Package ‘umx’

April 25, 2019

Version 2.10.0
Date 2019-04-29
Title Structural Equation and Twin Modeling in R
Maintainer Timothy C Bates <timothy.c.bates@gmail.com>
License GPL-3
Language en-US
Encoding UTF-8
URL https://github.com/tbates/umx
Description Quickly create, run, and report structural equation and twin models. See ‘?umx’ for help, and umx_open_CRAN_page("umx") for NEWS.
Depends R (>= 3.2.0), OpenMx (>= 2.11.5)
Imports cowplot, DiagrammeR, lavaan, ggplot2, knitr, MASS, Matrix, methods, MuMIn, mvtnorm, nlme, numDeriv, polycor, R2HTML, RCurl, sfsmisc, utils, xtable
Suggests cocor, devtools, gdata, Hmisc, spelling, testthat, rmarkdown, psych
BugReports https://github.com/tbates/umx/issues
LazyData true
RoxygenNote 6.1.1
NeedsCompilation no
Author Timothy C Bates [aut, cre] (<https://orcid.org/0000-0002-1153-9007>)
Repository CRAN
Date/Publication 2019-04-25 08:00:04 UTC

R topics documented:

dl_from_dropbox .................................................. 7
extractAIC.MxModel ............................................. 8
Fischbein_wt ...................................................... 9
R topics documented:

GFF ................................................................. 10
install.OpenMx ................................................. 12
iqdat ............................................................... 13
loadings .......................................................... 14
loadings.MxModel .............................................. 15
plot.MxLISRELModel ........................................... 16
plot.MxModel .................................................... 17
qm ................................................................. 19
reliability ....................................................... 20
residuals.MxModel ............................................. 21
RMSEA ........................................................... 22
RMSEA.MxModel ................................................ 23
RMSEA.summary.mxmodel ...................................... 24
tmx_genotypic_effect .......................................... 25
tmx_is_identified .............................................. 26
tmx_show ......................................................... 27
umx ............................................................... 29
umx-deprecated ................................................ 31
umxACE .......................................................... 33
umxACEcov ...................................................... 39
umxACEold ...................................................... 42
umxACEv ........................................................ 48
umxACE_cov_fixed ............................................ 54
umxAdd1 ........................................................ 56
umxAlgebra ...................................................... 57
umxAPA .......................................................... 58
umxBrownie ..................................................... 61
umxCI ............................................................. 62
umxCI_boot ...................................................... 64
umxCompare .................................................... 65
umxConfint ...................................................... 66
umxCov2cor ....................................................... 68
umxCovData ..................................................... 69
umxCP ............................................................. 70
umxCPold ......................................................... 75
umxDescribeDataWLS ......................................... 78
umxDiagnose ..................................................... 79
umxDrop1 ........................................................ 81
umxEFA .......................................................... 82
umxEquate ....................................................... 84
umxEval .......................................................... 86
umxExamples ................................................... 87
umxExpCov ....................................................... 92
umxExpMeans ................................................... 93
umxFactor ......................................................... 94
umxFactorScores ............................................... 96
umxFitIndices .................................................. 97
umxFixAll ....................................................... 98
R topics documented:

umxGetParameters .................................................. 99
umxGxE ............................................................... 100
umxGxEbiv ............................................................ 102
umxGxE_window ........................................................ 105
umxHetCor ............................................................ 107
umxIP ................................................................. 108
umxIPold .............................................................. 112
umxJiggle ............................................................. 114
umxLabel .............................................................. 115
umxLatent ............................................................ 116
umxLav2RAM .......................................................... 119
umxMatrix ............................................................ 121
umxMI ................................................................. 123
umxModel ............................................................. 124
umxModify ............................................................ 125
umxPadAndPruneForDefVars ........................................ 127
umxPath .............................................................. 129
umxPlotACE .......................................................... 132
umxPlotACEcov ...................................................... 134
umxPlotACEv ........................................................ 135
umxPlotCP .......................................................... 136
umxPlotCPold ........................................................ 138
umxPlotGxE .......................................................... 139
umxPlotGxEbiv ........................................................ 140
umxPlotIP ............................................................ 141
umxPlotSexLim ........................................................ 142
umxPlotSimplex ...................................................... 144
umxRAM .............................................................. 145
umxRAM2 ............................................................. 150
umxRAM2Ordinal ...................................................... 151
umxReduce ............................................................ 152
umxReduceACE ....................................................... 154
umxReduceGxE ....................................................... 155
umxRun ............................................................... 156
umxSetParameters ................................................... 157
umxSexLim ............................................................ 159
umxSimplex ........................................................... 163
umxSummarizeTwinData ............................................... 166
umxSummary .......................................................... 167
umxSummary.MxModel ................................................. 168
umxSummaryACE ..................................................... 170
umxSummaryACEcov .................................................. 171
umxSummaryACEv .................................................... 173
umxSummaryCP ........................................................ 174
umxSummaryGxE ....................................................... 176
umxSummaryGxEbiv ..................................................... 177
umxSummaryIP ......................................................... 179
umxSummarySexLim .................................................... 180
R topics documented:

- umxSummarySimplex ........................................ 183
- umxSuperModel ............................................. 184
- umxThresholdMatrix ....................................... 186
- umxTwoStage ................................................ 189
- umxUnexplainedCausalNexus ............................... 191
- umxValues ................................................... 192
- umxVersion .................................................. 193
- umxWeightedAIC ............................................ 194
- umx_aggregate .............................................. 195
- umx_APA_pval ............................................... 196
- umx_apply .................................................... 197
- umx_array_shift ............................................ 198
- umx_as_numeric ............................................. 199
- umx_cell_is_on ............................................. 200
- umx_check .................................................... 201
- umx_check_model .......................................... 202
- umx_check_names .......................................... 203
- umx_check_OS ............................................... 204
- umx_check_parallel ........................................ 205
- umx_cont_2_quantiles ...................................... 206
- umx_cor ...................................................... 208
- umx_cov2raw ................................................ 209
- umx_default_option ...................................... 210
- umx_dot_define_shapes ................................... 211
- umx_dot_mat2dot .......................................... 212
- umx_dot_rank ............................................... 214
- umx_drop_ok ................................................. 215
- umx_explode ................................................ 216
- umx_explode_twin_names .................................. 217
- umx_find_object .......................................... 218
- umx_fix_first_loadings ................................... 219
- umx_fix_latents .......................................... 220
- umx_fun_mean_sd ......................................... 221
- umx_get_bracket_addresses ................................ 222
- umx_get_checkpoint ...................................... 223
- umx_get_options .......................................... 224
- umx_grep .................................................... 225
- umx_has_been_run ....................................... 226
- umx_has_CIs ............................................... 227
- umx_has_means ............................................. 228
- umx_has_square_brackets ................................ 229
- umx_is_class ................................................. 230
- umx_is_cov .................................................. 231
- umx_is_endogenous ....................................... 232
- umx_is_exogenous ......................................... 233
- umx_is_MxData ............................................. 234
- umx_is_MxMatrix .......................................... 234
- umx_is_MxModel .......................................... 235
R topics documented:

umx_set_table_format ................................. 290
umx_show_fit_or_comparison .......................... 291
umx_stack ............................................ 292
umx_standardize ...................................... 293
umx_standardize_ACE .................................. 293
umx_standardize_ACEcov ................................ 294
umx_standardize_ACEv .................................. 295
umx_standardize_CP .................................... 296
umx_standardize_IP .................................... 297
umx_standardize_RAM ................................... 298
umx_standardize_SexLim ................................ 299
umx_standardize_Simplex ................................ 299
umx_stash_CIs ........................................ 300
umx_string_to_algebra .................................. 301
umx_swap_a_block ..................................... 302
umx_time ............................................... 303
umx_trim .............................................. 304
umx_var ............................................... 305
umx_wide2long ........................................ 306
umx_write_to_clipboard ................................ 307
us_skinfold_data ....................................... 308
xmuHasSquareBrackets .................................. 309
xmuLabel_Matrix ....................................... 310
xmuLabel_MATRIX_Model ................................ 311
xmuLabel_RAM_Model .................................... 312
xmuMakeDeviationThresholdsMatrices .................. 313
xmuMakeOneHeadedPathsFromPathList .................. 314
xmuMakeTwoHeadedPathsFromPathList .................. 315
xmuMaxLevels .......................................... 315
xmuMI .................................................. 316
xmuMinLevels .......................................... 317
xmuPropagateLabels ..................................... 317
xmu_assemble_twin_supermodel ......................... 318
xmu_check_levels_identical ............................ 319
xmu_check_variance .................................... 320
xmu_clean_label ....................................... 321
xmu_dot_maker .......................................... 322
xmu_dot_make_paths .................................... 322
xmu_dot_make_residuals ................................ 323
xmu_dot_move_ranks ................................... 324
xmu_dot_rank_str ...................................... 325
xmu_get_CI ............................................ 326
xmu_lavaan_process_group .............................. 327
xmu_make_mxData ...................................... 328
xmu_make_top_twin ..................................... 329
xmu_model_needs_means ................................ 333
xmu_safe_run_summary ................................... 334
xmu_set_sep_from_suffix ................................ 335
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

- https://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_write_to_clipboard, 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)
extractAIC.mxModel  Extract AIC from MxModel

Description

Returns the AIC for an OpenMx model. Original Author: Brandmaier

Usage

```r
# 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

- [https://openmx.ssri.psu.edu/thread/931#comment-4858](https://openmx.ssri.psu.edu/thread/931#comment-4858)

See Also

- `AIC`, `umxCompare`, `logLik`

Other Reporting functions: RMSEA, MxModel, RMSEA, summary, mxmodel, RMSEA, loadings, residuals, MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary, MxModel, umxSummaryACEv, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples

```r
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
ml <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
        umxPath(latents, to = manifests),
        umxPath(var = manifests),
        umxPath(var = latents,fixedAt = 1)
)
extractAIC(ml)
# -2.615998
AIC(ml)
```
**Fischbein_wt**

Weight data across time.

---

**Description**

A dataframe containing correlations of weight for 66 females measured 6 times at 6-month intervals.

**Usage**

```r
data(Fischbein_wt)
```

**Format**

A 6*6 correlation matrix based on n = 66 females

**Details**

- Weight1: Weight at time 1 (t0)
- Weight2: Weight at time 2 (t0 + 6 months)
- Weight3: Weight at time 3 (t0 + 12 months)
- Weight4: Weight at time 4 (t0 + 18 months)
- Weight5: Weight at time 5 (t0 + 24 months)
- Weight6: Weight at time 6 (t0 + 32 months)

Created as follows:

```r
Fischbein_wt = umx_read_lower(file = "", diag = TRUE, names = paste0("Weight", 1:6), ensurePD= TRUE)
```

```
1.000
0.985 1.000
0.968 0.981 1.000
0.957 0.970 0.985 1.000
0.932 0.940 0.964 0.975 1.000
0.890 0.897 0.927 0.949 0.973 1.000
```

**References**


**See Also**

Other datasets: `GFF`, `iqdat`, `umx`, `us_skinfold_data`

**Examples**

```r
data(Fischbein_wt) # load the data
df = Fischbein_wt  # data.frame
as.matrix(Fischbein_wt) # convert to matrix
```
**Description**

Measures of family functioning, happiness and related variables in twins, and their brothers and sisters. (see details)

**Usage**

```r
data(GFF)
```

**Format**

A data frame with 1000 rows and 8 variables:

**Details**

Several scales in the data are described in van der Aa et al. (2010). General Family Functioning (GFF) refers to adolescents’ evaluations general family health vs. pathology. It assesses problem solving, communication, roles within the household, affection, and control. GFF was assessed with a Dutch translation of the General Functioning sub-scale of the McMaster Family Assessment Device (FAD) (Epstein et al., 1983).

Family Conflict (FC) refers to adolescents’ evaluations of the amount of openly expressed anger, aggression, and conflict among family members. Conflict sub-scale of the Family Environment Scale (FES) (Moos, 1974)

Quality of life in general (QLg) was assessed with the 10-step Cantril Ladder from best- to worst-possible life (Cantril, 1965).

- **zyg_6grp**  Six-level zygosity: MZMM, DZMM, MZFF, DZFF, DZMF, DZFM
- **zyg_2grp** Two-level zygosity measure: ‘MZ’, ‘DZ’
- **divorce** Parental divorce status: 0 = No, 1 = Yes
- **sex_T1** Sex of twin 1: 0 = "male", 1 = "female"
- **age_T1** Age of twin 1 (years)
- **gff_T1** General family functioning for twin 1
- **fc_T1** Family conflict sub-scale of the FES
- **qol_T1** Quality of life for twin 1
- **hap_T1** General happiness for twin 1
- **sat_T1** Satisfaction with life for twin 1
- **AD_T1** Anxiety and Depression for twin 1
- **SOMA_T1** Somatic complaints for twin 1
- **SOC_T1** Social problems for twin 1
- **THOU_T1** Thought disorder problems for twin 1
sex_T2  Sex of twin 2
age_T2  Age of twin 2
gff_T2  General family functioning for twin 2
fc_T2  Family conflict sub-scale of the FES
qol_T2  Quality of life for twin 2
hap_T2  General happiness for twin 2
sat_T2  Satisfaction with life for twin 2
AD_T2  Anxiety and Depression for twin 2
SOMA_T2  Somatic complaints for twin 2
SOC_T2  Social problems for twin 2
THOU_T2  Thought disorder problems for twin 2
sex_Ta  Sex of sib 1
age_Ta  Age of sib 1
gff_Ta  General family functioning for sib 1
fc_Ta  Family conflict sub-scale of the FES
qol_Ta  Quality of life for sib 1
hap_Ta  General happiness for sib 1
sat_Ta  Satisfaction with life for sib 1
AD_Ta  Anxiety and Depression for sib 1
SOMA_Ta  Somatic complaints for sib 1
SOC_Ta  Social problems for sib 1
THOU_Ta  Thought disorder problems for sib 1
sex_Ts  Sex of sib 2
age_Ts  Age of sib 2
gff_Ts  General family functioning for sib 2
fc_Ts  Family conflict sub-scale of the FES
qol_Ts  Quality of life for sib 2
hap_Ts  General happiness for sib 2
sat_Ts  Satisfaction with life for sib 2
AD_Ts  Anxiety and Depression for sib 2
SOMA_Ts  Somatic complaints for sib 2
SOC_Ts  Social problems for sib 2
THOU_Ts  Thought disorder problems for sib 2

References

install.OpenMx

See Also

Other datasets: Fischbein_wt, iqdat.umx, us_skinfold_data

Examples

# Twin 1 variables (end in '_1T')
data(GFF)
umx_names(GFF, "1$") # Just variables ending in 1 (twin 1)
str(GFF) # first few rows

m1 = umxACE(seldVs = "gff", sep = "_T",
                mzData = subset(GFF, zyg_2grp == "MZ"),
                dzData = subset(GFF, zyg_2grp == "DZ")
)

install.OpenMx

Install OpenMx, with choice of builds

Description

You can install OpenMx, including the latest parallel/NPSOL enabled build of OpenMx. Options are:

1. "NPSOL": Install from our repository (default: This is where we maintain binaries supporting parallel processing and NPSOL).
2. "travis": Install the latest travis built (currently MacOS only).
3. "CRAN": Install from CRAN.
4. "open travis build page": Open the list of travis builds in a browser window.

Usage

install.OpenMx(loc = c("NPSOL", "travis", "CRAN",
                "open travis build page", "UVa"), url = NULL, lib, repos = getOption("repos"))

Arguments

loc Which install to get: "UVa" (the default), "travis" (latest build), or open the travis list of builds on the web to view/pick a url.
url A custom URL if you have/need one (probably not). If you’re on a Mac, you can set this to "Finder" and the package selected in the Finder will be installed. Overrides other settings.
lib Where to install the package.
repos Which repository to use (ignored currently).
**iqdat**

**Value**

- 

**References**

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

**See Also**

`umxVersion`

Other Miscellaneous Utility Functions: `qm`, `umxBrownie`, `umxFactor`, `umxVersion`, `umx_array_shift`, `umx_cell_is_on`, `umx_cont_2_quantiles`, `umx_find_object`, `umx_make`, `umx_msg`, `umx_open_CRAN_page`, `umx_pad`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`, `umx_xmu_check_variance`

**Examples**

```r
## Not run:
install.OpenMx() # gets the NPSOL version
install.OpenMx("NPSOL") # gets the NPSOL version explicitly
install.OpenMx("CRAN") # Get the latest CRAN version
install.OpenMx("open travis build page") # Open web page of travis builds
## End(Not run)
```

---

**iqdat**

*Twin data: IQ measured longitudinally*

---

**Description**

Measures of IQ across four ages in 261 pairs of identical twins and 301 pairs of fraternal (DZ) twins. (see details)

**Usage**

`data(iqdat)`

**Format**

A data frame with 562 rows and 9 variables

**Details**

- `zygosity` Zygosity (MZ or DZ)
- `IQ_age1_T1` T1 IQ measured at age 1
- `IQ_age2_T1` T1 IQ measured at age 2
- `IQ_age3_T1` T1 IQ measured at age 3
• IQ_age4_T1 T1 IQ measured at age 4
• IQ_age1_T2 T2 IQ measured at age 1
• IQ_age2_T2 T2 IQ measured at age 2
• IQ_age3_T2 T2 IQ measured at age 3
• IQ_age4_T2 T2 IQ measured at age 4

References
TODO

See Also
Other datasets: Fischbein_wt, GFF, umx, us_skinfold_data

Examples
data(iqdat)
str(iqdat)
par(mfrow = c(1, 3)) # 1 rows and 3 columns
plot(IQ_age4_T1 ~ IQ_age4_T2, ylim = c(50, 150), data = subset(iqdat, zygosity == "MZ"))
plot(IQ_age4_T1 ~ IQ_age4_T2, ylim = c(50, 150), data = subset(iqdat, zygosity == "DZ"))
plot(IQ_age1_T1 ~ IQ_age4_T2, data = subset(iqdat, zygosity == "MZ"))
par(mfrow = c(1, 1)) # back to as it was

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
loadings.MxModel

Description

loadings extracts the factor loadings from an EFA (factor analysis) model. It behaves equivalently to stats::loadings, returning the loadings from an EFA (factor analysis). However it does not store the rotation matrix.

Usage

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

Arguments

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

Value

- loadings matrix

References


See Also

- factanal, loadings

Other Reporting Functions: tmx_is_identified, tmx_show, umxAP, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI

Examples

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

Create and display a graphical path diagram for a LISREL model.

**Description**

plot() produces SEM diagrams using DiagrammeR (or a graphviz application) to create the image. The commercial application “OmniGraffe” is great for editing these images.

**Usage**

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

**Arguments**

- `x`: A LISREL `mxModel` from which to make a path diagram
- `std`: Whether to standardize the model (default = FALSE).
- `fixed`: Whether to show fixed paths (defaults to TRUE)
- `means`: Whether to show means or not (default = TRUE)
- `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")
- `resid`: How to show residuals and variances default is "circle". Options are "line" & "none"
- `strip_zero`: Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
- `...`: Optional parameters

**Details**

If you use umx_set_plot_format("graphviz"), figures will open in a graphviz helper app (if installed). 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 OmniGraffe professional), then set “change all”.

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

*Note:* By default, plots open in your browser (or plot pane if using RStudio). The underlying format is graphviz.
References

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

Other Core Modeling Functions: plot.MxModel, umxAlgebra, umxMatrix, umxModify, umxPath, umxRAM, umxRun, umxSuperModel, umx

Other Plotting functions: plot.MxModel, umxPlotACEcov, umxPlotACEv, umxPlotACE, umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex, umx

Examples

```r
# plot()
# TODO get LISREL example model
# Figure out how to map its matrices to plot. Don't do without establishing demand.
```

plot.MxModel  

Create and display a graphical path diagram for a model.

Description

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.

Usage

```r
## S3 method for class 'MxModel'
plot(x = NA, std = FALSE, fixed = TRUE,
     means = TRUE, digits = 2, file = "name", pathLabels = c("none", "labels", "both"), resid = c("circle", "line", "none"),
     strip_zero = FALSE, splines = TRUE, min = NULL, same = NULL, max = NULL, ...)
```

Arguments

- `x`: An `MxModel` from which to make a path diagram
- `std`: Whether to standardize the model (default = FALSE).
- `fixed`: Whether to show fixed paths (defaults to TRUE)
- `means`: Whether to show means or not (default = TRUE)
- `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. "none", "labels", or "both" (parameter + label).

resid: How to show residuals and variances default is "circle". Options are "line" & "none"

strip_zero: Whether to strip the leading "0" and decimal point from parameter estimates (default = FALSE)

splines: Whether to allow lines to curve: defaults to TRUE (nb: some models look better with FALSE)

min: optional list of objects to group at the top of the plot. Default (NULL) chooses automatically.

same: optional list of objects to group at the same rank in the plot. Default (NULL) chooses automatically.

max: optional list of objects to group at the bottom of the plot. Default (NULL) chooses automatically.

... Optional parameters

Details

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

Note: DiagrammeR is supported out of the box. By default, plots open in your browser.

If you use umx_set_plot_format("graphviz"), they will open in a graphviz helper app (if installed).
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


See Also

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

Other Core Modeling Functions: plot.MxLISRELModel, umxAlgebra, umxMatrix, umxModify, umxPath, umxRAM, umxRun, umxSuperModel, umx

Other Plotting functions: plot.MxLISRELModel, umxPlotACEcov, umxPlotACEv, umxPlotACE, umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex, umx

Examples

```r
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
ml <- umxRAM("One Factor", data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
```


qm

```r
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1)
)
plot(m1)
plot(m1, std = TRUE, resid = "line", digits = 3, strip_zero = FALSE)

m1 = umxRAM("grow", data = myGrowthMixtureData,
umxPath(var = manifests, free = TRUE),
umxPath(means = manifests, fixedAt = 0),
umxPath(v.m. = c("intercept","slope")),
umxPath("intercept", with = "slope"),
umxPath("intercept", to = manifests, fixedAt = 1),
umxPath("slope", to = manifests, arrows = 1, fixedAt = c(0, 1, 2, 3, 4))
)
plot(m1, means=FALSE, std=TRUE, strip=TRUE, splines="FALSE", max="intercept")
```

**Description**

Quickmatrix function

**Usage**

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

**See Also**

Other Miscellaneous Utility Functions: install.OpenMx, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance
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 Miscellaneous Stats Helpers: umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_round, umx_var, umx

Examples

# treat vehicle aspects as items of a test
data(mtcars)
reliability(cov(mtcars))
residuals.MxModel

Get residuals from an MxModel

Description

Return the residuals from an OpenMx RAM model. You can format these (with digits), and suppress small values.

Usage

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

Arguments

- `object`: An fitted `mxFit` 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


See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEV, umxSummary.MxModel, umxSummaryACEV, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples

```r
require(umx)
data(demoOneFactor)
latents = c("g")
manifests = names(demoOneFactor)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
            umxPath(latents, to = manifests),
            umxPath(var = manifests),
            umxPath(var = latents, fixedAt = 1.0)
)
```

# ===============
# = Show the residuals of the model =
# ============
residuals(m1)
# |
# | x1 | x2 | x3 | x4 | x5 |
# |----|----|----|----|----|
# | x1 | . | . | | . | 
# | x2 | . | . | 0.01 | | |
# | x3 | . | . | 0.01 | -0.01 | |
# | x4 | . | -0.01 | . | |
# | x5 | . | . | | |
# [1] "nb: You can zoom in on bad values with, e.g. suppress = .01, which
# will hide values smaller than this. Use digits = to round"

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)

#### See Also

Other Reporting functions: `RMSEA.MxModel`, `RMSEA.summary.mxmodel`, `extractAIC.MxModel`, `loadings.residuals.MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEV`, `umxSummary.MxModel`, `umxSummaryACEV`, `umxSummaryACE`, `umxSummarySexLim`, `umxSummarySimplex`, `umx_drop_ok`, `umx_standardize_RAM`
RMSEA.MxModel  RMSEA function for MxModels

Description

Compute the confidence interval on RMSEA

Usage

```r
## 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


See Also

Other Reporting functions: `RMSEA.summary.mxmodel`, `RMSEA`, `extractAIC.MxModel`, `loadings`, `residuals.MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEv`, `umxSummary.MxModel`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummarySexLim`, `umxSummarySimplex`, `umx_drop_ok`, `umx_standardize_RAM`

Examples

```r
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
```r
## 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

See Also
- Other Reporting functions: `RMSEA.MxModel`, `RMSEA`, `extractAIC.MxModel`, `loadings`, `residuals.MxModel`, `umxCI.boot`, `umxCI`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEv`, `umxSummary.MxModel`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummarySexLim`, `umxSummarySimplex`, `umx_drop_ok`, `umx_standardize_RAM`

Examples
```r
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)
```
Description

tmx_genotypic_effect allows you to explore the concept of genotypic effect at a locus. With it, you can interactively explore the effects of allele frequency, additive variance, and dominance.

This function lets you explore the simplest two-allele system (B and b), with three possible genotypes, BB, Bb, and bb.

The point between the two homozygotes is m – the mean effect of the homozygous genotypes.

Parameter a is half the measured phenotypic difference between the homozygotes BB and bb. It corresponds to the additive effect of each additional B allele, relative to the bb phenotype.

Parameter d is the deviation of the heterozygote Bb phenotype from the homozygote mid-point m. It corresponds to the non-additive (dominance) effect of the B allele. The heterozygote phenotype may lie on either side of m and the sign of d will vary accordingly.

Old system from book ed 2:

u = Frequency of the dominant allele (now = p).
v = Frequency of the recessive allele (now = q).

m = midpoint between the two homozygotes d = half the difference between the two homozygote (now a)
h = deviation of the heterozygote from m (now = d)

New system:
u and v -> p and q
d and h -> a and d

See BGBook issue 23

Usage

tmx_genotypic_effect(p = 0.75, q = (1 - p), a = 0.5, d = 0, m = 0, show = TRUE)

Arguments

p The frequency of the B allele (Default .5)
q The frequency of the b allele (Default 1-p)
a Half the difference between the two homozygote phenotypes (Default .5)
d The deviation of the heterozygote from m (Default 0)
m The value of the midpoint between the homozygotes (Default 0)
show Whether to draw the plot or just return it (Default = TRUE)
tmx_is.identified

Value

• optional plot

References


See Also

Other Teaching and testing Functions: tmx_is.identified, umx

Examples

library(umx);

# = Pure additivity: d = 0 =
# tmx_genotypic_effect(p = .5, a = 1, d = 0, m = 0, show = TRUE);

# = Complete dominance: a=d=1 =
# tmx_genotypic_effect(p = .5, q = .5, a = 1, d = 1, m = 0, show = TRUE);

# = Over dominance: a< d =1 =
# tmx_genotypic_effect(p = .5, q = .5, a =.5, d = 1, m = 0)

p = tmx_genotypic_effect(p = .5, q = .5, a = 1, d = .5, m = 0, show = TRUE);
# p + ggplot2::geom_point() + ggplot2::geom_text(hjust = 0, nudge_x = 0.05)
# ggsave(paste0(base, "c03_genotypic_effect_by_gene_dose.pdf"), width = 4.6, height = 4.6)

tmx_is.identified Test if a factor model is identified

Description

Test if a factor model is identified by establishing if the number of variables is equal too or grater than the number of model parameters. See also mxCheckIdentification for checking actual models.

Usage

tmx_is.identified(nVariables, nFactors)
Arguments

- `nVariables`: the number of variables measured.
- `nFactors`: the number of factors posited.

Value

- Binary

References

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

See Also

- [mxCheckIdentification](https://github.com/tbates/umx)

Other Teaching and testing Functions:
- [tmx_genotypic_effect.umx](https://github.com/tbates/umx)

Other Reporting Functions:
- [loadings.MxModel. tmx_show. umxAPA. umxEval. umxFactorScores. umxGetParameters. umxReduce. umxSummary. umxWeightedAIC. umx_APA_pval. umx_aggregate. umx_names. umx_parameters. umx_print. umx_show_fit_or_comparison. umx_time. umx_xmu_get_CI](https://github.com/tbates/umx)

Examples

```r
tmx_is.identified(nVariables = 2, nFactors = 1) # FALSE
tmx_is.identified(nVariables = 3, nFactors = 1) # TRUE
tmx_is.identified(nVariables = 4, nFactors = 2) # FALSE
tmx_is.identified(nVariables = 5, nFactors = 2) # TRUE
```

---

tmx_show Show matrices of RAM models in a easy-to-learn-from format.

Description

A great way to learn about models is to look at the matrix contents. `tmx_show` is designed to do this in a way that makes it easy to process for users: The matrix contents are formatted as tables, and can even be displayed as tables in a web browser.

Usage

```r
tmx_show(model, what = c("values", "free", "labels", "nonzero_or_free"),
          show = c("free", "fixed", "all"), matrices = c("S", "A", "M"),
          digits = 2, report = c("markdown", "inline", "html", "report"),
          na.print = "", zero.print = ".")
```
Arguments

- **model**: an `mxModel` from which to show parameters.
- **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. Default = round to 2 decimal places.
- **report**: How to report the results. "html" = open in browser.
- **na.print**: How to display NAs (default = ")
- **zero.print**: How to display 0 values (default = ")

Details

The user can select which matrices to view, whether to show values, free, and/or labels, and the precision of rounding.

Value

-

References

- https://tbates.github.io

See Also

Other Reporting Functions: `loadings.MxModel`, `tmx_is.identified`, `umxAPA`, `umxEval`, `umxFactorScores`, `umxGetParameters`, `umxReduce`, `umxSummary`, `umxWeightedAIC`, `umx APA_pval`, `umx aggregate`, `umx_names`, `umx_parameters`, `umx_print`, `umx_show_fit_or_comparison`, `umx_time`, `umx xmu_get_CI`

Examples

```r
require(umx)
data(demoOneFactor)
lats = c("g")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov", umxPath(lats, to = manifests), umxPath(var = manifests), umxPath(var = lats, fixedAt = 1.0))

tmx_show(m1)
 tmx_show(m1, digits = 3)
 tmx_show(m1, matrices = "S")
 tmx_show(m1, what = "free")
 tmx_show(m1, what = "labels")
 tmx_show(m1, what = "free", matrices = "A")
```
Functions for Structural Equation Modeling in OpenMx

Description

umx allows you to more easily build, run, modify, and report structural models, building on the OpenMx package. All core functions are organized into families, so they are easier to find (see "families" below under See Also).

All the functions have full-featured and well commented examples, some even have figures, so use the help, even if you think it won’t help ;-) Have a look, for example at umxRAM.

Check out NEWS about new features at news(package = "umx")

Details

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 https://tbates.github.io

If you want the bleeding-edge version:

devtools::install_github("tbates/umx")

References

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

See Also

Other Teaching and testing Functions: tmx_genotypic_effect, tmx_is_identified

Other Core Modeling Functions: plot.MxLISRELModel, plot.MxModel, umxAlgebra, umxMatrix, umxModify, umxPath, umxRAM, umxRun, umxSuperModel

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, xmu_get_CI

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

Other Plotting functions: plot.MxLISRELModel, plot.MxModel, umxPlotACEcov, umxPlotACEv, umxPlotACE, umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex

Other Super-easy helpers: umxEFA, umxLav2RAM, umxRAM2, umxTwoStage

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, xmu_twin_check
Other Twin Reporting Functions: `umxPlotCPold`, `umxPlotCP`, `umxReduceACE`, `umxReduceGxE`, `umxReduce`,
`umxSummarizeTwinData`

Other Twin Data functions: `umx_long2wide`, `umx_make_TwinData`, `umx_residualize`, `umx_scale_wide_twin_data`,
`umx_wide2long`

Other Get and set: `umx_default_option`, `umx_get_checkpoint`, `umx_get_options`, `umx_set_auto_plot`,
`umx_set_auto_run`, `umx_set_checkpt`, `umx_set_condensed_slots`, `umx_set_cores`, `umx_set_data_variance_check`,
`umx_set_optimization_options`, `umx_set_optimizer`, `umx_set_plot_file_suffix`, `umx_set_plot_format`,
`umx_set_table_format`

Other Check or test: `umx_check_names`, `umx_is_class`, `umx_is_numeric`, `xmu_twin_check`

Other Data Functions: `umxcovdata`, `umxdescribedatawls`, `umxhetcor`,
`umxpadandprunefordefvars`, `umx_as_numeric`, `umx_covraw`, `umx_lowerfull`, `umx_make_MR_data`, `umx_make_TwinData`,
`umx_make_bin_cont_pair_data`, `umx_make_fake_data`, `umx_polychoric`, `umx_polypairwise`, `umx_polytriowise`,
`umx_read_lower`, `umx_rename`, `umx_reorder`, `umx_stack`, `umx_swap_a_block`

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

Other String Functions: `umx_explode_twin_names`, `umx_explode`, `umx_grep`, `umx_names`,
`umx_object_as_str`, `umx_paste_names`, `umx_rot`, `umx_trim`, `umx_write_to_clipboard`

Other Miscellaneous Stats Helpers: `reliability`, `umxCovCor`, `umxHetCor`, `umx_apply`,
`umx_cor`, `umx_means`, `umx_r_test`, `umx_round`, `umx_var`

Other Miscellaneous Utility Functions: `installNopenmx`, `qm`, `umxbrownie`,
`umx_factor`, `umxversion`, `umx_array_shift`, `umx_cell_is_on`,
`umx_cont_R_quantiles`, `umx_find_object`, `umx_make`, `umx_msg`,
`umx_open_cran_page`, `umx_pad`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`,
`xmu_check_variance`

Other Advanced Model Building Functions: `umxjiggle`, `umxlabel`, `umxlatent`,
`umxRamOrdinal`, `umxthresholdmatrix`, `umxvalues`, `umx_fix_first_loadings`, `umx_fix_latents`,
`umx_get_bracket_addresses`, `umx_string_to_algebra`

Other zAdvanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEcov`, `umx_standardize_ACEv`,
`umx_standardize_ACE`, `umx_standardize_CP`, `umx_standardize_IP`, `umx_standardize_SexLim`,
`umx_standardize_Simplex`, `umx_stash_CIs`

Other xmu internal not for end user: `umxModel`, `xmuHasSquareBrackets`, `xmulabel MATRIX Model`,
`xmulabel Matrix`, `xmulabel_RAM Model`, `xmuMI`, `xmuMakeDeviationThresholdsMatrices`, `xmuMakeOneHeadedPathsFromPathList`,
`xmuMakeTwoHeadedPathsFromPathList`, `xmuMaxLevels`, `xmuMinLevels`, `xmuPropagateLabels`,
`xmu_assemble_twin_supermodel`, `xmu_check_levels_identical`, `xmu_clean_label`, `xmu_dot_make_paths`,
`xmu_dot_make_residuals`, `xmu_dot_maker`, `xmu_dot_move_ranks`, `xmu_dot_move_ranks`,
`xmu_lavaan_process_group`, `xmu_make_mxdata`, `xmu_make_top_twin`, `xmu_model_needs_means`, `xmu_safe_run_summary`,
`xmu_set_sep_from_suffix`, `xmu_simplex_corner`, `xmu_start_value_list`, `xmu_starts`

Examples

```r
require("umx")
data(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = nrow(demoOneFactor))
latents = c("G")
manifests = names(demoOneFactor)
ml <- 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) # OpenMx's SE-based confidence intervals
umxConfint(m1, parm = 'all', run = TRUE) # likelihood-based CIs

# 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

xmuMakeThresholdsMatrices should be replaced with umxThresholdMatrix
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

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 mxEvalByName instead

umxReportCI 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 umxAPA_pval

Arguments

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

References

**Description**

Implementing a core task in twin modeling, umxACE models the genetic and environmental structure of one or more phenotypes (measured variables) using the Cholesky ACE model (Neale and Cardon, 1996).

Classical twin modeling uses the genetic and environmental differences among pairs of monozygotic (MZ) and di-zygotic (DZ) twins reared together.

umxACE implements a 2-group model to capture these data and represent the phenotypic variance as a sum of Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

The following figure shows how the ACE model appears as a path diagram (for one variable):

![Path Diagram for ACE Model](image)

umxACE allows multivariate analyses, and this brings us to the Cholesky part of the model.

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 manifest into successively restricted factors. The following figure shows how the ACE model appears as a path diagram:

![Path Diagram for Multivariate ACE Model](image)

In this model, the variance-covariance matrix of the raw data is recovered as the product of the lower Cholesky and its transform.
This 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 figure also contains the key to understanding how to modify models that umxACE produces. Read the “Matrices and Labels in ACE model” section in details below...

NOTE: Scroll down to details for how to use the function, a figure and multiple examples.

Usage

```
umxACE(name = "ACE", selDVs, selCovs = NULL, dzData, mzData,
sep = NULL, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS",
"ULS"), allContinuousMethod = c("cumulants", "marginals"),
numObsDZ = NULL, numObsMZ = NULL, boundDiag = 0,
autoRun = getOption("umx_auto_run"), intervals = FALSE,
tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal",
"mxTryHardWideSearch"), optimizer = NULL, covMethod = c("fixed",
"random"), dzAr = 0.5, dzCr = 1, weightVar = NULL,
equateMeans = TRUE, addStd = TRUE, addCI = TRUE)
```

Arguments

- **name**
  The name of the model (defaults to "ACE").

- **selDVs**
  The variables to include from the data: preferably, just "dep" not c("dep_T1", "dep_T2").

- **selCovs**
  (optional) covariates to include from the data (do not include sep in names)

- **dzData**
  The DZ dataframe.

- **mzData**
  The MZ dataframe.

- **sep**
  The separator in twin variable names, often ",_T", e.g. "dep_T1". Simplifies selDVs.

- **type**
  Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS")

- **allContinuousMethod**
  "cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

- **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**
  Numeric lbound for diagonal of the a, c, and e matrices. Defaults to 0 since umx version 1.8

- **autoRun**
  Whether to run the model, and return that (default), or just to create it and return without running.

- **intervals**
  Whether to run mxCI confidence intervals (default = FALSE)

- **tryHard**
  Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

- **optimizer**
  Optionally set the optimizer (default NULL does nothing).

- **covMethod**
  How to treat covariates: "fixed" (default) or "random".
The DZ genetic correlation (defaults to .5, vary to examine assortative mating).

The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model).

If provided, a vector objective will be used to weight the data. (default = NULL).

Whether to equate the means across twins (defaults to TRUE).

Whether to add the algebras to compute a std model (defaults to TRUE).

Whether to add intervals to compute CIs (defaults to TRUE).

Details

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).

The type parameter can select how you want the model data treated. "FIML" is the normal treatment. "cov" and "cor" will turn raw data into cor data for analysis, or check that you’ve provided cor data as input.

Types "WLS", "DWLS", and "ULS" will process raw data into WLS data of these types.

The default, "Auto" will treat data as the type they are provided as.

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 modeling.

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 likelihoods summed to form the model-likelihood, which is to be minimized. 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 modeling family-level effects shared 100 to .25 to model dominance effects.

Matrices and Labels in ACE model

Matrices ’a’, ’c’, and ’e’ contain the path loadings of the Cholesky ACE factor model.

So, labels relevant to modifying the model are of the form "a_r1c1", ”c_r1c1" etc.

Variables are in rows, and factors are in columns. So to drop the influence of factor 2 on variable 3, you would say

```
code{m2 = umxModify(m1, update = "c_r3c2")}
```

Less commonly-modified matrices are the mean matrix expMean. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2.

So, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this script:

```
m1stop$expMean$labels[1, 4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
```

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 (Eaves et al. 1978 p267).
Value

- `mxModel` of subclass `mxModel.ACE`

References


See Also

- `umxPlotACE`, `umxSummaryACE`, `umxModify`

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxCPold`, `umxCP`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
# = How heritable is height?
require(umx)
data(twinData) # twinData from Australian twins.
# Pick the variables
selDVs = c("ht")
# 1. Height has a tiny variance, and this makes solution finding very hard.
# We'll scale height up by 10x to make the Optimizer's task easier.
twinData[, c("ht1", "ht2")] = twinData[, c("ht", "ht")]*10

# 2. umxACE picks the variables it needs from the data.
mzData <- twinData[twinData$zygosity == "MZ",]
dzData <- twinData[twinData$zygosity == "DZ",]

# 3. umxACE can figure out variable names using sep:
# e.g. selVars = "wt" + sep="_" -> "wt_T1" "wt_T2"m1 = umxACE(se1DVs = selDVs, sep = "", dzData = dzData, mzData = mzData)

# tip: with report = "html", umxSummary can print the table to your browser!
# un-standardized
# tip: plot gives a figure of the model and parameters
# plot(m1)

# = ADE: Evidence for dominance? (DZ correlation set to .25)
# m2 = umxACE(se1DVs = selDVs, sep = "", dzData = dzData, mzData = mzData, dzCr = .25)
# umxCompare(m2, m1) # ADE is better
# umxSummary(m2, comparison = m1)
# nb: Although summary is smart enough to print d, the underlying
C matrices are still called a, c & e.

C = WLS analysis =

```r
m3 = umxACE(selDVs = selDVs, sep = "", dzData = dzData, mzData = mzData, type = "DWLS", allContinuousMethod='marginals',)
```

C Things to note:

C 1. Weight has a large variance, and this makes solution finding very hard.
C We'll scale wt to make the Optimizer's task easier.

```r
twinData = umx_scaleWide_twin_data(data = twinData, varsToScale = c("wt"), sep = "")
mzData <- twinData[twinData$zygosity %in% "MZFF", ]
dzData <- twinData[twinData$zygosity %in% "DZFF", ]
```

C 4. note: the default boundDiag = 0 lower-bounds a, c, and e at 0.
C Prevents mirror-solutions. If not desired: set boundDiag = NULL.

```r
m1 = umxACE(selDVs = "wt", dzData = dzData, mzData = mzData, sep = "", boundDiag = NULL)
```

C MODEL MODIFICATION
C We can modify this model, e.g. test shared environment.
C Set comparison to modify, and show effect in one step.

