Package 'umx'

October 13, 2026

Version 4.80.1

Date 2026-10-15

Title Structural Equation Modeling 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#readme

Description Quickly create, run, and report structural equation and twin models.
See '?umx' for help, and umx_open_CRAN_page(``umx") for NEWS.
Citations are:
* Timothy C. Bates, Michael C. Neale, Hermine H. Maes, (2019). umx: A library for Structural Equation and Twin Modelling in R. Twin Research and Human Genetics, 22, 27-41. <doi:10.1017/thg.2019.2>.
* Luis F.S. Castro-de-Araujo, Nathan A. Gillespie, Michael C. Neale, and Timothy Charles Bates (2026). umx version 4.5: Extending Twin and Path-Based SEM in R with CLPM, MR-DoC, Definition Variables, Onyx Integration, and Censored Distributions. Twin Research and Human Genetics, 1-6. <doi:10.1017/thg.2026.10056>.

Depends R (>= 4.1.0), OpenMx (>= 2.20.0),

Imports cowplot, DiagrammeR, gert, ggplot2, kableExtra, knitr, lavaan, MASS, Matrix, methods, mvtnorm, nlme, openxlsx, paran, polycor, quantmod, R2HTML, RCurl, scales, utils, xtable, zoo

Suggests remotes, devtools, pak, spelling, testthat, withr, R.utils, data.table, car, cocor, GPArotation, httr2, parallel, psychTools, psych, pwr, piggyback, rmarkdown, rhub, ssh

Enhances DiagrammeRsvg, rsvg

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

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Timothy C. Bates [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1153-9007>),
Luis De Araujo [ctb],
Nathan Gillespie [wit],
Hermine Maes [ctb],
Michael C. Neale [ctb],
Joshua N. Pritikin [ctb],
Brenton Wiernik [ctb],
Michael Zakharin [wit]

Repository CRAN

Date/Publication 2026-10-13 06:40:02 UTC

Contents

aic_weights . . . 11
Anthro_LDSC . . . 12
bucks . . . 13
calculateStrictSb . . . 14
deg2rad . . . 15
dl_from_dropbox . . . 16
docData . . . 17
extractAIC.MxModel . . . 18
fin_options_Greeks . . . 19
fin_options_LeapSimulate . . . 20
fin_options_plotGreeks . . . 21
fin_options_teach . . . 22
fin_resolveVolatility . . . 23
fin_stock_CAGR . . . 24
fin_stock_justifiedPE . . . 25
fin_stock_target . . . 27
fin_stock_ticker . . . 28
fin_stock_valuation . . . 29
fin_tax_FIF . . . 30
fin_tax_NI . . . 31
fin_value_CAGR . . . 32
fin_value_CarryCost . . . 33
fin_value_interest . . . 35
fin_value_NPV . . . 37
fin_value_percent . . . 38
Fischbein_wt . . . 39
FishersMethod . . . 40
geometric_mean . . . 41
GFF . . . 43
ggAddR . . . 45
gh_feature_branch . . . 46
gh_message_search . . . 47

gh_open_PR_url . . . 48
harmonic_mean . . . 50
HSwls . . . 51
install.OpenMx . . . 53
iqdat . . . 55
libs . . . 56
loadings . . . 57
loadings.MxModel . . . 58
mx_make . . . 59
noNAs . . . 61
oddsratio . . . 62
plot.MxLISRELModel . . . 63
plot.MxModel . . . 65
plot.MxModelTwinMaker . . . 68
plot.percent . . . 70
plot.umx_GWAS . . . 71
power.ACE.test . . . 72
print.oddsratio . . . 76
print.percent . . . 77
print.reliability . . . 78
print.RMSEA . . . 78
prolific_anonymize . . . 79
prolific_check_ID . . . 80
prolific_read_demog . . . 81
prolific_scoring_stub . . . 83
Psych_LDSC . . . 84
qm . . . 85
rad2deg . . . 86
reliability . . . 87
residuals.MxModel . . . 88
RMSEA . . . 89
RMSEA.MxModel . . . 90
RMSEA.summary.mxmodel . . . 91
SE_from_p . . . 92
tmx_genotypic_effect . . . 93
tmx_is.identified . . . 95
tmx_show . . . 96
tmx_show.MxMatrix . . . 97
tmx_show.MxModel . . . 99
tmx_show.MxModelGSEM . . . 101
tvars . . . 101
umx . . . 103
umxACE . . . 107
umxACEcov . . . 115
umxACEv . . . 118
umxACE_DE . . . 124
umxACE_GLM . . . 130
umxAlgebra . . . 133

umxAPA . . . 134
umxBrownie . . . 138
umxCI . . . 138
umxCI_boot . . . 141
umxCloud . . . 142
umxCLPM . . . 144
umxCompare . . . 146
umxConfint . . . 150
umxCov2cor . . . 152
umxCP . . . 153
umxDiagnose . . . 158
umxDiffMZ . . . 160
umxDiscTwin . . . 162
umxDoC . . . 164
umxDoCp . . . 167
umxEFA . . . 169
umxEquate . . . 173
umxExamples . . . 175
umxExpCov . . . 180
umxExpMeans . . . 181
umxFactor . . . 182
umxFactorScores . . . 184
umxFitIndices . . . 185
umxFixAll . . . 187
umxGetLatents . . . 188
umxGetManifests . . . 189
umxGetModel . . . 190
umxGetParameters . . . 191
umxGSEM . . . 193
umxGSEM_dl_tutorial_files . . . 196
umxGSEM_GWAS . . . 197
umxGSEM_label_ldsc . . . 200
umxGSEM_ldsc . . . 201
umxGSEM_munge . . . 203
umxGSEM_sim_snps . . . 205
umxGSEM_std . . . 206
umxGSEM_sumstats . . . 207
umxGxE . . . 210
umxGxEbiv . . . 214
umxGxE_window . . . 216
umxHetCor . . . 219
umxIP . . . 220
umxJiggle . . . 224
umxLav2RAM . . . 225
umxLISREL . . . 229
umxMatrix . . . 232
umxMatrixFree . . . 234
umxMeasurementInvariance . . . 235

umxMI . . . 236
umxModel . . . 238
umxModelNames . . . 239
umxModify . . . 240
umxMRDoC . . . 243
umxParameters . . . 245
umxParan . . . 247
umxPath . . . 248
umxPlot . . . 252
umxPlotACE . . . 254
umxPlotACEcov . . . 255
umxPlotACEv . . . 257
umxPlotACE_DE . . . 258
umxPlotACE_GLM . . . 259
umxPlotCP . . . 260
umxPlotDoC . . . 262
umxPlotFun . . . 264
umxPlotGxE . . . 266
umxPlotGxEbiv . . . 267
umxPlotIP . . . 269
umxPlotPredict . . . 270
umxPlotSexLim . . . 271
umxPlotSimplex . . . 273
umxPower . . . 275
umxPrint_double_entry_data . . . 278
umxRAM . . . 278
umxRAM2Lav . . . 286
umxRAM_DE . . . 287
umxRAM_GLM . . . 290
umxReduce . . . 292
umxReduceACE . . . 293
umxReduceGxE . . . 295
umxRenameMatrix . . . 296
umxRotate . . . 297
umxRotate.MxModelCP . . . 298
umxRun . . . 300
umxSetParameters . . . 301
umxSexLim . . . 303
umxSimplex . . . 308
umxSummarizeTwinData . . . 311
umxSummary . . . 313
umxSummary.list . . . 313
umxSummary.MxDataStatic . . . 315
umxSummary.MxLISRELModel . . . 316
umxSummary.MxModel . . . 317
umxSummary.MxModelGLM . . . 321
umxSummaryACE . . . 322
umxSummaryACEcov . . . 324

umxSummaryACEv . . . 326
umxSummaryACE_DE . . . 327
umxSummaryACE_GLM . . . 329
umxSummaryCP . . . 330
umxSummaryDoC . . . 331
umxSummaryGxE . . . 333
umxSummaryGxEbiv . . . 335
umxSummaryIP . . . 337
umxSummaryMRDoC . . . 338
umxSummarySexLim . . . 339
umxSummarySimplex . . . 342
umxSuperModel . . . 343
umxThresholdMatrix . . . 346
umxTwinMaker . . . 349
umxTwoStage . . . 352
umxUnexplainedCausalNexus . . . 354
umxVersion . . . 355
umxWeightedAIC . . . 356
umx_aggregate . . . 357
umx_APA_pval . . . 359
umx_apply . . . 360
umx_array_shift . . . 361
umx_as_numeric . . . 362
umx_check . . . 363
umx_check_model . . . 364
umx_check_names . . . 365
umx_check_OS . . . 367
umx_check_parallel . . . 368
umx_complete_dollar . . . 369
umx_cont_2_quantiles . . . 370
umx_cor . . . 371
umx_data . . . 373
umx_detect_cores . . . 373
umx_explode . . . 374
umx_explode_twin_names . . . 375
umx_file_load_pseudo . . . 376
umx_find_object . . . 377
umx_fun_mean_sd . . . 378
umx_get_bracket_addresses . . . 379
umx_get_checkpoint . . . 381
umx_get_options . . . 382
umx_grep . . . 382
umx_has_been_run . . . 384
umx_has_CIs . . . 385
umx_has_means . . . 386
umx_has_square_brackets . . . 387
umx_is_class . . . 388
umx_is_cov . . . 389

umx_is_endogenous . . . 390
umx_is_exogenous . . . 391
umx_is_GSEM . . . 392
umx_is_LISREL . . . 393
umx_is_MxData . . . 394
umx_is_MxMatrix . . . 394
umx_is_MxModel . . . 395
umx_is_numeric . . . 396
umx_is_ordered . . . 397
umx_is_RAM . . . 398
umx_log_wide_twin_data . . . 399
umx_long2wide . . . 400
umx_lower.tri . . . 402
umx_lower2full . . . 403
umx_make . . . 405
umx_make_double_entry_data . . . 407
umx_make_fake_data . . . 409
umx_make_GxE_data . . . 411
umx_make_MR_data . . . 413
umx_make_raw_from_cov . . . 414
umx_make_sql_from_excel . . . 415
umx_make_TwinData . . . 417
umx_make_twin_data_nice . . . 421
umx_means . . . 422
umx_merge_randomized_columns . . . 423
umx_move_file . . . 424
umx_msg . . . 425
umx_names . . . 427
umx_open . . . 429
umx_open_CRAN_page . . . 430
umx_pad . . . 431
umx_polychoric . . . 432
umx_polypairwise . . . 433
umx_polytriowise . . . 434
umx_print . . . 435
umx_read_lower . . . 437
umx_rename . . . 439
umx_rename_file . . . 440
umx_reorder . . . 442
umx_residualize . . . 443
umx_rot . . . 444
umx_round . . . 445
umx_r_test . . . 446
umx_scale . . . 447
umx_scale_reliabilities . . . 448
umx_scale_wide_twin_data . . . 449
umx_scholastica_report . . . 450
umx_score_scale . . . 453

umx_select_valid . . . 457
umx_set_auto_plot . . . 458
umx_set_auto_run . . . 459
umx_set_checkpoint . . . 459
umx_set_condensed_slots . . . 461
umx_set_cores . . . 462
umx_set_data_variance_check . . . 463
umx_set_dollar_symbol . . . 464
umx_set_optimization_options . . . 465
umx_set_optimizer . . . 466
umx_set_plot_file_suffix . . . 467
umx_set_plot_format . . . 468
umx_set_separator . . . 469
umx_set_silent . . . 470
umx_set_table_format . . . 471
umx_stack . . . 472
umx_standardize . . . 473
umx_strings2numeric . . . 474
umx_string_to_algebra . . . 475
umx_str_chars . . . 476
umx_str_from_object . . . 477
umx_tail . . . 478
umx_time . . . 479
umx_trim . . . 481
umx_tutorials . . . 481
umx_var . . . 482
umx_wide2long . . . 484
umx_wide2longTwinData . . . 486
umx_wide4lmer . . . 487
umx_write_to_clipboard . . . 488
umx_yj_wide_twin_data . . . 489
us_skinfold_data . . . 490
xgh_check_base_name . . . 491
xgh_get_current_branch . . . 492
xmuCalculateSRMR . . . 493
xmuHasSquareBrackets . . . 493
xmuLabel . . . 494
xmuLabel_Matrix . . . 496
xmuLabel_MATRIX_Model . . . 498
xmuLabel_RAM_Model . . . 500
xmuMakeOneHeadedPathsFromPathList . . . 501
xmuMakeTwoHeadedPathsFromPathList . . . 503
xmuMaxLevels . . . 504
xmuMI . . . 505
xmuMinLevels . . . 506
xmuPropagateLabels . . . 507
xmuRAM2Ordinal . . . 509
xmuTwinSuper_Continuous . . . 510

xmuTwinSuper_NoBinary . . . 512
xmuTwinUpgradeMeansToCovariateModel . . . 514
xmuValues . . . 516
xmu_ace_de_apply_censor_thresholds . . . 517
xmu_ace_de_parse_censor_meta . . . 519
xmu_ace_de_parse_censor_rule . . . 520
xmu_ace_de_release_binary_v1 . . . 521
xmu_bracket_address2rclabel . . . 522
xmu_catml_df3_diagnostic . . . 524
xmu_catml_discrepancy_at_WLS . . . 525
xmu_catml_eval_model . . . 526
xmu_catml_implied_correlation . . . 528
xmu_catml_wls_v . . . 529
xmu_cell_is_on . . . 530
xmu_check_levels_identical . . . 532
xmu_check_needs_means . . . 533
xmu_check_variance . . . 535
xmu_CI_merge . . . 536
xmu_CI_stash . . . 538
xmu_clean_label . . . 539
xmu_compare_constraint_note . . . 540
xmu_compare_robust_ML . . . 542
xmu_compare_WLS . . . 543
xmu_custom_completer . . . 544
xmu_data_missing . . . 544
xmu_data_swap_a_block . . . 546
xmu_describe_data_WLS . . . 547
xmu_DF_to_mxData_TypeCov . . . 549
xmu_dot_define_shapes . . . 550
xmu_dot_maker . . . 551
xmu_dot_make_paths . . . 552
xmu_dot_make_residuals . . . 554
xmu_dot_mat2dot . . . 556
xmu_dot_move_ranks . . . 559
xmu_dot_rank . . . 560
xmu_dot_rank_str . . . 561
xmu_equate_threshold_values . . . 562
xmu_extract_column . . . 563
xmu_get_CI . . . 564
xmu_gsem_expand_snp . . . 566
xmu_gsem_extract_snp_path . . . 567
xmu_gsem_subset_covstruc . . . 568
xmu_has_summary_mxData . . . 569
xmu_invert_matrix . . . 570
xmu_is_ordinal_WLS . . . 571
xmu_is_wls . . . 572
xmu_lavaan_process_group . . . 573
xmu_make_bin_cont_pair_data . . . 574

xmu_make_mxData . . . 576
xmu_make_TwinSuperModel . . . 578
xmu_match.arg . . . 583
xmu_mxData_summary . . . 585
xmu_mxRefModels . . . 586
xmu_name_from_lavaan_str . . . 587
xmu_openmx_engine_status . . . 589
xmu_openmx_install_message . . . 589
xmu_PadAndPruneForDefVars . . . 590
xmu_parse_completion_context . . . 591
xmu_path2twin . . . 592
xmu_path_regex . . . 593
xmu_print_algebras . . . 595
xmu_ram_de_apply_censor_thresholds . . . 596
xmu_ram_de_parse_censor_meta . . . 598
xmu_rclabel_2_bracket_address . . . 599
xmu_relevel_factors . . . 600
xmu_require_summary_mxData . . . 602
xmu_robust_ML_fit . . . 603
xmu_robust_WLS_fit . . . 605
xmu_safe_run_summary . . . 607
xmu_savalei_fit_cfi . . . 610
xmu_savalei_fit_rmsea . . . 611
xmu_savalei_fit_tli . . . 612
xmu_savalei_null_scaling_factor . . . 614
xmu_savalei_polycor_blocks . . . 615
xmu_savalei_scaling_factor . . . 617
xmu_set_sep_from_suffix . . . 619
xmu_show_fit_or_comparison . . . 620
xmu_simplex_corner . . . 621
xmu_standardize_ACE . . . 622
xmu_standardize_ACEcov . . . 624
xmu_standardize_ACEv . . . 625
xmu_standardize_CP . . . 627
xmu_standardize_IP . . . 628
xmu_standardize_LISREL . . . 629
xmu_standardize_RAM . . . 630
xmu_standardize_SexLim . . . 632
xmu_standardize_Simplex . . . 633
xmu_starts . . . 635
xmu_start_value_list . . . 637
xmu_summary_RAM_group_parameters . . . 638
xmu_threshold_id_RAM . . . 640
xmu_threshold_id_twin_check . . . 641
xmu_topo_sort . . . 642
xmu_twin_add_WeightMatrices . . . 644
xmu_twin_check . . . 645
xmu_twin_get_var_names . . . 647

xmu_twin_make_def_means_mats_and_alg 649
xmu_twin_upgrade_selDvs2SelVars 650
xmu_update_covar 652
xmu_WLS_align_jacobian 653
xmu_WLS_align_weight 654
xmu_wls_display_chi 655
xmu_wls_extract_WV 657
xmu_WLS_polycor_names 658

Index **660**

`aic_weights` *Akaike weights for a set of models*

Description

`aic_weights` converts a vector of AIC values into Akaike weights: the conditional probability that each model is the best in the set, given the data and the candidate models (Wagenmakers and Farrell, 2004).

Usage

```
aic_weights(x)
```

Arguments

`x`	Numeric vector of AIC values, one per model. `NA` values are not dropped.

Details

Weights are computed from AIC differences relative to the best model in the set:

$$w_i = \frac{\exp(-\Delta_i/2)}{\sum_j \exp(-\Delta_j/2)}, \quad \Delta_i = \mathrm{AIC}_i - \min(\mathrm{AIC})$$

The best model has $\Delta = 0$ and therefore the largest weight. Weights sum to 1. A model 2 AIC units worse than the best has a likelihood ratio of $\exp(-1) \approx 0.37$ relative to it, before normalisation across the set. The result is a statement about this candidate set only, not about the absolute probability that a model is true.

Value

Numeric vector of weights, the same length as `x`, summing to 1.

References

Wagenmakers, E. J., and Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, **11**, 192-196. doi:10.3758/BF03206482

See Also

umxWeightedAIC(), AIC()

Other Reporting Functions: plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
aic_weights(c(100, 102, 110))
# best model, 2 AIC worse, 10 AIC worse
```

Anthro_LDSC *Anthropometric Genomic SEM LDSC dataset.*

Description

A dataset containing the genetic covariance structure from LD score regression (LDSC) for 9 anthropometric phenotypes: BMI, WHR, CO, Waist, Hip, Height, IHC, BL, and BW. Same list layout as Psych_LDSC - pass the object as covstruc to umxGSEM() or umxGSEM_GWAS().

Usage

```
data(Anthro_LDSC)
```

Format

A list with elements S, V, I, N, and m (see Details).

Details

LDSC list fields (same S/V/I/N/m layout as GenomicSEM ldsc; V **names are OpenMx residual labels**):

- S: 9 x 9 genetic covariance matrix among the phenotypes (diagonals are SNP heritabilities on the LDSC scale).
- V: 45 x 45 sampling covariance of the unique elements of S (OpenMx vech order; dimnames like var_BMI, poly_WHR_BMI). Used as asymptotic covariance for WLS/DWLS. Not GenomicSEM BMI_BMI / WHR_BMI.
- I: 9 x 9 **LDSC intercept** matrix. Diagonal = per-trait LDSC intercepts (expected near 1 without confounding; elevations can flag stratification). Off-diagonal = cross-trait intercepts reflecting sample overlap. Used for genomic control when expanding SNPs in umxGSEM_GWAS().
- N: 1 x 45 effective sample sizes for each unique S element.
- m: Number of SNPs used in LDSC (here 1,173,569).

See Also

Psych_LDSC, umxGSEM(), umxGSEM_GWAS(), umxGSEM_ldsc()

Other GSEM: Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Other datasets: Fischbein_wt, GFF, HSwls, Psych_LDSC, docData, iqdat, umx, us_skinfold_data

Examples

```
## Not run:
data(Anthro_LDSC)
str(Anthro_LDSC, max.level = 1)
names(Anthro_LDSC) # S, V, I, N, m
# I = LDSC intercepts (diag) and cross-trait intercepts (off-diag)
Anthro_LDSC$I[1:3, 1:3]

## End(Not run)
```

---

bucks *Print a money object*

Description

Print function for "money" objects, e.g. fin_value_interest().

Usage

```
bucks(
  x,
  symbol = umx_set_dollar_symbol(silent = TRUE),
  big.mark = ",",
  decimal.mark = ".",
  trim = TRUE,
  largest_with_cents = 1e+05,
  negative_parens = c("hyphen", "minus", "parens"),
  ...
)
```

Arguments

x	money object.
symbol	Default prefix if not set.
big.mark	option defaulting to ","
decimal.mark	option defaulting to "."
trim	option defaulting to TRUE

largest_with_cents
: option defaulting to 1e+05

negative_parens
: option defaulting to "hyphen"

... : further arguments passed to or from other methods. also cat =F to return string

Value

- invisible

See Also

- fin_value_percent(), fin_value_interest(), scales::dollar()

Examples

```
bucks(100 * 1.05^32)
fin_value_interest(deposits = 20e3, interest = 0.07, yrs = 20)
```

---

calculateStrictSb *calculateStrictSb*

Description

calculateStrictSb calculates the Satorra-Bentler (2010) scaled difference chi-square test (the "strictly positive" design relative to SB-2001, which can yield a negative scaling constant c_d). The returned statistic is only interpretable when both models share the same WLS moments, weight matrix, and N, and when the freer model's optimized discrepancy is not larger than the nested model's ($F_{nested} \geq F_{base}$). Non-monotone F (typically failed optimization) returns NA with a warning.

Usage

```
calculateStrictSb(baseModel, nestedModel)
```

Arguments

baseModel : The base model (more parameters)

nestedModel : The nested model (fewer parameters)

Value

A named vector containing strictSbChisq, deltaDf, scalingFactor, and pValue. May carry attribute nonMonotoneF = TRUE when $\Delta F < 0$.

deg2rad *Convert Degrees to Degrees*

Description

A helper to convert degrees (360 in a circle) to Rad (2π in a circle).

note: R's trig functions, e.g. `sin()` use Radians for input!

The formula is radians = $degx180/\pi$.

- 180 Degrees is equal to π radians.
- 1 Rad = $180/\pi$ degrees = ~ 57.296 degrees.

Usage

```
deg2rad(deg)
```

Arguments

deg	The value in degrees you wish to convert to radians

Value

- value in radians

References

https://en.wikipedia.org/wiki/Radian

See Also

- `rad2deg()`, `sin()`

Other Miscellaneous Functions: `rad2deg()`, `umxBrownie()`, `umx_check_parallel()`, `umx_complete_dollar()`

Examples

```
deg2rad(180) == pi # TRUE!
```

dl_from_dropbox *dl_from_dropbox*

Description

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

Usage

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

Arguments

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

Details

Improvements would include error handling...

Value

None

See Also

Other File Functions: umx, umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_move_file(), umx_open(), umx_rename_file(), umx_write_to_clipboard()

Examples

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

## End(Not run)
```

docData *Twin data for Direction of causation modelling*

Description

A dataset containing indicators for two traits varA and varB, each measured in MZ and DZ twins.

Usage

```
data(docData)
```

Format

A data frame 6 manifests for each of two twins in 1400 families of MZ and DZ twins

Details

It is designed to show off umxDoC() testing the hypothesis varA causes varB, varB causes varA, both cause each other.

- *zygosity* "MZFF", "DZFF", "MZMM", or "DZMM"
- *varA1_T1* Twin one's manifest 1 for varA
- *varA2_T1* Twin one's manifest 2 for varA
- *varA3_T1* Twin one's manifest 3 for varA
- *varB1_T1* Twin one's manifest 1 for varB
- *varB2_T1* Twin one's manifest 2 for varB
- *varB3_T1* Twin one's manifest 3 for varB
- *varA1_T2* Twin two's manifest 1 for varA
- *varA2_T2* Twin two's manifest 2 for varA
- *varA3_T2* Twin two's manifest 3 for varA
- *varB1_T2* Twin two's manifest 1 for varB
- *varB2_T2* Twin two's manifest 2 for varB
- *varB3_T2* Twin two's manifest 3 for varB

References

N.A. Gillespie and N.G. Martin (2005). Direction of Causation Models. In *Encyclopedia of Statistics in Behavioral Science*, **1**, 496-499. Eds. Brian S. Everitt & David C. Howell

See Also

- umxDoC(), plot.MxModelDoC(), umxSummary.MxModelDoC(), umxModify()

Other datasets: Anthro_LDSC, Fischbein_wt, GFF, HSwls, Psych_LDSC, iqdat, umx, us_skinfold_data

Examples

```
data(docData)
str(docData)
mzData = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData = subset(docData, zygosity %in% c("DZFF", "DZMM"))
par(mfrow = c(1, 2))  # 1 rows and 3 columns
plot(varA1_T2 ~varA1_T1, ylim = c(-4, 4), data = mzData, main="MZ")
tmp = round(cor.test(~varA1_T1 + varA1_T2, data = mzData)$estimate, 2)
text(x=-4, y=3, labels = paste0("r = ", tmp))
plot(varA1_T2 ~varA1_T1, ylim = c(-4, 4), data = dzData, main="DZ")
tmp = round(cor.test(~varA1_T1 + varA1_T2, data = dzData)$estimate, 2)
text(x=-4, y=3, labels = paste0("r = ", tmp))
par(mfrow = c(1, 1))  # back to as it was
```

---

extractAIC.MxModel *Extract AIC from MxModel*

Description

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

Usage

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

Arguments

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

Value

- AIC value

See Also

- AIC(), umxCompare(), logLik()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
extractAIC(m1)
# -2.615998
AIC(m1)

## End(Not run)
```

---

fin_options_Greeks *Black-Scholes Call Option Price and Greeks Calculator*

Description

`fin_options_Greeks` calculates the theoretical European Call option price and its corresponding Greeks (Delta, Gamma, Theta, Vega, and Rho) using the Black-Scholes model.

Usage

```
fin_options_Greeks(
  spotPrice,
  strikePrice,
  daysToExpiry,
  riskFreeRate = 0.04,
  impliedVol = 0.2
)
```

Arguments

`spotPrice`	The current price of the underlying asset.
`strikePrice`	The strike price of the option.
`daysToExpiry`	The number of days remaining until option expiration.
`riskFreeRate`	The annual risk-free interest rate (expressed as a decimal, default = 0.04 for 4%).
`impliedVol`	The annualized implied volatility (expressed as a decimal, default = 0.20 for 20%).

Value

A data frame containing:

- `price`: Theoretical Call option price
- `delta`: Sensitivity of option price to underlying price (Delta)
- `gamma`: Sensitivity of Delta to underlying price (Gamma)
- `theta`: Daily time decay of option price (Theta)
- `vega`: Sensitivity of option price to a 1% change in implied volatility
- `rho`: Sensitivity of option price to a 1% change in risk-free interest rate

See Also

Other financial functions: `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
fin_options_Greeks(spotPrice = 100, strikePrice = 95, daysToExpiry = 30,
riskFreeRate = 0.04, impliedVol = 0.20)
```

---

fin_options_LeapSimulate

Simulate and Compare LEAP Extrinsic Premium Decay

Description

`fin_options_LeapSimulate` compares two European Call options over a 931-day horizon: one starting at a 0.80 Delta and another at a 0.95 Delta. It simulates how their extrinsic value (rent/time-decay premium) bleeds to 0 as time runs out, plotting the results side-by-side.

Usage

```
fin_options_LeapSimulate(
  spotPrice = 100,
  impliedVol = 0.2,
  riskFreeRate = 0.04
)
```

Arguments

`spotPrice`	The constant price of the underlying asset (default = 100).
`impliedVol`	The constant implied volatility of the underlying asset (default = 0.20).
`riskFreeRate`	The annual risk-free interest rate (default = 0.04).

Value

A ggplot object comparing the extrinsic premium decay side-by-side.

See Also

Other financial functions: `fin_options_Greeks()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
## Not run:
fin_options_LeapSimulate(spotPrice = 100, impliedVol = 0.20, riskFreeRate = 0.04)

## End(Not run)
```

---

fin_options_plotGreeks

Plot Option Delta and Gamma Curves

Description

`fin_options_plotGreeks` plots Call Option Delta and Gamma curves across a range of spot prices (from -30% to +30% of the strike price) to show how Delta accelerates and Gamma peaks.

Usage

```
fin_options_plotGreeks(
  strikePrice,
  daysToExpiry = 30,
  riskFreeRate = 0.04,
  impliedVol = 0.2
)
```

Arguments

`strikePrice`	The strike price of the option (default = 100 if omitted).
`daysToExpiry`	The number of days remaining until option expiration (default = 30).
`riskFreeRate`	The annual risk-free interest rate (default = 0.04).
`impliedVol`	The annualized implied volatility of the underlying asset (default = 0.20).

Value

A ggplot object visualizing Delta and Gamma.

See Also

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_NPV(), fin_value_interest(), fin_value_percent()

Examples

```
## Not run:
fin_options_plotGreeks(strikePrice = 100)
# Run with missing arguments to print definitions
fin_options_plotGreeks()

## End(Not run)
```

---

fin_options_teach *Teaching function for options*

Description

fin_options_teach is a teaching function for understanding options (intrinsic/extrinsic value, annualized rent, leverage, and time decay).

Usage

```
fin_options_teach(
  premium = 134,
  strike = 200,
  stock = 304,
  delta = 0.85,
  years = 1.8,
  type = c("call", "put")
)
```

Arguments

premium	Cost to buy the option contract per share.
strike	The strike price of the option.
stock	The current stock price.
delta	The delta of the option (default = 0.85). Used to calculate effective leverage (Omega).
years	How far in time the option ends (e.g. 1.8 years for a LEAP).
type	Whether it is a "call" or "put" option.

Value

A list containing intrinsic value, extrinsic value, break-even price, annualized rent percent, effective leverage (Omega), and daily theta.

See Also

- `fin_value_interest()`, `fin_tax_NI()`, `fin_value_percent()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
# Call Option (In-The-Money LEAP)
fin_options_teach(premium = 134, strike = 200, stock = 304, delta = 0.85, years = 1.8)

# Put Option (Out-Of-The-Money)
fin_options_teach(premium = 10, strike = 280, stock = 304, delta = -0.30, years = 0.5, type = "put")
```

`fin_resolveVolatility` *Resolve Option Implied Volatility from Ticker*

Description

`fin_resolveVolatility` looks up the implied volatility of a ticker from its option chain closest to the target expiry and strike. If the retrieved IV is stale/zero (< 1%), it falls back to calculating the 90-day Historical Volatility (HV) of the stock.

Usage

```
fin_resolveVolatility(impliedVol, strikePrice, daysToExpiry)
```

Arguments

`impliedVol`	Either a numeric value (returned directly) or a character stock symbol.
`strikePrice`	The strike price of the option.
`daysToExpiry`	The number of days to expiration.

Value

A numeric volatility value.

`fin_stock_CAGR` *Compute the CAGR of a stock*

Description

`fin_stock_CAGR` uses stock info from Yahoo to work out the CAGR over time.

Usage

```
fin_stock_CAGR(priceSeries, from = "1900-01-01")
```

Arguments

`priceSeries`	A price series using yahoo
`from`	The date in the series to start from (blank = all)

Value

- value

See Also

- `fin_value_interest()`, `fin_tax_NI()`, `fin_value_percent()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
## Not run:
libs(c("quantmod", "ggplot2", "scales", "lubridate"))
getSymbols(c("NVDA"), from = "2010-01-01", to = Sys.Date())
startDate = "2016-01-01"
nvdaCagr = fin_stock_CAGR(NVDA, startDate)

## End(Not run)
```

`fin_stock_justifiedPE` *Justified P/E Ratio*

Description

Compute the Justified Price-to-Earnings (P/E) ratio for a stock using a single-stage or two-stage (high-growth + ROIC) Gordon Growth Model.

Usage

```
fin_stock_justifiedPE(
  dividend = 0.8,
  EPS = 2,
  growthRate = 0.06,
  discountRate = 0.1,
  buybackYield = 0,
  ROIC = NULL,
  yearsHighGrowth = 0,
  terminalGrowth = 0.03,
  type = c("trailing", "leading")
)
```

Arguments

`dividend`	Dividend per share (D_0 for trailing, D_1 for leading). Default = 0.80. Ignored if `ROIC` is specified.
`EPS`	Earnings per share (E_0 for trailing, E_1 for leading). Default = 2.00.
`growthRate`	Sustainable growth rate (g), as a decimal (e.g. 0.20 for 20%). Default = 0.06.
`discountRate`	Required rate of return / cost of equity (r), as a decimal (e.g. 0.10 for 10%). Default = 0.10.
`buybackYield`	Net share repurchase yield added to payout ratio, as a decimal (e.g. 0.015 for 1.5%). Default = 0.
`ROIC`	Return on Invested Capital, as a decimal (e.g. 0.30 for 30%). If specified, payout ratio during high growth is derived as $1 - g/\text{ROIC}$.
`yearsHighGrowth`	Duration of high-growth phase in years (default = 0 for single-stage model). If $g \geq r$ and `yearsHighGrowth == 0`, defaults to 5 years.
`terminalGrowth`	Long-term terminal growth rate after high-growth phase, as a decimal (e.g. 0.03 for 3%). Default = 0.03.
`type`	Character string indicating `"trailing"` (default) or `"leading"` P/E.

Details

Justified P/E represents the fundamental value multiple that a stock should trade at based on its payout ratio (or ROIC reinvestment), expected growth rate (g), and required rate of return / discount rate (r).

Single-Stage Model (when `yearsHighGrowth = 0`): For stable-growth companies where $r > g$:

$$\text{Justified Trailing P/E} = \frac{\text{Payout Ratio} \times (1+g)}{r-g}$$

$$\text{Justified Leading P/E} = \frac{\text{Payout Ratio}}{r-g}$$

Two-Stage Model (when `yearsHighGrowth > 0`): For growth companies where high-growth rate g_{high} may exceed the cost of capital (r), payout ratio during high growth is derived from Return on Invested Capital (ROIC):

$$\text{Payout Ratio} = 1 - \frac{g}{\text{ROIC}} + \text{buybackYield}$$

Cash flows during the `yearsHighGrowth` period are discounted at rate r. After `yearsHighGrowth`, growth transitions to `terminalGrowth` ($g_{\text{term}} < r$), and the terminal value is discounted back to present value.

Typical Values & Interpretation: Typical justified P/E ratios for mature, stable-growth companies generally fall between 12 and 25. High-growth companies with high ROIC (e.g. 30%+ ROIC and 20% growth) justify 20x-30x+ P/E multiples.

Worked Example ($NVDA): Suppose Nvidia ($NVDA) has high growth of 20% ($g = 0.20$) for 5 years, high ROIC of 30% (ROIC $= 0.30$), cost of equity of 10% ($r = 0.10$), and long-term terminal growth of 3% ($g_{\text{term}} = 0.03$). The justified P/E is calculated as: `fin_stock_justifiedPE(EPS = 2.70, growthRate = 0.20, discountRate = 0.10, ROIC = 0.30, yearsHighGrowth = 5, terminalGrowth = 0.03)`

Value

Numeric justified P/E ratio.

References

- Gordon, M. J. (1962). *The Investment, Financing, and Valuation of the Corporation*. R. D. Irwin.
- Pinto, J. E., Henry, C., Robinson, T. R., & Stowe, J. D. (2020). *Equity Asset Valuation* (4th ed.). Wiley.
- Mauboussin, M. J., & Rappaport, A. (2021). *Expectations Investing: Reading Stock Prices for Better Returns*. Columbia University Press.

See Also

- `fin_value_interest()`, `fin_value_percent()`, `fin_tax_NI()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
# Example 1: Standard trailing Justified P/E
fin_stock_justifiedPE(dividend = 0.8, EPS = 2.0, growthRate = 0.06, discountRate = 0.10)

# Example 2: Two-Stage growth model for 20% growth company with 30% ROIC for 5 years
fin_stock_justifiedPE(EPS = 2.70, growthRate = 0.20, discountRate = 0.10,
                      ROIC = 0.30, yearsHighGrowth = 5, terminalGrowth = 0.03)
```

`fin_stock_target` *Compute the future value and gain of an investment*

Description

fin_stock_target takes a current and fair value, as well as a cost of capital, and returns the expected gain.

Usage

```
fin_stock_target(
  current = 89,
  fair = 140,
  ticker = "NVDA",
  capital = 0.15,
  verb = FALSE
)
```

Arguments

`current`	The current market value of the instrument
`fair`	The user's estimated fair value.
`ticker`	A label for printing
`capital`	The cost of capital (defaults to .15)
`verb`	Verbose or concise (FALSE)

Value

- expected gain

See Also

- `fin_value_interest()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
fin_stock_target(114,fair=140, ticker="NVDA", capital=.15, verb=TRUE)
# NVDA  return =  41 %
# delta (fair-current)= $ 26
# growth = $ 21
# expected gain = $ 47
# future value (final) = $ 161

fin_stock_target(24, 130, ticker="SMMT")
# SMMT  return =  523 %

fin_stock_target(24, 75, ticker="SMMT", verb=TRUE)
# SMMT  return =  259 %
# delta (fair-current)= $ 51
# growth = $ 11.25
# expected gain = $ 62.25
# future value (final) = $ 86.25

fin_stock_target(750, 1000, ticker="LLY", verb=TRUE)
# LLY  return =  53 %
# delta (fair-current)= $ 250
# growth = $ 150
# expected gain = $ 400
# future value (final) = $ 1150
```

`fin_stock_ticker` *Open a ticker in yahoo finance.*

Description

Open a stock ticker, currently in yahoo finance

Usage

```
fin_stock_ticker(
  ticker = "NVDA",
  exchange = "NASDAQ",
  provider = c("GOOGLE", "YAHOO")
)
```

Arguments

`ticker`	A stock symbol to look up, e.g., "OXY"
`exchange`	Stock exchange code (default = "NASDAQ").
`provider`	Financial data provider site ("GOOGLE" or "YAHOO", default = "GOOGLE").

Value

- Open a ticker in a finance site online

See Also

- fin_value_interest(), fin_value_percent(), fin_tax_NI()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_NPV(), fin_value_interest(), fin_value_percent()

Examples

```
# Open $NVDA in google, MRVL in yahoo finance.
## Not run:
fin_stock_ticker("NVDA")
fin_stock_ticker("MRVL", provider= "YAHOO")

## End(Not run)
```

---

fin_stock_valuation *Work the valuation of a company*

Description

fin_stock_valuation uses the revenue, operating margin, expenses and PE to compute a market capitalization. Better to use a more powerful online site.

Usage

```
fin_stock_valuation(
  revenue = 6e+06 * 30000,
  opmargin = 0.08,
  expenses = 0.2,
  PE = 30,
  symbol = "$",
  use = c("B", "M")
)
```

Arguments

revenue	Revenue of the company
opmargin	Margin on operating revenue
expenses	Additional fixed costs
PE	of the company
symbol	Currency
use	reporting values in "B" (billion) or "M" (millions)

Details

Revenue is multiplied by opmargin to get a gross profit. From this the proportion specified in `expenses` is subtracted and the resulting earnings turned into a price via the `PE`

Value

- value

See Also

- `fin_value_interest()`, `fin_tax_NI()`, `fin_value_percent()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
fin_stock_valuation(rev=7e9, opmargin=.1, PE=33)
# Market cap =  $18,480,000,000
# (Based on PE= 33, operating Income of $0.70 B, and net income =$0.56B
```

---

fin_tax_FIF *NZ FIF tax offset: NAV-neutral leverage (FDR on the whole pile)*

Description

Fair Dividend Rate tax is applied to **opening equity plus assets bought with loan funds**.

Usage

```
fin_tax_FIF(
  portfolioValue = 1e+06,
  marginRate = 0.056,
  expectedReturn = 0.12,
  taxRate = 0.38,
  fifRate = 0.05
)
```

Arguments

`portfolioValue`	Opening FIF value (1 April), before the new loan.
`marginRate`	IBKR (or other) annual margin rate (e.g. 0.05).
`expectedReturn`	Expected annual return of the asset (e.g. 0.12).
`taxRate`	Marginal tax rate (e.g. 0.38).
`fifRate`	FDR deemed rate (default 0.05).

Details

Neutral loan / opening equity:

$$L/E = (ft)/(r - i(1-t) - ft)$$

When $f = i$ (both 5%), the leftover on the loan collapses to $r - i$: interest deduction pays FDR on the borrowed slice, and the remaining spread pays FDR on the original book.

$$z = \frac{\bar{x} - \mu}{\sigma/\sqrt{n}}$$

LTV on the plot is loan / opening equity, not loan / total assets.

Value

A ggplot of net annual impact vs LTV. Invisibly, a list with `loan`, `assets`, `ltvOpening`, `fdrDrag`, `leftoverOnLoan`.

See Also

`fin_value_interest()`, `fin_tax_NI()`, `fin_value_percent()`

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_NI()`, `fin_value_CAGR()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
# $3.8m opening, 5% IBKR, 12% expected, 38% tax -> ~$1.03m loan, ~$4.83m assets
fin_tax_FIF(portfolioValue = 3.8e6, marginRate = 0.05, expectedReturn = 0.12, taxRate = 0.38)
```

`fin_tax_NI` *Compute UK NI given annual Earnings.*

Description

Employees pay contributions at 12%% on annual earnings between GBP 9,568 and GBP 50,270. Above that you pay at 2%%. Employers pay at 13.8%% on all annual earnings of more than GBP 8,840, although there are different thresholds for those under the age of 21 and for apprentices under the age of 25.

Usage

```
fin_tax_NI(annualEarnings, symbol = "£")
```

Arguments

annualEarnings	Employee annual earnings.
symbol	Currency symbol to embed in the result.

Value

- NI

See Also

- fin_value_interest(), fin_value_percent(), fin_stock_valuation()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_NPV(), fin_value_interest(), fin_value_percent()

Examples

```
fin_tax_NI(42e3)
fin_tax_NI(142000)
```

---

fin_value_CAGR *Calculate Compound Annual Growth Rate (CAGR)*

Description

Calculates the constant, period-over-period growth rate required for an investment to grow from a beginning value to an ending value over a specified number of periods.

The Compound Annual Growth Rate (CAGR) is computed as: CAGR = (End value/Start value)^(1/t) - 1

Where *t* is the number of years (periods).

Usage

```
fin_value_CAGR(beginningValue, endingValue, numYears, digits = 3)
```

Arguments

beginningValue	Starting value of investment
endingValue	Ending value of investment
numYears	Number of periods (e.g., years) elapsing from begin to end
digits	rounding the returned value (default = 3)

Value

A numeric value representing the Compound Annual Growth Rate as a decimal (e.g., 0.096 for 9.6%).

Note

This function includes input validation and will `stop()` with an error if any inputs are non-numeric or non-positive.

See Also

Other financial functions: `fin_options_Greeks()`, `fin_options_LeapSimulate()`, `fin_options_plotGreeks()`, `fin_options_teach()`, `fin_stock_CAGR()`, `fin_stock_justifiedPE()`, `fin_stock_target()`, `fin_stock_ticker()`, `fin_stock_valuation()`, `fin_tax_FIF()`, `fin_tax_NI()`, `fin_value_CarryCost()`, `fin_value_NPV()`, `fin_value_interest()`, `fin_value_percent()`

Examples

```
rate = fin_value_CAGR(beginningValue = 100, endingValue = 190, numYears = 7)
print(rate)

# --- Example with a Loss ---
fin_value_CAGR(100, 50, 5)

# --- Formatting as Percentage ---
percent = paste0(round(rate * 100, 2), "%")
print(percent)
```

---

fin_value_CarryCost *Work the carry cost of a house*

Description

`fin_value_CarryCost` uses the purchase price, holding expenses, appreciation, and opportunity cost to compute a carrying cost for a house purchase.

Usage

```
fin_value_CarryCost(
  property_cost,
  appreciation = 0.02,
  QQQ = 0.14,
  rent_saved = 0.04,
  interest = 0.06,
  rates = 5000,
  insurance = 2000,
  maintenance = 0.015,
  years = 5,
```

```
  inflation = 0.025,
  verbose = TRUE
)
```

Arguments

property_cost	Purchase price
appreciation	rate of property increase
QQQ	Opportunity cost of leaving money in the markets
rent_saved	But now you have to rent somewhere
interest	Cost of borrowing
rates	Council rates per year at t=1 (absolute).
insurance	The cost of property owners insurance per year at t=1 (absolute).
maintenance	New kitchen roof etc. If <1, treated as rate of property_cost at t=1; if >=1, treated as absolute per year at t=1.
years	Holding time (integer >=1).
inflation	Annual inflation applied to rent, rates, insurance and maintenance (default .025). Set 0 to recover flat model. Property and QQQ remain compound totals.
verbose	Logical; if TRUE, print a one-line per-year schedule when years <= 20.

Value

Invisibly the total net cost of buying (scalar). When verbose is TRUE, also returns a schedule data.frame as attribute "schedule".

See Also

- fin_value_interest(), fin_tax_NI(), fin_value_percent()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_NPV(), fin_value_interest(), fin_value_percent()

Examples

```
fin_value_CarryCost(property_cost=1.2e6)
fin_value_CarryCost(property_cost=1.1e6, appreciation = .035, QQQ=.15, years=10)
fin_value_CarryCost(property_cost=1.2e6, inflation=0) # flat, recovers pre-inflation total
```

`fin_value_interest` *Compute the value of a principal & annual deposits at a compound interest over a number of years*

Description

Allows you to determine the final value of an initial `principal` (with optional periodic `deposits`), over a number of years (`yrs`) at a given rate of `interest`. Principal and deposits are optional. You control compounding periods each year (n) and whether deposits occur at the beginning or end of the year. The function outputs a nice table of annual returns, formats the total using a user-settable currency `symbol`. Can also `report` using a web table.

notes: Graham valuation: fair P/E = 9 + (1.5 * growth%). e.g. $INTEL fair P/E = 9+.5*3 = 10.5 up to 9+2*10 = 29 Can move the weighting between a conservative .5 and an optimistic 2 (in terms of how long the growth will last and how low the hurdle rate is)

Usage

```
fin_value_interest(
  principal = 100,
  deposits = 0,
  inflate = 0,
  interest = 0.05,
  yrs = 10,
  final = NULL,
  n = 12,
  when = "beginning",
  symbol = NULL,
  largest_with_cents = 0,
  baseYear = as.numeric(format(Sys.time(), "%Y")),
  table = TRUE,
  report = c("markdown", "html"),
  deflate = TRUE
)
```

Arguments

`principal`	The initial investment at time 0 (default 100)
`deposits`	Optional periodic additional investment each *year*.
`inflate`	How much to inflate deposits over time (default 0)
`interest`	Annual interest rate (default .05)
`yrs`	Duration of the investment (default 10).
`final`	if set (default = NULL), returns the rate required to turn principal into final after yrs (principal defaults to 1)
`n`	Compounding intervals per year (default 12 (monthly), use 365 for daily)

when	Deposits made at the "beginning" (of each year) or "end"
symbol	Currency symbol to embed in the result.
largest_with_cents	Default = 0
baseYear	Default = current year (for table row labels)
table	Whether to print a table of annual returns (default TRUE)
report	"markdown" or "html",
deflate	Final capital is inflation adjusted when inflation is non zero (default TRUE).

Value

- Value of balance after yrs of investment.

References

https://en.wikipedia.org/wiki/Compound_interest

See Also

- umx_set_dollar_symbol(), fin_value_percent(), fin_tax_NI(), fin_stock_valuation()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_NPV(), fin_value_percent()

Examples

```
# 1. Value of a principal after yrs years at 5% return, compounding monthly.
# Report in browser as a nice table of annual returns and formatted totals.
fin_value_interest(principal = 5000, interest = 0.05, rep= "html")

# Report as a nice markdown table
fin_value_interest(principal = 5000, interest = 0.05, yrs = 10)

umx_set_dollar_symbol("$")
# 2 What rate is needed to increase principal to final value in yrs time?
fin_value_interest(1, final = 1.4, yrs=5)
fin_value_interest(principal = 50, final=200, yrs = 5)

# 3. What's the value of deposits of $100/yr after 10 years at 7% return?
fin_value_interest(0, deposits = 100, interest = 0.07, yrs = 10, n = 12)

# 4. What's the value of $20k + $100/yr over 10 years at 7% return?
fin_value_interest(principal= 20e3, deposits= 100, interest= .07, yrs= 10, symbol="$")

# 5. What is $10,000 invested at the end of each year for 5 years at 6%?
fin_value_interest(deposits = 10e3, interest = 0.06, yrs = 5, n=1, when= "end")

# 6. What will $20k be worth after 10 years at 15% annually (n=1)?
```

```
fin_value_interest(deposits=20e3, interest = 0.15, yrs = 10, n=1, baseYear=1)
# $466,986

# manual equivalent
sum(20e3*(1.15^(10:1))) # 466985.5

# 7. Annual (rather than monthly) compounding (n=1)
fin_value_interest(deposits = 100, interest = 0.07, yrs = 10, n=1)

# 8 Interest needed to increase principal to final value in yrs time.
fin_value_interest(principal = 100, final=200, yrs = 5)
```

`fin_value_NPV` *Compute the net present value of a future income stream.*

Description

`fin_stock_valuation` uses the revenue, operating margin, expenses and PE to compute a market capitalization. Better to use a more powerful online site.

Usage

```
fin_value_NPV(
  income = 27000,
  discount_rate = 0.05,
  periods = 25,
  symbol = umx_set_dollar_symbol(silent = TRUE)
)
```

Arguments

`income`	Value of expected recurring payment
`discount_rate`	Percent return to discount against (.05 = 5%)
`periods`	How many periods the stream delivers, e.g., 25 years of pension.
`symbol`	Currency symbol to use

Details

Revenue stream is discounted back to a present day cash amount which is equivalent.

Value

- value

See Also

- fin_value_interest(), fin_tax_NI(), fin_value_percent()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_interest(), fin_value_percent()

Examples

```
fin_value_NPV(27e3, .05, 25)
```

---

fin_value_percent *Compute the percent change needed to return to the original value after percent off (or on).*

Description

Determine the percent change needed to "undo" an initial percent change. Has a plot function as well. If an amount of $100 has 20% added, what percent do we need to drop it by to return to the original value?

`fin_value_percent(20)` yields $100 increased by 20% = $120 (Percent to reverse = -17%)

Usage

```
fin_value_percent(
  percent,
  value = 100,
  symbol = "$",
  digits = 2,
  plot = TRUE,
  logY = TRUE
)
```

Arguments

percent	Change in percent (enter 10 for 10%, not 0.1)
value	Principal
symbol	value units (default = "$")
digits	Rounding of results (default 2 places)
plot	Whether to plot the result (default TRUE)
logY	Whether to plot y axis as log (TRUE)

Value

- new value and change required to return to baseline.

See Also

- fin_value_interest()

Other financial functions: fin_options_Greeks(), fin_options_LeapSimulate(), fin_options_plotGreeks(), fin_options_teach(), fin_stock_CAGR(), fin_stock_justifiedPE(), fin_stock_target(), fin_stock_ticker(), fin_stock_valuation(), fin_tax_FIF(), fin_tax_NI(), fin_value_CAGR(), fin_value_CarryCost(), fin_value_NPV(), fin_value_interest()

Examples

```
# Percent needed to return to original value after 10% taken off
fin_value_percent(-10)
# Percent needed to return to original value after 10% added on
fin_value_percent(10)
# Percent needed to return to original value after 50% off 34.50
fin_value_percent(-50, value = 34.5)
```

Fischbein_wt *Weight data across time.*

Description

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

Usage

```
data(Fischbein_wt)
```

Format

A 6*6 correlation matrix based on n = 66 female subjects.

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:

```
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

Fischbein, S. (1977). Intra-pair similarity in physical growth of monozygotic and of dizygotic twins during puberty. *Annals of Human Biology*, **4**. 417-430. doi:10.1080/03014467700002401

See Also

Other datasets: Anthro_LDSC, GFF, HSwls, Psych_LDSC, docData, iqdat, umx, us_skinfold_data

Examples

```
## Not run:
data(Fischbein_wt) # load the data
str(Fischbein_wt) # data.frame
as.matrix(Fischbein_wt) # convert to matrix

## End(Not run)
```

FishersMethod *Fishers Method of combining p-values.*

Description

FishersMethod implements R.A. Fisher's (1925) method for creating a meta-analytic p-value by combining a set of p-values from tests of the same hypothesis in independent samples. See also Stouffer's method for combining Z scores, which allows weighting.

Usage

```
FishersMethod(pvalues, ...)
```

Arguments

pvalues	A vector of p-values, e.g. c(.041, .183)
...	More p-values if you want to offer them up one by one instead of wrapping in a vector for pvalues

Value

- A meta-analytic p-value

References

Fisher, R.A. (1925). *Statistical Methods for Research Workers*. Oliver and Boyd (Edinburgh). ISBN 0-05-002170-2.

- Fisher, R. A (1948). "Questions and answers #14". *The American Statistician*. **2**: 30-31. doi:10.2307/2681650.
- Stouffer, S. A. and Suchman, E. A. and DeVinney, L. C. and Star, S. A. and Williams, R. M. Jr. (1949) The American Soldier, Vol. 1 - Adjustment during Army Life. Princeton, Princeton University Press.

See Also

Other Miscellaneous Stats Functions: SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
FishersMethod(c(.041, .378))
```

---

geometric_mean *Geometric Mean*

Description

The Geometric mean is the nth-root of the product of n input values. Common uses include:

- Compound Annual Growth Rate (CAGR) & Financial Returns: Averaging investment yields over multiple periods. The arithmetic mean overstates performance when volatility is present (e.g., a +50% gain followed by a -50% loss leaves you with a 25% loss, even though the arithmetic average is 0%; the geometric mean correctly yields -29.29% per period, totaling -25%).
- Utilitarian evaluation of value. For example, the geometric mean utility of `c(1, 2, 10)` is

$$(1 * 2 * 10)^{\frac{1}{3}}$$

  = 2.7 not the arithmetic mean: 4.3.
- Proportional Scaling: Proportional growth of organisms and log-normally distributed physical phenomena.
- Normalizing metrics measured on different scales: e.g., computer hardware so no single benchmark disproportionately skews the score.

Usage

```
geometric_mean(x, na.rm = c(TRUE, FALSE))
```

Arguments

x	A vector of values.
na.rm	Remove NAs (default = TRUE). If FALSE, NA input returns NA (like mean()).

Details

The geometric mean is defined as the n-th root of the product of n numbers:

$$G = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

In practice, to prevent numerical overflow or underflow with large vectors, it is calculated via the logarithmic transform:

$$\ln G = \frac{1}{n}\sum_{i=1}^{n} \ln(x_i) \implies G = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \ln(x_i)\right)$$

Value

- Geometric mean of x

References

https://en.wikipedia.org/wiki/Geometric_mean

See Also

- harmonic_mean(), mean()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
geometric_mean(c(1.02, 1.03, 1.1))
# = 1.049

# For a given sum, geometric mean is maximised when all values are equal:
geometric_mean(c(75,75))
geometric_mean(c(50, 100))

v = c(1, 149, 500); round(c(geometric_mean(v), mean(v), median(v)), 0)
# 42 217 149

# Alternate form using mean of the logs
exp(mean(log(c(50, 100))))

# Underlying logic
sqrt(50 * 100)

# Reciprocal duality
1/geometric_mean(c(100, 50))
geometric_mean(c(1/100, 1/50))
```

```
# Missing data are dropped by default (na.rm = TRUE)
geometric_mean(c(50, 100, NA)) # 70.7
# With na.rm = FALSE, NA input gives NA, like mean()
geometric_mean(c(50, 100, NA), na.rm = FALSE) # NA
## Not run:
# Non-positive input is an error, even alongside NAs
geometric_mean(c(50, -100))
geometric_mean(c(50, -100, NA))

## End(Not run)
```

GFF *Twin data: General Family Functioning, divorce, and well-being.*

Description

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

Usage

```
data(GFF)
```

Format

A data frame with 1000 rows of twin-family data columns.

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

van der Aa, N., Boomsma, D. I., Rebollo-Mesa, I., Hudziak, J. J., & Bartels, M. (2010). Moderation of genetic factors by parental divorce in adolescents' evaluations of family functioning and subjective wellbeing. Twin Research and Human Genetics, **13**, 143-162. doi:10.1375/twin.13.2.143

See Also

Other datasets: Anthro_LDSC, Fischbein_wt, HSwls, Psych_LDSC, docData, iqdat, umx, us_skinfold_data

Examples

```
## Not run:
# Twin 1 variables (end in '_T1')
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")
)

## End(Not run)
```

---

ggAddR *Add a fit statistic to a ggplot*

Description

Add a fit statistic to a ggplot

Usage

```
ggAddR(model, effect = NA, xloc = 8, yloc = 10)
```

Arguments

model	a statistical model which contains a fit measure.
effect	optional hard coded fit/effect.
xloc	x location of R.
yloc	y location of R.

