aggregate
what	function to use. Defaults to a built-in "smart" mean (sd)
digits	for rounding of results
kable	Report as a formatted table? (Default is TRUE)

Value

- table

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umx_apply, aggregate

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_print, umx_show, umx_time, umx

Examples

```
aggregate(mpg ~ cyl, FUN = mean, na.rm = TRUE, data = mtcars)
umx_aggregate(mpg ~ cyl, data = mtcars)
umx_aggregate(mpg ~ cyl, data = mtcars, kable = FALSE)
umx_aggregate(cbind(mpg, qsec) ~ cyl, data = mtcars, digits = 3)
t(umx_aggregate(cbind(mpg, qsec) ~ cyl, data = mtcars))
## Not run:
umx_aggregate(cbind(moodAvg, mood) ~ condition, data = study1)

## End(Not run)
```

---

umx_APA_model_CI *umx_APA_model_CI*

Description

Look up CIs for free parameters in a model, and return as APA-formatted text string

Usage

```
umx_APA_model_CI(model, cellLabel, prefix = "top.", suffix = "_std",
  digits = 2, verbose = FALSE)
```

Arguments

model	an mxModel to get CIs from
cellLabel	the label of the cell to interogate for a CI, e.g. "ai_r1c1"
prefix	This submodel to look in (i.e. "top.")
suffix	The suffix for algebras ("_std")
digits	= 2
verbose	= FALSE

Value

- the CI string, e.g. ".73 [-.2, .98]"

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Misc: umxEval, umx_add_variances, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
## Not run:
umx_APA_model_CI(fit_IP, cellLabel = "ai_r1c1", prefix = "top.", suffix = "_std")

## End(Not run)
```

---

umx_APA_pval *umx_APA_pval*

Description

round a p value so you get < .001 instead of .000000002 or 1.00E-09

Usage

```
umx_APA_pval(p, min = 0.001, digits = 3, addComparison = NA,
  rounding = NULL)
```

Arguments

p	The p-value to round
min	Values below min will be reported as "< min"
digits	Number of decimals to which to round (default = 3)
addComparison	Whether to add '=' '<' etc. (NA adds when needed)
rounding	deprecated - please replace 'rounding' with 'digits'

Value

- p-value formatted in APA style

See Also

- round

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_aggregate, umx_print, umx_show, umx_time, umx

Examples

```
umx_APA_pval(.052347)
umx_APA_pval(1.23E-3)
umx_APA_pval(1.23E-4)
umx_APA_pval(c(1.23E-3, .5))
umx_APA_pval(c(1.23E-3, .5), addComparison = TRUE)
```

umx_apply *umx_apply*

Description

Tries to make apply more readable. Other functions to think of include cumsum, rowSums, colMeans, etc.

Usage

```
umx_apply(FUN, of, by = "columns", ...)
```

Arguments

FUN	The function to apply
of	The dataframe to work with
by	What to apply the function to: columns or rows (default = "columns")
...	optional arguments to FUN, i.e., na.rm = T

Value

- object

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

- umx_aggregate

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
umx_apply(mean, mtcars, by = "columns")
umx_apply(mean, of = mtcars, by = "columns")
umx_apply(mean, by = "rows", of = mtcars[1:3,], na.rm = TRUE)
```

umx_as_numeric *umx_as_numeric*

Description

Convert each column of a dataframe to numeric

Usage

```
umx_as_numeric(df, force = FALSE)
```

Arguments

df	a data.frame to convert
force	whether to force conversion to numeric for non-numeric columns (defaults to FALSE)

Value

- data.frame

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
df = mtcars
# make one variable non-numeric
df$mpg = c(letters,letters[1:6]); str(df)
df = umx_as_numeric(df)
```

umx_check *umx_check*

Description

Check that a test evaluates to TRUE. If not, stop, warn, or message the user

Usage

```
umx_check(boolean.test, action = c("stop", "warning", "message"),
  message = "check failed")
```

Arguments

boolean.test	test evaluating to TRUE or FALSE
action	One of "stop" (the default), "warning", or "message"
message	what to tell the user when boolean.test is FALSE

Value

- boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
umx_check(length(1:3)==3, "stop", "item must have length == 3")
```

umx_check_model *umx_check_model*

Description

Check an OpenMx model

Usage

```
umx_check_model(obj, type = NULL, hasData = NULL, beenRun = NULL,
  hasMeans = NULL, checkSubmodels = FALSE)
```

Arguments

obj	an object to check
type	what type the model must be, i.e., "RAM", "LISREL", etc. (defaults to not checking NULL)
hasData	whether the model should have data or not (defaults to not checking NULL)
beenRun	whether the model has been run or not (defaults to not checking NULL)
hasMeans	whether the model should have a means model or not (defaults to not checking NULL)
checkSubmodels	whether to check submodels (not implemented yet) (default = FALSE)

Value

- boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
```

```
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umx_check_model(m1)
umx_check_model(m1, type = "RAM") # equivalent to umx_is_RAM()
umx_check_model(m1, hasData = TRUE)
## Not run:
umx_check_model(m1, hasMeans = TRUE)
umx_check_model(m1, beenRun = FALSE)

## End(Not run)
```

---

umx_check_names *umx_check_names*

Description

Check if a list of names are in the names() of a dataframe (or the of a matrix)

Usage

```
umx_check_names(namesNeeded, data = NA, die = TRUE, no_others = FALSE,
  intersection = FALSE, message = "")
```

Arguments

namesNeeded	list of variable names to find (a dataframe is also allowed)
data	data.frame (or matrix) to search in for names (default = NA)
die	whether to die if the check fails (defaults to TRUE)
no_others	Whether to test that the data contain no columns in addition to those in namesNeeded (defaults to FALSE)
intersection	Show the intersection of names
message	Some helpful text to append when dieing.

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Other Building Functions: umx_var

Examples

```
require(umx)
data(demoOneFactor) # "x1" "x2" "x3" "x4" "x5"
umx_check_names(c("x1", "x2"), demoOneFactor)
umx_check_names(c("x1", "x2"), as.matrix(demoOneFactor))
umx_check_names(c("x1", "x2"), cov(demoOneFactor[, c("x1","x2")]))
umx_check_names(c("z1", "x2"), data = demoOneFactor, die = FALSE)
umx_check_names(c("x1", "x2"), data = demoOneFactor, die = FALSE, no_others = TRUE)
umx_check_names(c("x1","x2","x3","x4","x5"), data = demoOneFactor, die = FALSE, no_others = TRUE)
## Not run:
umx_check_names(c("bad_var_name", "x2"), data = demoOneFactor, die = TRUE)

## End(Not run)
```

---

umx_check_OS *umx_check_OS*

Description

Check what OS we are running on (current default is OS X). Returns a boolean. Optionally warn or die on failure of the test

Usage

```
umx_check_OS(target = c("OSX", "SunOS", "Linux", "Windows"),
  action = c("ignore", "warn", "die"))
```

Arguments

target	Which OS(s) you wish to check for (default = "OSX")
action	What to do on failure of the test: nothing (default), warn or die

Value

- TRUE if on the specified OS (else FALSE)

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Test: umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
umx_check_OS()
```

umx_check_parallel *umx_check_parallel*

Description

Shows how many cores you are using, and runs a test script so user can check CPU usage

Usage

```
umx_check_parallel(nCores = -1, testScript = NULL, rowwiseParallel = TRUE,
  nSubjects = 1000)
```

Arguments

nCores	How many cores to run (defaults to -1 (all available))
testScript	A user-provided script to run (NULL)
rowwiseParallel	Whether to parallel-ize rows (default) or gradient computation
nSubjects	Number of rows to model (Default = 1000) Reduce for quicker runs.

Value

- NULL

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
## Not run:
# On a fast machine, takes a minute with 1 core
umx_check_parallel()

## End(Not run)
```

umx_cont_2_quantiles *umx_cont_2_quantiles*

Description

Recode a continuous variable into n-quantiles (default = deciles (10 levels)). It returns an mxFactor, with the levels labeled with the max value in each quantile (i.e., open on the left-side). quantiles are labeld "quantile1" "quantile2" etc.

Usage

```
umx_cont_2_quantiles(x, nlevels = NULL, type = c("mxFactor", "ordered",
  "unordered"), verbose = FALSE, returnCutpoints = FALSE)
```

Arguments

x	a variable to recode as ordinal (email me if you'd like this upgraded to handle df input)
nlevels	How many bins or levels (at most) to use (i.e., 10 = deciles)
type	what to return (Default is "mxFactor") options: "ordered" and "unordered")
verbose	report the min, max, and decile cuts used (default = FALSE)
returnCutpoints	just return the cutpoints, for use directly

Details

Note: Redundant quantiles are merged. i.e., if the same score identifies all deciles up to the fourth, then these will be merged into one bin, labeled "quantile4".

Value

- recoded variable as an mxFactor

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
x = umx_cont_2_quantiles(rnorm(1000), nlevels = 10, verbose = TRUE)
x = data.frame(x)
str(x); levels(x)
table(x)
## Not run:
ggplot2::qplot(x$x)
y = mxDataWLS(x, type = "WLS")

## End(Not run)

# ============================
# = Use with twin variables =
# ============================

x = twinData
y = rbind(x$wt1, x$wt2)
cuts  = umx_cont_2_quantiles(y, nlevels = 10, returnCutpoints = TRUE)
x$wt1 = umx_cont_2_quantiles(x$wt1, nlevels = cuts) # use same for both...
x$wt2 = umx_cont_2_quantiles(x$wt2, nlevels = cuts) # use same for both...
str(x[, c("wt1", "wt2")])

# More examples

x = umx_cont_2_quantiles(mtcars[, "mpg"], nlevels = 5) # quintiles
x = umx2ord(mtcars[, "mpg"], nlevels = 5) # using shorter alias
x = umx_cont_2_quantiles(mtcars[, "cyl"], nlevels = 10) # more than integers exist
x = umx_cont_2_quantiles(rbinom(10000, 1, .5), nlevels = 2)
```

---

umx_cor *umx_cor*

Description

Report correlations and their p-values

Usage

```
umx_cor(X, df = nrow(X) - 2, use = c("pairwise.complete.obs",
  "complete.obs", "everything", "all.obs", "na.or.complete"), digits = 2,
  type = c("r and p-value", "smart"))
```

Arguments

X	a matrix or dataframe
df	the degrees of freedom for the test
use	how to handle missing data (defaults to pairwise complete)
digits	rounding of answers
type	Unused argument for future directions

Value

- Matrix of correlations and p-values

References

- http://www.github.com/tbates/umx

See Also

Other Stats Functions: reliability, umxCov2cor, umx_means, umx

Examples

```
umx_cor(myFADataRaw[1:8,])
```

---

umx_cov2raw *Turn a cov matrix into raw data with umx_cov2raw*

Description

Turns a covariance matrix into comparable raw data :-)

Usage

```
umx_cov2raw(myCovariance, n, means = 0)
```

Arguments

myCovariance	a covariance matrix
n	how many rows of data to return
means	the means of the raw data (defaults to 0)

Value

- data.frame

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

- cov2cor

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
covData <- matrix(nrow=6, ncol=6, byrow=TRUE, dimnames=list(paste0("v", 1:6), paste0("v", 1:6)),
data = c(0.9223099, 0.1862938, 0.4374359, 0.8959973, 0.9928430, 0.5320662,
           0.1862938, 0.2889364, 0.3927790, 0.3321639, 0.3371594, 0.4476898,
           0.4374359, 0.3927790, 1.0069552, 0.6918755, 0.7482155, 0.9013952,
           0.8959973, 0.3321639, 0.6918755, 1.8059956, 1.6142005, 0.8040448,
           0.9928430, 0.3371594, 0.7482155, 1.6142005, 1.9223567, 0.8777786,
           0.5320662, 0.4476898, 0.9013952, 0.8040448, 0.8777786, 1.3997558))
myData = umx_cov2raw(covData, n = 100, means = 1:6)
```

---

umx_default_option *umx_default_option*

Description

Handle parameter options given as a default list in a function. This is just a version of x = match.arg(x) which allows items not in the list.

Usage