```r
m2 = umxModify(m1, update = "c_r1c1", name = "no_c", comparison = TRUE)
```

C nb: You can see names of free parameters with parameters(m2)

```r
m2 = umxModify(m1, update = "c_r1c1", name = "no_c", comparison = TRUE)
```

C = Bivariate height and weight model =

```r
data(twinData)
```

```r
selDVs = c("ht", "wt") # umx will add sep (in this case "") + "1" or '2'
twinData = umx_scaleWide_twin_data(data = twinData, varsToScale = c("ht", "wt"), sep = "")
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,]
```

```r
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = ' ')
m2 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = ' ')
```

```r
type = "DWLS", allContinuousMethod='marginals')
```

```r
umxSummary(m1)
```

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

```r
# = Univariate model of weight =
# = Bivariate height and weight model =
# Ordinal example =
# =

deck(umx)
data(twinData)
twinData= umx_scale_wide_twin_data(data=twinData, varsToScale=c("wt"), sep="")
# Cut BMI column to form ordinal obesity variables
obLevels = c('normal', 'overweight', 'obese')
cuts <- quantile(twinData[, "bmi"], probs = c(.5, .2), na.rm = TRUE)
twinData$obese1=cut(twinData$bmi1, breaks=c(-Inf, cuts, Inf), labels=obLevels)
twinData$obese2=cut(twinData$bmi2, breaks=c(-Inf, cuts, Inf), labels=obLevels)
# Make the ordinal variables into umxFactors
ordDVs = c("obese1", "obese2")
twinData[, ordDVs] <- mxFactor(twinData[, ordDVs], levels = obLevels)
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
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!

# Data-prep done - here's where the model starts:
seLDVs = c("obese")
m1 = umxACE(seLDVs = selDVs, dzData = dzData, mzData = mzData, sep = '')
umxSummary(m1)

# Bivariate continuous and ordinal example =
# =

deck(umx)
data(twinData)
twinData = umx_scale_wide_twin_data(data=twinData, varsToScale="wt", sep="")
# Cut BMI column to form ordinal obesity variables
obLevels = c('normal', 'overweight', 'obese')
cuts = quantile(twinData[, "bmi"], probs = c(.5, .2), na.rm = TRUE)
twinData$obese1=cut(twinData$bmi1, breaks=c(-Inf, cuts, Inf), labels=obLevels)
twinData$obese2=cut(twinData$bmi2, breaks=c(-Inf, cuts, Inf), labels=obLevels)
# Make the ordinal variables into mxFactors
ordDVs = c("obese1", "obese2")
twinData[, ordDVs] = mxFactor(twinData[, ordDVs], levels = obLevels)
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
mzData <- mzData[1:80, ] # just top 80 so example runs in a couple of secs
dzData <- dzData[1:80, ]
m1 = umxACE(seLDVs = c("wt", "obese"), dzData = dzData, mzData = mzData, sep = '')

# Mixed continuous and binary example =
# =

require(umx)
data(twinData)
twinData = umx_scale_wide_twin_data(data = twinData, varsToScale = "wt", sep="")
# Cut to form category of RPE obese subjects
# and make into mxFactors (ensure ordered is TRUE, and require levels)
umxACEcov

Run a Cholesky with covariates, either fixed (def var in the means) or random (in the expected covariance matrix)

Description

Often, researchers include covariates in 2-group Cholesky umxACE twin models. A simple method is to regressing covariates from the data (see umx_residualize). A second method (supported in umxACEcov) is to include the covariates in the means model. This is the 'fixed' option for covariates models them in the mean as definition variables. On the plus side, there is no distributional assumption for this method. A downside of this approach is that all covariates must be non-NA, thus dropping any rows where one or more covariates are missing. This is wasteful of data.

Usage

umxACEcov(name = "ACEcov", selDVs, selCovs, dzData, mzData, sep = NULL, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"), dzAr = 0.5, dzCr = 1, addStd = TRUE, addCI = TRUE, boundDiag = 0,
equateMeans = TRUE, bVector = FALSE,
autoRun = getOption("umx_auto_run"), tryHard = c("no", "yes",
"mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"),
optimizer = NULL)

Arguments

name The name of the model (defaults to "ACE").
selDVs The variables to include from the data (do not include sep).
selCovs The covariates to include from the data (do not include sep).
dzData The DZ dataframe.
mzData The MZ dataframe.
sep Separator text between basename for twin variable names. Often ".T". Used to
expand selDVs into full column names, i.e., "dep" -> c("dep_T1", "dep_T2").
type Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS",
"ULS")
allContinuousMethod "cumulants" or "marginals". Used in all-continuous WLS data to determine if a
means model needed.
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).
autoRun Whether to run the model, and return that (default), or just to create it and return
without running.
tryHard Default (’no’) uses normal mxRun. "yes" uses mxTryHard. Other options: "mx-
TryHardOrdinal", "mxTryHardWideSearch"
optimizer optionally set the optimizer. Default (NULL) does nothing.

details

The umxACEcov 'random' option models the covariates in the expected covariance matrix, thus
allowing all data to be preserved. The downside is that this method has a strong assumption of
multivariate normality. Covariates like age, which are perfectly correlated in twins cannot be used.
Covariates like sex, which are ordinal, violate the normality assumption.

The following figure shows how the ACE model with random covariates appears as a path diagram:
Value

- `mxModel` of subclass `mxModel.ACEcov`

References


See Also

Other Twin Modeling Functions: `umxACE_cov_fixed, umxACEold, umxACEv, umxACE, umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check`

Examples

```r
# =DMI,can't use Age as a random covariate =
# =------------------------------------------------------------------------
require(umx)
data(twinData)
# Replicate age to age1 & age2
twinData$age1 = twinData$age2 = twinData$age
# 80 rows so example runs fast for CRAN
mzData = subset(twinData, zygosity == "MZFF")[1:80, ]
dzData = subset(twinData, zygosity == "DZFF")[1:80, ]
## Not run:
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
## End(Not run)```
implementing a core task in twin modeling, umxACEold models the genetic and environmental structure of one or more phenotypes (measured variables) using the Cholesky ACE model (Neale and Cardon, 1996).

Classical twin modeling uses the genetic and environmental differences among pairs of monozygotic (MZ) and di-zygotic (DZ) twins reared together.
umxACEold implements a 2-group model to capture these data and represent the phenotypic variance as a sum of Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

The following figure shows how the ACE model appears as a path diagram (for one variable):

![Path Diagram of ACE Model](image)

umxACEold allows multivariate analyses, and this brings us to the Cholesky part of the model. 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 manifest into successively restricted factors. The following figure shows how the ACE model appears as a path diagram:

![Path Diagram of Multivariate ACE Model](image)

In this model, the variance-covariance matrix of the raw data is recovered as the product of the lower Cholesky and its transform.

This 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 figure also contains the key to understanding how to modify models that umxACEold produces. Read the "Matrices and Labels in ACE model" section in details below...

**NOTE:** Scroll down to details for how to use the function, a figure and multiple examples.

**Usage**

```r
umxACEold(name = "ACE", selDVs, selCovs = NULL, covMethod = c("fixed", "random"), dzData, mzData, sep = NULL, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), dzAr = 0.5, dzCr = 1, addStd = TRUE, addCI = TRUE, numObsDZ = NULL, numObsMZ = NULL, boundDiag = 0)
```
Arguments

- **name**: The name of the model (defaults to "ACE").
- **selDVs**: The variables to include from the data: preferably, just "dep" not c("dep_T1", "dep_T2").
- **selCovs**: (optional) covariates to include from the data (do not include sep in names).
- **covMethod**: How to treat covariates: "fixed" (default) or "random".
- **dzData**: The DZ dataframe.
- **mzData**: The MZ dataframe.
- **sep**: The separator in twin variable names, often ".T", e.g. "dep_T1". Simplifies selDVs.
- **type**: Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS")
- **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**: Numeric lbound for diagonal of the a, c, and e matrices. Defaults to 0 since umx version 1.8
- **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).
- **autoRun**: Whether to run the model, and return that (default), or just to create it and return without running.
- **tryHard**: Default ("no") uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"
- **optimizer**: Optionally set the optimizer (default NULL does nothing).
- **intervals**: Whether to run mxCI confidence intervals (default = FALSE)

Details

**Data Input** The function flexibly accepts raw data, and also summary covariance data (in which case the user must also supply 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 modeling.
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 likelihoods summed to form the model-likelihood, which is to be minimized. This feature is used in the non-linear GxE model functions.

**Additional features** The umxACEold 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 modeling family-level effects shared 100 to .25 to model dominance effects.

**Matrices and Labels in ACE model**

Matrices 'a', 'c', and 'e' contain the path loadings of the Cholesky ACE factor model. So, labels relevant to modifying the model are of the form "a_r1c1", "c_r1c1" etc.

Variables are in rows, and factors are in columns. So to drop the influence of factor 2 on variable 3, you would say

```
m2 = umxModify(m1, update = "c_r3c2")
```

Less commonly-modified matrices are the mean matrix expMean. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2. So, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this script:

```
m1$top$expMean$labels[1, 4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
```

**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 (Eaves et al. 1978 p267).

**Value**

- mxModel of subclass mxModel.ACE

**References**


**See Also**

- plot.MxModelACE, plot.MxModelACE, umxSummaryACE, umxModify

*Other Twin Modeling Functions:* umxACE_cov_fixed, umxACEcov, umxACEv, umxACE, umxCPOld, umxCp, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

**Examples**

```r
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# = How heritable is height? =
```
require(umx)
data(twinData) # ?twinData from Australian twins.

# Pick the variables
twinData[,c("ht1", "ht2")]
twinData[,c("ht1", "ht2")]*100
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]

m = umxACEold(selDVs = "ht", sep = "", dzData = dzData, mzData = mzData) # -2ll= 9659, a1 = .92

umxSummary(m, std = FALSE) # un-standardized

# tip: with report = "html", umxSummary can print the table to your browser!
plot(m)

# Evidence for dominance (DZ correlation set to .25) =

m = umxACEold("ADE", selDVs = "ht", sep = "", dzData = dzData, mzData = mzData, dzCr = .25)

umxCompare(m2, m) # ADE is better

umxSummary(m2, comparison = m)

# nb: Although summary is smart enough to print d, the underlying matrixes are still called a, c & e.

# Univariate model of weight =

# Things to note:

# 1. This variable has a large variance, and this makes solution finding very hard.
# We'll scale weight to make the Optimizer's task easier.

twinData = umx_scale_wide_twin_data(data = twinData, varsToScale = c("wt"), sep = "")
mzData <- twinData[twinData$zygosity %in% "MZFF",]
dzData <- twinData[twinData$zygosity %in% "DZFF",]

# 2. umxACEold can figure out variable names: provide sep="_" and selVar = "wt" -> "wt_T1" "wt_T2"

# 3. umxACEold picks the variables it needs from the data.
# 4. expert user note: by default, umxACEold lower-bounds a, c, and e at 0.
# This prevents mirror-solutions.
# You can remove this by setting boundDiag = NULL

m = umxACEold(selDVs = "wt", dzData = dzData, mzData = mzData, sep = "")

# MODEL MODIFICATION
# We can modify this model, say testing shared environment, and see a comparison:

m2 = umxModify(m, update = "c_r1c1", name = "no_c", comparison = TRUE)

# nb: You can see names of free parameters with parameters(m)

# Bivariate height and weight model =

data(twinData)
```r
twinData = umx_scale_wide_twin_data(data = twinData, varsToScale = c("wt"), sep = "")
mdzData = twinData[,tvzygosity %in% c("MZFF", "MZMM", ]
dzData = twinData[,tzygosity %in% c("DZFF", "DZMM", "DZOS"), ]
mzData = mzData[1:80,] # quicker run to keep CRAN happy
dzData = dzData[1:80,]
selDV = c("ht", "wt") # umx will add sep (in this case ") + "1" or '2'
m1 = umxACEold(selDV = selDV, dzData = dzData, mzData = mzData, sep = ') # 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
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints = quantile(twinData[, "bmi"], probs = c(.5, .2), na.rm = TRUE)
twinData$sobese1 = cut(twinData$sbmi, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$sobese2 = cut(twinData$sbmi, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
ordVs = c("obese1", "obese2")
twinData[, ordVs] = mxFactor(twinData[, ordVs], levels = obesityLevels)
mzData = twinData[,tvzygosity %in% "MZFF", ]
dzData = twinData[,tvzygosity %in% "DZFF", ]
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!

# Data-prep done - here's where the model starts:
selDV = c("obese")
m1 = umxACEold(selDV = selDV, dzData = dzData, mzData = mzData, sep = ') # umxSummary(m1)

# ===================================================================================
# = Binvariate continuous and ordinal example =
# ===================================================================================
data(twinData)
twinData = umx_scale_wide_twin_data(data = twinData, varsToScale = c("wt"), sep = "")
# Cut BMI column to form ordinal obesity variables
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints = quantile(twinData[, "bmi"], probs = c(.5, .2), na.rm = TRUE)
twinData$sobese1 = cut(twinData$sbmi, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$sobese2 = cut(twinData$sbmi, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
ordVs = c("obese1", "obese2")
twinData[, ordVs] = mxFactor(twinData[, ordVs])
mzData = twinData[,tvzygosity %in% "MZFF", ]
dzData = twinData[,tvzygosity %in% "DZFF", ]
```
Build and run 2-group uni- or multi-variate ACE models based on VARIANCE (not paths).

Description

A common task in twin modeling 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).

Usage

umxACEv(name = "ACEv", selDVs, selCovs = NULL, sep = NULL, dzData, mzData, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"), dzAr = 0.5, dzCr = 1, addStd = TRUE, addCI = TRUE, numObsDZ = NULL, numObsMZ = NULL, boundDiag = NULL, weightVar = NULL, equateMeans = TRUE, bVector = FALSE, covMethod = c("fixed", "random"), autoRun = getOption("umx_auto_run"), tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"), optimizer = NULL)

Arguments

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

selDVs The variables to include from the data: preferably, just "dep" not c("dep_T1", "dep_T2").

selCovs (optional) covariates to include from the data (do not include sep in names)

sep The separator in twin var names, often "_T" in vars like "dep_T1". Simplifies selDVs.

dzData The DZ dataframe.

mzData The MZ dataframe.

type Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS").

allContinuousMethod "cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

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 = Numeric lbound for diagonal of the a, c, and e matrices. Default = NULL (no bound)

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).

covMethod How to treat covariates: "fixed" (default) or "random".
**autoRun**
Whether to run the model, and return that (default), or just to create it and return without running.

**tryHard**
Default ("no") uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

**optimizer**
Optionally set the optimizer (default NULL does nothing).

### Details
The ACE variance-based model decomposes phenotypic variance into additive genetic (A), unique environmental (E) and, optionally, either common environment (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 following figure shows the A components of a trivariate ACEv model:

```
NOTE: This function does not use the Cholesky decomposition. Instead it directly models variance. This ensures unbiased type-I error rates. It means that occasionally estimates of variance may be negative. This should be used as an occasion to inspect you model choices and data. ‘umxACEv’ can be used as a base model to validate the ACE Cholesky model, a core model in behavior genetics (Neale and Cardon, 1992).
```

**Data Input**
The function flexibly accepts raw data, and also summary covariance data (in which case the user must also supply 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.

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 likelihoods summed to form the model-likelihood, which is to be minimized. This feature is used in the non-linear GxE model functions.

**Additional features**
The umxACEv 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 modeling family-level effects shared 100 to .25 to model dominance effects.)
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 (Eaves et al. 1978 p267).

Value

- `mxModel` of subclass `mxModel.ACE`

References


See Also

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACE`, `umxCPold`, `umxCP`, `umxGxWindow`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
# = Univariate model of weight =
# = Univariate model of weight =
require(umx)
data(twinData) # twinData from Australian twins.

# Things to note: ACE model of weight will return a NEGATIVE variance in C.
# This is exactly why we have ACEv! It suggests we need a different model
# In this case: ADE.
# Other things to note:
# 1. `umxACEv` can figure out variable names: provide "sep", and selVars.
#    Function generates: "wt" -> "wt1" "wt2"
# 2. `umxACEv` picks the variables it needs from the data.

selDV = "wt"
mzData <- twinData[twinData$zygosity %in% "MZFF",]
dzData <- twinData[twinData$zygosity %in% "DZFF",]
m1 = umxACEv(selDV = selDV, sep = "", dzData = dzData, mzData = mzData)

# Evidence for dominance? (DZ correlation set to .25) =
# Evidence for dominance? (DZ correlation set to .25) =
m2 = umxACEv("ADE", selDV = selDV, sep = "", dzData = dzData, mzData = mzData, dzCr = .25)
# note: the underlying matrices are still called A, C, and E.
# I catch this in the summary table, so columns are labeled A, D, and E.
# However, currently, the plot will say A, C, E.
# We can modify this model, dropping dominance component (still called C),
```
# and see a comparison:
m3 = umxModify(m2, update = "C.r1c1", comparison = TRUE, name="AE")
# = Well done! Now you can make modify twin models in umx =
# = How heritable is height?
# = Note: Height has a small variance. umx can typically picks good starts,
#   but scaling is advisable.
#
# require(umx)
data(twinData) # twinData from Australian twins.
# height var is very small: move from m to cm to increase.
twinData[,c("ht1", "ht2")]= twinData[,c("ht1", "ht2")]*100
mzData <- twinData[twinData$zygosity %in% "MZFF", ]
dzData <- twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACEv(selDV = selDV, sep = "", dzData = dzData, mzData = mzData)
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) =
# = Ordinal example =
# = Bivariate height and weight model =
# = Quicker run to keep CRAN happy

# Cut bmi column to form ordinal obesity variables
ordDVs = c("obese1", "obese2")
umxACEv

selDV$s = c("obese")

obesity$levels = c('normal', 'overweight', 'obese')
cut$points = quantile(twin$Data[, "bmi"], probs = c(.5, .2), na.rm = TRUE)
twin$Data$obese$1 = cut(twin$Data$BMI, breaks = c(-Inf, cut$points, Inf), labels = obesity$levels)
twin$Data$obese$2 = cut(twin$Data$BMI, breaks = c(-Inf, cut$points, Inf), labels = obesity$levels)

# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
twin$Data[, ord$DV$s] = mxFactor(twin$Data[, ord$DV$s], levels = obesity$levels)
mz$Data = twin$Data[twin$Data$zygositiy %in% "MZFF", ][1:80,] # 80 pairs for speed
dz$Data = twin$Data[twin$Data$zygositiy %in% "DZFF", ][1:80,]
str(mz$Data) # make sure mz, dz, and t1 and t2 have the same levels!
ml = umxACEEv(selDV$s = selDV$s, dz$Data = dz$Data, mz$Data = mz$Data, sep = "")

umxSummary(ml)

# = Bivariate continuous and ordinal example =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
data(twin$Data)

mz$Data = twin$Data[twin$Data$zygositiy %in% "MZFF", ]
dz$Data = twin$Data[twin$Data$zygositiy %in% "DZFF", ]
ml = umxACEEv(selDV$s = selDV$s, dz$Data = dz$Data, mz$Data = mz$Data, sep = "")
plot(ml)

# = Mixed continuous and binary example =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
require(umx)
data(twin$Data)

mz$Data = twin$Data[twin$Data$zygositiy %in% "MZFF", ]
dz$Data = twin$Data[twin$Data$zygositiy %in% "DZFF", ]
ml = umxACEEv(selDV$s = selDV$s, dz$Data = dz$Data, mz$Data = mz$Data, sep = "")

# Not run:
ml = umxACEEv(selDV$s = selDV$s, dz$Data = dz$Data, mz$Data = mz$Data, sep = "")
umxACE_cov_fixed

```r
umxSummary(m1)
```

## End(Not run)

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

```r
require(umx)
data(twinData)
se1DVs = c("wt")
mz = cov(twinData[twinData$zygosity %in% "MZFF", tvars(se1DVs, "")], use = "complete")
dz = cov(twinData[twinData$zygosity %in% "DZFF", tvars(se1DVs, "")], use = "complete")
m1 = umxACEv(se1DVs = se1DVs, sep="", dzData = dz, mzData= mz, numObsDZ= 569, numObsMZ= 351)
```

---

**umxACE_cov_fixed** Run a Cholesky with covariates ("fixed" / definition variables in the means style)

**Description**

Often, it is appropriate to include covariates in models. A simple method is to regress covariates from the data using *lm*. This is a 'fixed' effects approach. *umx_residualize* makes this easier, even on twin data, and with complex regression formulas.

**Usage**

```r
umxACE_cov_fixed(name = "ACEcov", se1DVs, se1Covs = NULL, dzData, 
mzData, sep = NULL, dzAr = 0.5, dzCr = 1, addStd = TRUE, 
addCI = TRUE, boundDiag = 0, weightVar = NULL, 
equateMeans = TRUE, bVector = FALSE, optimizer = NULL, 
autoRun = getOption("umx_auto_run"), tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"))
```

**Arguments**

- **name**: The name of the model (defaults to "ACEcov").
- **se1DVs**: The variables to include from the data (do not include sep).
- **se1Covs**: The covariates to include from the data (do not include sep).
- **dzData**: The DZ dataframe.
- **mzData**: The MZ dataframe.
- **sep**: Separator text between basename for twin variable names. Often ".T". Used to expand se1DVs 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).
umxACE_cov_fixed

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dzCr</td>
<td>The DZ &quot;C&quot; correlation (defaults to 1: set to .25 to make an ADE model).</td>
</tr>
<tr>
<td>addStd</td>
<td>Whether to add the algebras to compute a std model (defaults to TRUE).</td>
</tr>
<tr>
<td>addCI</td>
<td>Whether to add intervals to compute CIs (defaults to TRUE).</td>
</tr>
<tr>
<td>boundDiag</td>
<td>Whether to bound the diagonal of the a, c, and e matrices.</td>
</tr>
<tr>
<td>weightVar</td>
<td>(optional) Variable containing the weights to apply to data.</td>
</tr>
<tr>
<td>equateMeans</td>
<td>Whether to equate the means across twins (defaults to TRUE).</td>
</tr>
<tr>
<td>bVector</td>
<td>Whether to compute row-wise likelihoods (defaults to FALSE).</td>
</tr>
<tr>
<td>optimizer</td>
<td>(optionally) set the optimizer. Default (NULL) does nothing.</td>
</tr>
<tr>
<td>autorun</td>
<td>Whether to run the model, and return that (default), or just to create it and return without running.</td>
</tr>
<tr>
<td>tryHard</td>
<td>Default (‘no’) uses normal mxRun. &quot;yes&quot; uses mxTryHard. Other options: &quot;mxTryHardOrdinal&quot;, &quot;mxTryHardWideSearch&quot;</td>
</tr>
</tbody>
</table>

Details

While these estimates are unbiased, modeling this regression in the means element of the twin model allows correct tests for significance. Also, if DVs are not continuous, the lm-based approach cannot be used.

For this reason, we have implemented umxACE_cov_fixed, which allows including covariates as definition variables. The following figure shows how the ACE model with fixed covariates appears as a path diagram:

On the plus side, there is no distributional assumption for this method. A downside of this approach is that all covariates must be non-NA, thus dropping any rows where one or more covariates are missing. This is wasteful of data, but often cannot be avoided (though see note below).

*note:* An alternative is the umxACEcov 'random' option. This model adds covariates to the expected covariance matrix, thus allowing all data to be preserved. The (BIG) downside is that this method has a strong assumption of multivariate normality. Covariates like age, which are perfectly correlated in twins cannot be used. Covariates like sex, which are ordinal, violate the normality assumption.
Value

- `mxModel` of subclass `mxModel.ACEcov`

See Also

`umx_residualize umxACE`

Other Twin Modeling Functions: `umxACEcov, umxACEold, umxACEv, umxACE, umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACE, umxSummaryACE, umxSummaryPC, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check`

Examples

```r
require(umx)
data(twinData) # ?twinData from Australian twins.
# Pick the variables
selDVs = "ht"
selCovs = "age"
mzData <- twinData[twinData$zygosity %in% "MZFF",]
dzData <- twinData[twinData$zygosity %in% "DZFF",]
m1 = umxACE_cov_fixed(selDVs = selDVs, selCovs = selCovs, sep = "",
                      dzData = dzData, mzData = mzData)
m2 = umxACE(selDVs = selDVs, sep = "", dzData = dzData, mzData = mzData)
# = lm-based equivalent =
# = lm-based equivalent =
df_res <- umx_residualize(ht ~ age, suffixes = c("1", "2"), data = twinData)
mzData <- df_res[df_res$zygosity %in% "MZFF",]
dzData <- df_res[df_res$zygosity %in% "DZFF",]
m3 = umxACE("lm_based", selDVs = selDVs, sep = "", dzData = dzData, mzData = mzData)
# = Example with two covariates =
# = Example with two covariates =
selDVs = "wt"
selCovs = c("age", "cohort")
twinData$cohort1 = twinData$cohort2 = as.numeric(as.factor(twinData$cohort))
mzData <- twinData[twinData$zygosity %in% "MZFF",]
dzData <- twinData[twinData$zygosity %in% "DZFF",]
m1 = umxACE_cov_fixed(selDVs = selDVs, selCovs = selCovs, sep = "",
                      dzData = dzData, mzData = mzData)
m1 = umxACE(selDVs = selDVs, sep = "", dzData = dzData, mzData = mzData)
```

---

**umxAdd1**

**umxAdd1**

**Description**

Add each of a set of paths you provide to the model, returning a table of their effects on fit.
umxAlgebra

Usage

```r
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

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

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

Examples

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

## End(Not run)
```

umxAlgebra

_A simple wrapper for mxAlgebra with name as the first parameter for more readable compact code._

Description

`umxAlgebra` is a wrapper for `mxAlgebra` which has the name parameter first in order.

Usage

```r
umxAlgebra(name = NA, expression, dimnames = NA, ..., fixed = FALSE,
            joinKey = as.character(NA), joinModel = as.character(NA),
            verbose = 0L)
```
Arguments

- **name**: The name of the matrix (Default = NA). Note the different order compared to `mxMatrix`!
- **expression**: The algebra.
- **dimnames** (Dimnames)
- **fixed**: See `mxAlgebra` documentation
- **joinKey**: See `mxAlgebra` documentation
- **joinModel**: See `mxAlgebra` documentation
- **verbose**: Quiet or informative

Value

- `mxAlgebra`

See Also

- `umxMatrix`

Other Core Modeling Functions: `plot.MxLISRELModel`, `plot.MxModel`, `umxMatrix`, `umxModify`, `umxPath`, `umxRAM`, `umxRun`, `umxSuperModel`, `umx`

Examples

```r
x = umxAlgebra("circ", 2 * pi)
class(x$formula)
x = mxAlgebra(name = "circ", 2 * pi)
class(x$formula) # "call"
```

---

**umxAPA**  
Creates nicely formatted journal style summaries of lm models, p-values, data-frames etc.

Description

`umxAPA` creates summaries from a range of inputs. Use it for reporting lm models, effects, and summarizing data.

1. Given an lm, will return a formatted effect, including 95% CI in square brackets, for one of the effects (specified by name in `se`). e.g.: `umxAPA(m1, "wt")` yields:
   \[ \beta = -5.344 \pm 6.486, -4.203 \], 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:
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

```r
umxAPA(obj = .Last.value, se = NULL, std = FALSE, digits = 2,
        use = "complete", min = 0.001, addComparison = NA,
        report = c("markdown", "html"), lower = TRUE, test = c("Chisq",
        "LRT", "Rao", "F", "Cp"), SES = TRUE, means = TRUE)
```

Arguments

- **obj**: A model (e.g. `lm`, `lme`, `glm`, t-test), beta-value, or data.frame
- **se**: If obj is a beta, se treated as standard-error (returning a CI). If obj is a model, used to select effect of interest (blank for all effects). Finally, set se to the CI c(lower, upper), to back out the SE.
- **std**: Whether to report std betas (re-runs model on standardized data).
- **digits**: How many digits to round output.
- **use**: If obj is a data.frame, how to handle NAs (default = "complete")
- **min**: For a p-value, the smallest value to report numerically (default .001)
- **addComparison**: For a p-value, whether to add "<=" default (NA) adds "<" if necessary
- **report**: What to return (default = 'markdown'). Use 'html' to open a web table.
- **lower**: Whether to not show the lower triangle of correlations for a data.frame (Default TRUE)
- **test**: If obj is a glm, which test to use to generate p-values options = "Chisq", "LRT". "Rao", "F", "Cp"
- **SEs**: Whether or not to show correlations with their SE (Default TRUE)
- **means**: Whether or not to show means in a correlation table (Default TRUE)

Value

- string

References

See Also

Other Reporting Functions: loadings_MxModel, tmx_is_identified, tmx_show, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx_xmu_get_CI

Examples

```
# = Report lm (regression/anova) results =
# =----------------------------------------------------------------------
umxAPA(lm(mpg ~ wt + disp, mtcars)) # All parameters
umxAPA(lm(mpg ~ wt + disp, mtcars), "disp") # Just disp effect
umxAPA(lm(mpg ~ wt + disp, mtcars), std = TRUE) # Standardize effects

# glm example
def = mtcars
df$mpg_thresh = 0
df$mpg_thresh[df$mpg>16] = 1
ml = glm(mpg_thresh ~ wt + gear, data = df, family = binomial)
numAPA(ml)

# A t-Test
ml = t.test(1:10, y = c(7:20))
umxAPA(ml)

# =----------------------------------------------------------------------
# = Summarize a DATA FRAME: Correlations + Means and SDs =
# =----------------------------------------------------------------------
umxAPA(mtcars[,1:3])
umxAPA(mtcars[,1:3], digits = 3)
umxAPA(mtcars[,1:3], lower = FALSE)
## Not run:
umxAPA(mtcars[,1:3], report = "html")
## End(Not run)

# =----------------------------------------------------------------------
# = CONFIDENCE INTERVAL text from effect and se =
# =----------------------------------------------------------------------
umxAPA(.4, .3) # parameter 2 interpreted as SE

# Input beta and CI, and back out the SE
umxAPA(-.030, c(-.073, .013), digits = 3)

# =----------------------------------------------------------------------
# = Format a p-value =
# =----------------------------------------------------------------------
umxAPA(.0182613)
umxAPA(.000182613)
umxAPA(.000182613, addComparison=FALSE)
```
### Description

How to cook steak.

### Usage

```r
umxBrownie()
```

### Details

Equipment matters. You should buy a heavy cast-iron skillet, and a digital internal thermometer. Preferably cook over a gas flame.

*note*: Cheaper cuts like blade steak can come out fine.

A great reference is *The Food Lab* by Kenji Alt Lopez. [https://www.amazon.co.uk/Food-Lab-Cooking-Through-Science/dp/0393081087](https://www.amazon.co.uk/Food-Lab-Cooking-Through-Science/dp/0393081087).

### References

- The Food Lab

### See Also

- omxBrownie

Other Miscellaneous Utility Functions: `install.OpenMx, qm, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance`

### Examples

```r
umxBrownie()
```
umxCI adds mxCI() calls for requested (default all) parameters in a model, runs these CIs if necessary, and reports them in a neat summary.

Usage

```r
umxCI(model = NULL, which = c("ALL", NA, "list of your making"),
       remove = FALSE, run = c("no", "yes", "if necessary", "show"),
       interval = 0.95, type = c("both", "lower", "upper"),
       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". 'show' means print the intervals if computed, or list their names if not.
- `interval` The interval for newly added CIs (defaults to 0.95)
- `type` The type of CI (defaults to "both", options are "lower" and "upper")
- `showErrorCodes` Whether to show errors (default == TRUE)

Details

umxCI 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 derivatives 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
- https://www.github.com/tbates/umx/

See Also
- `confint`, `umxConfint`, `umxCI`, `umxModify`

Other Reporting functions: `RMSEA`, `summary`, `confint.AIC`, `summary.AIC`, `loadings`, `residuals`, `mxCI_boot`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`,
`umxFitIndices`, `umxPlotACEv`, `umxSummary.AIC`, `umxSummarySexLim`,
`umxSummarySimplex`, `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))

m1$intervals # none yet - empty list()
m1 = umxCI(m1)
m1$intervals # $G_to_x1
m1 = umxCI(m1, remove = TRUE) # remove CIs from the model and return it

# ===========
# = A twin model example =
# ===========
data(twinData)
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
m1 = umxACE(seoDevs = c("bmi", "BMI2"), dzData = dzData, mzData = mzData)

# Not run:
umxCI(m1, run = "show") # show what will be requested
umxCI(m1, run = "yes") # actually compute the CIs
# Don't force update of CIs, but if they were just added, then calculate them
umxCI(m1, run = "if necessary")
m1 = umxCI(m1, remove = TRUE) # remove them all
m1$intervals # none!
# Show what parameters are available to get CIs on
umxParameters(m1)
# Request a CI by label:
m1 = umxCI(m1, "a_r1c1", run = "yes")

# End(Not run)
```
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

```r
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

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

See Also

- `umxExpMeans, umxExpCov`

Other Reporting functions: `RMSEA, MxModel, RMSEA, summary, mxmodel, RMSEA, extractAIC, MxModel, loadings, residuals, MxModel, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary, MxModel, umxSummaryACEv, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM`
umxCompare

print(a comparison table of one or more mxModels, formatted nicely.)

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 format (default is markdown, options include latex)
3. Table columns are arranged to make for easy comparison for readers.
4. report = 'inline', will provide an English sentence suitable for a paper.
5. report = "html" opens a web table in your browser to paste into a word processor.

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

Usage

umxCompare(base = NULL, comparison = NULL, all = TRUE, digits = 3, report = c("markdown", "inline", "html", "report"), compareWeightedAIC = FALSE, 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-values etc.
umxConfdint

Get confidence intervals from a umx model

**Description**

Implements confidence interval function for umx models.

**Parameters**

- `report`: "markdown" (default), "inline" (a sentence suitable 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!)
- `compareWeightedAIC`: Show the Wagenmakers AIC weighted comparison (default = FALSE)
- `file`: file to write html too if report = "html" (defaults to "tmp.html")

**Examples**

```r
require(umx)
data(demoOneFactor)latents = c("G")manifests = names(demoOneFactor)m1 <- umxRAM("One Factor", data = mxData(demoOneFactor), type = "cov", numObs = 500), umxPath(latents, to = manifests), umxPath(var = manifests), umxPath(var = latents, fixedAt = 1) m2 = umxModify(m1, update = "G_to_x2", name = "drop_path_2_x2")umxCompare(m1, m2)umxCompare(m1, m2, report = "report") # 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(m1, c(m2, m3), compareWeightedAIC = TRUE)umxCompare(c(m1, m2), c(m2, m3), all = TRUE)
```

**References**

- [https://www.github.com/tbates/umx/](https://www.github.com/tbates/umx/)

**See Also**

- `mxCompare, umxSummary, umxRAM`

Other Reporting functions: `RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA.extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxConfdint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary.MxModel, umxSummaryACEv, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM`
**umxConfint**

**Usage**

```r
umxConfint(object, parm = c("existing", "smart", "all", "or one or more labels"), wipeExistingRequests = TRUE, level = 0.95, run = FALSE, showErrorCodes = FALSE, optimizer = c("current", "SLSQP"))
```

**Arguments**

- **object**: An `mxModel`, possibly already containing `mxCI`s that have been `mxRun` with intervals = TRUE))
- **parm**: Which parameters to get confidence intervals. Can be "existing", "smart", "all", or a vector of names.
- **wipeExistingRequests**: Whether to remove existing CIs when adding new ones (ignored if parm = 'existing').
- **level**: The confidence level required (default = .95)
- **run**: Whether to run the model (defaults to FALSE)
- **showErrorCodes** (default = FALSE)
- **optimizer**: defaults to "SLSQP". Might try "NelderMead"
- **...**: Additional argument(s) for umxConfint.

**Details**

Note: By default, requesting new CIs wipes the existing ones. To keep these, set `wipeExistingRequests = FALSE`.

**Note**: `confint` is an OpenMx function which will return SE-based CIs. Because these can take time to run, by default only CIs already computed will be reported. Set `run = TRUE` to run new CIs. If `parm` is empty, and `run = FALSE`, a message will alert you to add `run = TRUE`.

**Value**

- `mxModel`

**References**

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

**See Also**

- `confint`, `umxCI`

Other Reporting functions: `RMSEA`, `MxModel`, `RMSEA.summary`, `mxmodel`, `RMSEA.extractAIC`, `MxModel`, `loadings`, `residuals`, `MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEv`, `umxSummary`, `MxModel`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummarySexLim`, `umxSummarySimplex`, `umx_drop_ok`, `umx_standardize_RAM`
Examples

```r
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))

m1 = umxConfint(m1, run = TRUE) # There are no existing CI requests...

# Add a CI request for "G_to_x1", run, and report. Save with this CI computed
m2 = umxConfint(m1, parm = "G_to_x1", run = TRUE)

# Just print out any existing CIs
umxConfint(m2)

# CI requests added for free matrix parameters. User prompted to set run = TRUE
m3 = umxConfint(m1, "all")

# Run the requested CIs
m3 = umxConfint(m3, run = TRUE)

# Run CIs for free one-headed (asymmetric) paths in RAM model.
# note: Deletes other existing requests,
tmp = umxConfint(m1, parm = "A", run = TRUE)

# Wipe existing CIs, add G_to_x1
tmp = umxConfint(m1, parm = "G_to_x1", run = TRUE, wipeExistingRequests = TRUE)

## Not run:
# For complex twin models, where algebras have parameters in some cells, smart might help
# note: only implemented for umxCov2cor so far
m2 = umxConfint(m1, "smart")

## End(Not run)
```

---

### umxCov2cor

(Convert a covariance matrix into a correlation matrix)

**Description**

A 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

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

See Also
cov2cor

Other Miscellaneous Stats Helpers: reliability, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_round, umx_var, umx

Examples

umxCov2cor(cov(mtcars[,1:5]))

umxCovData

Convert a dataframe into a cov mxData object

Description

umxCovData converts a dataframe into an mxData, taking the covariance, defaulting to nrow as the numObs, and optionally adding means.

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 an mxData type cov object.
- columns = Which columns to keep (default is all).
- use = Default is "complete.obs".

Value

- mxData of type = cov

References

See Also

Other Data Functions: `umxDescribeDataWLS`, `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_polychoric`, `umx_polypairwise`, `umx_polytriwise`, `umx_read_lower`, `umx_rename`, `umx_reorder`, `umx_stack`, `umx_swap_a_block`, `umx

Examples

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

\textbf{dzAr} The DZ genetic correlation (defaults to .5, vary to examine assortative mating).

\textbf{dzCr} The DZ "C" correlation (defaults to 1: set to .25 to make an ADE model).

\textbf{autoRun} Whether to run the model, and return that (default), or just to create it and return without running.

\textbf{tryHard} Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

\textbf{optimizer} optionally set the optimizer (default NULL does nothing).

\textbf{equateMeans} Whether to equate the means across twins (defaults to TRUE).

\textbf{weightVar} If provided, a vector objective will be used to weight the data. (default = NULL).

\textbf{bVector} Whether to compute row-wise likelihoods (defaults to FALSE).

\textbf{boundDiag} = Numeric lbound for diagonal of the a\_cp, c\_cp, & e\_cp matrices. Set = NULL to ignore.

\textbf{addStd} Whether to add the algebras to compute a std model (defaults to TRUE).

\textbf{addCI} Whether to add the interval requests for CIs (defaults to TRUE).

\textbf{numObsDZ} = not yet implemented: Ordinal Number of DZ twins: Set this if you input covariance data.

\textbf{numObsMZ} = not yet implemented: Ordinal Number of MZ twins: Set this if you input covariance data.

\textbf{freeLowerA} (ignore): Whether to leave the lower triangle of A free (default = FALSE).

\textbf{freeLowerC} (ignore): Whether to leave the lower triangle of C free (default = FALSE).

\textbf{freeLowerE} (ignore): Whether to leave the lower triangle of E free (default = FALSE).

**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 phenotype. Features include the ability to include more than one common pathway, to use ordinal data.

**note:** The function umx_set_optimization_options() allow users to see and set mvnReleps and mvnMaxPointsA It defaults to .001. You might find that ‘0.01’ works better for ordinal models.

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).

**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 modeling family-level effects shared 100 to .25 to model dominance effects.

**Matrices and Labels in CP model**
A good way to see which matrices are used in umxCP is to run an example model and plot it.
All the shared matrices are in the model "top".
Matrices top$as, top$cs, and top$es contain the path loadings specific to each variable on their diagonals.
So, to see the ‘as’ values, labels, or free states, you can say:
Labels relevant to modifying the specific loadings take the form "as_r1c1", "as_r2c2" etc. The common-pathway loadings on the factors are in matrices `top$a_cp`, `top$c_cp`, `top$e_cp`. The common factors themselves are in the matrix `top$cp_loadings` (an nVar * 1 matrix)

Less commonly-modified matrices are the mean matrix `expMean`. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2. So, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this line:

```r
m1$top$expMean$labels[1,4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
```

Value

- `mxModel`

References

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

See Also

- `umxSummaryCP`, `umxPlotCP`. See `umxACE` for more examples of twin modeling. `link{plot}` and `link{umxSummary}` work for IP, CP, GxE, SAT, and ACE models. For a deep dive, see `xmu_make_top_twin`

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPold`, `umxGxE_window`, `umxGxEbiv`, `umxGXE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
## Not run:
# require(umx)
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP("new", selDVs = selDVs, sep = ",", nFac = 3, optimizer = "SLSQP")
dzData = dzData, mzData = mzData, tryHard = "mxTryHardOrdinal")
mold = umxCPold("old", selDVs = selDVs, sep = "_", nFac = 3, dzData = dzData, mzData = mzData)

umxCompare(m1, mold)
# = Do it using WLS =
```
```r
# ==================================================
m2 <- umxCP("new", selDVs = selDVs, sep = ".T", nFac = 3, optimizer = "SLSQP",
dzData = dzData, mzData = mzData, 
tryHard = "mxTryHardOrdinal", type = "DWLS", allContinuousMethod = "marginals"
)
# ==================================================
# Find and test dropping of shared environment
# Show all labels for C parameters
umxParameters(m1, patt = "c")
# Test dropping the 9 specific and common-factor C paths
m2 <- umxModify(m1, regex = "((cs_.*)|(c._cp_.))", name = "dropC", comp = TRUE)
umxSummaryCP(m2, comparison = m1, file = NA)
umxCompare(m1, m2)
# ==================================================
# Mixed continuous and binary example
# ==============================================================
data(GFF)
# Cut to form umxFactor 20% depressed DEP
cutPoints <- quantile(GFF[, "AD_T1"], probs = .2, na.rm = TRUE)
ADLevels <- c("normal", "depressed")
GFF$DEP_T1 <- cut(GFF$AD_T1, breaks = c(-Inf, cutPoints, Inf), labels = ADLevels)
GFF$DEP_T2 <- cut(GFF$AD_T2, breaks = c(-Inf, cutPoints, Inf), labels = ADLevels)
ordDVs <- c("DEP_T1", "DEP_T2")
GFF[, ordDVs] <- umxFactor(GFF[, ordDVs])

selDV <- c("gff", "fc", "qol", "hap", "sat", "DEP")
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
# umx_set_optimizer("NPSOL")
# umx_set_optimization_options("mvnRelEps", .01)
m1 <- umxCP(selDV = selDV, sep = ".T", nFac = 3, dzData = dzData, mzData = mzData)
m2 <- umxModify(m1, regex = "((cs_r[3-5]|c._cp_r[12]))", name = "dropC", comp = TRUE)
# Do it using WLS
m3 <- umxCP(selDV = selDV, sep = ".T", nFac = 3, dzData = dzData, mzData = mzData,
# tryHard = "mxTryHardOrdinal", type = "DWLS")
# TODO fix WLS with umxCP
# label at row 1 and column 1 of matrix 'top.binLabels' in model 'CP3fac' : object 'Vtot'
# Correlated factors example
data(GFF)
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
selDV <- c("gff", "fc", "qol", "hap", "sat", "AD")
m1 <- umxCP("new", selDV = selDV, sep = ".T", dzData = dzData, mzData = mzData,
nFac = 3, correlatedA = TRUE, tryHard = "mxTryHard")
# End(Not run)
```
umxCpold

umxCpold: Build and run a Common pathway twin model

Description

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

Usage

```r
umxCpold(name = "CPold", selDVs, dzData, mzData, sep = NULL, nFac = 1, freeLowerA = FALSE, freeLowerC = FALSE, freeLowerE = FALSE, correlatedA = FALSE, equateMeans = TRUE, dzAr = 0.5, dzCr = 1, boundDiag = 0, addStd = TRUE, addCI = TRUE, numObsDZ = NULL, numObsMZ = NULL, autorun = getOption("umx_auto_run"), tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"), optimizer = NULL)
```

Arguments

- `name`: The name of the model (defaults to "CP").
- `selDVs`: The variables to include.
- `dzData`: The DZ dataframe.
- `mzData`: The MZ dataframe.
- `sep`: The suffix for twin 1 and twin 2, often ".T". If set, selDVs is just the base variable names, 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 = FALSE).
- `freeLowerC`: Whether to leave the lower triangle of C free (default = FALSE).
- `freeLowerE`: Whether to leave the lower triangle of E free (default = FALSE).
- `correlatedA`: ?? (default = FALSE).
- `equateMeans`: Whether to equate the means across twins (default = TRUE).
- `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).
- `boundDiag`: Numeric lbound for diagonal of the a_cp, c_cp, & e_cp matrices. Set = NULL to ignore.
- `addStd`: Whether to add the algebras to compute a std model (default = TRUE).
- `addCI`: Whether to add the interval requests for CIs (default = TRUE).
- `numObsDZ`: 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 run the model, and return that (default), or just to create it and return without running.

tryHard Default (’no’) uses normal mxRun. ”yes” uses mxTryHard. Other options: ”mxTryHardOrdinal”, ”mxTryHardWideSearch”

optimizer optionally set the optimizer (default NULL does nothing).