Value

- plot

See Also

- umxPlot(), umxPlotFun()

Other Plotting functions: plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
m1 = lm(mpg ~ wt, data = mtcars)
p = ggplot2::ggplot(data = mtcars, aes(x = wt, y = mpg))+ geom_point() +geom_smooth()+
ggAddR(m1, effect = NA, xloc=2, yloc= 10); p

## End(Not run)
```

---

gh_feature_branch *Make a feature branch on a github fork.*

Description

gh_feature_branch takes a feature name and creates a branch on the user's fork (createing this if necessary from an "owner/repo:branch".

It will

1. Fork to your account (if not already existing)
2. Clone to your machine (if not already cloned)
3. Move to preferred location
4. Create the requested fix/feature branch and switch to it

You can then 5. Edit, commit, push, repeat: success! 6. Squash if necessary 7. pull request from your/branch to upstream/main using gh_open_PR_url if you wish.

Usage

```
gh_feature_branch(
  feature = "fix-piglet",
  base = "textmatelives/textmate:main",
  head_owner = "tbates",
  local_path = "~/bin/tm/bundles/"
)
```

Arguments

feature	= "fix-piglet"
base	= "textmatelives/textmate:main"
head_owner	= "tbates"
local_path	= "~/bin/tm/bundle/"

Value

- status

See Also

- gh_open_PR_url()

Other github: gh_message_search(), gh_open_PR_url(), xgh_check_base_name(), xgh_get_current_branch()

Examples

```
## Not run:
 gh_feature_branch("fix/font_issue", base = "textmatelives/textmate:main")

## End(Not run)
```

---

gh_message_search *Search commit messages and link the hits on GitHub.*

Description

gh_message_search asks git for commits on the current branch whose message matches regex, and prints the newest max of them. sort = "asc" flips that set so the oldest of those hits is first.

The pattern is applied to the whole commit message. Each row has a number, the date, a short sha, and the subject. The subject is only the first line, so the match may be further down the message. When origin is a GitHub remote, the row also has the commit page. Pass open as that row number to show it in the browser. The same page is hits$url[n] on the data.frame returned.

regex is an extended regular expression (git log -E --grep). A plain word works as-is. +, (), and | are special; escape them to match those characters literally. Matching ignores case unless ignore_case = FALSE.

Usage

```
gh_message_search(
  regex,
  repo = "~/bin/umx",
  max = 10,
  sort = c("desc", "asc"),
  ignore_case = TRUE,
  open = NULL
)
```

Arguments

regex	Pattern matched against the commit message.
repo	Local checkout. Default "~/bin/umx".
max	How many of the newest matches to keep. Default 10.
sort	"desc" (newest first) or "asc". Default "desc".
ignore_case	Default TRUE.
open	Row number to open on GitHub. Default NULL (print only).

Value

A data.frame (invisibly) with columns n, date, sha, short, subject, url.

See Also

Other github: gh_feature_branch(), gh_open_PR_url(), xgh_check_base_name(), xgh_get_current_branch()

Examples

```
## Not run:
hits <- gh_message_search("double entry")
gh_message_search("double entry", open = 1)

## End(Not run)
```

---

gh_open_PR_url *Build and open the compare URL GitHub renders as a PR page.*

Description

gh_open_PR_url takes the owner, repo and branch you want to open a PR on, along with your owner name, repo and head branch you want to pull from, and opens github at the exact page you need.

Usage

```
gh_open_PR_url(
  head_branch = NULL,
  base_owner = "textmatelives",
  base_repo = "textmate",
  base_branch = "main",
  head_owner = "tbates",
  local_path = ".",
  browse = TRUE
)
```

Arguments

head_branch	Default NULL.
base_owner	Default "textmatelives".
base_repo	Default "textmate".
base_branch	Default "main".
head_owner	Default "tbates".
local_path	Default ".".
browse	Default TRUE.

Details

Before doing this, you want to

1. Fork to your account
2. Clone to your machine
3. Move to preferred location
4. Make a fix or feature branch and switch to it
5. Edit, commit, push, repeat: success!
6. Squash if necessary
7. PR from your/branch to upstream/main

gh_open_PR_url solves #7: pull requesting

Value

- url

References

https://tbates.github.io, tutorials

See Also

- gh_feature_branch()

Other github: gh_feature_branch(), gh_message_search(), xgh_check_base_name(), xgh_get_current_branch()

Examples

```
## Not run:
gh_open_PR_url(head_branch="fix", browse = FALSE)

## End(Not run)
```

harmonic_mean *Harmonic Mean*

Description

The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of the input values. Common uses include computing the mean of ratios, for instance the average P/E ratio in a portfolio. Also it is the correct mean for averaging speeds weighted for distance.

The harmonic mean is calculated using the following formula:

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

Usage

```
harmonic_mean(x, weights = NULL, na.rm = c(TRUE, FALSE))
```

Arguments

x	A vector of values to take the harmonic mean for
weights	Optional vector of weights, one per value in x.
na.rm	Remove NAs (default = TRUE). With weights, cases with NA in x or weights are dropped together so the two stay aligned. If FALSE, NA input returns NA.

Value

- Harmonic mean of x

References

https://en.wikipedia.org/wiki/Harmonic_mean

See Also

- geometric_mean(), aggregate()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
# Harmonic means are suitable for ratios
tmp = c(33/1, 23/1)
harmonic_mean(tmp)

geometric_mean(tmp)
mean(tmp)
```

```
# Example with weights
harmonic_mean(c(33/1, 23/1), weights= c(.2, .8))

# If Jack travels outbound at 1 mph, and returns at 10 miles an hour, what is his average speed?
harmonic_mean(c(1,10)) # 1.81 mph

# Missing data are dropped by default (na.rm = TRUE)
harmonic_mean(c(33, 23, NA)) # 27.1
harmonic_mean(c(33, 23, NA), na.rm = FALSE) # NA
# With weights, incomplete cases are dropped together, keeping x and weights aligned
harmonic_mean(c(33, NA, 23), weights = c(.2, .3, .5)) # 25.18
harmonic_mean(c(33, 23), weights = c(.2, NA)) # 33
## Not run:
# weights must match x in length; non-positive input is an error
harmonic_mean(c(33, 23), weights = c(.2, .3, .5))
harmonic_mean(c(33, -23))

## End(Not run)
```

`HSwls` *Holzinger and Swineford (1939) partitioned into ordered factors for WLS teaching.*

Description

A dataset containing cognitive test scores `x1` to `x9` for 301 students from two schools. This is a modified version of the classic lavaan::HolzingerSwineford1939 or OpenMx's OpenMx::HS.ability.data dataset (which contains the original continuous data), where the variables `x1` to `x9` have been converted into ordered factors (2, 3, or 4 categories) using quantile-based cuts. It is designed for demonstrating and teaching Weighted Least Squares (WLS) estimation with ordinal data.

Usage

```
data(HSwls)
```

Format

A data frame with 301 rows and 15 variables.

Details

The variables are:

- `id`: Student ID
- `sex`: Gender (1 = male, 2 = female)
- `ageyr`: Age in years
- `agemo`: Age in months

- school: School (Grant-White or Pasteur)
- grade: Grade (7 or 8)
- x1: Visual perception (2-category ordered factor)
- x2: Cubes (2-category ordered factor)
- x3: Lozenges (2-category ordered factor)
- x4: Paragraph comprehension (3-category ordered factor)
- x5: Sentence completion (3-category ordered factor)
- x6: Word meaning (3-category ordered factor)
- x7: Speeded addition (4-category ordered factor)
- x8: Code (4-category ordered factor)
- x9: Speeded counting of dots (4-category ordered factor)

References

Holzinger, K. J., & Swineford, F. (1939). A study in factor analysis: The stability of a bi-factor solution. *Supplementary Educational Monographs*, no. 48. Chicago: University of Chicago Press.

See Also

OpenMx::HS.ability.data, umxRAM()

Other datasets: Anthro_LDSC, Fischbein_wt, GFF, Psych_LDSC, docData, iqdat, umx, us_skinfold_data

Examples

```
## Not run:
data(HSwls)
# x1:x3 represent visual perception
# x4:x6 represent verbal ability
# x7:x9 represent speed

m1 = umxRAM("WLS_HS", data = HSwls, type = "DWLS",
  umxPath(v1m0 = c("visual", "verbal", "speed")),
  umxPath(from = "visual", to = c("x1", "x2", "x3")),
  umxPath(from = "verbal", to = c("x4", "x5", "x6")),
  umxPath(from = "speed" , to = c("x7", "x8", "x9")),

  umxPath(unique.bivariate = c("visual", "verbal", "speed")),
  umxPath(v.m. = c("x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9"))
)
plot(m1, means=FALSE, std=TRUE, splines=FALSE, min = c("visual", "verbal", "speed"))
umxSummary(m1, uncertainty= "MLR")

data(HS.ability.data)
df = HS.ability.data

verbal = c("general" , "paragrap", "sentence")
space  = c("visual"  , "cubes"   , "flags"   )
math   = c("addition", "code"    , "counting")
```

```
manifests = c(verbal, space, math)
out = factanal(df[, manifests], factors = 3, scores = "Bartlett", na.action = na.exclude)

m1 = umxRAM("One Factor", data = df,
 umxPath(v1m0 = "G"),
 umxPath(v.m. = c(manifests, "V", "S", "M")),
 umxPath(from = "V", to = verbal),
 umxPath(from = "S", to = space),
 umxPath(from = "M", to = math),

 umxPath(from = "G", to = c("V", "S", "M"))
)
plot(m1, means=FALSE, std=TRUE, max= manifests, min="G", same= c("V", "S", "M"))

## End(Not run)
```

`install.OpenMx` *Install OpenMx (GenomicMx binary build)*

Description

Install stock **CRAN** OpenMx, or the **GenomicMx** OpenMx build used by umx for modern WLS/GSEM (Jacobians, `type = "summary"` WLS data, multi-group stack). The GenomicMx build **replaces** whatever OpenMx is in the target library (same idea as the old NPSOL binary installs).

Public install path (everyone): pre-built OpenMx binaries are published on the umx GitHub Release tag **genomicmx** (`https://github.com/tbates/umx/releases/tag/genomicmx`). Call `install.OpenMx("GenomicMx")` (the default). That is the supported way to get the GenomicMx engine while the OpenMx source tree lives privately on GitHub. Binaries are CRAN flavour (no NPSOL) and are built for R `release` on Linux x86_64, Windows x86_64, and macOS arm64 (Phase 1).

Options for `loc`:

1. `"GenomicMx"` (**default**): download the platform binary for this OS/arch/R from the `genomicmx` Release tag when available.
2. `"CRAN"`: install stock OpenMx from CRAN (enough for many twin/RAM models, not the full GenomicMx WLS stack).
3. `"open release page"`: open the umx GenomicMx Release tag in a browser so you can pick a binary by hand.

GenomicMx binaries are linked against **oneTBB** and declare current-CRAN `Imports` floors (including `RcppParallel >= 6.2.0` and `Matrix >= 1.7-6`). `install.OpenMx("GenomicMx")` (and a custom `url` binary) upgrades those packages first. A `repos = NULL` binary install does not upgrade Imports, so R's recommended `Matrix` (often one patch behind CRAN) used to fail with `package 'Matrix' 1.7-5 was found, but >= 1.7.6 is required`. After the OpenMx files are in place, this function tries `loadNamespace("OpenMx")` when OpenMx is not already loaded. If OpenMx is already loaded, restart R to pick up the new `.so` / `.dll`.

Source installs from a private GitHub fork are **not** offered here (they cannot work for general users). Maintainers building OpenMx locally should use `mx_make()` on their source tree.

Usage

```
install.OpenMx(
  loc = c("GenomicMx", "open GenomicMx release page", "CRAN"),
  url = NULL,
  lib,
  repos = getOption("repos")
)

umx_update_OpenMx(
  loc = c("GenomicMx", "open GenomicMx release page", "CRAN"),
  url = NULL,
  lib,
  repos = getOption("repos")
)
```

Arguments

loc	Which build to get (default "GenomicMx").
url	Custom URL or local path to an OpenMx binary package. On Mac, "Finder" uses the package selected in the Finder; "" opens a file chooser.
lib	Where to install the package (passed to install.packages()).
repos	CRAN repository (for loc = "CRAN").

Value

None (called for side effects).

References

https://github.com/tbates/umx, https://github.com/tbates/umx/releases/tag/genomicmx

See Also

umxVersion(), xmu_openmx_engine_status(), mx_make()

Other Miscellaneous Utility Functions: libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
## Not run:
install.OpenMx("GenomicMx")         # GenomicMx OpenMx binary from umx Releases (default)
install.OpenMx("open release page") # browse releases and install a file yourself
install.OpenMx("CRAN")                # stock CRAN OpenMx only

## End(Not run)
```

iqdat *Twin data: IQ measured longitudinally across 4 ages.*

Description

Measures of IQ across four ages in 261 pairs of identical twins and 301 pairs of fraternal (DZ) twins. (see details). It is used as data for the `umxSimplex()` examples.

Usage

```
data(iqdat)
```

Format

A data frame with 562 rows (twin families). Nine measures on each twin.

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

Boomsma, D. I., Martin, N. G., & Molenaar, P. C. (1989). Factor and simplex models for repeated measures: application to two psychomotor measures of alcohol sensitivity in twins. *Behavior Genetics*, **19**, 79-96. doi:10.1007/BF01065885.

See Also

`umxSimplex()`

Other datasets: `Anthro_LDSC`, `Fischbein_wt`, `GFF`, `HSwls`, `Psych_LDSC`, `docData`, `umx`, `us_skinfold_data`

Examples

```
## Not run:
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

## End(Not run)
```

---

libs *load libraries*

Description

`libs` allows loading multiple libraries in one call

Usage

```
libs(..., force.update = FALSE)
```

Arguments

`...`	library names as symbols/naked names, strings, or a vector of names
`force.update`	install.package even if present (to get new version) FALSE

Value

- nothing.

See Also

- `library()`, `install.packages()`, `remove.packages()`

Other Miscellaneous Utility Functions: `install.OpenMx()`, `qm()`, `umx`, `umxLav2RAM()`, `umxModelNames()`, `umxRAM2Lav()`, `umxVersion()`, `umx_array_shift()`, `umx_data()`, `umx_find_object()`, `umx_lower.tri()`, `umx_msg()`, `umx_open_CRAN_page()`, `umx_pad()`, `umx_print()`, `umx_scholastica_report()`, `umx_tail()`, `umx_wide2long()`, `umx_wide4lmer()`

Examples

```
## Not run:
libs(ggplot2)
libs("car")
libs(c("OpenMx", "car"))
libs(OpenMx, car)
# remove.packages()

## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

Details

Base loadings handles factanal() objects.

Value

- matrix of loadings

References

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

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

`loadings.MxModel` *Extract factor loadings from an EFA (factor analysis).*

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

```
## 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

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

See Also

- factanal(), loadings()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
myVars = c("mpg", "disp", "hp", "wt", "qsec")
m1 = umxEFA(name = "test", factors = 2, data = mtcars[, myVars])
loadings(m1)

## End(Not run)
```

mx_make *Developer "make" helper for local OpenMx / GenomicMx*

Description

This is a utility function for the OpenMx package maintainer. For convenience, it lives in **umx**, not the OpenMx package. mx_make mirrors umx_make(). It uses OpenMx's **Makefile** to compile and install (C++, OpenMP, NPSOL vs CRAN build). Use **devtools** for win-builder and package hygiene. After install, **restart R** then library(OpenMx). It does **not** use devtools::load_all() due to OpenMx (DLL / S4 load issues).

Two independent targets matter:

- **Installed library** - what library(OpenMx) runs. Updated by mx_make() / "install" / "cran-install" / "GenomicMx" (or CRAN).
- Source tree at pkg - what "win" packages for win-builder. Independent of whatever is currently installed in R.

Usage

```
mx_make(
 what = c("install", "NPSOL", "cran-install", "build", "Rd", "check", "win", "spell",
   "sitrep", "deps_install", "testthat", "git", "GenomicMx", "build_GenomicMx",
   "--help"),
 pkg = "~/bin/OpenMx",
 deploymentTarget = "14.0",
 openmp = TRUE,
 email = NULL
)
```

Arguments

what	Target. One of:
	"install" / "NPSOL" make install (NPSOL-capable tree when available).
	"cran-install" make cran-install (no NPSOL).
	"build" make build (local binary package).
	"Rd" make roxygen (OpenMx util/rox; refresh man/*.Rd).
	"check" make cran-check.
	"win" devtools::check_win_devel(pkg) – source tree at pkg.
	"spell", "sitrep", "deps_install", "testthat", "git" Same idea as umx_make().
	"GenomicMx" install.OpenMx() prebuilt binary from GitHub Releases.
	"help" / "-help" List mx_make targets and run make help.
pkg	Path to the OpenMx source tree (default "~/bin/OpenMx").
deploymentTarget	
	macOS MACOSX_DEPLOYMENT_TARGET for gfortran/clang (default "14.0"). Ignored on non-darwin.

openmp	Logical; pass OPENMP=yes or OPENMP=no to make (default TRUE).
email	Optional. For what = "win" only: override the win-builder results address (otherwise devtools uses DESCRIPTION Maintainer – e.g. upstream OpenMx goes to Kirkpatrick). Example: mx_make("win", pkg = "~/bin/OpenMx-upstream-wincheck", email = "you@example.com").

Value

Invisibly NULL, or the result of a returning target (e.g. sitrep).

Flight checklist

1. **Update Rd files** after editing roxygen in R: mx_make("Rd") runs OpenMx make roxygen (util/rox: compile DLL + roxygenize rd). Do **not** use bare devtools::document() on OpenMx; man pages are git-tracked and the Makefile owns the official path.
2. **Install this tree into R's library**: mx_make() or mx_make("install") (NPSOL when the Makefile enables it). Or mx_make("cran-install") without NPSOL.
3. **Restart R**, then verify identity: library(OpenMx); packageVersion("OpenMx"); find.package("OpenMx"). The path must be the library you just installed into (not a forgotten CRAN copy on another .libPaths() entry). Optional: xmu_openmx_engine_status() for GenomicMx capability.
4. **Win-builder** (same source tree, not the installed DLL): Prefer a clean commit if you care what is archived; then mx_make("win") – devtools::check_win_devel(pkg). Local install is **not** required for win-builder; only the files under pkg are uploaded.
5. **Prebuilt release binary** (not local WIP (work in progress)): mx_make("GenomicMx") -> install.OpenMx() from GitHub Releases.

References

https://github.com/tbates/umx/releases, OpenMx Makefile

See Also

umx_make(), install.OpenMx(), xmu_openmx_engine_status()

Other xmu internal not for end user: umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(),

xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
# Flight checklist (local engine + win-builder)
mx_make("Rd")             # util/rox: refresh man/*.Rd
mx_make()                 # make install this tree into R library
# restart R, then:
# library(OpenMx); packageVersion("OpenMx"); find.package("OpenMx")
mx_make("win")            # upload source tree to win-builder (devel)
mx_make("win", pkg = "~/bin/OpenMx-upstream-wincheck",
       email = "timothy.c.bates@gmail.com")  # results to you, not DESCRIPTION Maintainer

mx_make("cran-install")
mx_make("spell")
mx_make("sitrep")
mx_make("deps_install")
mx_make("GenomicMx")      # prebuilt binary via install.OpenMx("GenomicMx")
mx_make("NPSOL")          # alias of install
mx_make("--help")

## End(Not run)
```

---

noNAs *Succinctly select complete rows from a dataframe*

Description

Succinctly select complete rows from a dataframe.

Usage

```
noNAs(df, rows = NULL, cols = NULL, drop = TRUE)
```

Arguments

df	an data.frame() to select on
rows	Rows to keep (optional, incomplete rows still discarded)
cols	Cols to keep
drop	Whether to return a vector when only 1 column is selected (default TRUE)

Value

- Complete rows and (optionally) selected columns

See Also

Other Data Functions: prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
tmp = mtcars
tmp[2,1] = NA
noNAs(tmp, cols="mpg")
noNAs(tmp, cols="mpg", drop = FALSE)
noNAs(tmp) # no Mazda RX4 Wag
```

---

oddsratio *Compute odds ratio (OR)*

Description

Returns the odds in each group, and the odds ratio. Takes the cases (n) and total N as a list of two numbers for each of two groups.

Usage

```
oddsratio(grp1 = c(n = 3, N = 10), grp2 = c(n = 1, N = 10), alpha = 0.05)
```

Arguments

grp1	either odds for group 1, or cases and total N , e.g c(n=3, N=10)
grp2	either odds for group 2, or cases and total N , e.g c(n=1, N=20)
alpha	for CI (default = 0.05)

Details

Returns a list of odds1, odds2, and OR + CI. Has a pretty-printing method so displays as:

```
Group 1 odds = 0.43
Group 2 odds = 0.11
          OR = 3.86 CI95[0.160, 3.64]
```

Value

- List of odds in group 1 and group2, and the resulting OR and CI

See Also

- umx_r_test()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
oddsratio(grp1 = c(1, 10), grp2 = c(3, 10))
oddsratio(grp1 = 0.111, grp2 = 0.429)
oddsratio(grp1 = c(3, 10), grp2 = c(1, 10))
oddsratio(grp1 = c(3, 10), grp2 = c(1, 10), alpha = .01)
```

---

plot.MxLISRELModel *Create and display a graphical path diagram for a LISREL model.*

Description

plot.MxLISRELModel produces SEM diagrams using [DiagrammeR::DiagrammeR() to create the image.

Usage

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

```
  splines = c("TRUE", "FALSE", "compound", "ortho", "polyline"),
  min = NULL,
  same = NULL,
  max = NULL,
  ...
)
```

Arguments

x	A LISREL `OpenMx::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
labels	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)
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

Note: By default, plots open in your browser (or plot pane if using RStudio).

Opening in an external editor/app

The underlying format is graphviz. 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 MacOS, you may need to associate the '.gv' extension with your 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".

The commercial application "OmniGraffle" is great for editing these images.

References

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

See Also

- umx_set_plot_format(), umx_set_auto_plot(), umx_set_plot_format(), plot.MxModel(), umxPlotACE(), umxPlotCP(), umxPlotIP(), umxPlotGxE()

Other Plotting functions: ggAddR(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
library(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxLISREL("one_factor", data = demoOneFactor, type = "cov",
               umxPath("G", to = manifests),
               umxPath(var = manifests),
               umxPath(var = "G", fixedAt = 1))
plot(m1)

## End(Not run)
```

---

plot.MxModel *Create and display a graphical path diagram for a model.*

Description

plot() produces SEM diagrams in graphviz format, and relies on DiagrammeR::DiagrammeR() to create the image.

Usage

```
## S3 method for class 'MxModel'
plot(
  x = NA,
  std = FALSE,
  fixed = TRUE,
  means = TRUE,
  digits = 2,
  uncertainty = c("none", "SE", "MLR", "CI"),
  file = "name",
  labels = c("none", "labels", "both"),
```

```
  resid = c("circle", "line", "none"),
  strip_zero = FALSE,
  splines = c("TRUE", "FALSE", "compound", "ortho", "polyline"),
  min = NULL,
  same = NULL,
  max = NULL,
  ...
)

## S3 method for class 'MxRAMModel'
plot(
  x = NA,
  std = FALSE,
  fixed = TRUE,
  means = TRUE,
  digits = 2,
  uncertainty = c("none", "SE", "MLR", "CI"),
  file = "name",
  labels = c("none", "labels", "both"),
  resid = c("circle", "line", "none"),
  strip_zero = FALSE,
  splines = c("TRUE", "FALSE", "compound", "ortho", "polyline"),
  min = NULL,
  same = NULL,
  max = NULL,
  ...
)
```

Arguments

x	An OpenMx::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
uncertainty	What type of parameter uncertainty to report: "SE" (standard ML standard errors), "MLR" (robust standard errors and robust fit), "CI" (profile likelihood confidence intervals), or "none" (none).
file	The name of the dot file to write: NA = none; "name" = use the name of the model
labels	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

Note: DiagrammeR::DiagrammeR() is supported out of the box. By default, plots open in your browser. Other options include pdf SVG etc.

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

The commercial application "OmniGraffle" is great for editing these images. On unix and windows, plot() will create a pdf and open it in your default pdf reader.

If you use graphviz, we try and use that app, but YOU HAVE TO INSTALL IT!

MacOS note: On Mac, we will 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

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

See Also

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

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
```

```
)
plot(m1)
plot(m1, std = TRUE, resid = "line", digits = 3, strip_zero = FALSE)

# ============================================================
# = With a growth model, demonstrate splines= false to force =
# = straight lines, and move "rank" of intercept object      =
# ============================================================

m1 = umxRAM("grow", data = myGrowthMixtureData,
 umxPath(var = manifests, free = TRUE),
 umxPath(means = manifests, fixedAt = 0),
 umxPath(v.m. = c("int","slope")),
 umxPath("int", with = "slope"),
 umxPath("int", to = manifests, fixedAt = 1),
 umxPath("slope", to = manifests, arrows = 1, fixedAt = c(0,1,2,3,4))
)

plot(m1, means=FALSE, strip=TRUE, splines="FALSE", max="int")

## End(Not run) # end dontrun
```

`plot.MxModelTwinMaker` *Create and display a graphical path diagram for a path-based twin model.*

Description

Assumes the model has a group called "MZ" inside.

Usage

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

```
  max = NULL,
  ...
)
```

Arguments

x	A umxTwinMaker() model 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)
oneTwin	(whether to plot a pair of twins, or just one (default = TRUE)
sep	The separator for twin variables ("_T")
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
labels	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

If you use umx_set_plot_format("graphviz"), they will open in a graphviz helper app (if installed). The commercial application "OmniGraffle" is great for editing these images. On unix and windows, plot() will create a pdf and open it in your default pdf reader.

See Also

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

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
require(umx)
#
# ====================
# = Make an ACE model =
# ====================
# 1. Clean data: Add separator and scale
data(twinData)
tmp = umx_make_twin_data_nice(data=twinData, sep="", zygosity="zygosity", numbering=1:2)
tmp = umx_scale_wide_twin_data(varsToScale= c("wt", "ht"), sep= "_T", data= tmp)
mzData = subset(tmp, zygosity %in%  c("MZFF", "MZMM"))
dzData = subset(tmp, zygosity %in%  c("DZFF", "DZMM"))

# 2. Define paths: You only need the paths for one person:
paths = c(
umxPath(v1m0 = c("a1", 'c1', "e1")),
umxPath(means = c("wt")),
umxPath(c("a1", 'c1', "e1"), to = "wt", values=.2)
)
m1 = umxTwinMaker("test", paths, mzData = mzData, dzData= dzData)
plot(m1, std= TRUE, means= FALSE)
plot(m1, means=FALSE, std=TRUE, strip=TRUE, splines="FALSE", max="intercept")

## End(Not run) # end dontrun

# =================
# = An ACEv model =
# =================
# Not complete

paths = c(
umxPath(v1m0 = c("A1", 'C1', "E1")),
umxPath(v1m0 = c("A2", 'C2', "E2")),
umxPath(v.m0 = c("l1", 'l2')),
umxPath(v.m. = c("wt", "ht")),
umxPath(c("A1", 'C1', "E1"), to = "l1", values= .2),
umxPath(c("A2", 'C2', "E2"), to = "l2", values= .2),
umxPath(c("l1", 'l2'), to = c("wt", "ht"), values= .2)
)
```

`plot.percent` *Plot a percent change graph*

Description

Plot method for "percent" objects: e.g. `fin_value_percent()`.

Usage

```
## S3 method for class 'percent'
plot(x, ...)
```

Arguments

x	percent object.
...	further arguments passed to or from other methods.

Value

- invisible

See Also

- fin_value_percent()

Examples

```
# Percent needed to return to original value after 10% off
fin_value_percent(-10)
# Percent needed to return to original value after 10% on
tmp = fin_value_percent(10)
plot(tmp)

# Percent needed to return to original value after 50% off 34.50
fin_value_percent(-50, value = 34.5, logY = FALSE)
```

---

plot.umx_GWAS *Plot GWAS Results*

Description

Generates a Manhattan or Q-Q plot for GWAS results. Requires the ggwas package.

Usage

```
## S3 method for class 'umx_GWAS'
plot(x, type = c("manhattan", "qq"), ...)
```

Arguments

x	A umx_GWAS object (returned by umxGSEM_GWAS).
type	Type of plot: "manhattan" (default) or "qq".
...	Additional arguments passed to ggwas manhattan_plot or qq_plot (optional package; install from GitHub bczech/ggwas if needed).

Details

See ?manhattan_plot for more parameters. Key ones include chromosomes to restrict chromosomes, highlight_snps to highlight a SNP, label_top_n, downsample (TRUE), and downsample_n (2e+05), title = NULL

Value

A ggplot object.

See Also

Other Reporting Functions: aic_weights(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Other GSEM: Anthro_LDSC, Psych_LDSC, umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
# Requires optional GitHub package ggwas (not on CRAN):
#   remotes::install_github("bczech/ggwas")
# data(example_gwas, package = "ggwas")
# plot.umx_GWAS(example_gwas, type = "manhattan")

## End(Not run)
```

---

power.ACE.test *Test the power of an ACE model to detect paths of interest.*

Description

power.ACE.test simulates a univariate ACE model. It computes power to detect dropping one or more paths (a, c, or a after dropping c), specified in drop=.

The interface and functionality of this service are experimental and subject to change.

Usage

```
power.ACE.test(
  AA = 0.5,
  CC = 0,
  EE = NULL,
  DD = NULL,
  update = c("a", "c", "a_after_dropping_c", "d"),
  value = 0,
  n = NULL,
```

```
  MZ_DZ_ratio = 1,
  sig.level = 0.05,
  power = 0.8,
  method = c("ncp", "empirical"),
  search = FALSE,
  tryHard = c("yes", "no", "ordinal", "search"),
  digits = 2,
  optimizer = NULL,
  nSim = 4000
)
```

Arguments

AA	Additive genetic variance (Default .5)
CC	Shared environment variance (Default 0)
EE	Unique environment variance. Leave NULL (default) to compute an amount summing to 1.
DD	Dominance Is set (default= NULL) compute an ADE rather than ACE model (DZr=.25)
update	Component to drop (Default "a", i.e., drop a)
value	Value to set dropped path to (Default 0)
n	If provided, solve at the given number of MZ+DZ pairs (Default NULL)
MZ_DZ_ratio	MZ pairs per DZ pair (Default 1 = equal numbers.)
sig.level	alpha (p-value) Default = 0.05
power	Default = .8 (80 percent power, equal to 1 - Type II rate)
method	How to estimate power: Default = use non-centrality parameter ("ncp"). Alternative is "empirical"
search	Whether to return a search across power or just a point estimate (Default FALSE = point)
tryHard	Whether to tryHard to find a solution (default = "yes", alternatives are "no"...)
digits	Rounding for reporting parameters (default 2)
optimizer	If set, will switch the optimizer.
nSim	Total number of pairs to simulate in the models (default = 4000)

Details

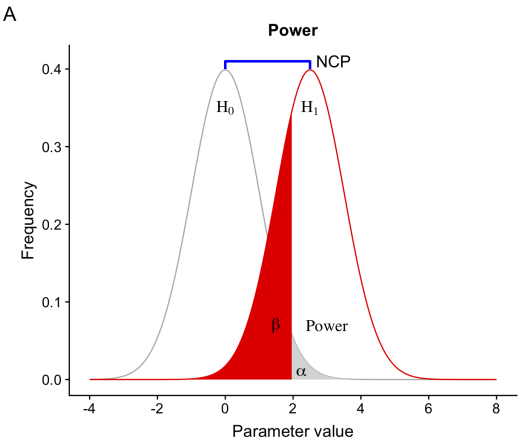
Statistical power is the proportion of studies that, over the long run, one should expect to yield a statistically significant result given certain study characteristics such as sample size (N), the expected effect size (β), and the criterion for statistical significance (α).

(with nMZpairs= 2000 and MZ_DZ_ratio*nMZpairs DZ twins.

A typical target for power is 80%. Much as the accepted critical p-value is .05, this has emerged as a trade off, in this case of resources required for more powerful studies against the cost of missing a true effect. People interested in truth discourage running studies with low power: A study with 20 percent power will fail to detect real effects 80% of the time. But even with zero power, the

Type-I error rate remains a nominal 5% (and with any researcher degrees of freedom, perhaps much more than that). Low powered research, then, fails to detect true effects, and generates support for random false theories about as often. This sounds silly, but empirical rates are often as low as 20% (Button, et al., 2013).

Illustration of α, β, and power (1-β):



Value

`OpenMx::mxPower()` object

References

- Visscher, P.M., Gordon, S., Neale, M.C. (2008). Power of the classical twin design revisited: II detection of common environmental variance. *Twin Res Hum Genet*, **11**: 48-54. doi:10.1375/twin.11.1.48.
- Button, K. S., Ioannidis, J. P., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S., and Munafo, M. R. (2013). Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, **14**, 365-376. doi:10.1038/nrn3475

See Also

- `umxPower()`, `OpenMx::mxPower()`, `umxACE()`

Other Twin Modeling Functions: `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
# ====================================================
# = N for .8 power to detect a^2 = .5 equal MZ and DZ =
# ====================================================
```

```
power.ACE.test(AA = .5, CC = 0, update = "a")
# Suggests n = 84 MZ and 94 DZ pairs.

## Not run:
# ================================
# = Show power across range of N =
# ================================
power.ACE.test(AA= .5, CC= 0, update = "a", search = TRUE)

# Salutary note: You need well fitting models with correct betas in the data
# for power to be valid.
# tryHard helps ensure this, as does the default nSim= 4000 pair data.
# Power is important to get right, so I recommend using tryHard = "yes" (the default)

# ====================
# = Power to detect C =
# ====================

# 102 of each of MZ and DZ pairs for 80% power (default).
power.ACE.test(AA= .5, CC= .3, update = "c")

# ========================================
# = Set 'a' to a fixed, but non-zero value =
# ========================================

power.ACE.test(update= "a", value= sqrt(.2), AA= .5, CC= 0)

# ======================================
# = Drop More than one parameter (A & C) =
# ======================================
# E vs AE: the hypothesis that twins show no familial similarity.
power.ACE.test(update = "a_after_dropping_c", AA= .5, CC= .3)

# ================================================
# = More power to detect A > 0 when more C present  =
# ================================================

power.ACE.test(update = "a", AA= .5, CC= .0)
power.ACE.test(update = "a", AA= .5, CC= .3)

# ================================================
# = More power to detect C > 0 when more A present?  =
# ================================================

power.ACE.test(update = "c", AA= .0, CC= .5)
power.ACE.test(update = "c", AA= .3, CC= .5)


# ==================================
# = Power with more DZs or more MZs =
# ==================================

# Power about the same: total pairs with 2 MZs per DZ
```

```
power.ACE.test(MZ_DZ_ratio= 2/1, update= "a", AA= .3, CC= 0, method="ncp", tryHard="yes")
power.ACE.test(MZ_DZ_ratio= 1/2, update= "a", AA= .3, CC= 0, method="ncp", tryHard="yes")
power.ACE.test(update= "a", AA= .3, CC= 0, method="ncp", tryHard="yes")


# ====================================
# = Compare ncp and empirical methods =
# ====================================

power.ACE.test(update= "a", AA= .5, CC= 0, method = "ncp")
# method = "ncp": For 80% power, you need 166 MZ and 166 DZ pairs
power.ACE.test(update= "a", AA= .5, CC= 0, method= "empirical")
# method= "empirical": For 80% power, you need 154 MZ and 154 DZ pairs

# ====================
# = Show off options =
# ====================
# 1. tryHard

power.ACE.test(update = "a", AA= .5, CC= 0, tryHard= "no")

# 2. toggle optimizer

power.ACE.test(update= "a", AA= .5, CC= 0, optimizer= "SLSQP")

# 3. You can raise or lower the number of pairs used in the true model
#    by varying nSim (twin pairs in the simulated data).

power.ACE.test(update = "a", AA= .5, CC= 0, nSim= 20)


## End(Not run)
```

`print.oddsratio` *Print a scale "oddsratio" object*

Description

Print method for the `oddsratio()` function.

Usage

```
## S3 method for class 'oddsratio'
print(x, digits = 3, ...)
```

Arguments

`x`	A `oddsratio()` result.
`digits`	The rounding precision.
`...`	further arguments passed to or from other methods.

Value

- invisible oddsratio object (x).

See Also

- print(), oddsratio(),

Examples

```
oddsratio(grp1 = c(1, 10), grp2 = c(3, 10))
oddsratio(grp1 = c(3, 10), grp2 = c(1, 10))
oddsratio(grp1 = c(3, 10), grp2 = c(1, 10), alpha = .01)
```

---

print.percent *Print a percent object*

Description

Print method for "percent" objects: e.g. fin_value_percent().

Usage

```
## S3 method for class 'percent'
print(x, ...)
```

Arguments

x	percent object.
...	further arguments passed to or from other methods.

Value

- invisible

See Also

- fin_value_percent()

Examples

```
# Percent needed to return to original value after 10% off
fin_value_percent(-10)
# Percent needed to return to original value after 10% on
fin_value_percent(10)

# Percent needed to return to original value after 50% off 34.50
fin_value_percent(-50, value = 34.5)
```

print.reliability *Print a scale "reliability" object*

Description

Print method for the `reliability()` function.

Usage

```
## S3 method for class 'reliability'
print(x, digits = 4, ...)
```

Arguments

x	A `reliability()` result.
digits	The rounding precision.
...	further arguments passed to or from other methods

Value

- invisible reliability object (x)

See Also

- `print()`, `reliability()`,

Examples

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

print.RMSEA *Print a RMSEA object*

Description

Print method for "RMSEA" objects: e.g. `RMSEA()`.

Usage

```
## S3 method for class 'RMSEA'
print(x, ...)
```

Arguments

x	RMSEA object.
...	further arguments passed to or from other methods.

Value

- invisible

See Also

- RMSEA(), print()

Examples

```
## Not run:
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type= "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)
tmp = summary(m1)
RMSEA(tmp)

## End(Not run)
```

---

prolific_anonymize *Clean up a prolific file for sharing by removing anonymity-compromising columns.*

Description

prolific.ac IDs and other columns like IP and lat/long might compromise subject anonymity when shared. `prolific_anonymize` replaces PIDs with a simple numeric sequence, preserving repeated measures in long data, and removing other columns. You can delete additional columns by adding them to `extraColumns`. It is ideal for use when sharing data to ResearchBox (researchbox.org) which enforces anonymization.

Usage

```
prolific_anonymize(
  df = NULL,
  PID = "PID",
  alsoDrop = NA,
  baseOffset = 10000,
```

```
  extraColumns = "deprecated"
)
```

Arguments

df	Existing datafile to anonymize.
PID	The prolific ID col name to anonymize
alsoDrop	Any extra columns to delete (default NA)
baseOffset	The numeric to start renumbering PIDs from (default = 1e4)
extraColumns	A deprecated synonym for alsoDrop

Value

- [data.frame]

References

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

See Also

- prolific_check_ID(), prolific_read_demog(), umx_merge_randomized_columns()

Other Data Functions: noNAs(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
tmp = prolific_anonymize(df, PID = "PID")

## End(Not run)
```

---

prolific_check_ID *Return PIDs in df*

Description

Participants may time-out on Prolific, but still complete on Qualtrics. This identifies them.

Usage

```
prolific_check_ID(IDs, df, IDcol = "PROLIFIC_PID")
```

Arguments

IDs	Timed-out (or other) IDs to look for.
df	to search.
IDcol	Name of prolific ID column (default PROLIFIC_PID)

Value

- list of IDs in the dataframe

See Also

- prolific_read_demog(), prolific_anonymize(), umx_merge_randomized_columns() # prolific_check_ID()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
# IDs = c("59d0ec2446447f00011edb063","5a08c9a7f2e3460001edb063f0254")
# prolific_check_ID(IDs, df)
```

---

prolific_read_demog *Read and optionally merge demographics file from prolific academic*

Description

prolific academic provides a demographics file. This reads it and merges with your data using PID and participant_id

Usage

```
prolific_read_demog(
  file,
  base = "",
  df = NULL,
  by.df = "PROLIFIC_PID",
  by.demog = "Participant.id",
  age = "age",
  sex = "Gender",
  vars = NULL,
  all.df = TRUE,
  all.demog = FALSE,
  verbose = FALSE
)
```

Arguments

file	Path to demographics file.
base	Optional path to folder, in which case 'file' is just filename.
df	Existing datafile to merge demographics into (optional)
by.df	The ID name in existing df (default = "PROLIFIC_PID")
by.demog	The ID name in the prolific demographics file (default = "Participant id" was by.demog)
age	Name of age var in demographics file ("age")
sex	Name of sex var in demographics file ("Sex")
vars	Additional vars to keep from demographics file (WAS age & Sex)
all.df	Whether to keep all lines of df (default = TRUE)
all.demog	Whether to keep all lines (people) in the demographics file (default = FALSE)
verbose	Print variable names found in the file.

Value

- [data.frame]

References

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

See Also

- prolific_check_ID(), prolific_anonymize(), umx_merge_randomized_columns()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
fp = "~/Desktop/prolific_export_5f20c3e662e3b6407dcd37a5.csv"
df = prolific_read_demog(fp, sex = "Gender", age = "Age", df = df)
tmp = prolific_read_demog(fp, by.df = "PROLIFIC_PID", vars=c("Ethnicity.simplified"))

## End(Not run)
```

prolific_scoring_stub *prolific_scoring_stub*

Description

Automatically generates a scoring stub for scales in a Prolific/Qualtrics dataset.

Usage

```
prolific_scoring_stub(df = NULL, deleteJunk = FALSE)
```

Arguments

df	The data frame imported from Qualtrics/Prolific.
deleteJunk	Logical. If TRUE, return the dataframe with the metadata/junk columns removed. (default = FALSE)

Value

The data frame (invisibly). If deleteJunk = TRUE, junk columns are removed.

References

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

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
prolific_scoring_stub(df, deleteJunk = FALSE)

## End(Not run)
```

Psych_LDSC *Psychiatric and educational attainment Genomic SEM LDSC dataset.*

Description

A dataset containing the genetic covariance structure from LD score regression (LDSC) for 5 psychiatric and educational attainment phenotypes: **SCZ**, **BIP**, **MDD**, **EA**, and **INSOM**. This is the object shape returned by GenomicSEM / `umxGSEM_ldsc()` and the example `covstruc` used throughout the GSEM helpers (see also Anthro_LDSC).

Usage

```
data(Psych_LDSC)
```

Format

A list with elements `S`, `V`, `I`, `N`, and `m` (see Details).

Details

LDSC list fields (same `S/V/I/N/m` layout as GenomicSEM `ldsc`; `V` **names are OpenMx residual labels**):

- `S`: 5 x 5 genetic covariance matrix among the phenotypes (diagonals are SNP heritabilities on the LDSC scale).
- `V`: 15 x 15 sampling covariance of the unique elements of `S` (OpenMx vech order; dimnames like `var_SCZ`, `poly_BIP_SCZ`). Used as asymptotic covariance for WLS/DWLS. Not GenomicSEM `SCZ_SCZ` / `BIP_SCZ`.
- `I`: 5 x 5 **LDSC intercept** matrix. Diagonal = per-trait LDSC intercepts (expected near 1 without confounding). Off-diagonal = cross-trait intercepts reflecting sample overlap. Preferred input for GC in `umxGSEM_GWAS()`.
- `N`: 1 x 15 effective sample sizes for each unique `S` element.
- `m`: Number of SNPs used in LDSC.

See Also

Anthro_LDSC, `umxGSEM()`, `umxGSEM_GWAS()`, `umxGSEM_ldsc()`

Other GSEM: `Anthro_LDSC`, `plot.umx_GWAS()`, `umxGSEM()`, `umxGSEM_GWAS()`, `umxGSEM_dl_tutorial_files()`, `umxGSEM_label_ldsc()`, `umxGSEM_ldsc()`, `umxGSEM_munge()`, `umxGSEM_std()`, `umxGSEM_sumstats()`, `umxSummary.MxDataStatic()`, `umxSummary.list()`

Other datasets: `Anthro_LDSC`, `Fischbein_wt`, `GFF`, `HSwls`, `docData`, `iqdat`, `umx`, `us_skinfold_data`

Examples

```
data(Psych_LDSC)
str(Psych_LDSC, max.level = 1)
names(Psych_LDSC) # "V" "S" "I" "N" "m"
# Pretty-print S, I, V, ... (see also umxSummary on WLS mxData)
umxSummary(Psych_LDSC)
# I = LDSC intercept matrix (diag ~ 1; off-diag = sample overlap)
round(Psych_LDSC$I, 3)
# Common factor of all five traits (Grotzinger et al. 2019 style DWLS GSEM)
## Not run:
m1 = umxGSEM(model = "g ~= SCZ + BIP + MDD + EA + INSOM",
             covstruc = Psych_LDSC, estimation = "DWLS")
umxSummary(m1)

## End(Not run)
```

---

qm *qm*

Description

Quickmatrix function

Usage

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

Arguments

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

Value

- matrix

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

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)
```

`rad2deg` *Convert Radians to Degrees*

Description

Just a helper to multiply radians by 180 and divide by π to get degrees.

note: R's trig functions, e.g. `sin()` use Radians for input!

There are 2π radians in a circle. 1 Rad = $180/\pi$ degrees = ~ 57.296 degrees.

Usage

```
rad2deg(rad)
```

Arguments

`rad`	The value in Radians you wish to convert

Value

- value in degrees

References

https://en.wikipedia.org/wiki/Radian

See Also

- `deg2rad()`, `sin()`

Other Miscellaneous Functions: `deg2rad()`, `umxBrownie()`, `umx_check_parallel()`, `umx_complete_dollar()`

Examples

```
rad2deg(pi) #180 degrees
```

`reliability` *Report coefficient alpha (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

None

References

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

See Also

- `print.reliability()`,

Other Miscellaneous Stats Functions: `FishersMethod()`, `SE_from_p()`, `geometric_mean()`, `harmonic_mean()`, `oddsratio()`, `umx`, `umxCov2cor()`, `umxHetCor()`, `umxParan()`, `umxWeightedAIC()`, `umx_apply()`, `umx_cor()`, `umx_means()`, `umx_r_test()`, `umx_round()`, `umx_scale()`, `umx_var()`

Examples

```
# treat car data 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

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

Arguments

`object`	An fitted `OpenMx::mxModel()` from which to get residuals
`digits`	round to how many digits (default = 2)
`suppress`	smallest deviation to print out (default = NULL = show all)
`reorder`	optionally reorder the variables in the residuals matrix to show patterns
`...`	Optional parameters

Value

- matrix of residuals

References

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

See Also

Other Reporting functions: `RMSEA()`, `RMSEA.MxModel()`, `RMSEA.summary.mxmodel()`, `extractAIC.MxModel()`, `loadings()`, `loadings.MxModel()`, `tmx_show()`, `tmx_show.MxMatrix()`, `umxCI()`, `umxCI_boot()`, `umxConfint()`, `umxExpCov()`, `umxExpMeans()`, `umxFitIndices()`, `umxRotate()`, `umxSummaryACE_DE()`

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)
```

```
# ==================================
# = Show the residuals of the model =
# ==================================
residuals(m1)
# |   |x1   |x2    |x3   |x4    |x5 |
# |:--|:----|:-----|:----|:-----|:--|
# |x1 |.    |.     |0.01 |.     |.  |
# |x2 |.    |.     |0.01 |-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

## End(Not run)
```

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(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

RMSEA.MxModel *RMSEA function for MxModels*

Description

Return RMSEA and its confidence interval on a model. RMSEA(tmp, silent=TRUE)$RMSEA

Usage

```
## S3 method for class 'MxModel'
RMSEA(x, ci.lower = 0.025, ci.upper = 0.975, digits = 3)
```

Arguments

x	an OpenMx::mxModel() from which to get RMSEA
ci.lower	the lower CI to compute (only 95%, i.e., .025 supported)
ci.upper	the upper CI to compute (only 95%, i.e., .975 supported)
digits	digits to show (default = 3)

Value

- object containing the RMSEA, lower and upper bounds, and p-close

References

https://github.com/tbates/umx

See Also

Other Reporting functions: RMSEA(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
RMSEA(m1)

x = RMSEA(m1)
x$RMSEA # 0.0309761

# Raw: needs to be run by umx to get RMSEA
m2 = umxRAM("One Factor", data = demoOneFactor,
 umxPath("G", to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = "G")
)
RMSEA(m2)

## End(Not run)
```

RMSEA.summary.mxmodel *RMSEA function for MxModel summary*

Description

Extract the RMSEA and confidence interval from a model summary and returns it as an RMSEA object. To report just the RMSEA, you can use `RMSEA(model)$RMSEA`

Usage

```
## S3 method for class 'summary.mxmodel'
RMSEA(x, ci.lower = 0.025, ci.upper = 0.975, digits = 3)
```

Arguments

x	an `OpenMx::mxModel()` summary from which to get RMSEA
ci.lower	the lower CI to compute (only 95% CI (.025) is implemented)
ci.upper	the upper CI to compute (only 95% CI (.975) is implemented)
digits	The number of digits to round data (defaults to 3)

Value

- object containing the RMSEA and lower and upper bounds

References

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

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor[1:100,], type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)
tmp = summary(m1)
RMSEA(tmp)

## End(Not run)
```

SE_from_p *Compute an SE from a beta and p value*

Description

SE_from_p takes beta and p, and returns an SE.

Usage

```
SE_from_p(beta = NULL, p = NULL, SE = NULL, lower = NULL, upper = NULL)
```

Arguments

beta	The effect size
p	The p-value for the effect
SE	Standard error
lower	Lower CI
upper	Upper CI

Value

- Standard error

See Also

- umxAPA()

Other Miscellaneous Stats Functions: FishersMethod(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
SE_from_p(beta = .0020, p = .780)
SE_from_p(beta = .0020, p = .01)
SE_from_p(beta = .0020, SE = 0.01)
umxAPA(.0020, p = .01)
```

---

tmx_genotypic_effect *Graphical display of genotypic effects.*

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:

Adapted from Mather and Jinks, 1977, p. 32). See book issue old-style nomenclature https://github.com/tbates/BGBook/issue

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)

Value

- optional plot

References

Neale, M. C. (2005). Quantitative Genetics. In Encyclopedia of Life Sciences. New York: John Wiley & Sons, Ltd. pdf

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 = p + ggplot2::geom_point()
# p + ggplot2::geom_text(hjust = 0, nudge_x = 0.05, label= "x")
# 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 an ORTHOGONAL factor model is identified by establishing if the number of variables is equal-to or greater than the number of model parameters.

This standard formula to check if the degrees of freedom for an EFA with **orthogonal** factors are greater than or equal to zero. See also `OpenMx::mxCheckIdentification()` for checking model identification.

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://tbates.github.io

See Also

- `OpenMx::mxCheckIdentification()`

Other Teaching and testing Functions: `tmx_genotypic_effect()`, `umx`

Examples

```
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 models in a easy-to-learn-from format.*

Description

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

Usage

```
tmx_show(
  x,
  what = c("values", "free", "labels", "nonzero_or_free"),
  show = c("free", "fixed", "all"),
  matrices = c("S", "A", "M"),
  digits = 2,
  report = c("html", "markdown"),
  na.print = "",
  zero.print = ".",
  html_font = NULL,
 style = c("paper", "material_dark", "classic", "classic_2", "minimal", "material"),
  bootstrap_options = c("hover", "bordered", "condensed", "responsive"),
  lightable_options = "striped",
  freeColor = c("black", "#AAAAAA")
)
```

Arguments

x	an object e.g. umxRAM() umxMatrix() 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 = ".")
html_font	Default is null. Set (e.g. "Optima") to override the style's default font.
style	The style for the table (Defaults to "paper". Other options are "material_dark", "classic", "classic_2", "minimal", "material")
bootstrap_options	border etc. Defaults to c("hover", "bordered", "condensed", "responsive")
lightable_options	Default is "striped"
freeColor	Default is "black" for free, fixed is gray "#AAAAAA"

Value

None

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

---

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

Description

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

Usage

```
## S3 method for class 'MxMatrix'
tmx_show(
  x,
  what = c("values", "free", "labels", "nonzero_or_free"),
  show = c("free", "fixed", "all"),
  matrices = c("S", "A", "M"),
  digits = 2,
  report = c("html", "markdown"),
  na.print = "",
  zero.print = ".",
  html_font = NULL,
 style = c("paper", "material_dark", "classic", "classic_2", "minimal", "material"),
  bootstrap_options = c("hover", "bordered", "condensed", "responsive"),
  lightable_options = "striped",
  freeColor = c("green", "red")
)
```

Arguments

x	an object e.g. umxRAM() umxMatrix() 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 = ".")
html_font	Default is null. Set (e.g. "Optima") to override the style's default font.
style	The style for the table (Defaults to "paper". Other options are "material_dark", "classic", "classic_2", "minimal", "material")
bootstrap_options	border etc. Defaults to c("hover", "bordered", "condensed", "responsive")
lightable_options	Default is "striped"
freeColor	Default is green free red fixed.

Value

None

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
nameStr = c('x1', 'x2', 'g')
amat = mxMatrix(type='Full', name='demo', nrow=2, ncol=3,
  values=c(0, 0, 1,
           0, 0, 0),
  labels=c(NA   , NA, 'alice',
           "bob", NA, 'bob'),
  free=c(TRUE, FALSE, TRUE,
         TRUE, TRUE , TRUE),
  byrow=TRUE, dimnames=list(nameStr[1:2], nameStr)
)
amat
tmx_show(amat, freeColor = c("green","red")) #green=#26D71E red=#D7261E
tmx_show(amat, report = "markdown")
tmx_show(amat, "labels", report = "markdown")
tmx_show(amat, "labels", report = "markdown", show= "all")
tmx_show(amat, na.print = "NA")
tmx_show(amat, zero.print = "0.00", freeColor = c("green","red"))
tmx_show(amat, style = "classic_2")
tmx_show(amat, lightable_options = "hover")

## End(Not run)
```

`tmx_show.MxModel` *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

```
## S3 method for class 'MxModel'
tmx_show(
  x,
  what = c("values", "free", "labels", "nonzero_or_free"),
  show = c("free", "fixed", "all"),
  matrices = c("S", "A", "M"),
  digits = 2,
  report = c("html", "markdown"),
  na.print = "",
  zero.print = ".",
  html_font = NULL,
 style = c("paper", "material_dark", "classic", "classic_2", "minimal", "material"),
  bootstrap_options = c("hover", "bordered", "condensed", "responsive"),
  lightable_options = "striped",
  freeColor = c("black", "#AAAAAA")
)
```

Arguments

x	an object e.g. umxRAM() umxMatrix() 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", "M")). Summary / genomic SEM data (modern observedStats): "data.S" / "observed", "data.V" / "asymCov", "data.W" / "useWeight". Legacy OpenMx name "acov" is refused (it meant useWeight, not sampling covariance).
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 = ".")
html_font	Default is null. Set (e.g. "Optima") to override the style's default font.
style	The style for the table (Defaults to "paper". Other options are "material_dark", "classic", "classic_2", "minimal", "material")

`bootstrap_options`
: border etc. Defaults to c("hover", "bordered", "condensed", "responsive")

`lightable_options`
: Default is "striped"

`freeColor`
: Default is "black" for free, fixed is gray "#AAAAAA"

Details

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

Value

None

References

https://tbates.github.io

See Also

Other Teaching and Testing functions: `umxDiagnose()`, `umxPower()`

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("tmx_sh", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

# ============================================
# = Show smart table on the web (the default) =
# ============================================
tmx_show(m1, report = "html")
tmx_show(m1, what = "free", matrices = "thresholds")
tmx_show(m1, zero.print = "-")

tmx_show(m1, report = "markdown")
tmx_show(m1, digits = 3, report = "markdown")
tmx_show(m1, matrices = "S", report = "markdown")
tmx_show(m1, what = "free"  , report = "markdown")
tmx_show(m1, what = "labels", report = "markdown")
tmx_show(m1, what = "free", matrices = "A", report= "markdown")

## End(Not run)
```

tmx_show.MxModelGSEM *Show genomic SEM matrices*

Description

tmx_show method for MxModelGSEM models.

Default matrices: RAM paths (A), residual S, then LDSC data in observedStats (data.S = genetic cov, data.V = sampling cov of vech(S)). Bare "V" is not a RAM matrix; use "data.V" (the RAM method still aliases "V" to asymCov if you pass it). Means (M) are omitted unless requested.

Usage

```
## S3 method for class 'MxModelGSEM'
tmx_show(
  x,
  what = c("values", "free", "labels", "nonzero_or_free"),
  show = c("free", "fixed", "all"),
  matrices = c("A", "S", "data.S", "data.V"),
  ...
)
```

Arguments

x	An MxModelGSEM model.
what	legal options are "values" (default), "free", or "labels".
show	filter on what to show c("all", "free", "fixed").
matrices	to show (default c("A", "S", "data.S", "data.V")).
...	Arguments passed to tmx_show.MxModel().

tvars *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. tvars adds suffixes to names so you can work with that nice short list. So, you provide bmi, and you get back fully specified family-wise names: c("bmi_T1", "bmi_T2")

Usage

```
tvars(varNames, sep = "", suffixes = 1:2, covNames = NULL, prefix = NULL)

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:

```
tvars(c("var1", "var2"), suffixes = c("_T1", "_T2"))
```

Method 2: *Use sep and a suffix vector.*

Alternatively, you can use sep to add a constant like "_T" after each basename, along with a vector of suffixes. This has the benefit of showing what is varying: This is then suffixed with e.g. "1", "2".

```
tvars(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:

```
`tvars(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")

See Also

namez() umx_explode_twin_names()

Other String Functions: umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

Examples

```
# two styles doing the same thing: first is more general
umx_paste_names("bmi", suffixes = c("_T1", "_T2"))
tvars("bmi", sep = "_T", suffixes = 1:2)
varNames = tvars(c("N", "E", "O", "A", "C"), "_T", 1:2)
tvars(c("IQ", "C"), cov = c("age"), sep = "_T", suffixes = 1:2)
tvars(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 = "age", sep = "_T", prefix= "mean_")
tvars("IQ")
```

---

umx *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 (so if you know a function similar to what you are looking for, look at other members of its "family" at the bottom of its help file.

Please cite as: Bates, T. C., Neale, M. C., & Maes, H. H. (2019). umx: A library for Structural Equation and Twin Modelling in R. *Twin Research and Human Genetics*, **22**, 27-41. doi:10.1017/thg.2019.2. and Castro-de-Araujo, L. F. S., Gillespie, N. A., Neale, M. C., & Bates, T. C. (2026). umx version 4.5: Extending Twin and Path-Based SEM in R with CLPM, MR-DoC, Definition Variables, Onyx Integration, and Censored Distributions. *Twin Research and Human Genetics*, 1-6. doi:10.1017/thg.2026.10056.