```
umx_default_option(x, option_list, check = TRUE)
```

Arguments

x	the value chosen (may be the default option list)
option_list	A vector of valid options
check	Whether to check that single items are in the list. Set false to accept abbreviations (defaults to TRUE)

Value

- one validated option

References

- http://www.github.com/tbates/umx

See Also

- match.arg

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_apply, umx_get_bracket_addresses, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
## Not run:
option_list = c("default", "par.observed", "empirical")
umx_default_option("par.observed", option_list)
umx_default_option("bad", option_list)
umx_default_option("allow me", option_list, check = FALSE)
umx_default_option(option_list, option_list)
option_list = c(NULL, "par.observed", "empirical")
umx_default_option(option_list, option_list) # fails with NULL!!!!!
option_list = c(NA, "par.observed", "empirical")
umx_default_option(option_list, option_list) # use NA instead
option_list = c(TRUE, FALSE, NA)
umx_default_option(option_list, option_list) # works with non character

## End(Not run)
```

---

umx_drop_ok *umx_drop_ok*

Description

Print a meaningful sentence about a model comparison. If you use this, please email me and ask to have it merged with umxCompare() :-)

Usage

```
umx_drop_ok(model1, model2, text = "parameter")
```

Arguments

model1	the base codemxModel
model2	the nested codemxModel
text	name of the thing being tested, i.e., "Extraversion" or "variances"

Value

-

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_standardize_RAM

Examples

```
require(umx)
data(demoOneFactor)
latents   = c("g")
manifests = names(demoOneFactor)
myData    = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
m1 <- umxRAM("OneFactor", data = myData,
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
m2 = umxModify(m1, update = "g_to_x1", name = "no effect on x1")
umx_drop_ok(m1, m2, text = "the path to x1")
```

---

umx_explode *umx_explode - like the php function 'explode'*

Description

Takes a string and returns an array of delimtted strings (by default, each character)

Usage

```
umx_explode(delimiter = character(), string)
```

Arguments

delimiter	what to break the string on. Default is empty string ""
string	an character string, e.g. "dog"

Value

- a vector of strings, e.g. c("d", "o", "g")

References

- http://tbates.github.io, https://github.com/tbates/umx, http://www.php.net/

See Also

Other String Functions: umx_explode_twin_names, umx_rot, umx_trim

Examples

```
umx_explode("", "dog") # "d" "o" "g"
umx_explode(" ", "cats and dogs") # [1] "cats" "and"  "dogs"
```

umx_explode_twin_names *umx_explode_twin_names*

Description

Break names like Dep_T1 into a list of base names, a seperator, and a vector of twin indexes. e.g. c("Dep_T1", "Dep_T2") -> list(varnames = c("Dep"), sep = "_T", twinIndexes = c(1,2))

Usage

```
umx_explode_twin_names(df, sep = "_T")
```

Arguments

df	vector of names or data.frame containing the data
sep	text constant separating name from numeric 1:2 twin index

Value

- list(varnames = c("Dep"), sep = "_T", twinIndexes = c(1,2))

See Also

Other String Functions: umx_explode, umx_rot, umx_trim

Examples

```
require(umx)
data("twinData")
umx_explode_twin_names(twinData, sep = "")
# Single-character single variable test case
x = round(10 * rnorm(1000, mean = -.2))
y = round(5 * rnorm(1000))
x[x < 0] = 0; y[y < 0] = 0
umx_explode_twin_names(data.frame(x_T1 = x, x_T2 = y), sep = "_T")
umx_explode_twin_names(data.frame(x_T11 = x, x_T22 = y), sep = "_T")
```

umx_find_object *umx_find_object*

Description

Find objects a certain class, whose name matches a search string. The string (pattern) is grep-enabled, so you can match wild-cards

Usage

```
umx_find_object(pattern = ".*", requiredClass = "MxModel")
```

Arguments

pattern	the pattern that matching objects must contain
requiredClass	the class of object that will be matched

Value

- a list of objects matching the class and name

References

-

See Also

Other Utility Functions: qm, umx_grep, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
## Not run:
umx_find_object("^m[0-9]") # mxModels beginning "m1" etc.
umx_find_object("", "MxModel") # all MxModels

## End(Not run)
```

umx_fix_first_loadings *umx_fix_first_loadings*

Description

Fix the loading of the first path from each latent at selected value (default = 1).

Usage

```
umx_fix_first_loadings(model, latents = NULL, at = 1)
```

Arguments

model	an mxModel to set
latents	(If NULL then all latentVars in model)
at	(Default = 1)

Value

- mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_latents, umx

Examples

```
require(umx)
data(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
 manifestVars = names(demoOneFactor),
 latentVars = "g",
mxPath(from = "g", to = names(demoOneFactor)),
mxPath(from = names(demoOneFactor), arrows = 2),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umx_fix_first_loadings(m1)
umx_show(m1) # variance of g is fixed at 1
```

umx_fix_latents *umx_fix_latents*

Description

Fix the variance of all, or selected, exogenous latents at selected values. This function adds a variance to the factor if it does not exist.

Usage

```
umx_fix_latents(model, latents = NULL, exogenous.only = TRUE, at = 1)
```

Arguments

model	an mxModel to set
latents	(If NULL then all latentVars)
exogenous.only	only touch exogenous latents (default = TRUE)
at	(Default = 1)

Value

- mxModel

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx

Examples

```
require(umx)
data(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
 manifestVars = names(demoOneFactor),
 latentVars = "g",
mxPath(from = "g", to = names(demoOneFactor)),
mxPath(from = names(demoOneFactor), arrows = 2),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umx_show(m1, matrices = "S") # variance of g is not set
m1 = umx_fix_latents(m1)
umx_show(m1, matrices = "S") # variance of g is fixed at 1
```

umx_fun_mean_sd *umx_fun*

Description

Misellaneous functions that are handy in summary and other tasks where you might otherwise have to craft a custom nameless funtions. e.g.

Usage

```
umx_fun_mean_sd(x, na.rm = TRUE, digits = 2)
```

Arguments

x	input
na.rm	How to handle missing (default = TRUE = remove)
digits	Rounding (default = 2)

Details

- umx_fun_mean_sd: returns "mean (SD)" of x.
- Second item

note: if a factor is given, then the mode is returned instead of the mean and SD.

Value

- function result

References

- https://github.com/tbates/umx, https://tbates.github.io

Examples

```
umxAPA(mtcars[,1:3]) # uses umx_fun_mean_sd
```

umx_get_bracket_addresses *get mat[r,c] style cell address from an mxMatrix*

Description

Sometimes you want these :-) This also allows you to change the matrix name: useful for using mxMatrix addresses in an mxAlgebra.

Usage

```
umx_get_bracket_addresses(mat, free = NA, newName = NA)
```

Arguments

mat	an mxMatrix to get address labels from
free	how to filter on free (default = NA: take all)
newName	= NA

Value

- a list of bracket style labels

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_apply, umx_default_option, umx_object_as_str, umx_string_to_algebra, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umx_get_bracket_addresses(m1$matrices$A, free= TRUE)
```

umx_get_checkpoint *umx_get_checkpoint*

Description

get the checkpoint status for a model or global options

Usage

```
umx_get_checkpoint(model = NULL)
```

Arguments

model	an optional model to get options from

Value

- NULL

References

- http://tbates.github.io

See Also

Other Get and set: umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
umx_get_checkpoint() # current global default
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
m1 = umx_set_checkpoint(interval = 2, model = m1)
umx_get_checkpoint(model = m1)
```

umx_get_cores *umx_get_cores*

Description

This function is now deprecated: Get the number of cores using \codeumx_set_cores with no parameters.

Usage

```
umx_get_cores(model = NULL)
```

Arguments

model	an (optional) model to get from. If left NULL, the global option is returned

Value

- number of cores

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other umx deprecated: umx-deprecated, umx_get_optimizer

Examples

```
# Deprecated function: to get cores, use umx_set_cores() with no value
```

umx_get_optimizer *umx_get_optimizer*

Description

This function is now deprecated: Get the current optimizer, use \codeumx_set_optimizer with no parameters.

Usage

```
umx_get_optimizer(model = NULL)
```

Arguments

model	(optional) model to get from. If left NULL, the global option is returned

Value

- the optimizer - a string

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other umx deprecated: umx-deprecated, umx_get_cores

Examples

```
# Deprecated function: to get cores, use umx_set_cores() with no value
```

---

umx_get_options *umx_get_options*

Description

Show the umx options. Useful for beginners to discover, or people like me to remember :-)

Usage

```
umx_get_options()
```

Value

- message

See Also

Other Miscellaneous Functions: install.OpenMx, umx_open_CRAN_page, umx_pad

Examples

```
umx_get_options()
```

umx_grep *umx_grep*

Description

Search for text. Will search names if given a data.frame, or strings if given a vector of strings. NOTE: Handy feature is that this can search the labels of data imported from SPSS

Usage

```
umx_grep(df, grepString, output = c("both", "label", "name"),
  ignore.case = TRUE, useNames = FALSE)
```

Arguments

df	The data.frame or string to search
grepString	the search string
output	the column name, the label, or both (default)
ignore.case	whether to be case sensitive or not (default TRUE = ignore case)
useNames	whether to search the names as well as the labels (for SPSS files with label metadata)

Details

To simply grep for a pattern in a string just use R built-in grep* functions, e.g.: grepl("^NA\[0-9]", "NA.3")

Value

- list of matched column names and/or labels

References

- http://www.github.com/tbates/umx

See Also

- grep umx_aggregate

Other Utility Functions: qm, umx_find_object, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
umx_grep(mtcars, "hp", output="both", ignore.case= TRUE)
umx_grep(c("hp", "ph"), "hp")
umx_grep(mtcars, "^h.*", output="both", ignore.case= TRUE)
## Not run:
umx_grep(spss_df, "labeltext", output = "label")
umx_grep(spss_df, "labeltext", output = "name")

## End(Not run)
```

---

umx_has_been_run *umx_has_been_run*

Description

check if an mxModel has been run or not

Usage

```
umx_has_been_run(model, stop = FALSE)
```

Arguments

model	The mxModel you want to check has been run
stop	Whether to stop if the model has not been run (defaults to FALSE)

Value

- boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
```

```
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umx_has_been_run(m1)
```

---

umx_has_CIs *umx_has_CIs*

Description

A utility function to return a binary answer to the question "does this mxModel have confidence intervals?"

Usage

```
umx_has_CIs(model, check = c("both", "intervals", "output"))
```

Arguments

model	The mxModel to check for presence of CIs
check	What to check for: "intervals" requested, "output" present, or "both". Defaults to "both"

Value

- TRUE or FALSE

References

- http://www.github.com/tbates/umx/

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
```

```
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umx_has_CIs(m1) # FALSE: no CIs and no output
m1 = mxModel(m1, mxCI("G_to_x1"))
umx_has_CIs(m1, check = "intervals") # TRUE intervals set
umx_has_CIs(m1, check = "output")  # FALSE not yet run
m1 = mxRun(m1)
umx_has_CIs(m1, check = "output")  # Still FALSE: Set and Run
m1 = mxRun(m1, intervals = TRUE)
umx_has_CIs(m1, check = "output")  # TRUE: Set, and Run with intervals = T
```

`umx_has_means` *umx_has_means*

Description

A utility function to return a binary answer to the question "does this `mxModel` have a means model?"