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 phenotype. 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).

**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 modeling family-level effects shared 100 to .25 to model dominance effects.

**Matrices and Labels in CP model** A good way to see which matrices are used in umxCP is to run an example model and plot it.

The diagonals of matrices as, cs, and es contain the path loadings specific to each variable. So labels relevant to modifying these are of the form "as_r1c1", "as_r2c2" etc. All the shared matrices are in the model "top". So to see the as values, you can say:

```
m$top$as$values
```

The common-pathway loadings on the factors are in matrices a_cp, c_cp, e_cp.

The common factors themselves are in the matrix cp_loadings (an nVar * 1 matrix)

Less commonly-modified matrices are the mean matrix expMean. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2.

So, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this script:

```
m$top$expMean$labels[1,4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
```

**Value**

- `mxModel`

**References**

- [https://www.github.com/tbates/umx](https://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: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCp`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`
### Examples
```r
## Not run:
require(umx)
data(GFF)
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
selDVs = c("gff","fc","qol","hap","sat","AD") # These will be expanded into "gff_T1" "gff_T2" etc.
m1 = umxCpold(selDVs = selDVs, sep = ".", nFac = 3, dzData = dzData, mzData = mzData)
summary(m1)
parameters(m1, patt = "^c")
m2 = umxModify(m1, regex = "(cs_.|$)|(c_.cp_.)", name = "dropC")
summaryCP(m2, comparison = m1, file = NA)
compare(m1, m2)
## End(Not run)
```

---

**umxDescribeDataWLS**

Determine whether a dataset will have weights and summary statistics for the means if used with *mxFitFunctionWLS*

---

**Description**

Given either a data.frame or an mxData of type raw, this function determines whether *mxFitFunctionWLS* will generate expectations for means.

**Usage**

```r
umxDescribeDataWLS(data, allContinuousMethod = c("cumulants", "marginals"), verbose = FALSE)
```

**Arguments**

- `data`: the (currently raw) data being used in a *mxFitFunctionWLS* model.
- `allContinuousMethod`: the method used to process data when all columns are continuous.
- `verbose`: logical. Whether to report diagnostics.

**Details**

All-continuous data processed using the "cumulants" method lack means, while all continuous data processed with `allContinuousMethod = "marginals"` will have means.

When data are not all continuous, `allContinuousMethod` is ignored, and means are modeled.

**Value**

- list describing the data.
umxDiagnose

See Also

- `mxFitFunctionWLS`, `omxAugmentDataWithWLSSummary`

Other Data Functions: `umxCovData`, `umxHetCor`, `umxPadAndPruneForDefVars`, `umx_as_numeric`, `umx_cov2raw`, `umx_lower2full`, `umx_make_MR_data`, `umx_make_bin_cont_pair_data`, `umx_make_fake_data`, `umx_polychoric`, `umx_polypairwise`, `umx_polytriowise`, `umx_read_lower`, `umx_rename`, `umx_reorder`, `umx_stack`, `umx_swap_a_block`, `umx``

Examples

```r
# = All continuous, data.frame input =
# ===================================

tmp = umxDescribeDataWLS(mtcars, allContinuousMethod = "cumulants", verbose = TRUE)
tmp$hasMeans # FALSE - no means with cumulants
tmp = umxDescribeDataWLS(mtcars, allContinuousMethod = "marginals")
tmp$hasMeans # TRUE we get means with marginals

# = mxData object as input =
# ==========================

tmp = mxData(mtcars, type="raw")
umxDescribeDataWLS(tmp, allContinuousMethod = "cumulants", verbose = TRUE)$hasMeans # FALSE
umxDescribeDataWLS(tmp, allContinuousMethod = "marginals")$hasMeans # TRUE

# = One var is a factor: Means modeled =
# =========================================

tmp = mtcars
tmp$cyl = factor(tmp$cyl)
umxDescribeDataWLS(tmp, allContinuousMethod = "cumulants")$hasMeans # TRUE - always has means
umxDescribeDataWLS(tmp, allContinuousMethod = "marginals")$hasMeans # TRUE
```

**umxDiagnose**

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

**Description**

The goal of this function is to diagnose problems in a model and return suggestions to the user. It is a work in progress, and of no use as yet.

**Usage**

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

model    an \texttt{mxFit} to diagnose
tryHard  whether I should try and fix it? (defaults to \texttt{FALSE})
diagonalizeExpCov
  Whether to diagonalize the \texttt{ExpCov}

Details

Best diagnostics are:

1. observed data variances and means
2. expected variances and means
3. Difference of these?

Try * diagonalizeExpCov diagonal. * \texttt{umx}_any_ordinal()
more tricky - we should really report the variances and the standardized thresholds.

Value

- helpful messages and perhaps a modified model

References

- \url{https://tbates.github.io,https://github.com/tbates/umx}

Examples

```r
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))
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**

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

**See Also**

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

**Examples**

```r
## 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)
```
FIML-based Exploratory Factor Analysis (EFA)

**Description**

Perform full-information maximum-likelihood factor analysis on a data matrix.

**Usage**

```r
umxefa(x = NULL, factors = NULL, data = NULL, n.obs = NULL, 
scores = c("none", "ML", "WeightedML", "Regression"), 
minmanifests = NA, rotation = c("varimax", "promax", "none"), 
name = "efa", digits = 2, return = c("model", "loadings"), 
report = c("markdown", "html"), covmat = NULL)
```

**Arguments**

- `x`: Either 1: data, 2: A formula (not implemented yet), 3: A vector 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.
- `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.
- `minmanifests`: The least number of variables required to return a score for a participant (Default = NA).
- `rotation`: A rotation to perform on the loadings (default = "varimax" (orthogonal)).
- `name`: A name for your model.
- `digits`: Rounding (default = 2).
- `return`: By default, the resulting MxModel is returned. Say "loadings" to get a fact.anal object.
- `report`: Report as markdown to the console, or open a table in browser ("html")
- `covmat`: Covariance matrix of data you are modeling (not implemented).

**Details**

As in `factanal`, you need only specify the number of factors and offer up some manifest data, e.g:

```r
umxefa(factors = 2, data = mtcars)
```

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

```r
umxefa(factors = c("g", "v"), data = mtcars)
```

The factor model is implemented as a structural equation model, e.g.
You can request scores from the model. Unlike factanal, these can cope with missing data.
You can also rotate the factors using any rotation function.
In an EFA, all items may load on all factors.

For identification we need $m^2/2$ degrees of freedom. We get $m \times (m+1)/2$ from fixing factor variances to 1 and covariances to 0. We get another $m(m-1)/2$ degrees of freedom by fixing the upper-right hand corner of the factor loadings component of the A matrix. The manifest variances are also bounded at 0.

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

*note:* Bear in mind that factor scores are indeterminate.

Thanks to @ConorDolan for code implementing the rotation matrix and other suggestions!

**Value**

- EFA mxModel

**References**

- [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

- factanal, mxFactorScores

Other Super-easy helpers: umxLav2RAM, umxRAM2, umxTwoStage, umx
**Examples**

```r
## Not run:
myVars <- c("mpg", "disp", "hp", "wt", "qsec")
ml = umxEFA(mtcars[, myVars], factors = 2, rotation = "promax")
loadings(ml)

# Formula interface in base-R factanal()
ml2 = factanal(~ mpg + disp + hp + wt + qsec, factors = 2, rotation = "promax", data = mtcars)
loadings(ml2)
plot(ml2)

# Return a loadings object
x = umxEFA(mtcars[, myVars], factors = 2, rotation = "promax")
names(x)

ml = umxEFA(myVars, factors = 2, data = mtcars, rotation = "promax")
ml = umxEFA(name = "named", factors = "g", data = mtcars[, myVars])
ml = umxEFA(name = "by_number", factors = 2, rotation = "promax", data = mtcars[, myVars])
x = 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 modeling 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**

```r
umxEquate(model, master, slave, free = c(TRUE, FALSE, NA),
verbose = FALSE, name = NULL, autoRun = FALSE, tryHard = c("no",
"yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"),
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
**umxEquate**

free  Should parameter(s) initially 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 run the model, and return that (default), or just to create it and return without running.
tryHard Default (‘no’) uses normal mxRun. “yes” uses mxTryHard. Other options: “mxTryHardOrdinal”, "mxTryHardWideSearch"
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 = TRUE

*Tip:* To find labels by name, use the regex parameter of **umxGetParameters**

**Value**

- mxModel

**References**

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

**See Also**

umxModify, umxCompare

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

**Examples**

```r
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))
  # By default, umxEquate just equates master and slave labels
m2 = umxEquate(m1, master = "G_to_x1", slave = "G_to_x2", name = "Eq x1 x2 loadings")
  # Set autoRun = TRUE and comparison = TRUE to run and output a comparison
m2 = umxEquate(m1, autoRun = TRUE, comparison = TRUE, name = "Eq x1 x2",
               master = "G_to_x1", slave = "G_to_x2")
```

umxEval

Description
Takes an expression as a string, and evaluates it as an expression in model, optionally computing the result. # TODO umxEval Currently broken... delete submit as update to OpenMx?

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

See Also
Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_ANP_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI

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"),
mxFitFunctionAlgebra(algebra = "ab", numObs= NA, numStats = NA)
)
m1 = mxRun(m1)
mxEval(list(ab = ab), m1)
Description

This is the example code used in our Twin Research and Human Genetics Paper on umx

Usage

umxExamples()

References


See Also

• umx

Examples

```r
## Not run:

# = Example code from Twin Research and Human Genetics Paper on umx(model) =

# Installing \pkg{umx} can be done using the R-code:
install.packages("umx")
# load as usual
library("umx")
# current package version can be shown with:
umxVersion("umx")

# Get the latest parallel and NPSOL build of OpenMx
install.OpenMx("NPSOL")

# NPSOL and multi-core version of OpenMx
install.OpenMx("travis")

# = CFA Code =

# Load the umx library (this is assumed in subsequent examples
library("umx")
# Load demo data consisting of 5 correlated variables, x1:x5
data(demoOneFactor)
```
# Create a list of the manifest variables for use in specifying the model
manifests <- as.list(paste0("x", 1:5))  # 'x1', 'x2', ...'x5'

# Create model cfal, with name 'CFA', data demoOneFactor, and the CFA paths.
cfal <- umxRAM("CFA", data = demoOneFactor,  
  # Create latent variable 'G', with fixed variance of 1 and mean of 0  
  umxPath(v = 1, m = 0, label = "G"),  
  # Create 5 manifest variables, x1:x5, with free variance and mean  
  umxPath(v = manifests, m = manifests),  
  # Create 1-headed paths from G to each of the manifests  
  umxPath("G", to = manifests))

# = Parameter labels =
# = Inspecting model parameters and residuals =

# Show parameters, below .1, with label containing 'x2'
parameters(cfal, "above", .1, pattern = "x2")
residuals(cfal, suppress = .005)

# = Modifying and comparing models =

# Variable names in the Duncan data
dimnames <- c("RespOccAsp", "RespEduAsp", "RespParAsp", "RespIQ", "RespSES",  
  "FrndOccAsp", "FrndEduAsp", "FrndParAsp", "FrndIQ", "FrndSES")

# lower-triangle of correlations among these variables
tmp <- c(  
  0.6247,  
  0.2137, 0.2742,
0.4105, 0.4043, 0.1839, 0.3240, 0.4047, 0.0489, 0.2220, 0.3269, 0.3669, 0.1124, 0.2903, 0.3054, 0.4216, 0.3275, 0.0839, 0.2598, 0.2786, 0.6404, 0.0760, 0.0702, 0.1147, 0.1021, 0.0931, 0.2784, 0.1988, 0.2995, 0.2863, 0.0782, 0.3355, 0.2302, 0.5191, 0.5007, 0.2087, 0.2930, 0.2407, 0.0186, 0.1861, 0.2707, 0.4105, 0.3067, -0.0438, 0.2950

# Use the umx_lower2full function to create a full correlation matrix
duncanCov = umx_lower2full(tmp, diag = FALSE, dimnames = dimnames)

# Turn the duncan data into an mxData object for the model
duncanCov = mxData(duncanCov, type = "cov", numObs = 300)

respondentFormants = c("RespSES", "FrndSES", "RespIQ", "RespParAsp")
friendFormants = c("FrndSES", "RespSES", "FrndIQ", "FrndParAsp")
latentAspiration = c("RespLatentAsp", "FrndLatentAsp")
respondentOutcomeAsp = c("RespOccAsp", "RespEduAsp")
friendOutcomeAsp = c("FrndOccAsp", "FrndEduAsp")

duncan1 = umxRAM("Duncan", data = duncanCov,
# Working from the left of the model, as laid out in the figure, to right...

# 1. Add all distinct paths between variables to allow the
# exogenous manifests to covary with each other.
umxPath(unique.bivariate = c(friendFormants, respondentFormants)),

# 2. Add variances for the exogenous manifests,
# These are assumed to be error-free in this model,
# and are fixed at their known value).
umxPath(var = c(friendFormants, respondentFormants), fixedAt = 1),

# 3. Paths from IQ, SES, and parental aspiration
# to latent aspiration for Respondents:
umxPath(respondentFormants, to = "RespLatentAsp"),
# And same for friends
umxPath(friendFormants, to = "FrndLatentAsp"),

# 4. Add residual variance for the two aspiration latent traits.
umxPath(var = latentAspiration),

# 5. Allow the latent traits each influence the other.
# This is done using fromEach, and the values are
# bounded to improve stability.
# note: Using one-label would equate these 2 influences
umxPath(fromEach = latentAspiration, lbound = 0, ubound = 1),

# 6. Allow latent aspiration to affect respondent's
# occupational & educational aspiration.
# note: firstAt = 1 is used to provide scale to the latent variables.
umxPath("RespLatentAsp", to = respondentOutcomeAsp, firstAt = 1),
# And their friends
umxPath("FrndLatentAsp", to = friendOutcomeAsp, firstAt = 1),

# 7. Finally, on the right hand side of figure, we add
# residual variance for the endogenous manifests.
umxPath(var = c/respondentOutcomeAsp, friendOutcomeAsp))
)

# = Collecting models =
# = Collecting models =

# Collect a list of paths to drop

# Modify the model duncan1, requesting a comparison table:
duncan2 = umxModify(duncan1, update = pathList, name = "No_influence", comparison = TRUE)

# An example using regex, to drop all paths beginning "G_to_
Cfa2 = umxModify(cfal, regex = "G_to.*")

# = Comparing models =
# = Comparing models =

umxCompare(duncan1, duncan2, report = "inline")

# To open the output as an html table in a browser, say:
umxCompare(duncan1, duncan2, report = "html")

# = Equating model parameters =
# = Equating model parameters =

parameters(duncan1, pattern = "IQ_to_")

duncan3 = umxModify(duncan1, name = "Equate IQ effect", comparison = TRUE,
master = "RespIQ_to_ResLatentAsp",
update = "FrndIQ_to_FrndLatentAsp"
)

# = ACE examples =
# = ACE examples =

require(umx);
# open the built in dataset of Australian height and weight twin data
data("twinData")
seIDVs = c("wt")
dz = twinData[twinData$zygosity == "DZFF", ]
mz = twinData[twinData$zygosity == "MZFF", ]
ACE1 = umxACE(selDV = selDVs, dzData = dz, mzData = mz, sep = "")
ACE2 = umxModify(ACE1, update = "c_r1c1", name = "dropC")
umxSummary(ACE1, std = FALSE, report = "html", digits = 3, comparison = ACE2)
parameters(ACE1)

ACE2 = umxModify(ACE1, update = "c_r1c1", name = "dropC")

# = Example Common Pathway model =
# = Example Common Pathway model =

# load twin data built into umx
data("twinData")

# Selecting the 'ht' and 'wt' variables
selDV = c("ht", "wt")
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")

# Run and report a common-pathway model
CP1 = umxCP(selDV = selDV, dzData = dzData, mzData = mzData, suffix = "")
paths = c("c_cp_r1c1", "cs_r1c1", "cs_r2c2")
CP2 = umxModify(CP1, update = paths, name = "dropC", comparison = TRUE)

# = Example Gene x environment model =
# = Example Gene x environment model =

data("twinData")
twinData$age1 = twinData$age2 = twinData$age
# Define the DV and definition variables
selDV = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
seVars = c(selDV, selDefs)

# Create datasets
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")

# Build, run and report the GxE model using selected DV and moderator
# umxGxE will remove and report rows with missing data in definition variables.
GE1 = umxGxE(selDV = selDV, selDefs = selDefs,
            dzData = dzData, mzData = mzData, dropMissingDef = TRUE)

# Shift the legend to the top right
umxSummary(GE1, location = "topright")

# plot standardized and raw output in separate graphs
umxSummary(GE1, separateGraphs = TRUE)

GE2 = umxModify(GE1, update = "am_r1c1", comparison = TRUE)
umxReduce(GE1)

# = Example GxE windowed analysis =
# = Example GxE windowed analysis =
require(umx);
data("twinData")
mod = "age"
selDVs = c("bmi1", "bmi2")

# select the younger cohort of twins
tmpTwin = twinData[twinData$cohort == "younger", ]
# Drop twins with missing moderator
tmpTwin = tmpTwin[!is.na(tmpTwin[mod]), ]
mzData = subset(tmpTwin, zygosity == "MZFF", c(selDVs, mod))
dzData = subset(tmpTwin, zygosity == "DZFF", c(selDVs, mod))
# toggle autoplot off, so we don't plot every level of the moderator
umx_set_auto_plot(FALSE)

umxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData)

## End(Not run)

### umxExpCov

Get the expected vcov matrix

**Description**

Extract the expected covariance matrix from an **mxModel**

**Usage**

```r
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**)
umxExpMeans

Value
- expected covariance matrix

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

See Also
- umxRun, umxCI_boot
Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary.MxModel, umxSummaryACEv, umxSummarySEXlim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples
require(umx)
data(demoOneFactor)
lats = c("g")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
  umxPath(lats, to = manifests),
  umxPath(var = manifests),
  umxPath(var = lats, fixedAt = 1.0)
)
vcov(m1) # supplied by OpenMx
umxExpCov(m1, digits = 3)

-----

umxExpMeans

Extract the expected means matrix from an mxModel

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)
lats Whether to select the latent variables (defaults to TRUE)
digits precision of reporting. Default (NULL) will not round at all.
Value

- expected means

References

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

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxFitIndices, umxPlotACEv, umxSummary.MxModel, umxSummaryACEv, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples

```r
require(umx)
data(demoOneFactor)
latents  = c("g")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, umxPath(latents, to = manifests),
   umxPath(var = manifests),
   umxPath(var = latents, fixedAt = 1.0)
)
umxExpMeans(m1)
umxExpMeans(m1, digits = 3)
```

Description

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

Usage

```r
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`
umxFactor

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

See Also
- umxFactanal

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxVersion, umx_array_shift,
   umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page,
   umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

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, "bmi1", 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)
umxFactorScores takes a model, and computes factors scores using the selected method (one of 'ML', 'WeightedML', or 'Regression'). It is a simple wrapper around mxFactorScores. For missing data, you must specify the least number of variables allowed for a score (subjects with fewer than minManifests will return a score of NA).

Usage

umxFactorScores(model, type = c("ML", "WeightedML", "Regression"), minManifests = NA)

Arguments

- model: The model to generate scores from.
- type: The method used to compute the score ('ML', 'WeightedML', or 'Regression').
- minManifests: The least number of variables required to return a score for a participant (Default = NA).

Value

- dataframe of scores.

References


See Also

- mxFactorScores

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI

Examples

m1 = umxEFA(mtcars, factors = 2)
x = umxFactorScores(m1, type = c('Regression'), minManifests = 3)
## Not run:
m1 = umxEFA(mtcars, factors = 1)
x = umxFactorScores(m1, type = c('Regression'), minManifests = 3)
x

## End(Not run)
umxFitIndices

Get additional fit-indices for a model with umxFitIndices

Description

A list of fit indices. Originated in this thread: https://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, MECVIchis, 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, extractAIC, MxModel, loadings, residuals, MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxPlotACEv, umxSummary, MxModel, umxSummaryACEv, umxSummaryACE, umxSummarySexlim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM
Examples

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

```r
# And with raw data
m1 <- umxRAM("m1", data = demoOneFactor,
umxPath(latents, to = manifests),
umxPath(v.m. = manifests),
umxPath(v.m@ = latents)
)
umxFitIndices(m1)
umxAPA(numxFitIndices(m1), digits = 3)
```

---

**umxFixAll**  
*umxFixAll: Fix all free parameters*

**Description**

Fix all free parameters in a model using omxGetParameters()

**Usage**

```r
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**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)
umxGetParameters

See Also

Other Modify or Compare Models: umxAdd1, umxDrop1, umxEquate, umxMI, umxModify, 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)
xmCompare(m1, m2)
```

umxGetParameters: Get parameters from a model, with support for pattern matching!

Description

umxGetParameters retrieves parameter labels from a model, like omxGetParameters. However, it is supercharged with regular expressions, so you can get labels that match a pattern.

Usage

```
umxGetParameters(inputTarget, regex = NA, free = NA,
                  fetch = c("values", "free", "lbound", "ubound", "all"),
                  verbose = FALSE)
```

Arguments

- **inputTarget**: An object to get parameters from: could be a RAM mxModel
- **regex**: A regular expression to filter the labels. Default (NA) returns all labels. Vector treated as raw labels to search for.
- **free**: A Boolean determining whether to return only free parameters.
- **fetch**: What to return: "values" (default) or "free", "lbound", "ubound", or "all"
- **verbose**: How much feedback to give

Details

In addition, if regex contains a vector, this is treated as a list of raw labels to search for, and return if all are found. note: To return all labels, just leave regex as is.
References
- https://www.github.com/tbates/umx

See Also
omxGetParameters, umx_parameters

Other Reporting Functions: loadings, MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI

Examples

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

# Show all parameters
umxGetParameters(ml)
umxGetParameters(ml, free = TRUE) # only parameters which are free
umxGetParameters(ml, free = FALSE) # only parameters which are fixed
# Complex regex patterns
umxGetParameters(ml, regex = "x[1-3]_with_x[2-5]", free = TRUE)
```

---

**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 (usually linear effects on path estimates) across levels of environment. umxGxE allows detecting, testing, and visualizing GxE (or C or E x E) interaction forms.

**Usage**

```r
umxGxE(name = "G_by_E", selDVs,.selDefs, dzData, mzData, sep = NULL,
lboundACE = NA, lboundM = NA, dropMissingDef = FALSE,
autorun = getOption("umx_auto_run"), tryHard = c("no", "yes",
"mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"),
optimizer = NULL)
```
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. socioeconomic 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)
sep
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.
tryHard
Default ("no") uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"
optimizer
optionally set the optimizer (default NULL does nothing)

Details

The following figure the GxE model as a path diagram:

Value

- GxE mxModel

References

See Also

- plot(), umxSummary, umxReduce

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE,
umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov,
umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP,
umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

Examples

require(umx)
data(twinData)

umx_set_optimizer("SLSQP")
twinData$age1 = twinData$age2 = twinData$age
selDVs = "bmi"

mzData = subset(twinData, zygosity == "MZFF")[100,]
dzData = subset(twinData, zygosity == "DZFF")[100,

umx_time("start")
m1 = umxGxE(selDVs = "bmi", selDefs = "age", sep = ",", dzData = dzData, mzData = mzData,

umx_time("stop")

## Not run:
# Controlling umxSummary

umxSummaryGxE(m1)

umxSummary(m1, location = "topright")

umxSummary(m1, separateGraphs = TRUE)

m2 = umxModify(m1, regex = "am_.*", comparison = TRUE, tryHard = "mxTryHard")

# umxReduce knows how to test all relevant hypotheses for GxE models,
# reporting these in a nice table.

umxReduce(m1)

## End(Not run)

---

\textit{umxGxEbiv}

\textit{Bivariate GxE models where twins differ on the moderator.}

Description

GxE interaction models test the hypothesis that the strength of genetic (or environmental) influence
varies parametrically across levels of environment. univariate (see \textit{umxGxE}) models assume the
twins share the moderator. \textit{umxGxE umxGxEbiv} allows testing moderation in cases where twins
differ on the moderator, (Purcell, 2002; van der Sluis et al., 2012).
umxGxEbiv

Usage

```r
umxGxEbiv(name = "GxEbiv", selDVs, selDefs, dzData, mzData, sep = NULL,
  lboundACE = NA, lboundM = NA, dropMissingDef = FALSE,
  autoRun = getOption("umx_auto_run"), tryHard = c("no", "yes",
    "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"),
  optimizer = NULL)
```

Arguments

- **name**: The name of the model (defaults to "GxEbiv")
- **selDVs**: The dependent variable (e.g. IQ)
- **selDefs**: The definition variable (e.g. socioeconomic 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)
- **sep**: Expand variable base names, i.e., "_T" makes var -> var_T1 and var_T2
- **lboundACE**: If !NA, then lbound the main effects at this value (default = NA)
- **lboundM**: If !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.
- **tryHard**: Default (‘no’) uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"
- **optimizer**: Optionally set the optimizer (default NULL does nothing)

Details

It supports testing, and visualizing GxE bivariate (or C or E x E) interactions.

The following figure shows the GxE model as a path diagram: *note*: Only Twin 1 is shown. Twin 1 and twin 2 A, C, and E latent traits are connected in the standard fashion, with the covariance of the T1 and T2 latent genetic traits set to .5 for DZ and 1.0 for MZ pairs. For the sake of clarity, C, and E
paths are omitted here. These mirror those for A.

**Value**

- GxEbiv mxModel

**References**


**See Also**

- plot(), umxSummary, umxReduce

Other Twin Modeling Functions: umxACE_c cov.Fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCovold, umxCov, umxGxE_window, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

**Examples**

```r
require(umx)
data(twinData)
```
### umxGxE_window

**Implement the moving-window form of GxE analysis.**

#### 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 (usually 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 center of the window highly relative to subjects far above or below the window center. This allows detecting and visualizing arbitrary GxE (or CxE or ExE) interaction forms.

#### Usage

```r
umxGxE_window(selDVs = NULL, moderator = NULL, mzData = mzData, 
    dzData = dzData, sep = 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.
umxGxE_window

mzData  | Dataframe containing the DV and moderator for MZ twins
dzData  | Dataframe containing the DV and moderator for DZ twins
sep     | (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


See Also

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPold`, `umxCP`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACEv`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
library(umx);
#
# umxGxE_window takes a dataframe consisting of a moderator and two DV columns: one for each twin.
# The model assumes two groups (MZ and DZ). Moderator can't be missing
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: "younger" and "older".
# zygosity is a factor. levels = MZFF, MZMM, DZFF, DZMM, DZOS.
# Delete missing moderator rows
twinData = twinData[!is.na(twinData[mod]), ]
mzData = subset(twinData, zygosity == "MZFF", c(selDVs, mod))
```
umxHetCor

Create a matrix of correlations for variables of diverse types (binary, ordinal, continuous)

Description

umxHetCor is a helper to 1. return just the correlations from John Fox’s polycor::hetcor function 2. If you give it a covariance matrix, return the nearest positive-definite correlation matrix.

Usage

```
umxHetCor(data, ML = FALSE, use = c("pairwise.complete.obs", "complete.obs"),
treatAllAsFactor = FALSE, verbose = FALSE,
return = c("correlations", "hetcor object"), std.err = FALSE)
```

Arguments

data A data.frame of columns for which to compute heterochoric correlations. OR an existing covariance matrix.

ML Whether to use Maximum likelihood computation of correlations (default = FALSE)

use How to handle missing data: Default="pairwise.complete.obs". Alternative ="complete.obs".

treatAllAsFactor Whether to treat all columns as factors, whether they are or not (Default = FALSE)

verbose How much to tell the user about what was done.
**Value**

- A matrix of correlations

**References**

-

**See Also**

Other Data Functions: `umxCovData`, `umxDescribeDataWLS`, `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_polychoric`, `umx_polypairwise`, `umx_polytriowise`, `umx_read_lower`, `umx_rename`, `umx_reorder`, `umx_stack`, `umx_swap_a_block`, `umx`

Other Miscellaneous Stats Helpers: `reliability`, `umxCov2cor`, `umx_apply`, `umx_cor`, `umx_fun_mean_sd`, `umx_means`, `umx_r_test`, `umx_round`, `umx_var`, `umx`

**Examples**

```r
umxHetCor(mtcars[, , c("mpg", "am")])

umxHetCor(mtcars[, , c("mpg", "am")], treatAllAsFactor = TRUE, 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 diagrammatically:
As can be seen, each phenotype also by default has A, C, and E influences specific to that phenotype. Features of the model include the ability to include add more one set of independent pathways, different numbers of pathways for a, c, and e, as well the ability to use ordinal data, and different fit functions, e.g. WLS.

**note:** The function `umx_set_optimization_options()` allow users to see and set `mvnRelEps` and `mvnMaxPoints`. It defaults to .001. You might find that '0.01' works better for ordinal models.

**Usage**

```r
umxIP(name = "IP", selDVs, dzData, mzData, sep = NULL, nFac = c(a = 1, c = 1, e = 1), type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"), dzAr = 0.5, dzCr = 1, correlatedA = FALSE, numObsDZ = NULL, numObsMZ = NULL, autoRun =getOption("umx_auto_run"), tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"), optimizer = NULL, equateMeans = TRUE, weightVar = NULL, addStd = TRUE, addCI = TRUE, freeLowerA = FALSE, freeLowerC = FALSE, freeLowerE = FALSE)
```

**Arguments**

- **name**
  - The name of the model (defaults to "IP").

- **selDVs**
  - The base names of the variables to model. note: Omit suffixes - just "dep" not c("dep_T1", "dep_T2")

- **dzData**
  - The DZ dataframe.

- **mzData**
  - The MZ dataframe.

- **sep**
  - The suffix for twin 1 and twin 2. e.g. selDVs= "dep", sep="_T" -> c("dep_T1", "dep_T2")

- **nFac**
  - How many common factors for a, c, and e. If one number is given, applies to all three.

- **type**
  - Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS")

- **allContinuousMethod**
  - "cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

- **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).

- **numObsDZ**
  - For cov data, the number of DZ pairs.

- **numObsMZ**
  - For cov data, the number of MZ pairs.

- **autoRun**
  - Whether to run and return the model (default), or just to create and return without running.

- **tryHard**
  - Whether to tryHard (default ‘no’ uses normal mxRun). options: "mxTryHard", "mxTryHardOrdinal", or "mxTryHardWideSearch"
optimizer optionally set the optimizer (default NULL does nothing).
equateMeans Whether to equate the means across twins (defaults to TRUE).
weightVar If a weighting variable is provided, a vector objective will be used to weight the data. (default = NULL).
addStd Whether to add algebras for a standardized model (defaults to TRUE).
addCI Whether to add CIs (defaults to TRUE).
freeLowerA ignore: Whether to leave the lower triangle of A free (default = FALSE).
freeLowerC ignore: Whether to leave the lower triangle of C free (default = FALSE).
freeLowerE ignore: Whether to leave the lower triangle of E free (default = FALSE).

Details
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 common factors which, in turn, account for variance in the phenotypes (see Figure).

Data Input
Currently, umxIP accepts only raw data. This may change in future versions. You can choose other fit functions, e.g. WLS.

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
umxIP 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 modeling family-level effects shared 100 to .25) to model dominance effects.

Matrices and Labels in CP model
A good way to see which matrices are used in umxIP is to run an example model and plot it.
All the shared matrices are in the model "top".
Matrices as, cs, and es contain the path loadings specific to each variable on their diagonals.
To see the 'as' values, you can simply execute:
m1$top#as$values
m1$top#as$labels
m1$top#as$free
Labels relevant to modifying the specific loadings take the form "as_r1c1", "as_r2c2" etc.
The independent-pathway loadings on the manifests are in matrices a_ip, c_ip, e_ip.
Less commonly-modified matrices are the mean matrix expMean. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2.
So, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this line:
m1$top$expMean$labels[1,4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
umxIP

Value

- `mxModel`

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

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

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPold`, `umxCp`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
# Not run:
require(umx)
data(GFF)
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
se1DVs = c("gff","fc","qol","hap","sat","AD") # These will be expanded into "gff_T1" "gff_T2" etc.
m1 = umxIP(se1DVs = se1DVs, sep = ".T", dzData = dzData, mzData = mzData)
m2 = umxIPold(se1DVs = se1DVs, sep = ".T", dzData = dzData, mzData = mzData)

# Use "marginals" method to enable all continuous data with missingness.
m3 = umxIP(se1DVs = se1DVs, sep = ".T", dzData = dzData, mzData = mzData,
type = "DWLS", allContinuousMethod='marginals')
# omit missing to enable default WLS method to work on all continuous data
dzD = na.omit(dzData[, tvars(se1DVs, ".T")])
mzD = na.omit(mzData[, tvars(se1DVs, ".T")])
m4 = umxIP(se1DVs = se1DVs, sep = ".T", dzData = dzD, mzData = mzD, type = "DWLS")

# = Try with a non-default number of a, c, and e independent factors =
# = Try with a non-default number of a, c, and e independent factors =
nFac = c(a = 2, c = 1, e = 1)
m2 = umxIP(se1DVs = se1DVs, sep = ".T", dzData = dzData, mzData = mzData, nFac = nFac,
tryHard = "mxTryHard")
umxCompare(m1, m2)

## End(Not run)
# TODO: sep enforcement: move to test case
# m1 = umxIP(se1DVs = se1DVs, dzData = dzData, mzData = mzData)
```
umxIPold

**umxIPold: 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 diagrammatically:

**Usage**

```r
umxIPold(name = "IP", selDVs, dzData, mzData, sep = NULL, nFac = c(a = 1, c = 1, e = 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 = getModelOption("umx_auto_run"), tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"), optimizer = NULL)
```

**Arguments**

- **name**: The name of the model (defaults to "IP").
- **selDVs**: The variables to include.
- **dzData**: The DZ dataframe.
- **mzData**: The MZ dataframe.
- **sep**: 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 for a, c, and e. If 1 number number is given, applies to all three.
- **freeLowerA**: Whether to leave the lower triangle of A free (default = FALSE).
- **freeLowerC**: Whether to leave the lower triangle of C free (default = FALSE).
- **freeLowerE**: Whether to leave the lower triangle of E free (default = FALSE).
- **equateMeans**: Whether to equate the means across twins (defaults to TRUE).
- **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 run the model, and return that (default), or just to create it and return without running.

tryHard: Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

optimizer: optionally set the optimizer (default NULL does nothing).

Details

Value

- mxModel

References

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

See Also

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

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCPOld, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

Examples

```r
# Not run:
require(umx)
data(GFF)
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
selIVs = c("gff","fc","qol","hap","sat","AD") # These will be expanded into "gff_T1" "gff_T2" etc.
m1 = umxIPold(selIVs = selIVs, sep = ",", dzData = dzData, mzData = mzData)
nFac = c(a=3, c = 1, e = 1)
```
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

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

See Also

Other Advanced Model Building Functions: umxLabel, umxLatent, umxRAMOrdinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx_get_bracket_addresses, umx_string_to_algebra, 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**: String to append to each label (might be used to distinguish, say male and female submodels in a model)
- **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**: Whether to label only the free paths (defaults to FALSE)
- **name**: Optional new name if given a model. Default (NULL) does not rename model.