All 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

(Only) if you want the bleeding-edge version:

devtools::install_github("tbates/umx")

Author(s)

Maintainer: Timothy C. Bates <timothy.c.bates@gmail.com> (ORCID)

Authors:

- Timothy C. Bates <timothy.c.bates@gmail.com> (ORCID)

Other contributors:

- Luis De Araujo <ldearaujo@unimelb.edu.au> [contributor]
- Nathan Gillespie <nathan.gillespie@vcuhealth.org> [witness]
- Hermine Maes <hmeaes@vcu.edu> [contributor]
- Michael C. Neale <neale@vcu.edu> [contributor]
- Joshua N. Pritikin <jpritikin@pobox.com> [contributor]
- Brenton Wiernik <wiernik@umn.edu> [contributor]
- Michael Zakharin <s1775682@sms.ed.ac.uk> [witness]

References

- Bates, T. C., Neale, M. C., & Maes, H. H. (2019). umx: A library for Structural Equation and Twin Modelling in R. *Twin Research and Human Genetics*, **22**, 27-41. doi:10.1017/thg.2019.2,
- Castro-de-Araujo, L. F. S., Gillespie, N. A., Neale, M. C., & Bates, T. C. (2026). umx version 4.5: Extending Twin and Path-Based SEM in R with CLPM, MR-DoC, Definition Variables, Onyx Integration, and Censored Distributions. *Twin Research and Human Genetics*, 1-6. doi:10.1017/thg.2026.10056,

See Also

Useful links:

- https://github.com/tbates/umx#readme
- Report bugs at https://github.com/tbates/umx/issues

Other Core Model Building Functions: umxLISREL(), umxMatrix(), umxModify(), umxPath(), umxRAM(), umxRAM_DE(), umxSuperModel()

Other Model Summary and Comparison: umxCompare(), umxEquate(), umxMI(), umxReduce(), umxSetParameters(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Other Super-easy helpers: umxEFA(), umxTwoStage()

Other Twin Modeling Functions: power.ACE.test(), umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(),

umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Other Twin Data functions: umx_long2wide(), umx_make_TwinData(), umx_make_twin_data_nice(), umx_residualize(), umx_scale_wide_twin_data(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Other Teaching and testing Functions: tmx_genotypic_effect(), tmx_is.identified()

Other Get and set: umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Other Check or test: umx_check_names(), umx_is_class(), umx_is_endogenous(), umx_is_exogenous(), umx_is_numeric(), umx_is_ordered()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Other File Functions: dl_from_dropbox(), umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_move_file(), umx_open(), umx_rename_file(), umx_write_to_clipboard()

Other String Functions: tvars(), umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Other datasets: Anthro_LDSC, Fischbein_wt, GFF, HSwls, Psych_LDSC, docData, iqdat, us_skinfold_data

Other Advanced Model Building Functions: umxAlgebra(), umxFixAll(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

```
## Not run:
require("umx")
data(demoOneFactor)
manifests <- names(demoOneFactor)
m1 <- umxRAM("One Factor",
    data = demoOneFactor, type = "cov",
```

```
    umxPath("G", to = manifests),
    umxPath(var = manifests),
    umxPath(var = "G", fixedAt = 1)
)

# umx added informative labels, created starting values,
# Ran your model (if autoRun is on), and displayed a brief summary
# including a comparison if you modified a model...!

# umxSummary generates journal-ready fit information.
# We can choose std=T for standardized parameters and can also
# filter out some types of parameter (e.g. means or residuals)

umxSummary(m1, std = TRUE, residuals = FALSE)

# parameters() flexibly retrieves model coefficients.
# For example just G-loadings greater than |.3| and rounded to 2-digits.
parameters(m1, thresh = "above", b = .3, pattern = "G_to.*", digits = 2)

# (The built-in coef works as for lm etc.)
coef(m1)

# ==================
# = Model updating =
# ==================
# umxModify modifies, renames, re-runs, and compares a model
# Can we set the loading of x1 on G to zero? (nope...)
m2 <- umxModify(m1, "G_to_x1", name = "no_effect_of_g_on_X1", comparison = TRUE)

# note1: umxSetParameters can do this with some additional flexibility
# note2 "comparison = TRUE" above is the same as calling
# umxCompare, like this
umxCompare(m1, m2)


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

# umxSummary() will show these, but you can also use the confint() function
confint(m1) # OpenMx's SE-based confidence intervals


# umxConfint formats everything nicely, and allows adding CIs (with parm=)
umxConfint(m1, parm = "all", run = TRUE) # likelihood-based CIs

# And make a Figure and open in browser
plot(m1, std = TRUE)

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

## End(Not run)
```

umxACE *Build and run a 2-group Cholesky ACE twin model (univariate or multivariate)*

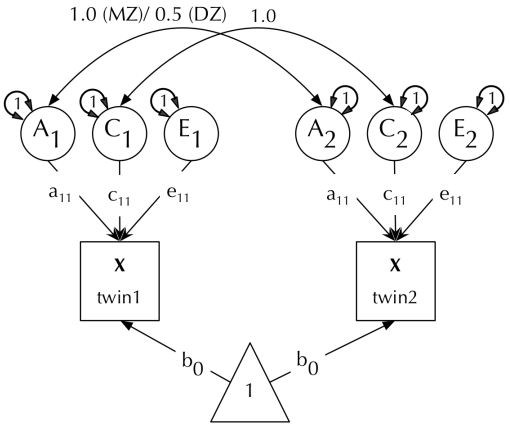
Description

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.

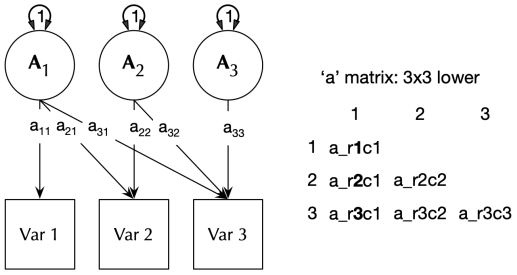
umxACE implements a 2-group model to capture these data and represent the phenotypic variance as a sum of Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

The following figure shows the ACE model for one variable "x" as a path diagram:



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

The Cholesky decomposition creates as many latent A (and C and E) latent variables as there are phenotypes, and, moving from left to right, decomposes the variance in each phenotype into successively restricted factors. The following figure shows the multivariate ACE model for three variables:



In this ACE model of three phenotypes, the expected variance-covariance matrix of the original data is the product of each lower Cholesky and its transform (i.e., `A = a %*% t(a)` summed for `A+C+E`.

This lower-triangle decomposition feature of the Cholesky yields a model which is certain to both account for all the variance (with some restrictions) in the data and be solvable.

This figure also contains the key to understanding how to modify models that `umxACE` produces using `umxModify()` to drop paths by label like `"a_r1c1"`. **n.b.**: 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 = NULL,
  mzData = NULL,
  sep = NULL,
  data = NULL,
  zyg = "zygosity",
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  numObsDZ = NULL,
  numObsMZ = NULL,
  boundDiag = 0,
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  intervals = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  residualizeContinuousVars = FALSE,
  nSib = 2,
  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.
data	If provided, dzData and mzData are treated as levels of zyg to select() MZ and DZ data sets (default = NULL)
zyg	If data provided, this column is used to select rows by zygosity (Default = "zygosity")
type	Analysis method one of c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS")
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
allContinuousMethod	"cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.
autoRun	Whether to run the model (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: "ordinal", "search"
optimizer	Optionally set the optimizer (default NULL does nothing).
residualizeContinuousVars	Not yet implemented.
nSib	Number of siblings in a family (default - 2). "3" = extra sib.
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).
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).
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).

Details

Covariates umxACE handles covariates by modelling them in the means. 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 can waste data. See also umx_residualize().

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% by twins in a pair), 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

- `OpenMx::mxModel()` of subclass mxModel.ACE

References

Eaves, L. J., Last, K. A., Young, P. A., & Martin, N. G. (1978). Model-fitting approaches to the analysis of human behaviour. *Heredity*, **41**, 249-320. doi:10.1038/hdy.1978.101

See Also

- `umxPlotACE()`, `umxSummaryACE()`, `power.ACE.test()`, `umxModify()`

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
require(umx)
# ============================
# = How heritable is height? =
# ============================

# 1. Height in meters has a tiny variance, and this makes optimising hard.
#    We'll scale it by 10x to make the Optimizer's task easier.
data(twinData) # ?twinData from Australian twins.
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 10

# 2. Make mz & dz data.frames (no need to select variables: umx will do this)
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]

# 3. Built & run the model, controlling for age in the means model
m1 = umxACE(selDVs = "ht", selCovs = "age", sep = "", dzData = dzData, mzData = mzData)

# sidebar: umxACE figures out variable names using sep:
#    e.g. selVars = "wt" + sep= "_T" -> "wt_T1" "wt_T2"

umxSummary(m1, std = FALSE) # un-standardized

# tip 1: set report = "html" and umxSummary prints the table to your browser!
# tip 2: plot works for umx: Get a figure of the model and parameters
# plot(m1) # Also, look at the options for ?plot.MxModel.

# ========================================
# = Test ADE, AE, CE, E, and generate table =
# ========================================

umxReduce(m1, report="html", silent= TRUE)

# ============================
# = Model, with 2 covariates =
# ============================

# Create another covariate: cohort
twinData$cohort1 = twinData$cohort2 =twinData$part
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]

# 1. def var approach
m2 = umxACE(selDVs = "ht", selCovs = c("age", "cohort"), sep = "", dzData = dzData, mzData = mzData)

# 2. Residualized approach: remove height variance accounted-for by age.
FFdata = twinData[twinData$zygosity %in% c("MZFF", "DZFF"), ]
FFdata = umx_residualize("ht", "age", suffixes = 1:2, data = FFdata)
mzData = FFdata[FFdata$zygosity %in% "MZFF", ]
dzData = FFdata[FFdata$zygosity %in% "DZFF", ]
m3 = umxACE(selDVs = "ht", sep = "", dzData = dzData, mzData = mzData)
```

```
# ============================================================
# = ADE: Evidence for dominance ? (DZ correlation set to .25) =
# ============================================================
m2 = umxACE(selDVs = "ht", 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
#     matrices are still called a, c & e.

# tip: try umxReduce(m1) to automatically build and compare ACE, ADE, AE, CE
# including conditional probabilities!

# ================================================
# = WLS example using diagonal weight least squares =
# ================================================

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


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

# Things to note:

# 1. Weight has a large variance, and this makes solution finding very hard.
# Here, we residualize the data for age, which also scales weight and height.

data(twinData)
tmp = umx_residualize(c("wt", "ht"), cov = "age", suffixes= c(1, 2), data = twinData)
mzData = tmp[tmp$zygosity %in% "MZFF", ]
dzData = tmp[tmp$zygosity %in% "DZFF", ]

# tip: You might also want transform variables
# tmp = twinData$wt1[!is.na(twinData$wt1)]
# car::powerTransform(tmp, family="bcPower"); hist(tmp^-0.6848438)
# twinData$wt1 = twinData$wt1^-0.6848438
# twinData$wt2 = twinData$wt2^-0.6848438

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

m2 = umxACE(selDVs = "wt", dzData = dzData, mzData = mzData, sep = "", boundDiag = NULL)

# A short cut (which is even shorter for "_T" twin data with "MZ"/"DZ" data in zygosity column is:
m1 = umxACE(selDVs = "wt", sep = "", data = twinData,
 dzData = c("DZMM", "DZFF", "DZOS"), mzData = c("MZMM", "MZFF"))
# |   |   a1|c1 |   e1|
# |:--|----:|:--|----:|
# |wt | 0.93|.  | 0.38|
```

```
# tip: umx_make_twin_data_nice() will make data into this nice format for you!

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

m2 = umxModify(m1, update = "c_r1c1", name = "no_C", comparison = TRUE)
#*tip* call umxModify(m1) with no parameters, and it will print the labels available to fix!
# nb: You can see parameters of any model with parameters(m1)

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

# ====================================
# = Bivariate height and weight model =
# ====================================
data(twinData)
# We'll scale height (ht1 and ht2) and weight
twinData = umx_scale_wide_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"), ]
m1 = umxACE(selDVs = c("ht", "wt"), sep = '', dzData = dzData, mzData = mzData)
umxSummary(m1)

# ==================
# = Ordinal example =
# ==================

# Prep data
require(umx)
data(twinData)
# Cut BMI column to form ordinal obesity variables
obLevels = c('normal', 'overweight', 'obese')
cuts = quantile(twinData[, "bmi1"], 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] = umxFactor(twinData[, ordDVs])

mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]

# Model and summary!
m1 = umxACE(selDVs = "obese", dzData = dzData, mzData = mzData, sep = '')

# And controlling age (otherwise manifests appearance as latent C)
m1 = umxACE(selDVs = "obese", selCov= "age", dzData = dzData, mzData = mzData, sep = '')
# umxSummary(m1)
```

```
# ==========================================
# = Bivariate continuous and ordinal example =
# ==========================================
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[, "bmi1"], 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] = umxFactor(twinData[, ordDVs])
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='')

# And controlling age
m1 = umxACE(selDVs = c("wt","obese"), selCov= "age", 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 20% obese subjects
# and make into mxFactors (ensure ordered is TRUE, and require levels)
obLevels   = c('normal', 'obese')
cuts       = quantile(twinData[, "bmi1"], probs = .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)
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",]
m1 = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = '')
umxSummary(m1)

# ==============
# = Two binary =
# ==============
require(umx)
data(twinData)
htLevels   = c('short', 'tall')
obLevels   = c('normal', 'obese')
cuts       = quantile(twinData[, "bmi1"], probs = .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)
ordDVs = c("obese1", "obese2")
twinData[, ordDVs] = umxFactor(twinData[, ordDVs])

twinData$short1 = cut(twinData$ht1, breaks=c(-Inf,1.6,Inf), labels=htLevels)
twinData$short2 = cut(twinData$ht2, breaks=c(-Inf,1.6,Inf), labels=htLevels)
ordDVs = c("short1", "short2")
twinData[, ordDVs] = umxFactor(twinData[, ordDVs])

mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
m1 = umxACE(selDVs = c("short", "obese"), dzData = dzData, mzData = mzData, sep = '')

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

require(umx)
data(twinData)
twinData= umx_scale_wide_twin_data(data=twinData, varsToScale= "wt", sep="")
selDVs = c("wt1", "wt2")
mz = cov(twinData[twinData$zygosity %in%  "MZFF", selDVs], use = "complete")
dz = cov(twinData[twinData$zygosity %in%  "DZFF", selDVs], use = "complete")
m1 = umxACE(selDVs=selDVs, dzData=dz, mzData=mz, numObsDZ=569, numObsMZ=351)
umxSummary(m1)
plot(m1)
```

---

umxACEcov *Run a Cholesky with covariates that are random (in the expected covariance matrix)*

Description

Often, researchers include covariates in 2-group Cholesky umxACE() twin models. 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. Binary and ordinal covariates like sex also violate the normality assumption. Which is most of the use cases :-(.

Usage

```
umxACEcov(
  name = "ACEcov",
  selDVs,
  selCovs,
  dzData,
```

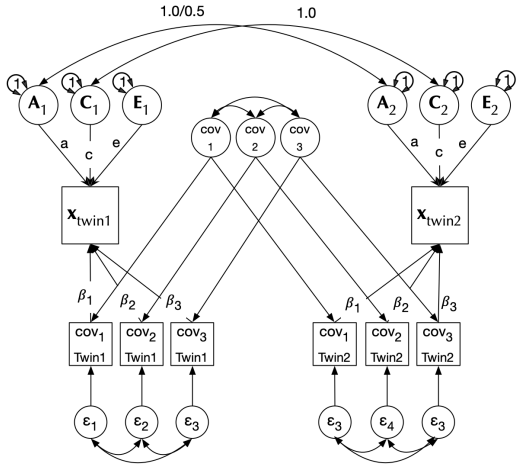
```
  mzData,
  sep = NULL,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals"),
  dzAr = 0.5,
  dzCr = 1,
  addStd = TRUE,
  addCI = TRUE,
  boundDiag = 0,
  equateMeans = TRUE,
  bVector = FALSE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
optimizer	optionally set the optimizer. Default (NULL) does nothing.

Details

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



Value

- `OpenMx::mxModel()` of subclass mxModel.ACEcov

References

- Neale, M. C., & Martin, N. G. (1989). The effects of age, sex, and genotype on self-report drunkenness following a challenge dose of alcohol. *Behavior Genetics*, **19**, 63-78. doi:10.1007/BF01065884.
- Schwabe, I., Boomsma, D. I., Zeeuw, E. L., & Berg, S. M. (2015). A New Approach to Handle Missing Covariate Data in Twin Research : With an Application to Educational Achievement Data. *Behavior Genetics*, **46**, 583-95. doi:10.1007/s1051901597711.

See Also

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
## Not run:
# ============================================
# = BMI, can't use Age as a random covariate =
# ============================================
require(umx)
data(twinData)
```

```
# Replicate age to age1 & age2
twinData$age1 = twinData$age2 = twinData$age
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")

# ====================================================================
# = Trying to use identical var (like age) as a random cov is ILLEGAL =
# ====================================================================
m1 = umxACEcov(selDVs = "bmi", selCovs = "age", dzData = dzData, mzData = mzData, sep = "")

# ======================================================
# = Use lm-based age residualisation approach instead =
# ======================================================

resid_data = umx_residualize("bmi", "age", suffixes = 1:2, twinData)
mzData = subset(resid_data, zygosity == "MZFF")
dzData = subset(resid_data, zygosity == "DZFF")
m2     = umxACE("resid", selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")

# Univariate BMI without covariate of age for comparison
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m3 = umxACE("raw_bmi", selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")

# ============================================================================
# = A bivariate example (need a dataset with a VIABLE COVARIATE to do this) =
# ============================================================================
selDVs  = "wt" # Set the DVs
selCovs = "ht" # Set the COV
selVars = umx_paste_names(selDVs, covNames = selCovs, sep = "", suffixes=1:2)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACEcov(selDVs = selDVs, selCovs = selCovs,
   dzData = dzData, mzData = mzData, sep = "", autoRun = TRUE
)

## End(Not run)
```

---

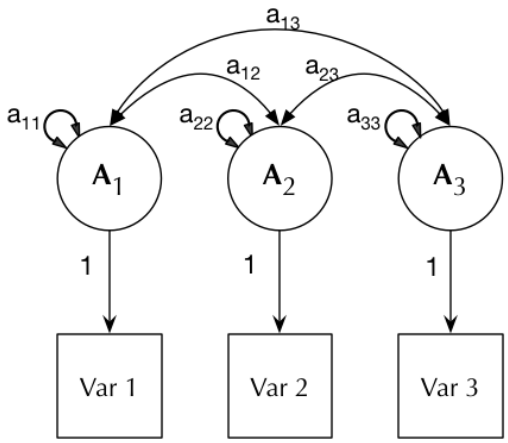
umxACEv *Build and run 2-group uni- or multi-variate ACE models based on VARIANCE (not paths).*

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. umxACEv directly estimates variance components (rather than paths, which are then squared to produce variance and therefore cannot be negative). It offers better power, correct Type I error and un-biased estimates (with no zero-bound for the variances) as a saturated model. (Verhulst et al, 2019).

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).

Usage

```
umxACEv(
  name = "ACEv",
  selDVs,
  selCovs = NULL,
  sep = NULL,
  dzData,
  mzData,
  dzAr = 0.5,
  dzCr = 1,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals"),
  data = NULL,
  zyg = "zygosity",
  weightVar = NULL,
  numObsDZ = NULL,
  numObsMZ = NULL,
  addStd = TRUE,
  addCI = TRUE,
  boundDiag = NULL,
  equateMeans = TRUE,
```

```
  bVector = FALSE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  nSib = 2
)
```

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.
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).
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.
data	If provided, dzData and mzData are treated as valid levels of zyg to select() data sets (default = NULL)
zyg	If data provided, this column is used to select rows by zygosity (Default = "zygosity")
weightVar	= If provided, a vector objective will be used to weight the data. (default = NULL).
numObsDZ	= Number of DZ twins: Set this if you input covariance data.
numObsMZ	= Number of MZ twins: Set this if you input covariance data.
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	= Numeric lbound for diagonal of the a, c, and e matrices. Default = NULL (no bound)
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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
optimizer	Optionally set the optimizer (default NULL does nothing).
nSib	Number of sibs, default is 2. Working on 3 :-)

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).

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% by twins in a pair), to .25 to model dominance effects.

note: Only one of C or D may be estimated simultaneously. This restriction reflects the lack of degrees of freedom to simultaneously model C and D with only MZ and DZ twin pairs (Eaves et al. 1978 p267).

Value

- `OpenMx::mxModel()` subclass `mxModelACEv`

References

- Verhulst, B., Prom-Wormley, E., Keller, M., Medland, S., & Neale, M. C. (2019). Type I Error Rates and Parameter Bias in Multivariate Behavioral Genetic Models. *Behav Genet*, **49**, 99-111. doi:10.1007/s105190189942y
- Eaves, L. J., Last, K. A., Young, P. A., & Martin, N. G. (1978). Model-fitting approaches to the analysis of human behaviour. *Heredity*, **41**, 249-320. doi:10.1038/hdy.1978.101

See Also

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
## Not run:
# ==============================
# = 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.

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

# A short cut (which is even shorter for "_T" twin data with "MZ"/"DZ" data in zygosity column is:
m1 = umxACEv(selDVs = "wt", sep = "", dzData = "MZFF", mzData = "DZFF", data = twinData)
# ========================================================
# = Evidence for dominance ? (DZ correlation set to .25) =
# ========================================================
m2 = umxACEv("ADE", selDVs = "wt", 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)
# Load data and rescale height to cm (var in m too small)
data(twinData) # ?twinData from Australian twins.
twinData[,c("ht1", "ht2")]= twinData[,c("ht1", "ht2")]*100

mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACEv(selDVs = "ht", sep = "", dzData = dzData, mzData = mzData)

umxSummary(m1, std = FALSE) # unstandardized
plot(m1)

# tip: with report = "html", umxSummary can print the table to your browser!
# tip: You can turn off auto-plot with umx_set_auto_plot(FALSE)

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

```
# ========================================================
m2 = umxACEv("ADE", selDVs = "ht", dzCr = .25, sep="", dzData = dzData, mzData = mzData)
umxCompare(m2, m1) # Is ADE better?
umxSummary(m2, comparison = m1) # nb: though this is ADE, matrices are still called A,C,E

# We can modify this model, dropping shared environment, and see a comparison:
m3 = umxModify(m2, update = "C_r1c1", comparison = TRUE, name = "AE")

# ====================================
# = Bivariate height and weight model =
# ====================================

data(twinData)
twinData[,c("ht1", "ht2")]= twinData[,c("ht1", "ht2")]*100
mzData = twinData[twinData$zygosity %in% c("MZFF", "MZMM"), ]
dzData = twinData[twinData$zygosity %in% c("DZFF", "DZMM", "DZOS"), ]
m1 = umxACEv(selDVs = c("ht", "wt"), sep = '', dzData = dzData, mzData = mzData)

# ===================
# = Ordinal example =
# ===================
require(umx)
data(twinData)

# Cut bmi column to form ordinal obesity variables
cutPoints = quantile(twinData[, "bmi1"], probs = c(.5, .2), na.rm = TRUE)
obesityLevels = c('normal', 'overweight', '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)

# Make the ordinal variables into mxFactors (ensure ordered is TRUE, and require levels)
twinData[, c("obese1", "obese2")] = umxFactor(twinData[, c("obese1", "obese2")])
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m2 = umxACEv(selDVs = "obese", dzData = dzData, mzData = mzData, sep = '')

# FYI: Show mz, dz, and t1 and t2 have the same levels!
str(mzData)

# ============================================
# = Bivariate continuous and ordinal example =
# ============================================
data(twinData)
# 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 ordered mxFactors
twinData[, ordDVs] = umxFactor(twinData[, ordDVs])
```

```
# umxACEv can trim out unused variables on its own
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]

m1 = umxACEv(selDVs = c("wt", "obese"), dzData = dzData, mzData = mzData, sep = '')
plot(m1)

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

selDVs = c("wt", "obese")
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACEv(selDVs = selDVs, dzData = dzData, mzData = mzData, sep = '')
umxSummary(m1)

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

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

## End(Not run)
```

---

umxACE_DE *Build a Double-Entry Censored Data ACE twin model*

Description

umxACE_DE implements a twin model for handling censored/floor-effect variables using a "double-entry" strategy. Each censored variable is represented by a pair of columns: one continuous (holding

non-censored values) and one binary/ordinal (holding the censored indicator). The path coefficients (A, C, E) are constrained to be equal across the two paired columns.

Fully continuous variables may be mixed with double-entry pairs in `selDVs` (e.g. `c("ht", "wt_cont", "wt_cens")`). Prepare the censored traits with `umx_make_double_entry_data()`. At least one contiguous `_cont`/`_cens` pair is required; for all-continuous models use `umxACE()`.

"Trust the Science" note

If you know the censoring value, keep fixCensorThresholds = "yes" / "auto" to lock that in. Estimated-free threshold can "wander" wildly. Reserve it only for unknown cut, and even then treat estimated thresholds with suspicion. Free threshold will "run" but it is **not great** at recovering the true floor/threshold.

Usage

```
umxACE_DE(
  name = "ACE_DE",
  selDVs,
  selCovs = NULL,
  dzData = NULL,
  mzData = NULL,
  sep = "_T",
  data = NULL,
  zyg = "zygosity",
  fixCensorThresholds = c("yes", "auto", "no"),
  censorCuts = NULL,
  doubleEntrySuffix = c("_cont", "_cens"),
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  numObsDZ = NULL,
  numObsMZ = NULL,
  boundDiag = 0,
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  intervals = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  nSib = 2,
  dzAr = 0.5,
  dzCr = 1,
  weightVar = NULL,
  equateMeans = TRUE,
  addStd = TRUE,
  addCI = TRUE
)
```

Arguments

`name`	The name of the model (defaults to "ACE").
`selDVs`	Base names of variables to model. Include fully observed continuous traits by base name (e.g. `"ht"`). Include each censored trait as an adjacent pair

of prepped names (e.g. "wt_cont", "wt_cens"). Prep censored data with umx_make_double_entry_data() first.

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".

data If provided, dzData and mzData are treated as levels of zyg to select() MZ and DZ data sets (default = NULL)

zyg If data provided, this column is used to select rows by zygosity (Default = "zygosity")

fixCensorThresholds
One of "yes" (default), "auto", or "no". "yes": fix every double-entry pair from censorCuts and/or prep attribute umxDoubleEntry. "auto": fix only pairs with a finite known cut in censorCuts or the prep attribute. "no": free binary thresholds.

censorCuts Optional named numeric vector of known cuts on the **analysis scale** (after any scaling). Names may be trait base ("wt"), continuous base ("wt_cont"), or censored base ("wt_cens"). When non-NULL, fixCensorThresholds must be "yes" or "auto"; only the named pairs are fixed (partial censorCuts is allowed).

doubleEntrySuffix
Suffixes for the continuous and censored variables (default = c("_cont", "_cens")).

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

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.

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

autoRun Whether to run the model (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: "ordinal", "search"

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

nSib Number of siblings in a family (default = 2).

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).

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).

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).

Details

Double-entry modeling represents a floor- or ceiling-censored trait using an adjacent pair of manifest columns: one continuous (_cont) and one binary/ordinal factor (_cens).

Likelihood Evaluation & Missingness Structure: Each individual contributes exactly one non-missing likelihood element for the censored trait:

- **Non-censored cases** ($x > cut$): _cont contains the observed numeric value evaluated via the continuous normal density $f(x)$, while _cens is set to NA.
- **Censored cases** ($x \leq cut$): _cont is set to NA, while _cens contains the ordinal factor level evaluated via the cumulative threshold probability $P(Y \leq \tau)$.

Leaving _cens non-missing for observed continuous rows would double-count the tail density (evaluating both $f(x)$ and $P(Y > \tau)$ for the same individual), introducing artificial covariance dependencies and inflating density estimates.

Value

- OpenMx::mxModel() of subclass mxModel.ACE

See Also

- umx_make_double_entry_data(), umxACE(), plot(), umxSummary(), umxModify(), umxCompare()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
require(umx)

################################
# Toy example 1: Threshold known #
################################

# Weighing scales lowest value is 60kg, with lower values registering as 60kg.

# NOTE: We first put height on a scale comparable to weight to ease estimation
data(twinData)
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] *100 # metre->cm

clinic = twinData
clinic$wt1[!is.na(twinData$wt1) & (twinData$wt1 <= 60)] = 60
clinic$wt2[!is.na(twinData$wt2) & (twinData$wt2 <= 60)] = 60

# Double-entry prep for weight (floor at 0) Creates "wt_cont*" and "wt_cens*"
prep = umx_make_double_entry_data(clinic, cols = list(wt = "<= 60"), sep = "")
```

```
mzData = prep[prep$zygosity %in% "MZFF", ]
dzData = prep[prep$zygosity %in% "DZFF", ]


# 1. Correct mixed model: continuous height + double-entry censored weight
# Default: free thresholds. For known LOD, use fixCensorThresholds = "auto" or "yes"
# with censorCuts = c(wt = 0) (cut on analysis scale; means cont=cens equated).

mDE = umxACE_DE(name = "htWtDE", mzData = mzData, dzData = dzData, sep = "",
 selDVs = c("ht", "wt_cont", "wt_cens"), tryHard = "yes")

# Table: Model Fit Summary for 'htWtDE'
#
# |Model  | EP|  -2LL|   df|   AIC|      BIC|
# |:------|--:|-----:|----:|-----:|--------:|
# |htWtDE | 11| 37082| 7781| 37104| 37165.51|


# Table: Standardized parameter estimates from a 2-trait double-entry Cholesky ACE model.
# A: additive genetic; C: common environment; E: unique environment.
#
# |        |    a1|a2    |    c1|c2    |     e1|e2    |
# |:-------|-----:|:-----|-----:|:-----|------:|:-----|
# |ht      | 0.874|      | 0.373|      |  0.312|      |
# |wt_cens | 0.437|0.381 | 0.079|0.301 | -0.148|0.738 |
# Double-entry thresholds fixed: wt_cens @ 60 (means equated to wt_cont).


# Table: Means (from model$top$expMean)
#
# |          |     ht1| wt_cont1| wt_cens1|     ht2| wt_cont2| wt_cens2|
# |:---------|-------:|--------:|--------:|-------:|--------:|--------:|
# |intercept | 162.499|   56.961|   56.961| 162.499|   56.961|   56.961|


####################################
# Toy example 2: Threshold unknown #
####################################

# GP only gets paid to record weight for higher-BMI people (~BMI 22 here).
# Others are coded "0").
# NOTE: We don't know what threshold 0 mapped to, so we *estimate* the threshold.
# NOTE: This example is contrived, and creates a nasty collider with height.

# Height fully observed; Weighing scales lowest value is 60kg, with all
# values beneath this registering as 60kg.

# NOTE: First we get the height data on a comparable scale to weight to ease model estimation
data(twinData)
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 100 # metres -> cms

bmiCut = 22 # about the 80th percentile in this sample.
```

```
clinic = twinData
clinic$wt1[!is.na(twinData$wt1) & (twinData$bmi1 <= 22)] = 0
clinic$wt2[!is.na(twinData$wt2) & (twinData$bmi2 <= 22)] = 0

# Double-entry prep for weight (floor at 0) Creates "wt_cont*" and "wt_cens*"
prep = umx_make_double_entry_data(data = clinic, cols = list(wt = 0), sep = "")

mzData = prep[prep$zygosity %in% "MZFF", ]
dzData = prep[prep$zygosity %in% "DZFF", ]

# 1. Correct mixed model: continuous height + double-entry censored weight
# Default: free thresholds. For known LOD, use fixCensorThresholds = "auto" or "yes"
# with censorCuts = c(wt = 0) (cut on analysis scale; means cont=cens equated).

# Note how we set fixCensorThresholds = "no"

mDE = umxACE_DE(name = "htWtDE", mzData = mzData, dzData = dzData, sep = "",
 selDVs = c("ht", "wt_cont", "wt_cens"), fixCensorThresholds="no", tryHard = "yes")

# Table: Model Fit Summary for 'htWtDE'
#
# |Model  | EP|     -2LL|   df|      AIC|      BIC|
# |:------|--:|--------:|----:|--------:|--------:|
# |htWtDE | 12| 32406.96| 7780| 32430.96| 32498.06|
#
#
# Table: Standardized parameter estimates from a 2-trait double-entry Cholesky ACE model.
#   A: additive genetic; C: common environment; E: unique environment.
#
# |        |    a1|a2    |    c1|c2    |    e1|e2    |
# |:-------|-----:|:-----|-----:|:-----|-----:|:-----|
# |ht      | 0.883|      | 0.268|      | 0.385|      |
# |wt_cens | 0.228|0.105 | 0.628|0.235 | 0.180|0.675 |
#
#
# Table: Means (from model$top$expMean)
#
# |          |     ht1| wt_cont1|wt_cens1 |     ht2| wt_cont2|wt_cens2 |
# |:---------|-------:|--------:|:--------|-------:|--------:|:--------|
# |intercept | 162.918|   68.623|0        | 162.918|   68.623|0        |

umxSummary(mDE, std = TRUE)

# 2. Gold standard: uncensored bivariate ACE on true height and weight
mzTrue = twinData[twinData$zygosity %in% "MZFF", ]
dzTrue = twinData[twinData$zygosity %in% "DZFF", ]
mTrue  = umxACE("htWtTrue", selDVs = c("ht", "wt"),
           mzData = mzTrue, dzData = dzTrue, sep = "", tryHard = "yes")

# Table: Standardized parameter estimates from 2-factor Cholesky ACE
#   A: additive genetic; C: common environment; E: unique environment.
# |   |    a1|a2    |     c1|c2 |    e1|e2    |
# |:--|-----:|:-----|------:|:--|-----:|:-----|
```

```
# |ht | 0.899|      |  0.252|   | 0.357|      |
# |wt | 0.419|0.766 | -0.049|.  | 0.074|0.479 |

umxSummary(mTrue, std = TRUE)

# 3. Mistaken analysis: treat 0 (not weighed) as a real continuous weight
mzNaive = clinic[clinic$zygosity %in% "MZFF", ]
dzNaive = clinic[clinic$zygosity %in% "DZFF", ]
mNaive = umxACE("htWtNaive0", selDVs= c("ht", "wt"),
            mzData= mzNaive, dzData= dzNaive, sep ="",tryHard = "yes")
umxSummary(mNaive, std = TRUE)

# Naive mean(wt) pulled toward 0, covariance obscured; prefer mDE when zeros mean censored.
#
# Table: Standardized parameter estimates from a 2-factor Cholesky ACE model.
#   A: additive genetic; C: common environment; E: unique environment.
#
# |   |    a1|a2    |     c1|c2 |     e1|e2    |
# |:--|-----:|:-----|------:|:--|------:|:-----|
# |ht | 0.899|      |  0.252|   |  0.357|      |
# |wt | 0.057|0.717 | -0.238|.  | -0.072|0.649 |
#
#
# Table: Means (from model$top$expMean)
#
# |          |     ht1|    wt1|     ht2|    wt2|
# |:---------|-------:|------:|-------:|------:|
# |intercept | 162.515| 20.984| 162.515| 20.984|
```

---

umxACE_GLM *Cholesky ACE twin model with mxFitFunctionGLM*

Description

RAM MZ/DZ Cholesky ACE for exponential-family phenotypes. Not a patch on `umxACE()`: that function uses `mxExpectationNormal` and a `%*% t(a)` algebras, which GLM cannot use.

Additive and shared factors have variance 1. MZ uses one shared `a_i` and (if `dzCr` is 1) one shared `c_i` per factor: two copies with correlation 1 are singular for Laplace. DZ has `a_i_T1` with `a_i_T2` covariance `dzAr`. Unique `e` is always per twin. Loadings use umxACE labels `a_r{row}c{col}`. Off-diagonal `e` is the GLM analogue of E covariances (shared unique / measurement error). Family leftover is still independent given eta. Unique environment is the `e` latent **and** the family leftover. `nAGQ` is 1. Cholesky is `nVar`-general. MZ Laplace dimension is `4 * nVar` latents.

Bernoulli: fix e diagonal at 1. Leftover (probit 1, logit $\pi^2/3$) already sets the observation scale. Univariate ACE still has three variance components and only two twin correlations, so a free `e_ii` is not identified (`e` went to 0 and `a` was eaten). For binomial items, `e_rjc_j` is fixed at 1 (same idea as fixing unique scale in threshold ACE). Off-diagonal `e` stays free so multivariate unique covariance is not forced to 0. Do **not** set `e11` to leftover: that would double-count unique on eta.

umxSummaryACE_GLM() still uses E = ee' + D for shares. Poisson keeps a free e diagonal (leftover is on the counts). Poisson ACE starts the Cholesky diagonals at $a = \sqrt{0.8}$, $e = \sqrt{0.2}$, c at 0.2 (80/20 A/E on eta; C not pinned to the floor) so the optimizer walks downhill from a genetic model. Free ACE diagonals have lbound = 1e-4 so the derivative stays alive at the floor.

Gamma and inverse Gaussian: not for ACE. Inverse Gaussian is the hitting-time law of Brownian motion with drift (reaction time, age of onset). Gamma is the constant-CV amount / duration model. Both need trial-level data and, for a process interpretation, structure on two parameters (mean and shape/lambda, or drift and barrier with diffusion scale fixed). Twin ACE is one phenotype per person, so those uses are out of scope. The wrapper still accepts the families: leftover shape (Gamma) or lambda (inverse Gaussian) is fixed at 1 (phi = 1/theta) unless you pass a numeric theta; a free theta label is rejected (collides with e). That is not the cognitive-psych constraint of fixing diffusion scale σ. Leave them in the too-hard box; use Poisson for counts and threshold umxACE() for binary.

Binomial trials= is not implemented. mxFamily() can take trials as a raw-data column (n successes out of n_i trials per row). This wrapper does not: there is no trials argument, and MZ/DZ data are subset to the phenotype _T1/_T2 columns, so any n columns are dropped. Binomial here is Bernoulli (n = 1) only. Grouped or item-sum binomials with a dynamic n cannot be wired. Most real binomial counts are overdispersed; there is no beta-binomial (or other extra-binomial leftover) to absorb that, so users will keep using liability umxACE() with more than one threshold, or a Gaussian for large counts. Negative binomial on the counts is the overdispersed alternative when there is no upper bound.

Hessian / standard errors stay off. umxRAM_GLM() (no latents) yields finite SEs. Twin ACE always has several latents. On univariate Poisson, n = 50 pairs gave a Hessian that was not positive definite and SEs all NA (optimizer status 6). At n = 200 the Hessian was positive definite and SEs were finite, but SEs on a and c were about 1 on loadings of about 0.5, and status was still 6. Default remains Calculate Hessian / Standard Errors = No. After a fitted model, family-score SEs: mxComputeOnce('fitfunction', 'information', 'meat') then mxComputeStandardError(). Poisson intercept matches glm(); Poisson ACE intercept matches a pair bootstrap in tests.

Usage

```
umxACE_GLM(
  name = "ACE_GLM",
  selDVs,
  mzData,
  dzData,
  sep = "_T",
  family = stats::poisson(),
  theta = NULL,
  nAGQ = 1L,
  dzAr = 0.5,
  dzCr = 1,
  equateMeans = TRUE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search")
)
```

Arguments

name	model name (default "ACE_GLM")
selDVs	base phenotype names (sep expands to _T1/_T2). Cholesky is nVar-general.
mzData	MZ data.frame
dzData	DZ data.frame
sep	twin separator (default "_T"; must not be "")
family	stats::family or mxFamily() for the phenotype (both twins). Poisson is the intended ACE family. Binomial is Bernoulli only: trials= and per-row n columns are not implemented (see Details). Scale-identified (e diagonal fixed at 1) but leftover eats heritability — use threshold umxACE() for binary. Gamma and inverse Gaussian are accepted but not supported for ACE (trial-level process models; see Details).
theta	passed to mxFamily(). For Gamma, leftover shape; for inverse Gaussian, leftover lambda. Both fixed (default 1, phi = 1). A free-parameter label is rejected (collides with e). Not used for Poisson or binomial.
nAGQ	must be 1 (more than one latent)
dzAr	DZ additive genetic correlation (default 0.5)
dzCr	DZ C correlation (default 1; 0.25 for ADE)
equateMeans	equate T1/T2 intercepts (default TRUE)
autoRun	whether to run
tryHard	xmu_safe_run_summary() tryHard

Value

supermodel of class MxModelACE_GLM

See Also

umxACE(), umxRAM_GLM(), umxSummaryACE_GLM(), umxPlotACE_GLM()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
set.seed(1)
n = 40
a = 0.4; c = 0.2; e = 0.5; b0 = 0.3
simPair = function(rA, n) {
 A = rnorm(n)
 C = rnorm(n)
```

```
 E1 = rnorm(n)
 E2 = rnorm(n)
 A2 = rA * A + sqrt(1 - rA^2) * rnorm(n)
 y1 = rpois(n, lambda = exp(b0 + a * A + c * C + e * E1))
 y2 = rpois(n, lambda = exp(b0 + a * A2 + c * C + e * E2))
 data.frame(y_T1 = y1, y_T2 = y2)
}
mzData = simPair(1, n)
dzData = simPair(0.5, n)
m1 = umxACE_GLM(selDVs = "y", mzData = mzData, dzData = dzData, sep = "_T")

## 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

```
umxAlgebra(
  name = NA,
  expression,
  dimnames = NA,
  ...,
  joinKey = as.character(NA),
  joinModel = as.character(NA),
  verbose = 0L,
  initial = matrix(as.numeric(NA), 1, 1),
  recompute = c("always", "onDemand"),
  fixed = "deprecated_use_recompute"
)
```

Arguments

name	The name of the algebra (Default = NA). Note the different order compared to mxAlgebra!
expression	The algebra
dimnames	Dimnames of the algebra
...	Other parameters
joinKey	See mxAlgebra documentation
joinModel	See mxAlgebra documentation
verbose	Quiet or informative

initial	See mxAlgebra documentation
recompute	See mxAlgebra documentation
fixed	= See mxAlgebra documentation

Value

- OpenMx::mxAlgebra()

See Also

- umxMatrix()

Other Advanced Model Building Functions: umx, umxFixAll(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

```
## Not run:
A = umxMatrix("A", "Full", nrow = 3, ncol = 3, values=2)
B = umxAlgebra("B", A)
C = umxAlgebra(A + B, name = "C")
D = umxAlgebra(sin(C), name = "D")
m1 = mxRun(mxModel("AlgebraExample", A, B, C, D ))
mxEval(D, m1)

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

## End(Not run)
```

---

umxAPA *Creates nicely formatted journal-style summaries of models, p-values, data-frames and much more.*

Description

umxAPA creates APA-style reports from a range of statistical models, or to summarize data. I wrote it to suit me.

Nice alternatives include jtools::summ.

Example functionality includes:

1. Given an stats::lm() model, umxAPA will return a formatted effect, including 95% CI. e.g.: umxAPA(lm(mpg~wt, data=mtcars), "wt") yields: β = -5.34 [-6.48, -4.20], p < 0.001. here "wt" restricts the output to just the named effect.
2. umxAPA also supports t.test(), stats::glm(), cor.test(), and others as I need them.
3. Get a CI from obj=beta and se=se : umxAPA(-0.30, .03) returns β = -0.3 [-0.36, -0.24]

4. Back out an SE from β and CI: `umxAPA(-0.030, c(-0.073, 0.013))` returns β = -0.03, se = 0.02
5. Given only a number as obj, will be treated as a p-value, and returned in APA format.
6. Given a dataframe, `umxAPA` will return a table of correlations with means and SDs in the last row. e.g.: `umxAPA(mtcars[,c("cyl", "wt", "mpg", )]` yields:

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

Usage

```
umxAPA(
  obj = .Last.value,
  se = NULL,
  p = NULL,
  std = FALSE,
  digits = 2,
  use = "complete",
  min = 0.001,
  addComparison = NA,
  report = c("markdown", "html", "none", "expression"),
  lower = TRUE,
  test = c("Chisq", "LRT", "Rao", "F", "Cp"),
  SEs = TRUE,
  means = TRUE,
  suffix = "",
  caption = NA,
  cols = NA,
  stars = TRUE
)

summaryAPA(
  obj = .Last.value,
  se = NULL,
  p = NULL,
  std = FALSE,
  digits = 2,
  use = "complete",
  min = 0.001,
  addComparison = NA,
  report = c("markdown", "html", "none", "expression"),
  lower = TRUE,
  test = c("Chisq", "LRT", "Rao", "F", "Cp"),
  SEs = TRUE,
  means = TRUE,
```

```
  suffix = "",
  caption = NA,
  cols = NA,
  stars = TRUE
)
```

Arguments

obj	A model (e.g. lm(), nlme::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.
p	If obj is a beta, use p-value to compute SE (returning a CI).
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. none doesn't print. expression can contain plotmath()
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)
suffix	A string to append to the result. Mostly used with report = "expression"
caption	Optional caption for html/markdown tables. NA (default) auto-generates "Effects on y of a, b, and c" from formula(obj) for lm/glm; NULL suppresses caption; a string uses that text.
cols	Optional, pass in a list of column names when using umxAPA with a dataframe input.
stars	Whether to append significance stars to p-values ("*p < .05,* "**p < .01,** "" p < .001). Default TRUE (APA-7 table standard: * p < .05, ** p < .01, *** p < .001; set stars=FALSE to suppress stars; enable for reviewers who request them).

Value

- string

See Also

SE_from_p()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
# ========================================
# = Report lm (regression/anova) results =
# ========================================
umxAPA(lm(mpg ~ wt + disp, mtcars)) # Report all parameters
umxAPA(lm(mpg ~ wt + disp, mtcars), "wt") # Just effect of weight
umxAPA(lm(mpg ~ wt + disp, mtcars), std = TRUE) # Standardize model!

###############
# GLM example #
###############

df = mtcars
df$mpg_thresh = 0
df$mpg_thresh[df$mpg > 16] = 1
m1 = glm(mpg_thresh ~ wt + gear,data = df, family = binomial)
umxAPA(m1)

###############
# A t-Test    #
###############

umxAPA(t.test(x = 1:10, y = c(7:20)))
umxAPA(t.test(extra ~ group, data = sleep))

# ======================================================
# = Summarize 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 from effect and se =
# ========================================
umxAPA(.4, .3) # parameter 2 interpreted as SE

# Input beta and CI, and back out the SE
umxAPA(-0.030, c(-0.073, 0.013), digits = 3)

# ====================
# = Format a p-value =
# ====================
umxAPA(.0182613)   #   0.02
umxAPA(.00018261) # < 0.001
umxAPA(.00018261, addComparison = FALSE) # 0.001

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

```
# = Report a correlation =
# ========================
data(twinData)
tmp = subset(twinData, zygosity %in% c("MZFF", "MZMM"))
m1 = cor.test(~ wt1 + wt2, data = tmp)
umxAPA(m1)
```

---

umxBrownie *A recipe Easter-egg for umx*

Description

How to cook steak.

Usage

```
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.

See Also

- OpenMx::omxBrownie()

Other Miscellaneous Functions: deg2rad(), rad2deg(), umx_check_parallel(), umx_complete_dollar()

Examples

```
umxBrownie()
```

---

umxCI *Add (and, optionally, run) confidence intervals to a structural model.*

Description

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

Usage

```
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"),
  regex = NULL,
  showErrorCodes = TRUE
)
```

Arguments

model	The OpenMx::mxModel() you wish to report OpenMx::mxCI()s 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")
regex	Add CIs for labels matching this regular expression (over-rides which)
showErrorCodes	Whether to show errors (default == TRUE)

Details

umxCI also reports if any problems were encountered. 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 run = "no", the function simply adds the CI requests, but returns the model without running them.

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

- stats::confint(), umxConfint(), umxCI(), umxModify()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
m1$intervals # none yet - empty list()
m1 = umxCI(m1)
m1$intervals # $G_to_x1...
m1 = umxCI(m1, remove = TRUE) # remove CIs from the model and return it
m1$intervals # none again

# Add CIs by name
parameters(m1, patt="_with_")
m1 = umxCI(m1, which = "x1_with_x1")
m1 = umxCI(m1, which = c("x1_with_x1", "x2_with_x2"))
m1 = umxCI(m1, regex = "x1_with_", run= "yes")
#           lbound estimate ubound lbound Code ubound Code
# x1_with_x1  0.036    0.041  0.047           0           0

# ========================
# = A twin model example =
# ========================
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACE(selDVs = c("bmi1","bmi2"), dzData = dzData, mzData = mzData)
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, which = "a_r1c1", run = "yes")

## End(Not run)
```

---

umxCI_boot *umxCI_boot*

Description

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

Usage

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

Arguments

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

Value

- expected covariance matrix

See Also

- umxExpMeans(), umxExpCov()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)

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

## End(Not run)
```

---

umxCloud *Run umx / OpenMx models on ephemeral Hetzner Cloud multicore nodes*

Description

umxCloud provisions an on-demand Linux server on Hetzner Cloud, transfers a model over SSH, runs optimization inside a Docker container, returns the fitted model, and (by default) destroys the server so billing stops when idle.

Usage

```
umxCloud(
  model,
  apiToken = Sys.getenv("HETZNER_API_TOKEN"),
  serverType = "cpx31",
  dockerImage = "rocker/r-ver:4.4.2",
  location = "fsn1",
  autoDestroy = TRUE,
  tryHard = c("no", "yes", "ordinal", "search"),
  maxBootMinutes = 15,
```

```
  keepOnError = FALSE,
  verbose = TRUE
)
```

Arguments

model	An MxModel (e.g. from umxACE(..., autoRun = FALSE)) with data attached.
apiToken	Hetzner API token. Default Sys.getenv("HETZNER_API_TOKEN").
serverType	Hetzner server type. Default "cpx31" (cheap smoke test). Use "ccx33", "ccx43", "ccx53", or "ccx63" for dedicated cores.
dockerImage	Docker image with R. Default public "rocker/r-ver:4.4.2". OpenMx and umx are installed inside the container on first use if missing.
location	Datacenter: "fsn1", "nbg1", "hel1", "ash", etc. Default "fsn1".
autoDestroy	Destroy server and Hetzner SSH key on exit? Default TRUE.
tryHard	Passed to remote umxRun. Default "no" for smoke tests.
maxBootMinutes	Max minutes to wait for SSH + Docker readiness. Default 15.
keepOnError	If TRUE, do not destroy the server when the run fails (for SSH debug). Default FALSE. Prints host IP when keeping the server.
verbose	Print progress messages? Default TRUE.

Details

Designed for small labs that need more cores than a laptop for heavy OpenMx/umx fits (long tryHard, large raw FIML, bootstrap grids), without keeping a permanent server.

First smoke test: use a cheap shared type such as serverType = "cpx31". First run often takes **8-20 minutes** (VM boot, Docker install, image pull, install OpenMx/umx from binaries). After plumbing works, switch to dedicated ccx* types for real multicore jobs.

The model must already contain its data (mxData / umx builders with data attached). There is no remote data= argument.

Value

Fitted MxModel from the remote run.

Step-by-Step Setup

1. Create a Hetzner Cloud account at https://console.hetzner.com/ (complete verification; avoid VPN during signup if fraud checks fail).
2. Project -> Security -> API Tokens -> Generate (Read & Write). Copy the token.
3. Store it in .Renviron (never commit the token):

   ```
   usethis::edit_r_environ()
   # HETZNER_API_TOKEN="..."
   ```

   Restart R. Confirm with nzchar(Sys.getenv("HETZNER_API_TOKEN")).
4. Install optional packages once: install.packages(c("httr2", "ssh")).

Cost notes (approximate, check Hetzner for current prices)

- **Smoke test:** cpx31 / similar shared – a few cents per short trial.
- ccx33 (8 dedicated cores): ~EUR0.22 / hour.
- ccx53 (32 dedicated cores): ~EUR0.86 / hour.
- ccx63 (64 dedicated cores): ~EUR1.60 / hour.

Default autoDestroy = TRUE deletes the server and temporary SSH key on exit (including errors), unless keepOnError = TRUE leaves the box for debugging.

See Also

umxRun, umxACE

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# Requires HETZNER_API_TOKEN in .Renviron and packages httr2, ssh
library(umx)
data(twinData)
mz = twinData[twinData$zygosity == "MZFF", ][1:200, ]
dz = twinData[twinData$zygosity == "DZFF", ][1:200, ]
m0 = umxACE(
  selDVs = c("bmi1", "bmi2"),
  mzData = mz, dzData = dz, sep = "",
  autoRun = FALSE
)
# Cheap first try (~cents). Watch Hetzner console: server should appear then vanish.
m1 = umxCloud(m0, serverType = "cpx31", tryHard = "no")
summary(m1)

# After plumbing works, use dedicated cores for heavy jobs:
# m1 = umxCloud(m0, serverType = "ccx53", tryHard = "yes")

## End(Not run)
```

---

umxCLPM *Runs cross-lagged panel models*

Description

One way of assessing causal relationships is by introducing time into the analyses. umxCLPM implements three cross-lagged panel models (CLPM) from the literature. The first is the classic CLPM from Heise (1969), the second is the CLPM from Hamaker et al. (2015), and the third is the CLPM from STARTS (1995). You simply pass the number of waves and the data set along with the model you wish to run.

Sketch mode is available; if you pass column names to data, a model object is returned for manipulation later.

Usage

```
umxCLPM(
  data = NULL,
  waves,
  name = NULL,
  model = c("Hamaker2015", "Heise1969", "STARTS1995", "IV_RI_CLPM"),
  counts = NULL,
  summary = !umx_set_silent(silent = TRUE),
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  verbose = FALSE,
  batteries = c("scale", "ordinaloptim"),
  std = FALSE,
  ivs = NULL,
  defn = NULL,
  defto = NULL,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals")
)
```

Arguments

data	Data frame for the analysis.
waves	Number of waves of data.
name	The name of the model (defaults to "Heise1969", "Hamaker2015", "STARTS1995" or "IV_RI_CLPM").
model	Model type ("Hamaker2015", "Heise1969", "STARTS1995", or "IV_RI_CLPM").
counts	Optional vector of count data columns.
summary	Logical indicating whether to show a summary (default: TRUE if silent is not set).
autoRun	Logical indicating whether to run the model (default to getOption("umx_auto_run")).
tryHard	Method for fitting the model ("no", "yes", "ordinal", "search").
verbose	Logical to control verbose output (default: FALSE).
batteries	A character vector of pre-processing options ("scale", "ordinaloptim", "thresholds").

std	Logical indicating whether to standardize the output (default: FALSE).
ivs	Optional vector of instrumental variable column names.
defn	Optional definition variable.
defto	Optional variable to which to define.
type	The method for handling missing data ("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS").
allContinuousMethod	Method for handling continuous data ("cumulants", "marginals").

Value

An OpenMx model object.

References

Kenny, D.A., & Zautra, A. (1995). The trait-state-error model for multiwave data. *Journal of Consulting and Clinical Psychology*, **63**, 52-59. doi:10.1037/0022006X.63.1.52 Hamaker E.L., Kuiper R.M., & Grasman R. (2015). A critique of the cross-lagged panel model. *Psychological Methods*, **20**, 102-116. doi:10.1037/a0038889 Heise D. R. (1970). Causal inference from panel data. *Sociological Methodology*, 2, 3-27. doi:10.2307/270780

Examples

```
## Not run:

# ================
# = 1. Load Data =
# ================
data(docData)
dt <- docData[2:9]

# ============================
# = 2. Make a CLPM model     =
# ============================
hamaker <- umxCLPM(waves = 4, name = "mymodel", model = "Hamaker2015", data = dt)

## End(Not run)
```

---

umxCompare *Print a comparison table of two or more* OpenMx::mxModel()*s, nicely formatted.*

Description

umxCompare compares two or more `OpenMx::mxModel()`s. It has several nice features:

1. Direct control of precision via digits=. p-values use APA style.
2. Report = directs publication to your chosen format: markdown, html, latex.
3. Columns arranged to make for easy visual comparison.
4. report = 'inline', includes a summary sentence suitable for your report.
5. report = "html" opens a web table in your browser to paste into a word processor.

Interpreting fit under WLS/DWLS and Genomic SEM

Hu & Bentler (1999) style cutoffs (e.g. CFI > .95, RMSEA < .06) were developed for maximum likelihood with continuous data. They do **not** transfer to WLS/DWLS (Shi et al., 2020), and still less to Genomic SEM (DWLS on LDSC genetic covariances with an estimated sampling covariance, often with bookkeeping `numObs = 1`).

- **Incremental indices (CFI, TLI, NFI, ...)** compare the model to an independence baseline under the same weight matrix. That baseline behaves differently than under ML. Report CFI/TLI only **descriptively**; do **not** apply conventional cutoffs.
- **Absolute fit: SRMR** (and residual matrices) is preferred. SRMR lives on a standardized residual metric and is less hostage to weight-matrix / N scaling. Rough guide: SRMR < 0.10 suggests acceptable residual structure (not a hard law).
- **RMSEA** folds in chi-square scaling and N; it is often misleading for WLS and especially for GSEM. Do not use the ML cutoff of < .06.
- **Nested model building:** prefer scaled chi-square difference tests and DeltaSRMR over DeltaCFI alone (see Details for what `umxCompare` computes).

Usage

```
umxCompare(
  base = NULL,
  comparison = NULL,
  all = TRUE,
  digits = 3,
  report = c("markdown", "html", "inline"),
  compareWeightedAIC = FALSE,
  silent = FALSE,
  file = "tmp.html",
  uncertainty = c("none", "SE", "MLR", "CI")
)
```

Arguments

`base`	The base `OpenMx::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.
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)
silent	(don't print, just return the table as a dataframe (default = FALSE)
file	file to write html too if report = "html" (defaults to "tmp.html")
uncertainty	What type of parameter uncertainty to report: "SE" (standard ML standard errors), "MLR" (robust standard errors and robust fit), "CI" (profile likelihood confidence intervals), or "none" (none).