Usage

```
umx_has_means(model)
```

Arguments

`model`	The `mxModel` to check for presence of means

Value

- TRUE or FALSE

References

- http://www.github.com/tbates/umx/

See Also

Other Test: `umx_check_OS`, `umx_check_model`, `umx_check_names`, `umx_check_parallel`, `umx_check`, `umx_has_CIs`, `umx_has_been_run`, `umx_has_square_brackets`, `umx_is_MxData`, `umx_is_MxMatrix`, `umx_is_MxModel`, `umx_is_RAM`, `umx_is_cov`, `umx_is_endogenous`, `umx_is_exogenous`, `umx_is_ordered`

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
umx_has_means(m1)
m1 <- mxModel(m1,
mxPath(from = "one", to = manifests),
mxData(demoOneFactor[1:100,], type = "raw")
)
umx_has_means(m1)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umx_has_means(m1)
```

---

umx_has_square_brackets

umx_has_square_brackets

Description

Helper function, checking if a label has sqaure brackets

Usage

```
umx_has_square_brackets(input)
```

Arguments

input	The label to check for square brackets (string input)

Value

- boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
umx_has_square_brackets("[hello]")
umx_has_square_brackets("goodbye")
```

---

umx_is_cov *umx_is_cov*

Description

test if a data frame or matrix is cov or cor data, or is likely to be raw...

Usage

```
umx_is_cov(data = NULL, boolean = FALSE, verbose = FALSE)
```

Arguments

data	dataframe to test
boolean	whether to return the type ("cov") or a boolean (default = string)
verbose	How much feedback to give (default = FALSE)

Value

- "raw", "cor", or "cov", or, if boolean= T, then T | F

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
df = cov(mtcars)
umx_is_cov(df)
df = cor(mtcars)
umx_is_cov(df)
umx_is_cov(df, boolean = TRUE)
umx_is_cov(mtcars, boolean = TRUE)
```

umx_is_endogenous *umx_is_endogenous*

Description

Return a list of all the endogenous variables (variables with at least one incoming single-arrow path) in a model.

Usage

```
umx_is_endogenous(model, manifests_only = TRUE)
```

Arguments

model	an mxModel from which to get endogenous variables
manifests_only	Whether to check only manifests (default = TRUE)

Value

- list of endogenous variables

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_exogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath("g", to = names(demoOneFactor)),
umxPath(var = "g", fixedAt = 1),
umxPath(var = names(demoOneFactor))
)
umx_is_endogenous(m1, manifests_only = TRUE)
umx_is_endogenous(m1, manifests_only = FALSE)
```

umx_is_exogenous *umx_is_exogenous*

Description

Return a list of all the exogenous variables (variables with no incoming single-arrow path) in a model.

Usage

```
umx_is_exogenous(model, manifests_only = TRUE)
```

Arguments

model	an mxModel from which to get exogenous variables
manifests_only	Whether to check only manifests (default = TRUE)

Value

- list of exogenous variables

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath("g", to = names(demoOneFactor)),
umxPath(var = "g", fixedAt = 1),
umxPath(var = names(demoOneFactor))
)
umx_is_exogenous(m1, manifests_only = TRUE)
umx_is_exogenous(m1, manifests_only = FALSE)
```

umx_is_MxData *Check if an object is an mxData object*

Description

Is the input an MxData?

Usage

```
umx_is_MxData(x)
```

Arguments

x	An object to test for being an MxData object

Value

- Boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
umx_is_MxData(mtcars)
umx_is_MxData(mxData(mtcars, type= "raw"))
umx_is_MxData(mxData(cov(mtcars), type= "cov", numObs = 73))
```

umx_is_MxMatrix *umx_is_MxMatrix*

Description

Utility function returning a binary answer to the question "Is this an OpenMx mxMatrix?"

Usage

```
umx_is_MxMatrix(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx mxMatrix

Value

- Boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
x = mxMatrix(name = "eg", type = "Full", nrow = 3, ncol = 3, values = .3)
if(umx_is_MxMatrix(x)){
message("nice OpenMx matrix!")
}
```

---

umx_is_MxModel *umx_is_MxModel*

Description

Utility function returning a binary answer to the question "Is this an OpenMx model?"

Usage

```
umx_is_MxModel(obj, listOK = FALSE)
```

Arguments

obj	An object to be tested to see if it is an OpenMx mxModel
listOK	Is it acceptable to pass in a list of models? (Default = FALSE)

Value

- Boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
m1 = mxModel("test")
if(umx_is_MxModel(m1)){
message("nice OpenMx model!")
}
if(umx_is_MxModel(list(m1,m1), listOK = TRUE)){
message("nice list of OpenMx models!")
}
```

---

umx_is_ordered *Test if one or more variables in a dataframe are ordered*

Description

Return the names of any ordinal variables in a dataframe

Usage

```
umx_is_ordered(df, names = FALSE, strict = TRUE, binary.only = FALSE,
  ordinal.only = FALSE, continuous.only = FALSE)
```

Arguments

df	A data.frame to look in for ordinal variables (if you offer a matrix or vector, it will be upgraded to a dataframe)
names	whether to return the names of ordinal variables, or a binary (T,F) list (default = FALSE)
strict	whether to stop when unordered factors are found (default = TRUE)
binary.only	only count binary factors (2-levels) (default = FALSE)
ordinal.only	only count ordinal factors (3 or more levels) (default = FALSE)
continuous.only	use with names = TRUE to get the names of the continuous variables

Value

- vector of variable names or Booleans

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_RAM, umx_is_cov, umx_is_endogenous, umx_is_exogenous

Examples

```
tmp = mtcars
tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$vs = ordered(mtcars$vs) # binary factor
umx_is_ordered(tmp) # numeric indices
umx_is_ordered(tmp, names = TRUE)
umx_is_ordered(tmp, names = TRUE, binary.only = TRUE)
umx_is_ordered(tmp, names = TRUE, ordinal.only = TRUE)
umx_is_ordered(tmp, names = TRUE, continuous.only = TRUE)
umx_is_ordered(tmp, continuous.only = TRUE)
isContinuous = !umx_is_ordered(tmp)
tmp$gear = factor(mtcars$gear) # unordered factor
# nb: Factors are not necessarily ordered! By default unordered factors cause an message...
## Not run:
tmp$cyl = factor(mtcars$cyl)
umx_is_ordered(tmp, names=TRUE)

## End(Not run)
```

---

umx_is_RAM *umx_is_RAM*

Description

Utility function returning a binary answer to the question "Is this a RAM model?"

Usage

```
umx_is_RAM(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx RAM mxModel

Value

- Boolean

References

- http://www.github.com/tbates/umx

See Also

Other Test: umx_check_OS, umx_check_model, umx_check_names, umx_check_parallel, umx_check, umx_has_CIs, umx_has_been_run, umx_has_means, umx_has_square_brackets, umx_is_MxData, umx_is_MxMatrix, umx_is_MxModel, umx_is_cov, umx_is_endogenous, umx_is_exogenous, umx_is_ordered

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umxSummary(m1, show = "std")
if(umx_is_RAM(m1)){
message("nice RAM model!")
}
if(!umx_is_RAM(m1)){
message("model must be a RAM model")
}
```

---

umx_lower2full *umx_lower2full*

Description

Take a lower triangle of data (for instance from a "lower" mxMatrix, or as you might find in a journal article), and turn it into a full matrix.

Usage

```
umx_lower2full(lower.data, diag = NULL, byrow = TRUE, dimnames = NULL)
```

Arguments

lower.data	An mxMatrix
diag	A boolean specifying whether the lower.data includes the diagonal
byrow	Whether the matrix is to be filled by row or by column (default = TRUE)
dimnames	Optional dimnames for the matrix (defaults to NULL)

Value

- mxMatrix

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
tmpn = c("ROccAsp", "REdAsp", "FOccAsp", "FEdAsp", "RParAsp",
         "RIQ", "RSES", "FSES", "FIQ", "FParAsp")
tmp = matrix(nrow = 10, ncol = 10, byrow = TRUE, dimnames = list(tmpn,tmpn), data =
c(1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
0.6247, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
0.3269, 0.3669, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
0.4216, 0.3275, 0.6404, 1.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
0.2137, 0.2742, 0.1124, 0.0839, 1.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
0.4105, 0.4043, 0.2903, 0.2598, 0.1839, 1.0000, 0.0000,  0.0000, 0.0000, 0,
0.3240, 0.4047, 0.3054, 0.2786, 0.0489, 0.2220, 1.0000,  0.0000, 0.0000, 0,
0.2930, 0.2407, 0.4105, 0.3607, 0.0186, 0.1861, 0.2707,  1.0000, 0.0000, 0,
0.2995, 0.2863, 0.5191, 0.5007, 0.0782, 0.3355, 0.2302,  0.2950, 1.0000, 0,
0.0760, 0.0702, 0.2784, 0.1988, 0.1147, 0.1021, 0.0931, -0.0438, 0.2087, 1)
)
umx_lower2full(tmp, diag= TRUE)
tmp = c(
1.0000,
0.6247, 1.0000,
0.3269, 0.3669, 1.0000,
0.4216, 0.3275, 0.6404, 1.0000,
0.2137, 0.2742, 0.1124, 0.0839, 1.0000,
0.4105, 0.4043, 0.2903, 0.2598, 0.1839, 1.0000,
0.3240, 0.4047, 0.3054, 0.2786, 0.0489, 0.2220, 1.0000,
0.2930, 0.2407, 0.4105, 0.3607, 0.0186, 0.1861, 0.2707,  1.0000,
0.2995, 0.2863, 0.5191, 0.5007, 0.0782, 0.3355, 0.2302,  0.2950, 1.0000,
0.0760, 0.0702, 0.2784, 0.1988, 0.1147, 0.1021, 0.0931, -0.0438, 0.2087, 1.000
)
umx_lower2full(tmp, diag = TRUE)
tmp = c(
0.6247,
0.3269, 0.3669,
0.4216, 0.3275, 0.6404,
0.2137, 0.2742, 0.1124, 0.0839,
0.4105, 0.4043, 0.2903, 0.2598, 0.1839,
0.3240, 0.4047, 0.3054, 0.2786, 0.0489, 0.2220,
```

```
0.2930, 0.2407, 0.4105, 0.3607, 0.0186, 0.1861, 0.2707,
0.2995, 0.2863, 0.5191, 0.5007, 0.0782, 0.3355, 0.2302,  0.2950,
0.0760, 0.0702, 0.2784, 0.1988, 0.1147, 0.1021, 0.0931, -0.0438, 0.2087
)
umx_lower2full(tmp, diag = FALSE)
```

---

umx_make *umx_make umx using devtools*

Description

Easily run devtools "install", "release", "win", or "examples".