Value

- mxModel

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

Other Advanced Model Building Functions: umxJiggle, umxLatent, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx_get_bracket_addresses, umx_string_to_algebra, umx
Examples

```r
# Show how OpenMx models are not labeled, and then add labels =
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
ml <- 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(ml) # Default "matrix address" labels, i.e "One Factor.S[2,2]"
ml = umxLabel(ml)
numxGetParameters(ml, free = TRUE) # Informative labels: "G_to_x1", "x4_with_x4", etc.

# Create a new model, with suffixes added to paths, and model renamed =
# suffixes added
m2 = umxLabel(ml, suffix = "_male", overRideExisting = TRUE, name = "male")
umxGetParameters(m2, free = TRUE) # suffixes added

# Example Labeling a matrix =
# suffixes added
a = umxLabel(mxMatrix(name = "a", "Full", 3, 3, values = 1:9))
a$labels
a = umxLabel(mxMatrix(name = "a", "Full", 3, 3, values = 1:9), baseName="bob")
a$labels
# note: labels with "data." in the name are left untouched!
a = mxMatrix(name = "a", "Full", 1,3, labels = c("data.a", "test", NA))
a$labels
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

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

Arguments

- `latent`: the name of the latent variable (string)
- `formedBy`: the list of manifest variables which latent reflects.
- `forms`: the list of variables which this latent forms (leave blank if using `formedBy`)
- `data`: the dataframe being used in this model
- `type`: of the latent variable: "exogenous" or "endogenous"
- `fixManifestVariances`: defaults to FALSE. Allows a model consisting of just a reflective latent to be identified.
- `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:

Note, a reflective latent on its own is not identified as a complete model. Fixing manifest variances at their observed values can allow this case.

Reflective (manifests reflect the value of the latent variable)

```
  G
 /|
/ | 
/  |
  x1 x2 x3 x4 x5
```

Formative (manifests provide the value of the latent variable)
Value
- path list

References
- https://www.github.com/tbates/umx

See Also
Other Advanced Model Building Functions: umxJiggle, umxLabel, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx_get_bracket_addresses, umx_string_to_algebra, umx

Examples
library(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor) # x1-5
theData = cov(demoOneFactor)
df = mxData(theData, type = "cov", numObs = nrow(demoOneFactor))
m1 = umxRAM("reflective", data = df,
umxLatent("G", forms = manifests, type = "exogenous", data = theData)
)
umxSummary(m1, show="std")
plot(m1, std = TRUE)

## Not run:
# I don’t recommend using umxLatent at present: It’s not a direction I am moving umx in
m2 = umxRAM("formative", data = df,
umxLatent("G", formedBy = manifests, data = df, fixManifestVariances=TRUE)
)
umxSummary(m2, show = "std")
plot(m2, std = TRUE)

## End(Not run)
Convert a lavaan syntax string to a umxRAM model

Description

Use lavaan syntax to create umxRAM models. If data are provided, a umxRAM model is returned. If more than one group is found, a superModel is returned.

This function is at the alpha quality stage, and should be expected to have bugs. Several features are not yet supported. Let me know if you’d like them.

Usage

```r
umxlav2RAM(model = NA, data = "auto", name = NULL,
            lavaanmode = "sem", printTab = TRUE, group = NULL)
```

Arguments

- `model`: A lavaan syntax string, e.g. "A~B"
- `data`: Data to add to model (defaults to auto, which is just sketch mode)
- `name`: Model name
- `lavaanmode`: Automagical path settings (default = "sem")
- `printTab`: = TRUE (more for debugging)
- `group`: = NULL TODO: define this

Details

Uses the defaults of `lavaan::sem`

- `int.ov.free` = TRUE
- `int.lv.free` = FALSE
- `auto.fix.first` = TRUE (unless `std.lv` = TRUE)
- `auto.fix.single` = TRUE
- `auto.var` = TRUE
- `auto.cov.lv.x` = TRUE
- `auto.th` = TRUE
- `auto.delta` = TRUE
- `auto.cov.y` = TRUE
- `fixed.x` = FALSE (not standard in lavaan::sem, but needed for RAM)

Lavaan is fabulously well documented. For quick reference, some common symbols in lavaan strings are

"=~"  lhs (Latent) is manifested by rhs
umxLav2RAM

"~"  lhs "is regressed on" (\(<\)) rhs
"~~"  lhs covarys with rhs
"~ 1"  lhs has mean
"::"  lhs is defined by rhs (see OpenMx::mxAlgebra())
"==="  lhs is constrained == to rhs (see OpenMx::mxConstraint())

Value

- list of umxPaths

See Also

- umxRAM

Other Super-easy helpers: umxEFA, umxRAM2, umxTwoStage, umx

Examples

# auto-data, print table, return umxRAM model
m1 = umxLav2RAM("y ~ x")

lav = "y ~ x1 + 2.4*x2 + x3"
tmp = umxLav2RAM(lav, data = "auto", printTab= FALSE)

# Add labels to parameters, e.g. "x3\_loading" as a loading for x3->x1
tmp = umxLav2RAM("x1 ~ x3\_loading\*x3")

# Fix values, e.g. x2 -> y fixed at 2.4
tmp = umxLav2RAM("y ~ x1 + 2.4*x2; s = -0*y11 + 1*y12 + 2*y13 + 3*y14")

tmp = umxLav2RAM("L = x1 + x2; L - Y")

# Factor model showing auto-addition of correlations among exogenous latents
# and auto-residuals on manifests
data("HS.ability.data", package = "OpenMx")

cov(HS.ability.data[, c("visual", "cubes", "flags")])
cov(HS.ability.data[, c("paragrap", "sentence", "wordm")])
cov(HS.ability.data[, c("addition", "counting", "straight")])

HS = "spatial =~ visual + cubes + flags
     verbal =~ paragrap + sentence + wordm
     speed =~ addition + counting + straight"

m1 = umxRAM2(HS, data = umx_scale(HS.ability.data))

# more complex:

lav = " # Model 14 PROCESS Hayes - moderated mediation
gnt = a*cb
INT = b1*gnt + b2*cn + b3*cn*gn + c*cb

indirect := a*b1"
umxMatrix

Make a mxMatrix with automatic labels. Also takes name as the first parameter for more readable code.

Description

umxMatrix is a wrapper for mxMatrix which labels cells by 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"), baseName = NA,
            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
### umxMatrix

```r
free 
values 
labels 
lbound 
ubound 
byrow 
basePath 
dimnames 
condenseSlots 
joinKey 
joinModel 
jiggle
```

**Values**

- `mxMatrix`

**References**

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

**See Also**

- `xmu_simplex_corner, mxMatrix, umxLabel, umxRAM`

Other Core Modeling Functions: `plotMxLISRELModel, plotMxModel, umxAlgebra, umxModify, umxPath, umxRAM, umxRun, umxSuperModel, umx`

**Examples**

```r
# = 1. Showing how name is first parameter, and how cells are labelled by default. =
# =-----------------------------------------------=
umxMatrix("test", "Full", 2, 2)$labels
# [1,]    [2,]
# [1,] "test_r1c1" "test_r1c2"
# [2,] "test_r2c1" "test_r2c2"

# = 2. Over-ride default (matrix name) as prefix for labels =
# =-----------------------------------------------=
numxMatrix("test", "Full", 2, 2, basename = "bob")$labels # bob_r1c1
```
umxMI

Report modifications which would improve fit.

Description

This function uses the mechanical modification-indices approach to detect single paths which, if added or dropped, would improve fit.

Usage

```r
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)

Details

Notes: 1. Runs much faster with full = FALSE (but this does not 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.

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)
See Also

- `mxMI`

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

Examples

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

```r
umxModel("")
```

Arguments

```r
... anything. We’re just going to throw an error.
```

Value

-`

References

umxModify

See Also
- umxRAM, mxModel

Other xmu internal not for end user: umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make_ranks, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

```r
## Not run:
umxModel()

## End(Not run)
```

---

### Description

umxModify allows you to modify, re-run and summarize an mxModel, all in one line of script.

#### Usage

```r
umxModify(lastFit, update = NULL, master = NULL, regex = FALSE, free = FALSE, value = 0, newlabels = NULL, freeToStart = NA, name = NULL, verbose = FALSE, intervals = FALSE, comparison = FALSE, autoRun =getOption("umx_auto_run"),
tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"))
```

#### 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.
- **master**
  If you set master, then the labels in update will be equated (slaved) to those provided in master.
- **regex**
  Whether or not update is a regular expression (default FALSE). If you provide a string, it overrides 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)
umxModify

newlabels If not NULL, used as a replacement set of labels (can be regular expression). value and free are ignored!
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() on the new and old models.
autoRun Whether to run the model, and return that (default), or just to create it and return without running.
tryHard Default (’no’) uses normal mxRun. “yes” uses mxTryHard. Other options: ”mxTryHardOrdinal”, ”mxTryHardWideSearch”

Details

You can add paths, or other model elements, set path values (default is 0), or replace labels. As an example, this one-liner drops a path labelled “Cs”, and returns the updated model:

\[
\text{fit2} = \text{umxModify(fit1, update = "Cs", name = "newModelName", comparison = TRUE)}
\]

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_and_Cr", comparison = TRUE)

You may 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)

Note: A (minor) limitation is that you cannot simultaneously set value to 0 AND relabel cells (because the default value is 0, so it is ignored when using newlabels).

Value

- mxModel

References

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

See Also

Other Core Modeling Functions: plot.MxLISRELModel, plot.MxModel, umxAlgebra, umxMatrix, umxPath, umxRAM, umxRun, umxSuperModel, umx

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

```R
require(umx)

# First we'll just build a 1-factor model
umx_set_optimizer("SLSQP")
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)

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

# 1. Drop the path to x1 (also updating the name so it's
#    self-explanatory, and get a fit comparison
m2 = umxModify(m1, update = "G_to_x1", name = "drop_X1", comparison = TRUE)
# 2. Add the path back (setting free = TRUE)
# 3. Fix a value at a non-zero value
m2 = umxModify(m1, update = "G_to_x1", free = TRUE, name = "addback_X1", comparison = TRUE)
# You can add objects to models. For instance this would add a path (overwriting the existing one)
# (thanks Johannes!)
# 4. Fix a value at a non-zero value
m3 = umxModify(m1, umxPath("G", with = "x1"), name = "addedPath")

# Use regular expression to drop multiple paths: e.g. G to x3, x4, x5
m2 = umxModify(m1, regex = "^G_to_x[3-5]", name = "no_G_to_x3_5", comparison = TRUE)

# Same, but don't autoRun
m2 = umxModify(m1, regex = "^G_to_x[3-5]", name = "no_G_to_x3_5", autoRun = FALSE)

# Re-write a label
newLabel = "A_rose_by_any_other_name"
newModelName = "model_doth_smell_as_sweet"
# Change labels in 2 places
m2 = umxModify(m1, update = "G_to_x1", newLabels= newLabel, name = newModelName, comparison = TRUE)

# Advanced!
# Regular expressions let you use pieces of the old names in creating new ones!
# searchString = "G_to_x([0-9])"
# newLabel = "loading_for_path\\1" # use value in regex group 1
m2 = umxModify(m1, regex = searchString, newLabels= newLabel, name = "grep", comparison = TRUE)
```
**umxPadAndPruneForDefVars**

**Description**

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

**Usage**

```r
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`: 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**

- dataframe

**References**


**See Also**

Other Data Functions: `umxCovData`, `umxDescribeDataWLS`, `umxHetCor`, `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_polychoric`, `umx_polypairwise`, `umx_polytriowise`, `umx_read_lower`, `umx_rename`, `umx_reorder`,
`umx_stack`, `umx_swap_a_block`, `umx`

**Examples**

```r
## Not run:
data(twinData)
sum(is.na(twinData$h1))
df = umxPadAndPruneForDefVars(twinData, varNames = "ht", defNames = "wt", c("1", "2"))

## End(Not run)
```
umxPath

Easier (and powerful) specification of paths in SEM.

Description

The goal of this function is to enable quick-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`, `v0m0`, `v.m0`, `v.m`, `fixedAt`, `freeAt`, `firstAt`, `unique.bivariate`, `unique.pairs`, `fromEach`, `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, you can say:

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

instead of `mxpath(from = A, to = A, arrows = 2, free = FALSE, values = 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” and “unique.pairs” 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 creates A<->A, B<->B, and A->B, you would say:

```
umxpath(unique.pairs = c('A',"B"))
```

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

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

`umxPath(fromEach = c('A','B','C'))` Creates one-headed arrows on the all.bivariate pattern

Setting up a latent trait, you can scale with a fixed first path thus:

```
umxpath("A", to = c("B","C","D"), firstAt = 1)
```

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

To create Cholesky-pattern connections:

```
umxpath(Cholesky = c("A1","A2"), to c("var1", "var2"))
```
Finally, not implemented in this release, but intended for the future is John Fox "sem"-package style notation,

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

**Usage**

```r
umxPath(from = NULL, to = NULL, with = NULL, var = NULL,
         cov = NULL, means = NULL, v1m0 = NULL, v.m. = NULL,
         v0m0 = NULL, v.m0 = NULL, fixedAt = NULL, freeAt = NULL,
         firstAt = NULL, unique.bivariate = NULL, unique.pairs = NULL,
         fromEach = NULL, forms = NULL, Cholesky = NULL, defn = 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`: One or more source variables e.g. "A" or c("A","B")
- `to`: One or more target variables for one-headed paths, e.g. "A" or c("A","B").
- `with`: 2-headed path <- from 'from' to 'with'.
- `var`: Equivalent to setting 'from' and 'arrows' = 2. nb: from, to, and with must be left empty.
- `cov`: Convenience to allow 2 variables to covary (equivalent to 'from' and 'with').
  nb: leave from, to, etc. empty
- `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)
- `unique.bivariate`: equivalent to setting from, and "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)
- `fromEach`: Like all.bivariate, but with one head arrows. "to" can be set.
- `forms`: Build a formative variable. "from" variables form the latent. Latent variance
  is fixed at 0. Loading of path 1 is fixed at 1. unique.bivariate between 'from'
  variables.
umxPath

Cholesky  Treat Cholesky variables as latent and to as measured, and connect as in an ACE model.
defn     Makes a latent variable, var@0 mean fixed, set label to ‘data.<defn>’
connect  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
1bound   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

- https://tbates.github.io

See Also

- mxPath

Other Core Modeling Functions: plot.MxLISRELModel, plot.MxModel, umxAlgebra, umxMatrix, umxModify, umxRAM, umxRun, umxSuperModel, umx

Examples

# A worked example
data(demoOneFactor)
lats $ = c(”G”)
manifests $ = names(demoOneFactor)
myData $ = mxData(cov(demoOneFactor), type = ”cov”, numObs = 500)
ml $<-$ umxRAM(”One Factor”, data = myData,
  umxPath(lats, to = manifests),
  umxPath(var = manifests),
  umxPath(var = lats, fixedAt = 1.0)
)
umxSummary(ml, show = ”std”)
require(umx)

# Examples of each path type, and option
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(vnm = "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"
numPath("A", with = "B", fixedAt = .5) # 2-head path fixed at .5
numPath("A", with = c("B", "C"), firstAt = 1) # first covariance fixed at 1
numPath(cov = c("A", "B")) # Covariance A <-> B
numPath(unique.bivariate = letters[1:4]) # bivariate paths a<-b, a<-c, a<-d, b<-c etc.
numPath(fromEach = letters[1:4]) # bivariate paths a<-b, a<-c, a<-d, b<-c etc.
numPath(unique.pairs = letters[1:4]) # bivariate paths a<-b, a<-c, a<-d, b<-c etc.
numPath(cholesky = c("A1","A2"), to = c("m1","m2")) # Cholesky

# ===============
# = Cholesky example =
# ===============

## Not run:
latents = paste0("A", 1:3)
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs = 500)
ml <- umxRAM("Chol", data = myData,
umxPath(cholesky = latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)

## End(Not run)

# 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

**Description**

Make a graphical display of an ACE model
Usage

umxPlotACE(x = NA, file = "name", digits = 2, means = FALSE,
           std = TRUE, strip_zero = TRUE, ...)

Arguments

x  
   mxModel to plot (created by umxACE in order to inherit the MxModelACE
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)
strip_zero  
   Whether to strip the leading "0" and decimal point from parameter estimates
   (default = TRUE)
...  
   Additional (optional) parameters

Value

- optionally return the dot code

References

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

See Also

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

Other Plotting functions: plot.MxLISRELModel, plot.MxModel, umxPlotACEcov, umxPlotACEv,
umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex,
umx

Examples

require(umx)
data(twinData)
   selDVs = "bmi"
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = "")
plot(m1, std = FALSE) # don't standardize
umxPlotACEcov  Make a graphical display of an ACE model with covariates.

Description

Make a graphical display of an ACE model with covariates.

Usage

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

Arguments

x \texttt{mxModel} to plot (created by umxACE in order to inherit the \texttt{MxModelACE} class)

file The name of the dot file to write: \texttt{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)
strip_zero Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
... Additional (optional) parameters

Value

- optionally return the dot code

References

- \url{https://tbates.github.io}

See Also

- \texttt{plot()}, \texttt{umxSummary()} work for IP, CP, GxE, SAT, and ACE models.
- \texttt{umxACE}

Other Plotting functions: \texttt{plotMxLISRELModel, plotMxModel, umxPlotACEv, umxPlotACE, umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex, umx}
Examples

```r
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("ht")
selVars = umx_paste_names(c(selDVs, selCovs), sep = "", suffixes = 1:2)
# Just top few pairs so example runs quickly
mzData = subset(twinData, zygosity == "MZFF", selVars)[1:100, ]
dzData = subset(twinData, zygosity == "DZFF", selVars)[1:100, ]
m = umxACEcov(selDVs = selDVs, selCovs = selCovs, dzData = dzData, mzData = mzData,
              sep = "", autoRun = TRUE)
plot(m)
plot(m, std = FALSE) # don't standardize
```

---

### umxPlotACEv

Produce a graphical display of an ACE variance-components twin model

#### Description

Plots an ACE model graphically, opening the result in the browser (or a graphviz application).

#### Usage

```r
umxPlotACEv(x = NA, file = "name", digits = 2, means = FALSE,
std = TRUE, strip_zero = TRUE, ...)
```

#### Arguments

- `x`: umxACEv model to plot.
- `file`: The name of the dot file to write: Default ("name") = use the name of the model. NA = don’t plot.
- `digits`: How many decimals to include in path loadings (default = 2)
- `means`: Whether to show means paths (default = FALSE)
- `std`: Whether to standardize the model (default = FALSE)
- `strip_zero`: Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
- `...`: Additional (optional) parameters

#### Value

- optionally return the dot code
Draw and display a graphical figure of Common Pathway model

Options include digits (rounding), showing means or not, and which output format is desired.

Usage

```r
umxPlotCP(x = NA, file = "name", digits = 2, means = FALSE, 
std = TRUE, format = c("current", "graphviz", "DiagrammeR"), 
SEstyle = FALSE, strip_zero = TRUE, ...)
```

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)
- `format`: = c("current", "graphviz", "DiagrammeR")
umxPlotCP

SEstyle report "b (se)" instead of "b [lower, upper]" (Default)

strip_zero Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)

... Optional additional parameters

Value

- Optionally return the dot code

References

- https://tbates.github.io

See Also

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

- `umxCp`

Other Plotting functions: `plot.MxLISRELModel`, `plot.MxModel`, `umxPlotACEcov`, `umxPlotACEv`, `umxPlotACE`, `umxPlotCPold`, `umxPlotGxEbiv`, `umxPlotGxE`, `umxPlotIP`, `umxPlotSexLim`, `umxPlotSimplex`, `umx`

Other Twin Reporting Functions: `umxPlotCPold`, `umxReduceACE`, `umxReduceGxE`, `umxReduce`, `umxSummarizeTwinData`, `umx`

Examples

```r
## Not run:
require(umx)
umx_set_optimizer("SLSQP")
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
seIDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP("new", seIDVs = seIDVs, sep = "_T",
dzData = dzData, mzData = mzData, nFac = 3)
# m1 = mxTryHardOrdinal(m1)
umxPlotCP(m1)
plot(m1) # No need to remember a special name: plot works fine!

## End(Not run)
```
umxPlotCPold

Draw and display a graphical figure of Common Pathway model

Description

Options include digits (rounding), showing means or not, and which output format is desired.

Usage

umxPlotCPold(x = NA, file = "name", digits = 2, means = FALSE,
               std = TRUE, format = c("current", "graphviz", "DiagrammeR"),
               SEstyle = FALSE, strip_zero = TRUE, ...)

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)
format = c("current", "graphviz", "DiagrammeR")
SEstyle report "b (se)" instead of "b [lower, upper]" (Default)
strip_zero Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
...

Value

- Optionally return the dot code

References

- https://tbates.github.io

See Also

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

Other Plotting functions: plot.MxLISRELModel, plot.MxModel, umxPlotACEcov, umxPlotACEv,
umxPlotACE, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotIP, umxPlotSexLim, umxPlotSimplex,
umx

Other Twin Reporting Functions: umxPlotCP, umxReduceACE, umxReduceGxE, umxReduce, umxSummarizeTwinData,
umxPlotGxE

Examples

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

## End(Not run)
```

```r
umxPlotGxE

Plot the results of a GxE univariate test for moderation of ACE components.

Description

Plot GxE results (univariate environmental moderation of ACE components). Options include plotting the raw and standardized graphs separately, or in a combined panel. You can also set the label for the x axis (xlab), and choose the location of the legend.

Usage

```r
umxPlotGxE(x, xlab = NA, location = "topleft",
separateGraphs = FALSE, acergb = c("red", "green", "blue", "black"),
...)
```

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)
- `acergb` Colors to use for plot `c(a = "red", c = "green", e = "blue", tot = "black")`
- `...` Optional additional parameters

Value

-

References

- https://tbates.github.io

See Also

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

Other Plotting functions: `plot.MxLISRELModel`, `plot.MxModel`, `umxPlotACEcov`, `umxPlotACEv`, `umxPlotACE`, `umxPlotCPold`, `umxPlotCP`, `umxPlotGxEbiv`, `umxPlotIP`, `umxPlotSexLim`, `umxPlotSimplex`, `umx`
Examples

```r
require(umx)

data(twinData)
twinData$sage1 = twinData$sage2 = twinData$sage
selDV$s = "bmi"

selDefs = "age"
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")

mQ = umxGxE(selDV$s = selDV$s, selDefs = selDefs,
            dzData = dzData, mzData = mzData, sep = ",", dropMissing = TRUE)

plot(mQ)

umxPlotGxE(x = mQ, xlab = "SES", separateGraphs = TRUE, location = "topleft")
```

Description

Plot GxE results (univariate environmental moderation of ACE components). Options include plotting the raw and standardized graphs separately, or in a combined panel. You can also set the label for the x axis (xlab), and choose the location of the legend.

Usage

```r
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

- https://tbates.github.io`
umxPlotIP

Draw a graphical figure for a Independent Pathway model

Description

Options include digits (rounding), showing means or not, standardization, and which output format is desired.

Usage

umxPlotIP(x = NA, file = "name", digits = 2, means = FALSE, std = TRUE, format = c("current", "graphviz", "DiagrammeR"), SEstyle = FALSE, strip_zero = TRUE, ...)

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)
umxPlotSexLim

std: whether to standardize the model (defaults to TRUE)
format: = c("current", "graphviz", "DiagrammeR")
SEstyle: report "b (se)" instead of "b [lower, upper]" (Default)
strip_zero: Whether to strip the leading "0" and decimal point from parameter estimates
(default = TRUE)
...

Value
- optionally return the dot code

References
- https://tbates.github.io

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

Other Plotting functions: plot.MxLISRELModel, plot.MxModel, umxPlotACEcov, umxPlotACEv,
umxPlotACE, umxPlotCPold, umxPlotCP, umxPlotGxEbiv, umxPlotGxE, umxPlotSexLim, umxPlotSimplex, umx

Examples

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

## End(Not run)
```

---

umxPlotSexLim

**Draw and display a graphical figure of a Sex limitation model**

**Description**

Options include digits (rounding), showing means or not, and which output format is desired.

**Usage**

```r
umxPlotSexLim(x = NA, file = "name", digits = 2, means = FALSE,
std = TRUE, format = c("current", "graphviz", "DiagrammeR"),
SEstyle = FALSE, strip_zero = TRUE, ...)
```
Arguments

- `x`: `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)
- `format`: = c("current", "graphviz", "DiagrammeR")
- `SEstyle`: report "b (se)" instead of "b [lower, upper]" (Default)
- `strip_zero`: Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
- `...`: Optional additional parameters

Details

# @aliases plot.MxModelCP

Value

- Optionally return the dot code

References

- [https://tbates.github.io](https://tbates.github.io)

See Also

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

Other Plotting functions: `plot.MxLISRELModel`, `plot.MxModel`, `umxPlotACEcov`, `umxPlotACEv`, `umxPlotACE`, `umxPlotCPold`, `umxPlotCP`, `umxPlotGxEbiv`, `umxPlotGxE`, `umxPlotIP`, `umxPlotSimplex`, `umx`

Examples

```r
## Not run:
require(umx)
umx_set_optimizer("SLSQP")
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

# Data for each of the 5 twin-type groups
```
umxPlotSimplex

Draw and display a graphical figure of a simplex model

Description

Options include digits (rounding), showing means or not, and which output format is desired.

Usage

umxPlotSimplex(x = NA, file = "name", digits = 2, means = FALSE,
std = TRUE, format = c("current", "graphviz", "DiagrammeR"),
strip_zero = TRUE, ...)

Arguments

x       The umxSimplex model 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)
format = c("current", "graphviz", "DiagrammeR")
strip_zero Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
...     Optional additional parameters
**Value**

- Optionally return the dot code

**See Also**

- `plot()`, `umxSummary()` work for IP, CP, GxE, SAT, simplex, ACEv, or ACE model.
- `umxSimplex`

Other Plotting functions: `plot.MxLISRELModel`, `plot.MxModel`, `umxPlotACEcov`, `umxPlotACEv`, `umxPlotACE`, `umxPlotCPold`, `umxPlotCP`, `umxPlotGxEbiv`, `umxPlotGxE`, `umxPlotIP`, `umxPlotSxLims`, `umx`

**Examples**

```r
## Not run:
# TODO Add example from umxSimplex help
data(iqdat)
mzData = subset(iqdat, zygosity == "MZ")
dzData = subset(iqdat, zygosity == "DZ")
seLDVs = c("IQ_age1", "IQ_age2", "IQ_age3", "IQ_age4")
ml = umxSimplex(seLDVs = seLDVs, sep = ".", dzData = dzData, mzData = mzData)
# plot(ml)

## End(Not run)
```

---

**umxRAM**

Easier path-based SEM modeling.

**Description**

`umxRAM` expedites creation of path-based models, still without doing invisible things to the model.

Here’s an example that makes a model of miles per gallon (mpg) as a function of weight (wt) and engine displacement (disp) using `mtcars` data.

```r
ml = umxRAM("tim", data = mtcars,
          umxPath(c("wt", "disp"), to = "mpg"),
          umxPath("wt", with = "disp"),
          umxPath(v.m. = c("wt", "disp", "mpg")))
```

As you can see, most of the work is done by `umxPath()`. `umxRAM` just wraps these paths up, takes the data as input, and then internally sets up all the labels and start values for the model, runs it, displays a summary, and a plot!

Try it, or one of the several models in the examples at the bottom of this page.

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).`
Because it uses the presence of a variable in the data to detect if a variable is latent or not, umxRAM needs data at build time.

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

```r
m1 = umxRAM("sketch", data = c("A", "B", "C"),
            umxPath("A", to = "B"),
            umxPath("B", with = "C"),
            umxPath(v.m. = c("A", "B", "C")))
```

Will create this figure:

.Usage:

```r
umxRAM(model = NA, ..., data = NULL, name = NA, comparison = TRUE,
        suffix = ",", showEstimates = c("none", "raw", "std", "both",
          "list of column names"), type = c("Auto", "FIML", "cov", "cor", "WLS",
          "DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"),
        autoRun =getOption("umx_auto_run"), tryHard = c("no", "yes",
          "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"),
        refModels = NULL, remove_unused_manifests = TRUE, independent = NA,
        setValues = TRUE, optimizer = NULL, verbose = FALSE)
```

**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)
suffix
String to append to each label (useful if model will be used in a multi-group model)

showEstimates
Whether to show estimates. Defaults to no (alternatives = "raw", "std", etc.)

type
One of "Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"

allContinuousMethod
"cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

autoRun
Whether to run the model, and return that (default), or just to create it and return without running.

tryHard
Default (’no’) uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

refModels
pass in reference models if available. Use FALSE to suppress computing these if not provided.

remove_unused_manifests
Whether to remove variables in the data to which no path makes reference (defaults to TRUE)

independent
Whether the model is independent (default = NA)

setValues
Whether to generate likely good start values (Defaults to TRUE)

optimizer
optionally set the optimizer (default NULL does nothing)

verbose
Whether to tell the user what latents and manifests were created etc. (Default = FALSE)

Details

Comparison with mxModel

umxRAM differs from mxModel in the following ways:

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 with data = (as elsewhere in R, e.g. lm).
5. You don’t need to add labels: paths are automatically labelled "a_to_b" etc.
6. You don’t need to set start values, they will be done for you.
7. You don’t need to mxRun the model: it will run automatically, and print a summary.
8. You don’t need to run summary: with autoRun, it will print a summary.
9. You get a plot of the model.
10. umxPath offers powerful verbs to describe paths.

Comparison with other software

Start values. Currently, manifest variable means are set to the observed means, residual variances are set to 80 of the observed variance of each variable, and single-headed paths are set to a positive starting value (currently .9). note: The start-value strategy is subject to improvement, and will be documented in the help for umxRAM.
Black-box software, defaults, and automatic addition of paths. Some SEM software does a lot of behind-the-scenes defaulting and path addition. I’ve explored 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 modeling. Learning the handful of umxPath short cuts allows modeling to stay both efficient and unambiguous!

Value

- mxModel

References


See Also

umxPath, umxSummary, plot, parameters, umxSuperModel

Other Core Modeling Functions: plot, MxLISRELModel, plot, MxModel, umxAlgebra, umxMatrix, umxModify, umxPath, umxRun, umxSuperModel, umx

Examples

```r
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# 1. Here's a simple example with raw data =
# 1. = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
ml = umxRAM("tim", data = mtcars,
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(v.m. = c("wt", "disp", "mpg"))
)

# 2. Use parameters to see the parameter estimates and labels
parameters(ml)

# And umxSummary to get standardized parameters, CIs etc from the run model.
umxSummary(ml, show = "std")
#   | name   | Std.Estimate | Std.SE | CI
# |--------|-------------|--------|-----|
# | wt_to_mpg | -0.54   | 0.17  | [-0.9, -0.2] |
# | disp_to_mpg | -0.36 | 0.18  | [-0.71, -0.02] |
# | mpg_with_mpg | 0.22  | 0.07  | [0.08, 0.35] |
# | wt_with_wt | 1.00 | 0.00 | [1, 1] |
# | b1 | 0.89 | 0.04 | [0.81, 0.96] |
# | disp_with_disp | 1.00 | 0.00 | [1, 1] |

# 3. '# Of course you can plot the model
plot(ml)
plot(ml, std=TRUE, means=FALSE)
```
plot(m1, std = TRUE, strip= FALSE, resid = "line")

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

# *note: The variance of displacement is in cubic inches and is very large.
# to help the optimizer, one might, say, multiply disp * .016 to work in litres
tmp = mtcars; tmp$disp = tmp$disp * .016

# We can just give the raw data and ask for it to be made into type cov:
ml = umxRAM("tim", data = tmp, type="cov",
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(var = c("mpg", "wt", "disp"))
)

# (see ?umxPath for more nifty options making paths...)

# ================================
# = umxRAM can also accept mxData as data =
# ================================
# For convenience, list up the manifests you will be using

selVars = c("mpg", "wt", "disp")
myCov = mxFData(cov(tmp[, selVars]), type = "cov", numObs = nrow(mtcars) )

ml = umxRAM("tim", data = myCov,
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(var = selVars)
)

# ================================
# = umxRAM supports WLS =
# ================================

# 1. Run an all-continuous WLS model
mw = umxRAM("raw", data = mtcars[, c("mpg", "wt", "disp")],
type = "WLS", allContinuousMethod = "cumulants",
umxPath(var = c("wt", "disp", "mpg")),
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(var = c("wt", "disp", "mpg"))
)

# 2. Switch to marginals to support means
mw = umxRAM("raw", data = mtcars[, c("mpg", "wt", "disp")],
type = "WLS", allContinuousMethod = "marginals",
umxPath(var = c("wt", "disp", "mpg")),
umxPath(c("wt", "disp"), to = "mpg"),
umxPath("wt", with = "disp"),
umxPath(var = c("wt", "disp", "mpg"))
umxRAM2

Make RAM model using lavaan syntax

Description

Can detect lavaan string input. TODO document once merged with 'umxRAM'.
umxRAM2Ordinal

Usage

umxRAM2Ordinal(model, data = NULL, lavaanMode = "sem", printTab = FALSE, name = NULL)

Arguments

model A lavaan string
data Data for the model (optional)
lavaanMode = "sem"
printTab Print the table (defaults to FALSE) # TODO just verbose
name Name for model (optional)

Value

- mxModel

See Also

- umxLav2RAM

Other Super-easy helpers: umxEFA, umxLav2RAM, umxTwoStage, umx

Examples

ml = umxRAM2("y - x")
ml = umxRAM2("y is x") # not a lavaan string
namedStr = " " # my name
y ~ x
ml = umxRAM2(namedStr)

# test for removal of bad chars from name
lav = " # Model 14 PROCESS Hayes + - ', ':, and '=' moderated mediation
gnt ~ a * cb
"
ml = umxRAM2(lav)

umxRAM2Ordinal umxRAM2Ordinal

Description

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

Usage

umxRAM2Ordinal(model, verbose = TRUE, name = NULL)
`umxReduce` Reduce models, and report the results.

**Arguments**

- `model` (An RAM model to add thresholds too.)
- `verbose` (Tell the user what was added and why (Default = TRUE).)
- `name` (A new name for the modified model. Default (NULL) = leave it as is).

**Value**

- `mxModel`

**See Also**

- `umxRAM`

**Other Advanced Model Building Functions**: `umxJiggle, umxLabel, umxLatent, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx_get_bracket_addresses, umx_string_to_algebra, umx`

**Examples**

```r
## Not run:
data(twinData)
# Cut to form category of 20% obese subjects
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)
ordDVs = c("obese1", "obese2")
twinData[, ordDVs] = umxFactor(twinData[, ordDVs])
mzData = twinData[twinData$zygosity %in% "MZFF",]
ml = umxRAM("tim", data = mzData,
umxPath("bmi1", with = "bmi2"),
umxPath(v.m. = c("bmi1", "bmi2"))
)
ml = umxRAM("tim", data = mzData,
umxPath("obese1", with = "obese2"),
umxPath(v.m. = c("obese1", "obese2"))
)
## End(Not run)
```
Description

Given a umx model (currently umxACE and umxGxE are supported - ask for more!) umxReduce will conduct a formalised reduction process.

GxE model reduction For umxGxE models, each form of moderation is tested on its own, and jointly. Also, C is removed, and moderation tested in this model.

ACE model reduction For umxACE models, A and then C are removed and tested.

It reports the results in a table. Set the format of the table with umx_set_table_format(), or set report to "html" to open a table for pasting into a word processor.

umxReduce is a work in progress, with more automatic reductions coming as demand emerges. I am thinking for RAM models to drop NS paths, and report that test.

Usage

umxReduce(model, report = c("markdown", "inline", "html", "report"), baseFileName = "tmp", ...)

Arguments

model The mxModel which will be reduced.
report How to report the results. "html" = open in browser
baseFileName (optional) custom filename for html output (defaults to "tmp")
... Other parameters to control model summary

References


See Also

umxReduceGxE, umxReduceACE

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx_xmu_get_CI

Other Twin Reporting Functions: umxPlotCPold, umxPlotCP, umxReduceACE, umxReduceGxE, umxSummarizeTwinData, umx
umxReduceACE

Reduce an ACE model.

Description

This function can perform model reduction on umxACE models, testing dropping A and C, as well as an ADE or ACE model, displaying the results in a table, and returning the best model.

Usage

umxReduceACE(model, report = c("markdown", "inline", "html", "report"),
               baseFileName = "tmp", intervals = TRUE, ...)

Arguments

- **model**: an ACE or ADE mxModel to reduce
- **report**: How to report the results. "html" = open in browser
- **baseFileName**: (optional) custom filename for html output (defaults to "tmp")
- **intervals**: Recompute CIs (if any included) on the best model (default = TRUE)
- **...**: Other parameters to control model summary

Details

It is designed for testing univariate models. You can offer up either the ACE or ADE base model.

Suggestions for more sophisticated automation welcomed!

Value

Best fitting model

References


See Also

umxReduceGxE, umxReduce

Other Twin Reporting Functions: umxPlotCPold, umxPlotCP, umxReduceGxE, umxReduce, umxSummarizeTwinData, umx
Examples

data(twinData)
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
ml = umxACE(selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")
m2 = umxReduce(ml)

Summary(ml)
ml = umxACE(selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "", dzCr = .25)
m2 = umxReduce(ml)

---

umxReduceGxE

Reduce a GxE model.

Description

This function can perform model reduction for umxGxE, testing dropping means-moderation, a, c & e, as well as c & c, a & a' etc.

It reports the results in a table. Set the format of the table with umx_set_table_format. Or set report to "html" to open a table for pasting into a word processor.

Usage

umxReduceGxE(model, report = c("markdown", "inline", "html", "report"),
baseFileName = "tmp_gxe", ...)

Arguments

model An mxModel to reduce.
report How to report the results. "html" = open in browser.
baseFileName (optional) custom filename for html output (defaults to "tmp").
... Other parameters to control model summary.

Value

best model

References


See Also

umxReduceACE, umxReduce

Other Twin Reporting Functions: umxPlotCPold, umxPlotCP, umxReduceACE, umxReduce, umxSummarizeTwinData, umx
Examples

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

**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. **Note** this is not needed for `umxRAM` models or twin models - it is just a convenience to get base `OpenMx` models to run.

**Usage**

```r
umxRun(model, n = 1, calc_SE = TRUE, calc_sat = TRUE,
setValues = FALSE, setLabels = FALSE, intervals = FALSE,
comparison = 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 (ignored when n = 1) for the summary (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 (default = FALSE)
- **setLabels**: Whether to set the labels (default = FALSE)
- **intervals**: Whether to run `mxCI` confidence intervals (default = FALSE) intervals = FALSE
- **comparison**: Whether to run `umxCompare()` after `umxRun`

**Value**

- `mxModel`

**References**

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)
umxSetParameters

See Also

Other Core Modeling Functions: plot.MxLISRELModel, plot.MxModel, umxAlgebra, umxMatrix, umxModify, umxPath, umxRAM, umxSuperModel, umx

Examples

```r
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 = latents, 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) # OpenMx's SE-based CIs
umxConfint(m1, run = TRUE) # get likelihood-based 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

```r
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

See Also
- umxModify, umxLabel

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

Examples
require(umx)
data(demoOneFactor)
lats   = c("G")
manifests = names(demoOneFactor)
m1 <- umxRAM("One Factor", data = mxData(demoOneFactor[,1:80], type = "raw"),
            umxPath(from = lats, to = manifests),
            umxPath(v.m. = manifests),
            umxPath(v1m0 = lats))
parameters(m1)
umxSetParameters(m1, regex = "\^\", newlabels = "m\_", test = TRUE)

m2 = umxSetParameters(m1, "G_to_x\_1", newlabels = "G_to_x\_2", test = FALSE)
parameters(m2)
Description

Multivariate twin analysis allowing for sex limitation (factors operate differently in males vs. females) based on a correlated factors model. With 5-groups of twins, this model allows for both Quantitative & Qualitative Sex-Limitation.

Quantitative differences refer to different amounts of phenotypic variance produced by the same A, C, or E components when operating in one sex compared to the other sex.

Qualitative differences refer to phenotypic variance attributable to an A, C, or E component which operates in one sex one but not in the other.

The correlation approach ensures that variable order does not affect the ability of the model to account for DZOS data.

Usage

```r
umxSexLim(name = "sexlim", selDVs, mzmData, dzmData, mzfData, dzfData, 
dzoData, sep = NA, A_or_C = c("A", "C"), sexlim = c("Nonscalar", 
"Scalar", "Homogeneity"), dzAr = 0.5, dzCr = 1, 
autoRun =getOption("umx_auto_run"), tryHard = c("no", "yes", 
"mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"), 
optimizer = NULL)
```

Arguments

- **name**: The name of the model (Default = "sexlim")
- **selDVs**: BASE NAMES of the variables in the analysis. You MUST provide sep.
- **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).
- **sep**: Suffix used for twin variable naming. Allows using just the base names in sel-Vars.
- **A_or_C**: Whether to model sex-limitation on A or on C. (Defaults to "A").
- **sexlim**: Which model type: "Nonscalar" (default), "Scalar", or "Homogeneity".
- **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).
- **autoRun**: Whether to mxRun the model (default TRUE: the estimated model will be returned).
- **optimizer**: optionally set the optimizer. Default (NULL) does nothing.
Details

A or C
Due to limitations on the degrees of freedom allowed by the twin model, we can model qualitative sex differences for only one of A or C at a time.

**1. Nonscalar Sex Limitation**
Allow quantitative (distinct male and female paths) and qualitative sex differences on A or C. Allows distinct between variable correlations ($Ra$, $Rc$ and $Re$) for males and for females. Male-Female correlations also free ($Rao$ or $Rco$ free in DZO group).

2. Scalar Sex Limitation
Quantitative sex differences only (distinct Male and female paths). Just one set of $Ra$, $Rc$ and $Re$ between variables (same for males and females)

3. Homogeneity
This is the model assumed by the basic ACE model: equal variance components in both sexes. Different means may be allowed for males and females.

notes: There is a half-way house model of heterogeneity in which $a$, $c$, and $e$ components are scaled by a scalar constant in one sex. # TODO sexlim: This k scalar heterogeneity model is not yet implemented in umx.

General restrictions: Assumes means and variances can be equated across birth order within zygosity groups.

Value

- mxModel of subclass mxModel.CFSexLim

References


See Also

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCovold, umxCV, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

Examples

```r
# = Load and Process Data =
# = ==Crowess==

## Not run:
require(umx)
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

# Data for each of the 5 twin-type groups
mzmData = subset(us_skinfold_data, zyg == 1)
mzfData = subset(us_skinfold_data, zyg == 2)
dzmData = subset(us_skinfold_data, zyg == 3)
dzfData = subset(us_skinfold_data, zyg == 4)
dzoData = subset(us_skinfold_data, zyg == 5)

umxSummarizeTwinData(us_skinfold_data, selVars="bic", zyg="zyg", sep="_T", MZFF=2, DZFF=4, MZMM=1, DZMM=3, DZOS=5)

# Run univariate example
# ----------------------------------------------

m1 = umxSexLim(selDVs = "bic", sep = "_T", A_or_C = "A", tryHard = "yes", mzmData = mzmData, dzmData = dzmData, mzfData = mzfData, dzfData = dzfData, dzoData = dzoData)

# Drop qualitative sex limitation
m1a = umxModify(m1, regex = "^Rao_", value=1, name = "no_qual", comparison = TRUE)

# Equate a, ac, and try ace across m & f in scalar model
m1b = umxModify(m1a, regex = "^a[fm]_", newlabels="a_", name = "eq_a_no_qual", comparison = TRUE)
m1c = umxModify(m1b, regex = "^c[fm]_", newlabels="c_", name = "eq_ac_no_qual", comparison = TRUE)
m1d = umxModify(m1c, regex = "^e[fm]_", newlabels="e_", name = "eq_ace_no_qual", comparison = TRUE)
umxCompare(m1, c(m1a, m1b, m1c, m1d))

# Scalar Sex Limitation
# ----------------------------------------------

m2 = umxSexLim(selDVs = "bic", sep = "_T", sexlim = "Scalar", tryHard = "yes", mzmData = mzmData, dzmData = dzmData, mzfData = mzfData, dzfData = dzfData, dzoData = dzoData)

# Show our manual drop of qualitative is the same as umxSexLim with sexlim= "scalar"
umxCompare(m1a, m2)

# ----------------------------------------------

m3 = umxSexLim(selDVs = "bic", sep = "_T", sexlim = "Homogeneity", tryHard = "yes",
mzmData = mzmData, dzmData = dzmData,  
mzfData = mzfData, dzfData = dzfData,  
dzoData = dzoData  
)  
umxCompare(m1, c(m2, m3))  

# Bivariate example with manual reduction =  
#  
m1 = umxSexLim(se1DV = c("bic", "tri"), sep = ".", A_or_C = "A", tryHard="yes",  
mzmData = mzmData, dzmData = dzmData,  
mzfData = mzfData, dzfData = dzfData,  
dzoData = dzoData  
)  

# Scalar sex limitation (same correlation among components for m and f)  
m2 = umxSexLim(se1DV = c("bic", "tri"), sep = ".",  
A_or_C = "A", tryHard="yes", sexlim="Scalar",  
mzmData = mzmData, dzmData = dzmData,  
mzfData = mzfData, dzfData = dzfData,  
dzoData = dzoData  
)  

# Drop qualitative sex limitation  
# i.e., same correlations for males and females.  
mla = umxModify(m1, regex = "Ra[mfo]_", newlabels="Ra_", name = "no_qual_a", comparison = TRUE)  
mlb = umxModify(mla, regex = "Rc[mfo]_", newlabels="Rc_", name = "no_qual_ac", comparison = TRUE)  
mlc = umxModify(mlb, regex = "Re[mfo]_", newlabels="Re_", name = "no_qual_ace", comparison = TRUE)  
umxCompare(m1, c(mla, mlb, mlc, m2))  

# In one smart regular expression  
m2 = umxModify(m1, regex = "R\[ace]\[fmo]\_", newlabels = "R\[1]",  
name = "scalar", comparison = TRUE)  

# Equate a, ac, and try ace across m & f in scalar model  
m2a = umxModify(m2, regex = "^a\[fm\]_", newlabels="a_", name = "eq_a_no_qual", comparison = TRUE)  
m2b = umxModify(m2a, regex = "^c\[fm\]_", newlabels="c_", name = "eq_ac_no_qual", comparison = TRUE)  
m2c = umxModify(m2b, regex = "^e\[fm\]_", newlabels="e_", name = "eq_ace_no_qual", comparison = TRUE)  
umxCompare(m1, c(mla, mlb, mlc, m2))  

# Run multi-variate example =  
#  
# Variables for Analysis  
se1DV = c('ssc','sil','caf','tri','bic')  
se1DV = c('ssc','tri','bic')  
m1 = umxSexLim(se1DV = se1DV, sep = ".", A_or_C = "A", tryHard = "mxTryHard",  
mzmData = mzmData, dzmData = dzmData,  
mzfData = mzfData, dzfData = dzfData, dzoData = dzoData  
)  

m2 = umxSexLim(se1DV = se1DV, sep = ".", A_or_C = "A", sexlim = "NonScalar",  
tryHard = "mxTryHard",  

Build and run a simplex twin model (not ready for use!)