Details

What umxCompare does for WLS

- Refuses mixed WLS vs ML comparisons (same engine required).
- **Continuous WLS** with cached `implied_jacobian`:
  - Table **Chi** is the same Satorra-Bentler (2010) scaled omnibus as `umxSummary()` (F/c from `output$fit`), **not** OpenMx Browne residual `output$chi` (those two differ under DWLS). Saturated residual df reports Chi = 0, CFI = 1, RMSEA = 0.
  - Nested **diffFit** is SB-2010 scaled $\Delta F = F_{nested} - F_{base}$ (requires both models at comparable minima of the same WLS objective: same moments, `useWeight`, and N). If $F_{nested} < F_{base}$ (often multigroup optimization), diffFit is **NA** with a warning - not a negative chi-square with p = 1.
- **Genomic SEM** (`MxModelGSEM`): nested difference on the **GSEM DWLS chi-square scale** (same discrepancy as the structural fit; not a substitute for ML LRT). If LDSC matrices were `nearPD`-smoothed, a fiduciary warning notes that difference tests may look artificially precise.
- Table also reports **SRMR / DeltaSRMR** (preferred absolute residual summary) and **CFI / DeltaCFI** (descriptive only). AIC may appear but is not a primary GSEM decision rule under asymptotic / bookkeeping N.

Ordinal WLS is a separate track: robust CFI/TLI/RMSEA use Savalei (2021) catML corrections in `umxSummary()`; Hu-Bentler cutoffs can apply to **those** robust indices. That exception does **not** apply to continuous WLS or Genomic SEM.

Best practice: report estimates with SEs, SRMR (and residuals), nested SB/GSEM difference tests, and avoid single-number "good fit" claims from CFI or RMSEA.

Constraints and the EP column

EP is OpenMx's count of free parameters, `summary(model)$estimatedParameters`. Equality `OpenMx::mxConstraint()`s are not removed from that count. OpenMx adds their rows to `observedStatistics`, and degrees of freedom already include them. A common-pathway model that sets each factor with `A + C + E = 1` can therefore show a larger EP than an independent-pathway model while Δ df is 0. Both numbers are the OpenMx counts. When `sum(summary(model)$constraints)` is above 0, `umxCompare` adds one line for that model:

```
Note: df for 'CP' reflects 3 constraints: 51 estimated parameters (48 effective after constraints).
```

With `silent = TRUE`, the lines are `attr(table, "constraintNotes")`, one element per model. A comparison in which no model has an equality constraint does not set that attribute. Inequality constraints contribute 0 and do not produce a line.

References

https://github.com/tbates/umx

See Also

- umxSummary(), umxRAM(),umxCompare()

Other Model Summary and Comparison: umx, umxEquate(), umxMI(), umxReduce(), umxSetParameters(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

m2 = umxModify(m1, update = "G_to_x2", name = "drop_path_2_x2")
umxCompare(m1, m2)
umxCompare(m1, m2, report = "inline") # Add English-sentence descriptions
umxCompare(m1, m2, report = "html") # Open table in browser

# Two comparison models
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)

manifests = names(demoOneFactor)
m1 = umxRAM("WLS", data = demoOneFactor, type = "DWLS",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

m2 = umxModify(m1, update = "G_to_x2", name = "drop_path_2_x2")
umxCompare(m1, m2)
umxCompare(m1, m2, report = "inline") # Add English-sentence descriptions
umxCompare(m1, m2, report = "html") # Open table in browser

## End(Not run)
```

umxConfint *Get confidence intervals from a umx model*

Description

Implements confidence interval function for umx models.

Usage

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

Arguments

object	An OpenMx::mxModel(), possibly already containing OpenMx::mxCI()s that have been OpenMx::mxRun() with intervals = TRUE))
parm	Which parameters to get confidence intervals for. Can be "existing", "all", or one or more parameter 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	For difficult CIs, trying other optimizers can help!

Details

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

Because CIs can take time to run, by default only already-computed CIs will be reported. To run new CIs, set run = TRUE .

Note: OpenMx defines a `confint` function which will return SE-based CIs.

If `parm` is empty, and `run = FALSE`, a message will alert you to set `run = TRUE`.

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

- stats::confint(), OpenMx::mxSE(), umxCI(), OpenMx::mxCI()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)

manifests = names(demoOneFactor)
m1 = umxRAM("OneFactor", data = demoOneFactor, type = "cov",
 umxPath(from = "G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", 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)

# For some twin models, a "smart" mode is implemented
# note: only implemented for umxCP 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://github.com/tbates/umx

See Also

cov2cor()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

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

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

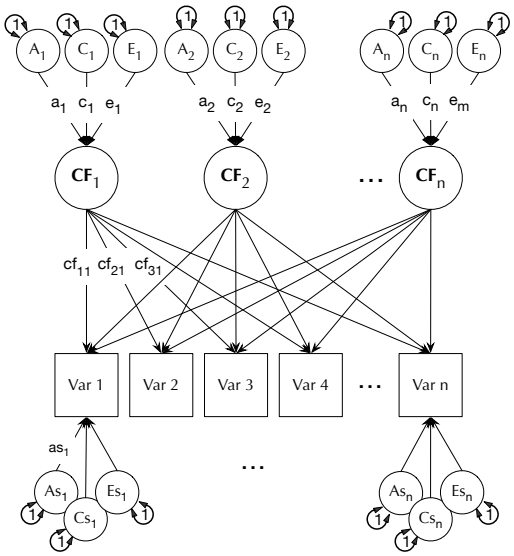
Description

Make a 2-group Common Pathway twin model.

The common-pathway model (aka "psychometric model" (McArdle and Goldsmith, 1990) provides a powerful tool for theory-based testing of genetic and environmental differences. It proposes that A, C, and E components act on a latent substrate (organ, mental mechanism etc.) and this is manifested in the measured phenotypes.

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() allows users to see and set mvnRelEps and mvnMaxPointsA mvnRelEps defaults to .005. For ordinal models, you might find that '0.01' works better.

Usage

```
umxCP(
  name = "CP",
  selDVs,
  selCovs = NULL,
  dzData = NULL,
```

```
  mzData = NULL,
  sep = NULL,
  nFac = 1,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  data = NULL,
  zyg = "zygosity",
  allContinuousMethod = c("cumulants", "marginals"),
  correlatedACE = FALSE,
  dzAr = 0.5,
  dzCr = 1,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("yes", "no", "ordinal", "search"),
  optimizer = NULL,
  equateMeans = TRUE,
  weightVar = NULL,
  bVector = FALSE,
  boundDiag = 0,
  addStd = TRUE,
  addCI = TRUE,
  numObsDZ = NULL,
  numObsMZ = NULL,
  freeLowerA = FALSE,
  freeLowerC = FALSE,
  freeLowerE = FALSE,
  correlatedA = "deprecated"
)
```

Arguments

name	The name of the model (defaults to "CP").
selDVs	The variables to include. omit sep in selDVs, i.e., just "dep" not c("dep_T1", "dep_T2").
selCovs	basenames for covariates
dzData	The DZ dataframe.
mzData	The MZ dataframe.
sep	(required) The suffix for twin 1 and twin 2, often "_T".
nFac	How many common factors (default = 1)
type	One of "Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"
data	If provided, dzData and mzData are treated as valid levels of zyg to select() data sets (default = NULL)
zyg	If data provided, this column is used to select rows by zygosity (Default = "zygosity")
allContinuousMethod	"cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

correlatedACE	DON'T USE THIS! Allows correlations between the factors built by each of the a, c, and e matrices. Default = FALSE.
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 run the model (default), or just to create it and return without running.
tryHard	Default ("yes") uses mxTryHard, "no" uses normal mxRun. Other options: "ordinal", "search"
optimizer	optionally set the optimizer (default NULL does nothing).
equateMeans	Whether to equate the means across twins (defaults to 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).
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 (defaults to TRUE).
addCI	Whether to add the interval requests for CIs (defaults to TRUE).
numObsDZ	= not yet implemented: Ordinal Number of DZ twins: Set this if you input covariance data.
numObsMZ	= not yet implemented: Ordinal Number of MZ twins: Set this if you input covariance data.
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).
correlatedA	deprecated.

Details

Like the umxACE() model, the CP model decomposes phenotypic variance into additive genetic (A), 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% by twins in a pair), 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:

```
m1$top$as$values
```

```
m1$top$as$free
```

```
m1$top$as$labels
```

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:

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

For a deep-dive, see `xmu_make_TwinSuperModel()`

Value

- `OpenMx::mxModel()`

References

- Martin, N. G., & Eaves, L. J. (1977). The Genetical Analysis of Covariance Structure. *Heredity*, **38**, 79-95.
- Kendler, K. S., Heath, A. C., Martin, N. G., & Eaves, L. J. (1987). Symptoms of anxiety and symptoms of depression. Same genes, different environments? *Archives of General Psychiatry*, **44**, 451-457. doi:10.1001/archpsyc.1987.01800170073010.
- McArdle, J. J., & Goldsmith, H. H. (1990). Alternative common factor models for multivariate biometric analyses. *Behavior Genetics*, **20**, 569-608. doi:10.1007/BF01065873.
- https://github.com/tbates/umx

See Also

- `umxSummaryCP()`, `umxPlotCP()`. See `umxRotate.MxModelCP()` to rotate the factor loadings of a `umxCP()` model. See `umxACE()` for more examples of twin modeling. `plot()` and `umxSummary()` work for all twin models, e.g., `umxIP()`, `umxCP()`, `umxGxE()`, and `umxACE()`.

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`,

umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# ========================================================
# = Run a 3-factor Common pathway twin model of 6 traits =
# ========================================================
require(umx)
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
#  # These will be expanded into "gff_T1" "gff_T2" etc.
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP(selDVs = selDVs, sep = "_T", nFac = 3, tryHard = "yes",
  dzData = dzData, mzData = mzData)

# Shortcut using "data ="
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP(selDVs= selDVs, nFac= 3, data=GFF, zyg="zyg_2grp")

# ====================
# = Do it using WLS =
# ====================
m2 = umxCP("new", selDVs = selDVs, sep = "_T", nFac = 3, optimizer = "SLSQP",
  dzData = dzData, mzData = mzData, tryHard = "ordinal",
 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])

# These will be expanded into "gff_T1" "gff_T2" etc.
selDVs = 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(selDVs = selDVs, 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(selDVs = selDVs, sep = "_T", nFac = 3, dzData = dzData, mzData = mzData,
  tryHard = "ordinal", type= "DWLS")
# TODO umxCPL fix WLS here
# label at row 1 and column 1 of matrix 'top.binLabels'' in model 'CP3fac' : object 'Vtot'

# ==============================
# = Correlated factors example =
# ==============================
# ====================
# = DON'T USE THIS!!! =
# ====================
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP("base_model", selDVs = selDVs, sep = "_T", correlatedACE = TRUE,
  dzData = dzData, mzData = mzData, nFac = 3, tryHard = "yes")

# What are the ace covariance labels? (two ways to get)
umx_lower.tri(m1$top$a_cp$labels)
parameters(m1, patt = "[ace]_cp")

# 1. Now allow a1 and a2 to correlate
m2=umxModify(m1,regex="a_cp_r2c1",name="a2_a1_cov",free=TRUE,tryHard="yes")
umxCompare(m2, m1)

# 2. Drop all (a|c|e) correlations from a model
tmp= namez(umx_lower.tri(m2$top$a_cp$labels), "a_cp", replace= "[ace]_cp")
m3 = umxModify(m2, regex= tmp, comparison = TRUE)

## End(Not run)
```

---

umxDiagnose *Diagnose problems in a model - not working!*

Description

The goal of this function **WILL BE** (not currently functional) 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

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

Arguments

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

Details

Best diagnostics are:

1. Observed data variances and means
2. Expected variances and means
3. Difference of these?

Try * diagonalizeExpCov diagonal * umx_is_ordered()

Tricky, but reporting variances and standardized thresholds is ideal. Guidance is to start with unit variances and thresholds within +/- 2 SD of the mean. Like %p option in Classic Mx.

Value

- helpful messages and perhaps a modified model

References

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

See Also

Other Teaching and Testing functions: tmx_show.MxModel(), umxPower()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("OneFactor", data = demoOneFactor, type= "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
```

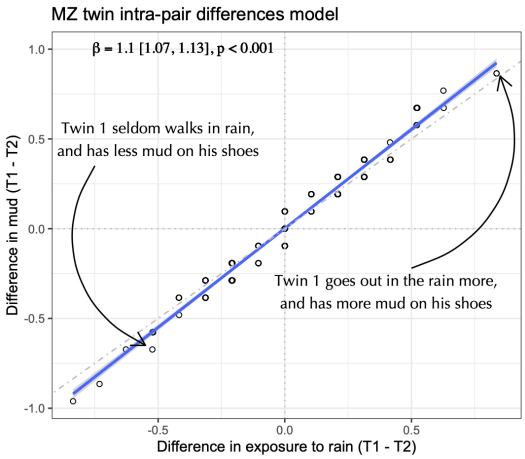
```
 umxPath(var = "G", fixedAt = 1)
)
m1 = mxRun(m1)
umxSummary(m1, std = TRUE)
umxDiagnose(m1)

## End(Not run)
```

---

umxDiffMZ *MZ differences method for testing evidence for causality.*

Description

umxDiffMZ implements the simple twin1-twin2 based correlation method, e.g. De Moor (2008), in which MZ differences on a variable x asserted to be causal of an outcome variable y are tested for association with differences on y. The logic of the design is shown below:



Usage

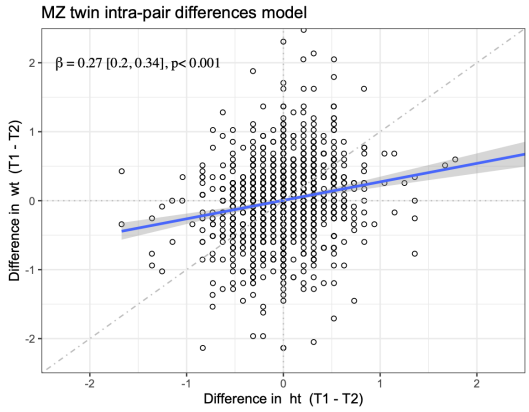
```
umxDiffMZ(
  x,
  y,
  data,
  sep = "_T",
  mzZygs = c("MZFF", "MZMM"),
  zyg = "zygosity",
  labxy = c(-1.2, 1.8),
  xylim = c(NA, NA),
  digits = 2
)
```

Arguments

x	Presumed causal variable, e.g. "effort"
y	Presumed caused outcome, e.g. "score"
data	Dataframe containing the twin data.
sep	The separator "_T" used to make twin var names from x and y.
mzZygs	The MZ zygosity codes c("MZFF", "MZMM")
zyg	The column containing "zygosity" data
labxy	Where to locate the R2 label (default = c(x=-2,y=3))
xylim	= clip x any axes to range, e.g c(-3,-3)
digits	Rounding for beta (def2)

Details

Example output is shown below, with the fitted line and fit inscribed. The plot is just a ggplot2 graph that is returned and can be edited and formatted.



For a more sophisticated linear mixed model approach, see umxDiscTwin().

Value

- Graph for decorating

References

De Moor, M. H., Boomsma, D. I., Stubbe, J. H., Willemsen, G., & de Geus, E. J. (2008). Testing causality in the association between regular exercise and symptoms of anxiety and depression. Archives of General Psychiatry, 65(8), 897-905. doi:10.1001/archpsyc.65.8.897.

See Also

- umxDoC(), umxDiscTwin(), umxMR()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(),

umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
data(twinData)
umxDiffMZ(x="ht", y="wt", labxy = c(-.5, 3), data = twinData, sep = "")
umxDiffMZ(x="ht", y="wt", xylim = c( -2, 2), data = twinData, sep = "")
```

---

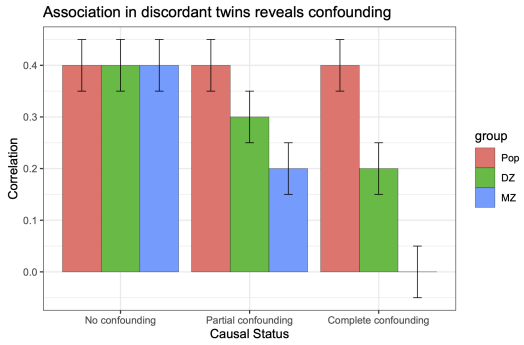
umxDiscTwin *Intra-pair association in MZ, DZ twin models. (ALPHA quality!)*

Description

Testing causal claims is often difficult due to an inability to experimentally randomize traits and situations. A combination of control data and data from twins discordant for the putative causal trait can falsify causal hypotheses.

umxDiscTwin uses nlme::nlme() to compute the beta for x in y ~ x in models either a) Only controlling non-independence, and b) MZ and DZ subsample models in which the family level of the predictor y is also controlled.

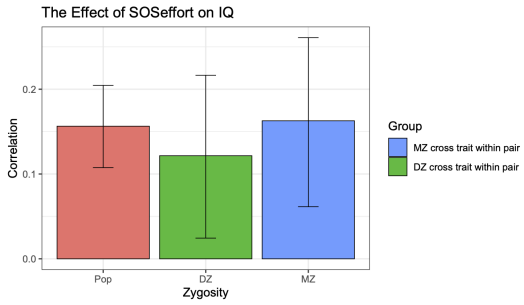
If x is causal, then the effect size of x on y is expected to be equally large in all three samples. If the population association reflects confounded genes or shared environments, then the association in MZ twins will reduce to zero/non-significance.



The function uses the nlme::lme() function to compute the effect of the presumed causal variable on the outcome, controlling, for mid-family score and with random means model using familyID. e.g.:

```
mzModel = lme(fixed = y ~ x + FamMeanX, random = ~ 1+FamMeanX|FAMID, data = umx_scale(MZ),
na.action = "na.omit")
```

Example output from umxDiscTwin



Usage

```
umxDiscTwin(
  x,
  y,
  data,
  mzZygs = c("MZFF", "MZMM"),
  dzZygs = c("DZFF", "DZMM", "DZOS"),
  FAMID = "FAMID",
  out = c("table", "plot", "model"),
  use = "complete.obs",
  sep = "_T"
)
```

Arguments

x	Cause
y	Effect
data	dataframe containing MZ and DZ data
mzZygs	MZ zygosities c("MZFF", "MZMM")
dzZygs	DZ zygosities c("DZFF", "DZMM", "DZOS")
FAMID	The column containing family IDs (default = "FAMID")
out	Whether to return the table or the ggplot (if you want to decorate it)
use	NA handling in corr.test (default= "complete.obs")
sep	The separator in twin variable names, default = "_T", e.g. "dep_T1".

Value

- table of results

References

- Begg, M. D., & Parides, M. K. (2003). Separation of individual-level and cluster-level covariate effects in regression analysis of correlated data. Stat Med, 22(16), 2591-2602. doi:10.1002/sim.1524

- Bergen, S. E., Gardner, C. O., Aggen, S. H., & Kendler, K. S. (2008). Socioeconomic status and social support following illicit drug use: causal pathways or common liability? *Twin Res Hum Genet*, **11**, 266-274. doi:10.1375/twin.11.3.266
- McGue, M., Osler, M., & Christensen, K. (2010). Causal Inference and Observational Research: The Utility of Twins. *Perspectives on Psychological Science*, **5**, 546-556. doi:10.1177/1745691610383511

See Also

- umxDoC(), umxDiffMZ(), umxMR()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
data(twinData)
# add to test must set FAMID umxDiscTwin(x = "ht", y = "wt", data = twinData, sep="")
tmp = umxDiscTwin(x = "ht", y = "wt", data = twinData, sep="", FAMID = "fam")
print(tmp, digits = 3)

## End(Not run)
```

---

umxDoC *Build and run a 2-group Direction of Causation twin models.*

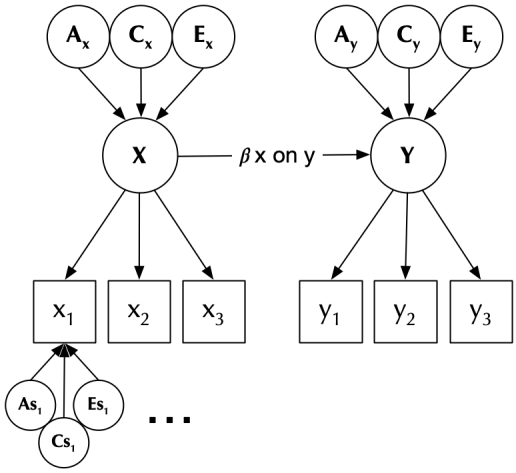
Description

Testing causal claims is often difficult due to an inability to conduct experimental randomization of traits and situations to people. When twins are available, even when measured on a single occasion, the pattern of cross-twin cross-trait correlations can (given distinguishable modes of inheritance for the two traits) falsify causal hypotheses.

umxDoC implements a 2-group model to form latent variables for each of two traits, and allows testing whether trait 1 causes trait 2, vice-versa, or even reciprocal causation.

Using latent variables instead of a manifest measure for testing causation, avoids the bias created by differences in measurement error in which the more reliable measure appears to "cause" the less reliable one (Gillespie and Martin, 2005).

The following figure shows how the DoC model appears as a path diagram (for two latent variables X and Y, each with three indicators). Note: For pedagogical reasons, only the model for 1 twin is shown, and only one DoC pathway drawn.



Usage

```
umxDoC(
  name = "DoC",
  var1Indicators,
  var2Indicators,
  mzData = NULL,
  dzData = NULL,
  sep = "_T",
  causal = TRUE,
  autoRun = getOption("umx_auto_run"),
  intervals = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  data = NULL,
  zyg = "zygosity"
)
```

Arguments

name	The name of the model (defaults to "DOC").
var1Indicators	variables defining latent trait 1
var2Indicators	variables defining latent trait 2
mzData	The MZ dataframe
dzData	The DZ dataframe
sep	The separator in twin variable names, default = "_T", e.g. "dep_T1".
causal	whether to add the causal paths (default TRUE)
autoRun	Whether to run the model (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: "ordinal", "search"
optimizer	Optionally set the optimizer (default NULL does nothing).
data	= NULL If building the MZ and DZ datasets internally from a complete data set.
zyg	= "zygosity" (for the data= method of using this function)

Value

- OpenMx::mxModel() of subclass MxModelDoC

References

- Gillespie, N.A. and Martin, N.G. (2005). Direction of Causation Models. In *Encyclopedia of Statistics in Behavioral Science*, **1**. 496-499. Eds. Brian S. Everitt & David C. Howell.
- McGue, M., Osler, M., & Christensen, K. (2010). Causal Inference and Observational Research: The Utility of Twins. *Perspectives on Psychological Science*, **5**, 546-556. doi:10.1177/1745691610383511
- Rasmussen, S. H. R., Ludeke, S., & Hjelmborg, J. V. B. (2019). A major limitation of the direction of causation model: non-shared environmental confounding. *Twin Res Hum Genet*, **22**, 1-13. doi:10.1017/thg.2018.67

See Also

- umxDiscTwin(), umxDiffMZ(), umxMR()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# ========================
# = Does Rain cause Mud? =
# ========================

# ================
# = 1. Load Data =
# ================
data(docData)
docData = umx_scale_wide_twin_data(c(var1, var2), docData, sep= "_T")
mzData  = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData  = subset(docData, zygosity %in% c("DZFF", "DZMM"))

# ======================================
# = 2. Define manifests for var 1 and 2 =
# ======================================
```

```
var1 = paste0("varA", 1:3)
var2 = paste0("varB", 1:3)

# ========================================================
# = 3. Make the non-causal (Cholesky) and causal models =
# ========================================================
Chol = umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= FALSE)
# note: DoC initially has causal paths fixed @0
DoC  = umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= TRUE)
a2b   = umxModify(DoC, "a2b", free = TRUE, name = "a2b"); summary(a2b)
b2a   = umxModify(DoC, "b2a", free = TRUE, name = "b2a"); summary(b2a)
Recip = umxModify(DoC, c("a2b", "b2a"), free = TRUE, name = "Recip"); summary(Recip)

# Compare fits
umxCompare(Chol, c(a2b, b2a, Recip))

# ========================================
# = Alternative call with data in one file =
# ========================================
data(docData)
docData = umx_scale_wide_twin_data(c(var1, var2), docData, sep= "_T")
DoC = umxDoC(var1= paste0("varA", 1:3), var2= paste0("varB", 1:3),
   mzData= c("MZFF", "MZMM"), dzData= c("DZFF", "DZMM"), data = docData
)

## End(Not run)
```

---

umxDoCp *Make a direction of causation model based on umxPath statements*

Description

Makes a direction of causation model with umxPath() statements

Usage

```
umxDoCp(
  var1Indicators,
  var2Indicators,
  mzData = NULL,
  dzData = NULL,
  sep = "_T",
  causal = TRUE,
  name = "DoC",
  autoRun = getOption("umx_auto_run"),
  intervals = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL
)
```

Arguments

var1Indicators	The indicators of trait 1
var2Indicators	The indicators of trait 2
mzData	The MZ twin dataframe
dzData	The DZ twin dataframe
sep	(Default "_T")
causal	(Default TRUE)
name	= "DoC"
autoRun	Default: getOption("umx_auto_run")_
intervals	Whether to run intervals (Default FALSE)
tryHard	Default "no" (valid = "yes", "ordinal", "search")
optimizer	Whether to set this for this run (Default no))

Details

See also umxDoC()

Value

- [A direction of causation model with umxPath() statements.

See Also

- umxDoC()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# ================
# = 1. Load Data =
# ================
data(docData)
var1 = paste0("varA", 1:3)
var2 = paste0("varB", 1:3)
tmp = umx_scale_wide_twin_data(varsToScale= c(var1, var2), sep= "_T", data= docData)
mzData = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData = subset(docData, zygosity %in% c("DZFF", "DZMM"))
m1 = umxDoCp(var1, var2, mzData= mzData, dzData= dzData, sep = "_T", causal= TRUE)


## End(Not run)
```

umxEFA *FIML-based Exploratory Factor Analysis (EFA)*

Description

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

Usage

```
umxEFA(
  x = NULL,
  factors = NULL,
  data = NULL,
  scores = c("none", "ML", "WeightedML", "Regression"),
  minManifests = NA,
  rotation = c("varimax", "promax", "none"),
  return = c("model", "loadings"),
  report = c("markdown", "html"),
  summary = FALSE,
  name = "efa",
  digits = 2,
  tryHard = c("no", "yes", "ordinal", "search"),
  n.obs = NULL,
  covmat = NULL
)

umxFactanal(
  x = NULL,
  factors = NULL,
  data = NULL,
  scores = c("none", "ML", "WeightedML", "Regression"),
  minManifests = NA,
  rotation = c("varimax", "promax", "none"),
  return = c("model", "loadings"),
  report = c("markdown", "html"),
  summary = FALSE,
  name = "efa",
  digits = 2,
  tryHard = c("no", "yes", "ordinal", "search"),
  n.obs = NULL,
  covmat = NULL
)
```

Arguments

x	Either data or Right-hand-side formula or a vector of variable names, or the name for the model.

factors	Either number of factors to extract or a vector of factor names.
data	A dataframe you are modeling.
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))
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")
summary	run umxSummary() on the underlying umxRAM model? (Default = FALSE)
name	A name for your model (default = efa)
digits	rounding (default = 2)
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
n.obs	Number of observations in if covmat provided (default = NA)
covmat	Covariance matrix of data you are modeling (not implemented)

Details

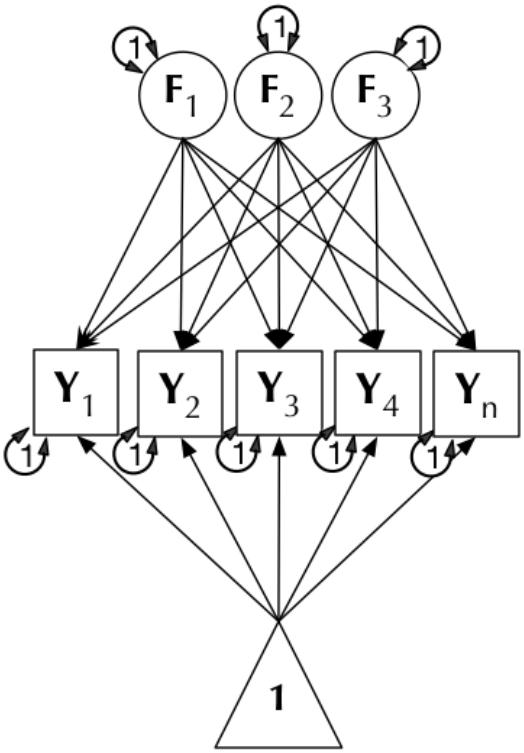
As in factanal(), you need only specify the number of factors and offer up some manifest data, e.g:

```
umxEFA(factors = 2, data = mtcars
```

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

```
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.

Should work with rotations provided in `libs("GPArotation")` and `libs("psych")`, e.g.,

Orthogonal: "varimax", "quartimax", "bentlerT", "equamax", "varimin", "geominT" and "bifactor"
Oblique: "Promax", "promax", "oblimin", "simplimax", "bentlerQ", "geominQ", "biquartimin" and "cluster"

For identification we need m^2 degrees of freedom. We get $m(m+1)/2$ from fixing factor variances to 1 and covariances to `0`. We get another $m(m-1)/2$ degrees of freedom by fixing the upper-right hand corner of the factor loadings component of the `A` matrix @ 0.

To aid optimization, manifest residual variances are `1bounded` at 0.

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

note: Bear in mind that factor scores are indeterminate (can be rotated to an infinity of equivalent solutions).

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

Value

- EFA `OpenMx::mxModel()`

References

- Hendrickson, A. E. and White, P. O. (1964). Promax: a quick method for rotation to orthogonal oblique structure. *British Journal of Statistical Psychology*, **17**, 65-70. doi:10.1111/j.20448317.1964.tb00244.x.
- Kaiser, H. F. (1958). The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, **23**, 187-200. doi:10.1007/BF02289233.

See Also

- factanal(), OpenMx::mxFactorScores()

Other Super-easy helpers: umx, umxTwoStage()

Examples

```
## Not run:
myVars = c("mpg", "disp", "hp", "wt", "qsec")
m1 = umxEFA(mtcars[, myVars], factors =   2, rotation = "promax")
# By default, returns the model
umx_is_MxModel(m1) # TRUE
# The loadings are stashed in the model:
loadings(m1)

# Formula interface in umxEFA
m2 = umxEFA(~ mpg + disp + hp + wt + qsec, factors = 2, rotation = "promax", data = mtcars)
loadings(m2)

# base-R factanal Formula interface for comparison
m2 = factanal(~ mpg + disp + hp + wt + qsec, factors = 2, rotation = "promax", data = mtcars)
loadings(m2)

# Return the loadings object
x = umxEFA(mtcars[, myVars], factors = 2, return = "loadings")
names(x) # "loadings" "rotmat"

# scores requested, so these will be returned
x = umxEFA(name = "score", factors = "g", data = mtcars[, myVars], scores= "Regression")
head(x)
#       g
# 1  -0.481
# 2  -0.423
# 3  -0.871

m1 = umxEFA(myVars, factors = 2, data = mtcars, rotation = "promax")
m1 = umxEFA(name = "named", factors = "g", data = mtcars[, myVars])
m1 = umxEFA(name = "by_number", factors = 2, rotation = "promax", data = mtcars[, myVars])

## 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

```
umxEquate(
  model,
  a,
  b,
  newlabels = NULL,
  free = c(TRUE, FALSE, NA),
  verbose = FALSE,
  name = NULL,
  autoRun = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  comparison = TRUE,
  master = NULL,
  slave = NULL
)
```

Arguments

model	An OpenMx::mxModel() within which to equate parameters listed in "a" with those in "b"
a	one or more labels to equate with those in the "b" set.
b	one or more labels to equate with those in the 'a' set. (if 'newlabels' is NULL, labels will be set to 'a' list).
newlabels	(optional) list of new labels for the equated parameters.
free	Must the 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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"

comparison	Compare the new model to the old (if updating an existing model: default = TRUE)
master	synonym for 'a'
slave	synonym for 'b'

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

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

umxModify(), umxCompare()

Other Model Summary and Comparison: umx, umxCompare(), umxMI(), umxReduce(), umxSetParameters(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
# By default, umxEquate just equates master and slave labels: doesn't run model
m2 = umxEquate(m1, a = "G_to_x1", b = "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",
      a = "G_to_x1", b = "G_to_x2"
)

# rename the equated paths
m2 = umxEquate(m1, autoRun = TRUE, comparison = TRUE, name = "Eq_x1_x2",
      a = "G_to_x1", b = "G_to_x2", newlabels = c("equated")
)
parameters(m2)
```

```
## End(Not run)
```

---

umxExamples *Example code from Twin Research and Human Genetics Paper on umx*

Description

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

Usage

```
umxExamples()
```

References

Bates, T. C., Neale, M. C., & Maes, H. H. (2019). umx: A library for Structural Equation and Twin Modelling in R. *Twin Research and Human Genetics*, **22**, 27-41. doi:10.1017/thg.2019.2.

See Also

- umx()

Examples

```
## Not run:

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

# Installing umx can be done using the R-code:
install.packages("umx")
# load as usual
library("umx")

# The current package version can be shown with:
umxVersion("umx")

# Get the latest NPSOL and multi-core build of OpenMx
install.OpenMx("NPSOL")

# Bleeding edge version of OpenMx for MacOS
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 = paste0("x", 1:5) # 'x1', 'x2', ...'x5'

# Create model cfa1, with name 'CFA', data demoOneFactor, and the CFA paths.

cfa1 = umxRAM("CFA", data = demoOneFactor,
 # Create latent variable 'G', with fixed variance of 1 and mean of 0
 umxPath(v1m0 = "G"),
 # Create 5 manifest variables, x1:x5, with free variance and mean
 umxPath(v.m. = manifests),
 # Create 1-headed paths from G to each of the manifests
 umxPath("G", to = manifests)
)

# ====================
# = Parameter labels =
# ====================

x = xmuLabel(mxMatrix(name="means", "Full", ncol = 2, nrow = 2))
x$labels

# ========
# = Plot =
# ========

plot(cfa1, means = FALSE, fixed = TRUE)
plot(cfa1, std = TRUE, digits = 3, resid= 'line')


m1 = umxRAM("play", data = c("A", "B", "C"),
 umxPath(unique.pairs = c("A", "B", "C"))
)

# ============================================
# = Inspecting model parameters and residuals. =
# ============================================

# Show parameters, below .1, with label containing `x2'
parameters(cfa1, "above", .5, pattern= "x2")

residuals(cfa1, 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, 0.3607, -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))
)

# ====================
# = Modifying models =
# ====================

# Collect a list of paths to drop
pathList = c("RespLatentAsp_to_FrndLatentAsp", "FrndLatentAsp_to_RespLatentAsp")

# 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(cfa1, regex = "^G_to.*")

# ====================
# = 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 =
# ============================

parameters(duncan1, pattern = "IQ_to_")


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

# ================
# = ACE examples =
# ================

require(umx);
```

```
# open the built in dataset of Australian height and weight twin data
data("twinData")
selDVs = c("wt")
dz = twinData[twinData$zygosity == "DZFF", ]
mz = twinData[twinData$zygosity == "MZFF", ]

ACE1 = umxACE(selDVs = 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 =
# ================================

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

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

# Run and report a common-pathway model
CP1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = "")

paths = c("c_cp_r1c1", "cs_r1c1", "cs_r2c2")
CP2 = umxModify(CP1, update = paths, name = "dropC", comparison = TRUE)

CP2 = umxModify(CP1, regex = "(^cs_)|(^c_cp_)", name = "dropC")
umxSummary(CP2, comparison = CP1)

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

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

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)
umx_set_auto_plot(TRUE)


## End(Not run)
```

---

umxExpCov *Get the expected vcov matrix*

Description

Extract the expected covariance matrix from an `OpenMx::mxModel()`

Usage

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

Arguments

object	an `OpenMx::mxModel()` to get the covariance matrix from
latents	Whether to select the latent variables (defaults to TRUE)
manifests	Whether to select the manifest variables (defaults to TRUE)

digits	precision of reporting. NULL (Default) = no rounding.
...	extra parameters (to match vcov())

Value

- expected covariance matrix

See Also

- umxRun(), umxCI_boot()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpMeans(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)#'
vcov(m1) # supplied by OpenMx
umxExpCov(m1, digits = 3)


## End(Not run)
```

---

umxExpMeans *Extract the expected means matrix from an OpenMx::mxModel()*

Description

Extract the expected means matrix from an OpenMx::mxModel()

Usage

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

Arguments

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

Value

- expected means

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxFitIndices(), umxRotate(), umxSummaryACE_DE()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor,
 umxPath("G", to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = "G")
)

umxExpMeans(m1)
umxExpMeans(m1, digits = 3)

## End(Not run)
```

---

umxFactor *umxFactor*

Description

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

Usage

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

umx_factor(
```

```
  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 OpenMx::mxFactor())
levels	(default NULL). Like factor() but UNLIKE OpenMx::mxFactor(), unique values will be used if levels not specified.
labels	= levels (see OpenMx::mxFactor())
exclude	= NA (see OpenMx::mxFactor())
ordered	= TRUE By default return an ordered mxFactor
collapse	= FALSE (see OpenMx::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

- OpenMx::mxFactor()

See Also

- umxFactanal(), OpenMx::mxFactor()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
umxFactor(letters)
umxFactor(letters, verbose = TRUE) # report coercions
umxFactor(letters, ordered = FALSE) # non-ordered factor like factor(x)
# Dataframe example:
x = umx_factor(mtcars[,c("cyl", "am")], ordered = FALSE); str(x)
# =================
# = Twin example: =
# =================
data(twinData)
```

```
tmp = twinData[, c("bmi1", "bmi2")]
tmp$bmi1[tmp$bmi1 <= 22] = 22
tmp$bmi2[tmp$bmi2 <= 22] = 22
# remember to factor _before_ breaking into MZ and DZ groups
x = umxFactor(tmp, sep = ""); str(x)
xmu_check_levels_identical(x, "bmi", sep="")

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

---

umxFactorScores *Return factor scores from a model as an easily consumable dataframe.*

Description

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,
  return = c("Scores", "StandardErrors")
)
```

Arguments

model	The model from which to generate scores.
type	Method of computing the score ('ML', 'WeightedML', or 'Regression').
minManifests	The minimum number of variables not NA to return a score for a participant (Default = ask).
return	What to return (defaults to "Scores", which is what most users want, but can return "StandardErrors" on each score.

Value

- dataframe of scores.

See Also

- OpenMx::mxFactorScores()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

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

# ============================================================================
# = Histogram of F1 and plot of F1 against F2 showing they are orthogonal =
# ============================================================================
hist(x$F1)
plot(F1 ~ F2, data = x)

m1 = umxEFA(mtcars, factors = 1)
x = umxFactorScores(m1, type = 'Regression', minManifests = 3)
x

## End(Not run)
```

---

umxFitIndices *Get additional fit-indices for a model with umxFitIndices*

Description

Computes a variety of fit indices.

Usage

```
umxFitIndices(model, ...)
```

Arguments

model	The OpenMx::mxModel() for which you want fit indices.
...	Additional parameters passed to OpenMx::summary.MxModel().

Details

Note: This function is currently not robust across multi-group designs or definition variables. It is designed to provide residual-based fit indices (SRMR, CRMR, SMAR, CMAR, etc.) and less-often reported fit indices where Reviewer 2 wants something other than CFA/TLI/RMSEA.

Fit information reported includes:

Model characteristics: numObs, estimated parameters, observed statistics, observed summary statistics, -2*log(Likelihood), degrees of freedom

Chi-squared test: Chi, ChiDoF, p (of Chi), ChiPerDoF,

Noncentrality-based indices: RMSEA, RMSEACI, RMSEANull, RMSEAClose (p value), independenceRMSEA, NCP, NCPCI, F0, F0CI, Mc (aka NCI, MFI)

Comparative fit indices: TLI (aka NNFI), CFI, IFI, PRATIO, PCFI

Residual-based indices: RMR, SRMR, SRMR_mplus, CRMR, MAR, SMAR, SMAR_mplus, CMAR

Information-theory criteria (computed using chi-square or -2LL; df or parameters penalties) AIC, AICc, BIC, SABIC, CAIC, BCC ECVI, ECVICI, MECVI, MECVICI

LISREL and other early fit indices (we recommend not reporting these) GFI, AGFI, PGFI, GH, NFI, PNFI, RFI

Want more? *Open an Issue* at GitHub.

Value

List of fit statistics

Author(s)

Brenton M. Wiernik, Athanassios Protopapas, Paolo Ghisletta, Markus Brauer

See Also

Other Reporting functions: `RMSEA()`, `RMSEA.MxModel()`, `RMSEA.summary.mxmodel()`, `extractAIC.MxModel()`, `loadings()`, `loadings.MxModel()`, `residuals.MxModel()`, `tmx_show()`, `tmx_show.MxMatrix()`, `umxCI()`, `umxCI_boot()`, `umxConfint()`, `umxExpCov()`, `umxExpMeans()`, `umxRotate()`, `umxSummaryACE_DE()`

Examples

```
## Not run:
library(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor",
 data = mxData(cov(demoOneFactor), type = "cov", numObs = 500),
 umxPath(latents, to = manifests),
 umxPath(var = manifests),
 umxPath(var = latents, fixedAt = 1)
)
umxFitIndices(m1)

# And with raw data
m2 = umxRAM("m1", data = demoOneFactor,
 umxPath(latents, to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = latents)
)
```

```
umxFitIndices(m1, refModels = mxRefModels(m2, run = TRUE))

## End(Not run)
```

---

umxFixAll *umxFixAll: Fix all free parameters*

Description

Fix all free parameters in a model using omxGetParameters()

Usage

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

Arguments

model	an OpenMx::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 OpenMx::mxModel()

References

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

See Also

Other Advanced Model Building Functions: umx, umxAlgebra(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("OneFactor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
```

```
m2 = umxFixAll(m1, run = TRUE, verbose = TRUE)
mxCompare(m1, m2)


## End(Not run)
```

---

umxGetLatents *Get the latentVars from a RAM model*

Description

Get the latentVars from a RAM model, optionally targeting a submodel.

Usage

```
umxGetLatents(model, targetModel = NULL)
```

Arguments

model	a umxRAM()
targetModel	name of the model to extract from

Value

- variables

See Also

- umxGetManifests(), umxRAM(), umxSuperModel()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
## Not run:
library(umx)
# Create two sets of data in which X & Y correlate ~ .4 in both datasets.
manifests = c("x", "y")
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 150,
  AA = 0, CC = .4, EE = .6, varNames = manifests)

grp1   = tmp[tmp$zygosity == "MZ", manifests]
g1Data = mxData(cov(grp1), type = "cov", numObs = nrow(grp1), means=umx_means(grp1))

grp2   = tmp[tmp$zygosity == "DZ", manifests]
g2Data = mxData(cov(grp2), type = "cov", numObs = nrow(grp2), means=umx_means(grp2))
# Model 1 (could add autoRun = FALSE if you don't want to run this as it is being built)
m1 = umxRAM("m1", data = g1Data,
```

```
 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 = g2Data,
 umxPath("x", to = "y", labels = "beta"),
 umxPath(var = manifests, labels=c("Var_x", "Resid_y_grp2")),
 umxPath(means = manifests, labels=c("Mean_x", "Mean_y"))
)

m3 = umxSuperModel('top', m1, m2)
umxGetLatents(m3)
umxGetLatents(m3, targetModel = "m1")

## End(Not run)
```

---

umxGetManifests *Get the manifestVars from a RAM model*

Description

Get the latentVars from a RAM model, optionally targeting a submodel.

Usage

```
umxGetManifests(model, targetModel = NULL)
```

Arguments

model	a umxRAM()
targetModel	name of the model to extract from

Value

- variables

See Also

- umxGetManifests(), umxRAM(), umxSuperModel()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
## Not run:
library(umx)
# Create two sets of data in which X & Y correlate ~ .4 in both datasets.
manifests = c("x", "y")
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 150,
  AA = 0, CC = .4, EE = .6, varNames = manifests)

grp1   = tmp[tmp$zygosity == "MZ", manifests]
g1Data = mxData(cov(grp1), type = "cov", numObs = nrow(grp1), means=umx_means(grp1))

grp2   = tmp[tmp$zygosity == "DZ", manifests]
g2Data = mxData(cov(grp2), type = "cov", numObs = nrow(grp2), means=umx_means(grp2))
# Model 1 (could add autoRun = FALSE if you don't want to run this as it is being built)
m1 = umxRAM("m1", data = g1Data,
 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 = g2Data,
 umxPath("x", to = "y", labels = "beta"),
 umxPath(var = manifests, labels=c("Var_x", "Resid_y_grp2")),
 umxPath(means = manifests, labels=c("Mean_x", "Mean_y"))
)

m3 = umxSuperModel('top', m1, m2)
umxGetManifests(m3)
umxGetManifests(m3, targetModel = "m1")

## End(Not run)
```

---

umxGetModel *Used to get a RAM submodel by name*

Description

Get any model from a RAM model, including submodels.

Usage

```
umxGetModel(model, targetModel = NULL)
```

Arguments

model	a umxRAM() model.
targetModel	name of the model to extract from

Value

- model

See Also

- umxGetManifests(), umxRAM(), umxSuperModel()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetParameters(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
## Not run:
library(umx)
# Create two sets of data in which X & Y correlate ~ .4 in both datasets.
manifests = c("x", "y")
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 150,
  AA = 0, CC = .4, EE = .6, varNames = manifests)

grp1   = tmp[tmp$zygosity == "MZ", manifests]
g1Data = mxData(cov(grp1), type = "cov", numObs = nrow(grp1), means=umx_means(grp1))

grp2   = tmp[tmp$zygosity == "DZ", manifests]
g2Data = mxData(cov(grp2), type = "cov", numObs = nrow(grp2), means=umx_means(grp2))
# Model 1 (could add autoRun = FALSE if you don't want to run this as it is being built)
m1 = umxRAM("m1", data = g1Data,
 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 = g2Data,
 umxPath("x", to = "y", labels = "beta"),
 umxPath(var = manifests, labels = c("Var_x", "Resid_y_grp2")),
 umxPath(means = manifests, labels = c("Mean_x", "Mean_y"))
)

m3 = umxSuperModel('top', m1, m2)
umxGetModel(m3)
umxGetModel(m3, targetModel = "m1")

## End(Not run)
```

---

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

Description

umxGetParameters retrieves parameter labels from a model, like OpenMx::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("labels", "values", "free", "lbound", "ubound", "all"),
  verbose = FALSE
)
```

Arguments

inputTarget	An object to get parameters from: could be a RAM OpenMx::mxModel()
regex	A regular expression to filter the labels. Default (NA) returns all labels. If vector, treated as raw labels to find.
free	A Boolean determining whether to return only free parameters.
fetch	What to return: "labels" (default) or "values", "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://github.com/tbates/umx

See Also

OpenMx::omxGetParameters(), parameters()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxParameters(), umx_aggregate(), umx_time()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
```

```
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

# Show all parameters
umxGetParameters(m1)
umxGetParameters(m1, free = TRUE)  # Only free parameters
umxGetParameters(m1, free = FALSE) # Only fixed parameters
# Complex regex pattern
umxGetParameters(m1, regex = "x[1-3]_with_x[2-5]", free = TRUE)

## End(Not run)
```

---

umxGSEM *Fit genomic Structural Equation Models*

Description

umxGSEM is a wrapper around umxRAM() and OpenMx WLS to fit genomic structural equation models (genomic SEM) using genetic covariance matrices (S) and sampling covariance matrices (V) estimated from LD Score Regression (LDSC).

Usage

```
umxGSEM(
  model,
  covstruc = NULL,
  S = NULL,
  V = NULL,
  estimation = c("DWLS", "WLS", "ULS"),
  name = "gsem",
  numObs = 1,
  smooth = TRUE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("yes", "no", "ordinal", "search"),
  std.lv = TRUE,
  ...
)
```

Arguments

model	A lavaan-style model string (also accepts umx ~= loadings), **or** an existing OpenMx::mxModel() / umxRAM() RAM model whose manifestVars name traits in S. For SNP GWAS, use umxGSEM_GWAS() instead of passing SNPs here.
covstruc	LDSC list with genetic covariance S and sampling covariance V (and typically I, N, m) as from GenomicSEM's ldsc / umxGSEM_ldsc() - see e.g. Psych_LDSC. If provided, S and V are extracted from it.

S	Genetic covariance matrix (if covstruc not provided).
V	Sampling covariance matrix (asymptotic covariance of S). If covstruc not provided.
estimation	Method for estimation. One of "DWLS", "WLS", or "ULS" (defaults to "DWLS").
name	The model name (defaults to "gsem").
numObs	Sample size passed to OpenMx for WLS bookkeeping. With LDSC V already on the sampling-covariance scale, the GSEM χ^2 is $r'Wr$; use numObs = 1 (default) so OpenMx reports that quantity directly. Larger dummy N rescales the displayed fit without changing point estimates.
smooth	Whether to smooth non-positive definite matrices using Matrix::nearPD() (defaults to TRUE).
autoRun	Whether to run the model (defaults to getOption("umx_auto_run")).
tryHard	Method for fitting the model ("no", "yes", "ordinal", "search"). Defaults to "yes" because DWLS on LDSC V often has multiple local minima.
std.lv	If TRUE (default), free the first factor loading and fix latent variances to 1 (unit-variance identification). Preferred for genetic covariance scale where unit-loading ID often produces Heywood cases. Passed to umxRAM() / lavaan.
...	Additional arguments passed to umxRAM().

Details

Data Triage and Matrix Smoothing

Genomic SEM relies on asymptotic covariance matrices (V) generated by LDSC. These matrices are frequently non-positive definite due to sampling variation or minor missingness across GWAS cohorts. umxGSEM implements an automatic triage system:

1. **Level 1 (Clean)**: Matrices are strictly positive definite and pass directly to the optimizer.
2. **Level 2 (Smoothed)**: Negative eigenvalues are detected. The matrices are smoothed using Matrix::nearPD to restore positive definiteness before estimation.

Fiduciary Warning: Models estimated using smoothed matrices are tagged with a gsem_triage attribute. Downstream functions like umxCompare will read this attribute and warn the user that Satorra-Bentler difference tests may report artificial precision.

Engine Integration umxGSEM automatically configures the C++ backend to cache the implied Jacobian matrix. This is a structural requirement for computing Satorra-Bentler (2010) scaled difference tests in umxCompare.

Tutorial files (1k-SNP sumstats, HapMap3 list, EUR LD scores) are fetched with umxGSEM_dl_tutorial_files(). Production HapMap3 / Alkes LD scores / 1000G plink are documented there; this helper does not download them. For a real model prefer Psych_LDSC rather than ldsc on the tutorial SNP list.

OpenMx WLS data (modern only)

Genomic SEM needs precomputed summary matrices S (genetic cov) and V (sampling cov of vech(S)). OpenMx WLS consumes these via the **modern** mxData interface only:

```
mxData(numObs = 1, type = "summary",
       observedStats = list(cov = S, useWeight = W, asymCov = V))
```

with `W = diag(1/diag(V))` for DWLS. `umxGSEM` also reorders `V` into OpenMx residual order (all variances, then free covariances) and sets dimnames `var_*` / `poly_*_*`.

umx hard-refuses forever the removed OpenMx WLS data interface (`type = "acov"` / `"none"`, `MxDataLegacyWLS`, top-level `acov=`/`fullWeight=`, or `observedStats$acov`/`$fullWeight` - historical name trap: `acov` meant useWeight). Use `observedStats` as above or raw data + `type = "DWLS"`.

Value

An `OpenMx::mxModel()` object of class `MxModelGSEM`.

References

Grotzinger, A. D., Rhemtulla, M., de Vlaming, R., Ritchie, G. R., Mallard, T. T., Hill, W. D., ... & Tucker-Drob, E. M. (2019). Genomic structural equation modeling. *Nature Human Behaviour*, **3**, 513-525. doi:10.1038/s415620190566x

See Also

Other GSEM: `Anthro_LDSC`, `Psych_LDSC`, `plot.umx_GWAS()`, `umxGSEM_GWAS()`, `umxGSEM_dl_tutorial_files()`, `umxGSEM_label_ldsc()`, `umxGSEM_ldsc()`, `umxGSEM_munge()`, `umxGSEM_std()`, `umxGSEM_sumstats()`, `umxSummary.MxDataStatic()`, `umxSummary.list()`

Examples

```
## Not run:
data(Psych_LDSC)
# umx path style (~=) or lavaan (=~); unit-variance ID is the default
m1 = umxGSEM(model = "g ~= SCZ + BIP + MDD + EA + INSOM",
             covstruc = Psych_LDSC, estimation = "DWLS")
umxSummary(m1)

# Explicit lavaan (same model)
m2 = umxGSEM(model = "g =~ NA*SCZ + BIP + MDD + EA + INSOM\ng ~~ 1*g",
             covstruc = Psych_LDSC, estimation = "DWLS")

# Existing umxRAM structure (manifest names must match S)
m0 = umxRAM("tmp", data = mxData(Psych_LDSC$S[1:3, 1:3], type = "cov", numObs = 2),
  type = "cov", autoRun = FALSE, std.lv = TRUE,
  umxPath("g", to = c("SCZ", "BIP", "MDD")),
  umxPath(var = c("SCZ", "BIP", "MDD")),
  umxPath(var = "g", fixedAt = 1)
)
m3 = umxGSEM(m0, covstruc = Psych_LDSC, estimation = "DWLS")

## End(Not run)
```

umxGSEM_dl_tutorial_files *Download tutorial-scale Genomic SEM files*

Description

Fetches the IBG / GenomicSEM practical pack (~45 MB): 1k-SNP SCZ/BIP/MDD sumstats, a tiny 1000G MAF table, HapMap3 `w_hm3.snplist`, and EUR LD scores (`eur_w_ld_chr/`). That is enough to run `umxGSEM_munge()` and see ldsc on a toy SNP list. It is **not** a production data installer.

Files land in `tools::R_user_dir("umx", "data")/GSEM_tutorial` unless `path` is set. 1k-SNP tables and LD scores come from the umx GitHub Release tag `gsem-tutorial` (not from the CRAN tarball). Existing files are left alone unless `overwrite = TRUE`.

ldsc on these SNPs is pedagogical. About 1k variants will not recover sensible heritabilities. For a real structural model use Psych_LDSC / Anthro_LDSC (or a covstruc from full-sumstat ldsc).

Production files (do not use this helper):

- Full GWAS sumstats from the relevant consortium (not "top hits").
- HapMap3 list: `https://zenodo.org/records/7773502/files/w_hm3.snplist.gz` (gunzip to `w_hm3.snplist`; header must include `SNP`).
- Local ldsc only: Alkes EUR LD scores `https://alkesgroup.broadinstitute.org/LDSCORE/eur_w_ld_chr.tar.b` (~500 MB -> ~1.2 GB) and `weights_hm3_noMHC.tgz` (~150 MB). Ancestry must match the GWAS. Skip if you already have a covstruc.
- 1000G EUR plink (~10-20 GB) only if sumstats lack allele frequency and you need an FRQ filter in munge. GenomicSEM's 1000G MAF table is the usual GSEM path, not bim/bed/bam.

Usage

```
umxGSEM_dl_tutorial_files(path = NULL, overwrite = FALSE, url = "default")
```

Arguments

`path`	Directory to write into. Default: `file.path(tools::R_user_dir("umx", "data"), "GSEM_tutorial")`.
`overwrite`	If `TRUE`, re-download the Release tarball even when files already exist. Default `FALSE`.
`url`	Tarball URL, or a path to a local `.tar.gz`. Default is the `gsem-tutorial` Release asset on `tbates/umx`.

Value

A named list of absolute paths: `dir`, `hm3`, `ld`, `sumstats` (character vector of three subset files), `ref`.