Usage

```
umx_make(what = c("install", "release", "win", "check", "examples"))
```

Arguments

what	whether to "install", "release" to CRAN, check on "win", "check", or check "examples"))

Value

-

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Miscellaneous Utility Functions: umxVersion, umx_write_to_clipboard

Examples

```
## Not run:
umx_make(what = "release"))

## End(Not run)
```

umx_make_bin_cont_pair_data *umx_make_bin_cont_pair_data*

Description

Takes a dataframe of left-censored variables (vars with a floor effect) and does two things to it: 1. It creates new binary (1/0) copies of each column (with the suffix "bin"). These contain 0 where the variable is below the minimum and NA otherwise. 2. In each existing variable, it sets all instances of min for that var to NA

Usage

```
umx_make_bin_cont_pair_data(data, vars = NULL, suffixes = NULL)
```

Arguments

data	A data.frame to convert
vars	The variables to process
suffixes	Suffixes if the data are family (wide, more than one persona on a row)

Value

- copy of the dataframe with new binary variables and censoring

References

- https://github.com/tbates/umx, https://tbates.github.io, http://openmx.ssri.psu.edu

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
df = umx_make_bin_cont_pair_data(mtcars, vars = c("mpg"))
str(df)
df[order(df$mpg), c(1,12)]
# Introduce a floor effect
tmp = mtcars; tmp$mpg[tmp$mpg<=15]=15
tmp$mpg_T1 = tmp$mpg_T2 = tmp$mpg
df = umx_make_bin_cont_pair_data(tmp, vars = c("mpg"), suffixes = c("_T1", "_T2"))
df[order(df$mpg), 12:15]
```

umx_make_fake_data *umx_make_fake_data*

Description

This function takes as argument an existing dataset, which must be either a matrix or a data frame. Each column of the dataset must consist either of numeric variables or ordered factors. When one or more ordered factors are included, then a heterogeneous correlation matrix is computed using John Fox’s polycor package. Pairwise complete observations are used for all covariances, and the exact pattern of missing data present in the input is placed in the output, provided a new sample size is not requested. Warnings from the polycor::hetcor function are suppressed.

Usage

```
umx_make_fake_data(dataset, digits = 2, n = NA, use.names = TRUE,
  use.levels = TRUE, use.miss = TRUE, mvt.method = "eigen",
  het.ML = FALSE, het.suppress = TRUE)
```

Arguments

dataset	The original dataset of which to make a simulacrum
digits	= Round the data to the requested digits (default = 2)
n	Number of rows to generate (NA = all rows in dataset)
use.names	Whether to name the variables (default = TRUE)
use.levels	= Whether to use existing levels (default = TRUE)
use.miss	Whether to have data missting as in original (defaults to TRUE)
mvt.method	= Passed to hetcor (default = "eigen")
het.ML	= Passed to hetcor (default = FALSE)
het.suppress	Passed to hetcor (default = TRUE)

Value

- new dataframe

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
fakeCars = umx_make_fake_data(mtcars)
```

umx_make_MR_data *Simulate Mendelian Randomization data*

Description

umx_make_MR_data returns a dataset containing 4 variables: A variable of interest (Y), a putative cause (X), a qtl (quantitative trait locus) influencing X, and a confounding variable (U) affecting both X and Y.

Usage

```
umx_make_MR_data(nSubjects = 1000, Vqtl = 0.02, bXY = 0.1, bUX = 0.5,
  bUY = 0.5, pQTL = 0.5, seed = 123)
```

Arguments

nSubjects	Number of subjects in sample
Vqtl	Variance of QTL affecting causal variable X (Default 0.02)
bXY	Causal effect of X on Y (Default 0.1)
bUX	Confounding effect of confounder 'U' on X (Default 0.5)
bUY	Confounding effect of confounder 'U' on Y (Default 0.5)
pQTL	Decreaser allele frequency (Default 0.5)
seed	value for the random number generator (Default 123)

Details

The code to make these Data. Modified from Dave Evans 2016 Boulder workshop talk.

Value

- data.frame

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
df = umx_make_MR_data(10000)
str(df)
## Not run:
m1 = umxTwoStage(Y ~ X, ~qtl, data = df)
plot(m1)

## End(Not run)
```

umx_make_sql_from_excel *umx_make_sql_from_excel*

Description

Unlikely to be of use to anyone but the package author :-) Read an xlsx file and convert into SQL insert statements (placed on the clipboard) On OS X, the function can access the current frontmost Finder window.

Usage

```
umx_make_sql_from_excel(theFile = "Finder")
```

Arguments

theFile	The xlsx file to read. If set to "Finder" (and you are on OS X) it will use the current frontmost Finder window. If it is blank, a choose file dialog will be thrown.

Details

The file name should be the name of the test. Columns should be headed: itemText direction scale type [optional response options]

The SQL fields generated are: itemID, test, native_item_number, item_text, direction, scale, format, author

Value

-

References

- http://www.github.com/tbates/umx

See Also

Other File Functions: dl_from_dropbox, umx_move_file, umx_open, umx_rename_file, umx

Examples

```
## Not run:
# An example xcel spreadsheet
fp = system.file("inst/extdata", "GQ6.sql.xlsx", package = "umx")
fp = system.file("extdata", "GQ6.sql.xlsx", package = "umx")
umx_open(fp)
umx_make_sql_from_excel() # Using file selected in front-most Finder window
umx_make_sql_from_excel("~/Desktop/test.xlsx") # provide a path
```

```
## End(Not run)
```

---

umx_make_TwinData *Simulate twin data with control over A, C, E, and moderation*

Description

Makes MZ and DZ twin data, optionally with moderated A. y default, the three variance components must sum to 1.

See examples for how to use this: it is pretty flexible.

Note, if you want a power calculator, see here. You supply the number of pairs of each zygosity that wish to simulate (nMZpairs, nDZpairs), along with the values of AA, CC,and EE.

Shortcuts

You can omit nDZpairs. You can also give any 2 of A, C, or E and the function will add the value which makes the ACE total = 1.

Moderation

AA can take a list c(avg = .5, min = 0, max = 1). If specified will act like a moderated heritability, with average value avg, and swinging down to min and up to max across 3 SDs of the moderator.

Usage

```
umx_make_TwinData(nMZpairs, nDZpairs = nMZpairs, AA = NULL, CC = NULL,
  EE = NULL, nThresh = NULL, sum2one = TRUE, varNames = "var",
  seed = NULL, empirical = FALSE)
```

Arguments

nMZpairs	Number of MZ pairs to simulate
nDZpairs	Number of DZ pairs to simulate (if omitted defaults to nMZpairs)
AA	value for A variance. Optionally a vecotr: c(avg=.5, min=0, max=1)
CC	value for C variance.
EE	value for E variance.
nThresh	If supplied, use as thresholds and return mxFactor output? (default is not too)
sum2one	Whether to enforce AA + CC + EE summing the one (default = TRUE)
varNames	name for var (defaults to 'var')
seed	Allows user to set.seed() if wanting reproducible dataset
empirical	Passed to mvrnorm

Value

- list of mzData and dzData dataframes containing T1 and T2 plus, if needed M1 and M2 (moderator values)

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
# ====================================================================
# = Basic Example, with all elements of std univariate data specified =
# ====================================================================
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 100, AA = .36, CC = .04, EE = .60)
# Show list of 2 data sets
str(tmp)
# = How to consume the built datasets =
mzData = tmp[[1]];
dzData = tmp[[2]];
cov(mzData); cov(dzData)
str(mzData); str(dzData);

# Prefer to work in path coefficient values? (little a?)
tmp = umx_make_TwinData(200, AA = .6^2, CC = .2^2)
# Check the correlations
umxAPA(tmp[[1]]); umxAPA(tmp[[2]])

# =============
# = Shortcuts =
# =============

# Omit nDZpairs (equal numbers of both by default)
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 100, AA = .36, CC = .04, EE = .60)
tmp = umx_make_TwinData(100, AA = 0.5, CC = 0.3) # omit any one of A, C, or E (sums to 1)
cov(tmp[[1]])
# Not limited to unit variance
tmp = umx_make_TwinData(100, AA = 3, CC = 2, EE = 3, sum2one = FALSE)
cov(tmp[[1]])

# =====================
# = Moderator Example =
# =====================

x = umx_make_TwinData(100, AA = c(avg = .7, min = 0, max = 1), CC = .55, EE = .63)
str(x)

# =====================
# = Threshold Example =
# =====================
tmp = umx_make_TwinData(100, AA = .6, CC = .2, nThresh = 3)
```

```
str(tmp)
umxAPA(tmp[[1]]); umxAPA(tmp[[2]])
```

---

umx_means *umx_means*

Description

Helper to get means from a df that might contain ordered or string data. Factor means are set to "ordVar"

Usage

```
umx_means(df, ordVar = 0, na.rm = TRUE)
```

Arguments

df	a dataframe of raw data from which to get variances.
ordVar	value to return for the means of factor data = 0
na.rm	passed to mean - defaults to "na.rm"

Value

- frame of means

See Also

Other Stats Functions: reliability, umxCov2cor, umx_cor, umx

Examples

```
tmp = mtcars[,1:4]
tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$hp  = ordered(mtcars$hp)  # binary factor
umx_means(tmp, ordVar = 0, na.rm = TRUE)
```

umx_merge_CIs *umx_merge_CIs*

Description

if you compute some CIs in one model and some in another (copy of the same model, perhaps to get some parallelism), this is a simple helper to cludge them together.

Usage

```
umx_merge_CIs(m1, m2)
```

Arguments

m1	first copy of the model
m2	second copy of the model

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
## Not run:
umx_merge_CIs(m1, m2)
# TODO remove duplicates...
# TODO (check they are the same as well!)
# TODO Support arbitrarily long list of input models with ...
# TODO check the models are the same, with same fit
# TODO check the models have CIs

## End(Not run)
```

umx_move_file *umx_move_file*

Description

move files. On OS X, the function can access the current frontmost Finder window. The file moves are fast and, because you can use regular expressions, powerful

Usage

```
umx_move_file(baseFolder = NA, findStr = NULL, fileNameList = NA,
  destFolder = NA, test = TRUE, overwrite = FALSE)
```

Arguments

baseFolder	The folder to search in. If set to "Finder" (and you are on OS X) it will use the current frontmost Finder window. If it is blank, a choose folder dialog will be thrown.
findStr	= regex string select files to move (WARNING: NOT IMPLEMENTED YET)
fileNameList	List of files to move
destFolder	Folder to move files into
test	Boolean determining whether to change the names, or just report on what would have happened
overwrite	Boolean determining whether to overwrite files or not (default = FALSE (safe))

Value

-

See Also

Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_open, umx_rename_file, umx

Examples

```
## Not run:
base = "/Users/tim/Music/iTunes/iTunes Music/"
dest = "/Users/tim/Music/iTunes/iTunes Music/Music/"
umx_move_file(baseFolder = base, fileNameList = toMove, destFolder = dest, test= FALSE)

## End(Not run)
```

umx_msg *umx_msg*

Description

Helper function to make dumping "ObjectName has the value: <objectvalue>" easy.

Usage

```
umx_msg(x)
```

Arguments

x	the thing you want to print

Value

- NULL

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Utility Functions: qm, umx_find_object, umx_grep, umx_names, umx_paste_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
a = "brian"
umx_msg(a)
b = c("brian", "sally", "jane")
umx_msg(b)
```

umx_names *umx_names*

Description

Convenient equivalent of grep("fa[rl].*", names(df), value = TRUE, ignore.case = TRUE) Can handle dataframe (uses names), model (uses parameter names), or a vector of strings.