Description

The simplex model provides a powerful tool for theory-based decomposition of genetic and environmental differences. `umxSimplex` makes a 2-group simplex twin model.

Usage

```r
umxSimplex(name = "simplex", selDVs, dzData, mzData, sep = NULL, 
  equateMeans = TRUE, dzAr = 0.5, dzCr = 1, addStd = TRUE, 
  addCI = TRUE, autoRun = getOption("umx_auto_run"), 
  tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", 
              "mxTryHardWideSearch"), optimizer = NULL)
```

Arguments

- **name**
  - The name of the model (defaults to "simplex")
- **selDVs**
  - The BASENAMES of the variables i.e., c(obese), not c(obese_T1, obese_T2)
- **dzData**
  - The DZ dataframe
- **mzData**
  - The MZ dataframe
- **sep**
  - The string preceding the final numeric twin identifier (often ".T") Combined with selDVs to form the full var names, i.e., just "dep" -> c("dep_T1", "dep_T2")
- **equateMeans**
  - Whether to equate the means across twins (defaults to TRUE).
- **dzAr**
  - The DZ genetic correlation (default = .5. Vary to examine assortative mating).
- **dzCr**
  - The DZ "C" correlation (defaults = 1. To make an ADE model, set = .25).
- **addStd**
  - Whether to add the algebras to compute a std model (default = TRUE).
- **addCI**
  - Whether to add the interval requests for CIs (default = TRUE).
- **autoRun**
  - Whether to run the model, and return that (default), or just to create it and return without running.
- **tryHard**
  - Default ("no") uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"
- **optimizer**
  - Optionally set the optimizer (default NULL does nothing).
Details

This code is beta quality: not for publication use. It will be completed by Boulder 2020.

The simplex model decomposes phenotypic variance into Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

In the simplex model, these influences are modeled as a combination of:

- Innovations at a given time (ai ci and ei matrices).
- Influences transmitted from previous time (at, ct, and et matrices).
- Influences specific to a single time (as, cs, es).

These combine to explain the causes of variance in the phenotype (see Figure).

Simplex path diagram:

Data Input Currently, the umxSimplex function accepts only raw data.

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 umxSimplex 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 modeling family-level effects shared 100 to .25 to model dominance effects.

Matrices and Labels in the simplex model A good way to see which matrices are used in umxSummary is to run an example model and plot it.

The loadings specific to each time point are contained on the diagonals of matrices as, cs, and es. So labels relevant to modifying these are of the form "as_r1c1", "as_r2c2" etc.
All the shared matrices are in the model "top". So to see the 'as' values, you can simply execute:
```r
m$top$as$values
```
The transmitted loadings are in matrices at, ct, et.
The innovations are in the matrix ai, ci, and ei.
Less commonly-modified matrices are the mean matrix expMean. This has 1 row, and the columns are laid out for each variable for twin 1, followed by each variable for twin 2.
Thus, in a model where the means for twin 1 and twin 2 had been equated (set = to T1), you could make them independent again with this script:
```r
m$top$expMean$labels[1,4:6] = c("expMean_r1c4", "expMean_r1c5", "expMean_r1c6")
```

### Value

- `mxModel`

### References

- [https://www.github.com/tbates/umx](https://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: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPOld`, `umxC`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSummaryACEcov`, `umxSummaryACEV`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

### Examples

```r
## Not run:
data(iqdat)
mzData = subset(iqdat, zygosity == "MZ")
dzData = subset(iqdat, zygosity == "DZ")
baseVars = c("IQ_age1", "IQ_age2", "IQ_age3", "IQ_age4")
m1 = umxSimplex(selDVs= baseVars, dzData= dzData, mzData= mzData, sep= "_T", tryHard= "mxTryHard")

umxSummary(m1)
parameters(m1, patt = "^s")
m2 = umxModify(m1, regex = "as_r1c1", name = "no_as", comp = TRUE)
umxCompare(m1, m2)

# -----------------------------
# = Test a 3 time-point model =
# -----------------------------
m1 = umxSimplex(selDVs = paste0("IQ_age", 1:3),
dzData = dzData, mzData = mzData, sep = "_T", tryHard = "mxTryHard")

## End(Not run)
```
umxSummarizeTwinData  Summarize twin data

Description

Produce a summary of wide-format twin data, showing the number of individuals, the mean and SD for each trait, and the correlation for each twin-type.

Set MZ and DZ to summarize the two-group case.

Usage

```r
umxSummarizeTwinData(data = NULL, selVars = "wt", sep = "_T",
zyg = "zygosity", MZ = NULL, DZ = NULL, MZFF = "MZFF",
DZFF = "DZFF", MZMM = "MZMM", DZMM = "DZMM", DZOS = "DZOS",
digits = 2)
```

Arguments

data  The twin data.

selVars Collection of variables to report on, e.g. c("wt", "ht").

sep The separator string that will turn a variable name into a twin variable name, e.g. "_T" for wt_T1 and wt_T2.

zyg The zygosity variable in the dataset, e.g. "zygosity".

MZ Set level in zyg corresponding to MZ for two group case (defaults to using 5-group case).

DZ Set level in zyg corresponding to DZ for two group case (defaults to using 5-group case).

MZFF The level in zyg corresponding to MZ FF pairs: e.g., "MZFF".

DZFF The level in zyg corresponding to DZ FF pairs: e.g., "DZFF".

MZMM The level in zyg corresponding to MZ MM pairs: e.g., "MZMM".

DZMM The level in zyg corresponding to DZ MM pairs: e.g., "DZMM".

DZOS The level in zyg corresponding to DZ OS pairs: e.g., "DZOS".

digits Rounding precision of the report (default 2).

Value

- formatted table, e.g. in markdown.

References

- https://github.com/tbates/umx
umxSummary

See Also

- umxAPA

Other Twin Reporting Functions: umxPlotCPold, umxPlotCP, umxReduceACE, umxReduceGxE, umxReduce, umx

Examples

data(twinData)

umxSummarizeTwinData(twinData, sep = "", selVars = c("wt", "ht"))

Mzs = c("MZMM", "MZFF"); DZs = c("DZFF", "DZMM", "DZOS")

umxSummarizeTwinData(twinData, sep = "", selVars = c("wt", "ht"), MZ = Mzs, DZ = DZs)

umxSummary

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: umxSummaryMxModel.
View documentation on the ACE model subclass here: umxSummaryACE.
View documentation on the ACEv model subclass here: umxSummaryACEv.
View documentation on the IP model subclass here: umxSummaryIP.
View documentation on the CP model subclass here: umxSummaryCP.
View documentation on the GxE model subclass here: umxSummaryGxE.

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, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx_xmu_get_CI
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

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

Arguments

- `model`: The `MxModel` whose fit will be reported
- `refModels`: Saturated models if needed for fit indices (see example below: If NULL will be computed 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 (alternative = "markdown")
- `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`.

CI's 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 (see `http://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

  https://tbates.github.io

See Also

- umxRun

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummaryACEv, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples

```r
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") # output as raw
umx_set_table_format("markdown")umxSummary(m1, show = "raw")
m1 <- mxModel(m1,
mxData(demoOneFactor[1:100], type = "raw"),
umxPath(mean = manifests),
```
umxPath(mean = latents, fixedAt = 0)
)
ml <- mxRun(ml)
umxSummary(ml, showEstimates = "std", filter = "NS")

umxSummaryACE  Shows a compact, publication-style, summary of a umx Cholesky ACE model

Description

Summarize a fitted Cholesky model returned by umxACE. Can control digits, report comparison
model fits, optionally show the Rg (genetic and environmental correlations), and show confidence
intervals, the report parameter allows drawing the tables to a web browser where they may readily
be copied into non-markdown programs like Word.

Usage

umxSummaryACE(model, digits = 2, file =getOption("umx_auto_plot"),
            comparison = NULL, std = TRUE, showRg = FALSE, CIs = TRUE,
            report = c("markdown", "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)
zeero.print  How to show zeros (".")
...  Other parameters to control model summary

Details

See documentation for other umx models here: umxSummary.
umxSummaryACEcov

Value
- optional mxModel

References

See Also
- umxACE, plot.MxModelACE, umxModify
Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexlim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexlim, umxSummarySimplex, umx, xmu_twin_check
Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxConfit, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary.MxModel, umxSummaryACEv, umxSummarySexlim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM

Examples
```r
require(umx)
data(twinData)
selDVs = c("bmi1", "bmi2")
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
umxSummary(m1)
```

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

## End(Not run)

---

```r
umxSummaryACEcov

Present results of a twin ACE-model with covariates in table and graphical forms.
```

Description
Summarize a Cholesky model with covariates, as returned by umxACEcov

Usage
```r
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"), ...)
```
umxSummaryACEcov

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

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx]

See Also

- `umxACEcov`

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPOld`, `umxCP`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Examples

```r
require(umx)
data(twinData)
selDV = c("bmi1", "bmi2")
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
m1 = umxACE(selDV = selDV, dzData = dzData, mzData = mzData)
m1 = umxRun(m1)
uxSummaryACE(m1)
## Not run:
uxSummaryACE(m1, file = NA);
uxSummaryACE(m1, file = "name", std = TRUE)
stdFit = umxSummaryACE(m1, returnStd = TRUE);
## End(Not run)
```
umxSummaryACEv  Shows a compact, publication-style, summary of a variance-based Cholesky ACE model.

Description
Summarize a fitted Cholesky model returned by umxACEv. Can control digits, report comparison model fits, optionally show the Rg (genetic and environmental correlations), and show confidence intervals. the report parameter allows drawing the tables to a web browser where they may readily be copied into non-markdown programs like Word.

Usage
umxSummaryACEv(model, digits = 2, file =getOption("umx_auto_plot"),
comparison = NULL, std = TRUE, showRg = FALSE, CIs = TRUE,
report = c("markdown", "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 = no plot.
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 (TRUE)
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

Details
See documentation for other umx models here: umxSummary.

Value
- optional mxModel

References
See Also

- `umxACEv`

Other Twin Modeling Functions: `umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCPOld, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx.xmu_twin_check

Other Reporting functions: `RMSEA_MxModel, RMSEA.summary.mxmodel, RMSEA, extractAIC_MxModel, loadings, residuals_MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary_MxModel, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok, umx_standardize_RAM`

Examples

```r
require(umx)
data(twinData)
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
m1 = umxACEv(se1DV = "bmi", sep = " ", dzData = dzData, mzData = mzData)

umxSummary(m1, std = FALSE)
## Not run:
umxSummary(m1, file = NA);

umxSummary(m1, file = "name", std = TRUE)

stdFit = umxSummary(m1, returnStd = TRUE)
## End(Not run)
```

---

**umxSummaryCP**

Present the results of a Common-pathway twin model in table and graphical form.

### Description

Summarizes a Common-Pathway model, as returned by `umxCP`

### Usage

```r
umxSummaryCP(model, digits = 2, std = TRUE, CIs = FALSE, showRg = FALSE, comparison = NULL, report = c("markdown", "html"), file =getOption("umx_auto_plot"), returnStd = FALSE, ...)
```

### Arguments

- **model**: A fitted `umxCP` model to summarize
- **digits**: Round to how many digits (default = 2)
- **std**: Whether to show the standardized model (TRUE) (ignored: used extended = TRUE to get unstandardized)
umxSummaryCP

CIs  Confidence intervals (default FALSE)
showRg  Whether to show the genetic correlations (default FALSE)
comparison  Run mxCompare on a comparison model (default NULL)
report  Print tables to the console (as 'markdown'), or open in browser ('html')
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)
...

Optional additional parameters

Value
- optional mxModel

References

See Also
- umxCP(), plot(), umxSummary() work for IP, CP, GxE, SAT, and ACE models.
Other Twin Modeling Functions: umxACE_c cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex, umxSummaryACEcov, umxSummaryACE, umxSummaryACE, umxSummaryGxEbiv, umxSummaryGxE, umxSummaryIP, umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

Examples
```r
## Not run:
require(umx)
umx_set_optimizer("SLSQP")
data(twinData)
twinData$wt1 = twinData$wt1/10
twinData$wt2 = twinData$wt2/10
selDVs = c("ht", "wt")
mzData <- subset(twinData, zygosity == "MZFF")
dzData <- subset(twinData, zygosity == "DZFF")
umx_set_auto_plot(FALSE) # turn off autoplotting for CRAN
m1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = ",", optimizer = "SLSQP")
umxSummaryCP(m1, file = NA) # Suppress plot creation with file
umxSummary(m1, file = NA) # Generic summary is the same
stdFit = umxSummaryCP(m1, digits = 2, std = TRUE, file = NA, returnStd = TRUE);
umxSummary(m1, std = FALSE, showRg = TRUE, file = NA);
umxSummary(m1, std = FALSE, file = NA)
# = Print example =
# = Print example =
umxSummary(m1, file = "Figure 3", std = TRUE)
# = Confint example =
```
# umxSummaryGxE

```r
m1 = umxConfint(m1, "smart", run = FALSE);
m2 = umxConfint(m1, "smart", run = TRUE);
umxSummary(m1, CIs = TRUE, file = NA);
```

`umxSummaryGxE` is a function used to summarize a Moderation model, as returned by `umxGxE`.

## Description

Summarize a Moderation model, as returned by `umxGxE`.

## 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`
umxSummaryGxEbiv

References


See Also

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

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE, umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxe, umxIPold, umxIP, umxSexlim, umxSimplex, umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryIP, umxSummarySexlim, umxSummarySimplex, umx, xmu_twin_check

Examples

```r
# 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)
twinData$age1 = twinData$age2 = twinData$age 
selDV = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDV, selDefs)
mzData = subset(twinData, zygosity == "MZFF", selVars)
dzData = subset(twinData, zygosity == "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]])]
# Not run:
m1 = umxGxE(selDV = selDV, selDefs = selDefs, dzData = dzData, mzData = mzData)
# Plot Moderation
umxSummaryGxE(m1)
umxSummaryGxE(m1, location = "topright")
umxSummaryGxE(m1, separateGraphs = FALSE)
```

umxSummaryGxEbiv

**Summarize a bivariate GxE twin model**

Description

umxSummaryGxEbiv summarizes a bivariate moderation model, as returned by umxGxEbiv.
Usage

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

Arguments

- `model`: A fitted `umxGxEbiv` model to summarize
- `digits`: round to how many digits (default = 2)
- `xlab`: label for the x-axis of plot
- `location`: default = "topleft"
- `separateGraphs`: Std and raw plots in separate graphs? (default = FALSE)
- `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)
- `comparison`: mxCompare model with comparison (default = NULL).
- `std`: Whether to show the standardized model (not implemented! TRUE)
- `reduce`: Whether to run and tabulate a complete model reduction...(Defaults to FALSE)
- `CIs`: Confidence intervals (FALSE)
- `report`: markdown or html (html opens in browser)
- ... Optional additional parameters

Value

- optional `mxModel`

References


See Also

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

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPOld`, `umxCp`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexlim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySexLim`, `umxSummarySimplex`, `umx`, `xmu_twin_check`
Examples

data(twinData)
selDVs = "wt"

selDefs = "ht"
df = umx_scale_wide_twin_data(twinData, varsToScale = c("ht", "wt"), sep = "")
mzData = subset(df, zygosity %in% c("MZFF", "MZMM"))
dzData = subset(df, zygosity %in% c("DZFF", "DZMM", "DZOS"))

## Not run
m1 = umxGxEbiv(selDVs = selDVs, selDefs = selDefs,
dzData = dzData, mzData = mzData, sep = "", dropMissingDef = TRUE)
# Plot Moderation
umxSummary(m1)

## End(Not run)

---

umxSummaryIP

**Present the results of an independent-pathway twin model in table and graphical form**

Description

Summarize a Independent Pathway model, as returned by umxIP

Usage

```
umxSummaryIP(model, digits = 2, file = getOption("umx_auto_plot"),
returnStd = FALSE, std = TRUE, showRg = FALSE, 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 = FALSE)
- **std**: Whether to show the standardized model (TRUE)
- **showRg**: whether to show the genetic correlations (FALSE)
- **comparison**: Whether to run mxCompare on a comparison model (NULL)
- **CIs**: Confidence intervals (F)
- **...**: Optional additional parameters
Value

- optional mxModel

References


See Also

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

Other Twin Modeling Functions:

  - umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE,
    umxCold, umxCP, umxGxE_window, umxGxEmbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex,
    umxSummaryACEcov, umxSummaryACEv, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE,
    umxSummarySexLim, umxSummarySimplex, umx, xmu_twin_check

Examples

```r
require(umx)
data(GFF, family function and well-being data)
mzData <- subset(GFF, zyg_2grp == "MZ")
dzData <- subset(GFF, zyg_2grp == "DZ")
se1DVs = c("hap", "sat", "AD") # These will be expanded into "hap_T1" "hap_T2" etc.
m1 = umxIP(se1DVs = se1DVs, sep = "_", dzData = dzData, mzData = mzData)

plot(m1)

# Not run:
uxSummaryIP(m1, digits = 2, file = "Figure3", showRg = FALSE, CIs = TRUE);

# End(Not run)
```

```
| umxSummarySexLim | Shows a compact, publication-style, summary of a umx Sex Limitation model |
```

Description

Summarize a fitted Cholesky model returned by umxSexLim. Can control digits, report comparison model fits, optionally show the Rg (genetic and environmental correlations), and show confidence intervals. The report parameter allows drawing the tables to a web browser where they may readily be copied into non-markdown programs like Word.

Usage

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

- model: a `umxSexLim` model 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

Details

See documentation for other umx models summary here: `umxSummary`.

Value

- optional `mxModel`

References

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

- `umxSexLim`

Other Twin Modeling Functions: `umxACE_cov_fixed`, `umxACEcov`, `umxACEold`, `umxACEv`, `umxACE`, `umxCPold`, `umxCP`, `umxGxE_window`, `umxGxEbiv`, `umxGxE`, `umxIPold`, `umxIP`, `umxSexLim`, `umxSimplex`, `umxSummaryACEcov`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummaryCP`, `umxSummaryGxEbiv`, `umxSummaryGxE`, `umxSummaryIP`, `umxSummarySimplex`, `umx`, `xmu_twin_check`

Other Reporting functions: RMSEA, `MxModel`, RMSEA, `summary.mxmodel`, RMSEA, `extractAIC`, `MxModel`, `loadings`, `residuals`, `MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEv`, `umxSummary`, `MxModel`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummarySimplex`, `umx_drop_ok`, `umx_standardize_RAM`
Examples

```r
# Not run:
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# = Beta: Should be good to use for Boulder/March 2020 =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# = Run Qualitative Sex Differences ACE model =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# = Load and Process Data =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
require(umx)
umx_set_optimizer("SLSQP")
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

# Variables for Analysis
selDVs = c('ssc', 'sil', 'caf', 'tri', 'bic')
# Data for each of the 5 twin-type groups
mzmData = subset(us_skinfold_data, zyg == 1)
mzfData = subset(us_skinfold_data, zyg == 2)
dzmData = subset(us_skinfold_data, zyg == 3)
dzfData = subset(us_skinfold_data, zyg == 4)
dzoData = subset(us_skinfold_data, zyg == 5)

# = Bivariate example =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

selDVs = c('tri', 'bic')
ml = umxSexLim(selDVs = selDVs, sep = ".T", A_or_C = "A", tryHard = "yes",
mzmData = mzmData, dzmData = dzmData,
mzfData = mzfData, dzfData = dzfData,
dzoData = dzoData)
umxSummary(ml, file = NA);

# = Switch to C =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
ml = umxSexLim(selDVs = selDVs, sep = ".T", A_or_C = "C", tryHard = "yes",
mzmData = mzmData, dzmData = dzmData,
mzfData = mzfData, dzfData = dzfData,
dzoData = dzoData)
```

umxSummarySimplex

Shows a compact, publication-style, summary of a Simplex model.

Description
Summarize a fitted Simplex model returned by umxSimplex. Can control digits, report comparison model fits, optionally show the Rg (genetic and environmental correlations), and show confidence intervals. The report parameter allows drawing the tables to a web browser where they may readily be copied into non-markdown programs like Word.

Usage
umxSummarySimplex(model, digits = 2, file = getOption("umx_auto_plot"),
comparison = NULL, std = TRUE, showRg = FALSE, CIs = TRUE,
report = c("markdown", "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 = no plot.
- **comparison**: you can run mxCompare on a comparison model (default = NULL)
- **std**: Whether to standardize the output (default = TRUE)
- **showRg**: (T/F) Whether to show the genetic correlations (default = FALSE)
- **CIs**: Whether to show Confidence intervals if they exist (default = TRUE)
- **report**: If "html", then open an html table of the results (default = "markdown")
- **returnStd**: Whether to return the standardized form of the model (default = FALSE)
- **extended**: how much to report (default = FALSE)
- **zero.print**: How to show zeros (default = ".")
- ... Other parameters to control model summary

Details
See documentation for other umx models here: umxSummary.

Value
- optional **mxModel**
umxSuperModel

Make a multi-group model

Description

umxSuperModel takes 1 or more models and wraps them in a supermodel with a mxFitFunctionMultigroup fit function that minimizes the sum of the fits of the sub-models.

Usage

umxSuperModel(name = "top", ..., autoRun = getOption("umx_auto_run"),
   tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal",
   "mxTryHardWideSearch"))

Examples

## Not run:
# 4 time model
# Select Data
data(iqdat)
mzData <- subset(iqdat, zygosity == "MZ")
dzData <- subset(iqdat, zygosity == "DZ")
vars = c("IQ_age1", "IQ_age2", "IQ_age3", "IQ_age4")
m1 = umxsimplex(se1DV= vars, sep= "_T", dzData= dzData, mzData= mzData, tryHard= "mxTryHard")
umxSummary(m1, file = NA);

## End(Not run)
Arguments

- **name**: The name for the container model (default = 'top')
- **...**
  - Models forming the multiple groups contained in the supermodel.
- **autoRun**: Whether to run the model, and return that (default), or just to create it and return without running.
- **tryHard**: Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"

Value

- **mxModel**

References

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

See Also

- **mxFitFunctionMultigroup**, **umxRAM**


Examples

```r
library(umx)
# Create two sets of data in which X & Y correlate ~ .4 in both datasets.
tmp <- umx_make_TwinData(nMZpairs = 100, nDZpairs = 150,
AA = 0, CC = .4, EE = .6, varNames = c("x", "y"))

# Group 1
d1 = tmp[[1]];
m1Data = mxData(cov(d1), type = "cov", numObs = nrow(d1), means = umx_means(d1))

# Group 2
d2 = tmp[[2]];
m2Data = mxData(cov(d2), type = "cov", numObs = nrow(d2), means = umx_means(d2))
cor(d1); cor(d2)

manifests = names(d1)

# Model 1
m1 <- umxRAM("m1", data = m1Data,
umxPath("x", to = "y", labels = "beta"),
umxPath(var = manifests, labels = c("Var_x", "Resid_y_grp1")),
umxPath(means = manifests, labels = c("Mean_x", "Mean_y"))
)

# Model 2
m2 <- umxRAM("m2", data = m2Data,
umxPath("x", to = "y", labels = "beta"),
umxPath(var = manifests, labels = c("Var_x", "Resid_y_grp2")),
```

umxThresholdMatrix

Create the threshold matrix needed for modeling ordinal data.

Description

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

Usage

umxThresholdMatrix(df, selDVs = NULL, sep = NULL, method = c("auto", "Mehta", "allFree"), threshMatName = "threshMat", l_u_bound = c(NA, NA), droplevels = FALSE, verbose = FALSE)

Arguments

df The data being modeled (to allow access to the factor levels and quantiles within these for each variable)

selDVs The variable names. Note for twin data, just the base names, which sep will be used to fill out.

sep (e.g. ".T") Required for wide (twin) data. It is used to break the base names out from their numeric suffixes.

method How to implement the thresholds: auto (the default), Mehta, which fixes the first two (auto chooses this for ordinal) or "allFree" (auto chooses this for binary)

threshMatName name of the matrix which is returned. Defaults to "threshMat" - best not to change it.
1_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)

verbose: How much to say about what was done. (defaults to FALSE)

Details

We often need to model ordinal data: sex, low-med-hi, depressed/normal, etc., A useful conceptual strategy to handle these data is to build a standard model for normally-varying data 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) reach the criteria for the diagnosis.

Making this work can require fixing the first 2 thresholds of ordinal data, or fixing both the mean and variance of a latent variable driving binary data, in order to estimate its one-free parameter: where to place the single threshold separating low from high cases.

Twin Data: For twins (the function currently handles only pairs), the thresholds are equated for both twins using labels: $labels

   obese1    obese2

   dev_1  "obese_dev1"  "obese_dev1"

The function returns a 3-item list consisting of:

1. A thresholdsAlgebra (named threshMatName)
2. A matrix of deviations for the thresholds (deviations_for_thresh)
3. A lower matrix of 1s (lowerOnes_for_thresh)

Value

- list of thresholds matrix, deviations, lowerOnes

References


See Also

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

Examples

```r
# ------------------------------------------------
# = Simple non-twin examples =
# ------------------------------------------------
# One ordered factor with 2-levels
x = data.frame(ordered(rbinom(100,1,.5))); names(x) <- c("x")
```
tmp = umxThresholdMatrix(x, selDV = "x")
# The lower ones matrix (all fixed)
tmp[[1]]$values

# The deviations matrix
tmp[[2]]$values

tmp[[2]]$labels # note labels are equated across twins

# The algebra that assembles these into thresholds:
tmp[[3]]$formula

# Example of a warning to not omit the variable names
# tmp = umxThresholdMatrix(x)
# Just a polite message, but for coding safety, I recommend calling
# umxThresholdMatrix with the names of the variables in the model.
# Next time, please include selDVs (AND you MUST include sep if this is a twin model!!)

# One ordered factor with 5-levels
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, selDV = "x")

# Example 1
# Cut to form category of 20 % obese subjects
obesityLevels = c('normal', 'obese')
cutPoints <- quantile(twinData[, "bmi"], 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 umxFactors (ordered, with the levels found in the data)
selVars = c("obese1", "obese2")
twinData[, selVars] <- umxFactor(twinData[, selVars])

# use verbose = TRUE to see informative messages
tmp = umxThresholdMatrix(twinData, selDV = selVars, sep = "", verbose = TRUE)

# Ordinal (n categories > 2) example =
# Repeat for three-level weight variable
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints = quantile(twinData[, "bmi"], 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)
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**: The structural equation to be estimated; a regression constant is implied if not explicitly omitted.
- **instruments**: A 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 "tls")

... arguments to be passed down.

Details

The example is a Mendelian Randomization analysis to show the utility of two-stage regression.

Value

-

References


See Also

- umx_make_MR_data, tsls, umxRAM

Other Super-easy helpers: umxEFA, umxLav2RAM, umxRAM2, umx

Examples

library(umx)

# = Mendelian randomization analysis =

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

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

# = Now with sem::tst =
# library(sem) # will require you to install X11
m2 = sem::tst(formula = Y ~ X, instruments = ~ qtl, data = df)
  coef(m1)
  coef(m2)
m3 = tst(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 = NULL)

**Arguments**

- **from**: A variable in the model for which you want to compute the effect of a change.
- **delta**: The amount to simulate changing ‘from’ by.
- **to**: The dependent variable that you want to watch changing.
- **model**: The model containing variables from and to.

**References**

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

**See Also**

- `mxCheckIdentification`, `mxCompare`

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

Examples

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

## End(Not run)
```

---

**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 `mxModel`s. 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.

**Usage**

```r
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`)

**Details**

*note*: If you give `umxValues` a numeric input, it will use `obj` as the mean, and return a list of length `n`, with `sd = sd`.

**Value**

- `mxModel` with updated start values

**References**

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

**See Also**

- Core functions:
- Other Advanced Model Building Functions: `umxJiggle`, `umxLabel`, `umxLatent`, `umxRAM2Ordinal`, `umxThresholdMatrix`, `umx_fix_first_loadings`, `umx_fix_latents`, `umx_get_bracket_addresses`, `umx_string_to_algebra`, `umx`
umxVersion

Get or 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! This function can also test for a minimum version.

Usage

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

Arguments

- **model**
  - Optional to show optimizer in this model
- **min**
  - Optional minimum version string to test for, e.g. ’2.7.0’ (Default = NULL).
- **verbose**
  - = TRUE
- **return**
  - Which package (umx or OpenMx) to ’return’ version info for (Default = umx).

Value

- **mxModel**

Examples

```
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
# Make an base OpenMx model (which will lack start values and labels..) =
ml = 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, ml)  # default variances are 0
# Add start values to the model
m1 = umxValues(ml)
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
```
umxWeightedAIC

References

See Also
- packageVersion, install.OpenMx

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umx_array_shift,
    umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page,
    umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

Examples
x = umxVersion(); x

| umxWeightedAIC | AIC weight-based conditional probabilities. |

Description
Returns the best model by AIC, and computes the probabilities according to AIC weight-based conditional probabilities (Wagenmakers & Farrell, 2004).

Usage
umxWeightedAIC(models, digits = 2)

Arguments
models a list of models to compare.
digits (default 2)

Value
- Best model

References

See Also
- AIC

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval,
    umxFactorScores, umxGetParameters, umxReduce, umxSummary, umx_APA_pval, umx_aggregate,
    umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI
Examples

```r
l1 = lm(mpg ~ wt + disp, data=mtcars)
l2 = lm(mpg ~ wt, data=mtcars)
umxWeightedAIC(models = list(l1, l2))
```

---

**Description**

A common task is preparing summary tables, aggregating over some grouping factor. Like mean and sd of age, by sex. R’s `aggregate` function is useful and powerful, allowing xtabs based on a formula.

`umx_aggregate` makes using it a bit easier. In particular, it has some common functions for summarizing data built-in, like "mean (sd)" (the default).

```r
umx_aggregate(mpg ~ cyl, data = mtcars, what = "mean_sd")
```

<table>
<thead>
<tr>
<th>cyl</th>
<th>mpg</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (n = 11)</td>
<td>26.66 (4.51)</td>
</tr>
<tr>
<td>6 (n = 7)</td>
<td>19.74 (1.45)</td>
</tr>
<tr>
<td>8 (n = 14)</td>
<td>15.1 (2.56)</td>
</tr>
</tbody>
</table>

**Usage**

```r
umx_aggregate(formula = DV ~ condition, data = NA,
what = c("mean_sd", "n"), digits = 2, report = c("markdown",
"html", "txt"))
```

**Arguments**

- `formula` The aggregation formula. e.g., DV ~ condition.
- `data` frame to aggregate.
- `what` function to use. Default reports "mean (sd)".
- `digits` to round results to.
- `report` Format for the table: Default is markdown.

**Value**

- `table`

**References**

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)
See Also
- umx_apply, aggregate

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAP, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx_xmu_get_CI

Examples

# = Basic use, compare with aggregate =
aggregate(mpg ~ cyl, FUN = mean, na.rm = TRUE, data = mtcars)
umx_aggregate(mpg ~ cyl, data = mtcars)

# = Use different (or user-defined) functions =
umx_aggregate(mpg ~ cyl, data = mtcars, what = "n")
numx_aggregate(mpg ~ cyl, data = mtcars, what = function(x){sum(!is.na(x))})

# turn off markdown
umx_aggregate(mpg ~ cyl, data = mtcars, report = "txt")

# = More than one item on the left hand side =
umx_aggregate(cbind(mpg, qsec) ~ cyl, data = mtcars, digits = 3)
# Transpose table
t(umx_aggregate(cbind(mpg, qsec) ~ cyl, data = mtcars))

## Not run:
umx_aggregate(cbind(moodAvg, mood) ~ condition, data = study1)

## End(Not run)

---

umx_APA_pval

**Round p-values according to APA guidelines**

Description

‘umx_APA_pval’ formats p-values, rounded in APA style. So you get ‘<.001’ instead of .000000002 or 1.00E-09.

You probably would be better off using [umxAP](umxAP), which this, but handles many more object types.

You set the precision with digits. Optionally, you can add ‘=’ ‘<’ etc. The default for addComparison (NA) adds these when needed.
## umx_apply

**Usage**

`umx.Apply(p, min = 0.001, digits = 3, addComparison = NA)`

**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)

**Value**

- p-value formatted in APA style

**See Also**

- `umx APA`, `round`

Other Reporting Functions: `loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI`

**Examples**

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

## Description

Tries to make apply more readable. so "mean of x by columns", instead of "of x, by 2, mean" Other functions to think of include: `cumsum, rowSums, colMeans`, etc.

**Usage**

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

**Arguments**

- **FUN**: The function to apply.
- **of**: The dataframe to work with.
- **by**: Apply the function to columns or to rows (default = "columns")
- **...**: optional arguments to FUN, e.g., na.rm = TRUE.
umx_array_shift

Value
- object

References

See Also
- umx_aggregate

Other Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx.cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_round, umx_var, 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_array_shift

Like the php array_shift function: shifts an item off the beginning of a list

Description
Returns x[1]. Has the SIDE EFFECT of assigning x to x[2:end] in the container environment.

Usage
umx_array_shift(x)

Arguments
x the vector to shift

Value
- first item of x

See Also
Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

Examples
x = c("Alice", "Bob", "Carol")
umx_array_shift(x) # returns "Alice"
x # now only 2 items (altered in containing environment)
umx_as_numeric

Description
Convert each column of a dataframe to numeric

Usage
umx_as_numeric(df, which = NULL, force = FALSE)

Arguments
- df: A data.frame to convert
- which: which columns to convert (default (null) selects all)
- force: Whether to force conversion to numeric for non-numeric columns (defaults to FALSE)

Value
- data.frame

References
- https://www.github.com/tbates/umx

See Also
Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx

Examples
```r
def = mtcars
# make mpg into string
def$mpg = as.character(df$mpg)
def$cyl = factor(df$cyl)
def = umx_as_numeric(df); str(df)
def = umx_as_numeric(df, force=TRUE); str(df)
# Make one variable alpha
def$mpg = c(letters,letters[1:6]); str(df)
def = umx_as_numeric(df, force=TRUE); str(df)
```
umx_cell_is_on  
Return whether a cell is in a set location of a matrix

Description
Helper to determine if a cell is in a set location of a matrix or not. Left is useful for, e.g. twin means matrices.

Usage
umx_cell_is_on(r, c, where = c("diag", "lower", "lower_inc", "upper", "upper_inc", "any", "left"), mat = NULL)

Arguments
- r  which row the cell is on.
- c  which column the cell is in.
- where  the location (any, diag, lower or upper (or _inc) or left).
- mat  (optionally) provide matrix to check dimensions against r and c.

Value
- mxModel

References

See Also
- umxLabel

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cont_2.quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

Examples

umx_cell_is_on(r = 3, c = 3, "lower")
umx_cell_is_on(r = 3, c = 3, "lower_inc")
umx_cell_is_on(r = 3, c = 3, "upper")
umx_cell_is_on(r = 3, c = 3, "upper_inc")
umx_cell_is_on(r = 3, c = 3, "diag")
umx_cell_is_on(r = 2, c = 3, "diag")
umx_cell_is_on(r = 3, c = 3, "any")

a_cp = umxMatrix("a_cp", "Lower", 3, 3, free = TRUE, values = 1:6)
umx_cell_is_on(r = 3, c = 3, "left", mat = a_cp)
## Not run:
Check that a test evaluates to TRUE. If not, stop, warn, or message the user.

Usage

```r
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

- [https://www.github.com/tbates/umx](https://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

```r
umx_check(length(1:3)==3, "stop", "item must have length == 3")
```
umx_check_model \hspace{2em} \textit{Check for required features in an OpenMx.}

\textbf{Description}

Allows the user to straight-forwardly require a specific model type (i.e., "RAM", "LISREL", etc.), whether or not the model has data, if it has been run or not. You can also test whether is has a means model or not and (in future) test if it has submodels.

\textbf{Usage}

\begin{verbatim}
umx_check_model(obj, type = NULL, hasData = NULL, beenRun = NULL,
hasMeans = NULL, checkSubmodels = FALSE, callingFn = "a function")
\end{verbatim}

\textbf{Arguments}

- \textbf{obj} \hspace{2em} an object to check
- \textbf{type} \hspace{2em} what type the model must be, i.e., "RAM", "LISREL", etc. (defaults to not checking NULL)
- \textbf{hasData} \hspace{2em} whether the model should have data or not (defaults to not checking NULL)
- \textbf{beenRun} \hspace{2em} whether the model has been run or not (defaults to not checking NULL)
- \textbf{hasMeans} \hspace{2em} whether the model should have a means model or not (defaults to not checking NULL)
- \textbf{checkSubmodels} \hspace{2em} whether to check submodels (not implemented yet) (default = FALSE)
- \textbf{callingFn} \hspace{2em} = Name of the calling function to help the user locate the error.

\textbf{Value}

- boolean

\textbf{References}

- \url{https://www.github.com/tbates/umx}

\textbf{See Also}

Other Test: \texttt{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}
umx_check_names

Examples

```r
require(umx)
data(demoOneFactor)
latents = c("g")
manifests = names(demoOneFactor)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
)

umx_check_model(ml) # TRUE, this is a model
umx_check_model(ml, type = "RAM") # equivalent to umx_is_RAM()
umx_check_model(ml, hasData = TRUE)
## Not run:
umx_check_model(ml, hasMeans = TRUE)
umx_check_model(ml, 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

```r
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 names-Needed (defaults to FALSE)
- `intersection`: Show the intersection of names
- `message`: Some helpful text to append when dieing.

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)
### umx_check_Os

**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 Check or test: `umx_is_class`, `umx_is_numeric`, `umx`, `xmu_twin_check`

### Examples

```r
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("z1", "x2"), data = demoOneFactor, die = FALSE, no_others = TRUE)

umx_check_names(c("x1","x2"), data = demoOneFactor, die = FALSE, no_others = TRUE)

## Not run:

umx_check_names(c("bad_var_name", "x2"), data = demoOneFactor, die = TRUE)

## End(Not run)
```

### 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

```r
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://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)
umx_check_parallel

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_parallel()

  umx_check_parallel

  Check if OpenMx is using OpenMP, test cores, and get timings

Description

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

Usage

  umx_check_parallel(nCores = c(1, parallel::detectCores() / 2),
                      testScript = NULL, rowwiseParallel = TRUE, nSubjects = 1000)

Arguments

  nCores           How many cores to run (defaults to c(1, max/2). -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.

Details

  Some historical (starting 2017-09-06) speeds on my late 2015 iMac, 3.3 GHz Quad-core i7 desktop.

<table>
<thead>
<tr>
<th>date</th>
<th>type</th>
<th>Cores</th>
<th>Time</th>
</tr>
</thead>
</table>
| 2018-10-14 | v2.11.5 (OpenMP on CRAN)    | 1 core | 01 min, 16 sec | (NPSOL)  
| 2018-10-14 | v2.11.5 (OpenMP on CRAN)    | 4 cores| 00 min, 36 sec | ∆: -39.598)  
| 2018-09-17 | v2.11.3                     | 1      | 01 min, 31 sec |  
| 2018-09-17 | v2.11.3                     | 4      | 00 min, 30.6 sec | ∆: -61.49)  
| 2017-10-16 | v2.7.18-9                   | 1      | 01 min, 07.30 sec |  
| 2017-10-16 | v2.7.18-9                   | 4      | 00 min, 22.63 sec | ∆: -44.68)  
| 2017-10-16 | Clang OpenMP                | 1      | 01 min, 08.38 sec |  
| 2017-10-16 | Clang OpenMP                | 4      | 00 min, 24.89 sec | ∆: -43.49)  
| 2017-09-07 | Clang OpenMP                | 1      | 01 min, 12.90 sec |  
| 2017-09-07 | Clang notOpenMP             | 1      | 01 min, 09.90 sec |  
| 2017-09-07 | TRAVIS                      | 1      | 01 min, 06.20 sec |  
| 2017-09-07 | TRAVIS                      | 4      | 00 min, 21.10 sec | ∆: -45.00)  


**umx_cont_2_quantiles**

**Value**
- NULL

**References**

**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**
```r
## 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 labeled "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
umx_cont_2_quantiles

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 "quantile 4".