See Also

umxGSEM_munge(), umxGSEM(), umxGSEM_GWAS(), Psych_LDSC

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
tut = umxGSEM_dl_tutorial_files()
umxGSEM_munge(tut$sumstats, hm3 = tut$hm3, trait.names = c("SCZ", "BIP", "MDD"),
 Ns = c(105318, 16731, 173005), output_dir = tut$dir)
data(Psych_LDSC)
m1 = umxGSEM("g ~= SCZ + BIP + MDD", covstruc = Psych_LDSC)

## End(Not run)
```

---

umxGSEM_GWAS *Multivariate genomic SEM GWAS (SNP on a factor or user model)*

Description

Expands the LDSC genetic covariance structure with each SNP (using prepared betas from umxGSEM_sumstats()), fits DWLS/WLS, and returns SNP-level results. For a default common-factor model, estimates F1 ~ SNP.

Usage

```
umxGSEM_GWAS(
  covstruc,
  SNPs,
  model = NULL,
  estimation = c("DWLS", "WLS", "ULS"),
  traits = NULL,
  GC = c("standard", "conserv", "none"),
  uncertainty = c("MLR", "SE"),
  SnpSamplingError = 5e-04,
  maxSNPs = NULL,
  snpEffect = "SNP_to_F1",
  quiet = TRUE,
  fix_measurement = TRUE,
  force_fallback = FALSE
)
```

Arguments

covstruc	LDSC list with S (genetic covariance), V (sampling covariance of vech(S)), and preferably I (LDSC intercept matrix, for GC-corrected SEs). See e.g. Psych_LDSC (I = intercepts / sample-overlap; also N, m).
SNPs	Data frame from umxGSEM_sumstats() (columns beta.*, se.*, MAF, SNP, ...).
model	Optional lavaan/umx string or mxModel. If NULL, builds A RAM model with one latent (F1) loading on all the traits, and "SNP" loading on F1
estimation	"DWLS" (default), "WLS", or "ULS".
traits	Trait names. Default: colnames of covstruc$S.
GC	Genomic control for SNP sampling variances: "standard", "conserv", or "none".
uncertainty	Whether to compute the robust sandwich SE ("MLR", default) or naive SE ("SE").
SnpSamplingError	SE used for SNP variance (treated as nearly fixed; default 5e-4).
maxSNPs	Optional limit for smoke tests.
snpEffect	Character; path label to extract from the model (default "SNP_to_F1").
quiet	Suppress per-SNP messages.
fix_measurement	Logical; if TRUE (default), fixes the non-SNP measurement model parameters to their base optimized values when evaluating each SNP (only applies if analytic engine is bypassed).
force_fallback	Logical; internal use for testing to force the non-vectorized analytic path (default FALSE).

Details

The LDSC object is the usual GenomicSEM-style list (S, V, preferably I); see e.g. Psych_LDSC for a packaged example and field definitions.

Value

A data.frame of SNP results (SNP, CHR, BP, MAF, est, se, Z, P, status, ...).

See Also

umxGSEM_sumstats(), umxGSEM(), Psych_LDSC

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_dl_tutorial_files(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
1. load data
data(Psych_LDSC)
# Have a quick look if you like
# Psych_LDSC fields: S, V, I, N, m (see ?Psych_LDSC). V pairs are named var_SCZ, poly_BIP_SCZ, ...
umxSummary(Psych_LDSC)

#' # 2. build the snp data structure from your SNP data txt files, e.g.
dir = "~/bin/umx/inst/developer/GenomicSEM"
snps = umxGSEM_sumstats(
  files = file.path(dir, c("SCZ_subset.txt", "BIP_subset.txt", "MDD_subset.txt")),
  ref = file.path(dir, "reference.1000G.subset.txt"),
  trait.names = c("SCZ", "BIP", "MDD"), se.logit = TRUE
)

# 3. Run the GWAS on your trait measurement model adding SNP effects into the latent structure
GWAS = umxGSEM_GWAS(covstruc = Psych_LDSC, SNPs = snps,
  traits = c("SCZ", "BIP", "MDD"), maxSNPs = 20)

# 4. Examine results and make a new drug or something useful!
head(GWAS)
# Manhattan Plot (default)
plot(gwas_res)
plot(example_gwas, chromosomes= 1, label_top_n= 4, title="IQ GWAS") + theme_plos()
volcano_plot(example_gwas, size_by = "AF", label_top_n = 3)

# Q-Q Plot
plot(gwas_res, type = "qq")

# ================
# = Another demo =
# ================
# 1. Load the reference LDSC covariance structure
data(Psych_LDSC, package = "umx")

# 2. Generate a million simulated SNPs
traits = c("SCZ", "BIP", "MDD")
t0 = Sys.time()
snps = umxGSEM_sim_snps(n = 1e6, traits = traits)
print(Sys.time() - t0)

# 3. Run the GWAS and time it
t0 = Sys.time()
gwas_res = umxGSEM_GWAS(covstruc = Psych_LDSC, SNPs = snps, traits = traits)
print(Sys.time() - t0)

# 4. View the results
head(gwas_res)
# Manhattan Plot (default)
plot(gwas_res)
plot(example_gwas, chromosomes= 1, label_top_n= 4, title="IQ GWAS") + theme_plos()
```

```
volcano_plot(example_gwas, size_by = "AF", label_top_n = 3)

# Q-Q Plot
plot(gwas_res, type = "qq")


## End(Not run)
```

`umxGSEM_label_ldsc` *Label or Assemble S, V, I, N into an LDSC-style covstruc list*

Description

Takes an existing `covstruc` list (from GenomicSEM) OR loose `S` and `V` matrices, validates them, and applies strict OpenMx WLS-compliant dimnames (`var_X`, `poly_Y_X`). This guarantees that external GenomicSEM data correctly aligns with OpenMx's WLS engine.

Usage

```
umxGSEM_label_ldsc(
  covstruc = NULL,
  S = NULL,
  V = NULL,
  I = NULL,
  N = NULL,
  traits = NULL,
  overwrite = FALSE
)
```

Arguments

`covstruc`	An optional list with at least matrix `S` (genetic covariance).
`S`	A genetic covariance matrix (if assembling from loose matrices).
`V`	An asymptotic covariance matrix (if assembling from loose matrices).
`I`	An optional intercept matrix.
`N`	An optional sample size matrix or vector.
`traits`	Character vector of trait names. Default: `colnames(S)`, else `rownames(S)`, else `paste0("T", seq_len(ncol(S)))`.
`overwrite`	If `FALSE` (default), only fill missing dimnames; if `TRUE`, replace existing dimnames.

Value

A list with matrices `S`, `V`, (and optionally `I`, `N`) properly labelled.

See Also

Psych_LDSC, umxGSEM()

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files, umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
data(Psych_LDSC)
# Re-apply labels safely
Psych_LDSC = umxGSEM_label_ldsc(Psych_LDSC)
colnames(Psych_LDSC$V)[1:3]
```

---

umxGSEM_ldsc *Multivariate LD score regression (OpenMx C++ engine)*

Description

Runs multivariate LD score regression on munged .sumstats.gz files from umxGSEM_munge() via OpenMx imxLDSC. The return value is a umx covstruc: S and I named by trait, V / N named with OpenMx residual labels (var_SCZ, poly_BIP_SCZ, ...) so it can be passed to umxGSEM() as covstruc.

Usage

```
umxGSEM_ldsc(
  traits,
  sample.prev,
  population.prev,
  ld,
  wld,
  trait.names = NULL,
  sep_weights = FALSE,
  chr = 22,
  n.blocks = 200,
  ldsc.log = NULL,
  stand = FALSE,
  select = FALSE,
  chisq.max = NA
)
```

Arguments

traits	Character vector of paths to munged .sumstats.gz files (the return value of umxGSEM_munge()).
sample.prev	Numeric vector of sample prevalences (cases / N). Use NA for continuous traits.

population.prev
: Numeric vector of population prevalences for liability-scale conversion. Use NA for continuous traits.

ld
: Path to folder containing LD score files (e.g. eur_w_ld_chr/, or $ld from umxGSEM_dl_tutorial_files()).

wld
: Path to weight LD score files. Ignored when sep_weights = FALSE (then ld is used for both, matching GenomicSEM).

trait.names
: Optional names for the traits (default: munged file basenames with .sumstats.gz stripped).

sep_weights
: If TRUE, read regression weights from wld instead of ld.

chr
: Highest chromosome to include when select = FALSE (default 22).

n.blocks
: Jackknife blocks for V (default 200). Raised automatically when K > 18, as in GenomicSEM.

ldsc.log
: Optional path prefix for a short text log. If NULL, no log file is written (unlike GenomicSEM, which always writes one).

stand
: If TRUE, also return S_Stand / V_Stand (genetic correlations and their sampling covariance).

select
: Chromosome subset: FALSE (use 1:chr), "ODD", "EVEN", or a numeric vector of chromosome numbers.

chisq.max
: Optional chi-square outlier cutoff (NA = engine default).

Details

Tutorial 1k-SNP files will run but the estimates are not usable (too few SNPs). For a real structural model use full-sumstat ldsc or Psych_LDSC.

S is K x K genetic covariance (heritabilities on the diagonal). V is the sampling covariance of vech(S), length K(K+1)/2. I holds LDSC intercepts (sample overlap / stratification). Binary traits: supply both sample.prev and population.prev for liability scaling. Continuous traits: NA (a lone 0.5 with population.prev = NA is treated as continuous).

Value

A list with S, V, I, N, m (and S_Stand/V_Stand if stand). Ready for umxGSEM() / umxGSEM_label_ldsc().

See Also

umxGSEM_munge(), umxGSEM(), Psych_LDSC

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files, umxGSEM_label_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
tut = umxGSEM_dl_tutorial_files()
ss = umxGSEM_munge(
```

```
  files = tut$sumstats, hm3 = tut$hm3,
  Ns = c(105318, 16731, 173005),
  trait.names = c("SCZ", "BIP", "MDD"),
  output_dir = tut$dir, overwrite = TRUE
)
# Pedagogical only: ~1k SNPs. Use data(Psych_LDSC) for a real model.
covstruc = umxGSEM_ldsc(
  traits = ss,
  sample.prev = c(0.39, 0.45, 0.35),
  population.prev = c(0.01, 0.01, 0.16),
  ld = tut$ld, wld = tut$ld,
  trait.names = c("SCZ", "BIP", "MDD"),
  n.blocks = 20
)
umxSummary(covstruc)

## End(Not run)
```

---

umxGSEM_munge *Munge Genomic SEM Summary Statistics*

Description

umxGSEM_munge allows you to format and filter summary statistics files for Genomic SEM / LDSC using the high-speed OpenMx C++ munging engine. munge creates the *.sumstats.gz files for umxGSEM_ldsc().

For a tutorial run, umxGSEM_dl_tutorial_files() supplies 1k-SNP sumstats plus w_hm3.snplist. For production, download full consortium sumstats and gunzip the Zenodo HapMap3 list (https://zenodo.org/records/7 Local ldsc also needs Alkes EUR LD scores (https://alkesgroup.broadinstitute.org/LDSCORE/eur_w_ld_chr.tar.b skip that if you already have a covstruc such as Psych_LDSC. 1000G EUR plink (~10-20 GB) is only required if sumstats lack allele frequency and you need an FRQ filter.

Usage

```
umxGSEM_munge(
  files = NULL,
  hm3 = "w_hm3.snplist",
  Ns = NULL,
  trait.names = NULL,
  info.filter = 0.9,
  maf.filter = 0.01,
  column.names = list(),
  output_dir = getwd(),
  cores = -1,
  overwrite = FALSE,
  ...
)
```

Arguments

files	A character vector of summary statistics files to munge.
hm3	Path to the HapMap3 SNP reference list (e.g. $hm3 from umxGSEM_dl_tutorial_files(), or a production w_hm3.snplist).
Ns	A numeric vector of sample sizes (one per file). If a single value is provided, it will be replicated for all files.
trait.names	A character vector of names for the traits. Defaults to the basenames of the files.
info.filter	Info score threshold filter (defaults to 0.9).
maf.filter	Minor Allele Frequency threshold filter (defaults to 0.01).
column.names	A list of custom column name mappings. Synonyms are automatically resolved if left empty.
output_dir	Path to the directory where munged files should be saved. Defaults to getwd().
cores	Number of CPU cores/threads to use. Defaults to -1 (uses all available threads).
overwrite	Whether to overwrite existing munged files (defaults to FALSE).
...	Additional arguments passed to OpenMx imxMunge when available (GenomicMx).

Details

umxGSEM_munge will intelligently search for synonyms for heading names: SNP: snp, rsid, marker, snpid, rs, markername A1: a1, allele1, effect_allele, inc_allele, reference_allele A2: a2, allele2, non_effect_allele, dec_allele, other_allele P: p, p-value, pval, p.value effect: effect, or, beta, logor, est, estimate

note: Ensure that the summary statistics are full, not "23&me" style "top x thousand SNPs". Nor just pruned SNPs.

Value

A character vector of paths to the munged files.

See Also

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
tut = umxGSEM_dl_tutorial_files()
umxGSEM_munge(
  files = tut$sumstats,
  hm3 = tut$hm3,
  Ns = c(105318, 16731, 173005),
  trait.names = c("SCZ", "BIP", "MDD"),
  output_dir = tut$dir,
  overwrite = TRUE
```

```
)

## End(Not run)
```

`umxGSEM_sim_snps` *Simulate GWAS Summary Statistics for Genomic SEM*

Description

Generates a simulated summary statistics dataset (SNPs) suitable for testing `umxGSEM_GWAS`.

Usage

```
umxGSEM_sim_snps(n = 1e+06, traits = c("SCZ", "BIP", "MDD"), seed = 42)
```

Arguments

`n`	Number of SNPs to simulate. Default is 1,000,000.
`traits`	A character vector of traits to generate betas and SEs for.
`seed`	Random seed for reproducibility.

Value

A `data.frame` with columns for SNP, CHR, BP, A1, A2, MAF, and `beta.[trait]`/`se.[trait]` for each trait.

See Also

Other Data Functions: `noNAs()`, `prolific_anonymize()`, `prolific_check_ID()`, `prolific_read_demog()`, `prolific_scoring_stub()`, `umx`, `umxFactor()`, `umxHetCor()`, `umx_as_numeric()`, `umx_cont_2_quantiles()`, `umx_lower2full()`, `umx_make_GxE_data()`, `umx_make_MR_data()`, `umx_make_TwinData()`, `umx_make_fake_data()`, `umx_make_raw_from_cov()`, `umx_merge_randomized_columns()`, `umx_polychoric()`, `umx_polypairwise()`, `umx_polytriowise()`, `umx_read_lower()`, `umx_rename()`, `umx_reorder()`, `umx_score_scale()`, `umx_select_valid()`, `umx_stack()`, `umx_strings2numeric()`

Examples

```
snps = umxGSEM_sim_snps(n = 1000, traits = c("SCZ", "BIP", "MDD"))
head(snps)
```

umxGSEM_std *Standardize GSEM S and V matrices to the correlation metric via Delta Method*

Description

Standardizes the genetic covariance matrix S and propagates the uncertainty to the asymptotic sampling covariance matrix V using the Delta method. Standardizes the LDSC intercepts matrix I by dividing by the heritabilities.

Usage

```
umxGSEM_std(covstruc = NULL, S = NULL, V = NULL)
```

Arguments

covstruc	A list containing the genetic covariance matrix S, its asymptotic covariance V, and optional intercepts I.
S	Optional genetic covariance matrix.
V	Optional asymptotic covariance matrix.

Details

Standardizing S to a correlation matrix R and scaling V accordingly is highly recommended for model fitting to prevent numerical underflow and improve optimizer convergence rates when genetic variances are very small.

Value

A list containing the standardized covariance structure:

S	Standardized genetic correlation matrix.
V	Standardized asymptotic covariance matrix of the elements of S.
I	Standardized LDSC intercepts matrix (if present).

See Also

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files, umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_sumstats(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
# Example usage:
# std_cov = umxGSEM_std(covstruc)
```

umxGSEM_sumstats *Prepare multivariate SNP summary statistics for genomic SEM GWAS*

Description

Aligns per-trait GWAS summary statistic files to a 1000 Genomes reference (allele match + MAF filter), converts odds ratios to the log scale when needed, and applies the continuous-scale transformation used in Genomic SEM so that effects and standard errors are on a unit-variance phenotype scale. The result is ready for `umxGSEM_GWAS()`.

The returned `data.frame` has columns **SNP**, **CHR**, **BP**, **MAF**, **A1**, **A2**, and `beta.*` / `se.*` **per trait** (listwise-complete SNPs present in the reference and in every trait file).

Usage

```
umxGSEM_sumstats(
  files,
  ref,
  trait.names = NULL,
  se.logit = TRUE,
  OLS = FALSE,
  linprob = FALSE,
  N = NULL,
  info.filter = 0.6,
  maf.filter = 0.01,
  baseDir = getwd()
)
```

Arguments

`files`	Character vector of GWAS summary statistic file names or paths (same trait order as LDSC). Relative names are resolved under `baseDir` (default: working directory).
`ref`	Path or file name of 1000G reference with columns SNP, CHR, BP, MAF, A1, A2 (resolved under `baseDir` when relative).
`trait.names`	Character vector of trait names for output columns (defaults to file basenames).
`se.logit`	Logical (or vector, one per trait). If `TRUE` (default), treat effects/SEs as logistic-scale log(OR) and apply the Genomic SEM continuous-scale transform.
`OLS`	Logical (or vector). If `TRUE`, treat the trait as continuous OLS and standardize with Z and total `N` (see Details).
`linprob`	Logical (or vector). If `TRUE`, recover continuous-scale effects from Z and effective `N` for binary traits (see Details).
`N`	Optional numeric sample sizes (total N for OLS; sum of effective N for linprob). Recycled to `length(files)`.
`info.filter`	Numeric INFO threshold (default `0.6`); applied when INFO is present.

`maf.filter`	Numeric MAF threshold applied to the reference (default `0.01`).
`baseDir`	Directory for relative `files` and `ref` (default `getwd()`). Absolute paths are left unchanged.

Details

Example input (SCZ subset):

Workshop-style GWAS files are whitespace-separated. A few lines of the package toy file `SCZ_subset.txt` look like:

SNP	Freq.A1	CHR	BP	A1	A2	OR	SE	P
rs1000000	0.7763	12	126890980	G	A	1.0195	0.011559	0.0954
rs10000010	0.5099	4	21618674	T	C	0.9910	0.009582	0.3452
rs1000002	0.5199	3	183635768	C	T	1.0048	0.009435	0.6113

Column synonyms are resolved automatically (`SNP`/`rsid`, `OR`/`beta`/`effect`, `SE`/`stderr`, `INFO`, `N`, allele columns, etc.). When the median effect is near 1, values are treated as odds ratios and logged.

Example output:

After aligning SCZ, BIP, and MDD to a 1000G reference with `se.logit = TRUE` (toy subsets under `inst/developer/GenomicSEM/`), the first rows resemble:

SNP	CHR	BP	MAF	A1	A2	beta.SCZ	se.SCZ	beta.BIP	se.BIP	beta.MDD	se.MDD
rs1000073	1	157255396	0.417	A	G	-0.00053	0.00543	0.00696	0.0133	-0.00105	0.00447
rs1000050	1	162736463	0.147	C	T	0.00614	0.00742	-0.0150	0.0182	-0.00033	0.00700
rs1000053	2	12790328	0.090	C	T	0.00149	0.00915	-0.0187	0.0240	-0.00226	0.00744

Reference alleles and MAF come from `ref`; betas are flipped so that A1 matches the reference. SNPs that fail allele match, INFO, or listwise merge are dropped.

Scale flags (`se.logit`, `OLS`, `linprob`):

Pass a single logical or a vector recycled to `length(files)`. For each trait, choose the treatment that matches how that GWAS was run:

Flag	Typical traits	What it does
`se.logit = TRUE`	Case/control logistic GWAS (default)	Treat `effect`/`SE` as log(OR) and SE or
`linprob = TRUE`	Binary traits lacking usable logistic SEs	Reconstruct Z from effect/SE when pre
`OLS = TRUE`	Continuous phenotypes	Standardize using Z and total sample s
`se.logit = FALSE` (and not OLS/linprob)	Non-standard OR-scale SEs	Alternate SE scaling for odds-ratio rep

Other parameters:

- `files`: Paths (or basenames under `baseDir`) to one GWAS file per trait, in the same trait order you will use for LDSC / `umxGSEM_GWAS()`.

- ref: 1000 Genomes (or equivalent) reference table with columns SNP, CHR, BP, MAF, A1, A2. Used for allele alignment, MAF filter, and SNP annotation in the return value.
- trait.names: Labels for beta.* / se.* columns (default: file basenames without extension). Must match LDSC trait names when you expand into GWAS.
- N: Optional sample sizes, one per trait (or recycled). Required for meaningful OLS / linprob transforms (total N for OLS; effective N sum for linprob).
- info.filter: Drop SNPs with INFO below this value when an INFO column is present (default 0.6). Missing INFO is kept.
- maf.filter: Drop reference SNPs with MAF below this threshold (default 0.01). MAF is folded so it is at most 0.5.
- baseDir: Directory prepended to relative files and ref paths. Absolute paths are left unchanged.

Processing is **listwise**: a SNP must appear in the reference and in every trait file after QC. Expect a message like umxGSEM_sumstats: N SNPs after listwise merge with reference.

Value

A data.frame with columns SNP, CHR, BP, MAF, A1, A2, and beta.<trait> / se.<trait> for each trait in trait.names, sorted by CHR/BP.

References

Grotzinger, A. D., Rhemtulla, M., de Vlaming, R., Ritchie, S. J., Mallard, T. T., Hill, W. D., Ip, H. F., Marioni, R. E., McIntosh, A. M., Deary, I. J., Koellinger, P. D., Harden, K. P., Nivard, M. G., & Tucker-Drob, E. M. (2019). Genomic structural equation modelling provides insights into the multivariate genetic architecture of complex traits. *Nature Human Behaviour*, **3**, 513-525. doi:10.1038/s415620190566x

Tutorial materials and sumstats conventions follow the Genomic SEM documentation and workshops (see https://github.com/GenomicSEM/GenomicSEM).

See Also

umxGSEM_GWAS(), umxGSEM()

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files(), umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxSummary.MxDataStatic(), umxSummary.list()

Examples

```
## Not run:
# basenames only (files live in baseDir)
snps = umxGSEM_sumstats(
  files = c("SCZ_subset.txt", "BIP_subset.txt", "MDD_subset.txt"),
  ref = "reference.1000G.subset.txt",
  trait.names = c("SCZ", "BIP", "MDD"),
  se.logit = TRUE,
  baseDir = "path/to/umx/inst/developer/GenomicSEM"
)
```

```
head(snps)
# columns: SNP, CHR, BP, MAF, A1, A2, beta.SCZ, se.SCZ, beta.BIP, ...

## End(Not run)
```

---

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

Description

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

Usage

```
umxGxE(
  name = "G_by_E",
  selDVs,
  selDefs,
  dzData,
  mzData,
  sep = NULL,
  data = NULL,
  zyg = "zygosity",
  digits = 3,
  lboundACE = NA,
  lboundM = NA,
  dropMissingDef = TRUE,
  dzAr = 0.5,
  dzCr = 1,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  selCovs = NULL
)
```

Arguments

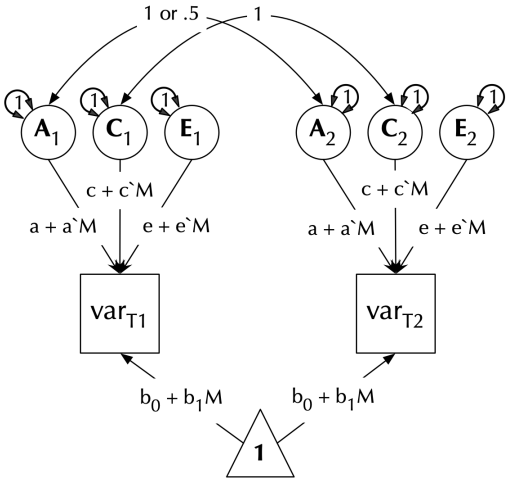
name	The name of the model (default= "G_by_E")
selDVs	The dependent variable (e.g. "IQ")
selDefs	The definition variable (e.g. "SES")
dzData	The DZ dataframe containing the Twin 1 and Twin 2 DV and moderator (4 columns)

`mzData`	The MZ dataframe containing the Twin 1 and Twin 2 DV and moderator (4 columns)
`sep`	How to expand selDVs into full names, i.e., "_T" makes "var" -> "var_T1" and "var_T2"
`data`	If provided, dzData and mzData are treated as valid levels of zyg to select() data sets (default = NULL)
`zyg`	If data provided, this column is used to select rows by zygosity (Default = "zygosity")
`digits`	Rounding precision for tables (default 3)
`lboundACE`	If not NA, then lbound the main effects at this value (default = NA, can help to set this to 0)
`lboundM`	If not NA, then lbound the moderator effects at this value (default = NA, can help to set this to 0)
`dropMissingDef`	Whether to automatically drop missing def var rows for the user (default = TRUE). You get a polite note.
`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`	Optionally run the model (default), or just to create it and return without running.
`tryHard`	Optionally tryHard to get the model to converge (Default = 'no'). "yes" uses mxTryHard. Other options: "ordinal", "search".
`optimizer`	Optionally set the optimizer (default NULL does nothing)
`selCovs`	(optional) covariates to include in the means model (do not include sep in names)

Details

Paths are linear in the moderator \(M\): \(a_i = a + a_m M\) (and likewise for \(c\), \(e\)). Estimated main effects (`a`, `c`, `e`) and moderation slopes (`am`, `cm`, `em`) live on the `top` submodel.

The following figure shows the GxE model as a path diagram:



Value

- GxE `OpenMx::mxModel()`

Synthetic data and recovery checks

For benchmarking and teaching, generate wide twin data with known Purcell paths via `umx_make_GxE_data()`. That helper returns a data frame ready for `umxGxE` (columns `zygosity`, `outcome_T*`, `outcomeAge_T*`, `mod_T*`, `age_T*`) plus attribute `"truth"` with the generative parameters. The attribute is ignored by fitting; use it only to compare estimates to ground truth, e.g. `attr(df, "truth")$am`.

Defaults in the simulator: \(a=0.50\), \(c=0.30\), \(e=0.60\), \(a_m=0.15\), \(c_m=e_m=0\). Two phenotypes share the same latent ACE draws:

`outcome` Pure GxE (no age mean effect). Fit with `selDVs = "outcome"`, `selDefs = "mod"`.

`outcomeAge` Same residuals plus a mean effect of age (target correlation of approximately 0.2 with age). Fit with `selDVs = "outcomeAge"`, `selDefs = "mod"`, `selCovs = "age"`.

Moderator and age are shared within family by default (`mod_T1 == mod_T2`, `age_T1 == age_T2`).

Power: Moderated ACE is low-powered. Even with thousands of pairs, recovery of \(a_m\) is noisy; large absolute error relative to the true slope is expected in single replications. Prefer `tryHard = "yes"`, adequate \(N\), and simulation studies rather than one-shot recovery demos.

References

Purcell, S. (2002). Variance components models for gene-environment interaction in twin analysis. *Twin Research*, **6**, 554-571. doi:10.1375/twin.5.6.554

See Also

`umx_make_GxE_data()`, `umxGxE_window()`, `umxReduce()`, `umxSummary()`

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
## Not run:
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = "bmi"
selDefs = "age"
mzData  = subset(twinData, zygosity == "MZFF")[1:100,]
dzData  = subset(twinData, zygosity == "DZFF")[1:100,]
m1 = umxGxE(selDVs= "bmi", selDefs= "age", sep= "", dzData= dzData, mzData= mzData, tryHard= "yes")

# Select the data on the fly with data= and zygosity levels
```

```
m1 = umxGxE(selDVs= "bmi", selDefs= "age", sep="", dzData= "DZFF", mzData= "MZFF", data= twinData)

# ================================================================
# = example with Twins having different values of the moderator =
# ================================================================

twinData$age1 = twinData$age2 = twinData$age
tmp = twinData
tmp$age2 = tmp$age2 +rnorm(n=length(tmp$age2))
selDVs  = "bmi"
selDefs = "age"
mzData = subset(tmp, zygosity == "MZFF")
dzData = subset(tmp, zygosity == "DZFF")
m1 = umxGxE(selDVs= "bmi", selDefs= "age", sep= "", dzData= dzData, mzData= mzData, tryHard= "yes")

# ====================================
# = Controlling output of umxSummary =
# ====================================
umxSummaryGxE(m1)
umxSummary(m1, location = "topright")
umxSummary(m1, separateGraphs = TRUE)

# # Test dropping moderation on a path
m2 = umxModify(m1, regex = "am_.*", comparison = TRUE, tryHard = "yes")

# umxReduce knows how to test all relevant hypotheses for GxE models,
# reporting these in a nice table.
umxReduce(m1)

# ============================================================
# = Synthetic ground-truth data: umx_make_GxE_data + umxGxE =
# ============================================================
# Generate N pairs with known a, c, e, am (see attr(df, "truth"))
df = umx_make_GxE_data(nMZpairs = 800, nDZpairs = 800, seed = 1)
truth = attr(df, "truth")
truth[c("a", "c", "e", "am", "cm", "em")]

# Pure GxE phenotype (no age in means)
mGxE = umxGxE(selDVs = "outcome", selDefs = "mod", sep = "_T",
  data = df, mzData = "MZ", dzData = "DZ", tryHard = "yes")
c(est_am = mGxE$top$am$values[1, 1], true_am = truth$am)

# Same latents + age mean effect via selCovs
mAge = umxGxE(selDVs = "outcomeAge", selDefs = "mod", selCovs = "age",
  sep = "_T", data = df, mzData = "MZ", dzData = "DZ", tryHard = "yes")

# Custom generative slopes (e.g. null am, nonzero em)
df2 = umx_make_GxE_data(nMZpairs = 500, nDZpairs = 500,
  am = 0, em = 0.10, seed = 2)
attr(df2, "truth")[c("am", "em")]

## End(Not run)
```

umxGxEbiv *Purcell (2002) Bivariate GxE model: Suitable when twins differ on the moderator.*

Description

GxE interaction models test the hypothesis that the strength of genetic and environmental influences vary parametrically across levels of a measured environment.

Usage

```
umxGxEbiv(
  name = "GxEbiv",
  selDVs,
  selDefs,
  dzData,
  mzData,
  sep = NULL,
  lboundACE = 0,
  lboundM = NA,
  dropMissingDef = FALSE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL
)
```

Arguments

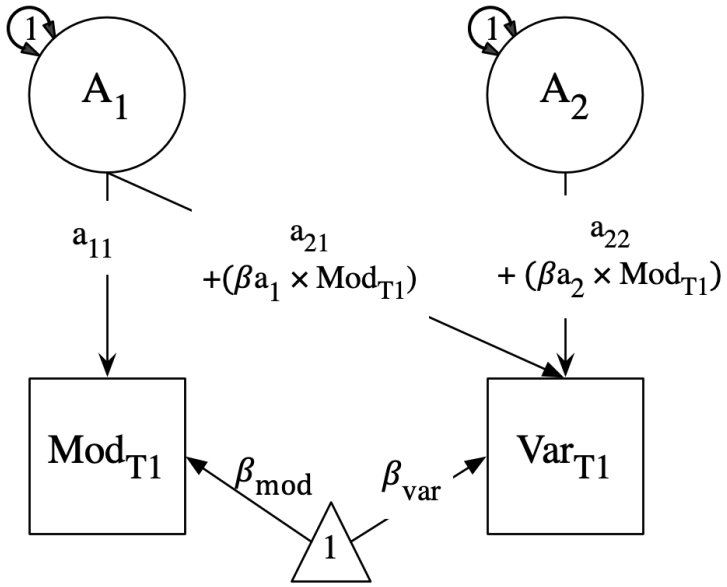
name	The name of the model (defaults to "GxEbiv")
selDVs	The dependent variable (e.g. IQ)
selDefs	The definition variable (e.g. socioeconomic status)
dzData	The DZ dataframe containing the Twin 1 and Twin 2 DV and moderator (4 columns)
mzData	The MZ dataframe containing the Twin 1 and Twin 2 DV and moderator (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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
optimizer	Optionally set the optimizer (default NULL does nothing)

Details

Whereas univariate `umxGxE()` models assume the twins share the moderator, or have zero correlation on the moderator, `umxGxEbiv()` allows testing moderation in cases where members of a twin pair differ on the moderator, (Purcell, 2002; van der Sluis et al., 2012).

This is the same model we teach at Boulder.

The following figure shows this bivariate GxE model as a path diagram (Twin 1 shown). Whereas the univariate model incorporates the moderator in the means model, the bivariate model incorporates the moderator as a first class variable, with its own ACE structure, shared pathways to the trait of interest, and the ability to moderate both specific and shared A, C, and E, influences on the trait of interest.



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 `OpenMx::mxModel()`

References

- Purcell, S. (2002). Variance components models for gene-environment interaction in twin analysis. *Twin Research*, **6**, 554-571. doi:10.1375/twin.5.6.554.
- van der Sluis, S., Posthuma, D., & Dolan, C. V. (2012). A note on false positives and power in G x E modelling of twin data. *Behavior Genetics*, **42**, 170-186. doi:10.1007/s1051901194803.

See Also

- `plot()`, `umxSummary()`, `umxReduce()`

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
require(umx)
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
umxSummaryGxEbiv(m1)
umxSummary(m1, location = "topright")
umxSummary(m1, separateGraphs = FALSE)
m2 = umxModify(m1, update = c("cBeta2_r1c1", "eBeta1_r1c1", "eBeta2_r1c1"), comparison = TRUE)

# TODO: teach umxReduce to test all relevant hypotheses for umxGxEbiv
umxReduce(m1)

## End(Not run)
```

---

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

```
umxGxE_window(
  selDVs = NULL,
  moderator = NULL,
  mzData = mzData,
  dzData = dzData,
  sep = NULL,
  weightCov = FALSE,
  target = NULL,
  width = 1,
  plotWindow = FALSE,
  tryHard = c("no", "yes", "ordinal", "search"),
  return = c("estimates", "last_model")
)
```

Arguments

selDVs	The dependent variables for T1 and T2, e.g. c("bmi_T1", "bmi_T2")
moderator	The name of the moderator variable in the dataset e.g. "age", "SES" etc.
mzData	Dataframe containing the DV and moderator for MZ twins
dzData	Dataframe containing the DV and moderator for DZ twins
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 the data window.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
return	whether to return the last model (useful for specifiedTargets) or the list of estimates (default = "estimates")

Value

- Table of estimates of ACE along the moderator

References

- Hildebrandt, A., Wilhelm, O, & Robitzsch, A. (2009). Complementary and competing factor analytic approaches for the investigation of measurement invariance. *Review of Psychology*, **16**, 87–107.
- Briley, D.A., Harden, K.P., Bates, T.C., Tucker-Drob, E.M. (2015). Nonparametric Estimates of Gene x Environment Interaction Using Local Structural Equation Modeling. *Behavior Genetics*, **45**, 581-96. doi:10.1007/s1051901597328.

See Also

umxGxE()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
library(umx);

# ==============================
# = 1. Open and clean the data =
# ==============================
# umxGxE_window takes a data.frame 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 full 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")
dzData = subset(twinData, zygosity == "DZFF")

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

umxGxE_window(selDVs = "bmi", sep="", moderator = mod, mzData = mzData, dzData = dzData,
  target = 40, plotWindow = TRUE, tryHard = "yes")

# Run with tryHard
umxGxE_window(selDVs = "bmi", sep="", moderator = "age", mzData = mzData, dzData = dzData)
umxGxE_window(selDVs="bmi", sep="", moderator="age", mzData=mzData, dzData=dzData, tryHard="yes")

# Run creating weighted covariance matrices (excludes missing data)
umxGxE_window(selDVs = "bmi", sep="", moderator= "age", mzData = mzData, dzData = dzData,
  weightCov = TRUE)
# This example runs multiple target moderator values
mxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData,
 target = c(39,40,50), plotWindow = TRUE)
```

```
## End(Not run)
```

---

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.
return	Return just the correlations (default) or the hetcor object (contains, method, SEs etc.)
std.err	Compute the SEs? (default = FALSE)

Details

Factor columns with fewer than two observed (non-NA) levels are excluded from the hetcor call (polychoric/polyserial is undefined). Their diagonal is 1 and off-diagonals are 0. This is the double-entry _cens case: that column is only ever "censored" or NA, so there is no pair to estimate.

Value

- A matrix of correlations

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

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

---

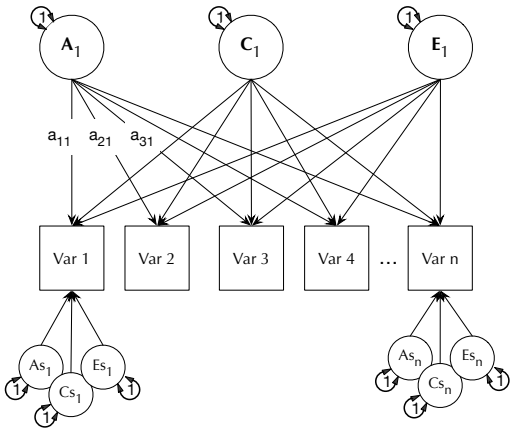
umxIP *umxIP: Build and run an Independent Pathway twin model*

Description

Make a 2-group Independent Pathway twin model.

The independent-pathway model (aka "biometric model" (McArdle and Goldsmith, 1990) proposes that A, C, and E components act directly on the manifest or measured phenotypes. This contrasts with the umxCP() model, in which these influences are collected on a hypothesized or latent causal variable, which is manifested in the measured phenotypes.

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() allows users to see and set mvnRelEps and mvnMaxPointsA mvnRelEps defaults to .005. For ordinal models, you might find that '0.01' works better.

Usage

```
umxIP(
  name = "IP",
  selDVs,
  dzData,
  mzData,
  sep = NULL,
  nFac = c(a = 1, c = 1, e = 1),
  data = NULL,
  zyg = "zygosity",
  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", "ordinal", "search"),
  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.
data	If provided, dzData and mzData are treated as levels of zyg to select() MZ and DZ data sets (default = NULL)
zyg	If data provided, this column is used to select rows by zygosity (Default = "zygosity")
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 IP model decomposes phenotypic variance into additive genetic (A), 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% by twins in a pair), to .25 to model dominance effects.

Matrices and Labels in IP 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")
```

Value

- OpenMx::mxModel()

References

- Kendler, K. S., Heath, A. C., Martin, N. G., & Eaves, L. J. (1987). Symptoms of anxiety and symptoms of depression. Same genes, different environments? *Archives of General Psychiatry*, **44**, 451-457. doi:10.1001/archpsyc.1987.01800170073010.
- McArdle, J. J., & Goldsmith, H. H. (1990). Alternative common factor models for multivariate biometric analyses. *Behavior Genetics*, **20**, 569-608. doi:10.1007/BF01065873.
- https://github.com/tbates/umx

See Also

- plot(), umxSummary(), umxCP()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## 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 =    umxIP(selDVs = selDVs, sep = "_T", dzData = dzData, mzData = mzData)

# WLS example: Use "marginals" method to enable all continuous data with missingness.
m3 = umxIP(selDVs = selDVs, 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(selDVs, "_T")])
mzD = na.omit(dzData[, tvars(selDVs, "_T")])
m4 = umxIP(selDVs = selDVs, sep = "_T", dzData = dzD, mzData = mzD, type = "DWLS")

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

## End(Not run)
```

---

umxJiggle *umxJiggle*

Description

umxJiggle takes values in a matrix and jiggles them

Usage

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

Arguments

matrixIn	an OpenMx::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

- OpenMx::mxMatrix()

References

https://github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

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

## End(Not run)
```

---

umxLav2RAM *Convert lavaan string to a umxRAM model*

Description

Takes a lavaan syntax string and creates the matching one or more umxRAM() models.

If data are provided, a umxRAM() model is returned.

If more than one group is found, a umxSuperModel() 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 would like them.

Usage

```
umxLav2RAM(
  model = NA,
  data = "auto",
  group = NULL,
  group.equal = NULL,
  name = NA,
  lavaanMode = c("sem", "lavaan"),
  std.lv = FALSE,
  suffix = "",
  comparison = TRUE,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  verbose = FALSE,
  optimizer = NULL,
  std = FALSE,
  printTab = TRUE
)
```

Arguments

model	A lavaan syntax string, e.g. "A~~B"
data	Data to add to model (defaults to auto, which is just sketch mode)
group	= Column to use for multi-group (default = NULL)
group.equal	= what to equate across groups. Default (NULL) means no equates. See details for what we might implement in future.
name	Model name (can also add name in # commented first line)
lavaanMode	Auto-magical path settings for cfa/sem (default) or no-defaults ("lavaan")
std.lv	= FALSE Whether to set var of latents to 1 (default FALSE). nb. Toggles fix first.
suffix	String to append to each label (useful if model will be used in a multi-group model)
comparison	Compare the new model to the old (if updating an existing model: default = TRUE)
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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
verbose	Whether to tell the user what latents and manifests were created etc. (Default = FALSE)
optimizer	optionally set the optimizer (default NULL does nothing)
std	Whether to print estimates. Defaults to FALSE ("raw"), TRUE = "std", for no parameter table use NULL.
printTab	= TRUE (more for debugging)

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 well documented. For quick reference, some common symbols in lavaan strings are:

lav	Mplus	sem	Action
A =~ B	A by B		A (Latent) is measured by B
A ~ B	A on B	A<- B	A "is regressed on" (<-) B
A ~~ B	A with B	A<->B	A covaries with B
A ~ 1	[A]		A has mean
A := B			A is defined by B (see `OpenMx::mxAlgebra()`)
A == B			A is constrained == to B (see `OpenMx::mxConstraint()`)

=~	lhs (Latent) is manifested by rhs
~	lhs "is regressed on" (<-) rhs
~~	lhs covaries with rhs
~ 1	lhs has mean
:=	lhs is defined by rhs (see `OpenMx::mxAlgebra()`)
==	lhs is constrained == to rhs (see `OpenMx::mxConstraint()`)

Naming of multiple groups

When multiple groups are found the groups are named `name_grouplevel` White space is replaced with "_" and illegal characters are replaced with "x"

note: Options for group.equal. In future, we might implement (but have not as yet):

1. c("loadings"
2. "intercepts"
3. "means"
4. "regressions"
5. "residuals"
6. "covariances"

Value

- list of `umxPath()`s

See Also

`umxRAM2Lav()`, `umxRAM()`

Other Miscellaneous Utility Functions: `install.OpenMx()`, `libs()`, `qm()`, `umx`, `umxModelNames()`, `umxRAM2Lav()`, `umxVersion()`, `umx_array_shift()`, `umx_data()`, `umx_find_object()`, `umx_lower.tri()`, `umx_msg()`, `umx_open_CRAN_page()`, `umx_pad()`, `umx_print()`, `umx_scholastica_report()`, `umx_tail()`, `umx_wide2long()`, `umx_wide4lmer()`

Examples

```
## Not run:

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

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")
umx_print(tmp$A$labels)
# |   |x1       |x3         |
# |:--|:--------|:----------|
# |x1 |x1_to_x1 |x3_loading |
# |x3 |x1_to_x3 |x3_to_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")
plot(tmp, min=c("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 = umxRAM(HS, data = umx_scale(HS.ability.data))

# Multiple groups
m1 = umxRAM(HS, data = umx_scale(HS.ability.data), group = "school")

# More examples

lav = " # Moderated mediation
gnt ~ a*cb
INT ~ b1*gnt + b2*cn + b3*cngn + c*cb

indirect := a*b1
direct := c

ab3 := a * b3
loCN := a * b1 + ab3 * -0.5
hiCN := a * b1 + ab3 * 0.5
```

```
"
tmp = umxRAM(lav)
# plot showing ability to influence layout with max min same groupings
plot(tmp, max = c("cb", "cn", "cngn"), same = "gnt", min= "INT")

# Algebra: e.g. b1^2
m1 = umxRAM("x1~b1*x2; B1_sq := b1^2", data = demoOneFactor)
m1$B1_sq$result # = 0.47

# Model with constraints and labeled parameters
lav = "
y ~ b1*x1 + b2*x2 + b3*x3
# constraints
b1 == (b2 + b3)^2
b1 > exp(b2 + b3)"

tmp = umxLav2RAM(lav)

namedModel = "  # my name
 y ~x"
m1 = umxRAM(namedModel)

# Formative factor
# lavaanify("f5 <~ z1 + z2 + z3 + z4")

## End(Not run)
```

---

umxLISREL *Make a LISREL model analogous to umxRAM*

Description

umxLISREL is a wrapper for OpenMx::mxModel() with type="LISREL". It automatically partitions manifest and latent variables into endogenous and exogenous sets, inserts data, handles raw/covariance types, adds means if raw data are used, handles ordinal variables, runs the model, and displays summaries.

Usage

```
umxLISREL(
  model = NA,
  ...,
  data = NULL,
  manifestVars = NULL,
  latentVars = NULL,
  name = NA,
  group = NULL,
  group.equal = NULL,
```

```
  suffix = "",
  comparison = TRUE,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  weight = NULL,
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  std = FALSE,
  refModels = NULL,
  remove_unused_manifests = TRUE,
  independent = NA,
  setValues = TRUE,
  optimizer = NULL,
  verbose = FALSE,
  std.lv = FALSE,
  printTab = FALSE
)
```

Arguments

model	NA, a name string, or an existing LISREL OpenMx::mxModel() to update.
...	Path statements (mxPath or umxPath), matrices, etc., to add to the model.
data	Data frame, matrix, or mxData to use.
manifestVars	Optional list of endogenous/exogenous manifest variables, or character vector of manifest variables to partition.
latentVars	Optional list of endogenous/exogenous latent variables, or character vector of latent variables to partition.
name	Optional name for the model (defaults to the model's current name or 'm1').
group	Column name in data to partition data for multi-group models.
group.equal	Not implemented.
suffix	Suffix to append to parameter labels.
comparison	Compare the model to saturated models in summary (Default = TRUE).
type	Data type: "Auto" (guesses), "FIML", "cov", "cor", "WLS", "DWLS", "ULS".
weight	Weight column name.
allContinuousMethod	Method for continuous variables when WLS is used.
autoRun	Run the model (default = TRUE).
tryHard	How to run the model: "no", "yes", "ordinal", "search".
std	Standardize output (Default = FALSE).
refModels	Reference models for summary comparison.
remove_unused_manifests	Remove manifests not used in paths (Default = TRUE).
independent	Whether the model is independent (Default = NA).

setValues	Automatically set starting values (Default = TRUE).
optimizer	Set the optimizer to use.
verbose	Print diagnostic info.
std.lv	Standardize latent variables (Default = FALSE).
printTab	Print parameter table.

Value

A LISREL OpenMx::mxModel()

References

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

See Also

umxRAM(), plot.MxLISRELModel(), xmu_standardize_LISREL()

Other Core Model Building Functions: umx, umxMatrix(), umxModify(), umxPath(), umxRAM(), umxRAM_DE(), umxSuperModel()

Examples

```
## Not run:
library(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

# 1. LISREL model with covariance data
m1 = umxLISREL("one_factor_cov", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

# 2. LISREL model with raw data (means are automatically added)
m2 = umxLISREL("one_factor_raw", data = demoOneFactor,
               umxPath("G", to = manifests),
               umxPath(var = manifests),
               umxPath(var = "G", fixedAt = 1))

# 3. Forced exogenous manifests using manifestVars list override
m3 = umxLISREL("forced_exogenous", data = demoOneFactor,
              manifestVars = list(endogenous = c("x4", "x5"), exogenous = c("x1", "x2")),
               latentVars = list(endogenous = "G", exogenous = "xi"),
               umxPath("xi", to = c("x1", "x2")),
               umxPath("G", to = c("x4", "x5")),
               umxPath("xi", to = "G"),
               umxPath(var = c("x1", "x2", "x4", "x5")),
               umxPath(var = "xi", fixedAt = 1),
               umxPath(var = "G"),
```

```
                umxPath(means = c("x1", "x2", "x4", "x5")))

## End(Not run)
```

---

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 buy default, and has the name parameter first in order.

Usage

```
umxMatrix(
  name = NA,
  type = "Full",
  nrow = NA,
  ncol = NA,
  free = FALSE,
  values = NA,
  labels = TRUE,
  lbound = NA,
  ubound = NA,
  byrow = getOption("mxByrow"),
  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
free	Whether cells are free (Default FALSE)
values	The values of the matrix (Default NA)
labels	Either whether to label the matrix (default TRUE), OR a vector of labels to apply.

lbound	Lower bounds on cells (Defaults to NA)
ubound	Upper bounds on cells (Defaults to NA)
byrow	Whether to fill the matrix down columns or across rows first (Default = getOption('mxByrow')
baseName	Set to override the default (which is to use the matrix name as the prefix).
dimnames	NA
condenseSlots	Whether to save memory by NULLing out unused matrix elements, like labels, ubound etc. Default = getOption('mxCondenseMatrixSlots')
...	Additional parameters (!! not currently supported by umxMatrix)
joinKey	See mxMatrix documentation: Defaults to as.character(NA)
joinModel	See mxMatrix documentation: Defaults to as.character(NA)
jiggle	= NA passed to xmuLabel to jiggle start values (default does nothing)

Value

- OpenMx::mxMatrix()

References

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

See Also

- xmu_simplex_corner(), OpenMx::mxMatrix(), xmuLabel(), umxRAM()

Other Core Model Building Functions: umx, umxLISREL(), umxModify(), umxPath(), umxRAM(), umxRAM_DE(), umxSuperModel()

Examples

```
## Not run:
# ================================================================================
# = 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 =
# ============================================================
umxMatrix("test", "Full", 2, 2, baseName = "bob")$labels # bob_r1c1


# ==========================================
# = 3. User-provided labels are left as-is =
# ==========================================
umxMatrix("foo", "Lower", nrow=2, ncol=2, labels= c(NA, "beta1", NA))
```

```
#      [,1]    [,2]
# [1,] NA      NA
# [2,] "beta1" NA


## End(Not run)
```

---

umxMatrixFree *Sets labeled matrix cells to free*

Description

In simulation studies, it is often necessary to rewrite the matrices while testing alternative specifications. This can become very tedious with increasing number of distinct specifications. This tool injects changes into umxMatrix so that this tasks gets more manageable. First, it sets byrow by default. Second, it infers the number of rows automatically. The user needs only passing ncol. Finally and most importantly this function disables auto-labeling, and whenever a label is set, that cell position will be freed. It is required to pass a matrix of labels, as well as a label name.

Usage

```
umxMatrixFree(
  name = name,
  nrow = NULL,
  ncol = NA,
  free = FALSE,
  values = NA,
  labels = labels,
  byrow = TRUE,
  ...
)
```

Arguments

name	The name of the matrix: Must be set
nrow	Number of rows in the matrix (Optional)
ncol	Number of columns in the matrix (Required)
free	Whether cells are free (Default FALSE)
values	The values of the matrix (Default NA)
labels	The labels of the matrix (Default NA)
byrow	Default for byrow (TRUE)
...	Accepts all other arguments from umxMatrix()

Value

- OpenMx::mxMatrix()

See Also

- umxMatrix()

Examples

```
## Not run:

umxMatrixFree('E', type='Symm',  ncol = 3,
  labels =c("eb2",NA,NA,
            NA,"es2",NA,
           NA,NA,NA),
  values=c(.2,0,0,
           0,.2,0,
           0,0,0))

 # Will return a umxMatrix free at the eb2 and es2 positions.

## End(Not run)
```

---

umxMeasurementInvariance

Test measurement invariance of a confirmatory factor analysis model

---

Description

umxMeasurementInvariance runs a sequence of multi-group CFA models to test for configural, weak (metric), strong (scalar), and strict invariance across groups.

Usage

```
umxMeasurementInvariance(
  model,
  data = NULL,
  group = NULL,
  type = "all",
  compare = c("both", "sequential", "configural"),
  silent = FALSE,
  ...
)
```

Arguments

model	A lavaan syntax string defining the CFA model.
data	A data frame containing the data.
group	Character string, the name of the grouping variable.
type	Character vector, which tests to run. Default is "all", which runs configural, weak, strong, and strict models.

compare	Character string, how to compare the models: "both", "sequential", or "configural". Default is "both".
silent	Boolean, whether to suppress output and messages. Default is FALSE.
...	Additional arguments passed to umxRAM().

Value

A model comparison table (data frame). If compare is "both" (default) or "sequential", the sequential comparison table is returned, and the configural/baseline comparison table is attached as the configural attribute. If compare is "configural", the baseline comparison table is returned.

See Also

- umxRAM(), umxCompare()

Examples

```
## Not run:
data("HS.ability.data", package = "OpenMx")
df = umx_scale(HS.ability.data[, c("visual", "cubes", "flags", "school")])
model = "spatial =~ visual + cubes + flags"
umxMeasurementInvariance(model, data = df, group = "school")

## End(Not run)
```

---

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

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

Arguments

model	An OpenMx::mxModel() for which to report modification indices
matrices	which matrices to test. The default (NA) will test A & S for RAM models
full	If FALSE (default), only the parameter under test can move. If TRUE, all free parameters re-adjust (much slower).
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. Default full = FALSE only moves the parameter under test (fast screen). Set full = TRUE to allow all free parameters to re-adjust (slower; closer to a full re-fit for each candidate).
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 OpenMx::mxMI() You are almost certainly doing better science when testing competing models rather than modifying a model to fit.

References

https://github.com/tbates/umx

See Also

- OpenMx::mxMI()

Other Model Summary and Comparison: umx, umxCompare(), umxEquate(), umxReduce(), umxSetParameters(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
umxMI(m1)
umxMI(m1, full = TRUE)  # slower: allow all parameters to re-adjust

## End(Not run)
```

umxModel *Catches users typing umxModel instead of umxRAM.*

Description

Catches a common typo, moving from mxModel to umx.

Usage

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

Arguments

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

Value

None

References

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

See Also

- umxRAM(), OpenMx::mxModel()

Other xmu internal not for end user: mx_make(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(

xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
umxModel()

## End(Not run)
```

---

umxModelNames *Return names of models found within a model*

Description

umxModelNames returns the names of each model contained in the model provided to it (optionally excluding the out model itself).

Usage

```
umxModelNames(model, includeOuterModelName = FALSE)
```

Arguments

model	an OpenMx::mxModel() to search for model names.
includeOuterModelName	FALSE

Value

- All models names

See Also

- OpenMx::mxRename(), umxSuperModel()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
## Not run:
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
selDVs = c("gff", "fc", "qol")
m1 = umxCP(selDVs= selDVs, nFac= 1, dzData= dzData, mzData= mzData, sep= "_T", autoRun= TRUE)
m2 = mxRename(m1, "model2")
umxModelNames(m1) # "top" "MZ" "DZ"
umxModelNames(m2) # "top" "MZ" "DZ"

super = umxSuperModel("myModel", m1, m2, autoRun = TRUE)
umxModelNames(super)

plot(super$CP1fac)

## End(Not run)
```

---

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

Description

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

Usage

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

Arguments

lastFit	The OpenMx::mxModel() you wish to update and run.
update	What to update before re-running. Can be a list of labels, a regular expression (set regex = TRUE) or an object such as mxCI etc.
regex	Whether or not update is a regular expression (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)
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
comparison	Whether to run umxCompare() on the new and old models.
autoRun	Whether to run the model (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
master	If you set master, then the update labels will be equated to these (i.e. replaced by them).
intervals	Whether to run confidence intervals (see OpenMx::mxRun())
verbose	How much feedback to give

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:

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

Regular expressions are a powerful feature: they let you drop collections of paths by matching patterns for instance, this would match labels containing either "Cs" or "Cr":

```
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 = umxSetParameters(fit1, labels = c("Cs", "Cr"), values = 0, free = FALSE, name = "newName")
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

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

Other Core Model Building Functions: umx, umxLISREL(), umxMatrix(), umxPath(), umxRAM(), umxRAM_DE(), umxSuperModel()

Examples

```
## Not run:
require(umx)
# First we'll just build a 1-factor model
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", 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)
m2 = umxModify(m1, update = "G_to_x1", free= TRUE, name = "addback_X1", comparison = TRUE)
# 3. Fix a value at a non-zero value
m3 = umxModify(m1, update = "G_to_x1", value = .35, name = "fix_G_x1_at_35", comp = TRUE)
# You can add objects to models. For instance this would add a path (overwriting the existing one)
# (thanks Johannes!)
m3 = umxModify(m1, umxPath("G", with = "x1"), name= "addedPath")

# Use regular expression to drop multiple paths: e.g. G to x3, x4, x5
m3 = umxModify(m1, regex = "^G_to_x[3-5]", name = "tried_hard", comp = TRUE, tryHard="yes")

# 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"
m2 = umxModify(m1, update = "G_to_x1", newlabels= newLabel, name = newModelName, comparison = TRUE)
# Change labels in 2 places
labsToUpdate = c("G_to_x1", "G_to_x2")
newLabel = "G_to_1_or_2"
m2 = umxModify(m1, update = labsToUpdate, newlabels= newLabel, name = "equated", comparison = TRUE)
```

```
# Advanced: Regular expressions

# 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)

## End(Not run) # end dontrun
```

---

umxMRDoC *Extends Mendelian randomization with the twin design to test evidence of causality*

Description

Testing causal claims is often difficult due to an inability to conduct experimental randomization of traits and situations to people. When twins are available, even when measured on a single occasion, the pattern of cross-twin cross-trait correlations can (given distinguishable modes of inheritance for the two traits) falsify causal hypotheses.

umxMRDoC implements a 2-group model to form latent variables for each of two traits, and allows testing whether trait 1 causes trait 2, vice-versa, or even reciprocal causation. This is robust to several types of confounding due to the instrumental variable approach included in the model.

This function applies both the MRDoC model and the MRDoC2 model depending on how many PRSs are passed as arguments.

Usage

```
umxMRDoC(
  data = NULL,
  pheno,
  prss = NULL,
  mzData = NULL,
  dzData = NULL,
  sibsData = NULL,
  zygosity = "zygosity",
  sep = "_T",
  summary = !umx_set_silent(silent = TRUE),
  name = NULL,
  autoRun = getOption("umx_auto_run"),
  sibs = FALSE,
  type = "FIML",
  tryHard = c("no", "yes", "ordinal", "search"),
  optimizer = NULL,
  covar = NULL,
  batteries = c("scale"),
```

```
  method = "Mehta",
  verbose = FALSE
)
```

Arguments

data	= NULL If building the MZ and DZ datasets internally from a complete data set.
pheno	Phenotypes of interest, order matters ("exposure", "outcome")
prss	Polygenic score(s). If a single one is passed MRDoC is run, MRDoC2 otherwise.
mzData	The MZ dataframe
dzData	The DZ dataframe
sibsData	The unrelated sibs dataframe, requires "sibs" as an extra zygosity level.
zygosity	= "zygosity" (for the data= method of using this function).
sep	The separator in twin variable names, default = "_T", e.g. "dep_T1".
summary	Optionally show a summary.
name	The name of the model (defaults to either "MRDoC" or "MRDoC2).
autoRun	Whether to run the model (default), or just to create it and return without running.
sibs	NEEDS DOCUMENTING (FALSE)
type	Basic switch for estimation type. WLS tends to work really well in MRDoC, it saves you time.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search".
optimizer	Optionally set the optimizer (default NULL does nothing).
covar	= Covariates that will be regressed on X and Y phenotypes.
batteries	Batteries included, currently scales continuous variables by default, "dump" will return data for inspection. Use NULL for disabling all.
method	Method for handling ordinal variables, defaults to Mehta.
verbose	Outputs the pre-processing steps/warnings.