Usage

```
umx_names(df, pattern = ".*", ignore.case = TRUE, perl = FALSE,
  value = TRUE, fixed = FALSE, useBytes = FALSE, invert = FALSE)
```

Arguments

df	dataframe to get names from
pattern	= "find.*"
ignore.case	default = TRUE (opposite default to grep)
perl	= FALSE
value	= default = TRUE (opposite default to grep)
fixed	= FALSE
useBytes	= FALSE
invert	= FALSE

Value

- vector of matches

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_paste_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
umx_names(mtcars, "mpg") #"mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear" "carb"
umx_names(mtcars, "^d") # "disp", drat
umx_names(mtcars, "r[ab]") # "drat", "carb"
```

---

umx_object_as_str *umx_object_as_str*

Description

Utility to return an object's name as a string

Usage

```
umx_object_as_str(x)
```

Arguments

x	an object

Value

- name as string

References

- http://www.github.com/tbates/umx

See Also

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_string_to_algebra, umx

Examples

```
umx_object_as_str(mtcars) # "mtcars"
```

---

umx_open *umx_open*

Description

Open a file or folder. Works on OS X, mostly on windows, and hopefully on unix.

Usage

```
umx_open(filepath = getwd())
```

Arguments

filepath	The file to open

Details

NOTE: Your filepath is shQuoted by this function.

Value

-

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_move_file, umx_rename_file, umx

Examples

```
## Not run:
umx_open(getwd())
umx_open("~/bin/umx/R/misc_and_utility copy.r")

## End(Not run)
```

---

umx_open_CRAN_page *Open the CRAN page for a package*

Description

On MacOS, this function opens the CRAN page for a package. Useful for looking up documentation, checking you have an up-to-date version, showing the package to people etc.

Usage

```
umx_open_CRAN_page(package = "umx")
```

Arguments

package	An R package name.

Value

-

See Also

Other Miscellaneous Functions: install.OpenMx, umx_get_options, umx_pad

Examples

```
## Not run:
umx_open_CRAN_page("umx")

## End(Not run)
```

umx_pad *Pad an Object with NAs*

Description

This function pads an R object (list, data.frame, matrix, atomic vector) with NAs. For matrices, lists and data.frames, this occurs by extending each (column) vector in the object.

Usage

```
umx_pad(x, n)
```

Arguments

x	An R object (list, data.frame, matrix, atomic vector).
n	The final length of each object.

Value

- padded object

References

- https://github.com/kevinushey/Kmisc/tree/master/man

See Also

Other Miscellaneous Functions: install.OpenMx, umx_get_options, umx_open_CRAN_page

Examples

```
umx_pad(1:3, 4)
umx_pad(1:3, 3)
```

umx_paste_names *Concatenate base variable names with suffixes to create wide-format variable names (i.e twin-format)*

Description

It's easier to work with base names, rather than the twice-as-long hard-to-typo list of column names. umx_paste_names adds suffixes to names so you can work with that nice short list. So, you provide bmi, and you get back fully specified family-wise names: c("bmi_T1", "bmi_T2")

Usage

```
umx_paste_names(varNames, sep = "", suffixes = 1:2, covNames = NULL,
  prefix = NULL)
```

Arguments

varNames	a list of *base* names, e.g c("bmi", "IQ")
sep	A string separating the name and the twin suffix, e.g. "_T" (default is "")
suffixes	a list of terminal suffixes differentiating the twins default = c("1", "2"))
covNames	a list of *base* names for covariates (sorted last in list), e.g c("age", "sex")
prefix	a string to pre=pend to each label, e.g c("mean_age", "mean_sex")

Details

Method 1: **Use complete suffixes**

You can provide complete suffixes like "_T1" and "_T2". This has the benefit of being explicit and very general:

```
umx_paste_names(c("var1", "var2"), suffixes = c("_T1", "_T2"))
```

Method 2: Use sep and a suffix vector

Alternatively, you can use sep to add a constant like "_T" after each basename, along with a vector of suffixes. This has the benefit of showing what is varying: This is then suffixed with e.g. "1", "2".

```
umx_paste_names(c("var1", "var2"), sep = "_T", suffixes = 1:2)
```

Working with covariates

If you are using umxACEcov, you **need** to keep all the covariates at the end of the list. Here's how:

```
umx_paste_names(c("var1", "var2"), cov = c("cov1"), sep = "_T", suffixes = 1:2)
```

note: in conventional twin models, the expCov matrix is T1 vars, followed by T2 vars. For covariates, you want T1vars, T2 vars, T1 covs, T2 covs. This is what covNames accomplishes.

Value

- vector of suffixed var names, i.e., c("v1_T1", "v2_T1", "v1_T2", "v2_T2", "cov_T1", "cov_T2")

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_names, umx_pb_note, umx_print, umx_rename, umx

Examples

```
# two styles doing the same thing: first is more general
umx_paste_names("bmi", suffixes = c("_T1", "_T2"))
umx_paste_names("bmi", sep = "_T", suffixes = 1:2)
varNames = umx_paste_names(c("N", "E", "O", "A", "C"), "_T", 1:2)
umx_paste_names(c("IQ", "C"), cov = c("age"), sep = "_T", suffixes = 1:2)
umx_paste_names(c("IQ", "C"), cov = c("age"), sep = "_T", prefix= "mean_")
```

---

umx_pb_note *umx_pb_note*

Description

Use the pushbullet service to push a note. You can also initialise this service by providing your autho_key one time

Usage

```
umx_pb_note(title = "test", body = "body", auth_key = c(NA, "GET"))
```

Arguments

title	of the note
body	of the note
auth_key	optional authkey (default = NA, set to value of your key to store key.

Details

If you supply auth_key, It will be writen to "~/.pushbulletkey" umx_pb_note(auth_key="mykeystring") once it exists there, you dont need to store it in code, so code is sharable.

You can get your authaurization key at https://www.pushbullet.com in the "account" section.

Note: You can show the existing stored key using "GET"

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

- umx_msg

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_names, umx_paste_names, umx_print, umx_rename, umx

Examples

```
## Not run:
umx_pb_note("done!", umx_time(m1))

## End(Not run)
```

---

umx_print *umx_print*

Description

A helper to aid the interpretability of printed tables from OpenMx (and elsewhere). Its most useful characteristics are allowing you to change how NA and zero appear. and supressing values below a certain cut-off. By default, Zeros have the decimals suppressed, and NAs are suppressed altogether.

Usage

```
umx_print(x, digits = getOption("digits"), quote = FALSE, na.print = "",
  zero.print = "0", justify = "none", file = c(NA, "tmp.html"),
  suppress = NULL, ...)
```

Arguments

x	A data.frame to print (matrices will be coerced to data.frame)
digits	The number of decimal places to print (defaults to getOption("digits")
quote	Parameter passed to print (defaults to FALSE)
na.print	String to replace NA with (default to blank "")
zero.print	String to replace 0.000 with (defaults to "0")
justify	Parameter passed to print (defaults to "none")
file	whether to write to a file (defaults to NA (no file). Use "tmp.html" to open as tables in browser.
suppress	minimum numeric value to print (default = NULL, print all values, no matter how small)
...	Optional parameters for print

See Also

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_rename, umx

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_show, umx_time, umx

Examples

```
umx_print(mtcars[1:10,], digits = 2, zero.print = ".", justify = "left")
## Not run:
umx_print(model)
# open in browser
umx_print(mtcars[1:10,], file = "Rout.html")

## End(Not run)
```

---

umx_read_lower *Read lower-triangle of data matrix from console or file*

Description

umx_read_lower will read a lower triangle of data, either from the console, or from file, and return a full matrix, optionally coerced to positive definite. This is useful, especially when copying data from a paper that includes just the lower triangle of a correlation matrix.

Usage

```
umx_read_lower(file = "", diag = TRUE, names = as.character(paste("X",
  1:n, sep = "")), ensurePD = FALSE)
```

Arguments

file	Path to a file to read (Default "" will read from user input)
diag	Whether the data unclude the diagonal or not: Defaults to TRUE
names	The default names for the variables. Defaults to as.character(paste("X", 1:n, sep=""))
ensurePD	Whether to coerce the resultant matrix to positive definite (Defaults to FALSE)

Value

- matrix

References

- https://github.com/tbates/umx, https://tbates.github.io

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
require(umx) # for umxRAM
## Not run:
df = umx_read_lower(file = "", diag = F, ensurePD=TRUE)
0.38
0.86 0.30
0.42 0.12 0.27
0.66 0.21 0.38 0.18
0.80 0.13 0.50 0.25 0.43
0.19 0.11 0.19 0.12 -0.06 0.22
0.27 0.09 0.33 0.05 -0.04 0.28 .73
0.52 0.17 0.38 0.37 0.39 0.44 0.18 0.13

IQtests = c("brainstorm", "matrix", "moral", "shopping", "typing")
n       = c("C", IQtests, "avgIQ", "maxIQ", "video")

dimnames(df) = list(n,n)

m1 = umxRAM("wooley", data = mxData(df, type="cov", numObs = 90),
umxPath("g", to = IQtests),
umxPath(var = "g", fixedAt=1),
umxPath(var = IQtests)
)
summary(m1)

## End(Not run)
```

---

umx_rename *umx_rename*

Description

Returns a dataframe with variables renamed as desired. Unlike some functions, it checks that the variables exist, and that the new names are not already used.

Usage

```
umx_rename(x, replace = NULL, old = NULL, grep = NULL, test = FALSE)
```

Arguments

x	the dataframe in which to rename variables
replace	If used alone, a named collection of c(oldName = "newName") pairs OR, if "old" is a list of existing names, the list of new names) OR, if "grep" is a regular expression, the replace string)
old	Optional list of old names that will be found and replaced by the contents of replace. Defaults to NULL

grep	Optional grep string. Matches will be replaced using replace as the replace string. Defaults to NULL
test	whether to report a "dry run" - and not actually change anything (defaults to false)

Details

As a courtesy function, it handles grep replace in strings of characters.

note: to use replace list, you must say c(old = "new"), not c(old -> "new")

Value

- dataframe with columns renamed.

See Also

Other Utility Functions: qm, umx_find_object, umx_grep, umx_msg, umx_names, umx_paste_names, umx_pb_note, umx_print, umx

Examples

```
# Re-name "cyl" to "cylinder"
x = mtcars
x = umx_rename(x, replace = c(cyl = "cylinder"))
# alternate style
x = umx_rename(x, old = c("disp"), replace = c("displacement"))
umx_check_names("displacement", data = x, die = TRUE)
# This will warn that "disp" doesn't exist (anymore)
x = umx_rename(x, old = c("disp"), replace = c("displacement"))
x = umx_rename(x, grep = "lacement", replace = "") # using grep to revert to disp
umx_names(x, "^d") # all names begining with a d
```

---

umx_rename_file *umx_rename_file*

Description

rename files. On OS X, the function can access the current frontmost Finder window. The file renaming is fast and, because you can use regular expressions, powerful

Usage

```
umx_rename_file(findStr = NA, replaceStr = NA, baseFolder = "Finder",
  listPattern = NA, test = TRUE, overwrite = FALSE)
```

Arguments

findStr	The (regex) string to find, i.e., "c[ao]t"
replaceStr	The (regex) replacement string "\1 are not dogs"
baseFolder	The folder to search in. If set to "Finder" (and you are on OS X) it will use the current frontmost Finder window. If it is blank, a choose folder dialog will be thrown.
listPattern	A pre-filter for files
test	Boolean determining whether to change files on disk, or just report on what would have happened (Defaults to test = TRUE)
overwrite	Boolean determining if an existing file will be overwritten (Defaults to the safe FALSE)

Value

-

References

- http://www.github.com/tbates/umx

See Also

Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_move_file, umx_open, umx

Examples

```
## Not run:
umx_rename_file(baseFolder = "~/Downloads/", findStr = "", replaceStr = "", test = TRUE)
umx_rename_file("[Ss]eason +([0-9]+)", replaceStr="S\1", baseFolder = "Finder", test = TRUE)

## End(Not run)
```

---

umx_reorder *umx_reorder*

Description

Reorder the variables in a correlation matrix. Can also remove one or more variables from a matrix using this function

Usage

```
umx_reorder(old, newOrder)
```

Arguments

old	a square matrix of correlation or covariances to reorder
newOrder	Variables you want in the order you wish to have

Value

- the re-ordered/resized matrix

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
oldMatrix = cov(mtcars)
umx_reorder(oldMatrix, newOrder = c("mpg", "cyl", "disp")) # first 3
umx_reorder(oldMatrix, newOrder = c("hp", "disp", "cyl")) # subset and reordered
umx_reorder(oldMatrix, "hp") # edge-case of just 1-var
```

---

umx_residualize *umx_residualize*

Description

Residualise one or more variables residualised against covariates, and return a complete dataframe with residualized variable in place. Optionally, this also works on wide (ie., twin) data. Just supply suffixes to identify the paired-wide columns (see examples)

Usage