Value

- recoded variable as an mxFactor

References


See Also

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, mxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

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 =
# ===============

data(twinData)
x = twinData
cuts = umx_cont_2_quantiles(rbind(x$wt1, x$wt2), 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 levels than integers exist
x = umx_cont_2_quantiles(rbinom(10000, 1, .5), nlevels = 2)
Report correlations and their p-values

Description
For reporting correlations and their p-values in a compact table. Handles rounding, and skipping non-numeric columns.

Usage
```r
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

Details
To compute heterochoric correlations, see `umxHetCor`.

note: The Hmisc package has a more robust function called rcorr

Value
- a matrix of correlations and p-values

References
- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also
- `umxHetCor`

Other Miscellaneous Stats Helpers: `reliability, umxCov2cor, umxHetCor, umx_apply, umx_fun_mean_sd, umx_means, umx_r_test, umx_round, umx_var, umx`

Examples
```r
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


See Also

- cov2cor

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_lower2full, umx_make_MRdata, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriwise, umx_read_lower, umx_rename, umx_reorder, umx_stack, 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, 0.8840448,
        0.9928430, 0.3371594, 0.7482155, 1.6142005, 1.9223567, 0.8777786,
        0.5320662, 0.4476898, 0.9013952, 0.8840448, 0.8777786, 1.3997558))
myData = umx_cov2raw(covData, n = 100, means = 1:6)
umx_default_option

Select first item in list of options, while being flexible about choices.

Description

Like a smart version of \texttt{match.arg}: Handles selecting parameter options when default is a list. Unlike \texttt{x = match.arg(x)} this allows items not in the list.

Usage

\texttt{umx_default_option(x, option_list, check = TRUE)}

Arguments

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

Value

- one validated option

References

- \url{https://www.github.com/tbates/umx}

See Also

- \texttt{match.arg}

Other Get and set: \texttt{umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx}

Examples

\begin{verbatim}
option_list = c("default", "par.observed", "empirical")
umx_default_option("par.observed", option_list)

# An example of checking a bad item and stopping
## Not run:
umx_default_option("bad", option_list)

## End(Not run)
umx_default_option("allow me", option_list, check = FALSE)
umx_default_option(option_list, option_list)
option_list = c(NULL, "par.observed", "empirical")
\end{verbatim}
umx_dot_define_shapes  Helper to make the list of vars and their shapes for a graphviz string

Description

Helper to make a graphviz rank string is a function which.

Usage

umx_dot_define_shapes(latents, manifests, preOut = "")

Arguments

latents list of latent variables
manifests list of manifest variables
preOut "" by default.

Value

string

See Also

- umx_dot_rank

Other Graphviz: umx_dot_mat2dot, umx_dot_rank, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker

Examples

umx_dot_define_shapes(c("as1"), c("E", "N"))
Return dot code for paths in a matrix

Description

Return dot code for paths in a matrix is a function which walks the rows and cols of a matrix. At each free cell, it creates a dot-string specifying the relevant path, e.g.:

```r
ai1 -> var1 [label=".35"]
```

Its main use is to correctly generate paths (and their sources and sink objects) without depending on the label of the parameter.

It is highly customizable:

1. You can specify which cells to inspect, e.g. "lower".
2. You can choose how to interpret path direction, from = "cols".
3. You can choose the label for the from to ends of the path (by default, the matrix name is used).
4. Offer up a list of from and toLabel which will be indexed into for source and sink
5. You can set the number of arrows on a path (e.g. both).
6. If type is set, then sources and sinks added manifests and/or latents output (p)

Finally, you can pass in previous output and new paths will be concatenated to these.

Usage

```r
umx_dot_mat2dot(x, cells = c("diag", "lower", "lower_inc", "upper", "upper_inc", "any", "left"), from = c("rows", "cols"), fromLabel = NULL, toLabel = NULL, showFixed = FALSE, arrows = c("forward", "both", "back"), fromType = NULL, toType = NULL, digits = 2, model = NULL, SStyle = FALSE, p = list(str = ",", latents = c(), manifests = c()))
```

Arguments

- `x` a `umxMatrix` to make paths from.
- `cells` which cells to process: "any" (default), "diag", "lower", "upper", "upper_inc", "any", "left")
- `from` one of "rows", "columns"
- `fromLabel` = NULL. NULL = use matrix name (default). If one, if suffixed with index, length() > 1, index into list. "one" is special.
- `toLabel` = NULL. NULL = use matrix name (default). If one, if suffixed with index, length() > 1, index into list.
- `showFixed` = FALSE.
- `arrows` "forward" "both" or "back"
- `fromType` one of "latent" or "manifest" NULL (default) = don’t accumulate new names.
toType
	one of "latent" or "manifest" NULL (default) = don’t accumulate new names.
digits
	to round values to (default = 2).
model
	If you want to get CIs, you can pass in the model (default = NULL).
SEstyle
	If TRUE, CIs shown as "b(SE)" ("b [l,h]" if FALSE (default)). Ignored if model NULL.
p
	input to build on. list(str = "", latents = c(), manifests = c())

Value

• list(str = "", latents = c(), manifests = c())

See Also

• plot

Other Graphviz: umx_dot_define_shapes, umx_dot_rank, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker

Examples

# Make a lower 3 * 3 value= 1:6 (1, 4, 6 on the diag)
a_cp = umxMatrix("a_cp", "Lower", 3, 3, free = TRUE, values = 1:6)

# Get dot strings for lower triangle (default from and to based on row and column number)
out = umx_dot_mat2dot(a_cp, cells = "lower", from = "cols", arrows = "both")
cat(out$str) # a_cp1 -> a_cp2 [dir = both label="2"];

# one arrow (the default = "forward")
out = umx_dot_mat2dot(a_cp, cells = "lower", from = "cols")
cat(out$str) # a_cp1 -> a_cp2 [dir = forward label="2"];

# label to (rows) using var names
out = umx_dot_mat2dot(a_cp, toLabel= paste0("v", 1:3), cells = "lower", from = "cols")

# First call also inits the plot struct
out = umx_dot_mat2dot(a_cp, from = "rows", cells = "lower", arrows = "both", fromType = "latent")
out = umx_dot_mat2dot(a_cp, from = "rows", cells = "diag", toLabel= "common", toType = "manifest", p = out)

# ==============
# = Add found sinks to manifests =
# ==============
out = umx_dot_mat2dot(a_cp, from = "rows", cells = "diag", toLabel= c('a','b','c'), toType = "manifest");

# ==============
# = Add found sources to latents =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
out = umx_dot_mat2dot(a_cp, from= "rows", cells= "diag",
toLabel= c('a', 'b', 'c'), fromType= "latent";
  umx_msg(out$latents)

# = Get a string which includes CI information =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
data(demoOneFactor)
  ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
    umxPath(latents, to = manifests),
    umxPath(var = manifests),
    umxPath(var = latents, fixedAt = 1.0)
  )
  ml = umxCI(ml, run = "yes")
  out = umx_dot_mat2dot(ml$A, from = "cols", cells = "any",
    toLabel= paste0("x", 1:5), fromType = "latent", model= ml);
  umx_msg(out$str); umx_msg(out$latents)

# = Label a means matrix =
# = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

  tmp = umxMatrix("expMean", "Full", 1, 4, free = TRUE, values = 1:4)
  out = umx_dot_mat2dot(tmp, cells = "left", from = "rows",
    fromLabel= "one", toLabel= c("v1", "v2")
  )
  cat(out$str)

umx_dot_rank  Helper to make a graphviz rank string

Description

Given a list of names, this filters the list, and returns a graphviz string to force them into the given
rank. e.g. "rank=same; as1;"

Usage

  umx_dot_rank(vars, pattern, rank)

Arguments

  vars  a list of strings
  pattern  regular expression to filter vars
  rank  "same", "max", "min"
**Value**

string

**See Also**

- `umx_dot_define_shapes`

Other Graphviz: `umx_dot_define_shapes`, `umx_dot_mat2dot`, `xmu_dot_paths`, `xmu_dot_make_residuals`, `xmu_dot_maker`

**Examples**

```r
umx_dot_rank(c("as1"), "^[ace][0-9]+", "same")
```

**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**

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

**Arguments**

- `model1` the base code `MxModel`
- `model2` the nested code `MxModel`
- `text` name of the thing being tested, i.e., "Extraversion" or "variances"

**Value**

-

**References**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

Other Reporting functions: `RMSEA.MxModel`, `RMSEA.summary.mxmodel`, `RMSEA.extractAIC.MxModel`, `loadings.residuals.MxModel`, `umxCI_boot`, `umxCI`, `umxCompare`, `umxConfint`, `umxExpCov`, `umxExpMeans`, `umxFitIndices`, `umxPlotACEv`, `umxSummary.MxModel`, `umxSummaryACEv`, `umxSummaryACE`, `umxSummarySexLim`, `umxSummarySimplex`, `umx_standardize_RAM`
Examples

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

---

**umx_explode**

- *like the php function 'explode'*

**Description**

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

**Usage**

```r
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**


**See Also**

Other String Functions: `umx_explode_twin_names`, `umx_grep`, `umx_names`, `umx_object_as_str`, `umx_paste_names`, `umx_rot`, `umx_trim`, `umx_write_to_clipboard`, `umx`

**Examples**

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

*Break twin variable names (BMI_T1, BMI_T2) into base variable names (BMI, "_T", 1:2)*

---

**Description**

Break names like Dep_T1 into a list of base names, a separator, and a vector of twin indexes. e.g.: c("Dep_T1", "Dep_T2", "Anx_T1", "Anx_T2") will become:

```
list(baseNames = c("Dep", "Anx"), 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(baseNames, sep, twinIndexes)

**See Also**

Other String Functions: umx_explode, umx_grep, umx_names, umx_object_as_str, umx_paste_names, umx_rotate, umx_pad, umx_write_to_clipboard, umx

**Examples**

```r
require(umx)
data("twinData")
umx_explode_twin_names(twinData, sep = "")
umx_explode_twin_names(twinData, sep = NULL)

# Ignore this: just a 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")
```
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 Miscellaneous Utility Functions: `install.OpenMx`, `qm`, `umxBrownie`, `umxFactor`, `umxVersion`, `umx_array_shift`, `umx_cell_is_on`, `umx_cont_2_quantiles`, `umx_make`, `umx_msg`, `umx_open_CRAN_page`, `umx_pad`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`, `umx`, `xmu_check_variance`

Examples

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

## End(Not run)
```
Description

Fix the loading of the first path from each latent at selected value. Seldom used; might be useful to show students how to scale models with fixed latent or fixed first path... Note: latents with fixed variance are toggled by default (change made in 2019).

Usage

```r
umx_fix_first_loadings(model, latents = NULL, at = 1,
    freeFixedLatent = TRUE)
```

Arguments

- `model`: An `mxModel` to set.
- `latents`: Which latents to fix from (NULL = all).
- `at`: The value to fix the first path at (Default = 1).
- `freeFixedLatent`: Whether to free a latent if it is fixed (default = TRUE)

Value

- `mxModel`

References


See Also

Other Advanced Model Building Functions: `umxJiggle, umxLabel, umxLatent, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_latents, umx_get_bracket_addresses, umx_string_to_algebra, umx`

Examples

```r
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
    umxPath("g", to = manifests),
    umxPath(var = manifests),
    umxPath(var = "g", fixedAt = 1.0)
```
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. # TODO: umx_fix_latents is deprecated - likely of no use.

Usage

```r
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

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://openmx.ssri.psu.edu](https://openmx.ssri.psu.edu)

See Also

Other Advanced Model Building Functions: `umxJiggle`, `umxLabel`, `umxLatent`, `umxRAM2Ordinal`, `umxThresholdMatrix`, `umxValues`, `umx_fix_first_loadings`, `umx_get_bracket_addresses`, `umx_string_to_algebra`, `umx`
Examples

```r
require(umx)

data(demoOneFactor)

manifests = names(demoOneFactor)

ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
               umxPath(latents, to = manifests),
               umxPath(var = manifests),
               umxPath(var = latents, fixedAt = 1.0))


also_show(ml, what = "free", matrices = "S") # variance of g is not set

also_show(ml, what = "free", matrices = "S") # variance of g is fixed at 1
```

umx_fun_mean_sd

Description

Miscellaneous functions that are handy in summary and other tasks where you might otherwise have to craft a custom nameless functions. 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://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)
See Also

Other Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_means, umx_r_test, umx_round, umx_var, umx

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

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

Other Advanced Model Building Functions: umxJiggle, umxLabel, umxLatent, umxRAM2Ordinal, umxThresholdMatrix, umxValues, umx_fix_first_loadings, umx_fix_latents, umx_string_to_algebra, umx
Examples

require(umx)
data(demoOneFactor)
latents = c("g")
manifests = names(demoOneFactor)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
            umxPath(latents, to = manifests),
            umxPath(var = manifests),
            umxPath(var = latents, fixedAt = 1.0))

umx_get_bracket_addresses(ml$matrices$A, free= TRUE)

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

- https://tbates.github.io

See Also

Other Get and set: umx_default_option, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx
Examples

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

Description

Show the umx options. Useful for beginners to discover, or people like me to remember :-)

Usage

```r
umx_get_options()
```

Value

- message

See Also

Other Get and set: `umx_default_option`, `umx_get_checkpoint`, `umx_set_auto_plot`, `umx_set_auto_run`,
`umx_set_checkpoint`, `umx_set_condensed_slots`, `umx_set_cores`, `umx_set_data_variance_check`,
`umx_set_optimization_options`, `umx_set_optimizer`, `umx_set_plot_file_suffix`, `umx_set_plot_format`,
`umx_set_table_format`, `umx`

Examples

```r
umx_get_options()
```
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

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

- `grep` `umx_names` `umx_aggregate`

Other String Functions: `umx_explode_twin_names`, `umx_explode`, `umx_names`, `umx_object_as_str`, `umx_paste_names`, `umx_rot`, `umx_trim`, `umx_write_to_clipboard`
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
- https://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)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
umxPath(latents, to = manifests),

**umx_has_CIs**

```r
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
```

`umx_has_been_run(m)`

---

### 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

- [https://www.github.com/tbates/umx/](https://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

```r
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m = umxRAM("One Factor", data = demoOneFactor, type = "cov",
             umxPath("g", to = manifests),
             umxPath(var = manifests),
             umxPath(var = "g", fixedAt = 1.0))
```
umx_has_means(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
umxSummary(m1)

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
- https://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 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
umxPath(latents, to = manifests),
umxPath(var = manifests),
umxPath(var = latents, fixedAt = 1.0)
```
umx_has_square_brackets

Check if a label contains square brackets

Description

Helper function to check if a label has square brackets, e.g. "A[1,1]"

Usage

umx_has_square_brackets(input)

Arguments

input  The label to check for square brackets (string input)

Value

- boolean

References

- https://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_class

Check if variables in a dataframe are in a list of classes.

Description

Checks the class of each column in a dataframe, seeing if they are Returns a vector of TRUE and FALSE, or, if all ==TRUE, a single binary (the default).

Usage

umx_is_class(df, classes = NULL, all = TRUE)

Arguments

- df: A dataframe to check
- classes: vector of valid classes, e.g. numeric
- all: Whether to return a single all() Boolean or each column individually.

Value

- Boolean or Boolean vector

References


See Also

- `umx_is_numeric`

Other Check or test: `umx_check_names`, `umx_is_numeric`, `umx`, `xmu_twin_check`

Examples

```r
umx_is_class(mtcars) # report class list
# Are the variables in mtcars type character?
umx_is_class(mtcars, "character") # FALSE
# They're all numeric data
umx_is_class(mtcars, "numeric") # TRUE
# Show the test-result for each variable in mtcars
umx_is_class(mtcars, "numeric") # TRUE
# Are they _either_ a char OR a num?
umx_is_class(mtcars, c("character", "numeric"))
# Is zygosity a factor (note we don't drop = F to keep as dataframe)
umx_is_class(twinData[,"zygosity", drop=FALSE], classes = "factor")
```
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, then T | F)

References

- https://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_hasBeenRun, 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_cow(df)
df = cor(mtcars)
umx_is_cow(df)
umx_is_cow(df, boolean = TRUE)
umx_is_cow(mtcars, boolean = TRUE)
### umx_is_endogenous

| 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**

```r
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**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://openmx.ssri.psu.edu](https://openmx.ssri.psu.edu)

**See Also**

Other Tests: `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**

```r
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)
```
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


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

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

- https://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(mtcars, type = "raw")
umx_is_MxData(mxData(cov(mtcars), type = "cov", numObs = 73))
umx_is_MxData(mxDataWLS(na.omit(twinData[, c("wt1", "wt2")]), type = "WLS"))

umx_is_MxMatrix

Description

Utility function returning a binary answer to the question "Is this an OpenMx mxMatrix?"

Usage

umx_is_MxMatrix(obj)
umx_is_MxModel

Arguments

obj an object to be tested to see if it is an OpenMx mxMatrix

Value

- Boolean

References

- https://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

- https://www.github.com/tbates/umx
umx_is_numeric

Check if variables in a dataframe are numeric

Description

Checks across columns of a dataframe, return a vector of TRUE and FALSE, or, if all == TRUE, a single binary (the default).

Usage

`umx_is_numeric(df, all = TRUE)`

Arguments

- `df` A dataframe to check
- `all` Whether to return a single all() Boolean or each column individually.

Value

- Boolean or Boolean vector

References

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

See Also

- `umx_is_class`

Other Check or test: `umx_check_names, umx_is_class, umx, xmu_twin_check`

Examples

```r
umx_is_numeric(mtcars) # TRUE
umx_is_numeric(mtcars, all=FALSE) # vector of TRUE
```
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

- https://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)
numx_is_ordered(tmp, names = TRUE, binary.only = TRUE)
numx_is_ordered(tmp, names = TRUE, ordinal.only = TRUE)
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

- https://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)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
umxPath(latents, to = manifests),
umxPath(var = manifests),
umx_long2wide

    umxPath(var = latents, fixedAt = 1.0)
    )

    if(umx_is_RAM(ml)){
        message("nice RAM model!")
    }
    if(!umx_is_RAM(ml)){
        message("model needs to be a RAM model")
    }

umx_long2wide Take a long twin-data file and make it wide (one family per row)

Description

umx_long2wide merges on famID. Family members are ordered by twinID. twinID is equivalent to birth order. Up to 10 twinIDs are allowed (family order).

Note: Not all data sets have an order column, but it is essential to rank subjects correctly.

Note: The functions assumes that if zygosity or any passalong variables are NA in the first family member, they are NA everywhere. i.e., it does not hunt for values that are present elsewhere to try and self-heal missing data.

Usage

    umx_long2wide(data, famID = NA, twinID = NA, zygosity = NA, vars2keep = NA, passalong = NA, twinIDs2keep = NA)

Arguments

data The original (long-format) data file
famID The unique identifier for members of a family
twinID The twinID. Typically 1, 2, 50 51, etc...
zygosity Typically MZFF, DZFF MZMM, DZMM DZOS
vars2keep = The variables you wish to analyse (these will be renamed with paste0("_T", twinID)
passalong = Variables you wish to pass-through (keep, even though not twin vars)
twinIDs2keep = If NA (the default) all twinIDs are kept, else only those listed here. Useful to drop sibs.

Value

• dataframe in wide format

References

# umx_lower2full

Convert lower-only matrix data to full (or enforce symmetry on a full matrix)

## Description

Takes a vector of the lower-triangle of cells in a matrix as you might read-in from a journal article), OR a matrix (for instance from a "lower" `mxMatrix`, and returns a full matrix, copying the lower triangle into the upper.

## Usage

```r
umx_lower2full(lower.data, diag = NULL, byrow = TRUE, 
               dimnames = NULL)
```

## Examples

```r
# First we have to make a long format file to base the demo on =
# tmp = twinData[, -2]
tmp$twinID1 = 1
tmp$twinID2 = 2
long = umx_wide2long(data = tmp, sep = "")
#
# OK. Now to demo long2wide...
#
# Keeping all columns
wide = umx_long2wide(data= long, famID= "fam", twinID= "twinID", zygosity= "zygosity")
names(wide) # some vars, like part, should have been passed along instead of made into "part_T1"
#
# Just keep bmi and wt
wide = umx_long2wide(data= long, famID= "fam", twinID= "twinID", 
                      zygosity= "zygosity", vars2keep = c("bmi", "wt"))
names(wide)
#
# "fam" "twinID" "zygosity" "bmi_T1" "wt_T1" "bmi_T2" "wt_T2"
#
# Keep bmi and wt, and pass through 'cohort'
wide = umx_long2wide(data= long, famID= "fam", twinID= "twinID", zygosity= "zygosity", 
                      vars2keep = c("bmi", "wt"), passalong = "cohort")
```
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)

Details

*note*: Can also take lower data presented in the form of a data.frame. Note also, if presented with a full matrix, the function will return a matrix with symmetry enforced. Can be handy when you have a "nearly-symmetrical" matrix (with differences in the tenth decimal place).

Value

- `mxMatrix`

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

Other Data Functions: `umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx`

Examples

```r
# 1. Test with a vector in byrow = TRUE order
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, 1.0000)
x = umx_lower2full(tmp, diag = TRUE)
# check
isSymmetric(x)

# 2. Test with matrix input
tmpn = c("ROccAsp", "REdAsp", "F0ccAsp", "FEdAsp", "RParAsp", "RIQ", "RSes", "Fses", "FIQ", "FParAsp")
```
umx_make

"make" the umx package using devtools: release to CRAN etc.

Description

Easily run devtools "install", "release", "win", "examples" etc.
Usage

```r
umx_make(what = c("quick_install", "install_full", "spell", "run_examples", "check", "win", "rhub", "release"), pkg = "~/bin/umx", check = TRUE, run = FALSE, start = NULL, spelling = "en_US")
```

Arguments

- **what**: whether to "install", "release" to CRAN, check on "win", "check", "rhub", "spell" check, or check "examples")
- **pkg**: the local path to your package. Defaults to my path to umx.
- **check**: Whether to run check on the package before release (default = TRUE).
- **run**: = If what is "examples", whether to also run examples marked don’t run. (default FALSE)
- **start**: If what is "examples", which function to start from (default (NULL) = beginning).
- **spelling**: Whether to check spelling before release (default = "en_US": set NULL to not check).

Value

-

References

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

See Also

Other Miscellaneous Utility Functions: `install.OpenMx`, `qm`, `umxBrownie`, `umxFactor`, `umxVersion`, `umx_array_shift`, `umx_cell_is_on`, `umx_cont_2_quantiles`, `umx_find_object`, `umx_msg`, `umx_open_CRAN_page`, `umx_pad`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`, `umx`, `xmu_check_variance`

Examples

```r
## Not run:
umx_make(what = "q") # Quick install
umx_make(what = "install") # Just installs the package
umx_make(what = "examples") # Run the examples
umx_make(what = "spell") # Spell check the documents
umx_make(what = "check") # Run R CMD check
umx_make(what = "win") # Check on win-builder
umx_make(what = "release") # Release to CRAN

## End(Not run)
```
umx_make_bin_cont_pair_data

Make pairs of bin & continuous columns to represent censored 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


See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriwise, umx_read_lower, umx_rename, umx_reorder, umx_stack, 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]
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
```r
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 missing 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, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, 
umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, 
umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, 
umx_stack, umx_swap_a_block, umx

Examples
```r
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**

```r
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**

- umx_make_TwinData

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_TwinData, umx_make_bin_cont_pair_data, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, 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

Convert an excel spreadsheet in a text file on sql statements.

Description

Unlikely to be of use to anyone but the package author :-)

Usage

umx_make_sql_from_excel(thefile = "Finder")

Arguments

  theFile  The xlsx file to read. Default = "Finder")

Details

On OS X, the default (thefile= "Finder" will use the file selected in the front-most Finder window. If it is blank, a choose file dialog will be thrown.

Read an xlsx file and convert into SQL insert statements (placed on the clipboard) On MacOS, the function can access the current front-most Finder window.

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

  tabbedPlus: list scored from 0 to n-1
  tabbedVertPlus: tabbed, but vertical lay-out
  number 2+2|itemBreak>min='0' max='7' step='1'

  5fm Scored 1-5, anchored: Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree
  intro (not) scored, and sequenced as item 0

Value

-
umx_make_TwinData

Simulate twin data with control over A, C, and E parameters, as well as moderation of A.

Description

Makes MZ and DZ twin data, optionally with moderated A. By default, the three variance components must sum to 1.

See examples for how to use this: it is pretty flexible).

If you provide 2 varNames, they will be used for twin 1 and twin 2. If you provide one, it will be expanded to var_T1 and var_T2

You supply the number of pairs of each zygosity that wish to simulate (nMZpairs, nDZpairs), along with the values of AA, CC, and EE.

Note, if you want a power calculator, see here.

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 Univariate GxE Data 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.

Bivariate GxE Data

To simulate data with a moderator that is not shared by both twins. Moderated heritability is specified via the bivariate relationship (AA, CC, EE) and two moderators in each component. AA =

Examples

```r
## Not run:
# An example Excel spreadsheet
# local uncompiled path
fp = system.file("inst/extdata", "GQ6.sql.xlsx", package = "umx")
# installed path
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)
```
list(a11 = .4, a12 = .1, a22 = .15) CC = list(c11 = .2, c12 = .1, c22 = .10) EE = list(e11 = .4, e12 = .3, e22 = .25) Amod = list(Beta_a1 = .025, Beta_a2 = .025) Cmod = list(Beta_c1 = .025, Beta_c2 = .025) Emod = list(Beta_e1 = .025, Beta_e2 = .025)

Usage

umx_make_TwinData(nMZpairs, nDZpairs = nMZpairs, AA = NULL, CC = NULL, EE = NULL, varNames = "var", mean = 0, sd = 1, nthresh = NULL, sum2one = TRUE, seed = NULL, empirical = FALSE, MZr = NULL, DZr = MZr, Amod = NULL, Cmod = NULL, Emod = NULL)

Arguments

nMZpairs Number of MZ pairs to simulate
nDZpairs Number of DZ pairs to simulate (defaults to nMZpairs)
AA value for A variance. NOTE: See options for use in GxE and Bivariate GxE
CC value for C variance.
EE value for E variance.
varNames name for variables (defaults to 'var')
mean mean for traits (default = 0) (not applied to moderated cases)
sd sd of traits (default = 1) (not applied to moderated cases)
nThresh If supplied, use as thresholds and return mxFactor output? (default is not to)
sum2one Whether to enforce AA + CC + EE summing the one (default = TRUE)
seed Allows user to set.seed() if wanting reproducible dataset
empirical Passed to mvrnorm
MZr If MZr and DZr are set (default = NULL), the function returns dataframes of the request n and correlation.
DZr NULL
Amod Used for Bivariate GxE data: list(Beta_a1 = .025, Beta_a2 = .025)
Cmod Used for Bivariate GxE data: list(Beta_c1 = .025, Beta_c2 = .025)
Emod Used for Bivariate GxE data: list(Beta_e1 = .025, Beta_e2 = .025)

Value

• list of mzData and dzData dataframes containing T1 and T2 plus, if needed M1 and M2 (moderator values)

References

See Also

- `umx_make_TwinData`, `umxGxEbiv`, `umxACE`, `umxGxE`

Other Twin Data functions:
- `umx_long2wide`, `umx_residualize`, `umx_scale_wide_twin_data`, `umx_wide2long`

Other Data Functions:
- `umxCovData`, `umxDescribeDataWLS`, `umxHetCor`, `umxPadAndPruneForDefVars`, `umx_as_numeric`, `umx_cov2raw`, `umx_lower2full`, `umx_make_MR_data`, `umx_make_bin_cont_pair_data`, `umx_make_fake_data`, `umx_polychoric`, `umx_polypairwise`, `umx_polytriowise`, `umx_read_lower`, `umx_rename`, `umx_reorder`, `umx_stack`, `umx_swap_a_block`

Examples

```R
# ==----------------------------------------------------------------------
# = Basic Example, with all elements of std univariate data specified =
# ==----------------------------------------------------------------------
tmp = umx_make_TwinData(nMzPairs = 10000, AA = .30, CC = .00, EE = .70)
# Show list of 2 data sets
str(tmp)
# = How to consume the built datasets =
mzData = tmp[[1]];  
dzData = tmp[[2]];  
cov(mzData);  cov(dzData)  
umxAPA(mzData)
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 = 0.04, 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 =
# =========
```
umx_means

```r
tmp <- umx_make_TwinData(100, AA = .6, CC = .2, nThresh = 3)
str(tmp)

umxAPA(tmp[[1]]); umxAPA(tmp[[2]])

# ===========
# = Just use MZr and DZr =
# ===========

tmp <- umx_make_TwinData(100, MZr = .86, DZr = .60, varNames = "IQ")

umxAPA(tmp[[1]]); umxAPA(tmp[[2]])

# Bivariate GxSES example (see umxGxEbiv)

AA <- list(a11 = .4, a12 = .1, a22 = .15)
CC <- list(c11 = .2, c12 = .1, c22 = .10)
EE <- list(e11 = .4, e12 = .3, e22 = .25)

Amod <- list(Beta_a1 = .025, Beta_a2 = .025)
Cmod <- list(Beta_c1 = .025, Beta_c2 = .025)

Emod <- list(Beta_e1 = .025, Beta_e2 = .025)

tmp <- umx_make_TwinData(5000, AA = AA, CC = CC, EE = EE, Amod = Amod, Cmod = Cmod, Emod = Emod)

# List of 2
# $ mzData:'data.frame': 5000 obs. of 6 variables:
# ..$ defM_T1: num [1:5000] -1.424 -1.606 -0.749 -0.358 -0.208 ... 
# ..$ defM_T2: num [1:5000] -1.703 -1.125 -1.136 0.366 -0.864 ... 
# ..$ M_T1 : num [1:5000] -1.424 -1.606 -0.749 -0.358 -0.208 ... 
# ..$ var_T1 : num [1:5000] -1.1197 -0.3074 -0.4156 -0.4149 0.0739 ... 
# ..$ M_T2 : num [1:5000] -1.703 -1.125 -1.136 0.366 -0.864 ... 
# ..$ var_T2 : num [1:5000] 0.381 -1.275 -1.114 1.297 -1.53 ... 
# $ dzData:'data.frame': 5000 obs. of 6 variables:
# ..$ defM_T1: num [1:5000] 0.253 -0.402 0.545 -1.494 -0.278 ... 
# ..$ defM_T2: num [1:5000] 1.7587 0.3025 -0.3864 0.0737 0.514 ... 
# ..$ M_T1 : num [1:5000] 0.253 -0.402 0.545 -1.494 -0.278 ... 
# ..$ var_T1 : num [1:5000] -0.835 -0.305 -0.299 -1.576 -0.26 ... 
# ..$ M_T2 : num [1:5000] 1.7587 0.3025 -0.3864 0.0737 0.514 ... 
# ..$ var_T2 : num [1:5000] 0.418 0.678 -0.78 -0.312 -0.272 ... 

# TODO tmx example showing how moderation of A introduces heteroskedasticity in a regression model.
# More residual variance at one extreme of the x axis (moderator)
# m1 = lm(var_T1 ~ M_T1, data = x);
# x = rbind(tmp[[1]], tmp[[2]])
# plot(residuals(m1) ~ x$M_T1, data=x)
```

---

**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 Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_r_test, umx_round, umx_var, 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)

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 kludge them together.

Usage

umx_merge_CIs(m1, m2)

Arguments

m1 first copy of the model
m2 second copy of the model

Value

- mxModel
umx_move_file

References
- https://www.github.com/tbates/umx

See Also
Other Advanced Helpers: umx_standardize_ACEcov, umx_standardize_ACEv, umx_standardize_ACE, umx_standardize_CP, umx_standardize_IP, umx_standardize_SexLim, umx_standardize_Simplex, umx_stash_CIs, umx

Examples
```r
## Not run:
umx_merge_CIs(m1, m2)

## End(Not run)
```

---

**umx_move_file**  
*Move files*

**Description**
On OS X, umx_move_file can access the current front-most Finder window. The file moves are fast and, because you can use regular expressions, powerful.

**Usage**
```
umx_move_file(baseFolder = NA, regex = 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 front-most Finder window. If it is blank, a choose folder dialog will be thrown.
- **regex**  
= 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_write_to_clipboard, umx

Examples

```r
## 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**

*Print the name and compact contents of variable.*

Description

Helper function to ease debugging with console notes like: "ObjectName = <object value>". This is primarily useful for inline debugging, where seeing "nVar = NULL" can be useful. The ability to say `umxMsg(nVar)` makes this easy.

Usage

```r
umx_msg(x)
```

Arguments

- `x` the thing you want to pretty-print

Value

- `NULL`

References

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance
Examples

```r
a = "brian"
umx_msg(a)
b = c("brian", "sally", "jane")
umx_msg(b)
umx_msg(mtcars)
```

Description

Convenient equivalent of running `grep` on `names`, with `value = TRUE` and `ignore.case = TRUE`.

**Plus:** `umx_names` can handle dataframes, a model, list of models, model summary, or a vector of strings as input.

In these cases, it will search column names, parameter or summary output names, or the literal string values themselves respectively.

In addition, `umx_names` can do replacement of a found string (see examples). It can also collapse the result (using `paste0`)

*Note:* `namez` (with a z) is a shortcut for `umx_names`, which makes it easy to replace where you’d otherwise use `names`.

You can learn more about the matching options (like inverting the selection etc.) in the help for base-R `grep`.

Usage

```r
umx_names(df, pattern = ".\.*", replacement = NULL, ignore.case = TRUE, perl = FALSE, value = TRUE, fixed = FALSE, useBytes = FALSE, invert = FALSE, global = FALSE, collapse = c("as.is", "vector", "formula"))
```

Arguments

- `df` - dataframe (or other objects, or a list of models) from which to get names.
- `pattern` - Used to find only matching names (supports grep/regular expressions)
- `replacement` - If not NULL, replaces the found string. Use backreferences ("\1" to "\9") to refer to (subexpressions).
- `ignore.case` - default = TRUE (opposite default to grep)
- `perl` - Should Perl-compatible regexps be used? Default = FALSE
- `value` - Return matching elements themselves (TRUE) or their indices (FALSE) default = TRUE (opposite default to grep)
- `fixed` - = FALSE (grep option If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.)
useBytes = FALSE logical grep option. If TRUE, matching is by byte rather than by character.
invert Return indices or values for elements that do not match (default = FALSE).
global replace all instances in each strong, or just the first (Default).
collapse "as.is" leaves alone. as.vector formats as pasteable code, i.e., "c('a', 'b')", not "a" "b" (default NULL), etc.

Value
• vector of matches

References

See Also
• Base-R pattern matching functions: grep. And umx_check_names to check for existence of names in a dataframe.
Other Reporting Functions: loadings.MxModel, tmx_is.identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APB_pval, umx_aggregate, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time, umx, xmu_get_CI
Other String Functions: umx_explode_twin_names, umx_explode, umx_grep, umx_object_as_str, umx_paste_names, umx_rot, umx_trim, umx_write_to_clipboard, umx

Examples
# Names from a dataframe, with character matching
umx_names(mtcars, "mpg") # only "mpg" matches this

# Easy-to-type alias "namez"
namez(mtcars, "mpg")

# Use a regular expression to match a pattern
namez(mtcars, "r\[ab\]") # "drat", "carb"
namez(mtcars, "d") # vars beginning with 'd' = "disp", drat

# Use this function to replace text in names!
umx_names(mtcars, "mpg", replacement = "hello") # "mpg" replaced with "hello"

# = Using the custom collapse option to quote each item, and wrap in c() =
# Other options passed to R's grep command
umx_names(mtcars, "mpg", invert = TRUE) # Non-matches (instead of matches)
umx_names(mtcars, "disp", value = FALSE) # Return indices of matches
umx_object_as_str

umx_names(mtcars, "^d", fixed = TRUE)  # Vars containing literal '^d' (none...)

# = Examples using built-in GFF dataset =
# = Examples using built-in GFF dataset =

# Just show phenotypes for Twin 1
umx_names(GFF, "_T1$")  # Twin 1
# "zyg" "sex1" "age_T1" "gff_T1" "fc_T1" "qol_T1" "hap_T1"...

umx_names(GFF, "^Q")  # names ending in 2
umx_names(GFF, "[^\dbs]$")  # doesn't end in '1', '2', 'b', or 's'
# "zyg_6grp" "zyg_2grp" "divorce"

umx_names(mxData(twinData[, c("wt1", "wt2")], type="raw"))
umx_names(mxData(cov(twinData[, c("wt1", "wt2")], use="comp"), type="cov", numObs=1000))

namez(umxMatrix("bob", "Full", 3,3)$labels)

---

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

See Also
Other String Functions: umx_explode_twin_names, umx_explode, umx_grep, umx_names, umx_paste_names, umx_rot, umx_trim, umx_write_to_clipboard, umx

Examples
umx_object_as_str(mtcars)  # "mtcars"
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

See Also
Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_move_file, umx_rename_file, umx_write_to_clipboard, umx

Examples
## Not run:
 umx_open(getwd())
 umx_open("~/bin/umx/R/misc_and_utility copy.r")

## End(Not run)
umx_open_CCRAN_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
```r
umx_open_CCRAN_page(package = "umx")
```

Arguments
- `package` An R package name.

Value
- 

See Also
Other Miscellaneous Utility Functions: `install.OpenMx`, `qm`, `umxBrownie`, `umxFactor`, `umxVersion`, `umx_array_shift`, `umx_cell_is_on`, `umx_cont_2_quantiles`, `umx_find_object`, `umx_make`, `umx_msg`, `umx_pad`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`, `umx`, `xmu_check_variance`

Examples
```r
## Not run:
umx_open_CCRAN_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
```r
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](https://github.com/kevinushey/Kmisc/tree/master/man)

See Also

Other Miscellaneous Utility Functions: `install.OpenMx`, `qm`, `umxBrownie`, `umxFactor`, `umxVersion`, `umx_array_shift`, `umx_cell_is_on`, `umx_cont_2_quantiles`, `umx_find_object`, `umx_make`, `umx_msg`, `umx_open_CRAN_page`, `umx_pb_note`, `umx_print`, `umx_scale`, `umx_score_scale`, `umx`, `xmu_check_variance`

Examples

```r
umx_pad(1:3, 4)
umx_pad(1:3, 3)
```

---

**umx_parameters**  
*Display path estimates from a model, filtering by name and value.*

**Description**

Often you want to see the estimates from a model, and often you don’t want all of them. `umx_parameters` helps in this case, allowing you to select parameters matching a name filter, and also to only show parameters above or below a certain value.

If pattern is a vector, each regular expression is matched, and all unique matches to the whole vector are returned.

**Usage**

```r
umx_parameters(x, thresh = c("all", "above", "below", "NS", "sig"),
    b = NULL, pattern = ".*", std = FALSE, digits = 2)
```

```r
umxParameters(x, thresh = c("all", "above", "below", "NS", "sig"),
    b = NULL, pattern = ".*", std = FALSE, digits = 2)
```

```r
parameters(x, thresh = c("all", "above", "below", "NS", "sig"),
    b = NULL, pattern = ".*", std = FALSE, digits = 2)
```
**Arguments**

- `x`: an `mxModel` or model summary from which to report parameter estimates.
- `thresh`: optional: Filter out estimates 'below' or 'above' a certain value (default = "all").
- `b`: Combine with `thresh` to set a minimum or maximum for which estimates to show.
- `pattern`: Optional string to match in the parameter names. Default ".*" matches all. `regex` allowed!
- `std`: Standardize output: NOT IMPLEMENTED YET
- `digits`: Round to how many digits (2 = default).

**Details**

It is on my TODO list to implement filtering by significance, and to add standardizing.

**Value**

- list of matching parameters, filtered by name and value

**References**

- [https://github.com/tbates/umx](https://github.com/tbates/umx), [https://tbates.github.io](https://tbates.github.io)

**See Also**

- `parameters`, `umxSummary`, `umx_names`

Other Reporting Functions: `loadings.MxModel`, `tmx_is_identified`, `tmx_show`, `umxAPA`, `umxEval`, `umxFactorScores`, `umxGetParameters`, `umxReduce`, `umxSummary`, `umxWeightedAIC`, `umx/APA_pval`, `umx_aggregate`, `umx_names`, `umx_print`, `umx_show_fit_or_comparison`, `umx_time`, `umx`, `xmu_get_CI`

**Examples**

```r
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
ml <- umxRAM("One Factor", data = mxData(demoOneFactor, type = "raw"),
             umxPath(from = "G", to = manifests),
             umxPath(v.m. = manifests),
             umxPath(v1m0 = "G")
)

# Parameters with values below .1
umx_parameters(ml, "below", .1)
# Parameters with values above .5
umx_parameters(ml, "above", .5)
# Parameters with values below .1 and containing "_to_" in their label
umx_parameters(ml, "below", .1, "_to_")
```
**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")`

*note: `tvars` is a shortcut for `umx_paste_names`*

### Usage

```r
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 = 1:2)
- `covNames`: a list of base names for covariates (to be sorted last in list), e.g `c("age", "sex")`
- `prefix`: a string to prepend to each label, e.g "mean" -> "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:

```r
umx_paste_names(c("var1", "var2"), suffixes = c(".T1", ".T2"))
```

*Note: for quick typing, `tvars` is an alias for `umx_paste_names`*

**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".

```r
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:

```r
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

- [https://tbates.github.io/][1]
- [https://github.com/tbates/umx][2]

See Also

`namez`

Other String Functions: `umx_explode_twin_names`, `umx_explode`, `umx_grep`, `umx_names`, `umx_object_as_str`, `umx_rot`, `umx_trim`, `umx_write_to_clipboard`, `umx`

Examples

```r
# 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)
numx_paste_names(c("IQ", "C"), cov = c("age"), sep = ",_T", prefix = "mean_")
# For quick-typing, tvars is an alias for umx_paste_names
tvars(c("IQ", "C"), cov = c("age"), sep = ",_T", prefix = "mean_")
```

Description

Use the pushbullet service to push a note. You can also initialise this service by providing your key one time.