Value

- `OpenMx::mxModel()` of subclass MxModelMRDoC

References

- Minica CC, Dolan CV, Boomsma DI, et al. (2018) Extending Causality Tests with Genetic Instruments: An Integration of Mendelian Randomization with the Classical Twin Design. Behavior Genetics 48(4): 337-349. doi:10.1007/s1051901899044
- McGue, M., Osler, M., & Christensen, K. (2010). Causal Inference and Observational Research: The Utility of Twins. *Perspectives on Psychological Science*, **5**, 546-556. doi:10.1177/1745691610383511
- Castro-de-Araujo LFS, Singh M, Zhou Y, et al. (2022) MR-DoC2: Bidirectional Causal Modeling with Instrumental Variables and Data from Relatives. Behavior Genetics. doi:10.1007/s1051902210122x

See Also

- umxDoC()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:

# ================
# = 1. Load Data =
# ================
data(docData)
mzData  = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData  = subset(docData, zygosity %in% c("DZFF", "DZMM"))

# ============================
# = 2. Make a MRDoC2 model   =
# ============================
out = umxMRDoC(mzData = mzData, dzData = dzData,
pheno = c("varA1", "varA2"), prss = c("varB1", "varB2") )

## End(Not run)
```

---

umxParameters *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. umxParameters() 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

```
umxParameters(
  x,
  thresh = c("all", "above", "below", ">", "<", "NS", "sig"),
  b = NULL,
  pattern = ".*",
  std = FALSE,
```

```
  digits = 2
)

parameters(
  x,
  thresh = c("all", "above", "below", ">", "<", "NS", "sig"),
  b = NULL,
  pattern = ".*",
  std = FALSE,
  digits = 2
)
```

Arguments

x	an OpenMx::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://tbates.github.io

See Also

- umxGetParameters(), umxSummary(), namez()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umx_aggregate(), umx_time()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
```

```
manifests = names(demoOneFactor)
m1 = umxRAM("OneFactor", data = demoOneFactor,
 umxPath(from = "G", to = manifests), # factor loadings
 umxPath(v.m. = manifests),           # residual variance
 umxPath(v1m0 = "G")                  # standardized latent
)
# Parameters with values below .1
umxParameters(m1, "below", .1)
# Parameters with values above .5
umxParameters(m1, "above", .5)
# Parameters with values below .1 and containing "_to_" in their label
umxParameters(m1, "below", .1, "_to_")

## End(Not run)
```

---

umxParan *A wrapper to make paran easier to use. Just automates applying* `complete.cases()`

Description

A wrapper to make paran easier to use. Just automates applying `complete.cases()`

Usage

```
umxParan(df, cols = NA, graph = TRUE, mapStrings = NULL, n = NULL)
```

Arguments

df	The df (just the relevant columns) or a covariance/correlation matrix.
cols	(optional) list of columns (default = use all)
graph	Whether to graph.
mapStrings	optional mapping if cols are strings
n	optional number of observations (required if passing a covariance/correlation matrix)

Value

- nothing

See Also

Other Miscellaneous Stats Functions: `FishersMethod()`, `SE_from_p()`, `geometric_mean()`, `harmonic_mean()`, `oddsratio()`, `reliability()`, `umx`, `umxCov2cor()`, `umxHetCor()`, `umxWeightedAIC()`, `umx_apply()`, `umx_cor()`, `umx_means()`, `umx_r_test()`, `umx_round()`, `umx_scale()`, `umx_var()`

Examples

```
library(psych)
library(psychTools)
data(bfi)
umxParan(bfi[, paste0("A", 1:5)])
umxParan(bfi, cols= paste0("A", 1:5))
# umxParan(bfi, paste0("AB", 1))
```

---

umxPath *Easier (and powerful) specification of paths in SEM.*

Description

umxPath is used to easily and compactly specify paths in models. In addition to `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`. To give a couple of the most common, time-saving examples:

- `umxPath("A", with = "B", fixedAt = 1)`
- `umxPath(var = c("A", "B"), fixedAt = 1)`
- `umxPath(v.m. = manifests)`# free variance and mean estimation
- `umxPath(v1m0 = latents)` # fixed at var 1 mean 0
- `umxPath(means = manifests)`
- `umxPath(fromEach = c('A',"B","C"), to = c("y1","y2"))`
- `umxPath(unique.bivariate = c('A',"B","C"))`
- `umxPath("A", to = c("B","C","D"), firstAt = 1)`

Usage

```
umxPath(
  from = NULL,
  to = NULL,
  with = NULL,
  var = NULL,
  cov = NULL,
  means = NULL,
  v1m0 = NULL,
  v.m. = NULL,
  v0m0 = NULL,
  v.m0 = NULL,
  v0m. = 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.
v0m.	variance fixed at 0, mean free.
fixedAt	Equivalent to setting "free = FALSE, values = fixedAt"
freeAt	Equivalent to setting "free = TRUE, values = freeAt"
firstAt	First path is fixed at this value (free is ignored: warning if other than 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.

Cholesky	Treat **Cholesky** variables as latent and **to** as measured, and connect as in an ACE model.
defn	Implements a definition variable as a latent with zero variance & mean and labeled 'data.defVar'
connect	as in mxPath - nb: from and to must also be set.
arrows	as in mxPath - nb: from and to must also be set.
free	whether the value is free to be optimised
values	default value list
labels	labels for each path
lbound	lower bounds for each path value
ubound	upper bounds for each path value
hasMeans	Used in 'forms' case to know whether the data have means or not.
...	Not used. Caught to provide a polite message if a reversed form like m.v. is passed instead of v.m..

Details

umxPath introduces the following new words to your path-defining vocabulary: with, var, cov, means, v1m0, v0m0, v.m0, v.m, fixedAt, freeAt, firstAt, unique.bivariate, unique.pairs, fromEach, Cholesky, defn, forms.

with creates covariances (2-headed paths): umxPath(A, with = B)

Specify a variance for A with umxPath(var = "A").

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)

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 to create paths 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"))

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

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

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

To create Cholesky-pattern connections:
umxPath(Cholesky = c("A1", "A2"), to c("var1", "var2"))

Value

- 1 or more OpenMx::mxPath()s

References

https://tbates.github.io

See Also

- OpenMx::mxPath()

Other Core Model Building Functions: umx, umxLISREL(), umxMatrix(), umxModify(), umxRAM(), umxRAM_DE(), umxSuperModel()

Examples

```
# ========================================
# = 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 = c("B","C"), 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(means = c("A","B")) # Create a means model for A: from = "one", to = "A"
umxPath(v1m0 = "A") # Give "A" variance and a mean, fixed at 1 and 0 respectively
umxPath(v.m. = "A") # Give "A" variance and a mean, leaving both free.
umxPath(v0m0 = "W", label = c(NA, "data.W"))
umxPath("A", with = "B") # using with: same as "to = B, arrows = 2"
umxPath("A", with = "B", fixedAt = .5) # 2-head path fixed at .5
umxPath("A", with = c("B", "C"), firstAt = 1) # first covariance fixed at 1
umxPath(cov = c("A", "B"))  # Covariance A <-> B
umxPath(defn = "mpg") # create latent called def_mpg, with var = 1 and label = "data.mpg"
umxPath(fromEach = c('a','b'), to = c('c','d')) # a->c, a<->d, b<->c, b<->d
umxPath(unique.bivariate = c('a','b','c')) # bivariate paths a<->b, a<->c, b<->c etc.
umxPath(unique.pairs = letters[1:3]) # all distinct pairs: a<->a, a<->b, a<->c, b<->b, etc.
umxPath(Cholesky = c("A1","A2"), to = c("m1", "m2")) # Cholesky

## Not run:
# A worked example
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type= "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)
umxSummary(m1, std = TRUE)
require(umx)


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

```
# ============================================================================
# = 3-factor Cholesky (A component of a 5-variable 3-factor ACE model) =
# ============================================================================
latents   = paste0("A", 1:3)
manifests = names(demoOneFactor)
m1 = umxRAM("Chol", data = demoOneFactor, type = "cov",
 umxPath(Cholesky = latents, to = manifests),
 umxPath(var = manifests),
 umxPath(var = latents, fixedAt = 1)
)
plot(m1, splines= FALSE)

# ============================================================================
# = Definition variable example. for a RAM model                           =
# = def vars are instantiated as dummy latents with data on the "mean" =
# ============================================================================
library(umx); libs("MASS") # for mvrnorm()
# 1. Create Data
N = 500 # size of each group
Sigma  = matrix(c(1,.5,.5,1),2,2) # cov (.5)
group1 = MASS::mvrnorm(N, c(1,2), Sigma)
group2 = MASS::mvrnorm(N, c(0,0), Sigma)
# rbind groups and name cols "x" and "y"
xy = rbind(group1, group2)
dimnames(xy)[2]= list(c("x", "y"))

# Create a definition variable for group status
groupID = rep(c(1,0), each = N)
df = data.frame(xy, groupID = groupID)

# Make the model with a definition variable on means
m1 = umxRAM("Def Means", data = df,
 umxPath(v.m. = c("x","y")),
 umxPath("x", with = "y"),
 # create a unit latent called "def_groupID" with data "data.groupID"
 umxPath(defn = "groupID"),
 # Add it to the x and y means
 umxPath("def_groupID", to = c("x", "y"))
)
plot(m1)


## End(Not run)
```

---

umxPlot *Quickly plot y ~ x with a regression line and R^2, and nice labels.*

Description

Want a figure for your paper or presentation but not the work of combining ggplot2::ggplot(), ggplot2::geom_smooth() and method options, plus ggplot2::geom_point(). Organizing ggplot2::labs()

and its x, y, and title components. Adding your preferred theme like ggplot2::theme_gray(), plus recalling for cowplot::draw_label(), and/or ggplot2::annotate() to draw math-enabled labels on the plot, as well as the required bquote(), and extracting the relevant fit statistics from lm() and the subsidiary tasks of reformulate() programmatic variables?

umxPlot just takes y ~ x (or "x" and "y" as strings), and gives you a nicely labelled plot, with a fitted line, the R^2 so readers can see how well this fitted. It knows how to put Greek symbols like $beta$ into axes.

Usage

```
umxPlot(
  x,
  y = NULL,
  data,
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  r2x = NA,
  r2y = NA,
  geom_point = TRUE,
  method = c("lm", "auto", "loess", "glm", "gam"),
  family = c("gaussian", "binomial", "Gamma", "inverse", "poisson", "quasi",
    "quasibinomial", "quasipoisson")
)
```

Arguments

x	formula or (alternatively) x as string
y	variable as string.
data	The data for the graph.
xlab	X-axis label (default y).
ylab	Y-axis label (default y).
title	Graph title. Default = paste0(y, " as a function of ", x)
r2x	x location for the fit summary (default 1).
r2y	y location for the fit summary (default 2).
geom_point	show points? (TRUE)
method	Method for fitting curve (default = lm)
family	for glm default = "gaussian"

Value

- plot you can edit.

See Also

- ggplot2::ggplot()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
data(mtcars)
umxPlot(mpg ~ wt, data = mtcars, r2x = 2, r2y = 10)
umxPlot(x = "wt", y = "mpg", mtcars, r2x = 2, r2y = 10)
```

---

umxPlotACE *Make a graphical display of an ACE model*

Description

plot method for umxACE() models. Make a graphical display of an ACE model

Usage

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

Arguments

x	OpenMx::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)
showFixed	Whether too draw fixed parameters.
...	Additional (optional) parameters

Value

- optionally return the dot code

References

https://github.com/tbates/umx

See Also

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

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
require(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACE("plotACE example", selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")
plot(m1, std = FALSE) # don't standardize

## End(Not run)
```

---

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	OpenMx::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://tbates.github.io

See Also

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

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
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, ]
m1 = umxACEcov(selDVs= selDVs, selCovs= selCovs, dzData= dzData, mzData= mzData, sep= "")
plot(m1)
plot(m1, std = FALSE) # don't standardize
```

```
## End(Not run)
```

---

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

```
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

References

https://github.com/tbates/umx

See Also

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
require(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACEv(selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")
umxSummary(m1)
umxPlotACEv(m1, std = FALSE) # Don't standardize
plot(m1, std = FALSE) # don't standardize

## End(Not run)
```

---

umxPlotACE_DE *Plot a double-entry censored twin model (umxACE_DE)*

Description

umxPlotACE_DE renders a GraphViz diagram for a umxACE_DE() model. It automatically filters out the redundant continuous partner (_cont) of each double-entry pair, plotting only the censored trait (_cens) and any unpaired continuous variables.

Usage

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

## S3 method for class 'MxModelACE_DE'
plot(
  x = NA,
  file = "name",
  digits = 2,
  means = FALSE,
```

```
  std = TRUE,
  strip_zero = TRUE,
  showFixed = FALSE,
  ...
)
```

Arguments

x	An OpenMx::mxModel() of class MxModelACE_DE 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 (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).
showFixed	Whether to draw fixed parameters (default = FALSE).
...	Additional (optional) parameters passed to xmu_dot_maker().

Value

- optionally returns the dot code string.

References

https://github.com/tbates/umx

See Also

- plot(), umxSummary(), umxACE_DE()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

---

umxPlotACE_GLM *Plot a GLM ACE twin model*

Description

GraphViz of the Cholesky on eta: same compact a/c/e-to-trait diagram as umxPlotACE(), not the full twin RAM (no T1/T2 duplicate latents). Printed e is chol(ee'+D) when leftover is on eta (Bernoulli), matching umxSummaryACE_GLM().

Usage

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

Arguments

x	a model from umxACE_GLM()
file	NA returns the dot string; "name" writes a file named after the model
digits	decimals on path labels (default 2)
means	show intercepts from one (default FALSE)
std	eta-scale row standardization (default TRUE), leftover included in E
strip_zero	strip leading 0. from labels (default TRUE)
showFixed	draw fixed non-zero paths (default TRUE so binomial e at 1 shows)
...	unused

Value

dot string (invisibly) when file is NA

See Also

umxACE_GLM(), umxSummaryACE_GLM(), umxPlotACE()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

---

umxPlotCP *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

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

Arguments

x	The umxCP() Common Pathway to display graphically
means	Whether to show means paths (defaults to FALSE)
std	Whether to standardize the model (defaults to TRUE)
digits	How many decimals to include in path loadings (defaults to 2)
showFixed	Whether to graph paths that are fixed but != 0 (default = TRUE)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
format	= c("current", "graphviz", "DiagrammeR")
SEstyle	use mxSE "b (se)" instead of mxCI "b [lower, upper]" (Default FALSE)
strip_zero	Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
showCIs	Whether to show confidence intervals (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: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## 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 = "_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)
```

---

umxPlotDoC *Plot a Direction of Causation Model.*

Description

Summarize a fitted model returned by umxDoC(). Can control digits, report comparison model fits, optionally show the *Rg* (genetic and environmental correlations), and show confidence intervals. *note*: std is not implemented as yet. See documentation for other umx models here: umxSummary().

Usage

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

Arguments

x	a umxDoC() model to display graphically
means	Whether to show means paths (defaults to FALSE)
std	Whether to standardize the model (defaults to TRUE)
digits	How many decimals to include in path loadings (defaults to 2)
showFixed	Whether to graph paths that are fixed but != 0 (default = TRUE)
file	The name of the dot file to write: NA = none; "name" = use the name of the model
format	= c("current", "graphviz", "DiagrammeR")
SEstyle	report "b (se)" instead of "b [lower, upper]" when CIs are found (Default FALSE)
strip_zero	Whether to strip the leading "0" and decimal point from parameter estimates (default = TRUE)
...	Other parameters to control model summary.

Value

- Optionally return the dot code

References

https://tbates.github.io

See Also

- umxDoC(), umxSummary.MxModelDoC(), umxModify(), umxDiscTwin(), umxDiffMZ(), umxMR()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
# ================
# = 1. Load Data =
# ================
data(docData)
mzData = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData = subset(docData, zygosity %in% c("DZFF", "DZMM"))

# ======================================
# = 2. Define manifests for var 1 and 2 =
# ======================================
var1 = paste0("varA", 1:3)
var2 = paste0("varB", 1:3)

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

```
# = 2. Make the non-causal (Cholesky) and causal models =
# ======================================================
Chol= umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= FALSE)
DoC = umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= TRUE)

# ================================================
# = Make the directional models by modifying DoC =
# ================================================
a2b = umxModify(DoC, "a2b", free = TRUE, name = "A2B")
plot(a2b)


## End(Not run)
```

---

umxPlotFun *Easily plot functions in R*

Description

A wrapper for ggplot2::stat_function() that handles single or multiple functions.

Usage

```
umxPlotFun(
  fun = c("sin(x)", "cos(x)"),
  min = -1,
  max = 5,
  xlab = NULL,
  ylab = NULL,
  title = NULL,
  logY = c("no", "log", "log10"),
  logX = c("no", "log", "log10"),
  p = NULL,
  x = NULL,
  ...
)
```

Arguments

fun	Function(s) to plot. Takes strings like c("3 + sin(x)^2", "cos(x)"), equations like "P = 100 - Q", or function objects.
min	x-range min.
max	x-range max.
xlab	Optional x axis label.
ylab	Optional y axis label.
title	Optional title for the plot.

logY	Set to "log" or "log10" to transform y coordinate.
logX	Set to "log" or "log10" to transform x coordinate.
p	Optional plot onto which to draw the function(s).
x	Optional name of the horizontal-axis variable when a string has several right-hand-side symbols and no x, e.g. x = "Q". Not needed when values leave exactly one symbol free.
...	Named single-number values for remaining right-hand-side symbols, e.g. c = 5. Names matching no plotted symbol warn.

Details

Easily plot functions-like sin or x^2-using ggplot. Accepts bare functions, strings, or lists of strings/functions. Automatically generates a legend when multiple functions are provided.

Strings may use arbitrary variable names: the sole right-hand-side symbol is plotted on the horizontal axis, e.g. "100 - Q" plots against Q. An equation left-hand side names the vertical axis, e.g. "P = 100 - Q" labels the y axis P. When all but one right-hand-side symbol has a value, the free symbol takes the horizontal axis automatically, e.g. umxPlotFun("alice = bob^2+c", c = 5) plots against bob. Otherwise name the horizontal variable with x, e.g. umxPlotFun("P = a*Q + b", x = "Q"); remaining symbols must have values where the plot is drawn. Bind leftover symbols to values as arguments, e.g. umxPlotFun("alice = bob^2+c", x = "bob", c = 5).

Value

A ggplot graph object

See Also

ggplot2::stat_function()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## Not run:
# Plotting multiple strings
p = umxPlotFun(c("sin(x)", "cos(x)"), max = 2*pi)

# Providing a named list to control legend labels
umxPlotFun(list(Sine = sin, Cosine = cos), max = 2*pi)

# Arbitrary symbols: Q plots on x, P labels y
umxPlotFun("P = 100 - Q")

# Several right-hand-side symbols: name the x variable
a = 2; b = 10
umxPlotFun("P = a*Q + b", x = "Q")
```

```
# Binding leftover symbols to values
umxPlotFun("alice = bob^2+c", x = "bob", c = 5)

# All-but-one bound: the free symbol takes the axis automatically
umxPlotFun("alice = bob^2+c", c = 5)

## End(Not run)
```

`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

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

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")
`gg`	Use ggplot2 (default = TRUE)
`moderatorValues`	If you want to pass in your own list of moderator values instead of the real ones in the data (Default = NULL)
`...`	Optional additional parameters

Details

note: If gg=TRUE, the plots are drawn in ggplot, and also returned as a `list(raw, std)` so you can edit them.

Value

None

References

https://tbates.github.io

See Also

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

Other Plotting functions: `ggAddR()`, `plot.MxLISRELModel()`, `plot.MxModel()`, `plot.MxModelTwinMaker()`, `umx`, `umxPlot()`, `umxPlotACE()`, `umxPlotACE_DE()`, `umxPlotACE_GLM()`, `umxPlotACEcov()`, `umxPlotACEv()`, `umxPlotCP()`, `umxPlotDoC()`, `umxPlotFun()`, `umxPlotGxEbiv()`, `umxPlotIP()`, `umxPlotPredict()`, `umxPlotSexLim()`, `umxPlotSimplex()`

Examples

```
## Not run:
require(umx)
data(twinData)
twinData$age1 = twinData$age2 = twinData$age
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1= umxGxE(selDVs= "bmi", selDefs= "age", dzData= dzData, mzData= mzData, sep="", tryHard="yes")
plot(m1)
# Directly call umxPlotGxE
umxPlotGxE(m1, xlab = "Age", separateGraphs = TRUE, gg = FALSE)
umxPlotGxE(m1, moderatorValues=18:67)

## End(Not run)
```

---

umxPlotGxEbiv *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

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

Arguments

x	A fitted umxGxEbiv() 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

None

References

https://tbates.github.io

See Also

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

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
require(umx)
data(twinData)
## Not run:
selDVs  = "wt"; selDefs = "ht"
df = umx_scale_wide_twin_data(twinData, varsToScale = c("ht", "wt"), suffix = "")
mzData  = subset(df, zygosity %in%  c("MZFF", "MZMM"))
dzData  = subset(df, zygosity %in%  c("DZFF", "DZMM", "DZOS"))

m1 = umxGxEbiv(selDVs = selDVs, selDefs = selDefs,
 dzData = dzData, mzData = mzData, sep = "", dropMissingDef = TRUE)
# Plot Moderation
plot(m1)
umxPlotGxEbiv(m1, xlab = "wt", separateGraphs = TRUE, location = "topleft")

## End(Not run)
```

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,
  showFixed = TRUE,
  format = c("current", "graphviz", "DiagrammeR"),
  SEstyle = FALSE,
  strip_zero = TRUE,
  showCIs = 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)
std	Whether to standardize the model (defaults to TRUE)
showFixed	Whether to graph paths that are fixed but != 0 (default = 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)
showCIs	Whether to show confidence intervals (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.
- umxIP()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotPredict(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
## 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 =    umxIP(selDVs = selDVs, sep = "_T", dzData = dzData, mzData = mzData)
plot(model)
umxPlotIP(model, file = NA)

## End(Not run)
```

---

umxPlotPredict umxPlotPredict *Take a model and plot the y against predicted(y)*

Description

umxPlotPredict is a function which

Usage

```
umxPlotPredict(
  model,
  xlab = "Predicted Y",
  ylab = "Observed Y",
  r2x = NULL,
  r2y = NULL,
  font_size = 13,
  rsq = FALSE,
  font = "Times"
)
```

Arguments

model	lm or other model that understands predict()
xlab	X-axis label (default x).
ylab	Y-axis label (default y).
r2x	x location for the fit summary (default NULL guesses upper-left from the data and prints the guess).
r2y	y location for the fit summary (default NULL guesses upper-left from the data and prints the guess).
font_size	Default 13
rsq	R^2 or r (defaults to FALSE = r)
font	Default "Times"

Value

- plot you can edit.

See Also

- ggplot2::ggplot()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotSexLim(), umxPlotSimplex()

Examples

```
data(mtcars)
tmp = lm(mpg ~ wt, data = mtcars)
umxPlotPredict(tmp) # R^2 location guessed (upper-left) and printed
umxPlotPredict(tmp, r2x = 25, r2y = 15)
```

---

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

Description

Will plot a graphical figure for a sex limitation model. Options include digits (rounding), showing means or not, and which output format is desired.

Usage

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

Arguments

x	OpenMx::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

Value

- Optionally return the dot code

References

https://tbates.github.io

See Also

- umxSexLim(), umxSummarySexLim()

Other Plotting functions: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSimplex()

Examples

```
## 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
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)

# ============================
# = 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
)
umxPlotSexLim(m1)
plot(m1) # no need to remember a special name: plot works fine!

## End(Not run)
```

---

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: ggAddR(), plot.MxLISRELModel(), plot.MxModel(), plot.MxModelTwinMaker(), umx, umxPlot(), umxPlotACE(), umxPlotACE_DE(), umxPlotACE_GLM(), umxPlotACEcov(), umxPlotACEv(), umxPlotCP(), umxPlotDoC(), umxPlotFun(), umxPlotGxE(), umxPlotGxEbiv(), umxPlotIP(), umxPlotPredict(), umxPlotSexLim()

Examples

```
## Not run:
data(iqdat)
mzData = subset(iqdat, zygosity == "MZ")
dzData = subset(iqdat, zygosity == "DZ")
selDVs = c("IQ_age1", "IQ_age2", "IQ_age3", "IQ_age4")
m1 = umxSimplex(selDVs = selDVs, sep = "_T", dzData = dzData, mzData = mzData)
# plot(m1)

## End(Not run)
```

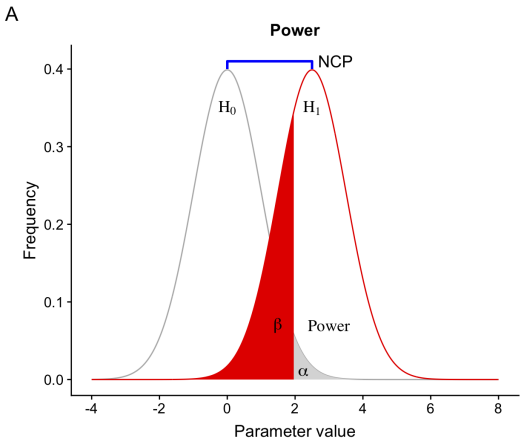
umxPower *Test power to detect specified path values in a model.*

Description

umxPower takes an input model (the model of the true data), and tests power (or determines n) to detect dropping (or changing the value) a path in this true model.

A typical target for power is 80%. Much as the accepted critical p-value is .05, this has emerged as a trade off, in this case of resources required for more powerful studies against the cost of missing a true effect. People interested in truth discourage running studies with low power: A study with 20 percent power will fail to detect real effects 80% of the time. But even with zero power, the Type-I error rate remains a nominal 5% (and with any researcher degrees of freedom, perhaps much more than that). Low powered research, then, fails to detect true effects, and generates support for random false theories about as often. This sounds silly, but empirical rates are often as low as 20% (Button, et al., 2013).

Illustration of α, β, and power (1-β):



Usage

```
umxPower(
  trueModel,
  update = NULL,
  n = NULL,
  power = NULL,
  sig.level = 0.05,
  value = 0,
  method = c("ncp", "empirical"),
  explore = FALSE,
  digits = 2,
  plot = TRUE,
  silent = TRUE
)
```

Arguments

trueModel	The model with the parameters at values you expect in the population.
update	The parameter(s) to drop
n	How many subjects? (Default = NULL)
power	Default = NULL (conventional level = .8)
sig.level	Default = .05
value	Value of dropped parameter (default = 0)
method	"ncp" (default) or "empirical"
explore	Whether to tabulate the range of n or effect size (if n specified). Default = FALSE.
digits	Rounding precision for reporting result.
plot	whether to plot the power.
silent	Suppress model runs printouts to console (TRUE)

Value

power table

References

- Miles, J. (2003). A framework for power analysis using a structural equation modelling procedure. *BMC Medical Research Methodology*, **3**, 27. doi:10.1186/14712288327
- Superpower package

See Also

- power.ACE.test(), umxRAM()

Other Teaching and Testing functions: tmx_show.MxModel(), umxDiagnose()

Examples

```
## Not run:
# ================================================
# = Power to detect correlation of .3 in 200 people =
# ================================================

# 1 Make some data
tmp = umx_make_raw_from_cov(qm(1, .3| .3, 1), n=2000, varNames= c("X", "Y"), empirical= TRUE)

# 2. Make model of true XY correlation of .3
m1 = umxRAM("corXY", data = tmp,
   umxPath("X", with = "Y"),
   umxPath(var = c("X", "Y"))
)
# 3. Test power to detect .3 versus 0, with n= 90 subjects
umxPower(m1, "X_with_Y", n= 90)
```

```
# ####################
# # Estimating power #
# ####################
#
#    method = ncp
#         n = 90
#     power = 0.83
# sig.level = 0.05
# statistic = LRT

# ================================================
# = Tabulate Power across a range of values of  n =
# ================================================
umxPower(m1, "X_with_Y", explore = TRUE)

# ====================================
# = Examples with method = empirical  =
# ====================================

# Power to detect r = .3 given n=90
umxPower(m1, "X_with_Y", n = 90, method = "empirical")
# power is .823
# Test using pwr library r.test doing the same thing.
pwr::pwr.r.test(r = .3, n = 90)
#           n = 90
#           r = 0.3
#   sig.level = 0.05
#       power = 0.827
# alternative = two.sided

# Power search for detectable effect size, given n = 90
umxPower(m1, "X_with_Y", explore = TRUE)
umxPower(m1, "X_with_Y", n= 90, explore = TRUE)
umxPower(m1, "X_with_Y", n= 90, method = "empirical", explore = TRUE)


data(twinData) # ?twinData from Australian twins.
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 10
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACE(selDVs = "ht", selCovs = "age", sep = "", dzData = dzData, mzData = mzData)

# drop more than 1 path
umxPower(m1, update = c("c_r1c1", "age_b_Var1"), method = 'ncp', n=90, explore = TRUE)

# Specify only 1 parameter (not 'age_b_Var1' and 'c_r1c1' ) to search a parameter:power relationship
# note: Can't use method = "ncp" with search)
umxPower(m1, update = c("c_r1c1", "age_b_Var1"), method = 'empirical', n=90, explore = TRUE)
umxPower(m1, update = c("c_r1c1"), method = 'empirical', n=90, explore = TRUE)


## End(Not run)
```

umxPrint_double_entry_data *Pretty-print method for double-entry censored datasets*

Description

Pretty-print method for double-entry censored datasets

Usage

```
umxPrint_double_entry_data(x, n = 6, ...)
```

Arguments

x	A data frame prepared with umx_make_double_entry_data().
n	Number of data frame rows to display below summary table (default = 6).
...	Additional arguments passed to print.

umxRAM *Build and run path-based SEM models*

Description

umxRAM expedites creation of structural equation models, still without doing invisible things to the model. It supports umxPath(). To support cross-language sharing and science learning, umxRAM also supports lavaan model strings.

Here's a path example that models miles per gallon (mpg) as a function of weight (wt) and engine displacement (disp) using the widely used mtcars data set.

```
m1 = 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() (including setting labels). umxRAM wraps these paths up, takes the data = input, and then internally sets sane start values. It also runs the model, and calls umxSummary().

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

nb: 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 also, 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 = df).

String Syntax

Here is an example using lavaan syntax (for more, see umxLav2RAM())

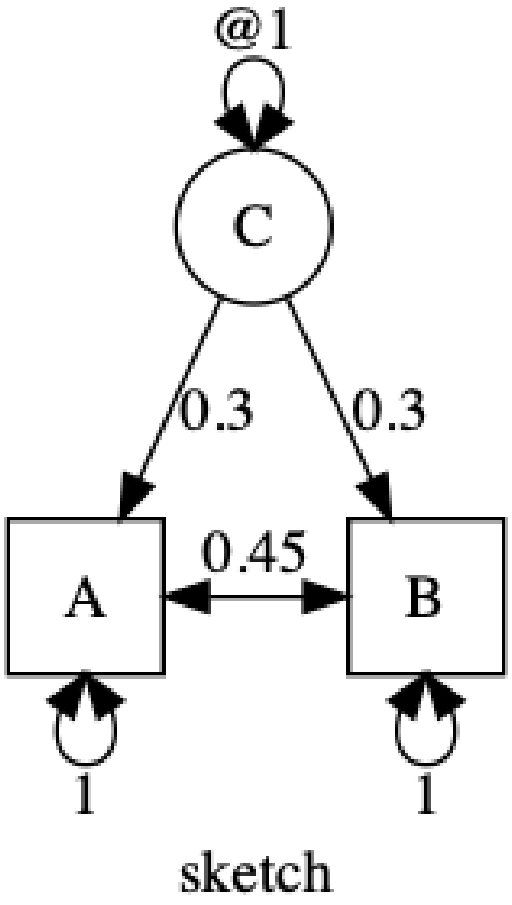
```
m1 = umxRAM("mpg ~ wt + disp", data = mtcars)
```

Sketch mode

If you are at the "sketching" stage of theory consideration, umxRAM supports setting data to a simple vector of manifest names. As usual in umxRAM, any variables you refer to that are not in data are treated as latents.

```
m1 = umxRAM("sketch", data = c("A", "B"),
 umxPath("C", to = c("A", "B"), values=.3),
 umxPath("A", with = "B", values=.45),
 umxPath(v.m. = c("A", "B")),
 umxPath(v1m0 = "C")
)
plot(m1, means = FALSE)
```

Will create this figure:



Usage

```
umxRAM(
  model = NA,
  ...,
  data = NULL,
  name = NA,
  group = NULL,
  group.equal = NULL,
  suffix = "",
  comparison = TRUE,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  weight = NULL,
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  std = FALSE,
  refModels = NULL,
  remove_unused_manifests = TRUE,
  independent = NA,
  setValues = TRUE,
  optimizer = NULL,
  verbose = FALSE,
  std.lv = FALSE,
  lavaanMode = c("sem", "lavaan"),
  printTab = FALSE
)
```

Arguments

model	A model to update (or set to string to use as name for new model)
...	umxPaths, mxThreshold objects, etc.
data	data for the model. Can be an OpenMx::mxData() or a data.frame
name	A friendly name for the model
group	(optional) Column name to use for a multi-group model (default = NULL)
group.equal	In multi-group models, what to equate across groups (default = NULL: all free)
suffix	String to append to each label (useful if model will be used in a multi-group model)
comparison	Compare the new model to the old (if updating an existing model: default = TRUE)
type	One of "Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"
weight	Passes weight values to mxData
allContinuousMethod	"cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.

autoRun	Whether to run the model (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
std	Whether to show standardized estimates, raw (NULL print fit only)
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)
std.lv	Whether to auto standardize latent variables when using string syntax (default = FALSE)
lavaanMode	Defaults when building out string syntax default = "sem" (alternative is "lavaan", with very few defaults)
printTab	(for string input, whether to output a table of paths (FALSE)

Details

WLS umxRAM supports WLS estimation via the type argument ("WLS", "DWLS", or "ULS").

Important for ordinal data: If your data contains ordered factors, umxRAM will automatically create the necessary mxThreshold objects. You do not need to add them manually.

For all-continuous data, use allContinuousMethod to control means modeling:

- "cumulants" (default): Faster. No means model.
- "marginals": Includes means and supports missing data.

Comparison for OpenMx users

umxRAM differs from OpenMx::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 don't need to create mxData when you already have a data.frame.
5. You add data with data = (as elsewhere in R, e.g. lm()).
6. You don't need to add labels: paths are automatically labelled "a_to_b" etc.
7. You don't need to set start values, they will be done for you.
8. You don't need to mxRun the model: it will run automatically, and print a summary.
9. You don't need to run summary: with autoRun=TRUE, it will print a summary.

10. You get a plot of the model with estimates on the paths, including multiple groups.
11. Less typing: umxPath() offers powerful verbs to describe paths.
12. Supports a subset of lavaan string input.

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().

Comparison with other software

Some SEM software does a lot of behind-the-scenes defaulting and path addition. If you want this, umxRAM can read lavaan strings so you can carry on that way.

WLS, DWLS, and ULS in umxRAM vs lavaan

umxRAM supports three weighted least squares estimators via the type argument. These control which weight matrix is used during estimation:

umx type	Weight matrix	Closest lavaan equivalent	Typical use
"WLS"	Full weight matrix	estimator = "WLS"	Full-information WLS (com
"DWLS"	Diagonal weight matrix	estimator = "DWLS" or the estimation step of "WLSMV"	Most common for ordinal da
"ULS"	Identity matrix	estimator = "ULS"	Simpler unweighted estimati

Important notes:

- In lavaan, WLSMV combines diagonal weighting (DWLS-style estimation) with a mean-and-variance adjusted test statistic. In umx/OpenMx, the weighting is controlled by type, while robust corrections to the chi-square and standard errors are handled via its Jacobian, and information matrix.
- When using ordinal data, make sure your variables are ordered factors. umxRAM will then automatically create the required mxThreshold objects.

Value

- OpenMx::mxModel()

References

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

See Also

umxPath(), umxSummary(), plot(), parameters(), umxSuperModel(), umxLav2RAM()

Other Core Model Building Functions: umx, umxLISREL(), umxMatrix(), umxModify(), umxPath(), umxRAM_DE(), umxSuperModel()

Examples

```
## Not run:

# ==========================================
# = 1. Here's a simple example with raw data =
# ==========================================
mtcars$litres = mtcars$disp/61.02
m1 = umxRAM("tim", data = mtcars,
 umxPath(c("wt", "litres"), to = "mpg"),
 umxPath("wt", with = "litres"),
 umxPath(v.m. = c("wt", "litres", "mpg"))
)

# 2. Use parameters to see the parameter estimates and labels
parameters(m1)

# And umxSummary to get standardized parameters, CIs etc from the run model.
umxSummary(m1, std=TRUE)
# |name           | Std.Estimate| Std.SE|CI                   |
# |:--------------|------------:|------:|:--------------------|
# |wt_to_mpg      |        -0.54|   0.17|-0.54 [-0.89, -0.2]  |
# |disp_to_mpg    |        -0.36|   0.18|-0.36 [-0.71, -0.02] |
# |mpg_with_mpg   |         0.22|   0.07|0.22 [0.08, 0.35]    |
# |wt_with_wt     |         1.00|   0.00|1 [1, 1]             |
# |b1             |         0.89|   0.04|0.89 [0.81, 0.96]    |
# |disp_with_disp |         1.00|   0.00|1 [1, 1]             |

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

# ================================================
# = lavaan string example (more at ?umxLav2RAM) =
# ================================================
m1 = umxRAM(data = mtcars, "#modelName
 mpg ~ wt + disp")


# ========================
# = A multi-group model =
# ========================

mtcars$litres = mtcars$disp/61.02
m1 = umxRAM("tim", data = mtcars, group = "am",
 umxPath(c("wt", "litres"), to = "mpg"),
 umxPath("wt", with = "litres"),
 umxPath(v.m. = c("wt", "litres", "mpg"))
)
# In this model, all parameters are free across the two groups.

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

```
# = 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:
m1 = 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")
tmp = mtcars; tmp$disp= tmp$disp *.016
myCov = mxData(cov(tmp[, selVars]), type = "cov", numObs = nrow(mtcars) )

m1 = 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")
 )
# 2. Switch to marginals to support means in WLS model
 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")
 )

#########################
# Ordinal-data WLS model #
```

```
##########################

# 1. Generate Data
set.seed(12345)
n  = 800
f  = rnorm(n)
x1 = 0.75 * f + rnorm(n, sd = 0.66)
x2 = 0.80 * f + rnorm(n, sd = 0.60)
x3 = 0.70 * f + rnorm(n, sd = 0.71)
x4 = 0.85 * f + rnorm(n, sd = 0.53)

dat = data.frame(x1 = x1, x2 = x2, x3 = x3, x4 = x4)

# Turn into ordered categorical (4 categories)
dat[] = lapply(dat, function(x) cut(x, breaks = 4, labels = FALSE))
dat[] = lapply(dat, ordered)

# 1. Model in lavaan
model = ' f =~ x1 + x2 + x3 + x4 '
lav1  = cfa(model, data = dat, ordered = names(dat), estimator = "WLSMV")

# 2. Model in umxRAM
# todo: Note if not clear: Data can be raw, or mxData()
# todo: implement m.v. as a synonym for v.m.?
m1 = umxRAM("WLS_umx", data = dat, type = "WLS", allContinuousMethod = "cumulants",
 # or "marginals"
 umxPath("f", to = c("x1", "x2", "x3", "x4") ),
 umxPath(v.m. = "f"),
 umxPath(v.m. = c("x1", "x2", "x3", "x4"))
)

# ==============================
# = Using umxRAM in Sketch mode =
# ==============================
# No data needed: just list variable names!
# Resulting model will be plotted automatically
m1 = umxRAM("what does unique pairs do, I wonder", data = c("A", "B", "C"),
   umxPath(unique.pairs = c("A", "B", "C"))
)

m1 = umxRAM("ring around the roses", data = c("B", "C"),
  umxPath(fromEach = c("A", "B", "C"))
)

m1 = umxRAM("fromEach with to", data = c("B", "C"),
   umxPath(fromEach = c("B", "C"), to= "D")
)

m1 = umxRAM("CFA_sketch", data = paste0("x", 1:4),
 umxPath("g", to = paste0("x", 1:4)),
 umxPath(var = paste0("x", 1:4)),
 umxPath(v1m0 = "g")
)
```

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

# ===========
# = Weights =
# ===========
# toy example weighting rows by cylinders.
mtcars$litres = mtcars$disp/61.02
m1 = umxRAM("tim", data = mtcars, weight= "cyl",
 umxPath(c("wt", "litres"), to = "mpg"),
 umxPath("wt", with = "litres"),
 umxPath(v.m. = c("wt", "litres", "mpg"))
)


## End(Not run)
```

---

umxRAM2Lav *Convert a RAM model to a lavaan string*

Description

Takes an OpenMx RAM model and creates the corresponding lavaan syntax string.

This function is at the alpha quality stage, and **should be expected to have bugs**. Also likely to change functionality and even parameters as new features are supported (e.g. groups) and lavaan-style strings exported. Several features are not yet supported. Let me know if you would like them.

Usage

```
umxRAM2Lav(model)
```

Arguments

model	an OpenMx RAM model

Value

A lavaan syntax string, e.g. "A~~B"

See Also

- umxLav2RAM(), umxRAM()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
## Not run:
umxRAM2Lav(umxLav2RAM("x ~ y", autoRun = FALSE, printTab = FALSE, lavaanMode = "lavaan"))

## End(Not run)
```

---

umxRAM_DE *Build and run path-based Double-entry SEM models including censored data*

Description

umxRAM_DE expedites SEM with censored data. Each name in DEvars is a **latent** trait. Prepared _cont / _cens columns are indicators of that latent (loading fixed at 1, residual fixed at 0). Write ordinary umxPath() calls using the base name (e.g. "litres"). Do not Cartesian-expand to _cont/_cens.

Fully continuous variables may be mixed with double-entry pairs. Prepare the censored traits with umx_make_double_entry_data(). At least one DEvar is required.

As you can see from the examples below, most of the work is done by umxPath(). umxRAM_DE wraps these paths up, takes the data = input, and then builds the latent measurement model for each DE trait and sets thresholds. By default it will also run it.

Gotchas 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 = df).

nb: 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.

If you are used to hacking a quick model with lavaan string syntax, use umxRAM() at present. Likewise, if you are at the "sketching" stage of theory consideration, umxRAM supports that, umxRAM_DE does not.

Usage

```
umxRAM_DE(
  model = NA,
  ...,
  data = NULL,
  DEvars = NULL,
  doubleEntrySuffix = c("_cont", "_cens"),
  fixCensorThresholds = c("yes", "auto", "no"),
  censorCuts = NULL,
  sep = NULL,
  name = NA,
  group = NULL,
  group.equal = NULL,
  suffix = "",
  comparison = TRUE,
  type = c("Auto", "FIML"),
  weight = NULL,
  allContinuousMethod = c("cumulants", "marginals"),
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  std = FALSE,
  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)
...	umxPaths, mxThreshold objects, etc.
data	data for the model. Can be an OpenMx::mxData() or a data.frame
DEvars	A character vector of base names for double-entry variables (e.g. c("litres") creates litres_cont/litres_cens). Base names only; do not include suffix.
doubleEntrySuffix	Suffixes for the continuous and censored variables (default = c("_cont", "_cens")).
fixCensorThresholds	One of c("yes","auto","no"). "yes" fix every DE pair from censorCuts/prep attr; "auto" fix only pairs with finite known cut; "no" free thresholds.
censorCuts	Optional named numeric vector of known cuts on analysis scale. Names may be base ("litres"), "_cont" or "_cens" form.
sep	Separator used in prep (default NULL infers from attr(data,"umxDoubleEntry")$sep or "").
name	A friendly name for the model.

group	(optional) Column name to use for a multi-group model (default = NULL). Fixed DE thresholds (known cut) are applied in every group.
group.equal	In multi-group models, what to equate across groups (default = NULL: all free)
suffix	String to append to each label (useful if model will be used in a multi-group model)
comparison	Compare the new model to the old (if updating an existing model: default = TRUE)
type	One of "Auto" or "FIML". note: other types, e.g., "WLS" are rejected (use umxRAM()).
weight	Passes weight values to mxData
allContinuousMethod	"cumulants" or "marginals". Used in all-continuous WLS data to determine if a means model needed.
autoRun	Whether to run the model (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
std	Whether to show standardized estimates, raw (NULL print fit only)
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

WLS / cov / cor type = "WLS", "DWLS", "ULS", "cov", or "cor" is not valid in umxRAM_DE. Double-entry needs raw-data FIML (each row is either the continuous density or the threshold CDF). Those types fit a correlation/moment structure and cannot use the mutual-NA _cont/_cens pattern. For WLS/cov/cor on ordinary (non-DE) data, use umxRAM().

Value

- OpenMx::mxModel()

References

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

See Also

umxPath(), umxSummary(), plot(), parameters(), umxSuperModel(), umxLav2RAM()

Other Core Model Building Functions: umx, umxLISREL(), umxMatrix(), umxModify(), umxPath(), umxRAM(), umxSuperModel()

Examples

```
# Here's a path example that models miles per gallon (mpg) as a function of weight (wt) and
# engine displacement (disp) using the widely used `mtcars` data set.

# 1: Create the censored data
# Just for a demo, we will create a censored litres column, censoring at 2.
data(mtcars)
tmp = mtcars
tmp$litres = tmp$disp / 61.02
tmp = umx_make_double_entry_data(tmp, cols = list(litres= 2), sep="", nSib=1)
# 2: Create the RAM_DE model
m1 = umxRAM_DE("test", data = tmp, DEvars = c("litres"),
 umxPath("litres", to = "mpg"),
 umxPath("wt", to = "mpg"),
 umxPath("wt", with = "litres"),
 umxPath(v.m. = c("litres", "wt", "mpg"))
)
```

---

umxRAM_GLM *Path-based RAM model with mxFitFunctionGLM*

Description

Standalone counterpart to umxRAM() for conditional exponential-family manifests. Latents are anything in the paths that is not in data. Manifests are anything in the paths (or in families) that is in data. Residual S on family items is fixed at 0. An observed column used only as from (a predictor) is not given residual S or a mean; the arrow uses that row's value. Manifests not listed in families that have residual S are Gaussian. Does not call mxRefModels() (no CFI yet).

Usage

```
umxRAM_GLM(
  model = NA,
  ...,
  data = NULL,
  families = NULL,
  theta = NULL,
  nAGQ = NA_integer_,
  name = NA,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
```

```
  setValues = TRUE,
  suffix = "",
  verbose = FALSE
)
```

Arguments

model	a model name (character). Updating an existing model is not supported.
...	umxPath() / mxPath() objects, optional mxMatrix(), mxConstraint(), and mxFamily() objects. Not data.
data	a data.frame or raw mxData(). Required (used to detect manifests vs latents and to set starts).
families	named list of stats::family or mxFamily() objects. Names are manifest variables. Items not listed are Gaussian.
theta	optional named numeric (fixed) or character (free-parameter label) for NB size / Gamma shape / inverse-Gaussian lambda.
nAGQ	passed to mxFitFunctionGLM(). NA (default) uses OpenMx's rule: 5 for binomial with one latent, else 1.
name	optional name; if NA, model is used.
autoRun	whether to run (default umx_set_auto_run()).
tryHard	xmu_safe_run_summary() tryHard options.
setValues	if TRUE, set start values (means by family; Gaussian residual variances from the data; GLM-item S at 0).
suffix	passed to xmuLabel().
verbose	currently unused; kept for umxRAM-like calling.

Value

an MxModel() (invisibly if autoRun prints a summary)

See Also

umxRAM(), mxFamily(), mxFitFunctionGLM()

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxJiggle(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

```
## Not run:
set.seed(1)
dat = data.frame(y = rpois(80, lambda = exp(0.3)))
m1 = umxRAM_GLM("poi",
 umxPath("one", to = "y"),
 data = dat,
 families = list(y = poisson()),
 autoRun = TRUE)

## End(Not run)
```

umxReduce *Reduce models, and report the results.*

Description

Given a umx model (currently umxACE and umxGxE are supported - ask for more!) umxReduce will conduct a formalised reduction process. It will also report Akaike weights are also reported showing relative support across models.

Specialized functions are called for different type of input:

1. **GxE model reduction** For umxGxE() models umxReduceGxE() is called.
2. **ACE model reduction** For umxACE() models,umxReduceACE() is called.

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

umxReduce can be extended to new cases as demand emerges.

Usage

```
umxReduce(
  model,
  report = c("markdown", "inline", "html"),
  intervals = TRUE,
  testD = TRUE,
  baseFileName = "tmp",
  tryHard = "yes",
  silent = FALSE,
  ...
)
```

Arguments

model	The OpenMx::mxModel() which will be reduced.
report	How to report the results. "html" = open in browser
intervals	Recompute CIs (if any included) on the best model (default = TRUE)
testD	Whether to test ADE and DE models (TRUE)
baseFileName	(optional) custom filename for html output (defaults to "tmp")
tryHard	Default = "yes"
silent	Default = FALSE
...	Other parameters to control model summary

References

Wagenmakers, E.J., & Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin and Review*, **11**, 192-196. doi:10.3758/BF03206482

See Also

umxReduceGxE(), umxReduceACE()

Other Model Summary and Comparison: umx, umxCompare(), umxEquate(), umxMI(), umxSetParameters(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

---

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"),
  intervals = TRUE,
  testD = TRUE,
  baseFileName = "tmp",
  tryHard = c("yes", "no", "ordinal", "search"),
  silent = FALSE,
  digits = 2,
  ...
)
```

Arguments

model	an ACE or ADE OpenMx::mxModel() to reduce
report	How to report the results. "html" = open in browser
intervals	Recompute CIs (if any included) on the best model (default = TRUE)
testD	Whether to test ADE and DE models (TRUE)
baseFileName	(optional) custom filename for html output (defaults to "tmp")
tryHard	(default = "yes")
silent	Don't print the ACE models (default = FALSE)
digits	rounding in printout (default = 2)
...	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

Wagenmakers, E.J., & Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin and Review*, **11**, 192-196. doi:10.3758/BF03206482

See Also

umxReduceGxE(), umxReduce()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACE(selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "")

# ============================================================================
# = Table of parameters + fit comparisons, ready too copy to word processor =
# ============================================================================
umxReduce(m1, silent=TRUE, digits=2, repo="h")

# ========================================
# = Function captures the preferred model =
# ========================================
m2 = umxReduce(m1)
umxSummary(m2)

# works for ADE input also
m1 = umxACE(selDVs = "bmi", dzData = dzData, mzData = mzData, sep = "", dzCr = .25)


## End(Not run)
```

umxReduceGxE *Reduce a GxE model.*

Description

This function can perform model reduction for umxGxE() models, testing dropping 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 = "html" to open a table for pasting into a word processor.

In addition to printing a table, the function returns the preferred model.

Usage

```
umxReduceGxE(
  model,
  report = c("markdown", "inline", "html", "report"),
  intervals = TRUE,
  testD = TRUE,
  baseFileName = "tmp_gxe",
  tryHard = c("yes", "no", "ordinal", "search"),
  silent = FALSE,
  ...
)
```

Arguments

model	A umxGxE() to reduce.
report	How to report the results. default = "markdown". "html" = open in browser.
intervals	Recompute CIs (if any included) on the best model (default = TRUE)
testD	Whether to test ADE and DE models (TRUE)
baseFileName	(optional) custom filename for html output (default = "tmp").
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
silent	Default (FALSE)
...	Other parameters to control model summary.

Value

best model

References

Wagenmakers, E.J., & Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin and Review*, **11**, 192-196. doi:10.3758/BF03206482.

See Also

umxReduce(), umxReduceACE()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

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

## End(Not run)
```

---

umxRenameMatrix *Rename a umxMatrix (even in a model)*

Description

Rename a umxMatrix(), including updating its labels to match the new name.

Usage

```
umxRenameMatrix(x, matrixName, name)
```

Arguments

x	A model or matrix
matrixName	Name of the matrix
name	The new name

Value

- updated matrix or model with updated matrix in it.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(),

xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(twinData) # ?twinData from Australian twins.
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 10
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m1  = umxACE(selDVs= "ht", sep= "", dzData= dzData, mzData= mzData, autoRun= FALSE)
tmp = umxRenameMatrix(m1$top, matrixName = "a", name="hello")
umx_check(tmp$hello$labels == "hello_r1c1") # new is there
umx_check(is.null(tmp$a))                   # old is gone

## End(Not run)
```

---

umxRotate *Generic SEM factor model loading rotation function*

Description

See umxRotate.MxModelCP() to rotate the factor loadings of a umxCP() model

Usage

```
umxRotate(
  model,
```

```
  rotation = c("varimax", "promax"),
  tryHard = "yes",
  freeLoadingsAfter = TRUE,
  verbose = TRUE
)
```

Arguments

model	a model to rotate
rotation	name of the rotation.
tryHard	Default ("yes") is to tryHard
freeLoadingsAfter	Whether to keep the rotated loadings fixed (Default, free them again)
verbose	print detail about the rotation

Value

- Rotated solution

See Also

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxSummaryACE_DE()

---

umxRotate.MxModelCP *Rotate a CP solution*

Description

Rotate a CP solution. Should work with rotations provided in libs("GPArotation") and libs("psych"), e.g.,

Orthogonal: "varimax", "quartimax", "bentlerT", "equamax", "varimin", "geominT" and "bifactor"

Oblique: "Promax", "promax", "oblimin", "simplimax", "bentlerQ", "geominQ", "biquartimin" and "cluster"

Usage

```
## S3 method for class 'MxModelCP'
umxRotate(
  model,
  rotation = c("varimax", "promax"),
  tryHard = "yes",
  freeLoadingsAfter = TRUE,
  verbose = TRUE
)
```

Arguments

model	a umxCP() model to rotate.
rotation	name of the rotation.
tryHard	Default ("yes") is to tryHard.
freeLoadingsAfter	return the model with factor loadings free (default) or fixed in the new locations.
verbose	print detail about the rotation

Details

This works by taking the common-pathways loadings matrix from a solved umxCP() model, rotating these, placing them back into the loadings matrix, re-estimating the model with the parameters fixed at this rotation, then return the new model.

Value

- Rotated solution.

See Also

- umxCP()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# Rotate a CP solution(param)
# Common pathway model rotation
library(umx)
# Fit 3 factor CPM
data(GFF)
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP(selDVs = selDVs, nFac = 2, data = data, tryHard = "yes")
m2 = umxRotate(m1, rotation = "varimax",  tryHard = "yes")

## End(Not run)
```

umxRun *umxRun: Run an mxModel*

Description

umxRun is a version of OpenMx::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

```
umxRun(
  model,
  tryHard = c("yes", "no", "ordinal", "search"),
  calc_sat = TRUE,
  setValues = FALSE,
  setLabels = FALSE,
  summary = !umx_set_silent(silent = TRUE),
  intervals = FALSE,
  optimizer = NULL,
  comparison = NULL
)
```

Arguments

model	The OpenMx::mxModel() you wish to run.
tryHard	How to tryHard. Default = "yes". Alternatives "no", "ordinal", "search"
calc_sat	Whether to calculate the saturated and independence models (for raw OpenMx::mxData() OpenMx::mxModel()s)
setValues	Whether to set the starting values of free parameters (default = FALSE)
setLabels	Whether to set the labels (default = FALSE)
summary	Whether to print summary or not (default = !umx_set_silent())
intervals	Whether to run mxCI confidence intervals (default = FALSE) intervals = FALSE
optimizer	optional to set the optimizer.
comparison	Comparison model (will be used to drive umxCompare() after umxRun

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxJiggle(), umxRAM_GLM(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel(), xmuValues()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 = mxModel("fact", type="RAM", manifestVars=manifests, latentVars=latents,
 mxPath(latents  , to = manifests),
 mxPath(manifests, arrows = 2),
 mxPath(latents  , arrows = 2, free = FALSE, values = 1),
 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, std = TRUE)
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, tryHard = "yes")

## End(Not run)
```

---

umxSetParameters *Change or fix parameters (e.g. their values, labels, bounds, ..) in a model.*

Description

umxSetParameters is used to alter values, and other parameter properties in an OpenMx::mxModel(). A common use is setting new values and changing parameters from free to false. *Note*: If you just want to modify and re-run a model, you probably want umxModify().

Usage

```
umxSetParameters(
  model,
  labels,
  free = NULL,
  values = NULL,
  newlabels = NULL,
```

```
  lbound = NULL,
  ubound = NULL,
  indep = FALSE,
  strict = TRUE,
  name = NULL,
  regex = FALSE,
  test = FALSE
)
```

Arguments

model	an OpenMx::mxModel() to set parameters in.
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	patterns to match for labels (or if TRUE, use labels as regular expressions)
test	Just show what you would do? (defaults to FALSE)

Details

Using umxSetParameters, you use labels= to select the parameters you want to update. You can set their free/fixed state with free=, and set new values with values = . Likewise for bounds.

umxSetParameters supports pattern matching (regular expressions) to select labels. Set regex= to a regular expression matching the labels you want to select. e.g. "G_to_.*" would match "G_to_anything".

Details Internally, umxSetParameters is equivalent to a call to omxSetParameters where you have the ability to generate a pattern-based label list, and, because this can create duplicate labels, we also call OpenMx::omxAssignFirstParameters() to equate the start values for parameters which now have identical labels.

Value

- OpenMx::mxModel()

References

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

See Also

- umxModify(), xmuLabel()

Other Model Summary and Comparison: umx, umxCompare(), umxEquate(), umxMI(), umxReduce(), umxSummary(), xmu_compare_WLS(), xmu_robust_WLS_fit()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = mxData(demoOneFactor[1:80,], type = "raw"),
 umxPath(from = latents, to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = latents)
)
parameters(m1)
# Match all labels
umxSetParameters(m1, regex = "^", newlabels= "m1_", test = TRUE)

# Change path to x1 to x2, equating these two paths
m2 = umxSetParameters(m1, "G_to_x1", newlabels= "G_to_x2", test = FALSE)
m2 = umxRun(m2) # umxSetParameters does not re-run he model, so make sure you do!
parameters(m2)


## End(Not run)
```

---

umxSexLim *Multivariate sex limitation twin model*

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 and 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.

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.