```
umx_residualize(var, covs = NULL, suffixes = NULL, data)
```

Arguments

var	The base name of the variable you want to residualize. Alternatively, a regression formula containing var on the lhs, and covs on the rhs
covs	Covariates to residualize on.
suffixes	Suffixes that identify the variable for each twin, i.e. c("_T1", "_T2") Up to you to check all variables are present!
data	The dataframe containing all the variables

Details

In R, residuals for a variable can be found with the following statement:

```
tmp <- residuals(lm(var ~ cov1 + cov2, data = data, na.action = na.exclude))
```

This tmp variable could then be written over the old data:

umx_residualize obviates the user having to build the lm, set na.action, or replace the data. In addition, it has the powerful feature of operating on a list of variables, and of operating on wide data, expanding the var name using a set of variable-name suffixes.

Value

- dataframe with var residualized in place (i.e under its original column name)

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_round, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
# Residualise mpg on cylinders and displacement
r1 = umx_residualize("mpg", c("cyl", "disp"), data = mtcars)
r2 = residuals(lm(mpg ~ cyl + disp, data = mtcars, na.action = na.exclude))
all(r1$mpg == r2)
# =====================
# = formula interface =
# =====================
r1 = umx_residualize(mpg ~ cyl + I(cyl^2) + disp, data = mtcars)
r2 = residuals(lm(mpg ~ cyl + I(cyl^2) + disp, data = mtcars, na.action = na.exclude))
all(r1$mpg == r2)

# ============================================================================
# = Demonstrate ability to residualize WIDE data (i.e. 1 family per row) =
# ============================================================================
tmp = mtcars
tmp$mpg_T1  = tmp$mpg_T2  = tmp$mpg
tmp$cyl_T1  = tmp$cyl_T2  = tmp$cyl
tmp$disp_T1 = tmp$disp_T2 = tmp$disp
umx_residualize("mpg", c("cyl", "disp"), c("_T1", "_T2"), data = tmp)[1:5,12:17]

# ===================================
# = Residualise several DVs at once =
# ===================================
df1 = umx_residualize(c("mpg", "hp"), cov = c("cyl", "disp"), data = tmp)
df2 = residuals(lm(hp ~ cyl + disp, data = tmp, na.action = na.exclude))
```

```
all(df1$hp == df2)
```

---

umx_rot *umx_rot*

Description

rotate a vector (default, rotate by 1)

Usage

```
umx_rot(vec)
```

Arguments

vec	vector to rotate

Value

- mxModel

References

- http://tbates.github.io

See Also

Other String Functions: umx_explode_twin_names, umx_explode, umx_trim

Examples

```
umx_rot(1:10)
umx_rot(c(3,4,5,6,7))
# [1] 4 5 6 7 3
```

umx_round *umx_round*

Description

A version of round() which works on dataframes that contain non-numeric data (or data that cannot be coerced to numeric) Helpful for dealing with table output that mixes numeric and string types.

Usage

```
umx_round(df, digits = getOption("digits"), coerce = FALSE)
```

Arguments

df	a dataframe to round in
digits	how many digits to round to (defaults to getOption("digits"))
coerce	whether to make the column numeric if it is not (default = FALSE)

Value

- mxModel

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_scale_wide_twin_data, umx_scale, umx_swap_a_block, umx

Examples

```
head(umx_round(mtcars, coerce = FALSE))
head(umx_round(mtcars, coerce = TRUE))
```

umx_r_test *umx_r_test*

Description

umx_r_test is a wrapper around the cocor test of difference between correlations.

Usage

```
umx_r_test(data = NULL, vars = vars, alternative = c("two.sided",
  "greater", "less"))
```

Arguments

data	the dataset
vars	the 4 vars needed: "j & k" and "h & m"
alternative	two (default) or one-sided (greater less) test

Details

Currently it handles the test of whether r.jk and r.hm differ in magnitude. i.e, two nonoverlapping (no variable in common) correlations in the same dataset. In the future it will be expanded to handle overlapping correlations, and to take corelation matrices as input.

Value

-

Examples

```
vars = c("mpg", "cyl", "disp", "hp")
umx_r_test(mtcars, vars)
```

umx_scale *umx_scale*

Description

Scale data columns, skipping ordinal

Usage

```
umx_scale(df, varsToScale = NULL, coerce = FALSE)
```

Arguments

df	a dataframe to scale
varsToScale	(leave blank for all)
coerce	Whether to coerce non-numerics to numeric (Defaults to FALSE)

Value

- new dataframe with scaled variables

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_swap_a_block, umx

Examples

```
data(twinData)
df = umx_scale(twinData, varsToScale = NULL)
plot(wt1 ~ wt2, data = df)
```

---

umx_scale_wide_twin_data

umx_scale_wide_twin_data

Description

Scale wide data across all cases: currently twins

Usage

```
umx_scale_wide_twin_data(varsToScale, suffix, data)
```

Arguments

varsToScale	The base names of the variables ("weight" etc)
suffix	The suffix that distinguishes each case, e.g. "_T")
data	a wide dataframe

Value

- new dataframe with variables scaled in place

References

- http://www.github.com/tbates/umx

See Also

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale, umx_swap_a_block, umx

Examples

```
data(twinData)
df = umx_scale_wide_twin_data(twinData, varsToScale = c("ht", "wt"), suffix = "" )
plot(wt1 ~ wt2, data = df)
```

---

umx_set_auto_plot *umx_set_auto_plot*

Description

Set autoPlot default for models like umxACE umxGxE etc

Usage

```
umx_set_auto_plot(autoPlot = NULL, silent = FALSE)
```

Arguments

autoPlot	If NA or "name", sets the umx_auto_plot option. Else returns the current value of umx_auto_plot
silent	If TRUE, no message will be printed.

Value

- Current umx_auto_plot setting

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
library(umx)
umx_set_auto_plot() # print current state
old = umx_set_auto_plot(silent = TRUE) # store existing value
umx_set_auto_plot("name")  # set to "name"
umx_set_auto_plot(old)    # reinstate
```

---

umx_set_auto_run *umx_set_auto_run*

Description

Set autorun default for models like umxACE umxGxE etc

Usage

```
umx_set_auto_run(autoRun = NA, silent = FALSE)
```

Arguments

autoRun	If TRUE or FALSE, sets the umx_auto_run option. Else returns the current value of umx_auto_run
silent	If TRUE, no message will be printed.

Value

- Current umx_auto_run setting

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
library(umx)
umx_set_auto_run() # print existing value
old = umx_set_auto_run(silent = TRUE) # store existing value
umx_set_auto_run(FALSE)  # set to FALSE
umx_set_auto_run(old)    # reinstate
```

umx_set_checkpoint *umx_set_checkpoint*

Description

Set the checkpoint status for a model or global options

Usage

```
umx_set_checkpoint(interval = 1, units = c("evaluations", "iterations",
  "minutes"), prefix = "", directory = getwd(), model = NULL)
```

Arguments

interval	How many units between checkpoints: Default = 1. A value of zero sets always to 'No' (i.e., do not checkpoint all models during optimization)
units	units to count in: Default unit is 'evaluations' ('minutes' is also legal)
prefix	string prefix to add to all checkpoint filenames (default = "")
directory	a directory, i.e "~/Desktop" (defaults to getwd())
model	(optional) model to set options in (default = NULL)

Value

- mxModel if provided

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
umx_set_checkpoint(interval = 1, "evaluations", dir = "~/Desktop/")
# turn off checkpointing with interval = 0
umx_set_checkpoint(interval = 0)
umx_set_checkpoint(2, "evaluations", prefix="SNP_1")
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
umxPath(latents, to = manifests),
```

```
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)
m1 = umx_set_checkpoint(model = m1)
m1 = mxRun(m1)
umx_checkpoint(0)
```

---

umx_set_condensed_slots *umx_set_condensed_slots*

Description

Sets whether newly-created mxMatrices are to be condensed (set to NULL if not being used) or not.

Usage

```
umx_set_condensed_slots(state = NA, silent = FALSE)
```

Arguments

state	what state (TRUE or FALSE) to set condensed slots (default NA returns current value).
silent	If TRUE, no message will be printed.

Value

- current value of condensed slots

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_cores, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
library(umx)
umx_set_condensed_slots() # print
old = umx_set_condensed_slots(silent = TRUE) # store the existing state
umx_set_condensed_slots(TRUE) # update globally
umx_set_condensed_slots(old) # set back
```

umx_set_cores *umx_set_cores*

Description

set the number of cores (threads) used by OpenMx

Usage

```
umx_set_cores(cores = NA, model = NULL, silent = FALSE)
```

Arguments

cores	number of cores to use. NA (the default) returns current value. "-1" will set to detectCores().
model	an (optional) model to set. If left NULL, the global option is updated.
silent	If TRUE, no message will be printed.

Value

- number of cores

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_optimizer, umx_set_plot_format, umx_set_table_format

Examples

```
library(umx)
manifests = c("mpg", "disp", "gear")
m1 <- mxModel("ind", type = "RAM",
manifestVars = manifests,
mxPath(from = manifests, arrows = 2),
mxPath(from = "one", to = manifests),
mxData(mtcars[, manifests], type = "raw")
)
umx_set_cores() # print current value
oldCores <- umx_set_cores(silent = TRUE)  # store existing value
umx_set_cores(parallel::detectCores()) # set to max
umx_set_cores(-1); umx_set_cores() # set to max
m1 = umx_set_cores(1, m1)  # set m1 useage to 1 core
umx_set_cores(model = m1)  # show new value for m1
umx_set_cores(oldCores)    # reinstate old global value
```

umx_set_optimizer *umx_set_optimizer*

Description

Set the optimizer in OpenMx

Usage

```
umx_set_optimizer(opt = NA, model = NULL, silent = FALSE)
```

Arguments

opt	default (NA) returns current value. Current alternatives are "NPSOL" "SLSQP" and "CSOLNP".
model	A model for which to set the optimizer. Default (NULL) sets the optimizer globally.
silent	If TRUE, no message will be printed.

Value

-

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_plot_format, umx_set_table_format

Examples

```
library(umx)
umx_set_optimizer() # print the existing state
old = umx_set_optimizer(silent = TRUE) # store the existing state
umx_set_optimizer("SLSQP") # update globally
umx_set_optimizer(old) # set back
```

umx_set_plot_format *umx_set_plot_format*

Description

Set output format of plots (default = "DiagrammeR", alternative is "graphviz")

Usage

```
umx_set_plot_format(umx.plot.format = NULL, silent = FALSE)
```

Arguments

umx.plot.format	format for plots (if empty, returns the current value of umx.plot.format)
silent	If TRUE, no message will be printed.

Value

- Current umx.plot.format setting

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_table_format

Examples

```
library(umx)
umx_set_plot_format() # print current state
old = umx_set_plot_format(silent = TRUE) # store current value
umx_set_plot_format("graphviz")
umx_set_plot_format("DiagrammeR")
umx_set_plot_format(old) # reinstate
```

umx_set_table_format *umx_set_table_format*

Description

Set knitr.table.format default (output style for tables). Legal values are "latex", "html", "markdown", "pandoc", and "rst".

Usage

```
umx_set_table_format(knitr.table.format = NULL, silent = FALSE)
```

Arguments

knitr.table.format	format for tables (if empty, returns the current value of knitr.table.format)
silent	If TRUE, no message will be printed.