Usage

```r
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 written to "/pushbulletkey" `umx_pb_note(auth_key="mykeystring")`

Once it exists there, you do not need to store it in code, so code is sharable.

You can get your authorization key at [https://www.pushbullet.com][3] in the "account" section.

Note: You can show the existing stored key using "GET"
umx_polychoric

References


See Also

- umx_msg

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umxopenCRAN_page, umx_pad, umx_print, umx_scale, umx_score_scale, umx, xmu_check_variance

Examples

```r
## Not run:
umx_pb_note("done!", umx_time(m1))

## End(Not run)
```

umx_polychoric  

**FIML-based polychoric, polyserial, and Pearson correlations**

Description

Compute polychoric/polyserial/Pearson correlations with FIML in OpenMx

Usage

```r
umx_polychoric(data, useDeviations = TRUE, tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"))
```

Arguments

- **data**: Dataframe
- **useDeviations**: Whether to code the mode using deviation thresholds (default = TRUE)
- **tryHard**: 'no' uses normal mxRun (default), "yes" uses mxTryHard, and others used named versions: "mxTryHardOrdinal", "mxTryHardWideSearch"

Value

- list of output and diagnostics. matrix of correlations = $polychorics

References

- https://doi.org/10.3389/fpsyg.2016.00528
umx_polypairwise

See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx

Examples

tmp <- mtcars
tmp$am <- umxFactor(mtcars$am)
tmp$vs <- umxFactor(mtcars$vs)
tmp <- umx_scale(tmp)
x <- umx_polychoric(tmp[, c("am", "vs")], tryHard = "yes")
x$polychorics
cor(mtcars[, c("am", "vs")])

umx_polypairwise

FIML-based Pairwise polychoric, polyserial, and Pearson correlations

Description

Compute polychoric/polyserial/Pearson correlations with FIML in OpenMx

Usage

umx_polypairwise(data, useDeviations = TRUE, printFit = FALSE, use = "any", tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"))

Arguments

data Dataframe
useDeviations Whether to code the mode using deviation thresholds (default = TRUE)
printFit Whether to print information about the fit achieved (default = FALSE)
use parameter (default = "any")
tryHard 'no' uses normal mxRun (default), "yes" uses mxTryHard, and others used named versions: "mxTryHardOrdinal", "mxTryHardWideSearch"

Value

- matrix of correlations

References

- https://doi.org/10.3389/fpsyg.2016.00528
umx_polytriowise

See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx

Examples

```r
umx_set_optimizer("SLSQP")
tmp <- mtcars
tmp$am <- umxFactor(mtcars$am)
tmp$vs <- umxFactor(mtcars$vs)
tmp <- umx_scale(tmp)
X <- umx_polypairwise(tmp[, c("hp", "mpg", "am", "vs")], tryHard = "yes")
X$R
cov2cor(X$R)
cor(mtcars[, c("hp", "mpg", "am", "vs")])
```

umx_polytriowise  FIIML-based trio-based polychoric, polyserial, and Pearson correlations

Description

Compute polychoric/polyserial/Pearson correlations with FIML in OpenMx.

Usage

```r
umx_polytriowise(data, useDeviations = TRUE, printFit = FALSE, use = "any", tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal", "mxTryHardWideSearch"))
```

Arguments

data: Dataframe
useDeviations: Whether to code the mode using deviation thresholds (default = TRUE)
printFit: Whether to print information about the fit achieved (default = FALSE)
use: parameter (default = "any")
tryHard: 'no' uses normal mxRun (default), "yes" uses mxTryHard, and others used named versions: "mxTryHardOrdinal", "mxTryHardWideSearch"

Value

- matrix of correlations

References

- [https://doi.org/10.3389/fpsyg.2016.00528](https://doi.org/10.3389/fpsyg.2016.00528)
See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_read_lower, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx

Examples

tmp = mtcars
tmp$am = umxFactor(mtcars$am)
tmp$vs = umxFactor(mtcars$vs)
tmp = umx_scale(tmp)
x = umx_polytriowise(tmp[, c("hp", "mpg", "am", "vs")], tryHard = "yes")
xS$
cor(mtcars[, c("hp", "mpg", "am", "vs")])

umx_print

Print tables in a range of formats (markdown default, see umx_set_table_format for other formats) or as a web browser table.

Description

To aid interpretability of printed tables from OpenMx (and elsewhere) you can change how NA and zero appear, and suppressing 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 table in browser.
suppress minimum numeric value to print (default = NULL, print all values, no matter how small)
... Optional parameters for print
umx_read_lower

Value

- A dataframe of text

See Also

umx_msg, umx_set_table_format

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_scale, umx_score_scale, umx, xmu_check_variance

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APB_pval, umx_aggregate, umx_names, umx_parameters, umx_show_fit_or_comparison, umx_time, umx

Examples

umx_print(mtcars[1:10,], digits = 2, zero.print = ".", justify = "left")
umx_print(mtcars[1,1:2], digits = 2, zero.print = ""
## Not run:
umx_print(mtcars[1:10,], file = "tmp.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 = NULL, ensurePD = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>Path to a file to read (Default &quot;&quot; will read from user input)</td>
</tr>
<tr>
<td>diag</td>
<td>Whether the data include the diagonal. Defaults to TRUE</td>
</tr>
<tr>
<td>names</td>
<td>The default names for the variables. Defaults to as.character(paste(&quot;X&quot;, 1:n, sep=&quot;&quot;))</td>
</tr>
<tr>
<td>ensurePD</td>
<td>Whether to coerce the resultant matrix to positive definite (Defaults to FALSE)</td>
</tr>
</tbody>
</table>
Value

- matrix

References


See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_c2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_rename, umx_reorder, umx_stack, umx_swap_a_block, umx

Examples

```r
## Not run:
require(umx) # for umxRAM
IQtests = c("brainstorm", "matrix", "moral", "shopping", "typing")
allCols = c("C", IQtests, "avgIQ", "maxIQ", "video")

df = umx_read_lower(file = ",", diag = FALSE)
  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

dimnames(df) = list(allCols, allCols) # manually add

df = umx_read_lower(file = ",", diag = FALSE, names = allCols, 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

round(df, 2)

m1 = umxRAM("wooley", data = mxData(df, type="cov", numObs = 90),
  umxPath("g", to = IQtests),
  umxPath(var = "g", fixedAt= 1),
  umxPath(var = IQtests)
)
summary(m1)
```
umx_rename

Description

Returns a dataframe with variables renamed as desired.

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

Unlike similar functions in other packages, it checks that the variables exist, and that the new names do not.

Importantly, it also supports regular expressions. This allows you to find and replace text based on patterns and replacements. so to change "replacement" to "in place", grep=re(place)ment, replace= in \1.

*note*: to use replace list, you must say `c(old = "new")`, not `c(old -> "new")`

Value

- dataframe with columns renamed.
See Also

namez to filter (and replace) names, Also umx_check_names to check for existence of names in a dataframe.

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_reorder, umx_stack, umx_swap_a_block, 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"), test= TRUE)
x = umx_rename(x, old = c("disp"), replace = c("displacement"))

umx_check_names("displacement", data = x, die = TRUE)

# This will warn that "disp" does not exist (anymore)
x = umx_rename(x, old = c("disp", "am"), replace = c("displacement", "auto"))
x = umx_rename(x, grep = "lacement", replace = ",", test=TRUE) # test using grep to revert to disp
x = umx_rename(x, grep = "lacement", replace = ",") # using grep to revert to disp

umx_names(x, "^d") # all names beginning with a d

----------

umx_rename_file umx_rename_file

Description

rename files. On OS X, the function can access the current front-most 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., "[a-zA-Z]t"
replaceStr The (regex) replacement string \l are not dogs
baseFolder The folder to search in. If set to "Finder" (and you are on OS X) it will use the current front-most 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)
umx_reorder

Value
-

References
- https://www.github.com/tbates/umx

See Also
Other File Functions: dl_from_dropbox, umx_make_sql_from_excel, umx_move_file, umx_open, umx_write_to_clipboard, umx

Examples
```r
# Not run:
# "Season 01" --> "S01" in current folder in MacOS Finder
umx_rename_file("[Ss]season +([0-9]+)", replaceStr="S\1", baseFolder = "Finder", test = TRUE)

# End(Not run)
```

---

| umx_reorder | Reorder or drop variables from a correlation/covariance matrix. |

Description
Reorder the variables in a correlation matrix. Can also remove one or more variables from a matrix using this function.

Usage
```r
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
- https://www.github.com/tbates/umx
**umx_residualize**

See Also

Other Data Functions: *umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_stack, umx_swap_a_block, umx*

Examples

```r
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**  
*Easily residualize variables in long or wide dataframes, returning them changed in-place.*

**Description**

Residualize one or more variables residualized against covariates, and return a complete dataframe with residualized variable in place. Optionally, this also works on wide (i.e., 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:

```r
tmp <- residuals(lm(var ~ cov1 + cov2, data = data, na.action = na.exclude))
```

This tmp variable could then be written over the old data:

```r
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


See Also

Other Twin Data functions: umx_long2wide, umx_make_TwinData, umx_scale_wide_twin_data, umx_wide2long, umx

Examples

# Residualize 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$mpg == r2)
# = formula interface =
# = Demonstrate ability to residualize WIDE data (i.e. 1 family per row) =
tmp = mtcars
tmp$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]

# = Residualize 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_round

### Usage

```
umx_rot(vec)
```

### Arguments

- `vec`  
  vector to rotate

### Value

- `mxModel`

### References

- [https://tbates.github.io](https://tbates.github.io)

### See Also

Other String Functions: `umx_explode_twin_names, umx_explode, umx_grep, umx_names, umx_object_as_str, umx_paste_names, umx_trim, umx_write_to_clipboard, umx`

### Examples

```
umx_rot(1:10)
numx_rot(c(3,4,5,6,7))
# [1] 4 5 6 7 3
```

---

## 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`
umx_r_test

References
- https://www.github.com/tbates/umx

See Also
Other Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_var, umx

Examples
head(umx_round(mtcars, coerce = FALSE))
head(umx_round(mtcars, coerce = TRUE))

umx_r_test  Test the difference between correlations for significance.

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 non-overlapping (no variable in common) correlations in the same dataset. In the future it will be expanded to handle overlapping correlations, and to take correlation matrices as input.

Value
-

See Also
Other Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_var, umx
umx_scale

Examples
vars = c("mpg", "cyl", "disp", "hp")
umx_r_test(mtcars, vars)

Description
umx_scale applies scale to the columns of a data.frame. By default it scales all numeric columns, and is smart enough to skip non-scalable columns (strings, factors, etc.).

You can also select which columns to convert. This is useful when you want to avoid numeric columns which are actually factors.

note: By default, the attributes which scale adds ("scaled:center" and "scaled:scale" removed to leave nice numeric columns. Set attr= TRUE to preserve these.

Usage
umx_scale(df, varsToScale = NULL, coerce = FALSE, attr = FALSE, verbose = FALSE)

Arguments
df A dataframe to scale (or a numeric vector)
varsToScale (leave blank to scale all)
coerce Whether to coerce non-numeric to numeric (Defaults to FALSE.
attr to strip off the attributes scale creates (FALSE by default)
verbose Whether to report which columns were scaled (default FALSE)

Value
• new dataframe with scaled variables

References
• https://www.github.com/tbates/umx

See Also
umx_scale-wide_twin_data scale

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_score_scale, umx, xmu_check_variance
umx_scale_wide_twin_data

Examples

data(twinData)
  df = umx_scale(twinData, varsToScale = c("wt1", "wt2"))
  df = umx_scale(twinData, attr = TRUE)
  plot(wt1 ~ wt2, data = df)

umx_scale_wide_twin_data
  df = umx_scale_wide_twin_data(data = twinData, varsToScale = c("ht", "wt"), sep = "")
  plot(wt1 ~ wt2, data = df)

Description

Scale wide data across all cases: currently 2 twins.

Usage

umx_scale_wide_twin_data(varsToScale, sep, data)

Arguments

varsToScale  The base names of the variables ("weight" etc.)
sep          The suffix that distinguishes each case, e.g. ".T"
data         a wide dataframe

Value

- new dataframe with variables scaled in place

References

- [https://www.github.com/tbates/umx](https://www.github.com/tbates/umx)

See Also

umx_scale

Other Twin Data functions: umx_long2wide, umx_make_TwinData, umx_residualize, umx_wide2long, umx

Examples

data(twinData)
  df = umx_scale_wide_twin_data(data = twinData, varsToScale = c("ht", "wt"), sep = "")
  plot(wt1 ~ wt2, data = df)
Score a psychometric scale by summing normal and reversed items

Description

use this function to generate scores as the sum of the responses to the normal and reversed items in a scale.

Items must be named on the pattern <base><n>. base is the string common to all item names (variable names). pos and rev are vectors of the item numbers for the normal and reverse-scored item numbers. To reverse items, the function uses itemMax is the high score (to compute how to reverse items). min defaults to 1.

Usage

umx_score_scale(base = NULL, pos = NULL, rev = NULL, min = 1, max = NULL, data = NULL, score = c("totals", "mean"), name = NULL)

Arguments

base String common to all item names.
pos The positive-scored item numbers.
rev The reverse-scored item numbers.
min Min possible score (default = 1). Not implemented for values other than 1 so far...
max Max possible score for an item (to compute how to reverse items).
data The data frame
score = Totals or Mean (default = "totals")
name = name of the scale to be returned. Defaults to ",score"

Value

- scores

See Also

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx, xmu_check_variance
Examples

library(psych)
tmp = umx_score_scale("A", pos = 2:5, rev = 1, max = 6, data = bfi, name = "A")
tmp = umx_score_scale("E", pos = c(3,4,5), rev = c(1,2), max = 6, data = tmp, name = "E")

# Using @BillRevelle's psych package: More diagnostics, including alpha
scores = psych::scoreItems(items = bfi, min = 1, max = 6, keys = list(
  E = c("-E1", "-E2", "E3", "E4", "E5"),
))
summary(scores)
print(scores)

# Compare output
# (note, by default psych::scoreItems replaces NAs with the sample median...)
RevelleE = as.numeric(scores$scores[,"E"]) * 5
all(RevelleE == tmp[,"E"], na.rm = TRUE)
umx_set_auto_run

Examples

library(umx)
umx_set_auto_plot() # print current state
old = umx_set_auto_plot(silent = TRUE) # store existing value
umx_set_auto_plot(TRUE) # set to on (internally stored as "name")
umx_set_auto_plot(FALSE) # set to off (internally stored as NA)
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


See Also

Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx

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

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

See Also
Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx

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)
lattents = c("G")
manifests = names(demoOneFactor)
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

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx]

See Also

Other Get and set: `umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx`

Examples

```r
library(umx)

# Set condensed slots
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**

---

**Description**

set the number of cores (threads) used by OpenMx

**Usage**

```r
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**


**See Also**

Other Get and set: `umx_default_option`, `umx_get_checkpoint`, `umx_get_options`, `umx_set_auto_plot`, `umx_set_auto_run`, `umx_set_checkpoint`, `umx_set_condensed_slots`, `umx_set_data_variance_check`, `umx_set_optimization_options`, `umx_set_optimizer`, `umx_set_plot_file_suffix`, `umx_set_plot_format`, `umx_set_table_format`, `umx`

**Examples**

```r
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 usage to 1 core
```
**umx_set_data_variance_check**

umx_set_cores(model = m1)  # show new value for m1
umx_set_cores(oldCores)    # reinstate old global value

---

**Description**

Set default for data checking in models like umxACE umxGxE etc.

**Usage**

```r
umx_set_data_variance_check(minVar = NULL, maxVarRatio = NULL,
                            silent = FALSE)
```

**Arguments**

- `minVar`: Set the threshold at which to warn user about variables with too-small variance. Else returns the current value of umx_minVar
- `maxVarRatio`: Set the option for threshold at which to warn user variances differ too much. Else returns the current value of umx_maxVarRatio
- `silent`: If TRUE, no message will be printed.

**Value**

- list of umx_minVar and umx_maxVarRatio settings

**See Also**

xmu_check_variance which uses these to check sanity in the variances of a data frame.

Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format, umx

**Examples**

```r
library(umx)
umx_set_data_variance_check()  # print current state
old = umx_set_data_variance_check(silent = TRUE)  # store existing value
umx_set_data_variance_check(minVar = .01)
umx_set_data_variance_check(maxVarRatio = 500)
umx_set_data_variance_check(minVar = old$minVar, maxVarRatio = old$maxVarRatio)  # reinstate
```
umx_set_optimization_options

Description
Set options that affect optimization in OpenMx. For mvnRelEps, values between .0001 to .01 are conventional. Smaller values slow optimization.

Usage
umx_set_optimization_options(opt = c("mvnRelEps", "mvnMaxPointsA"),
value = NULL, model = NULL, silent = FALSE)

Arguments
- opt
default returns current values of the options listed. Currently "mvnRelEps" and "mvnMaxPointsA".
- value
If not NULL, the value to set the opt to (can be a list of length(opt))
- 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

See Also
Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot,
umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check,
umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx_set_table_format,
umx

Examples
umx_set_optimization_options() # print the existing state(s)
umx_set_optimization_options("mvnRelEps") # show this one
## Not run:
umx_set_optimization_options("mvnRelEps", .01) # update globally
## End(Not run)
**umx_set_optimizer**

**Description**
Set the optimizer in OpenMx

**Usage**

```r
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**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

Other Get and set: `umx_default_option`, `umx_get_checkpoint`, `umx_get_options`, `umx_set_auto_plot`,
`umx_set_auto_run`, `umx_set_checkpoint`, `umx_set_condensed_slots`, `umx_set_cores`, `umx_set_data_variance_check`,
`umx_set_optimization_options`, `umx_set_plot_file_suffix`, `umx_set_plot_format`, `umx_set_table_format`, `umx`

**Examples**

```r
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_file_suffix

Set output suffix used in umx plot (structural diagrams) files to disk

Description

Set output file suffix (default = "gv", alternative is "dot"). If you call this with no value, it will return the current setting. If you call it with TRUE, it toggles the setting.

Usage

umx_set_plot_file_suffix(umx.plot.suffix = NULL, silent = FALSE)

Arguments

- umx.plot.suffix: the suffix for plots files (if empty, returns the current value of umx.plot.format). If "TRUE", then toggles
- silent: If TRUE, no message will be printed.

Value

- Current umx.plot.suffix setting

References


See Also

Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot, umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check, umx_set_optimization_options, umx_set_optimizer, umx_set_plot_format, umx_set_table_format, umx

Examples

umx_set_plot_file_suffix() # print current state
old = umx_set_plot_file_suffix(silent = TRUE) # store current value
umx_set_plot_file_suffix("dot")
umx_set_plot_file_suffix("gv")
umx_set_plot_file_suffix(old) # reinstate
umx_set_plot_format

**Set output format of plots (structural diagrams) in umx**

**Description**

Set output format of plots (default = "DiagrammeR", alternative is "graphviz"). If you call this with no value, it will return the current setting. If you call it with TRUE, it toggles the setting.

**Usage**

```r
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). If "TRUE", then toggles
- `silent`: If TRUE, no message will be printed.

**Value**

- Current umx.plot.format setting

**References**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

Other Get and set: `umx_default_option`, `umx_get_checkpoint`, `umx_get_options`, `umx_set_auto_plot`, `umx_set_auto_run`, `umx_set_checkpoint`, `umx_set_condensed_slots`, `umx_set_cores`, `umx_set_data_variance_check`, `umx_set_optimization_options`, `umx_set_optimizer`, `umx_set_plot_file_suffix`, `umx_set_table_format`, `umx

**Examples**

```r
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

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


See Also

Other Get and set: umx_default_option, umx_get_checkpoint, umx_get_options, umx_set_auto_plot,
umx_set_auto_run, umx_set_checkpoint, umx_set_condensed_slots, umx_set_cores, umx_set_data_variance_check,
umx_set_optimization_options, umx_set_optimizer, umx_set_plot_file_suffix, umx_set_plot_format, umx

Examples

```r
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_fit_or_comparison

Show model logLik of model or print comparison table

Description

Just a helper to show the logLik of a model or print a comparison table is a function which

Usage

umx_show_fit_or_comparison(model, comparison = NULL, digits = 2)

Arguments

model an mxModel to report on
comparison If not NULL, used as comparison model
digits (default = 2)

Value

-

See Also

- umxSummary

Other Reporting Functions: loadings, MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_time, umx, xmu_get_CI

Examples

## Not run:
umx_show_fit_or_comparison(model, comparison, digits=3)

## End(Not run)
umx_stack

Stack data like stack() does, with more control.

Description

Operates like stack, but can preserve ("passalong") other variables on each row, and allows the user control over the values and group column names for ease of use.

Usage

umx_stack(x, select, passalong, valuesName = "values",
          groupName = "ind")

Arguments

x             a dataframe containing twin data.
select        The variables to stack (wide 2 long)
passalong     Variables to preserve on each row (e.g. age)
valuesName    The name for the new stacked column (default = "values")
groupName     The name for the column containing the grouping variable (default = "ind")

Value

- long-format dataframe

See Also

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars,
umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair,
umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriwise, umx_read_lower,
umx_rename, umx_reorder, umx_swap_a_block, umx

Examples

# Base-R stack function
df = stack(mtcars, select = c("disp", "hp"), drop=FALSE)

# umx_stack, with additional variables passed along
df = umx_stack(mtcars, select = c("disp", "hp"), passalong = "mpg")
str(df) # ind is a factor, with levels select

ggplot2::qplot(x = mpg, y = values, color=ind, data = df)
df = umx_stack(mtcars, select = c("disp", "hp"), passalong = "mpg")

ggplot2::qplot(x = mpg, y = values, group="ind", data = df)
umx_standardize

Return a standardized version of a Structural Model

Description

Return the standardized version of a model (such as ACE, CP etc.)
Versions exist for RAM, ACE, ACEv, ACEcov, IP, CP and GxE models.

Usage

umx_standardize(model, ...)

Arguments

model The mxModel whose fit will be reported.
... Other parameters.

Details

umx_standardize takes umx models, including RAM and twin models, and returns a standardized version.

umx_standardize_ACE

Description

Standardize an ACE model

Usage

umx_standardize_ACE(model, ...)

Arguments

model an umxACE model to standardize
... Other options

Value

- Standardized ACE umxACE model

References

See Also

Other zAdvanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEcov`, `umx_standardize_ACE`, `umx_standardize_CP`, `umx_standardize_IP`, `umx_standardize_SexLim`, `umx_standardize_Simplex`, `umx_stash_CIs`, `umx`

Examples

```r
require(umx)
data(twinData)
selDV$s = c("bmi1", "bmi2")
mzData <- twinData[twinData$zygosity %in% "MZFF", selDV$s][1:80,] # 80 pairs for speed
dzData <- twinData[twinData$zygosity %in% "DZFF", selDV$s][1:80,]
m1 = umxACE(se1DV$s = selDV$s, dzData = dzData, mzData = mzData)
std = umx_standardize_ACE(m1)
```

Description

Standardize an ACE model with covariates

Usage

```r
umx_standardize_ACEcov(model, ...)
```

Arguments

- `model` an `umxACEcov` model to standardize
- `...` Other options

Value

- Standardized `umxACEcov` model

References

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

Other zAdvanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEev`, `umx_standardize_ACE`, `umx_standardize_CP`, `umx_standardize_IP`, `umx_standardize_SexLim`, `umx_standardize_Simplex`, `umx_stash_CIs`, `umx`
umx_standardize_ACEv

Examples

```r
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
selDVs = c("bmi")
selCovs = c("ht") # silly example
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,
               sep = "", autoRun = TRUE)
fit = umx_standardize_ACEcov(m1)
```

---

### umx_standardize_ACEv

Standardize an ACE variance components model (ACEv)

---

**Description**

*umx_standardize_ACE* allows *umx_standardize* to standardize an ACE variance components model.

**Usage**

```
umx_standardize_ACEv(model, ...)
```

**Arguments**

- `model`  
  An *umxACEv* model to standardize.

- `...`  
  Other parameters.

**Value**

- A standardized *umxACEv* model.

**References**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

Other zAdvanced Helpers: *umx_merge_CIs, umx_standardize_ACEcov, umx_standardize_ACE, umx_standardize_CP, umx_standardize_IP, umx_standardize_SexLim, umx_standardize_Simplex, umx_stash_CIs, umx*
Examples

```r
require(umx)
data(twinData)
selDVs = c("bmi")
mzData <- twinData[twinData$zygosity %in% "MZFF",][1:80,] # 80 pairs for speed
dzData <- twinData[twinData$zygosity %in% "DZFF",][1:80,]
m1 = umxACEv(selDVs = selDVs, sep="", dzData = dzData, mzData = mzData)
std = umx_standardize(m1)
```

Description

This function simply inserts the standardized CP components into the ai ci ei and as cs es matrices

Usage

```r
umx_standardize_CP(model, ...)
```

Arguments

- `model` an `umxCP` model to standardize
- `...` Other options

Value

- standardized `umxCP` model

References

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

Other zAdvanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEcov`, `umx_standardize_ACEv`, `umx_standardize_ACE`, `umx_standardize_IP`, `umx_standardize_SexLim`, `umx_standardize_Simplex`, `umx_stash_CIs`, `umx`

Examples

```r
## Not run:
model = umx_standardize_CP(model)
## End(Not run)
```
**Description**

This function simply copies the standardized IP components into the ai ci ei and as cs es matrices

**Usage**

```r
umx_standardize_IP(model, ...)
```

**Arguments**

- `model`: an `umxIP` model to standardize
- `...`: Other options

**Value**

- standardized IP `umxIP` model

**References**

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

**See Also**

Other zAdvanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEcov`, `umx_standardize_ACEv`, `umx_standardize_ACE`, `umx_standardize_CP`, `umx_standardize_SexLim`, `umx_standardize_Simplex`, `umx_stash_CIs`, `umx`

**Examples**

```r
## Not run:
model = umx_standardize_IP(model)

## End(Not run)```
umx_standardize_RAM  |  Return a standardized version of a Structural Model

Description

umx_standardize_RAM takes a RAM-style model, and returns standardized version.

Usage

umx_standardize_RAM(model, ...)

Arguments

model  |  The `mxModel` you wish to standardize
...  |  Other options

References

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

See Also

Other Reporting functions: RMSEA.MxModel, RMSEA.summary.mxmodel, RMSEA.extractAIC.MxModel, loadings, residuals.MxModel, umxCI_boot, umxCI, umxCompare, umxConfint, umxExpCov, umxExpMeans, umxFitIndices, umxPlotACEv, umxSummary.MxModel, umxSummaryACEV, umxSummaryACE, umxSummarySexLim, umxSummarySimplex, umx_drop_ok

Examples

require(umx)
data(demoOneFactor)
latents = c("g")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov", umxPath(latents, to = manifests), umxPath(var = manifests), umxPath(var = latents, fixedAt = 1.0))

m1 = umx_standardize_RAM(m1)
m1 = umx_standardize(m1)
umxSummary(m1)
\texttt{umx\_standardize\_SexLim}

\textit{Standardize a SexLim model}

\section*{Description}
\texttt{umx\_standardize\_SexLim} would move standardized Sexlim values into raw cells, but can’t as these are algebras.

\section*{Usage}
\texttt{umx\_standardize\_SexLim(model, \ldots)}

\section*{Arguments}
\begin{itemize}
  \item \texttt{model} an \texttt{umxSexLim} model to standardize
  \item \texttt{\ldots} Other options
\end{itemize}

\section*{Value}
- standardized \texttt{umxSexLim} model

\section*{See Also}
Other zAdvanced Helpers: \texttt{umx\_merge\_CIs, umx\_standardize\_ACEcov, umx\_standardize\_ACEv, umx\_standardize\_ACE, umx\_standardize\_CP, umx\_standardize\_IP, umx\_standardize\_Simplex, umx\_stash\_CIs, umx}

\section*{Examples}
\begin{verbatim}
### Not run:
model = umx\_standardize\_IP(model)
### End(Not run)
\end{verbatim}

\texttt{umx\_standardize\_Simplex}

\textit{Standardize a Simplex twin model}

\section*{Description}
\texttt{umx\_standardize\_Simplex}

\section*{Usage}
\texttt{umx\_standardize\_Simplex(model, \ldots)}
Arguments

- **model**: An `umxSimplex` model to standardize
- **digits**: rounding.
- **dropZeros**: makes strings for failed CIs?
- **stdAlg2mat**: treat std as algebra: stash in non std matrix.

Value

- Standardized Simplex `umxSimplex` model

References

- [https://tbates.github.io](https://tbates.github.io), [https://github.com/tbates/umx](https://github.com/tbates/umx)

See Also

Other Advanced Helpers: `umx_merge_CIs`, `umx_standardize_ACEcov`, `umx_standardize_ACEv`, `umx_standardize_ACE`, `umx_standardize_CP`, `umx_standardize_IP`, `umx_standardize_SexLim`, `umx_stash_CIs`, `umx`

Examples

```r
# Not run:
data(iqdat)
mzData = subset(iqdat, zygosity == "MZ")
dzData = subset(iqdat, zygosity == "DZ")
m1 = umxSimplex(seLVs = paste0("IQ_age", 1:4), sep = ",",
dzData = dzData, mzData = mzData, tryHard = "mxTryHard")
std = umx_standardize_simplex(m1)

# End(Not run)
```

---

**umx_stash_CIs**

Stash the CI values of a model as strings in the values of the model.

Description

Stash formatted CIs (e.g. "1 [-1, 1]") as strings overwriting the parameter values of the model.

Usage

```r
umx_stash_CIs(model, digits = 3, dropZeros = FALSE,
               stdAlg2mat = TRUE)
```

Arguments

- **model**: An `mxFit` to get CIs from.
- **digits**: rounding.
- **dropZeros**: makes strings for failed CIs?
- **stdAlg2mat**: treat std as algebra: stash in non std matrix.
umx_string_to_algebra

Details

I might change this to a lookup-function that gets a CI string if one exists.

Value

- mxModel

References

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

CIname = top.cp_loadings_std[1,1] thisSub = top; thisMat = cp_loadings_std; thisRow = 1; thisCol = 1

See Also

- umxConfint, xmu_get_CI

Other zAdvanced Helpers: umx_merge_CIs, umx_standardize_ACEcov, umx_standardize_ACEv, umx_standardize_ACE, umx_standardize_CP, umx_standardize_IP, umx_standardize_SexLim, umx_standardize_Simplex, umx

umx_string_to_algebra  Convert a string to an OpenMx algebra

Description

This is useful use to quickly and easily insert values from R variables into the string (using paste() and rep() etc.), then parse the string as an mxAlgebra argument. A use case is including 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

- https://www.github.com/tbates/umx
See Also

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

Examples

```r
## Not run:
alg = umx_string_to_algebra(paste(rep("A", nReps), collapse = " %"), name = "test_case")

## End(Not run)
```

---

\textit{umx_swap_a_block} \hspace{1cm} \textit{umx_swap_a_block}

Description

Swap a block of rows of a dataset between two sets of variables (typically twin 1 and twin 2)

Usage

```r
umx_swap_a_block(thedata, rowSelector, T1Names, T2Names)
```

Arguments

- **thedata**: A data frame to swap within.
- **rowSelector**: Rows to swap between first and second set of columns.
- **T1Names**: The first set of columns.
- **T2Names**: The second set of columns.

Value

- dataframe

See Also

- subset

Other Data Functions: umxCovData, umxDescribeDataWLS, umxHetCor, umxPadAndPruneForDefVars, umx_as_numeric, umx_cov2raw, umx_lower2full, umx_make_MR_data, umx_make_TwinData, umx_make_bin_cont_pair, umx_make_fake_data, umx_polychoric, umx_polypairwise, umx_polytriowise, umx_read_lower, umx_rename, umx_reorder, umx_stack, 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")

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.
See Also

Other Reporting Functions: loadings.MxModel, tmx_is_identified, tmx_show, umxAPAM, umxEval, umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval, umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx, xmu_get_CI

Examples

```R
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**

*Trim whitespace surrounding a string.*

Description

Returns string without leading or trailing whitespace, like the php function. See also built-in `trimws` does the same.

Usage

```R
umx_trim(string, removeThis = NULL)
```

Arguments

- `string` to trim
- `removeThis` if not NULL then this string is removed wherever found in 'string'

Value

- `string`

References

umx_var

See Also

trimws

Other String Functions: umx_explode_twin_names, umx_explode, umx_grep, umx_names, umx_object_as_str, umx_paste_names, umx_rot, umx_write_to_clipboard, umx

Examples

```r
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

```r
umx_var(df, ordVar = 1, format = c("full", "diag", "lower"),
        use = c("complete.obs", "pairwise.complete.obs", "everything",
                "all.obs", "na.or.complete"), digits = NULL, strict = TRUE)
```

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).
- **digits**: Ignored if NULL. Set for easy printing.
- **strict**: Whether to allow non-ordered factors to be processed (default = FALSE (no)).

Value

- mxModel

References

- [https://tbates.github.io](https://tbates.github.io)
See Also

Other Miscellaneous Stats Helpers: reliability, umxCov2cor, umxHetCor, umx_apply, umx_cor, umx_fun_mean_sd, umx_means, umx_r_test, umx_round, umx

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")
umx_var(tmp2, format = "full")

---

umx_wide2long

Change data family data from wide (2 twins per row) to long format.

Description

Just detects the data columns for twin 1, and twin 2, then returns them stacked on top of each other (rbind) with the non-twin specific columns copied for each as well.

Usage

umx_wide2long(data, sep = "_T", verbose = FALSE)

Arguments

data a dataframe containing twin data.
sep the string between the var name and twin suffix, i.e., var_T1 = _T
verbose Report the non-twin and twin columns (default = FALSE).

Value

- long-format dataframe

See Also

Other Twin Data functions: umx_long2wide, umx_make_TwinData, umx_residualize, umx_scale_wide_twin_data, umx

Examples

long = umx_wide2long(data = twinData, sep = "")
long = umx_wide2long(data = twinData, sep = "", verbose = TRUE)
str(long)
str(twinData)
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

 Works on Mac. Let me know if it fails on windows or Unix.

Value

 -

See Also

 Other String Functions: umx_explode_twin_names, umx_explode, umx_grep, umx_names, umx_object_as_str, umx_paste_names, umx_rot, umx_trim, umx

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

Examples

 ## Not run:
 umx_write_to_clipboard(“hello”)

 ## End(Not run)
us_skinfold_data

**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). Biceps and Triceps are folds above and below the upper arm (holding arm palm upward), Calf (fold on the calf muscle), Subscapular (fold over the shoulder blade), Suprailiacal (fold between the hip and ribs).