Usage

```
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", "ordinal", "search"),
  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 selVars.
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).

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

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.

General restrictions: Assumes means and variances can be equated across birth order within zygosity groups.

Value

- `OpenMx::mxModel()` of subclass mxModel.CFSexLim

References

Neale, M. C., Roysamb, E., Jacobson, K. (2006). Multivariate genetic analysis of sex-lim and G x E interaction. *Twin Research & Human Genetics*, **9**, pp. 481–489. doi:10.1375/183242706778024937

See Also

`umxSummarySexLim()`, `umxPlotSexLim()`

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
# ========================
# = Load and Process Data =
# ========================
## 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"s
umxCompare(m1a, m2)

# ===============
# = Homogeneity =
# ===============

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(selDVs = c("bic", "tri"), sep = "_T", 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(selDVs = c("bic", "tri"), sep = "_T",
 A_or_C = "A", tryHard="yes", sexlim="Scalar",
 mzmData = mzmData, dzmData = dzmData,
 mzfData = mzfData, dzfData = dzfData,
 dzoData = dzoData
)
# Drop qualitative sex limitation
#  Distinct af and am (& c & e), but shared Ra (& Rc & Re) between variables
#   i.e., same correlations for males and females.
m1a = umxModify(m1 , regex = "^Ra[mfo]_", newlabels="^Ra_", name = "no_qual_a", comparison = TRUE)
m1b = umxModify(m1a, regex = "^Rc[mfo]_", newlabels="^Rc_", name = "no_qual_ac", comparison = TRUE)
m1c = umxModify(m1b, regex = "^Re[mfo]_", newlabels="^Re_", name = "no_qual_ace", comparison = TRUE)
umxCompare(m1, c(m1a, m1b, m1c, 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(m1a, m1b, m1c, m1d))

# ============================
# = Run multi-variate example =
# ============================
# Variables for Analysis
selDVs = c('ssc','sil','caf','tri','bic')
selDVs = c('ssc','tri','bic')
m1 = umxSexLim(selDVs = selDVs, sep = "_T", A_or_C = "A", tryHard = "yes",
mzmData = mzmData, dzmData = dzmData,
 mzfData = mzfData, dzfData = dzfData, dzoData = dzoData
)

m2 = umxSexLim(selDVs = selDVs, sep = "_T", A_or_C = "A", sexlim = "Nonscalar",
 tryHard = "yes",
mzmData = mzmData, dzmData = dzmData,
 mzfData = mzfData, dzfData = dzfData, dzoData = dzoData
)

# umxSummary(m1)
```

```
# summary(m1)
# summary(m1)$Mi

## End(Not run)
```

---

umxSimplex *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.

This code is beta quality: **not** for publication use.

Usage

```
umxSimplex(
  name = "simplex",
  selDVs,
  dzData,
  mzData,
  sep = "_T",
  equateMeans = TRUE,
  dzAr = 0.5,
  dzCr = 1,
  addStd = TRUE,
  addCI = TRUE,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  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 (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
optimizer	Optionally set the optimizer (default NULL does nothing).

Details

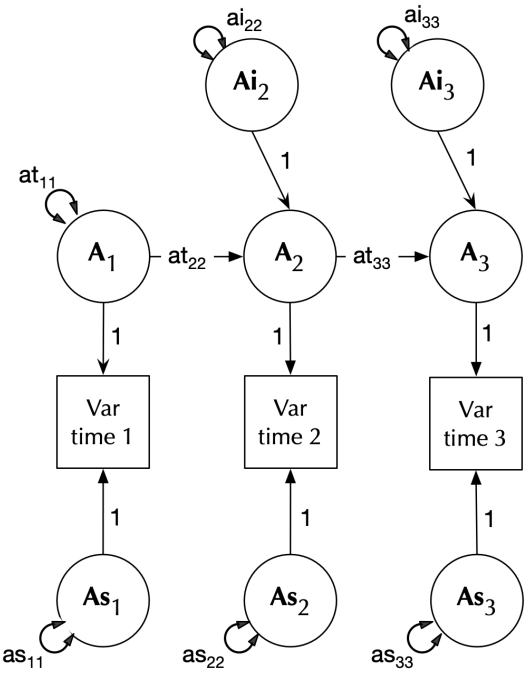
The simplex model decomposes phenotypic variance into Additive genetic, unique environmental (E) and, optionally, either common or shared-environment (C) or non-additive genetic effects (D).

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 multilevel 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% by twins in a pair), 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:

```
m1$top$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:

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

Value

- `OpenMx::mxModel()`

References

https://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: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umxTwinMaker()`, `umx_make_double_entry_data()`

Examples

```
## 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= "yes")

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, tryHard = "yes")

## 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 and 95% CI for each twin-type.

Set MZ and DZ to summarize the two-group case.

Usage

```
umxSummarizeTwinData(
  data = NULL,
  selVars = NULL,
  sep = "_T",
  zyg = "zygosity",
  age = "age",
  sex = "sex",
  MZ = NULL,
  DZ = NULL,
  MZFF = "MZFF",
  DZFF = "DZFF",
  MZMM = "MZMM",
  DZMM = "DZMM",
  DZOS = "DZOS",
  digits = 2,
  report = c("markdown", "html")
)
```

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, default= "_T" for wt_T1 and wt_T2.
zyg	The zygosity column in the dataset (default "zygosity").

age	The age column in the dataset (default "age").
sex	The sex column in the dataset (default "sex").
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 of zyg corresponding to MZ FF pairs: default= "MZFF".
DZFF	The level of zyg corresponding to DZ FF pairs: default= "DZFF".
MZMM	The level of zyg corresponding to MZ MM pairs: default= "MZMM".
DZMM	The level of zyg corresponding to DZ MM pairs: default= "DZMM".
DZOS	The level of zyg corresponding to DZ OS pairs: default= "DZOS".
digits	Rounding precision of the report (default 2).
report	What to return (default = 'markdown'). Use 'html' to open a web table.

Value

- Invisibly returns the formatted dataframe.

References

https://github.com/tbates/umx

See Also

- umxAPA()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

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 *Shows a compact, publication-style, summary of umx models*

Description

Report the fit of a RAM or other umx model (such as ACE, CP etc.) in a compact form suitable for reporting in a journal.

See documentation for RAM models summary here: `umxSummary.MxModel()`.

View documentation on the ACE model subclass here: `umxSummaryACE()`.

View documentation on the ACEv model subclass here: `umxSummaryACEv()`.

View documentation on the ACE_DE model subclass here: `umxSummaryACE_DE()`.

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 `OpenMx::mxModel()` whose fit will be reported
`...`	Other parameters to control model summary

See Also

Other Model Summary and Comparison: `umx`, `umxCompare()`, `umxEquate()`, `umxMI()`, `umxReduce()`, `umxSetParameters()`, `xmu_compare_WLS()`, `xmu_robust_WLS_fit()`

umxSummary.list *Summarize a Genomic SEM / LDSC covariance structure (covstruc)*

Description

Pretty-prints the trait list and the main LDSC/GSEM matrices (`S`, `I`, `V`, `N`, and scalar `m`) for objects like Psych_LDSC or the return value of `umxGSEM_ldsc()` (or OpenMx `imxLDSC` when present). Tables use markdown or HTML via umx_print.

Usage

```
## S3 method for class 'list'
umxSummary(
  model,
  digits = 3,
  report = c("markdown", "html"),
  matrices = c("S", "I", "V", "N"),
  traits = NULL,
  ...
)
```

Arguments

model	A covstruc list with at least matrices S (genetic cov) and V (sampling cov of vech(S)). Optional: I, N, m, S_Stand, V_Stand.
digits	Rounding for printed cells (default 3).
report	"markdown" (default) or "html" (browser-friendly kable).
matrices	Character vector of which components to show (default c("S", "I", "V", "N")). Unknown names are ignored. Scalar m is always mentioned in the header when present.
traits	Optional character vector of trait names to subset (default = NULL, which shows all).
...	Not used (S3 compatibility).

Value

Invisibly returns the input model (covstruc).

See Also

umxSummary.MxDataStatic(), Psych_LDSC, umxGSEM(), umx_print

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files, umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.MxDataStatic()

Other Model Summary: umxSummary.MxDataStatic(), umxSummary.MxLISRELModel()

Examples

```
data(Psych_LDSC)
umxSummary(Psych_LDSC)
## Not run:
umxSummary(Psych_LDSC, traits= c("MDD", "SCZ"), digits=5)
umxSummary(Psych_LDSC, report = "html")

## End(Not run)
```

umxSummary.MxDataStatic *Summarize WLS / summary-statistic OpenMx::mxData objects*

Description

Explains modern OpenMx WLS data for users who have not seen `type = "summary"` and `observedStats` (`cov`, `useWeight`, `asymCov`). Works for `MxDataStatic` from OpenMx::mxData (including GSEM-style summary data).

Usage

```
## S3 method for class 'MxDataStatic'
umxSummary(
  model,
  digits = 3,
  report = c("markdown", "html"),
  matrices = c("cov", "useWeight", "asymCov"),
  ...
)
```

Arguments

`model`	An OpenMx::mxData object (`MxDataStatic`).
`digits`	Rounding for printed cells (default 3).
`report`	`"markdown"` (default) or `"html"`.
`matrices`	Which observedStats blocks to print: any of `c("cov", "useWeight", "asymCov", "means", "thresholds")`. Default prints `cov`, `useWeight`, and `asymCov` when present. For LDSC-shaped content, extra args are passed through to `umxSummary.list()` (e.g. `matrices = c("S","I","V")`).
`...`	Passed to `umxSummary.list()` when LDSC-shaped `observedStats` is detected.

Details

If `observedStats` still holds **LDSC/GSEM covstruc keys** (`S`, `V`, ...) rather than modern WLS keys - e.g. `mxData(type="summary", observedStats = Psych_LDSC, numObs = 2)` - OpenMx stores those names as-is (it does **not** map `S->cov` or `V->asymCov`). This method detects that case, prints the LDSC content via `umxSummary.list()`, and notes how to build a working WLS `mxData`.

Value

Invisibly returns the input `model`.

See Also

umxSummary.list(), mxData, mxFitFunctionWLS, umxGSEM

Other GSEM: Anthro_LDSC, Psych_LDSC, plot.umx_GWAS(), umxGSEM(), umxGSEM_GWAS(), umxGSEM_dl_tutorial_files, umxGSEM_label_ldsc(), umxGSEM_ldsc(), umxGSEM_munge(), umxGSEM_std(), umxGSEM_sumstats(), umxSummary.list()

Other Model Summary: umxSummary.MxLISRELModel(), umxSummary.list()

Examples

```
data(Psych_LDSC)
S = Psych_LDSC$S[1:3, 1:3]
V = Psych_LDSC$V[1:6, 1:6]
W = diag(1 / diag(V))
dimnames(W) = dimnames(V)
## Not run:
# Needs modern OpenMx/GenomicMx (type = "summary" pure observedStats).
# CRAN OpenMx historically used type = "none" for this layout.
d = mxData(numObs = 1, type = "summary",
           observedStats = list(cov = S, useWeight = W, asymCov = V))
umxSummary(d)

## End(Not run)
```

---

umxSummary.MxLISRELModel

umxSummary.MxLISRELModel

Description

umxSummary.MxLISRELModel handles summary formatting for LISREL models.

Usage

```
## S3 method for class 'MxLISRELModel'
umxSummary(
  model,
  refModels = NULL,
  std = FALSE,
  digits = 2,
  report = c("markdown", "html"),
  means = TRUE,
  residuals = TRUE,
  SE = TRUE,
  filter = c("ALL", "NS", "SIG"),
  RMSEA_CI = FALSE,
  ...,
  matrixAddresses = FALSE
)
```

Arguments

model	A LISREL OpenMx::mxModel() to summarize.
refModels	Reference models for comparison.
std	Whether to show standardized path values.
digits	Number of decimal places to round to.
report	Formatting style: markdown or html.
means	Whether to print means.
residuals	Whether to print residual covariances.
SE	Whether to show parameter Standard Errors.
filter	Filter criteria for parameters.
RMSEA_CI	Print RMSEA Confidence Interval.
...	Extra arguments.
matrixAddresses	Print matrix addresses.

Value

- model

References

https://github.com/tbates/umx

See Also

umxLISREL(), plot.MxLISRELModel()

Other Model Summary: umxSummary.MxDataStatic(), umxSummary.list()

---

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. It reports parameters in a markdown or html table (optionally standardized), and fit indices RMSEA (an absolute fit index, comparing the model to a perfect model) and CFI and TLI (incremental fit indices comparing a model with the worst fit).

Usage

```
## S3 method for class 'MxModel'
umxSummary(
  model,
  refModels = NULL,
  std = FALSE,
  digits = 2,
  report = c("markdown", "html"),
  means = TRUE,
  residuals = TRUE,
  uncertainty = c("SE", "MLR", "CI", "none"),
  filter = c("ALL", "NS", "SIG"),
  RMSEA_CI = FALSE,
  SE = TRUE,
  matrixAddresses = FALSE,
  ...
)
```

Arguments

model	The OpenMx::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.
std	If TRUE, model is standardized (Default FALSE, NULL means "don't show").
digits	How many decimal places to report (Default 2)
report	If "html", then show results in browser (default = "markdown")
means	Whether to include means in the summary (TRUE)
residuals	Whether to include residuals in the summary (TRUE)
uncertainty	What type of uncertainty/inference packaging to report: "SE" (default SEs from the fit), "MLR" (engine-appropriate robust inference: ML casewise sandwich SEs + robust AFIs; for WLS, moment-sandwich SEs + robust WLS AFIs when available), "CI" (profile likelihood CIs), or "none".
filter	whether to show significant paths (SIG) or NS paths (NS) or all paths (ALL)
RMSEA_CI	Whether to compute the CI on RMSEA (Defaults to FALSE)
SE	Deprecated. Please use uncertainty instead.
matrixAddresses	Whether to show "matrix address" columns (Default = FALSE)
...	Other parameters to control model summary

Details

umxSummary alerts you when model fit is worse than accepted criterion (TLI >= .95 and RMSEA <= .06; (Hu & Bentler, 1999; Yu, 2002).

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

Robust inference (`uncertainty = "MLR"`)

This option means **engine-appropriate robust inference**, not one sandwich formula for every fit type:

- **ML / FIML (raw continuous):** casewise score sandwich SEs via `OpenMx::imxRobustSE()`, plus robust CFI/TLI/RMSEA via `xmu_robust_ML_fit()` (Yuan-Bentler / Brosseau-Liard & Savalei scaling).
- **WLS / DWLS / ULS:** do **not** use the ML casewise sandwich. Parameter SEs are the usual **WLS/GMM moment sandwich** already returned by OpenMx (W, Δ, Γ). Robust CFI/TLI/RMSEA are applied when `output$implied_jacobian` is present via `xmu_robust_WLS_fit()` (Satorra-Bentler 2010 for continuous WLS; Savalei 2021 catML for ordinal WLS). A statistical note states that SEs are WLS asymptotic (moment sandwich).

Robust Fit Statistics for ML Standard Maximum Likelihood (ML) is powerful and efficient, with chi2, CFI, TLI, and RMSEA measures of fit. Non-normality and heteroscedasticity distort the sampling distribution of the scores and the ML test statistic itself, necessitating changes to SEs and fit statistics. For ML, `uncertainty = "MLR"` implements the sandwich estimator for SEs and MLR-style fit scaling (Yuan-Bentler / Savalei 2018-style robust indices).

Robust Fit Statistics for WLS/DWLS (The GenomicMx Ecosystem)

Simulation work shows CFI/TLI behave differently under DWLS/WLS than under ML (Shi et al., 2020): incremental indices use an independence baseline under the weight matrix, so conventional cutoffs (e.g. CFI > 0.95) break down for continuous WLS and Genomic SEM. Prefer **SRMR** and residual inspection for absolute fit; use `umxCompare()` for nested tests. See `?umxCompare` for the full guidance.

Implementation: when a Jacobian is available, `umxSummary` routes WLS models through `xmu_robust_WLS_fit()` (independence baseline + trace scaling). Parameter SEs remain the OpenMx WLS moment sandwich (see `uncertainty = "MLR"` above). Robust CFI/TLI/RMSEA are **reported** for continuous WLS but should be treated as descriptive (no Hu-Bentler cutoffs). Genomic SEM has the same rule, with even less trust in RMSEA under bookkeeping N.

Two statistic families (by design). WLS output separates omnibus testing from fit indices:

- **Displayed χ^2 and p** - Satorra-Bentler (2010) scaled WLS test when robust fit succeeds (continuous and ordinal).
- **Displayed CFI, TLI, RMSEA** - robust/scaled indices; interpretation depends on continuous vs ordinal (below).

Continuous WLS (all manifests numeric): SB (2010) scaled pipeline ($\chi^2 = F/c$ from `output$fit`, same Chi as `umxCompare()` when Jacobians are present). Saturated residual df (ChiDoF = 0): Chi = 0, p = NA, CFI = TLI = 1, RMSEA = 0 (not NA). Printed note: no conventional cutoffs; prefer SRMR; nested Strict SB via `umxCompare()`. **Ordinal / categorical WLS** (at least one `ordered`/`factor` on raw data): robust CFI/TLI/RMSEA use **Savalei (2021)** catML (Categorical Maximum Likelihood) corrections; **Hu & Bentler (1999) cutoffs (CFI > .95, RMSEA < .06) apply to these robust indices**. Displayed χ^2/p remain SB-scaled WLS omnibus tests.

- **catML (Categorical Maximum Likelihood):** Evaluates parameter estimates inside a Maximum Likelihood correlation scaffold to obtain non-inflated fit indices for ordinal data.
- **c_{model} and c_{null}:** Mean-and-variance scaling correction multipliers for model and baseline discrepancies (values near 1.0 indicate sample sampling variability matches standard ML expectations; values > 1.0 adjust for categorical overdispersion).

- **Suggested Manuscript Copy:** *"Model fit was evaluated using DWLS with robust categorical ML fit indices (Savalei, 2021; CFI = X.XX, RMSEA = X.XX)."*

Requires `implied_jacobian` after `mxRun` on WLS. If robust computation fails, falls back to OpenMx `summary()` with a note.

CIs and Identification This function uses the standard errors reported by OpenMx to produce the CIs you see in umxSummary. These are used to derive confidence intervals based on the formula 95%CI = estimate +/- 1.96*SE.

Sometimes SEs appear NA. This may reflect a model which is not identified (see https://davidakenny.net/cm/identify.htm). This can include empirical under-identification - for instance two factors that are essentially identical in structure. Use `OpenMx::mxCheckIdentification()` to check identification.

Solutions: If there are paths estimated at or close to zero, this suggests that fixing one or two of these to zero may fix the standard error calculation.

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

Value

- parameterTable returned invisibly, if estimates requested

References

- Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling*, **6**, 1-55.
- Satorra, A., & Bentler, P. M. (2010). Ensuring positiveness of the scaled difference chi-square test statistic. *Psychometrika*, **75**(2), 243-248.
- Satorra, A. and Bentler P. M. (1994). Corrections to test statistics and standard errors in covariance structure analysis. Latent variables analysis: Applications for developmental research. A. von Eye and C. C. Clogg, Sage: 399-419.
- Savalei, V. (2018). On the computation of the RMSEA and CFI from the mean-and-variance corrected test statistic with nonnormal data in SEM. *Multivariate Behavioral Research*, **53**(3), 419–429.
- Savalei, V. (2021). Improving fit indices in SEM with categorical data. *Multivariate Behavioral Research*, **56**(3), 390–407.
- Brosseau-Liard, P. E., & Savalei, V. (2012). Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research*, **47**(5), 647–677.
- Yu, C.Y. (2002). Evaluating cutoff criteria of model fit indices for latent variable models with binary and continuous outcomes. University of California, Los Angeles. Retrieved from https://www.statmodel.com/download/Yudissertation.pdf
- Yuan, K.-H. and P. M. Bentler (2000). "5. Three Likelihood-Based Methods for Mean and Covariance Structure Analysis with Nonnormal Missing Data." Sociological Methodology 30(1): 165-200.

See Also

- umxRAM(), xmu_robust_WLS_fit()

Other Summary functions: umxSummaryACEcov(), umxSummaryCP(), umxSummaryGxE(), umxSummaryIP(), umxSummaryMRDoC()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
umxSummary(m1, std = TRUE)
# output as latex
umx_set_table_format("latex")
umxSummary(m1, std = TRUE)
umx_set_table_format("markdown")
# output as raw
umxSummary(m1, std = FALSE)

# switch to a raw data model
m1 = umxRAM("One Factor", data = demoOneFactor[1:100, ],
 umxPath("G", to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = "G")
)
umxSummary(m1, std = TRUE, filter = "NS")

## End(Not run)
```

---

umxSummary.MxModelGLM *Summary for a RAM model with mxFitFunctionGLM*

Description

Parameter table and -2LL / AIC / BIC. No CFI/TLI/RMSEA (no saturated GLM pair yet). Nested tests: umxCompare() / AIC. Called from umxSummary() when the fit function is GLM; not a new S4 class.

Usage

```
## S3 method for class 'MxModelGLM'
umxSummary(
```

```
  model,
  refModels = NULL,
  std = FALSE,
  digits = 2,
  report = c("markdown", "html"),
  means = TRUE,
  residuals = TRUE,
  uncertainty = c("SE", "MLR", "CI", "none"),
  filter = c("ALL", "NS", "SIG"),
  RMSEA_CI = FALSE,
  SE = TRUE,
  matrixAddresses = FALSE,
  ...
)
```

Arguments

model	an OpenMx::mxModel() with OpenMx::mxFitFunctionGLM()
refModels	ignored (no GLM saturated model yet)
std	ignored (the box is eta; standardization is not defined)
digits	rounding
report	"markdown" or "html"
means	unused (kept for umxSummary signature)
residuals	unused
uncertainty	"SE" (default) or "none"
filter	unused
RMSEA_CI	unused
SE	deprecated; use uncertainty
matrixAddresses	unused
...	unused

Value

parameter table (invisibly)

---

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

Arguments

model	an OpenMx::mxModel() to summarize.
digits	round to how many digits (default = 2). 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 (TRUE).
report	If "html", then open an html table of the results.
file	The name of the dot file for figure: "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).
zero.print	How to show zeros (".")
...	Other parameters to control model summary.

Details

See documentation for other umx models here: umxSummary().

Value

- optional OpenMx::mxModel()

References

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

See Also

- umxACE(), plot.MxModelACE(), umxModify()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

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

## End(Not run)
```

---

umxSummaryACEcov *Present results of a twin ACE-model with covariates in table and graphical forms.*

Description

Summarize a Cholesky model with random-effects covariates, as returned by umxACEcov()

Usage

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

```
  extended = FALSE,
  ...
)
```

Arguments

model	A umxACEcov() model to summarize
digits	Round to how many digits (default = 2)
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.
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)
...	Other parameters to control model summary

Value

- optional OpenMx::mxModel()

References

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

See Also

- umxACEcov()

Other Summary functions: umxSummary.MxModel(), umxSummaryCP(), umxSummaryGxE(), umxSummaryIP(), umxSummaryMRDoC()

Examples

```
## Not run:
require(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACEcov(selDVs = c("bmi", "wt"), selCovs = "ht", dzData = dzData, mzData = mzData, sep="")
umxSummaryACEcov(m1, file = NA)
umxSummaryACEcov(m1, file = "name", std = TRUE)
stdFit = umxSummary(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,
  comparison = NULL,
  std = TRUE,
  showRg = FALSE,
  CIs = TRUE,
  report = c("markdown", "html"),
  file = getOption("umx_auto_plot"),
  returnStd = FALSE,
  extended = FALSE,
  zero.print = ".",
  show = c("std", "raw"),
  ...
)
```

Arguments

model	an OpenMx::mxModel() to summarize
digits	round to how many digits (default = 2)
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
file	The name of the dot file to write: "name" = use the name of the model. Defaults to getOption("umx_auto_plot"), which is likely "name".
returnStd	Whether to return the standardized form of the model (default = FALSE)
extended	how much to report (FALSE)
zero.print	How to show zeros (".")
show	Here to support being called from generic xmu_safe_run_summary. User should ignore: can be c("std", "raw")
...	Other parameters to control model summary

Details

See documentation for other umx models here: umxSummary().

Value

- optional OpenMx::mxModel()

References

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

See Also

- umxACEv()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
require(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "DZFF")
m1 = umxACEv(selDVs = "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)
```

---

umxSummaryACE_DE *Present results of a double-entry twin ACE model (umxACE_DE)*

Description

Summarize a double-entry censored twin ACE model as returned by umxACE_DE(). Automatically filters out redundant _cont rows and fixed-zero factor columns from parameter tables, while displaying all means (_cont mean and _cens threshold).

Usage

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

Arguments

model	A fitted OpenMx::mxModel() of class MxModelACE_DE to summarize.
digits	How many decimals to print (default = 2).
comparison	Optional comparison model (default = NULL).
std	Whether to standardize parameter estimates (default = TRUE).
showRg	Whether to show genetic correlations (default = FALSE).
CIs	Whether to compute and report confidence intervals (default = TRUE).
report	Format to report tables: "markdown" or "html" (default = "markdown").
file	The name of the dot file to write (default = getOption("umx_auto_plot")).
returnStd	Whether to return standardized model (default = FALSE).
extended	Whether to print raw path estimates alongside standardized (default = FALSE).
zero.print	Character to print for zeroes (default = ".").
...	Additional arguments.

Value

- optionally returns parameter estimates table dataframe.

References

https://github.com/tbates/umx

See Also

- umxACE_DE(), umxPlotACE_DE(), umxSummary()

Other Reporting functions: RMSEA(), RMSEA.MxModel(), RMSEA.summary.mxmodel(), extractAIC.MxModel(), loadings(), loadings.MxModel(), residuals.MxModel(), tmx_show(), tmx_show.MxMatrix(), umxCI(), umxCI_boot(), umxConfint(), umxExpCov(), umxExpMeans(), umxFitIndices(), umxRotate()

umxSummaryACE_GLM *Summary for a GLM ACE twin model (Cholesky)*

Description

Path loadings a, c, e on the eta scale, intercepts, -2LL / AIC. No CFI. Extra-Gaussian e paths stay in the model (residual unique covariance). For binomial items the e Cholesky diagonal is fixed at 1 (scale). Bernoulli leftover is **not** that path (e11 = leftover would double-count). Leftover is added only in the E used for shares: `E = ee' + D`, with D = 1 (probit) or $\pi^2/3$ (logit). The printed e matrix is the Cholesky of that `E`. Poisson has `D = 0`. Standardized values divide each row by that phenotype's eta-scale sd (sqrt of diag(A+C+E) with leftover in E). For two traits, prints `rA`, `rC`, and `rE = cov2cor(E_total)` so unique correlation is diluted by leftover on the diagonal (genetic `rA` is not).

Usage

```
umxSummaryACE_GLM(
  model,
  digits = 2,
  comparison = NULL,
  std = TRUE,
  report = c("markdown", "html"),
  ...
)
```

Arguments

model	a model from umxACE_GLM()
digits	rounding
comparison	optional comparison model for umxCompare()
std	if TRUE, report eta-scale variance shares
report	"markdown" or "html"
...	unused

Value

parameter table (invisibly)

See Also

umxACE_GLM()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

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

```
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)
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 OpenMx::mxModel()

References

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

See Also

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

Other Summary functions: umxSummary.MxModel(), umxSummaryACEcov(), umxSummaryGxE(), umxSummaryIP(), umxSummaryMRDoC()

Examples

```
## Not run:
require(umx)
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")

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 =
# =================
umxSummary(m1, file = "Figure 3", std = TRUE)

# =================
# = Confint example =
# =================
m1 = umxConfint(m1, "smart", run = FALSE);
m1 = umxConfint(m1, "smart", run = TRUE);
umxSummary(m1, CIs = TRUE, file = NA);

## End(Not run)
```

---

umxSummaryDoC *Shows a compact, publication-style, summary of a umx Direction of Causation model*

Description

Summarize a fitted model returned by umxDoC(). 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

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

Arguments

model	a fitted umxDoC() model to summarize.
digits	round to how many digits (default = 2).
comparison	Run mxCompare on a comparison model (default 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	Print tables to the console (as 'markdown'), or open in browser ('html')
file	The name of the dot file to write: "name" = use the name of the model. Defaults to NA = do not create plot output.
returnStd	Whether to return the standardized form of the model (default = FALSE).
zero.print	How to show zeros (".")
...	Other parameters to control model summary.

Details

See documentation for other umx models here: umxSummary().

Value

- optional OpenMx::mxModel()

See Also

- umxDoC(), plot.MxModelDoC(), umxModify(), umxCP(), plot(), umxSummary()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# ================
# = 1. Load Data =
# ================
data(docData)
mzData = subset(docData, zygosity %in% c("MZFF", "MZMM"))
dzData = subset(docData, zygosity %in% c("DZFF", "DZMM"))

# ======================================
# = 2. Define manifests for var 1 and 2 =
# ======================================
var1 = paste0("varA", 1:3)
var2 = paste0("varB", 1:3)

# ========================================================
# = 2. Make the non-causal (Cholesky) and causal models =
# ========================================================
Chol= umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= FALSE)
DoC = umxDoC(var1= var1, var2= var2, mzData= mzData, dzData= dzData, causal= TRUE)

# ================================================
# = Make the directional models by modifying DoC =
# ================================================
A2B = umxModify(DoC, "a2b", free = TRUE, name = "A2B")
A2B = umxModify(DoC, "a2b", free = TRUE, name = "A2B", comp=TRUE)
B2A = umxModify(DoC, "b2a", free = TRUE, name = "B2A", comp=TRUE)
umxCompare(B2A, A2B)


## End(Not run)
```

---

umxSummaryGxE *Summarize a GxE model*

Description

Summarize a genetic moderation model, as returned by umxGxE(). Prints graphs of A, C, and E, standardized and raw.

Usage

```
umxSummaryGxE(
  model = NULL,
  digits = 2,
  xlab = NA,
  location = "topleft",
  separateGraphs = FALSE,
  gg = TRUE,
```

```
  file = getOption("umx_auto_plot"),
  returnStd = NULL,
  std = NULL,
  reduce = FALSE,
  CIs = NULL,
  report = c("markdown", "html"),
  show = NULL,
  ...
)
```

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	If TRUE, both std and raw plots in one figure (default FALSE)
gg	Whether to use ggplot to create the graphs (default TRUE)
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	"markdown" or "html" = open a browser for copyable tables
show	not doing anything yet (required for all summary functions)
...	Optional additional parameters

Details

Note: see also umxReduce() which knows how to reduce a GxE model.

Value

- optional OpenMx::mxModel()

References

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

See Also

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

[umxSummary)]: R:umxSummary)

Other Summary functions: umxSummary.MxModel(), umxSummaryACEcov(), umxSummaryCP(), umxSummaryIP(), umxSummaryMRDoC()

Examples

```
## Not run:
# 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
selDVs  = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDVs, 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]]),]
m1 = umxGxE(selDVs = "bmi", selDefs = "age", sep="", dzData = dzData, mzData = mzData)
# Plot Moderation
umxSummaryGxE(m1)
umxSummaryGxE(m1, location = "topright")
umxSummaryGxE(m1, separateGraphs = FALSE)

## End(Not run)
```

---

umxSummaryGxEbiv *Summarize a bivariate GxE twin model*

Description

umxSummaryGxEbiv summarizes a bivariate moderation model, as returned by umxGxEbiv().

Usage

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

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
comparison	mxCompare model with this model if offered up (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)
returnStd	Whether to return the standardized form of the model (default = FALSE)
...	Optional additional parameters

Value

- optional OpenMx::mxModel()

References

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

See Also

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

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
data(twinData)
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 = "wt", selDefs = "ht",
 dzData = dzData, mzData = mzData, sep = "", dropMissingDef = TRUE)
# Plot Moderation
umxSummary(m1)
```

```
umxSummary(m1, location = "topright")
umxSummary(m1, separateGraphs = FALSE)

## 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"),
  std = TRUE,
  showRg = FALSE,
  comparison = NULL,
  CIs = FALSE,
  returnStd = FALSE,
  report = c("markdown", "html"),
  ...
)
```

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
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)
returnStd	Whether to return the standardized form of the model (default = FALSE)
report	how to display the results ("html" will open in browser as table)
...	Optional additional parameters

Value

- optional OpenMx::mxModel()

References

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

See Also

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

Other Summary functions: umxSummary.MxModel(), umxSummaryACEcov(), umxSummaryCP(), umxSummaryGxE(), umxSummaryMRDoC()

Examples

```
## Not run:
require(umx)
data(GFF) # family function and well-being data
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
selDVs = c("hap", "sat", "AD") # These will be expanded into "hap_T1" "hap_T2" etc.
m1 = umxIP(selDVs = selDVs, sep = "_T", dzData = dzData, mzData = mzData)
umxSummaryIP(m1)
plot(m1)
umxSummaryIP(m1, digits = 2, file = "Figure3", showRg = FALSE, CIs = TRUE);

## End(Not run)
```

---

umxSummaryMRDoC *Present the results of a Mendelian Randomization Direction of Causation Model in a table*

Description

Summarizes a MR Direction of Causation model, as returned by umxMRDoC()

Usage

```
umxSummaryMRDoC(
  model,
  digits = 2,
  std = TRUE,
  CIs = FALSE,
  comparison = NULL,
  RMSEA_CI = FALSE,
  report = c("markdown", "html"),
  file = getOption("umx_auto_plot"),
  ...
)
```

Arguments

model	A fitted umxDoC() 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)
CIs	Confidence intervals (default FALSE)
comparison	Run mxCompare on a comparison model (default NULL)
RMSEA_CI	Optionally compute CI on RMSEA.
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
...	Optional additional parameters

Value

- nothing

See Also

- umxDoC(), plot(), umxSummary() work for DoC models.

Other Summary functions: umxSummary.MxModel(), umxSummaryACEcov(), umxSummaryCP(), umxSummaryGxE(), umxSummaryIP()

---

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

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

```
  report = c("markdown", "html"),
  extended = FALSE,
  zero.print = ".",
  show = c("std", "raw"),
  returnStd = FALSE,
  ...
)
```

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
extended	how much to report (FALSE)
zero.print	How to show zeros (".")
show	Here to support being called from generic xmu_safe_run_summary. User should ignore: can be c("std", "raw")
returnStd	Whether to return the standardized form of the model (default = FALSE)
...	Other parameters to control model summary

Details

See documentation for summary functions for other types of umx model here: umxSummary().

Value

- optional OpenMx::mxModel()

References

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

See Also

- umxSexLim(), umxPlotSexLim()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySimplex(), umxTwinMaker(), umx_make_double_entry_data()

Examples

```
## Not run:
# ======================================================
# = Beta: Should be good to use for Boulder/March 2020 =
# ======================================================

# ============================================
# = Run Qualitative Sex Differences ACE model =
# ============================================

# =========================
# = Load and Process Data =
# =========================
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

# 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')
m1 = umxSexLim(selDVs = selDVs, sep = "_T", A_or_C = "A", tryHard = "yes",
 mzmData = mzmData, dzmData = dzmData,
 mzfData = mzfData, dzfData = dzfData,
 dzoData = dzoData
)
umxSummary(m1, file = NA);

# ===============
# = Switch to C =
# ===============
m1 = umxSexLim(selDVs = selDVs, sep = "_T", A_or_C = "C", tryHard = "yes",
 mzmData = mzmData, dzmData = dzmData,
 mzfData = mzfData, dzfData = dzfData,
 dzoData = dzoData
)
```

```
## End(Not run)
```

---

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 = ".",
  show = c("std", "raw"),
  ...
)
```

Arguments

model	an OpenMx::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 = ".")

show	Here to support being called from generic xmu_safe_run_summary. User should ignore: can be c("std", "raw")
...	Other parameters to control model summary

Details

See documentation for other umx models here: umxSummary().

Value

- optional OpenMx::mxModel()

References

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

See Also

- umxSimplex()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxTwinMaker(), umx_make_double_entry_data()

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(selDVs= vars, sep= "_T", dzData= dzData, mzData= mzData, tryHard= "yes")
umxSummary(m1, file = NA);

## End(Not run)
```

---

umxSuperModel *Make a multi-group (super) model*

Description

umxSuperModel takes 1 or more models and wraps them in a supermodel with a OpenMx::mxFitFunctionMultigroup() fit function that minimizes the sum of the fits of the sub-models.

note: Any duplicate model-names are renamed to be unique by suffixing _1 etc.

Usage

```
umxSuperModel(
  name = "super",
  ...,
  autoRun = getOption("umx_auto_run"),
  tryHard = c("no", "yes", "ordinal", "search"),
  std = FALSE
)
```

Arguments

name	The name for the container model (default = 'super')
...	Models forming the multiple groups contained in the supermodel.
autoRun	Whether to run the model (default), or just to create it and return without running.
tryHard	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
std	Show standardized parameters, raw (default), or just the fit indices (null)

Value

OpenMx::mxModel()

References

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

See Also

OpenMx::mxFitFunctionMultigroup(), umxRAM()

Other Core Model Building Functions: umx, umxLISREL(), umxMatrix(), umxModify(), umxPath(), umxRAM(), umxRAM_DE()

Examples

```
## Not run:
library(umx)
# Create two sets of data in which X & Y correlate ~ .4 in both datasets.
manifests = c("x", "y")
tmp = umx_make_TwinData(nMZpairs = 100, nDZpairs = 150,
  AA = 0, CC = .4, EE = .6, varNames = manifests)

# Group 1
grp1   = tmp[tmp$zygosity == "MZ", manifests]
g1Data = mxData(cov(grp1), type = "cov", numObs = nrow(grp1), means=umx_means(grp1))

# Group 2
grp2   = tmp[tmp$zygosity == "DZ", manifests]
g2Data = mxData(cov(grp2), type = "cov", numObs = nrow(grp2), means=umx_means(grp2))
```

```
# Model 1 (could add autoRun = FALSE if you don't want to run this as it is being built)
m1 = umxRAM("m1", data = g1Data,
 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 = g2Data,
 umxPath("x", to = "y", labels = "beta"),
 umxPath(var = manifests, labels=c("Var_x", "Resid_y_grp2")),
 umxPath(means = manifests, labels=c("Mean_x", "Mean_y"))
)

# Place m1 and m2 into a supermodel, and autoRun it
# NOTE: umxSummary is only semi-smart/certain enough to compute saturated models etc
# and report multiple groups correctly.

m3 = umxSuperModel('top', m1, m2)

umxSummary(m3, std= TRUE)

# |name         | Std.Estimate| Std.SE|CI                |
# |:------------|------------:|------:|:-----------------|
# |beta         |         0.51|   0.05|0.51 [0.41, 0.61] |
# |Var_x        |         1.00|   0.00|1 [1, 1]          |
# |Resid_y_grp1 |         0.74|   0.05|0.74 [0.64, 0.84] |
# |beta         |         0.50|   0.05|0.5 [0.41, 0.6]   |
# |Var_x        |         1.00|   0.00|1 [1, 1]          |
# |Resid_y_grp2 |         0.75|   0.05|0.75 [0.65, 0.84] |

summary(m3)

# ====================================
# = Test models with duplicate names =
# ====================================
data(GFF)
mzData = subset(GFF, zyg_2grp == "MZ")
dzData = subset(GFF, zyg_2grp == "DZ")
selDVs = c("gff", "fc", "qol")
m1 = umxCP(selDVs= selDVs, nFac= 1, dzData= dzData, mzData= mzData, sep= "_T", autoRun= TRUE)
m2 = mxRename(m1, "CP2")
umxModelNames(m1) # "top" "MZ" "DZ"
umxModelNames(m2) # "top" "MZ" "DZ"
super = umxSuperModel("myModel", m1, m2, autoRun = TRUE)
umxModelNames(super)

## End(Not run)
```

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 set set of an algebra and matrices. Big time saver!

Usage

```
umxThresholdMatrix(
  df,
  fullVarNames = NULL,
  sep = NULL,
  method = c("Mehta", "allFree"),
  threshMatName = "threshMat",
  l_u_bound = c(NA, NA),
  droplevels = FALSE,
  verbose = FALSE,
  selDVs = "deprecated"
)
```

Arguments

df	The data being modeled (to allow access to the factor levels and quantiles within these for each variable)
fullVarNames	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 our from their numeric suffixes.
method	How to implement the thresholds: Mehta, (1 free thresh for binary, first two fixed for ordinal) or "allFree"
threshMatName	name of the matrix which is returned. Defaults to "threshMat" - best not to change it.
l_u_bound	c(NA, NA) by default, you can use this to bound the first (base) threshold.
droplevels	Whether to drop levels with no observed data (defaults to FALSE)
verbose	How much to say about what was done. (defaults to FALSE)
selDVs	deprecated. Use "fullVarNames"

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.

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 ones (`lowerOnes_for_thresh`)

Twin Data

With twin data, make sure to provide the **full names** for twin data... this is not standard I know...

For twins (the function currently handles only pairs), the thresholds are equated for both twins using labels:

$labels

```
  obese_T1          obese_T2
```

dev_1 "obese_dev1" "obese_dev1"

Value

- list of thresholds matrix, deviations, lowerOnes

References

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

See Also

`OpenMx::mxThreshold()`

Other Advanced Model Building Functions: `umx`, `umxAlgebra()`, `umxFixAll()`, `umxJiggle()`, `umxRAM_GLM()`, `umxRun()`, `umxUnexplainedCausalNexus()`, `xmuLabel()`, `xmuValues()`

Examples

```
# ============================
# = Simple non-twin examples =
# ============================

# data: 1 2-level ordered factor
x = data.frame(ordered(rbinom(100,1,.5))); names(x) = c("x")

tmp = umxThresholdMatrix(x, fullVarNames = "x")
# The lower ones matrix (all fixed)
tmp[[1]]$values
tmp[[1]]$free
```

```
# The deviations matrix
tmp[[2]]$values
tmp[[2]]$labels # note: for twins, labels will be equated across twins

# The algebra that adds the deviations to create thresholds:
tmp[[3]]$formula

# Example of a warning to not omit the variable names
# tmp = umxThresholdMatrix(x)
# Polite message: For coding safety, when calling umxThresholdMatrix, set fullVarNames...

# 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, fullVarNames = "x")
tmp[[2]]$name
tmp[[2]]$free # last one is free.. (method = Mehta)

tmp = umxThresholdMatrix(x, fullVarNames = "x", l_u_bound= c(-1,1))
tmp[[2]]$lbound # bounds applied to base threshold

# ================================
# = Binary example with twin data =
# ================================
# ================================================================
# = Create a series of binary and ordinal columns to work with =
# ================================================================
data(twinData)

# Make "obese" variable with ~20% subjects categorised as obese
obesityLevels   = c('normal', 'obese')
cutPoints       = quantile(twinData[, "bmi1"], probs = .2, na.rm = TRUE)
twinData$obese1 = cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obese2 = cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
# Step 2: Make the ordinal variables into umxFactors (ordered, with the levels found in the data)
selVars = c("obese1", "obese2")
twinData[, selVars] = umxFactor(twinData[, selVars])

# Example 1
# use verbose = TRUE to see informative messages
tmp = umxThresholdMatrix(twinData, fullVarNames = selVars, sep = "", verbose = TRUE)


# ====================================
# = Ordinal (n categories > 2) example =
# ====================================
# Repeat for three-level weight variable
obesityLevels = c('normal', 'overweight', 'obese')
cutPoints = quantile(twinData[, "bmi1"], probs = c(.4, .7), na.rm = TRUE)
twinData$obeseTri1 = cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obeseTri2 = cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
selDVs = "obeseTri"; selVars = tvars(selDVs, sep = "", suffixes = 1:2)
```

```
twinData[, selVars] = umxFactor(twinData[, selVars])
tmp = umxThresholdMatrix(twinData, fullVarNames = selVars, sep = "", verbose = TRUE)


# ========================================================
# = Mix of all three kinds example (and a 4-level trait) =
# ========================================================
obesityLevels = c('underWeight', 'normal', 'overweight', 'obese')
cutPoints = quantile(twinData[, "bmi1"], probs = c(.25, .4, .7), na.rm = TRUE)
twinData$obeseQuad1 = cut(twinData$bmi1, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
twinData$obeseQuad2 = cut(twinData$bmi2, breaks = c(-Inf, cutPoints, Inf), labels = obesityLevels)
selVars = c("obeseQuad1", "obeseQuad2")
twinData[, selVars] = umxFactor(twinData[, selVars])

selDVs =c("bmi", "obese", "obeseTri", "obeseQuad")
tmp = umxThresholdMatrix(twinData, fullVarNames = tvars(selDVs, sep= ""), sep = "", verbose = TRUE)
# The lower ones matrix (all fixed)
tmp[[1]]$values
# The deviations matrix
tmp[[2]]$values
tmp[[2]]$labels # note labels are equated across twins
# Check to be sure twin-1 column labels same as twin-2
tmp[[2]]$labels[,2]==tmp[[2]]$labels[,4]

# The algebra that assembles these into thresholds:
tmp[[3]]$formula
# ================================
# = Example with method = allFree =
# ================================

tmp = umxThresholdMatrix(twinData, fullVarNames = tvars(selDVs, sep= ""), sep = "",
 method = "allFree")
all(tmp[[2]]$free)
```

---

umxTwinMaker *Make a twin model from a model describing just one person*

Description

umxTwinMaker takes a collection of paths describing the model for 1 person and returns a completed twin model. This consists of a umxSuperModel() containing MZ and DZ umxRAM() models.

Pass into umxTwinMaker:

1. A list of paths making up the twin 1 model
2. t1_t2links, a vector describing the component relationships connecting twin 1 to twin 2 (The default works in most cases: 1 and .5 for a for MZ and DZ respectively, 1 for c and 0 for e.

Details

Some rules. All labels are expanded with a twin suffix: so "var1" -> "var1_T1" etc. You provide the person-model using *just the base name* `umxTwinMaker()` knows how to expand it using the separator string).

Second, the latent variables must match the base names given in t1_t2links (`a`, `c`, and `e`). To avoid clashes, manifests must *not* match variables in `t1_t2links` - by default names like "a1" are reserved for ace.

Usage

```
umxTwinMaker(
  name = "m1",
  paths,
  t1_t2links = list(a = c(1, 0.5), c = c(1, 1), e = c(0, 0)),
  mzData = NULL,
  dzData = NULL,
  sep = "_T",
  autoRun = getOption("umx_auto_run")
)
```

Arguments

`name`	The name for the resulting `umxSuperModel()` (Default "m1").
`paths`	A vector of `umxPath()`s describing one person.
`t1_t2links`	base name (and values) of paths that covary between T1 and T2. Default: c('a'=c(1,.5), 'c'=c(1,1), 'e'=c(0,0))
`mzData`	Data for MZ twins.
`dzData`	Data for DZ twins.
`sep`	The separator used to create twin 1 and 2 names (Default "_T")
`autoRun`	Whether to run the supermodel before returning it.

Value

- `umxSuperModel()`

References

tutorials, github

See Also

- `umxRAM()`, `umxSuperModel()`, `umxPath()`

Other Twin Modeling Functions: `power.ACE.test()`, `umx`, `umxACE()`, `umxACE_DE()`, `umxACE_GLM()`, `umxACEcov()`, `umxACEv()`, `umxCP()`, `umxCloud()`, `umxDiffMZ()`, `umxDiscTwin()`, `umxDoC()`, `umxDoCp()`, `umxGxE()`, `umxGxE_window()`, `umxGxEbiv()`, `umxIP()`, `umxMRDoC()`, `umxReduce()`, `umxReduceACE()`, `umxReduceGxE()`, `umxRotate.MxModelCP()`, `umxSexLim()`, `umxSimplex()`, `umxSummarizeTwinData()`, `umxSummaryACE()`, `umxSummaryACE_GLM()`, `umxSummaryACEv()`, `umxSummaryDoC()`, `umxSummaryGxEbiv()`, `umxSummarySexLim()`, `umxSummarySimplex()`, `umx_make_double_entry_data()`

Examples

```
## Not run:
# We'll make some ACE models, but first, let's clean up the twinData
# set for analysis
# 1. Add a separator to the twin variable names (with sep = "_T")
# 2. Scale the data so it's easier for the optimizer.
data(twinData)
tmp = umx_make_twin_data_nice(data=twinData, sep="", zygosity="zygosity", numbering=1:2)
tmp = umx_scale_wide_twin_data(varsToScale= c("wt", "ht"), sep= "_T", data= tmp)
mzData = subset(tmp, zygosity %in%  c("MZFF", "MZMM"))
dzData = subset(tmp, zygosity %in%  c("DZFF", "DZMM"))

# ==========================
# = Make an ACE twin model =
# ==========================
# 1. Define paths for *one* person:
paths = c(
   umxPath(v1m0 = c("a1", 'c1', "e1")),
   umxPath(means = c("wt")),
   umxPath(c("a1", 'c1', "e1"), to = "wt", values=.2)
)
# 2. Make a twin model from the paths for one person
m1 = umxTwinMaker("test", paths, mzData = mzData, dzData= dzData)
plot(m1, std= TRUE, means= FALSE)

# 3. comparison with umxACE...
m2 = umxACE(selDVs="wt", mzData = mzData, dzData=dzData, sep="_T")

# =====================
# = Bivariate example =
# =====================
latents = paste0(rep(c("a", "c", "e"), each = 2), 1:2)
biv = c(
umxPath(v1m0 = latents),
umxPath(mean = c("wt", "ht")),
umxPath(fromEach = c("a1", 'c1', "e1"), to = c("ht", "wt")),
umxPath(c("a2", 'c2', "e2"), to = "wt")
)
tmp= umxTwinMaker(paths= biv, mzData = mzData, dzData= dzData)
plot(tmp, means=FALSE)

# How to use latents other than a, c, and e: define in t1_t2links
paths = c(
umxPath(v1m0 = c("as1", 'c1', "e1")),
umxPath(means = c("wt")),
umxPath(c("as1", 'c1', "e1"), to = "wt", values=.2)
)
m1 = umxTwinMaker("test", paths, mzData = mzData, dzData= dzData,
 t1_t2links = list('as'=c(1, .5), 'c'=c(1, 1), 'e'=c(0, 0))
)
```

```
## End(Not run)
```

---

umxTwoStage *Build a SEM implementing the instrumental variable design*

Description

umxMR (umxTwoStage) implements a Mendelian randomization or instrumental variable Structural Equation Model. For ease of learning, the parameters follow the tsls() function in the sem package.

Usage

```
umxTwoStage(
  formula = Y ~ X,
  instruments = ~qtl,
  data,
  std = FALSE,
  subset,
  contrasts = NULL,
  name = "IV_model",
  tryHard = c("no", "yes", "ordinal", "search"),
  ...
)

umxMR(
  formula = Y ~ X,
  instruments = ~qtl,
  data,
  std = FALSE,
  subset,
  contrasts = NULL,
  name = "IV_model",
  tryHard = c("no", "yes", "ordinal", "search"),
  ...
)
```

Arguments

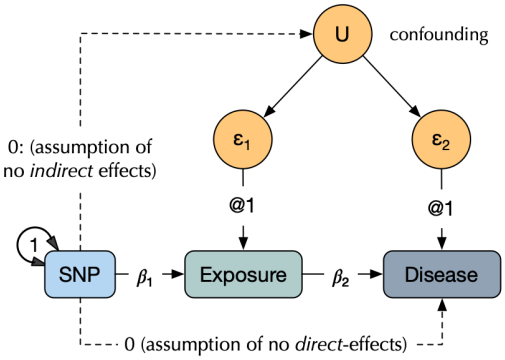
formula	The structural equation to be estimated (default = Y ~ X). A constant is implied if not explicitly deleted.
instruments	A one-sided formula specifying instrumental variables (default = qtl).
data	Frame containing the variables in the model.
std	Standardize the manifests before running model (default is FALSE)

`subset`	(optional) vector specifying a subset of observations to be used in fitting the model.
`contrasts`	An optional list (not supported)
`name`	The model name (default is "IVmodel")
`tryHard`	Default ('no') uses normal mxRun. "yes" uses mxTryHard. Other options: "ordinal", "search"
`...`	arguments to be passed along. (not supported)

Details

The example is a Mendelian Randomization analysis showing the utility of SEM over two-stage regression.

The following figure shows how the MR model appears as a path diagram:



Value

- `OpenMx::mxModel()`

References

- Fox, J. (1979) Simultaneous equation models and two-stage least-squares. In Schuessler, K. F. (ed.) *Sociological Methodology*, Jossey-Bass.
- Greene, W. H. (1993) *Econometric Analysis*, Second Edition, Macmillan.
- Sekula, P., Del Greco, M. F., Pattaro, C., & Kottgen, A. (2016). Mendelian Randomization as an Approach to Assess Causality Using Observational Data. *Journal of the American Society of Nephrology*, **27**), 3253-3265. doi:10.1681/ASN.2016010098

See Also

- `umx_make_MR_data()`, `umxDiffMZ()`, `umxDoC()`, `umxDiscTwin()`

Other Super-easy helpers: `umx`, `umxEFA()`

Examples

```
## Not run:
# ====================================
# = Mendelian Randomization analysis =
# ====================================

library(umx)
df = umx_make_MR_data(10e4, Vqtl = 0.02, bXY = 0.1, bUX = 0.5, bUY = 0.5, pQTL = 0.5)
m1 = umxMR(Y ~ X, instruments = ~ qtl, data = df)
parameters(m1)
plot(m1, means = FALSE, min="") # help DiagrammR layout the plot.
m2 = umxModify(m1, "qtl_to_X", comparison=TRUE, tryHard="yes", name="QTL_affects_X") # yip
m3 = umxModify(m1, "X_to_Y"  , comparison=TRUE, tryHard="yes", name="X_affects_Y") # yip
plot(m3, means = FALSE)

# Errant analysis using ordinary least squares regression (WARNING this result is CONFOUNDED!!)
ols1 = lm(Y ~ X    , data = df); coef(ols1) # Inflated .35 effect of X on Y
ols2 = lm(Y ~ X + U, data = df); coef(ols2) # Controlling U reveals the true 0.1 beta weight

# Simulate date with no causal X -> Y effect.
df = umx_make_MR_data(10e4, Vqtl = 0.02, bXY = 0, bUX = 0.5, bUY = 0.5, pQTL = 0.5)
m1 = umxMR(Y ~ X, instruments = ~ qtl, data = df)
parameters(m1)

# ======================
# = Now with sem::tsls =
# ======================
# libs("sem")
m2 = sem::tsls(formula = Y ~ X, instruments = ~ qtl, data = df)
coef(m2)

# Try with missing value for one subject: A benefit of the FIML approach in OpenMx.
m3 = tsls(formula = Y ~ X, instruments = ~ qtl, data = (df[1, "qtl"] = NA))

## End(Not run)
```

---

umxUnexplainedCausalNexus

umxUnexplainedCausalNexus

Description

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

Usage

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

Arguments

from	A variable in the model for which you want to compute the effect of a change.
delta	A the amount to simulate changing 'from' by.
to	The dependent variable that you want to watch changing.
model	The model containing variables from and to.

References

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

See Also

- OpenMx::mxCheckIdentification(), umxCompare()

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), xmuLabel(), xmuValues()

Examples

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

## End(Not run)
```

---

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 = c("umx_vers", "OpenMx_vers")
)
```

Arguments

model	Optional to show optimizer in this model
min	Optional minimum version to test for. Prefer a string such as "4.70.0". A lone number like 3 is accepted (treated as "3") with a note to use a string next time. Default NULL (no check).
verbose	= TRUE
return	Which package (umx or OpenMx) to 'return' version info for (Default = umx).

Value

- OpenMx::mxModel()

References

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

See Also

- packageVersion(), install.OpenMx()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
x = umxVersion(); x
## Not run:
umxVersion(min = "4.70.0")

## End(Not run)
```

---

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

Wagenmakers E.J., Farrell S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin and Review*. **11**, 192-196. doi:10.3758/bf03206482.

See Also

- AIC()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
l1 = lm(mpg ~ wt + disp, data = mtcars)
l2 = lm(mpg ~ wt, data = mtcars)
umxWeightedAIC(models = list(l1, l2))
```

---

umx_aggregate *Convenient formula-based cross-tabs & built-in summary functions*

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).

```
umx_aggregate(mpg ~ cyl, data = mtcars, what = "mean_sd")
```

cyl	mpg
4 (n = 11)	26.66 (4.51)
6 (n = 7)	19.74 (1.45)
8 (n = 14)	15.1 (2.56)

Usage

```
umx_aggregate(
  formula = DV ~ condition,
  data = df,
  what = c("mean_sd", "n"),
  digits = 2,
  report = c("markdown", "html", "txt"),
  output = c("table", "string")
)
```

Arguments

formula	The aggregation formula. e.g., DV ~ condition.
data	frame to aggregate (defaults to df for common case)
what	function to use. Default reports "mean (sd)".
digits	to round results to.
report	Format for the table: Default is markdown.
output	Format for the results: "table" (default) or "string".

Value

- table or character vector of formatted strings

References

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

See Also

- umx_apply(), aggregate()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_time()

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")
umx_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 (Table 1 style)=
# ========================================================
umx_aggregate(cbind(mpg, qsec) ~ cyl, data = mtcars, digits = 3)

# ========================
# = Output as a string    =
# ========================
umx_aggregate(mpg ~ cyl, data = mtcars, output = "string")

## 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 umxAPA(), which 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.

Usage

```
umx_APA_pval(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

- umxAPA(), round()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

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

---

umx_apply *umx_apply*

Description

Tries to make apply() more readable. so "mean of x by columns", instead of "x, 2, mean" Other functions you might use in FUN include cumsum(), base::rowSums(), base::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.