Value

- Current knitr.table.format setting

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other Get and set: umx_get_checkpoint, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimizer, umx_set_plot_format

Examples

```
library(umx)
umx_set_table_format() # show current state
old = umx_set_table_format() # store existing value
umx_set_table_format("latex")
umx_set_table_format("html")
umx_set_table_format("markdown")
umx_set_table_format("") # get available options
umx_set_table_format(old)    # reinstate
```

umx_show *umx_show*

Description

Show matrix contents. The user can select values, free, and/or labels, and which matrices to display

Usage

```
umx_show(model, what = c("values", "free", "labels", "nonzero_or_free"),
  show = c("all", "free", "fixed"), matrices = c("S", "A"), digits = 2)
```

Arguments

model	an mxModel to show data from
what	legal options are "values" (default), "free", or "labels")
show	filter on what to show c("all", "free", "fixed")
matrices	to show (default is c("S", "A")). "Thresholds" in beta
digits	precision to report, defaults to rounding to 2 decimal places

Value

- mxModel

References

- http://tbates.github.io

See Also

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_time, umx

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
umx_show(m1)
```

```
umx_show(m1, digits = 3)
umx_show(m1, matrices = "S")
umx_show(m1, what = "free")
umx_show(m1, what = "labels")
umx_show(m1, what = "free", matrices = "A")
```

---

umx_standardize_ACE *umx_standardize_ACE*

Description

Standardize an ACE model

Usage

```
umx_standardize_ACE(fit)
```

Arguments

fit	an umxACE model to standardize

Value

- Standardized ACE umxACE model

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other zAdvanced Helpers: umx_standardize_ACEcov, umx_standardize_CP, umx_standardize_IP, umx

Examples

```
require(umx)
data(twinData)
selDVs = c("bmi1", "bmi2")
mzData <- twinData[twinData$zyg == 1, selDVs][1:80,] # 80 pairs for speed
dzData <- twinData[twinData$zyg == 3, selDVs][1:80,]
m1  = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
std = umx_standardize_ACE(m1)
```

umx_standardize_ACEcov *umx_standardize_ACEcov*

Description

Standardize an ACE model with covariates

Usage

```
umx_standardize_ACEcov(fit)
```

Arguments

fit	an umxACEcov model to standardize

Value

- Standardized umxACEcov model

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other zAdvanced Helpers: umx_standardize_ACE, umx_standardize_CP, umx_standardize_IP, umx

Examples

```
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi")
selCovs = c("age")
selVars = umx_paste_names(c(selDVs, selCovs), sep = "", suffixes= 1:2)
mzData = subset(twinData, zyg == 1, selVars)[1:80, ]
dzData = subset(twinData, zyg == 3, selVars)[1:80, ]
m1 = umxACEcov(selDVs = selDVs, selCovs = selCovs, dzData = dzData, mzData = mzData,
 suffix = "", autoRun = TRUE)
fit = umx_standardize_ACEcov(m1)
```

umx_standardize_CP *umx_standardize_CP*

Description

This function simply inserts the standardized CP components into the ai ci ei and as cs es matrices

Usage

```
umx_standardize_CP(fit)
```

Arguments

fit	an umxCP model to standardize

Value

- standardized umxCP model

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other zAdvanced Helpers: umx_standardize_ACEcov, umx_standardize_ACE, umx_standardize_IP, umx

Examples

```
## Not run:
fit = umx_standardize_CP(fit)

## End(Not run)
```

umx_standardize_IP *umx_standardize_IP*

Description

This function simply inserts the standardized IP components into the ai ci ei and as cs es matrices

Usage

```
umx_standardize_IP(fit)
```

Arguments

fit	an umxIP model to standardize

Value

- standardized IP umxIP model

References

- http://tbates.github.io, https://github.com/tbates/umx

See Also

Other zAdvanced Helpers: umx_standardize_ACEcov, umx_standardize_ACE, umx_standardize_CP, umx

Examples

```
## Not run:
fit = umx_standardize_IP(fit)

## End(Not run)
```

---

umx_standardize_RAM *umx_standardize_RAM*

Description

umx_standardize_RAM takes a RAM-style model, and returns standardized version.

Usage

```
umx_standardize_RAM(model, return = "parameters", Amatrix = NA,
  Smatrix = NA, Mmatrix = NA)
```

Arguments

model	The mxModel you wish to standardise
return	What to return. Valid options: "parameters", "matrices", or "model"
Amatrix	Optionally tell the function what the name of the asymmetric matrix is (defaults to RAM standard A)
Smatrix	Optionally tell the function what the name of the symmetric matrix is (defaults to RAM standard S)
Mmatrix	Optionally tell the function what the name of the means matrix is (defaults to RAM standard M)

Value

- a mxModel or else parameters or matrices if you request those

References

- http://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, confint.MxModel, extractAIC.MxModel, loadings, logLik.MxModel, plot.MxModel, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEcov, umxPlotACE, umxPlotCP, umxPlotGxE, umxPlotIP, umxSummary.MxModel, umxSummaryACE, umx_drop_ok

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umxRun(m1, setLabels = TRUE, setValues = TRUE)
m1 = umx_standardize_RAM(m1, return = "model")
summary(m1)
```

---

umx_string_to_algebra *umx_string_to_algebra*

Description

This is useful because it lets you use paste() and rep() to quickly and easily insert values from R variables into the string, then parse the string as an mxAlgebra argument. The use case this time was to include a matrix exponent (that is A

Usage

```
umx_string_to_algebra(algString, name = NA, dimnames = NA)
```

Arguments

algString	a string to turn into an algebra
name	of the returned algebra
dimnames	of the returned algebra

Value

- mxAlgebra

References

- http://www.github.com/tbates/umx

See Also

Other Misc: umxEval, umx_APA_model_CI, umx_add_variances, umx_apply, umx_default_option, umx_get_bracket_addresses, umx_object_as_str, umx

Examples

```
## Not run:
alg = umx_string_to_algebra(paste(rep("A", nReps), collapse = " %*% "), name = "test_case")

## End(Not run)
```

---

umx_swap_a_block *umx_swap_a_block*

Description

Swap a block of rows of a datset between two lists variables (typically twin 1 and twin2)

Usage

```
umx_swap_a_block(theData, rowSelector, T1Names, T2Names)
```

Arguments

theData	a data frame to swap within
rowSelector	rows to swap amongst columns
T1Names	the first set of columns
T2Names	the second set of columns

Value

- dataframe

See Also

- subset

Other Data Functions: umxCovData, umxFactor, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cont_2_quantiles, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_merge_CIs, umx_read_lower, umx_reorder, umx_residualize, umx_round, umx_scale_wide_twin_data, umx_scale, umx

Examples

```
test = data.frame(
a = paste0("a", 1:10),
b = paste0("b", 1:10),
c = paste0("c", 1:10),
d = paste0("d", 1:10), stringsAsFactors = FALSE)
umx_swap_a_block(test, rowSelector = c(1,2,3,6), T1Names = "b", T2Names = "c")
umx_swap_a_block(test, rowSelector = c(1,2,3,6), T1Names = c("a","c"), T2Names = c("b","d"))
```

---

umx_time *umx_time*

Description

A function to compactly report how long a model took to execute. Comes with some preset styles User can set the format with C-style string formatting.

Usage

```
umx_time(x = NA, formatStr = c("simple", "std", "custom %H %M %OS3"),
  tz = "GMT", autoRun = TRUE)
```

Arguments

x	A mxModel or list of models for which to display elapsed time, or 'start' or 'stop'
formatStr	A format string, defining how to show the time (defaults to human readable)
tz	time zone in which the model was executed (defaults to "GMT")
autoRun	If TRUE (default), run the model if it appears not to have been.

Details

The default time format is "simple", which gives only the biggest unit used. i.e., "x seconds" for times under 1 minute. "std" shows time in the format adopted in OpenMx 2.0 e.g. "Wall clock time (HH:MM:SS.hh): 00:00:01.16"

If a list of models is provided, time deltas will also be reported.

If instead of a model the key word "start" is given in x, a start time will be recorded. "stop" gives the time since "start" was called (and clears the timer)

If a model has not been run, umx_time will run it for you.

Value

- invisible time string

References

- http://www.github.com/tbates/umx

See Also

Other Reporting Functions: loadings.MxModel, umxAPA, umxGetParameters, umxSummary, umx_APA_pval, umx_aggregate, umx_print, umx_show, umx

Examples

```
require(umx)
umx_time('start')
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
m1 <- umxRAM("One Factor", data = myData,
umxPath(from = latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
umx_time(m1)
m2 = umxRun(m1)
umx_time(c(m1, m2))
umx_time('stop')
# elapsed time: .3 seconds
```

---

umx_trim *umx_trim*

Description

returns string w/o leading or trailing whitespace

Usage

```
umx_trim(string, removeThis = NULL)
```

Arguments

string	to trim
removeThis	if not NULl then this string is removed whereever found in 'string'

Value

- string

References

- http://tbates.github.io, https://github.com/tbates/umx, http://openmx.ssri.psu.edu

See Also

Other String Functions: umx_explode_twin_names, umx_explode, umx_rot

Examples

```
umx_trim(" dog") # "dog"
umx_trim("dog ") # "dog"
umx_trim("\t dog \n") # "dog"
umx_trim("xlsx dog.xlsx", "\\.xlsx$") # "dog"
```

---

umx_var *Get variances from a df that might contain some non-numeric columns*

Description

Pass in any dataframe and get variances despite some non-numeric columns. Cells involving these non-numeric columns are set to ordVar (default = 1).

Usage

```
umx_var(df, ordVar = 1, format = c("full", "diag", "lower"),
  use = c("complete.obs", "pairwise.complete.obs", "everything", "all.obs",
  "na.or.complete"))
```

Arguments

df	a dataframe of raw data from which to get variances.
ordVar	The value to return at any ordinal columns (defaults to 1).
format	to return: options are c("full", "diag", "lower"). Defaults to full, but this is not implemented yet.
use	passed to cov - defaults to "complete.obs" (see param default for other options).

Value

- mxModel

References

- http://tbates.github.io

See Also

Other Building Functions: umx_check_names

Examples

```
tmp = mtcars[,1:4]
tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$hp  = ordered(mtcars$hp)  # binary factor
umx_var(tmp, format= "diag", ordVar = 1, use = "pair")
tmp2 = tmp[, c(1,3)]
umx_var(tmp2, format= "diag")
# todo: umx_var(tmp2, format = "full")
```

---

umx_write_to_clipboard *umx_write_to_clipboard*

Description

umx_write_to_clipboard writes data to the clipboard

Usage

```
umx_write_to_clipboard(x)
```

Arguments

x	something to put on the clipboard

Details

TODO: Make this more robust to OS. Let me know if it fails for you.