**Usage**

```r
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` log BMI of twin 1
- `bic_T1` Biceps Skinfold of twin 1
- `caf_T1` Calf Skinfold of twin 1
- `ssc_T1` Subscapular Skinfold 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` log 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

See Also
Other datasets: Fischbein_wt, GFF, iqdat, umx

Examples
```r
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
```

Description
Tests if an input has square brackets

Usage
```r
xmuHasSquareBrackets(input)
```

Arguments
- `input` an input to test

Value
- TRUE/FALSE

See Also
Other xmu internal not for end user: umxModel, umx, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts
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
- `setfree`: Whether to set free cells
- `drop`: What values to drop
- `jiggle`: Whether to jiggle start values
- `boundDiag`: Set diagonal element bounds 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 "name_r1c1" where name is the matrix name, and r1c1 = row 1 col 1. 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")`
xmuLabel_MATRIX_Model

Value
- The labeled \texttt{mxMatrix}

See Also
Other xmu internal not for end user: \texttt{umxModel}, \texttt{umx}, \texttt{xmuHasSquareBrackets}, \texttt{xmuLabel_MATRIX_Model},
\texttt{xmuLabel_RAM_Model}, \texttt{xmuMI}, \texttt{xmuMakeDeviationThresholdsMatrices}, \texttt{xmuMakeOneheadedPathsFromPathList},
\texttt{xmuMakeTwoheadedPathsFromPathList}, \texttt{xmuMaxLevels}, \texttt{xmuMinLevels}, \texttt{xmuPropagateLabels},
\texttt{xmu_assemble_twin_supermodel}, \texttt{xmu_check_levels_identical}, \texttt{xmu_clean_label}, \texttt{xmu_dot_make_paths},
\texttt{xmu_dot_make_residuals}, \texttt{xmu_dot_maker}, \texttt{xmu_dot_move_ranks}, \texttt{xmu_dot_rank_str}, \texttt{xmu_lavaan_process_group},
\texttt{xmu_make_mxData}, \texttt{xmu_make_top_twin}, \texttt{xmu_model_needs_means}, \texttt{xmu_safe_run_summary}, \texttt{xmu_set_sep_from_suffix},
\texttt{xmu_simplex_corner}, \texttt{xmu_start_value_list}, \texttt{xmu_starts}

---

\section*{xmuLabel_MATRIX_Model} \xmuLabel_MATRIX_Model (not a user function)

\section*{Description}
This function will label all the free parameters in a (non-RAM) OpenMx \texttt{mxModel}.
\texttt{nb}: We don’t assume what each matrix is for. Instead, the function just sticks labels like "a_r1c1" into each cell
i.e., matrix-name + _ + r + rowNumber + c + colNumber

\section*{Usage}
\texttt{xmuLabel_MATRIX_Model(model, suffix = ",", verbose = TRUE)}

\section*{Arguments}
- \texttt{model} a matrix-style \texttt{mxModel} to label
- \texttt{suffix} a string to append to each label
- \texttt{verbose} how much feedback to give

\section*{Details}
End users should just call \texttt{umxLabel}

\section*{Value}
- The labeled \texttt{mxModel}

\section*{See Also}
Other xmu internal not for end user: \texttt{umxModel}, \texttt{umx}, \texttt{xmuHasSquareBrackets}, \texttt{xmuLabel_Matrix},
\texttt{xmuLabel_RAM_Model}, \texttt{xmuMI}, \texttt{xmuMakeDeviationThresholdsMatrices}, \texttt{xmuMakeOneheadedPathsFromPathList},
\texttt{xmuMakeTwoheadedPathsFromPathList}, \texttt{xmuMaxLevels}, \texttt{xmuMinLevels}, \texttt{xmuPropagateLabels},
\texttt{xmu_assemble_twin_supermodel}, \texttt{xmu_check_levels_identical}, \texttt{xmu_clean_label}, \texttt{xmu_dot_make_paths},
\texttt{xmu_dot_make_residuals}, \texttt{xmu_dot_maker}, \texttt{xmu_dot_move_ranks}, \texttt{xmu_dot_rank_str}, \texttt{xmu_lavaan_process_group},
\texttt{xmu_make_mxData}, \texttt{xmu_make_top_twin}, \texttt{xmu_model_needs_means}, \texttt{xmu_safe_run_summary}, \texttt{xmu_set_sep_from_suffix},
\texttt{xmu_simplex_corner}, \texttt{xmu_start_value_list}, \texttt{xmu_starts}
Examples

```r
require(umx)
data(demoOneFactor)
m2 <- mxModel("One Factor",
  mxMatrix("Full", 5, 1, values = 0.2, free = TRUE, name = "A"),
  mxMatrix("Symm", 1, 1, values = 1.0, free = FALSE, name = "L"),
  mxMatrix("Diag", 5, 5, values = 1.0, 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)
```

Description

This function will label all the free parameters in a RAM mxModel.

Usage

```r
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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make_oneheadedpathsfrompathlist, xmu_dot_make_twoheadedpathsfrompathlist, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

```r
require(umx); data(demoOneFactor)
# raw but no means
m1 <- mxModel("One Factor", mxData(demoOneFactor, type = "raw"), type="RAM", manifestVars = "x1", latentVars= "G",
            umxPath("G", to = "x1"),
            umxPath(var = "x1"),
            umxPath(var = "G", fixedAt = 1)
        )
xmuLabel_RAM_Model(m1)
```

---

**xmuMakeDeviationThresholdsMatrices**

*Make a deviation-based mxRAMObjective for ordinal models.*

**Description**

Purpose: return a mxRAMObjective(A = "A", S = "S", F = "F", M = "M", thresholds = "thresh"), mxData(df, type = "raw") use-case see: umxMakeThresholdMatrix

**Usage**

```r
xmuMakeDeviationThresholdsMatrices(df, droplevels, verbose)
```

**Arguments**

- `df` a dataframe
- `droplevels` whether to droplevels or not
- `verbose` how verbose to be

**Value**

- list of matrices
xmuMakeOneHeadedPathsFromPathList

See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuale, xmu_dot_make_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuale, xmu_dot_make_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts
xmuMakeTwoHeadedPathsFromPathList

Description
Make two-headed paths

Usage
xmuMakeTwoHeadedPathsFromPathList(pathList)

Arguments
pathList A list of paths

Value
- added items

See Also
Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples
xmuMaxLevels(mtcars) # NA = no ordinal vars
xmuMaxLevels(umxFactor(mtcars))
xmuMaxLevels(umxFactor(mtcars), what = "name")

---

Description
A function to compute and report modifications which would improve fit. You will probably use xmuMI 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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts
**Description**

Get the min levels from df

**Usage**

```r
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`, `umx`, `xmuHasSquareBrackets`, `xmuLabel_MATRIX_Model`, `xmuLabel_Matrix`, `xmuLabel_RAM_Model`, `xmuMI`, `xmuMakeDeviationThresholdsMatrices`, `xmuMakeOneHeadedPathsFromPathList`, `xmuMaxLevels`, `xmuPropagateLabels`, `xmu_assemble_twin_supermodel`, `xmu_check_levels_identical`, `xmu_clean_label`, `xmu_dot_make_paths`, `xmu_dot_make_residuals`, `xmu_dot_maker`, `xmu_dot_move_ranks`, `xmu_dot_rank_str`, `xmu_lavaan_process_group`, `xmu_make_mxData`, `xmu_make_top_twin`, `xmu_model_needs_means`, `xmu_safe_run_summary`, `xmu_set_sep_from_suffix`, `xmu_simplex_corner`, `xmu_start_value_list`, `xmu_starts`

**Examples**

```r
xmuMinLevels(mtcars)  # NA = no ordinal vars
xmuMinLevels(umxFactor(mtcars))
xmuMinLevels(umxFactor(mtcars), what = "name")
```

---

**Description**

You should be calling `umxLabel`. This function is called by `xmuLabel_MATRIX_Model`

**Usage**

```r
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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathlist, xmuMakeTwoHeadedPathsFromPathlist, xmuMaxLevels, xmuMinLevels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

```r
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_assemble_twin_supermodel

*Assemble top, MZ and DZ into a supermodel*

Description

Assemble top, MZ and DZ into a supermodel: Also copes with weighted analyses.

Usage

```r
xmu_assemble_twin_supermodel(name, MZ, DZ, top, mzWeightMatrix = NULL,
dzWeightMatrix = NULL)
```
Description

Just checks that the factor levels for twins 1 and 2 are the same

Usage

```r
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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagatelabels, xmu_assemble_twin_supermodel, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

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(umx_factor(tmp, sep = ""), selDVs = baseNames, sep = "")  
## Not run: xmu_check_levels_identical(umx_factor(tmp), selDVs = baseNames, sep = "")
## End(Not run)
```

---

xmu_check_variance  
Check the minimum variance in data frame

**Description**

Check that each variable exceeds a minimum variance and all are on compatible scales. Let the user know what to do if not.

**Usage**

`xmu_check_variance(data, minVar = 0.1, maxVarRatio = 1000)`

**Arguments**

- `data`  
  the data frame to check
- `minVar`  
  Minimum allowed variance in variables before warning user variances differ too much.
- `maxVarRatio`  
  Maximum allowed ratio of variance in data before warning user variances differ too much.

**Value**

-
See Also

Other Miscellaneous Utility Functions: install.OpenMx, qm, umxBrownie, umxFactor, umxVersion, umx_array_shift, umx_cell_is_on, umx_cont_2_quantiles, umx_find_object, umx_make, umx_msg, umx_open_CRAN_page, umx_pad, umx_pb_note, umx_print, umx_scale, umx_score_scale, umx

Examples

data(twinData)
xmu_check_variance(twinData[, c("wt1", "ht1", "wt2", "ht2")])
twinData[,c("ht1", "ht2")]= twinData[,c("ht1", "ht2")]*100
xmu_check_variance(twinData[, c("wt1", "ht1", "wt2", "ht2")])

---

xmu_clean_label  Remove illegal characters from labels

Description

Replaces . with _ in labels - e.g. from lavaan where . is common.

Usage

xmu_clean_label(label, replace = ".")

Arguments

label  A label to clean.
replace  character to replace . with (default = ".")

Value

- legal label string

See Also

- umxLabel

Other xmu internal not for end user: umxModel, umx, umxHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

xmu_clean_label("data.var", replace = ".")
xmu_clean_label("my.var.lab", replace = ".")
xmu_dot_make_paths

Description

Helper to print a digraph to file and open it

Usage

xmu_dot_maker(model, file, digraph, strip_zero = TRUE)

Arguments

model An mxModel to get the name from
file Either "name" (use model name) or a file name
digraph Graphviz code for a model
strip_zero Whether to remove the leading "0." in digits in the diagram

Value

-

See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Other Graphviz: umx_dot_define_shapes, umx_dot_mat2dot, umx_dot_rank, xmu_dot_make_paths, xmu_dot_make_residuals

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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts
Other Graphviz: umx_dot_define_shapes, umx_dot_mat2dot, umx_dot_rank, xmu_dot_make_residuals, xmu_dot_maker

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"
**Description**

*xmu_dot_move_ranks* (not for end users)

**Usage**

\[ xmu_dot_move_ranks(min = \text{NULL}, \text{same} = \text{NULL}, \text{max} = \text{NULL}, \text{old_min}, \text{old_same}, \text{old_max}) \]

**Arguments**

- **min**: vars to group at top of plot
- **same**: vars to group at the same level
- **max**: vars to group at bottom of plot
- **old_min**: vars to group at top of plot
- **old_same**: vars to group at the same level
- **old_max**: vars to group at bottom of plot

**Value**

- list(min=min, same=same, max=max)
xmu_dot_rank_str

See Also

Other xmu internal not for end user: umxModel, umx, umxHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

---

xmu_dot_rank_str (not for end users)

---

Description

xmu_dot_rank_str (not for end users)

Usage

xmu_dot_rank_str(min = NULL, same = NULL, max = NULL)

Arguments

- **min**: vars to group at top of plot
- **same**: vars to group at the same level
- **max**: vars to group at bottom of plot

Value

- GraphViz rank string

See Also

Other xmu internal not for end user: umxModel, umx, umxHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts
xmu_get_CI

Description

Look up CIs for free parameters in a model, and return as APA-formatted text string. If std are available then they are reported.

Usage

xmu_get_CI(model, label, prefix = "top.", suffix = "_std",
            digits = 2, SEstyle = FALSE, verbose = FALSE)

Arguments

model       an mxModel to get CIs from
label       the label of the cell to interrogate for a CI, e.g. "ai_r1c1"
prefix      The submodel to look in (default = "top.")
suffix      The suffix for algebras when standardized (default = "_std")
digits      = 2
SEstyle     If TRUE, report "b(se)" instead of b CI95[l,u] (default = FALSE)
verbose     = FALSE

Value

- the CI string, e.g. ".73[-.20, .98]" or .73(.10)

References


See Also

Other Reporting Functions: loadings, MxModel, tmx_is_identified, tmx_show, umxAPA, umxEval,
umxFactorScores, umxGetParameters, umxReduce, umxSummary, umxWeightedAIC, umx_APA_pval,
umx_aggregate, umx_names, umx_parameters, umx_print, umx_show_fit_or_comparison, umx_time,
umx

Examples

require(umx); data(demoOneFactor)
latents = c("g"); manifests = names(demoOneFactor)
ml = umxRAM("One Factor", data = demoOneFactor, type = "cov",
             umxPath(latents, to = manifests),
             umxPath(var = manifests),
             umxPath(var = latents, fixedAt = 1.0)
# Get CI by parameter label
xmu_get_CI(model= m1, "x1_with_x1")
xmu_get_CI(model= m1, "x1_with_x1", SEstyle=TRUE, digits=3)

# prefix (submodel) and suffix (e.g. std) are ignored if not needed
xmu_get_CI(model =m1, "x1_with_x1", prefix = "top.", suffix = ".std")

## Not run:
xmu_get_CI(fit_IP, label = "ai_r1c1", prefix = "top.", suffix = ".std")
xmu_get_CI(fit_IP, label = "ai_r1c1", prefix = "top.", SEstyle = TRUE, suffix = ".std")

## End(Not run)

---

xmu_lavaan_process_group

**lavaan parameter table rows to model**

### Description

Process a set of lavaan tables rows forming a group (Model). Returns empty arrays if no rows matching the requested group are found.

### Usage

```r
xmu_lavaan_process_group(tab, groupNum)
```

### Arguments

- `tab` a parameter table
- `groupNum` group number to filter table on

### Value

- list(plist=plist, latents = latents, manifests = manifests)

### See Also

- `umxLav2RAM`

Other xmu internal not for end user: `umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts"
xmu_make_mxData

Upgrade a dataframe to an mxData type.

Description

'xmu_make_mxData' is an internal function to upgrade a dataframe to 'mxData'. It can also drop variables from the dataframe.

The most common use will be to give it a dataframe, and get back an 'mxData' object of type raw, cov, cor (WLS is just raw).

Usage

xmu_make_mxdata(data = NULL, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), manifests = NULL, verbose = FALSE)

Arguments

data A data.frame or mxData
type What data type is wanted out c("Auto", "FIML", "cov", "cor", 'WLS', 'DWLS', 'ULS')
manifests If set, only these variables will be retained.
verbose If verbose, report on columns kept and dropped (default FALSE)

Value

- mxData

See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

tab = lavaan::lavaanify("y=x")
xmu_lavaan_process_group(tab, groupNum = 1)
xmu_lavaan_process_group(tab, groupNum = 0)
Examples

# = Continuous ML example =
manVars = c("mpg", "cyl", "disp")

manvars = "mpg" "cyl" "disp"

tmp = xmu_make_mxData(data= mtcars, type = "Auto"); # class(tmp); # "MxDataStatic"

# names(tmp$observed) # "mpg" "cyl" "disp"
tmp = xmu_make_mxData(data= mtcars, type = "Auto", manifests = manVars);
tmp$type "raw" # TRUE

# = All continuous WLS example =
tmp = xmu_make_mxData(data= mtcars, type = "WLS", manifests = manVars, verbose = TRUE)
tmp$type "raw" # TRUE (WLS is triggered by the fit function, not the data type)

# = Missing data WLS example =
tmp = mtcars; tmp[1, "mpg"] = NA # add NA
tmp = xmu_make_mxData(data= tmp, type = "WLS", manifests = manVars, verbose = TRUE)

# = Cov and cor examples =
tmp = xmu_make_mxData(data= mtcars, type = "cov"

tmp = xmu_make_mxData(data= mtcars, type = "cor"

# = Pass string through =
xmu_make_mxData(data= c("a", "b", "c"), type = "Auto")

=================================================================================

xmu_make_top_twin  
Helper to make a basic top, MZ, and DZ model.

Description

xmu_make_top_twin makes basic top, MZ, and DZ models. It includes thresholds matrices in the
twin models if needed.

This is used in umxCP(), umxACE() and umxACEv() and will be added to the other models:  
umxGxE(), umxIP(), simplifying code maintenance.

xmu_make_top_twin takes mZData and dZData, a list of the selDVs to analyse (as well as sep and
nSib), along with other relevant information such as whether the user wants to equateMeans. It
can also handle a weightVar.

varStarts is computed as sqrt(variance)/3 of the DVs and meanStarts as the variable means.
For raw data, a check is made for ordered variables. For Binary variables, means are fixed at 0 and
total variance (A+C+E) is fixed at 1. For ordinal variables, the first 2 thresholds are fixed.
Modeling
top model
For raw and WLS data, top contains a means matrix (if needed). For summary data, the top model contains only a name.
For ordinal data, top gains top.threshMat (from a call to umxThresholdMatrix]).
MZ and DZ models
MZ and DZ contain the data, and an expectation referencing top.expCovMZ and top.expMean, and, vector = bVector. For continuous raw data, MZ and DZ contain mxExpectationNormal] and mxFitFunctionML]. For WLS these the fit function is switched to mxFitFunctionWLS] with appropriate type and allContinuousMethod.
For binary, a constraint and algebras are included to constrain Vtot (A+C+E) to 1.
If a weightVar is detected, this column is added to mzWeightMatrix/mzWeightMatrix.
If equateMeans is TRUE, then the Twin-2 vars in the mean matrix are equated by label with Twin-1.
Matrices created
If needed means matrices are added. Decent starts are guessed from the data. For continuous raw data, top contains a means matrix "expMean". For Models with ordinal, top adds an umxThresholdMatrix].
If binary variables are present, matrices and a constraint to hold A+C+E ==1 are added to top.
If a weight variable is offered up, an mzWeightMatrix will be added.
Data handling
In terms of data handling, xmu_make_top_twin was primarily designed to take data.frames and process these into mxData. It can also, however, handle cov and mxData input.
It can process data into all the types supported by mxData.
Raw data input with a target of cov or cor type requires the numObsMZ and numObsDZ to be set.
Type "WLS", "DWLS", or "ULS", data remain raw, but are handled as WLS in the mxFitFunctionWLS].
Unused columns are dropped.
If you pass in raw data, you can’t request type cov/cor yet. Will work on this if desired.
Usage
xmu_make_top_twin(mzData, dzData, selDVs, sep = NULL, type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"), nSib = 2, numObsMZ = NULL, numObsDZ = NULL, equateMeans = TRUE, weightVar = NULL, bVector = FALSE, verbose = FALSE)
Arguments
mzData Dataframe containing the MZ data
dzData Dataframe containing the DZ data
selDVs List of base (e.g. BMI) (i.e., NOT 'BMI_T1') variable names (OR, you don’t set "sep", the full variable names)
sep

(type)

(optional but desirable) string used to expand selDV into selVars, i.e., ",_T" to
expand BMI into BMI_T1 and BMI_T2

type

One of 'Auto', 'FIML', 'cov', 'cor', 'WLS', 'DWLS', or 'ULS'. Auto tries to react
to the incoming mxData type (raw/cov).

allContinuousMethod

"cumulants" or "marginals". Used in all-continuous WLS data to determine if a
means model needed.

nSib

Number of members per family (default = 2)

numObsMZ

Number of MZ observations contributing (for summary data only)

numObsDZ

Number of DZ observations contributing (for summary data only)

equateMeans

Whether to equate T1 and T2 means (default = TRUE).

weightVar

If provided, a vector objective will be used to weight the data. (default = NULL).

bVector

Whether to compute row-wise likelihoods (defaults to FALSE).

verbose

(default = FALSE)

Value

• mxModels for top, MZ and DZ.

See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model,
xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFrom
xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels,
xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths,
xmu_dot_make_residuals, xmu_dot MakeMaker, xmu_dot_move_ranks, xmu_lavaan_process_group,
xmu_make_mxdata, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix,
xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

# = Continuous =
# = = = = = = = = = = =
library(umx)
data(twinData)
selDVs = c("wt", "ht")
mxData = twinData[, twinData$zygosity %in% "MZFF",]
dzData = twinData[, twinData$zygosity %in% "DZFF",]
bits = xmu_make_top_twin(mxData = mxData, dzData = dzData, selDVs = selDVs, sep = "", nSib = 2)
names(bits) # "top" "MZ" "DZ" "bVector" "mzWeightMatrix" "dzWeightMatrix"
class(bits$MZ$fitfunction)[[1]] == "MxFitFunctionML"
bits = xmu_make_top_twin(mxData = mxData, dzData = dzData, selDVs = selDVs, sep = "", type = "WLS")
class(bits$MZ$fitfunction)[[1]] == "MxFitFunctionWLS"
bits$MZ$fitfunction$type == "WLS"
bits$MZ$fitfunction$continuousType == "cumulants"
# Choose non-default type (DWLS)
bits = xmu_make_top_twin(mzData= mzData, dzData= dzData, selDVs= selDVs, sep= "", type = "DWLS")
bits$M$fitfunction$type"=="DWLS"
class(bits$M$fitfunction)$[]"=="MxFitFunctionWLS"

# Switch continuous method
bits = xmu_make_top_twin(mzData= mzData, dzData= dzData, selDVs= selDVs, sep= "", type = "WLS", allContinuousMethod = "marginals")
bits$M$fitfunction$continuousType"=="marginals"
class(bits$M$fitfunction)$[]"=="MxFitFunctionWLS"

# = Bivariate continuous and ordinal example =
# = 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] = umxFactor(twinData[, ordDVs])
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
bits = xmu_make_top_twin(mzData= mzData, dzData= dzData, selDVs= selDVs, sep= "", nSib= 2)
names(bits) # "top" "MZ" "DZ"

# = One binary =
# = One binary =
data(twinData)
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] = umxFactor(twinData[, ordDVs])
selDVs = c("wt", "obese")
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
bits = xmu_make_top_twin(mzData= mzData, dzData= dzData, selDVs= selDVs, sep= "", nSib= 2)

# = Cov data =
# = Cov data =
data(twinData)
mz = cov(twinData[twinData$zygosity %in% "MZFF", tvars(c("wt", "ht"), sep="")], use = "complete")
dz = cov(twinData[twinData$zygosity %in% "DZFF", tvars(c("wt", "ht"), sep="")], use = "complete")
bits = xmu_make_top_twin(mzData= mzData, dzData= dzData, selDVs= "wt", sep= "", nSib= 2)
class(bit$MZ$fitfunction)[[1]] == "MxFitFunctionML"
names(bit$MZ$data$observed) == c("wt1", "wt2") # height columns dropped

xmu_model_needs_means  Check data to see if model needs means.

Description
Check data to see if model needs means.

Usage
xmu_model_needs_means(data, type = c("Auto", "FIML", "cov", "cor", "WLS", 
"DWLS", "ULS"), allContinuousMethod = c("cumulants", "marginals"))

Arguments
data  mxData to check.
type  of the data requested by the model.
allContinuousMethod  How data will be processed if used for WLS.

Value
- T/F

See Also
- xmu_make_mxData
Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model,
xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList,
xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels,
xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths,
xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group,
xmu_make_mxData, xmu_make_top_twin, xmu_safe_run_summary, xmu_set_sep_from_suffix,
xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples
xmu_model_needs_means(mtcars, type = "Auto")
xmu_model_needs_means(mtcars, type = "FIML")
# xmu_model_needs_means(mtcars, type = "cov")
# xmu_model_needs_means(mtcars, type = "cor")

# TRUE - marginals means means
xmu_model_needs_means(mtcars, type = "WLS", allContinuousMethod= "marginals")
xmu_model_needs_means(mtcars, type = "ULS", allContinuousMethod= "marginals")
xmu_safe_run_summary

Description
The main benefit is that it returns the model, even if it can’t be run.

The function will run the model if requested, wrapped in tryCatch() to avoid throwing an error. If summary = TRUE then umxSummary() is requested (again, wrapped in try).

note: If autoRun is logical, then it over-rides summary to match autoRun. This is useful for easy use umxRAM and twin models.

Usage

xmu_safe_run_summary(model1, model2 = NULL, autoRun = TRUE,
tryHard = c("no", "yes", "mxTryHard", "mxTryHardOrdinal",
"mxTryHardWideSearch"), summary = TRUE, comparison = TRUE)

Arguments

model1 The model to attempt to run and summarize.
model2 Optional second model to compare with model1.
autoRun Whether to run or not (default = TRUE) Options are FALSE and "if needed".
tryHard Default (’no’) uses normal mxRun. "yes" uses mxTryHard. Other options: "mxTryHardOrdinal", "mxTryHardWideSearch"
xmu_set_sep_from_suffix

summary Whether to print model summary (default = autoRun).
comparison Toggle to allow not making comparison, even if second model is provided (more flexible in programming).

Value

• mxModel

See Also

• mxTryHard

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make_paths, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

# xmu_safe_run_summary(model, autoRun = FALSE, summary = TRUE, comparison= FALSE)
# xmu_safe_run_summary(model, model2, autoRun = TRUE, summary = TRUE, comparison= FALSE)
# xmu_safe_run_summary(model, model2, autoRun = TRUE)

xmu_set_sep_from_suffix

Just a helper to cope with deprecated suffix lying around.

Description

Returns either suffix or sep, with a deprecation warning if suffix is set.

Usage

xmu_set_sep_from_suffix(sep, suffix)

Arguments

sep The separator (if suffix != 'deprecated', then this is returned).
suffix The suffix, defaults to 'deprecated'.

Value

- sep
See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_simplex_corner, xmu_start_value_list, xmu_starts

Examples

xmu_set_sep_from_suffix(sep = "_T", suffix = "deprecated")

---

xmu_simplex_corner  
Internal function to help building simplex models

Description

internal function to help building simplex models is a function which

Usage

xmu_simplex_corner(x, start = 0.9)

Arguments

x  
size of matrix, or an umxMatrix of which to free the bottom triangle.

start  
a default start value for the freed items.

Value

- umxMatrix

See Also

- umxMatrix

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_make_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_start_value_list, xmu_starts
xmu_starts

Examples

```r
x = umxMatrix('test', 'Full', nrow = 4, ncol = 4)
xmu_simplex_corner(x, start = .9)
# See how we have a diag free, but offset 1-down?
umx_print(xmu_simplex_corner(x, start = .9)$values, zero="."
```

xmu_starts

Helper providing twin models with boilerplate means and variance start values

Description

xmu_starts can handle several common/boilerplate situations in which means and variance start values are used in twin models.

Usage

```r
xmu_starts(mzData, dzData, selVars = selVars, sep = NULL,
equateMeans = NULL, nSib = 2, varForm = c("Cholesky"), SD = TRUE,
divideBy = 3)
```

Arguments

- mzData: Data for MZ pairs.
- dzData: Data for DZ pairs.
- selVars: Variable names: If sep = NULL, then treated as full names for both sibs.
- sep: All the variables full names.
- equateMeans: (NULL)
- nSib: How many subjects in a family.
- varForm: currently just "Cholesky" style.
- SD: = TRUE (FALSE = variance, not SD).
- divideBy: = 3 (A,C,E) 1/3rd each. Use 1 to do this yourself post-hoc.

Value

- `varstarts` and `meanstarts`

See Also

Other xmu internal not for end user: umxModel, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twinスーパーモデル, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_start_value_list
Examples

data(twinData)
seLDVs = c("wt", "ht")
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
round(sqrt(var(dzData[,tvars(seLDVs, "%" ), na.rm=TRUE])/3),3)
tmp = xmu_starts(mzData, dzData, selVars = seLDVs, sep= "", 
equateMeans = TRUE, varForm = "Cholesky")
tmp
round(var(dzData[,tvars(seLDVs, "%" ), na.rm=TRUE])/3,3)
tmp = xmu_starts(mzData, dzData, selVars = seLDVs, sep= "", 
equateMeans = TRUE, varForm = "Cholesky", SD=FALSE)
tmp
# one variable

tmp = xmu_starts(mzData, dzData, selVars = "wt", sep= "", 
equateMeans = TRUE, varForm = "Cholesky", SD= FALSE)

# Examples data

xmu_start_value_list

Make start values

Description

Purpose: Create start values 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, umx, xmuHasSquareBrackets, xmuLabel_MATRIX_Model, xmuLabel_Matrix, xmuLabel_RAM_Model, xmuMI, xmuMakeDeviationThresholdsMatrices, xmuMakeOneHeadedPathsFromPathList, xmuMakeTwoHeadedPathsFromPathList, xmuMaxLevels, xmuMinLevels, xmuPropagateLabels, xmu_assemble_twin_supermodel, xmu_check_levels_identical, xmu_clean_label, xmu_dot_make_paths, xmu_dot_make_residuals, xmu_dot_maker, xmu_dot_move_ranks, xmu_dot_rank_str, xmu_lavaan_process_group, xmu_make_mxData, xmu_make_top_twin, xmu_model_needs_means, xmu_safe_run_summary, xmu_set_sep_from_suffix, xmu_simplex_corner, xmu_starts

Check basic aspects of input for twin models.

Description

Check that DVs are in the data, that the data have rows, set the optimizer if requested.

Usage

xmu_twin_check(selDVs, dzData = dzData, mzData = mzData, sep = NULL,
enforceSep = TRUE, nSib = 2, numObsMZ = NULL, numObsDZ = NULL,
optimizer = NULL)

Arguments

selDVs Variables used in the data.
dzData The DZ twin data.
mzData The MZ twin data.
sep Separator between base-name and numeric suffix when creating variable names,
e.g. "_T"
enforceSep Whether to require sep to be set, or just warn if it is not (Default = TRUE: enforce).
nSib How many people per family? (Default = 2).
numObsMZ set if data are not raw.
numObsDZ set if data are not raw.
optimizer Set by name (if you want to change it).

Value

-

References


See Also

Other Twin Modeling Functions: umxACE_cov_fixed, umxACEcov, umxACEold, umxACEv, umxACE,
umxCPold, umxCP, umxGxE_window, umxGxEbiv, umxGxE, umxIPold, umxIP, umxSexLim, umxSimplex,
umxSummaryACEcov, umxSummaryACEv, umxSummaryACE, umxSummaryCP, umxSummaryGxEbiv, umxSummaryGxE,
umxSummaryIP, umxSummarySexLim, umxSummarySimplex

Other Check or test: umx_check_names, umx_is_class, umx_is_numeric, umx
Examples

```r
library(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "MZFF")
xmu_twin_check(se1DV = c("wt", "ht"), dzData = dzData, mzData = mzData,
               sep = "", enforceSep = TRUE)
xmu_twin_check(se1DV = c("wt", "ht"), dzData = dzData, mzData = mzData,
               sep = "", enforceSep = FALSE)
xmu_twin_check(se1DV = c("wt", "ht"), dzData = dzData, mzData = mzData,
               sep = "", enforceSep = TRUE, nSib = 2, optimizer = NULL)
```

```
## Not run:
# TODO xmu_twin_check: move to a test file:
# 1. stop on no rows
xmu_twin_check("Generativity", twinData=NULL, twinData=NULL, sep="_T")
# Error in xmu_twin_check("Generativity", twinData=NULL, twinData=NULL, :
#  Your DZ dataset has no rows!

# 2. Stop on a NULL sep = NULL IFF enforceSep = TRUE
xmu_twin_check(se1DV = c("wt", "ht"), dzData = dzData, mzData = mzData, enforceSep = TRUE)
# 3. stop on a factor with sep = NULL

## End(Not run)
```
Index

*Topic **datasets**
  Fischbein_wt, 9
  GFF, 10
  iqdat, 13
  us_skinfold_data, 308

aggregate, **195, 196**
AIC, **8, 194**

colMeans, **197**
confint, **63, 67**
cov, **305**
cov2cor, **68, 69, 209**
cumsum, **197**

data.frame, **107, 199, 225, 237, 244, 328**
DiagrammerR, **16, 17**
dl_from_dropbox, **7, 30, 248, 254, 258, 272, 307**

extractAIC.MxModel, **8, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298**

factanal, **14, 15, 82, 83**
factor, **94**
Fischbein_wt, **9, 12, 14, 30, 309**
formula, **273**

GFF, **9, 10, 14, 30, 309**
grep, **225, 255, 256**

install.OpenMx, **12, 19, 30, 61, 95, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321**

iqdat, **9, 12, 13, 30, 309**

lm, **54, 59, 147**
loadings, **8, 14, 14, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298**

loadings.MxModel, **14, 15, 27–29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326**

logLik, **8**

match.arg, **210**
matrix, **269**
merge, **240**
mxAlgebra, **58, 301**
mxCheckIdentification, **26, 27, 168, 191**
mxCI, **62, 67, 156**
mxCompare, **66, 81, 191**
mxData, **69, 146, 156, 328, 333**
mxEvalByName, **32**
mxFactor, **94, 95, 206, 207**
mxFactorScores, **83, 96**
mxFitFunctionMultigroup, **184, 185**
mxFitFunctionWLS, **78, 79**
mxMatrix, **114, 115, 122, 192, 235, 240, 241, 310, 311**

mxMI, **123, 124**

mxPath, **32, 115, 131**
mxRefModels, **31**
mxRun, **67, 126, 156**
mxTryHard, **32, 335**

names, **255**
namez, **263, 271**

341
namez (umx_names), 255
omxAssignFirstParameters, 157
omxAugmentDataWithWLSSummary, 79
omxBrownie, 61
omxGetParameters, 99, 100
omxSetParameters, 115
OpenMx::mxAlgebra(), 120
OpenMx::mxConstraint(), 120
packageVersion, 194
parameters, 148, 261
parameters (umx_parameters), 260
paste0, 255
plot, 16, 18, 32, 77, 102, 104, 111, 113, 133, 134, 137–139, 141–143, 145, 148, 165, 175, 177, 178, 180, 213
plot (plot.MxModel), 17
plot.MxLISRELModel, 16, 18, 29, 58, 122, 126, 131, 133, 134, 136–139, 141–143, 145, 148, 157, 185
plot.MxModel, 17, 18, 29, 58, 122, 126, 131, 133, 134, 136–139, 141–143, 145, 148, 157, 185
plot.MxModelACE, 45, 171
plot.MxModelACE (umxPlotACE), 132
plot.MxModelACEcov (umxPlotACEcov), 134
plot.MxModelACEv (umxPlotACEv), 135
plot.MxModelCP (umxPlotCP), 136
plot.MxModelGxE (umxPlotGxE), 139
plot.MxModelGxEbiv (umxPlotGxEbiv), 140
plot.MxModelIP (umxPlotIP), 141
plot.MxModelSimplex (umxPlotSimplex), 144
qm, 13, 19, 30, 61, 95, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
regex, 261
regular expressions, 270
reliability, 20, 30, 69, 108, 198, 208, 222, 252, 276, 306
replacement, 255
residuals, 21
residuals.MxModel, 8, 15, 21, 22–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
RMSEA, 8, 15, 21, 22, 23, 24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
RMSEA.MxModel, 8, 15, 21, 22, 23, 24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
RMSEA.summary.Mxmodel, 8, 15, 21–23, 24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
round, 197
rowSums, 197
stack, 292
subset, 302
summary, 168
summaryAPA (umxAPA), 58
tmx_genotypic_effect, 25, 27, 29
tmx_is_identified, 15, 26, 26, 28, 29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326
tmx_show, 15, 27, 27, 29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326
tryCatch(), 334
tsls, 189, 190
tvars (umx_paste_names), 262
umx-deprecated, 31
umx-package (umx), 29
umx2ord (umx_cont_2_quantiles), 206
umx_APA_pval, 15, 27–29, 32, 60, 86, 96, 100, 153, 167, 194, 196, 196, 256, 261, 268, 291, 304, 326
umx_apply, 20, 30, 69, 108, 196, 197, 208, 222, 252, 276, 306
umx_array_shift, 13, 19, 30, 61, 95, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
umx_cell_is_on, 13, 19, 30, 61, 95, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
umx_check, 201, 202, 204–206, 226–229, 231–238
umx_check_model, 201, 202, 204–206, 226–229, 231–238
umx_check_names, 30, 201, 202, 203, 205, 206, 226–238, 256, 271, 339
umx_check_OS, 201, 202, 204, 206, 226–229, 231–238
umx_check_parallel, 201, 202, 204, 205, 205, 226–229, 231–238
umx_checkpoint (umx_set_checkpoint), 282
umx_cont_2_quantiles, 13, 19, 30, 61, 95, 194, 198, 200, 206, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
umx_cor, 20, 30, 32, 69, 108, 198, 208, 222, 252, 276, 306
umx_default_option, 30, 210, 223, 224, 280–290
umx_dot_define_shapes, 211, 213, 215, 322–324
umx_dot_mat2dot, 211, 212, 215, 322–324
umx_dot_rank, 211, 213, 214, 322–324
umx_drop_ok, 8, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umx_explode, 30, 216, 217, 225, 256, 257, 263, 275, 305, 307
umx_explode_twin_names, 30, 216, 217, 225, 256, 257, 263, 275, 305, 307
umx_factor (umxFactor), 94
umx_find_object, 13, 19, 30, 61, 95, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
umx_fix_first-loadings, 30, 114, 115, 118, 152, 187, 192, 219, 220, 222, 302
umx_fix_latents, 30, 114, 115, 118, 152, 187, 192, 219, 220, 222, 302
umx_fun_mean_sd, 20, 30, 69, 108, 198, 208, 221, 221, 252, 276, 306
umx_get_bracket_addresses, 30, 114, 115, 118, 152, 187, 192, 219, 220, 222, 302
umx_get_checkpoint, 30, 210, 223, 224, 280–290
umx_get_options, 30, 210, 223, 224, 280–290
umx_grep, 30, 32, 216, 217, 225, 256, 257, 263, 275, 305, 307
umx_has been-run, 201, 202, 204–206, 226, 227–229, 231–238
umx_has_CIs, 201, 202, 204–206, 226, 227, 228, 229, 231–238
umx_has_means, 201, 202, 204–206, 226, 227, 228, 229, 231–238
umx_has_square_brackets, 32, 201, 202, 204–206, 226–228, 229, 231–238
umx_is_class, 30, 204, 230, 336, 339
umx_is_cov, 201, 202, 204–206, 226–229, 231, 232–238
umx_is_endogenous, 201, 202, 204–206, 226–229, 231, 232, 233–238
umx_is_MxData, 201, 202, 204–206, 226–229, 231–233, 234, 235–238
umx_is_MxMatrix, 201, 202, 204–206, 226–229, 231–234, 234, 236–238
umx_is_MxModel, 201, 202, 204–206, 226–229, 231–235, 235, 237, 238
umx_is_numeric, 30, 204, 230, 236, 339
umx_is_ordered, 201, 202, 204–206, 226–229, 231–236, 237, 238
umx_is_RAM, 201, 202, 204–206, 226–229, 231–237, 238
umx_long2wide, 30, 239, 250, 274, 278, 306
umx_make, 13, 19, 30, 61, 95, 194, 198, 200,
umx_set_cores, 30, 210, 223, 224, 280–283, 284, 285–290
umx_set_data_variance_check, 30, 210, 223, 224, 280–284, 285, 286–290
umx_set_optimization_options, 30, 210, 223, 224, 280–285, 286, 287–290
umx_set_optimizer, 30, 210, 223, 224, 280–286, 287, 288–290
umx_set_plot_file_suffix, 30, 210, 223, 224, 280–287, 288, 289, 290
umx_set_plot_format, 17, 18, 30, 210, 223, 224, 280–288, 289, 290
umx_set_table_format, 30, 153, 155, 210, 223, 224, 267, 268, 280–289, 290
umx_show_fit_or_comparison, 15, 27–29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 303, 326
umx_standardize, 293
umx_standardize_ACE, 30, 253, 293, 294–297, 299–301
umx_standardize_ACEcov, 30, 253, 294, 294, 295–297, 299–301
umx_standardize_ACEv, 30, 253, 294, 295, 296, 297, 299–301
umx_standardize_CP, 30, 253, 294, 295, 296, 297, 299–301
umx_standardize_IP, 30, 253, 294–296, 297, 299–301
umx_standardize_RAM, 8, 15, 21–24, 32, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umx_standardize_Sexlim, 30, 253, 294–297, 299, 300, 301
umx_standardize_Simplex, 30, 253, 294–297, 299, 300, 301
umx_stash_CIs, 30, 253, 294–297, 299, 300, 301
umx_string_to_algebra, 30, 32, 114, 115, 118, 152, 187, 192, 219, 220, 222, 301
umx_time, 15, 27–29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 303, 326
umx_trim, 30, 210, 217, 225, 256, 257, 263, 275, 304, 307
umx_update_OpenMx (install.OpenMx), 12
umx_var, 20, 30, 69, 108, 198, 208, 222, 252, 276, 305
umx_wide2long, 30, 240, 250, 274, 278, 306
umx_write_to_clipboard, 7, 30, 216, 217, 225, 248, 254, 256–258, 263, 272, 275, 303, 307
umxACE(), 329
umxACE_cov_fixed, 29, 36, 41, 45, 51, 54, 73, 77, 102, 104, 106, 111, 113, 160, 165, 171, 172, 174, 175, 177, 178, 180, 181, 184, 339
umxACEv(). 329
umxAdd1, 29, 56, 81, 85, 99, 124, 126, 158, 191
umxAlgebra, 17, 18, 29, 57, 122, 126, 131, 148, 157, 185
umxAPA, 15, 27–29, 58, 58, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326
umxCI, 8, 15, 21–24, 32, 62, 63, 64, 66, 67, 93, 94, 97, 136, 156, 169, 171, 174, 181, 184, 215, 298
umxCI_boot, 8, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umxCompare, 8, 15, 21–24, 32, 63, 64, 65, 67, 85, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umxConfint, 8, 15, 21–24, 63, 64, 66, 69, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298, 301
umxCov2cor, 20, 30, 68, 108, 198, 208, 222, 252, 276, 306
umxCovP(). 329
umxDiagnose, 79
umxDrop1, 29, 57, 81, 85, 99, 124, 126, 158, 191
umxEFA, 29, 82, 120, 151, 190
umxEquate, 29, 32, 57, 81, 84, 99, 115, 124, 126, 158, 191
umxEval, 15, 27–29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326
umxExampleCode_TRHpaper (umxExamples), 87
umxExamples, 87
umxExpCov, 8, 15, 21–24, 63, 64, 66, 67, 92, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umxExpMeans, 8, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umxFactanal, 95
umxFactanal (umxEFA), 82
umxFactor, 13, 19, 30, 61, 94, 194, 198, 200, 207, 218, 243, 254, 259, 260, 264, 268, 277, 279, 321
umxFactorScores, 15, 27–29, 60, 86, 96, 100, 153, 167, 194, 196, 197, 256, 261, 268, 291, 304, 326
umxFitIndices, 8, 15, 21–24, 63, 64, 66, 67, 93, 94, 97, 136, 169, 171, 174, 181, 184, 215, 298
umxFixAll, 29, 57, 81, 85, 98, 124, 126, 158, 191
umxGxE(). 329
umxGxE_window, 29, 36, 41, 45, 51, 56, 73, 77, 102, 104, 105, 111, 113, 160, 165, 171, 172, 174, 175, 177, 178, 180, 181, 184, 339
umxIP(). 329
umxJiggle, 30, 32, 114, 115, 118, 152, 187, 192, 219, 220, 222, 302
umxLatent, 30, 32, 114, 115, 116, 152, 187, 192, 219, 220, 222, 302
umxLatentRAM, 29, 83, 119, 151, 190, 327
umxMatrix, 17, 18, 29, 58, 121, 126, 131, 148, 157, 185, 212, 336
umxMI, 29, 57, 81, 85, 99, 123, 126, 158, 191, 316
umxModel, 30, 124, 309, 311, 313–325, 327,
INDEX

328, 331, 333, 335–338
umxModify, 17, 18, 29, 32, 36, 45, 57, 58, 63, 81, 85, 99, 122, 124, 125, 131, 148,
157, 158, 171, 185, 191, 192
umxPadAndPruneDefVars, 30, 70, 79, 108,
127, 199, 209, 241, 244–246, 250,
265–267, 269, 271, 273, 292, 302
umxParameters (umx_parameters), 260
umxPath, 17, 18, 29, 32, 58, 120, 122, 126,
129, 147, 148, 157, 185
umxPath(), 145
umxPlot (plot.MxModel), 17
umxPlotACE, 17, 18, 29, 32, 132, 134,
136–139, 141–143, 145,
153–155, 167
umxPlotACEcov, 17, 18, 29, 30, 73, 133, 134, 136,
136, 138, 139, 141–143, 145,
153–155, 167
umxPlotCPold, 17, 18, 29, 30, 133, 134, 136,
137, 138, 139, 141–143, 145,
153–155, 167
umxPlotGxE, 17, 18, 29, 133, 134, 136–138,
139, 141–143, 145
umxPlotGxEbiv, 17, 18, 29, 133, 134,
136–139, 140, 142, 143, 145
umxPlotIP, 17, 18, 29, 133, 134, 136–139,
141, 141, 143, 145
umxPlotSexLim, 17, 18, 29, 133, 134,
136–139, 141, 142, 145
umxPlotSimplex, 17, 18, 29, 133, 134,
136–139, 141–143, 144
umxRAM, 17, 18, 29, 58, 66, 120, 122, 125, 126,
131, 145, 152, 157, 185, 190
umxRAM2, 29, 83, 120, 150, 190
umxRAM2Ordinal, 30, 114, 115, 118, 151, 187,
192, 219, 220, 222, 302
umxReduce, 15, 27–30, 60, 86, 96, 100, 102,
104, 137, 138, 152, 154, 155, 167,
194, 196, 197, 256, 261, 268, 291,
304, 326
umxReduceACE, 30, 137, 138, 153, 154, 155,
167
umxReduceGxE, 30, 137, 138, 153, 154, 155,
INDEX

xmu_model_needs_means, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335–338
xmu_safe_run_summary, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 334, 336–338
xmu_set_sep_from_suffix, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335, 335, 336–338
xmu_simplex_corner, 30, 122, 125, 309, 311, 313–325, 327, 328, 331, 333, 335, 336, 337, 338
xmu_start_value_list, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335–337, 338
xmu_starts, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335, 336, 337, 338
xmuHasSquareBrackets, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335–338
xmuLabel_Matrix, 30, 125, 309, 310, 311, 313–325, 327, 328, 331, 333, 335–338
xmuLabel_MATRIX_Model, 30, 125, 309, 311, 313–325, 327, 328, 331, 333, 335–338
xmuLabel_RAM_Model, 30, 125, 309, 311, 312, 314–325, 327, 328, 331, 333, 335–338
xmuMakeDeviationThresholdsMatrices, 30, 125, 309, 311, 313, 314–325, 327, 328, 331, 333, 335–338
xmuMakeOneHeadedPathsFromPathList, 30, 125, 309, 311, 313, 314, 315–325, 327, 328, 331, 333, 335–338
xmuMakeTwoHeadedPathsFromPathList, 30, 125, 309, 311, 313, 314, 315, 316–325, 327, 328, 331, 333, 335–338
xmuMaxLevels, 30, 125, 309, 311, 313–315, 315, 316–325, 327, 328, 331, 333, 335–338
xmuMI, 30, 125, 309, 311, 313–316, 316, 317–325, 327, 328, 331, 333, 335–338
xmuMinLevels, 30, 125, 309, 311, 313–316, 317, 318–325, 327, 328, 331, 333, 335–338
xmuPropagateLabels, 30, 125, 309, 311, 313–317, 317, 319–325, 327, 328, 331, 333, 335–338