Value

-

See Also

Other Miscellaneous Utility Functions: umxVersion, umx_make

Examples

```
## Not run:
umx_write_to_clipboard("hello")

## End(Not run)
```

`us_skinfold_data` *Anthropometric data on twins*

Description

A dataset containing height, weight, bmi, and skin-fold fat measures in several hundred US twin families participating in the MCV Cardiovascular Twin Study (PI Schieken)

Usage

```
data(us_skinfold_data)
```

Format

A data frame with 53940 rows and 10 variables

Details

- fan FamilyID (t1=male,t2=female)
- zyg Zygosity 1:mzm, 2:mzf, 3:dzm, 4:dzf, 5:dzo
- ht_T1 Height of twin 1 (cm)
- wt_T1 Weight of twin 1 (kg)
- bmi_T1 BMI of twin 1
- bml_T1 BMI of twin 1
- bic_T1 Biceps Skinfold of twin 1
- caf_T1 Calf Skinfold of twin 1
- ssc_T1 Subscapular SSkinfold of twin 1
- sil_T1 Suprailiacal Skinfold of twin 1
- tri_T1 Triceps Skinfold of twin 1
- ht_T2 Height of twin 2
- wt_T2 Weight of twin 2
- bmi_T2 BMI of twin 2
- bml_T2 BMI of twin 2
- bic_T2 Biceps Skinfold of twin 2
- caf_T2 Calf Skinfold of twin 2
- ssc_T2 Subscapular Skinfold of twin 2
- sil_T2 Suprailiacal Skinfold of twin 2
- tri_T2 Triceps Skinfold of twin 2

References

Moskowitz, W. B., Schwartz, P. F., & Schieken, R. M. (1999). Childhood passive smoking, race, and coronary artery disease risk: the MCV Twin Study. Medical College of Virginia. Archives of Pediatrics and Adolescent Medicine, **153**, 446-453. https://www.ncbi.nlm.nih.gov/pubmed/10323623

Examples

```
data(us_skinfold_data)
str(us_skinfold_data)
par(mfrow = c(1, 2))  # 1 rows and 3 columns
plot(ht_T1 ~ht_T2, ylim = c(130, 165), data = subset(us_skinfold_data, zyg == 1))
plot(ht_T1 ~ht_T2, ylim = c(130, 165), data = subset(us_skinfold_data, zyg == 3))
par(mfrow = c(1, 1))  # back to as it was
```

---

xmuHasSquareBrackets *xmuHasSquareBrackets*

Description

Tests if an input has square brackets

Usage

```
xmuHasSquareBrackets(input)
```

Arguments

input	an input to test

Value

- TRUE/FALSE

See Also

Other xmu internal not for end user: umxModel, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
xmuHasSquareBrackets("A[1,2]")
```

xmuLabel_Matrix *xmuLabel_Matrix (not a user function)*

Description

This function will label all the free parameters in an mxMatrix

Usage

```
xmuLabel_Matrix(mx_matrix = NA, baseName = NA, setfree = FALSE,
  drop = 0, jiggle = NA, boundDiag = NA, suffix = "", verbose = TRUE,
  labelFixedCells = FALSE, overRideExisting = FALSE)
```

Arguments

mx_matrix	an mxMatrix
baseName	A base name for the labels NA
setfree	Whether to set free cells FALSE
drop	What values to drop 0
jiggle	= whether to jiggle start values
boundDiag	set diagonal element lbounds to this numeric value (default = NA = ignore)
suffix	a string to append to each label
verbose	how much feedback to give
labelFixedCells	= FALSE
overRideExisting	Whether to overRideExisting (Default FALSE)

Details

End users should just call umxLabel

Purpose: label the cells of an mxMatrix Detail: Defaults to the handy "matrixname_r1c1" where 1 is the row or column Use case: You should not use this: call umxLabel umx:::xmuLabel_Matrix(mxMatrix("Lower", 3, 3, values = 1, name = "a", byrow = TRUE), jiggle = .05, boundDiag = NA); umx:::xmuLabel_Matrix(mxMatrix("Full" , 3, 3, values = 1, name = "a", byrow = TRUE)); umx:::xmuLabel_Matrix(mxMatrix("Symm" , 3, 3, values = 1, name = "a", byrow = TRUE), jiggle = .05, boundDiag = NA); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "data.a")); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "data.a"), overRideExisting = TRUE); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "test"), overRideExisting = TRUE); See also: fit2 = omxSetParameters(fit1, labels = "a_r1c1", free = FALSE, value = 0, name = "drop_a_row1_c1")

Value

- The labeled mxMatrix

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

---

xmuLabel_MATRIX_Model *xmuLabel_MATRIX_Model (not a user function)*

Description

This function will label all the free parameters in a (non-RAM) OpenMx mxModel nb: We don't assume what each matrix is for. Instead, the function just sticks labels like "a_r1c1" into each cell i.e., matrixname _ r rowNumber c colNumber

Usage

```
xmuLabel_MATRIX_Model(model, suffix = "", verbose = TRUE)
```

Arguments

model	a matrix-style mxModel to label
suffix	a string to append to each label
verbose	how much feedback to give

Details

End users should just call umxLabel

Value

- The labeled mxModel

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFro xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
require(umx)
data(demoOneFactor)
m2 <- mxModel("One Factor",
mxMatrix("Full", 5, 1, values = 0.2, free = TRUE, name = "A"),
mxMatrix("Symm", 1, 1, values = 1, free = FALSE, name = "L"),
mxMatrix("Diag", 5, 5, values = 1, free = TRUE, name = "U"),
mxAlgebra(A %*% L %*% t(A) + U, name = "R"),
mxExpectationNormal("R", dimnames = names(demoOneFactor)),
mxFitFunctionML(),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m3 = umx:::xmuLabel_MATRIX_Model(m2)
m4 = umx:::xmuLabel_MATRIX_Model(m2, suffix = "male")
# explore these with omxGetParameters(m4)
```

---

xmuLabel_RAM_Model *xmuLabel_RAM_Model (not a user function)*

Description

This function will label all the free parameters in a RAM mxModel

Usage

```
xmuLabel_RAM_Model(model, suffix = "", labelFixedCells = TRUE,
  overRideExisting = FALSE, verbose = FALSE, name = NULL)
```

Arguments

model	a RAM mxModel to label
suffix	a string to append to each label
labelFixedCells	Whether to labelFixedCells (Default TRUE)
overRideExisting	Whether to overRideExisting (Default FALSE)
verbose	how much feedback to give
name	Add optional name parameter to rename returned model (default = leave it along)

Details

End users should just call umxLabel

Value

- The labeled mxModel

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

---

`xmuMakeDeviationThresholdsMatrices` *Make deviation threshold matrics*

Description

Purpose: return a mxRAMObjective(A = "A", S = "S", F = "F", M = "M", thresholds = "thresh"), mxData(df, type = "raw") usecase see: umxMakeThresholdMatrix

Usage

```
xmuMakeDeviationThresholdsMatrices(df, droplevels, verbose)
```

Arguments

`df`	a dataframe
`droplevels`	whether to droplevels or not
`verbose`	how verbose to be

Value

- list of matrices

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFrom
xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

xmuMakeOneHeadedPathsFromPathList *xmuMakeOneHeadedPathsFromPathList*

Description

Make one-headed paths

Usage

```
xmuMakeOneHeadedPathsFromPathList(sourceList, destinationList)
```

Arguments

sourceList	A sourceList
destinationList	A destinationList

Value

- added items

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeTwoHeadedPathsFro xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

xmuMakeThresholdsMatrices *xmuMakeThresholdsMatrices (not a user function)*

Description

You should not be calling this directly. This is not as reliable a strategy and likely to be superceeded...

Usage

```
xmuMakeThresholdsMatrices(df, droplevels = FALSE, verbose = FALSE)
```

Arguments

df	a data.frame containing the data for your mxData statement
droplevels	a binary asking if empty levels should be dropped (defaults to FALSE)
verbose	how much feedback to give (defaults to FALSE)

Value

- a list containing an mxMatrix called "thresh", an mxRAMObjective object, and an mxData object

References

- http://tbates.github.io

Examples

```
# x = mtcars
# x$cyl = mxFactor(x$cyl, levels = c(4,6,8))
# umx:::xmuMakeThresholdsMatrices(df = x, droplevels=FALSE, verbose= TRUE)
```

---

xmuMakeTwoHeadedPathsFromPathList

xmuMakeTwoHeadedPathsFromPathList

Description

Make two-headed paths

Usage

```
xmuMakeTwoHeadedPathsFromPathList(pathList)
```

Arguments

pathList	A pathlist

Value

- added items

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

xmuMaxLevels *xmuMaxLevels*

Description

Get the max levels from df

Usage

```
xmuMaxLevels(df, what = c("value", "name"))
```

Arguments

df	Dataframe to search through
what	Either "value" or "name" (of the max-level column)

Value

- max number of levels in frame

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
xmuMaxLevels(mtcars) # NA = no ordinal vars
xmuMaxLevels(umxFactor(mtcars))
xmuMaxLevels(umxFactor(mtcars), what = "name")
```

xmuMI *xmuMI*

Description

A function to compute and report modifications which would improve fit. You will probably use umxMI instead

Usage

```
xmuMI(model, vector = TRUE)
```

Arguments

model	an mxModel to derive modification indices for
vector	= Whether to report the results as a vector default = TRUE

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathL xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

---

xmuMinLevels *xmuMinLevels*

Description

Get the min levels from df

Usage

```
xmuMinLevels(df, what = c("value", "name"))
```

Arguments

df	Dataframe to search through
what	Either "value" or "name" (of the min-level column)

Value

- min number of levels in frame

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
xmuMinLevels(mtcars) # NA = no ordinal vars
xmuMinLevels(umxFactor(mtcars))
xmuMinLevels(umxFactor(mtcars), what = "name")
```

xmuPropagateLabels *xmuPropagateLabels (not a user function)*

Description

You should be calling umxLabel. This function is called by xmuLabel_MATRIX_Model

Usage

```
xmuPropagateLabels(model, suffix = "", verbose = TRUE)
```

Arguments

model	a model to label
suffix	a string to append to each label
verbose	whether to say what is being done

Value

- mxModel

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 <- mxModel("One Factor", type = "RAM",
manifestVars = manifests, latentVars = latents,
mxPath(from = latents, to = manifests),
mxPath(from = manifests, arrows = 2),
mxPath(from = latents, arrows = 2, free = FALSE, values = 1.0),
mxData(cov(demoOneFactor), type = "cov", numObs = 500)
)
m1 = umx:::xmuPropagateLabels(m1, suffix = "MZ")
```

xmu_check_levels_identical *xmu_check_levels_identical*

Description

Just checks that the factor levels for twins 1 and 2 are the same

Usage

```
xmu_check_levels_identical(df, selDVs, sep, action = c("stop", "ignore"))
```

Arguments

df	data.frame containing the data
selDVs	base names of variables (without suffixes)
sep	text-constant separating base variable names the twin index (1:2)
action	if unequal levels found: c("stop", "ignore")

Value

-

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_start_value_list

Examples

```
require(umx)
data(twinData)
baseNames = c("bmi")
selDVs = umx_paste_names(baseNames, "", 1:2)
tmp = twinData[, selDVs]
tmp$bmi1[tmp$bmi1 <= 22] = 22
tmp$bmi2[tmp$bmi2 <= 22] = 22
xmu_check_levels_identical(umxFactor(tmp, sep = ""), selDVs = baseNames, sep = "")
## Not run:
xmu_check_levels_identical(umxFactor(tmp), selDVs = baseNames, sep = "")

## End(Not run)
```

xmu_dot_make_paths *xmu_dot_make_paths (not for end users)*

Description

Makes graphviz paths

Usage

```
xmu_dot_make_paths(mxMat, stringIn, heads = NULL, fixed = TRUE,
  comment = "More paths", showResiduals = TRUE, pathLabels = "labels",
  digits = 2)
```

Arguments

mxMat	An mxMatrix
stringIn	Input string
heads	1 or 2 arrows (default NULL - you must set this)
fixed	Whether show fixed values or not (defaults to TRUE)
comment	A comment to include
showResiduals	Whether to show residuals
pathLabels	labels
digits	how many digits to report

Value

- string

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_residuals, xmu_start_value_list

xmu_dot_make_residuals *xmu_dot_make_residuals (not for end users)*

Description

xmu_dot_make_residuals (not for end users)

Usage

```
xmu_dot_make_residuals(mxMat, latents = NULL, fixed = TRUE, digits = 2,
  resid = c("circle", "line"))
```

Arguments

mxMat	An A or S mxMatrix
latents	Optional list of latents to alter location of circles (defaults to NULL)
fixed	Whether to show fixed values or not
digits	How many digits to report
resid	How to show residuals and variances default is "circle". Other option is "line"

Value

- list of variance names and variances

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_start_value_list

xmu_start_value_list *Make start values*

Description

Purpose: Create startvalues for OpenMx paths use cases umx:::xmuStart_value_list(1) umxValues(1) # 1 value, varying around 1, with sd of .1 umxValues(1, n=letters) # length(letters) start values, with mean 1 and sd .1 umxValues(100, 15) # 1 start, with mean 100 and sd 15

Usage

```
xmu_start_value_list(mean = 1, sd = NA, n = 1)
```

Arguments

mean	the mean start value
sd	the sd of values
n	how many to generate

Value

- start value list

See Also

Other xmu internal not for end user: umxModel, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals

Index