Value

- object

References

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

See Also

- umx_aggregate()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
umx_apply(mean, mtcars, by = "columns")
umx_apply("mean", of = mtcars, by = "columns")
tmp = mtcars[1:3,]; tmp[1,1] = NA
umx_apply("mean", by = "rows", of = tmp)
umx_apply("mean", by = "rows", of = tmp, 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(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
x = c("Alice", "Bob", "Carol")
umx_array_shift(x) # returns "Alice"
x # now only 2 items (altered in containing environment)
```

---

umx_as_numeric *umx_as_numeric*

Description

Apply as.numeric to multiple columns of a dataframe.

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://github.com/tbates/umx

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
# make mpg into string, and cyl into a factor
df = mtcars
df$mpg = as.character(df$mpg)
df$cyl = factor(df$cyl)
df$am = df$am==1
df = umx_as_numeric(df); str(df) # mpg not touched
df = umx_as_numeric(df, force=TRUE); str(df) # mpg coerced back to numeric
## Not run:
# coercing a real string will cause NAs
df$mpg = c(letters[1:16]); str(df) # replace mpg with letters.
df = umx_as_numeric(df, force=TRUE); str(df)

## End(Not run)
```

---

umx_check *umx_check*

Description

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

Usage

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

Arguments

boolean.test	test evaluating to TRUE or FALSE.
action	One of "stop" (the default), "warning", or "message".
message	what to tell the user when boolean.test is FALSE.
...	extra text will be pasted after the messages.

Value

- boolean

See Also

Other Test: umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
umx_check(length(1:3)==3, "message", "item must have length == 3", "another comment", "and another")
umx_check(1==2, "message", "one must be 2", ". Another comment", "and another")
```

---

umx_check_model *Check for required features in an OpenMx.*

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.

Usage

```
umx_check_model(
  obj,
  type = NULL,
  hasData = NULL,
  beenRun = NULL,
  hasMeans = NULL,
  checkSubmodels = FALSE,
  callingFn = "a function"
)
```

Arguments

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

Value

- boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("check_model_ex", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
umx_check_model(m1) # TRUE, this is a model
umx_check_model(m1, type = "RAM") # equivalent to umx_is_RAM()
umx_check_model(m1, hasData = TRUE)


umx_check_model(m1, hasMeans = TRUE)
umx_check_model(m1, beenRun = FALSE)
# Model with no data
m1 = umxRAM("x ~~ .3*y", autoRun = FALSE)
umx_check_model(m1, hasData = TRUE)

## End(Not run)
```

---

umx_check_names *Check if a request name exists in a dataframe or related object*

Description

Check if a list of names are in the namez() of a dataframe (or the dimnames() of a matrix), or the names of the observed data of an mxData()

Usage

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

Arguments

namesNeeded	Variable names to find (a dataframe is also allowed)
data	data.frame, matrix, or mxData to search in for names (default NA)
die	Whether to die if the check fails (default TRUE).
illegal	Optional list of names which must NOT be present.
no_others	Whether to test that the data contain no columns in addition to those in namesNeeded (default FALSE)
intersection	Show the intersection of names
message	Some helpful text to append when dying.

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Other Check or test: umx, umx_is_class(), umx_is_endogenous(), umx_is_exogenous(), umx_is_numeric(), umx_is_ordered()

Examples

```
require(umx)
data(demoOneFactor) # "x1" "x2" "x3" "x4" "x5"
umx_check_names(c("x1", "x2"), demoOneFactor)
umx_check_names(c("x1", "x2"), as.matrix(demoOneFactor))
umx_check_names(c("x1", "x2"), cov(demoOneFactor[, c("x1","x2")]))
umx_check_names(c("x1", "x2"), mxData(demoOneFactor, type="raw"))
umx_check_names(c("z1", "x2"), data = demoOneFactor, die = FALSE)
umx_check_names(c("x1", "x2"), data = demoOneFactor, die = FALSE, no_others = TRUE)
umx_check_names(c("x1","x2","x3","x4","x5"), data = demoOneFactor, die = FALSE, no_others = TRUE)
# no request
umx_check_names(c(), data = demoOneFactor, die = FALSE, no_others = TRUE)
```

```
## Not run:
# An example error from vars that don't exist in the data
umx_check_names(c("bad_var_name", "x2"), data = demoOneFactor, die = TRUE)

## End(Not run)
```

---

umx_check_OS *umx_check_OS*

Description

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

Usage

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

Arguments

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

Value

- TRUE if on the specified OS (else FALSE)

References

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

See Also

Other Test: umx_check(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
umx_check_OS()
```

`umx_check_parallel` *Check that 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, umx_detect_cores()),
  testScript = NULL,
  rowwiseParallel = TRUE,
  nSubjects = 1000,
  optimizer = NULL
)
```

Arguments

`nCores`	How many cores to run (defaults to c(1, max). -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.
`optimizer`	Set optimizer, e.g., "NPSOL")

Details

Some historical (starting 2017-09-06) speeds on my late 2015 iMac, 3.3 GHz Quad-core i7 desktop and then a quad i7 2018 MacBook Pro

Date	Version	Cores	Time	Notes
2026-07-28	2.5.6.19 (git)	5	00 min 36.00 sec	Δ: 58% (SLSQP M3 Pro laptop)
2026-07-28	2.5.6.19 (git)	1	01 min 02.00 sec	(SLSQP M3 Pro laptop)
2021-07-28	2.19.6.19 (git)	8	00 min 42.98 sec	Δ:-80 (SLSQP laptop (55 sec under NPSOL))
2021-07-28	2.19.6.19 (git)	1	02 min 03 sec	(SLSQP on laptop)
2020-08-09	2.17.3 (git)	1	01 min 52 sec	(CSOLNP on laptop)
2020-08-09	2.17.3 (git)	4	00 min 40.18 sec	(CSOLNP on laptop)
2019-06-13	v2.13.2 (git)	1	01 min, 11 sec	(NPSOL)
2019-06-13	v2.13.2 (git)	4	00 min, 22 sec	(NPSOL)
2019-06-13	v2.13.2 (git)	6	00 min, 21 sec	(NPSOL)
2018-10-14	v2.11.5 (CRAN)	4	00 min, 36 sec	Δ:-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	Δ: -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	Δ: -44.68)

2017-10-16	Clang OpenMP	1	01 min, 08.38 sec	
2017-10-16	Clang OpenMP	4	00 min, 24.89 sec	Δ: -43.49)
2017-09-07	Clang OpenMP	1	01 min, 12.90 sec	
2017-09-07	Clang OpenMP	4	00 min, 32.20 sec	Δ: -40.70
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	Δ: -45.00

Value

None

References

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

See Also

- umx_time(), umx_set_cores()

Other Miscellaneous Functions: deg2rad(), rad2deg(), umxBrownie(), umx_complete_dollar()

Examples

```
## Not run:
# In 2016 1core took 1 minute
umx_check_parallel()

## End(Not run)
```

---

umx_complete_dollar *umx_complete_dollar*

Description

Modifies the macOS RGUI (R.app) to extend auto-completion. It appends a "$" and attempts completion again when no stem matches are found. Implemented by Simon Urbanek.

Usage

```
umx_complete_dollar()
```

See Also

Other Miscellaneous Functions: deg2rad(), rad2deg(), umxBrownie(), umx_check_parallel()

umx_cont_2_quantiles *umx_cont_2_quantiles*

Description

Recode a continuous variable into n-quantiles (default = deciles (10 levels)). It returns an OpenMx::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
)

umx2ord(
  x,
  nlevels = NULL,
  type = c("mxFactor", "ordered", "unordered"),
  verbose = FALSE,
  returnCutpoints = FALSE
)
```

Arguments

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

Details

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

Value

- recoded variable as an OpenMx::mxFactor()

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
x = umx_cont_2_quantiles(rnorm(1000), nlevels = 10, verbose = TRUE)
x = data.frame(x)
str(x); levels(x)
table(x)
## Not run:
hist(x$x)
y = mxData(x, type = "raw") # then fit with mxFitFunctionWLS / umxRAM 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)
```

---

umx_cor *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

```
umx_cor(
  X,
  data = NULL,
  df = NULL,
  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 formula, or matrix or dataframe
data	dataframe (IFF X is a formula)
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 robust function called rcorr.

Value

- Matrix of correlations and p-values

See Also

umxHetCor(), cor.test()

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_means(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
tmp = myFADataRaw[1:8,1:8]
umx_cor(tmp)
tmp$x1 = letters[1:8] # make one column non-numeric
umx_cor(tmp)
umx_cor(~ x2 + x3, tmp)
```

umx_data *Show the datasets and vignettes in umx*

Description

umx_data lists the datasets and vignettes in umx with a 1-line description of each.

Usage

```
umx_data()
```

Value

None

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
umx_data()
```

umx_detect_cores *Detect number of CPU cores (including Performance cores)*

Description

Detects the number of cores on the local machine. On hybrid architectures (like Apple Silicon or Intel Alder Lake+), it prioritizes detecting only Performance (P) cores to avoid E-core bottlenecks in parallel loops.

Usage

```
umx_detect_cores(logical = FALSE, p_cores = TRUE)
```

Arguments

logical	Whether to count logical cores (hyperthreaded) or physical cores. Default is FALSE (physical).
p_cores	Whether to prioritize detecting Performance (P) cores only on hybrid architectures. Default is TRUE.

Details

A detector, not a decider: every probe is guarded, so this always returns a positive integer and never errors — worst case it falls back to `parallel::detectCores()`.

Method: on macOS, tiers are selected by NAME (`hw.perflevelN.name`), not rank: fast cores are tiers named "Super" or "Performance". Ranks shift yearly (L0 was Performance through M4, Super from M5 Pro/Max & M6 (h/t Brian Ripley, CRAN)), and M5/6 machines can have zero Efficiency cores, so rank-based rules undercount there. On Linux there is no sysfs P-core flag, so P-cores are the online cores whose max frequency is the highest on the box (true for Intel hybrid and ARM big.LITTLE; degrades to all cores on homogeneous machines).

Value

Integer count of cores.

See Also

Other Get and set: `umx`, `umx_get_checkpoint()`, `umx_get_options()`, `umx_scale_reliabilities()`, `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_dollar_symbol()`, `umx_set_optimization_options()`, `umx_set_optimizer()`, `umx_set_plot_file_suffix()`, `umx_set_plot_format()`, `umx_set_separator()`, `umx_set_silent()`, `umx_set_table_format()`

Examples

```
umx_detect_cores() # physical cores (P-cores if hybrid)
umx_detect_cores(logical = TRUE, p_cores = FALSE) # all logical cores
```

---

umx_explode *Explode a string (Like the php function* `explode`*)*

Description

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

Usage

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

Arguments

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

Value

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

References

https://tbates.github.io, https://www.php.net/manual/en/function.explode.php

See Also

Other String Functions: tvars(), umx, umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

Examples

```
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

umx_paste_names()

Other String Functions: tvars(), umx, umx_explode(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

Examples

```
## Not run:
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")
umx_explode_twin_names(c("x_T11", "x_T22"), sep = "_T")

## End(Not run)
```

---

umx_file_load_pseudo *Read in files from pseudocons.*

Description

Read in PRS scored files from pseudocons.

1. Read the file
2. Break it into pseudo and real rows
3. Clean-up by deleting the pseudo suffix
4. Rename NT vars with a suffix
5. Merge files on ID and return

	ID	FID	BMIS1	BMIS2	BMIS3	BMIS4	...
1	1234501	12345	-0.032	-0.77	-0.40	-3.87	...
2	1234501-pseudo-1	12345	0.117	-0.66	-0.33	-4.08	...

Usage

```
umx_file_load_pseudo(fn, bp, suffix = "_NT", chosenp = "S5")
```

Arguments

fn	The filename
bp	The path to the folder containing the file
suffix	to add to the NT columns (Default = "_NT")
chosenp	The suffix (pvalue) we desire to use (Default = "S5")

Value

- dataframe of real and pseudo PRS columns

See Also

Other File Functions: dl_from_dropbox(), umx, umx_make_sql_from_excel(), umx_move_file(), umx_open(), umx_rename_file(), umx_write_to_clipboard()

Examples

```
## Not run:
basepath = "~/Dropbox/2016 (1). project EA/2018/EA3/"
tmp = umx_file_load_pseudo("PRS_EA3_R9_autosomes_HRC1.1_pseudo.txt", bp = bp)
str(tmp)
head(tmp[, c("BMIS4", "BMIS4_NT")]

## End(Not run)
```

---

umx_find_object *umx_find_object*

Description

Find objects of a given 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

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

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

## End(Not run)
```

---

umx_fun_mean_sd *Summarizing functions used in umx_aggregate and for umxAPA*

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.

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

Value

- function result

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(),

xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

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

---

umx_get_bracket_addresses

Get bracket-style addresses from an mxMatrix

Description

Sometimes you want these. 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://github.com/tbates/umx

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPath, xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod, xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre, xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor, xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(, xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("get_add_ex", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)#'
```

```
umx_get_bracket_addresses(m1$matrices$A, free= TRUE)

## End(Not run)
```

---

umx_get_checkpoint *Get or set checkpointing for a model*

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

None

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
## Not run:
umx_get_checkpoint() # current global default
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
umx_get_checkpoint(model = m1)

## End(Not run)
```

umx_get_options *Display umx options*

Description

Show the umx options. Useful for beginners to discover, or people like me to remember :-)

Usage

```
umx_get_options()
```

Value

- message

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
umx_get_options()
```

umx_grep *Search for text*

Description

Search names if given a data.frame, or strings if given a vector of strings.

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

The namez function is more flexible. A handy feature of umx_grep is that it can search the labels of data imported from SPSS.

nb: To simply grep for a pattern in a string use R's built-in grep() functions, e.g.: grepl("^NA\\[0-9]", "NA.3")

Value

- list of matched column names and/or labels.

References

https://github.com/tbates/umx

See Also

- namez(), umx_aggregate(), grep()

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

Examples

```
umx_grep(mtcars, "hp", output="both", ignore.case= TRUE)
umx_grep(c("hp", "ph"), "hp")
umx_grep(mtcars, "^h.*", output="both", ignore.case= TRUE)
## Not run:
umx_grep(spss_df, "labeltext", output = "label")
umx_grep(spss_df, "labeltext", output = "name")

## End(Not run)
```

umx_has_been_run *umx_has_been_run*

Description

check if an mxModel has been run or not

Usage

```
umx_has_been_run(model, stop = FALSE)
```

Arguments

model	The OpenMx::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://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("has_been_run_example", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
umx_has_been_run(m1)

## End(Not run)
```

umx_has_CIs *umx_has_CIs*

Description

A utility function to return a binary answer to the question "does this OpenMx::mxModel() have confidence intervals?"

Usage

```
umx_has_CIs(model, check = c("both", "intervals", "output"))
```

Arguments

model	The OpenMx::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://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("_has_CI_ex", data = demoOneFactor, type = "cov",
 umxPath("g", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "g", fixedAt = 1.0)
)

umx_has_CIs(m1) # FALSE: no CIs and no output
m1 = mxModel(m1, mxCI("g_to_x1"))
umx_has_CIs(m1, check = "intervals") # TRUE intervals set
umx_has_CIs(m1, check = "output")  # FALSE not yet run
m1 = mxRun(m1)
```

```
umx_has_CIs(m1, check = "output")  # Still FALSE: Set and Run
m1 = mxRun(m1, intervals = TRUE)
umx_has_CIs(m1, check = "output")  # TRUE: Set, and Run with intervals = T
umxSummary(m1)

## End(Not run)
```

---

umx_has_means *umx_has_means*

Description

A utility function to return a binary answer to the question "does this OpenMx::mxModel() have a means model?"

Usage

```
umx_has_means(model)
```

Arguments

model	The OpenMx::mxModel() to check for presence of means

Value

- TRUE or FALSE

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("has_means_ex", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
```

```
umx_has_means(m1)
m1 = mxModel(m1,
 mxPath(from = "one", to = manifests),
 mxData(demoOneFactor[1:100,], type = "raw")
)
umx_has_means(m1)
m1 = mxRun(m1)
umx_has_means(m1)


## End(Not run)
```

`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

See Also

Other Test: `umx_check()`, `umx_check_OS()`, `umx_check_model()`, `umx_check_names()`, `umx_has_CIs()`, `umx_has_been_run()`, `umx_has_means()`, `umx_is_GSEM()`, `umx_is_LISREL()`, `umx_is_MxData()`, `umx_is_MxMatrix()`, `umx_is_MxModel()`, `umx_is_RAM()`, `umx_is_cov()`

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 %in% a list of classes. 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

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

See Also

- umx_is_numeric()

Other Check or test: umx, umx_check_names(), umx_is_endogenous(), umx_is_exogenous(), umx_is_numeric(), umx_is_ordered()

Examples

```
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")
umx_is_class(mtcars$mpg) # report class of this column (same as class(mpg))
```

umx_is_cov *umx_is_cov*

Description

test if a data frame, matrix or mxData is type cov or cor, 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://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM()

Examples

```
df = cov(mtcars)
umx_is_cov(df)
df = cor(mtcars)
umx_is_cov(df)
umx_is_cov(mxData(df[1:3,1:3], type= "cov", numObs = 200))
umx_is_cov(df, boolean = TRUE)
umx_is_cov(mtcars, boolean = TRUE)
```

umx_is_endogenous *List endogenous variables in a model*

Description

Return a list of all the endogenous variables (variables with at least one incoming single-arrow path) in a model.

Usage

```
umx_is_endogenous(model, manifests_only = TRUE)
```

Arguments

model	an OpenMx::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://github.com/tbates/umx

See Also

Other Check or test: umx, umx_check_names(), umx_is_class(), umx_is_exogenous(), umx_is_numeric(), umx_is_ordered()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
m1 = umxRAM("umx_is_endogenous", data = demoOneFactor, type = "cov",
 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)


## End(Not run)
```

umx_is_exogenous *umx_is_exogenous*

Description

Return a list of all the exogenous variables (variables with no incoming single-arrow path) in a model.

Usage

```
umx_is_exogenous(model, manifests_only = TRUE)
```

Arguments

model	an OpenMx::mxModel() from which to get exogenous variables
manifests_only	Whether to check only manifests (default = TRUE)

Value

- list of exogenous variables

References

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

See Also

Other Check or test: umx, umx_check_names(), umx_is_class(), umx_is_endogenous(), umx_is_numeric(), umx_is_ordered()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 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)


## End(Not run)
```

umx_is_GSEM *umx_is_GSEM*

Description

Utility function returning a binary answer to the question "Is this a GSEM model?"

Usage

```
umx_is_GSEM(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx GSEM OpenMx::mxModel()

Details

TRUE only for models of class MxModelGSEM (the class umxGSEM() assigns). Ordinary RAM / umxRAM models are FALSE (they used to match via a leftover MxRAMObjective check).

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
## Not run:
require(umx)
# LDSC object (list with V, S, I, N, m)
data(Psych_LDSC)

m1 = umxGSEM(model = "F1 ~= SCZ + BIP + MDD + EA + INSOM",
             covstruc = Psych_LDSC, estimation = "DWLS")
umx_is_GSEM(m1) # TRUE

# lavaan-style paths (quote the model string carefully)
m2 = umxGSEM(
  model = "F1 =~ SCZ + BIP + MDD\nINSOM ~ F1\nEA ~ INSOM",
  covstruc = Psych_LDSC, estimation = "DWLS")
```

```
umxCompare(m2, m1)

# Hand-built S/V heritability + genetic covariance
traits = c("Trt1", "Trt2")
S = matrix(c(0.25, 0.15, 0.15, 0.30), nrow = 2, ncol = 2,
           dimnames = list(traits, traits))
V = diag(c(0.002, 0.003, 0.002))
vech_names = c("Trt1 Trt1", "Trt2 Trt1", "Trt2 Trt2")
dimnames(V) = list(vech_names, vech_names)
modelStr = "Trt1 ~~ Trt1\nTrt2 ~~ Trt2\nTrt1 ~~ Trt2"
m0 = umxGSEM(model = modelStr, S = S, V = V, estimation = "DWLS")
umx_is_GSEM(m0)

## End(Not run)
```

---

umx_is_LISREL *umx_is_LISREL*

Description

Utility function returning a binary answer to the question "Is this a LISREL model?"

Usage

```
umx_is_LISREL(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx LISREL `OpenMx::mxModel()`

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: `umx_check()`, `umx_check_OS()`, `umx_check_model()`, `umx_check_names()`, `umx_has_CIs()`, `umx_has_been_run()`, `umx_has_means()`, `umx_has_square_brackets()`, `umx_is_GSEM()`, `umx_is_MxData()`, `umx_is_MxMatrix()`, `umx_is_MxModel()`, `umx_is_RAM()`, `umx_is_cov()`

Examples

```
## Not run:
# LISREL model example

## End(Not run)
```

umx_is_MxData *Check if an object is an mxData object*

Description

Is the input an MxData?

Usage

```
umx_is_MxData(x)
```

Arguments

x	An object to test for being an MxData object

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

Examples

```
umx_is_MxData(mtcars)
umx_is_MxData(mxData(mtcars, type= "raw"))
umx_is_MxData(mxData(cov(mtcars), type= "cov", numObs = 73))
umx_is_MxData(mxData(na.omit(twinData[, c("wt1", "wt2")]), type = "raw"))
```

umx_is_MxMatrix *umx_is_MxMatrix*

Description

Utility function returning a binary answer to the question "Is this an OpenMx mxMatrix?"

Usage

```
umx_is_MxMatrix(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx OpenMx::mxMatrix()

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxModel(), umx_is_RAM(), umx_is_cov()

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 OpenMx::mxModel()
listOK	Is it acceptable to pass in a list of models? (Default = FALSE)

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_RAM(), umx_is_cov()

Examples

```
m1 = mxModel("test")
if(umx_is_MxModel(m1)){
 message("nice OpenMx model!")
}
if(umx_is_MxModel(list(m1,m1), listOK = TRUE)){
 message("nice list of OpenMx models!")
}
```

---

umx_is_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://tbates.github.io

See Also

- umx_is_class()

Other Check or test: umx, umx_check_names(), umx_is_class(), umx_is_endogenous(), umx_is_exogenous(), umx_is_ordered()

Examples

```
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,
  summaryObject = FALSE
)
```

Arguments

`df`	A `data.frame()` or `OpenMx::mxData()` 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
`summaryObject`	whether to return a nice summary object. Overrides other settings (FALSE)

Value

- vector of variable names or Booleans

References

https://github.com/tbates/umx

See Also

Other Check or test: umx, umx_check_names(), umx_is_class(), umx_is_endogenous(), umx_is_exogenous(), umx_is_numeric()

Examples

```
x = data.frame(ordered(rbinom(100,1,.5))); names(x) = c("x")
umx_is_ordered(x, summaryObject= TRUE) # all ordered factors including binary
tmp = mtcars

tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$vs = ordered(mtcars$vs) # binary factor
umx_is_ordered(tmp) # true/false
umx_is_ordered(tmp, strict=FALSE)
umx_is_ordered(tmp, names = TRUE)
umx_is_ordered(tmp, names = TRUE, binary.only = TRUE)
umx_is_ordered(tmp, names = TRUE, ordinal.only = TRUE)
umx_is_ordered(tmp, names = TRUE, continuous.only = TRUE)
umx_is_ordered(tmp, continuous.only = TRUE)

x = umx_is_ordered(tmp, summaryObject= TRUE)

isContinuous = !umx_is_ordered(tmp)
## Not run:
# nb: By default, unordered factors cause a message...
tmp$gear = factor(mtcars$gear) # Unordered factor
umx_is_ordered(tmp)
umx_is_ordered(tmp, strict = FALSE) # compare: no warning

# also: not designed to work on single variables...
umx_is_ordered(tmp$cyl)
# Do this instead...
umx_is_ordered(tmp[, "cyl", drop= FALSE])

## End(Not run)
```

---

umx_is_RAM *umx_is_RAM*

Description

Utility function returning a binary answer to the question "Is this a RAM model?"

Usage

```
umx_is_RAM(obj)
```

Arguments

obj	an object to be tested to see if it is an OpenMx RAM OpenMx::mxModel()

Value

- Boolean

References

https://github.com/tbates/umx

See Also

Other Test: umx_check(), umx_check_OS(), umx_check_model(), umx_check_names(), umx_has_CIs(), umx_has_been_run(), umx_has_means(), umx_has_square_brackets(), umx_is_GSEM(), umx_is_LISREL(), umx_is_MxData(), umx_is_MxMatrix(), umx_is_MxModel(), umx_is_cov()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("is_RAM_ex", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)

if(umx_is_RAM(m1)){
 message("nice RAM model!")
}
if(!umx_is_RAM(m1)){
 message("model needs to be a RAM model")
}

## End(Not run)
```

---

umx_log_wide_twin_data

Log-transform wide twin data with a positive shift

Description

Log-transform wide twin data with a positive shift

Usage

```
umx_log_wide_twin_data(varsToTransform, sep, data, twins = 1:2)
```

Arguments

varsToTransform	The base names of the variables (e.g. "DEP")
sep	The separator (e.g. "_T")
data	A wide dataframe
twins	Suffixes for twins (default 1:2)

Value

dataframe with transformed variables

---

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.

You might start off with a TWID which is a concatenation of a familyID and a 2 digit twinID

Generating famID and twinID as used by this function

You can capture the last 2 digits with the mod function: twinID = df$TWID %% 100

You can *drop* the last 2 digits with integer div: famID = df$TWID %/% 100

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 or 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

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

See Also

- merge()

Other Twin Data functions: umx, umx_make_TwinData(), umx_make_twin_data_nice(), umx_residualize(), umx_scale_wide_twin_data(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

Examples

```
## Not run:
# ================================================
# = First make a long format file for the demo =
# ================================================
data(twinData)
tmp = twinData[, -2]
tmp$twinID1 = 1; tmp$twinID2 = 2
long = umx_wide2long(data = tmp, sep = "")
str(long)
# 'data.frame': 7616 obs. of  11 variables:
#  $ fam     : int  1 2 3 4 5 6 7 8 9 10 ...
#  $ zyg     : int  1 1 1 1 1 1 1 1 1 1 ...
#  $ part    : int  2 2 2 2 2 2 2 2 2 2 ...
#  $ cohort  : chr  "younger" "younger" "younger" "younger" ...
#  $ zygosity: Factor w/ 5 levels "MZFF","MZMM",..: 1 1 1 1 1 1 1 1 1 1 ...
#  $ wt      : int  58 54 55 66 50 60 65 40 60 76 ...
#  $ ht      : num  1.7 1.63 1.65 1.57 1.61 ...
#  $ htwt    : num  20.1 20.3 20.2 26.8 19.3 ...
#  $ bmi     : num  21 21.1 21 23 20.7 ...
#  $ age     : int  21 24 21 21 19 26 23 29 24 28 ...
#  $ twinID  : num  1 1 1 1 1 1 1 1 1 1 ...
```

```
# OK. Now to demo long2wide...

# Keeping all columns
wide = umx_long2wide(data= long, famID= "fam", twinID= "twinID", zygosity= "zygosity")
namez(wide) # some vars, like part, should have been passed along instead of made into "part_T1"

# ======================================
# = Demo requesting specific vars2keep =
# ======================================

# Just keep bmi and wt
wide = umx_long2wide(data= long, famID= "fam", twinID= "twinID",
    zygosity = "zygosity", vars2keep = c("bmi", "wt")
)

namez(wide)
# "fam" "twinID" "zygosity" "bmi_T1" "wt_T1" "bmi_T2" "wt_T2"

# ==================
# = Demo passalong =
# ==================
# 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"
)
namez(wide)


## End(Not run)
```

---

umx_lower.tri *Get values from lower triangle of a matrix*

Description

umx_lower.tri is a wrapper for lower.tri() and a selection to return values from a lower matrix in one step.

Usage

```
umx_lower.tri(x, diag = FALSE)
```

Arguments

x	a matrix() from which to extract values.
diag	whether to include the diagonal (default = FALSE).

Value

- values of cells of the lower triangle.

See Also

- lower.tri()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
x = qm(1,2,3|4,5,6|7,8,9)
umx_lower.tri(x)
# 4,7,8
umx_lower.tri(x, diag=TRUE) # 1 4 7 5 8 9
```

---

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" OpenMx::mxMatrix(), and returns a full matrix, copying the lower triangle into the upper.

Usage

```
umx_lower2full(lower.data, diag = NULL, byrow = TRUE, dimnames = NULL)
```

Arguments

lower.data	An OpenMx::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

- OpenMx::mxMatrix()

References

https://github.com/tbates/umx

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
# 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, 1.000
)
x = umx_lower2full(tmp, diag = TRUE)
# check
isSymmetric(x)

# 2. Test with matrix input
tmpn = c("ROccAsp", "REdAsp", "FOccAsp", "FEdAsp", "RParAsp",
         "RIQ", "RSES", "FSES", "FIQ", "FParAsp")
tmp = matrix(nrow = 10, ncol = 10, byrow = TRUE, dimnames = list(tmpn,tmpn), data =
 c(1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
 0.6247, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
 0.3269, 0.3669, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
 0.4216, 0.3275, 0.6404, 1.0000, 0.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
 0.2137, 0.2742, 0.1124, 0.0839, 1.0000, 0.0000, 0.0000,  0.0000, 0.0000, 0,
 0.4105, 0.4043, 0.2903, 0.2598, 0.1839, 1.0000, 0.0000,  0.0000, 0.0000, 0,
 0.3240, 0.4047, 0.3054, 0.2786, 0.0489, 0.2220, 1.0000,  0.0000, 0.0000, 0,
 0.2930, 0.2407, 0.4105, 0.3607, 0.0186, 0.1861, 0.2707,  1.0000, 0.0000, 0,
 0.2995, 0.2863, 0.5191, 0.5007, 0.0782, 0.3355, 0.2302,  0.2950, 1.0000, 0,
 0.0760, 0.0702, 0.2784, 0.1988, 0.1147, 0.1021, 0.0931, -0.0438, 0.2087, 1)
)
x = umx_lower2full(tmp, diag= TRUE)
```

```
isSymmetric(x)

# 3. Test with lower-vector, no diagonal.
tmp = c(
 0.6247,
 0.3269, 0.3669,
 0.4216, 0.3275, 0.6404,
 0.2137, 0.2742, 0.1124, 0.0839,
 0.4105, 0.4043, 0.2903, 0.2598, 0.1839,
 0.3240, 0.4047, 0.3054, 0.2786, 0.0489, 0.2220,
 0.2930, 0.2407, 0.4105, 0.3607, 0.0186, 0.1861, 0.2707,
 0.2995, 0.2863, 0.5191, 0.5007, 0.0782, 0.3355, 0.2302,  0.2950,
 0.0760, 0.0702, 0.2784, 0.1988, 0.1147, 0.1021, 0.0931, -0.0438, 0.2087
)
umx_lower2full(tmp, diag = FALSE)

 # An example with byrow = FALSE

 ldiag = c(
 1, -.17, -.22, -.19, -.12, .81, -.02, -.26, -.2, -.15,
 1, .11, .2, .21, -.01, .7, .1, .7, .1, .17, .22,
 1, .52, .68, -.12, .09, .49, .27, .46,
 1, .5, -.06, .17, .26, .80, .31,
 1, -.1, .19, .36, .23, .42,
 1, .02, -19, -.06, -.06,
 1, .1, .18, .27,
 1, .51, .7,
 1, .55,
 1)
umx_lower2full(tmp, byrow = FALSE, diag = TRUE)
```

---

umx_make *"make" the umx package using devtools: release to CRAN etc.*

Description

Easily run devtools "install", "release", "win", "examples" etc.

Usage

```
umx_make(
  what = c("load", "quickInst", "install", "spell", "sitrep", "deps_install",
   "checkCRAN", "testthat", "run_dontrun", "examples", "vignettes", "win", "rhub",
    "lastRhub", "release", "git", "dev"),
  pkg = "~/bin/umx",
  check = TRUE,
  runDontRun = FALSE,
  start = NULL,
```

```
  spelling = "en_US",
  which = c("win", "mac", "linux", "solaris"),
  run_dont_test = FALSE,
  spell = TRUE,
  dependencies = NA
)
```

Arguments

what	whether to "load", "install", "quickInst", "release" to CRAN, "testthat", test on "win", "spell", open "git" app, install "dev" version, or run "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).
runDontRun	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).
which	What rhub platform to use? c("mac", "linux", "win").
run_dont_test	When checking.
spell	for rhub, check spelling? TRUE
dependencies	for "deps_install", what to install: NA (default) installs hard dependencies only (`Depends`, `Imports`, `LinkingTo`); TRUE adds `Suggests` and dev dependencies (fresh-machine setup); FALSE installs none. Passed to `pak::local_install_dev_deps()`.

Value

None

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`,

xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
# umx_make()  # Just load new code (don't rebuild help etc)
# umx_make(what = "quickInst") # Quick install
# umx_make(what = "install")   # Full package rebuild and install
# umx_make(what = "spell")     # Spellcheck Rd documents
# umx_make(what = "sitrep")    # Are needed packages up to date?
# umx_make(what = "deps_install") # Update needed packages
# umx_make(what = "examples")  # Run the examples
# umx_make(what = "run_dontrun") # Run the dontrun examples
# umx_make(what = "checkCRAN") # Run R CMD check
# umx_make(what = "rhub")      # Check on rhub
# umx_make(what = "win")       # Check on win-builder
# umx_make(what = "release")   # Release to CRAN
# tmp           = umx_make(what = "lastRhub") # View rhub result
# umx_make(what = "git")       # Open Git Desktop app
# umx_make(what = "dev")       # Install dev version from GitHub

## End(Not run)
```

---

umx_make_double_entry_data

Prepare data for Double-Entry Censored Twin Models

Description

Helper to split one or more variables in a twin dataset into a paired continuous column (holding non-censored values) and an ordered factor column (indicating censoring status), ready for `umxACE_DE`.

The function has flexible rules for censoring. You give a `list()` of columns, with the censoring rule for each, and the function creates new matching `var_cens` and `var_cont` columns. A rule can be a simple numeric value to cut at, e.g., `list(wt=0)`) for cut column `wt` at zero. More complex

expressions should be in quotes `list(wt="<= cut")`. **Mutual-NA invariant enforced**: Each row contributes exactly *one* non-missing element per DE trait - continuous density $f(x)$ or threshold CDF $P(Y \leq \tau)$ - preventing double-counting. On non-censored rows, the function sets the var_cont to the observed value and the var_cens value to NA. On censored rows, the continuous column is set to NA, and the correct value of an ordered factor `c("censored","observed")` is set in var_cens (integer 1). Right-censor `>=cut` flips levels to `c("observed","censored")` so censored = integer 2 (upper tail). `NA` in raw propagates to `NA` in both.

Usage

```
umx_make_double_entry_data(
  data,
  cols = NULL,
  doubleEntrySuffix = c("_cont", "_cens"),
  sep = "_T",
  nSib = 2,
  levels = NULL
)
```

Arguments

`data`	The dataframe to process.
`cols`	A named list of variables and their censoring rules. A numeric scalar is left-censoring at that floor; length-2 numeric is interval; character comparison (e.g. `"<= 0"`, `">= 40"`) or a function are also allowed. Known finite bounds are stored in attribute `umxDoubleEntry` for optional use by `umxACE_DE()` with `fixCensorThresholds = "auto"` or `"yes"`.
`doubleEntrySuffix`	Suffixes for the continuous and censored columns (default = c("_cont", "_cens")).
`sep`	Suffix/separator for twin indices (default = "_T").
`nSib`	Number of siblings/twins (default = 2).
`levels`	Ordered factor levels for the censored column. `NULL` (default) chooses by censor side: left/interval/unknown use `c("censored", "observed")`; right uses `c("observed", "censored")`. If supplied, must include the level name `"censored"` (assigned by name when censored, never by index alone).

Details

Double-entry data preparation creates paired continuous (`_cont`) and ordinal factor (`_cens`) columns for censored traits.

To prevent likelihood double-counting during FIML estimation:

- Non-censored observations ($x > cut$) retain their numeric continuous score in `_cont`, while `_cens` is set to `NA`.
- Censored observations ($x \leq cut$) have `_cont` set to `NA`, while `_cens` records the ordinal censored status level.

This ensures each case contributes exactly one mutually exclusive likelihood component (either continuous PDF or ordinal CDF threshold probability).

Value

The modified dataframe with expanded double-entry pairs and attribute `umxDoubleEntry` listing per-trait cut, side, and whether the cut is fixable in a model.

See Also

- umxACE_DE(), umxRAM_DE(), plot(), umxSummary(), umxModify(), umxCompare()

Other Twin Modeling Functions: power.ACE.test(), umx, umxACE(), umxACE_DE(), umxACE_GLM(), umxACEcov(), umxACEv(), umxCP(), umxCloud(), umxDiffMZ(), umxDiscTwin(), umxDoC(), umxDoCp(), umxGxE(), umxGxE_window(), umxGxEbiv(), umxIP(), umxMRDoC(), umxReduce(), umxReduceACE(), umxReduceGxE(), umxRotate.MxModelCP(), umxSexLim(), umxSimplex(), umxSummarizeTwinData(), umxSummaryACE(), umxSummaryACE_GLM(), umxSummaryACEv(), umxSummaryDoC(), umxSummaryGxEbiv(), umxSummarySexLim(), umxSummarySimplex(), umxTwinMaker()

Examples

```
## Not run:
data(twinData)
prep = umx_make_double_entry_data(twinData, cols = list(wt = 0), sep = "")
# attr(prep, "umxDoubleEntry")$pairs[[1]]$cut  # 0

# Then
selDVs = c("ht", "wt_cont", "wt_cens")
# Known LOD
umxACE_DE(data = prep, selDVs = selDVs, sep = "")
# Un-known LOD
umxACE_DE(data = prep, selDVs = selDVs, sep = "", fixCensorThresholds = "no")
# Mix with continuous traits:
umxACE_DE(data = prep, selDVs = c("ht", "wt_cont", "wt_cens"))

## End(Not run)
```

`umx_make_fake_data` *umx_make_fake_data*

Description

This function takes as argument an existing dataset, which must be either a matrix or a data frame. Each column of the dataset must consist either of numeric variables or ordered factors. When one or more ordered factors are included, then a heterogeneous correlation matrix is computed using John Fox's polycor package. Pairwise complete observations are used for all covariances, and the exact pattern of missing data present in the input is placed in the output, provided a new sample size is not requested. Warnings from the polycor::hetcor function are suppressed.

Usage

```
umx_make_fake_data(
  dataset,
  digits = 2,
  n = NA,
  use.names = TRUE,
  use.levels = TRUE,
  use.miss = TRUE,
  mvt.method = "eigen",
  het.ML = FALSE,
  het.suppress = TRUE
)
```

Arguments

dataset	The original dataset of which to make a simulacrum
digits	= Round the data to the requested digits (default = 2)
n	Number of rows to generate (NA = all rows in dataset)
use.names	Whether to name the variables (default = TRUE)
use.levels	= Whether to use existing levels (default = TRUE)
use.miss	Whether to have data 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

OpenMx::mxGenerateData()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
fakeCars = umx_make_fake_data(mtcars)
```

umx_make_GxE_data *Simulate ground-truth Purcell GxE twin data for benchmarking umxGxE*

Description

Generates a wide twin data frame with known path coefficients in the Purcell (2002) moderated ACE form used by `umxGxE()`: \(a_i = a + a_m M\), and similarly for \(c\) and \(e\).

Usage

```
umx_make_GxE_data(
  nMZpairs = 2500L,
  nDZpairs = 2500L,
  a = 0.5,
  c = 0.3,
  e = 0.6,
  am = 0.15,
  cm = 0,
  em = 0,
  betaAge = NULL,
  targetAgeCor = 0.2,
  seed = 1L,
  sharedModerator = TRUE,
  sharedAge = TRUE,
  dzAr = 0.5,
  mu = 0,
  betaMod = 0,
  ageMean = 40,
  ageSd = 10,
  clipModerator = TRUE
)
```

Arguments

nMZpairs	Number of MZ pairs (default 2500).
nDZpairs	Number of DZ pairs (default 2500; total N = 5000 pairs).
a	Main additive path at \(M = 0\) (default 0.50).
c	Main common-environment path at \(M = 0\) (default 0.30).
e	Main unique-environment path at \(M = 0\) (default 0.60).
am	Additive moderation slope \(a_m\) / a-prime (default 0.15).
cm	Common-environment moderation slope (default 0).
em	Unique-environment moderation slope (default 0).
betaAge	Mean slope of age on `outcomeAge`. If `NULL` (default), chosen so that the correlation r(outcomeAge_T1, age_T1) is approximately `targetAgeCor`.

targetAgeCor	Target correlation between outcomeAge and age when calibrating betaAge (default 0.2).
seed	Random seed (default 1).
sharedModerator	If TRUE (default), one moderator draw per family.
sharedAge	If TRUE (default), one age draw per family.
dzAr	DZ additive genetic correlation (default 0.5).
mu	Intercept for both outcomes (default 0).
betaMod	Mean effect of moderator on both outcomes (default 0).
ageMean	Mean of age in years (default 40).
ageSd	SD of age (default 10).
clipModerator	If TRUE (default), clip moderator draws to the interval from -3 to 3.

Details

Two outcomes share the same latent ACE draws and moderation:

outcome GxE only (no age mean effect).

outcomeAge Same as outcome plus a mean effect of age calibrated so the sample correlation with age is approximately targetAgeCor.

Family-level moderator and age are shared by co-twins (mod_T1 == mod_T2, age_T1 == age_T2) by default.

Value

data.frame with columns zygosity, outcome_T1/T2, outcomeAge_T1/T2, mod_T1/T2, age_T1/T2. Attribute "truth" is a named list of generative parameters (a, c, e, am, cm, em, betaAge_outcomeAge, cor_outcomeAge_age, sample sizes, seed, etc.). Base R and umxGxE ignore attributes on the data frame; use attr(df, "truth") only for comparison to estimates.

References

Purcell, S. (2002). Variance components models for gene-environment interaction in twin analysis. Twin Research, 5, 554-571. doi:10.1375/twin.5.6.554

See Also

umxGxE(), umx_make_TwinData()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
df = umx_make_GxE_data(nMZpairs = 200, nDZpairs = 200, seed = 1)
str(df)
truth = attr(df, "truth")
truth$am
truth$a
## Not run:
m1 = umxGxE(selDVs = "outcome", selDefs = "mod", sep = "_T",
  data = df, mzData = "MZ", dzData = "DZ", tryHard = "yes")

## End(Not run)
```

---

umx_make_MR_data *Simulate Mendelian Randomization data*

Description

umx_make_MR_data returns a dataset containing 4 variables: A variable of interest (Y), a putative cause (X), a qtl (quantitative trait locus) influencing X, and a confounding variable (U) affecting both X and Y.

Usage

```
umx_make_MR_data(
  nSubjects = 1000,
  Vqtl = 0.02,
  bXY = 0.1,
  bUX = 0.5,
  bUY = 0.5,
  pQTL = 0.5,
  seed = 123
)
```

Arguments

nSubjects	Number of subjects in sample
Vqtl	Variance of QTL affecting causal variable X (Default 0.02)
bXY	Causal effect of X on Y (Default 0.1)
bUX	Confounding effect of confounder 'U' on X (Default 0.5)
bUY	Confounding effect of confounder 'U' on Y (Default 0.5)
pQTL	Decreaser allele frequency (Default 0.5)
seed	value for the random number generator (Default 123)

Details

The code to make these Data. Modified from Dave Evans 2016 Boulder workshop talk.

Value

- data.frame

See Also

umx_make_TwinData

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

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_raw_from_cov *Turn a cov matrix into raw data*

Description

A wrapper for MASS::mvrnorm() to simplify turning a covariance matrix into matching raw data.

Usage

```
umx_make_raw_from_cov(covMat, n, means = 0, varNames = NULL, empirical = FALSE)
```

Arguments

covMat	A covariance matrix
n	How many rows of data to return
means	the means of the raw data (default = 0)
varNames	default uses "var1", "var2"
empirical	(passed to mvrnorm) Default = FALSE

Value

- data.frame

References

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

See Also

- cov2cor(), MASS::mvrnorm()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

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

myData = umx_make_raw_from_cov(covData, n = 100, means = 1:6)
umxAPA(myData)
covMat = matrix(c(1, .3, .3, 1), nrow=2)
tmp= umx_make_raw_from_cov(covMat, n=10, varNames= c("x", "y"))
cov(tmp)
tmp= umx_make_raw_from_cov(covMat, n=10, varNames= c("x", "y"), empirical= TRUE)
cov(tmp)
tmp= umx_make_raw_from_cov(qm(1, .3| .3, 1), n=10, varNames= c("x", "y"))
cov(tmp)
```

---

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, by default, the file selected in the front-most Finder window will be chosen. 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

None

References

https://github.com/tbates/umx

See Also

Other File Functions: dl_from_dropbox(), umx, umx_file_load_pseudo(), umx_move_file(), umx_open(), umx_rename_file(), umx_write_to_clipboard()

Examples

```
## 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)
```

umx_make_TwinData *Simulate twin data with control over A, C, and E (and D) parameters*

Description

Makes MZ and DZ twin data for standard ACE/ADE, fixed MZ/DZ correlations, thresholds, or **bivariate** GxE (umxGxEbiv()). By default, variance components must sum to 1.

For **univariate** Purcell GxE data for umxGxE(), use umx_make_GxE_data() instead (path form \(a + a_m M\), ground-truth attributes, optional age covariate phenotype). Passing the old univariate style AA = c(avg = ..., min = ..., max = ...) to this function is an error and points you to that helper.

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. **note**: the function was designed around nSib = 2 and var names = var_T1. It isn't yet smart enough to do, for instance scaling or shifting to make the min value 0 (normal for most traits we analyse) for nonstandard varNames and nSib.

Note, if you want a power calculator, see power.ACE.test() and umxPower().

Usage

You must supply nMZpairs (you can omit nDZpairs). You can give any two of A, C, or E and the function deduces the missing parameter so A+C+E == 1.

Bivariate GxE Data (for umxGxEbiv())

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 = 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,
  DD = NULL,
  varNames = "var",
  MZr = NULL,
  DZr = MZr,
  nSib = 2,
  dzAr = 0.5,
  scale = FALSE,
  mean = 0,
  sd = 1,
```

```
  nThresh = NULL,
  sum2one = TRUE,
  bivAmod = NULL,
  bivCmod = NULL,
  bivEmod = NULL,
  seed = NULL,
  empirical = FALSE
)
```

Arguments

nMZpairs	Number of MZ pairs to simulate
nDZpairs	Number of DZ pairs to simulate (defaults to nMZpairs)
AA	value for A variance (scalar). For bivariate GxE, a list of Cholesky elements (see details).
CC	value for C variance.
EE	value for E variance.
DD	value for D variance (univariate ACE/ADE only).
varNames	name for variables (defaults to 'var')
MZr	If MZr and DZr are set (default = NULL), the function returns dataframes of the request n and correlation.
DZr	Set to return dataframe using MZr and Dzr (Default NULL)
nSib	Number of siblings in a family (default = 2). "3" = extra sib.
dzAr	DZ Ar (default .5)
scale	Whether to scale output to var=1 mean=0 (Default FALSE)
mean	mean for traits (default = 0) (not applied to bivariate GxE)
sd	sd of traits (default = 1) (not applied to bivariate GxE)
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)
bivAmod	Used for Bivariate GxE data: list(Beta_a1 = .025, Beta_a2 = .025)
bivCmod	Used for Bivariate GxE data: list(Beta_c1 = .025, Beta_c2 = .025)
bivEmod	Used for Bivariate GxE data: list(Beta_e1 = .025, Beta_e2 = .025)
seed	Allows user to set.seed() if wanting reproducible dataset
empirical	Passed to mvrnorm

Value

- data.frame with twin columns and zygosity (plus moderator columns for bivariate GxE)

References

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

See Also

- umxACE(), umx_make_GxE_data(), umxGxE(), umxGxEbiv()

Other Twin Data functions: umx, umx_long2wide(), umx_make_twin_data_nice(), umx_residualize(), umx_scale_wide_twin_data(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
# ======================================================================
# = Basic Example, with all elements of std univariate data specified =
# ======================================================================
tmp = umx_make_TwinData(nMZpairs = 10000, AA = .30, CC = .00, EE = .70)
# Show dataframe with 20,000 rows and 3 variables: var_T1, var_T2, and zygosity
str(tmp)

# ================================
# = How to consume the datasets =
# ================================

mzData = tmp[tmp$zygosity == "MZ", ]
dzData = tmp[tmp$zygosity == "DZ", ]
str(mzData); str(dzData);
cov(mzData[, c("var_T1", "var_T2")])
cov(dzData[, c("var_T1", "var_T2")])
umxAPA(mzData[, c("var_T1", "var_T2")])

# Prefer to work in path coefficient values? (little a?)
tmp    = umx_make_TwinData(2000, AA = .7^2, CC = .0)
mzData = tmp[tmp$zygosity == "MZ", ]
dzData = tmp[tmp$zygosity == "DZ", ]
m1 = umxACE(selDVs="var", sep="_T", mzData= mzData, dzData= dzData)

# Examine correlations
cor(mzData[,c("var_T1","var_T2")])
cor(dzData[,c("var_T1","var_T2")])

# Example with D (left un-modeled in ACE)
tmp = umx_make_TwinData(nMZpairs = 500, AA = .4, DD = .2, CC = .2)
m1 = umxACE(selDVs="var", data = tmp, mzData= "MZ", dzData= "DZ")
# |    |   a1|   c1|   e1|
# |:---|----:|----:|----:|
# |var | 0.86| 0.24| 0.45|

m1 = umxACE(selDVs="var", data = tmp, mzData= "MZ", dzData= "DZ", dzCr=.25)
# |    |  a1|d1 |   e1|
# |:---|---:|:--|----:|
```

```
# |var | 0.9|.  | 0.44|


# =============
# = Shortcuts =
# =============

# Omit nDZpairs (equal numbers of both by default)
tmp = umx_make_TwinData(100, AA = 0.5, CC = 0.3) # omit any one of A, C, or E (sums to 1)
cov(tmp[tmp$zygosity == "DZ", c("var_T1","var_T2")])

# Not limited to unit variance
tmp = umx_make_TwinData(100, AA = 3, CC = 2, EE = 3, sum2one = FALSE)
cov(tmp[tmp$zygosity == "MZ", c("var_T1","var_T2")])

# Output can be scaled (mean=0, std=1)
tmp = umx_make_TwinData(100, AA = .7, CC = .1, scale = TRUE)
cov(tmp[tmp$zygosity == "MZ", c("var_T1","var_T2")])

# Univariate GxE for umxGxE: use umx_make_GxE_data() (not umx_make_TwinData)
# df = umx_make_GxE_data(nMZpairs = 500, nDZpairs = 500, am = 0.15)
# m1 = umxGxE(selDVs = "outcome", selDefs = "mod", sep = "_T",
#   data = df, mzData = "MZ", dzData = "DZ")

## Not run:

# =====================
# = Threshold Example =
# =====================
tmp = umx_make_TwinData(100, AA = .6, CC = .2, nThresh = 3)
str(tmp)
umx_polychoric(subset(tmp, zygosity=="MZ", c("var_T1", "var_T2")))$polychorics
# Running model with 7 parameters
#           var_T1    var_T2
# var_T1 1.0000000 0.7435457
# var_T2 0.7435457 1.0000000


# ================================================
# = Just use MZr and DZr (also works with nSib>2) =
# ================================================
tmp = umx_make_TwinData(100, MZr = .86, DZr = .60, nSib= 3, varNames = "IQ")
umxAPA(subset(tmp, zygosity == "MZ", paste0("IQ_T", 1:2)))
umxAPA(subset(tmp, zygosity == "DZ", paste0("IQ_T", 1:2)))
m1 = umxACE(selDVs= "IQ", data = tmp)
m1 = umxACE(selDVs= "IQ", data = tmp, nSib=3)
# TODO tmx_ examples of unmodeled D etc.

# 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,
   bivAmod = Amod, bivCmod =Cmod, bivEmod =Emod)
str(tmp)
# 'data.frame': 10000 obs. of  7 variables:
#  $ defM_T1 : num  0.171 0.293 -0.173 0.238 -0.73 ...
#  $ defM_T2 : num  0.492 -0.405 -0.696 -0.829 -0.858 ...
#  $ M_T1    : num  0.171 0.293 -0.173 0.238 -0.73 ...
#  $ var_T1  : num  0.011 0.1045 0.5861 0.0583 1.0225 ...
#  $ M_T2    : num  0.492 -0.405 -0.696 -0.829 -0.858 ...
#  $ var_T2  : num  -0.502 -0.856 -0.154 0.065 -0.268 ...
#  $ zygosity: Factor w/ 2 levels "MZ","DZ": 1 1 1 1 1 1 1 1 1 ...

## End(Not run)
```

`umx_make_twin_data_nice` *Convert a twin dataset into umx standard format.*

Description

`umx_make_twin_data_nice` is a function to convert your twin data into a format used across `umx`. Specifically:

1. Existing column for zygosity is renamed to "zygosity".
2. `sep` is set to "_T"
3. The twinID is is set to sequential digits, i.e. 1,2...

Usage

```
umx_make_twin_data_nice(
  data,
  sep = "",
  zygosity = "zygosity",
  numbering,
  labelNumericZygosity = FALSE,
  levels = 1:5,
  labels = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
)
```

Arguments

`data`	a `data.frame()` to check/convert.
`sep`	existing separator string (will be updated to "_T").
`zygosity`	existing zygosity column name (will be renamed `zygosity`).

numbering	existing twin sequence string (will be updated to _T1, _T2, _T3).
labelNumericZygosity	If TRUE numeric zygosity levels will be set to labels.
levels	legal levels of zygosity (ignored if labelNumericZygosity = FALSE (default 1:5)
labels	labels for each zyg level c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS").

Value

- data.frame()

References

tutorials, tbates/umx

See Also

- umx_wide2long(), umx_long2wide(),

Other Twin Data functions: umx, umx_long2wide(), umx_make_TwinData(), umx_residualize(), umx_scale_wide_twin_data(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

Examples

```
data(twinData)
tmp = twinData
tmp2 = umx_make_twin_data_nice(twinData, sep="", numbering = 1:5, zygosity="zygosity")
tmp$zygosity=NULL
tmp = umx_make_twin_data_nice(twinData, sep="", numbering = 1:5, zygosity="zygosity")
namez(tmp, "zyg")
levels(tmp$zygosity)
```

---

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 Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_r_test(), umx_round(), umx_scale(), umx_var()

Examples

```
tmp = mtcars[,1:4]
tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$hp  = ordered(mtcars$hp)  # binary factor
umx_means(tmp, ordVar = 0, na.rm = TRUE)
```

---

umx_merge_randomized_columns

umx_merge_randomized_columns

Description

umx_merge_randomized_columns is designed to merge data where subjects have been randomized to conditions, so they have a value in one column, and NA in the other condition columns.

It returns a new column of merged scores, and a new column of associated conditions.

Usage

```
umx_merge_randomized_columns(
  colNames,
  df,
  levels = colNames,
  newVarName = "score",
  newCondName = "condition",
  as.factor = FALSE
)
```

Arguments

colNames	Names of the columns containing the condition data.
df	The data frame
levels	optional names for the levels of condition (default = colNames).
newVarName	Name for the new column holding the newVarName (default "score").
newCondName	Name for the new column holding the condition (default "condition").
as.factor	Turn condition into a factor? (FALSE)

Value

- df with new cols

See Also

- umx_long2wide(), prolific_check_ID(), prolific_read_demog(), prolific_anonymize()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
fp = "~/Desktop/Political Ideology_September 13, 2022_10.47.xlsx"
df = readxl::read_excel(fp)
df = df[c(-1,-2), ] # delete temp data and question text
df = data.frame(df)
namez(df, "ris", coll = "vec") # c('RiskAversionNoLotter', 'RiskAversionLottery')
colNames= c('RiskAversionNoLotter', 'RiskAversionLottery')
df = umx_as_numeric(df, colNames, force=TRUE)
tmp = umx_merge_randomized_columns(colNames, df); table(tmp$condition)
tmp = umx_merge_randomized_columns(colNames, df,
 levels = c("treatment", "control")); table(tmp$condition)

## 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	string to select files to process within the selected folder.
fileNameList	List of files to move.
destFolder	Folder to move files to.
test	Boolean determining whether to change the names, or just report a dry run.
overwrite	Boolean determining whether to overwrite files or not (default = FALSE (safe)).

Value

None

See Also

file.rename(), regex()

Other File Functions: dl_from_dropbox(), umx, umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_open(), umx_rename_file(), umx_write_to_clipboard()

Examples

```
## Not run:
base = "~/Desktop/"
dest = "~/Music/iTunes/iTunes Music/Music/"
umx_move_file(baseFolder = base, fileNameList = toMove, destFolder = dest, test= TRUE)

# ============================================================
# = Move all files in downloads ending in ".jpeg" to Desktop =
# ============================================================
umx_move_file(baseFolder = "~/Downloads/", regex=".jpeg",
 destFolder = "~/Desktop/", test= TRUE)

## End(Not run)
```

---

umx_msg *Print the Name and Compact Contents of a Variable*

Description

A highly utilitarian function for rapid inline debugging. umx_msg safely captures the name of a variable (or expression) and prints its contents to the console via message() (stderr), before invisibly returning the original object.

Usage

```
umx_msg(x)
```

Arguments

x	An object, variable, or inline expression to be debug-printed.

Details

Because the return is invisible, you can wrap umx_msg around live assignments or pipeline stages without breaking the logic. It safely handles NULL values, multi-line expressions, and cleanly formats complex structures like data frames.

Value

- Invisibly returns x (allowing inline pass-through).

References

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

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
a = "brian"
umx_msg(a)

# Can be used inline without breaking assignment
a = umx_msg(mtcars[1,2]^2)

b = c("brian", "sally", "jane")
umx_msg(b)

# Safely handles missing data
umx_msg(NULL)

# Cleanly routes dataframe structures to stderr
umx_msg(mtcars[1:2, 1:3])
```

umx_names *umx_names*

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 would otherwise use names.

You can learn more about the matching options (like inverting the selection etc.) in the help for base-R grep.

Usage

```
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")
)

namez(
  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

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

See Also

- Base-R pattern matching functions: grep(). And umx_check_names() to check for existence of names in a dataframe.

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_rot(), umx_str_chars(), umx_str_from_object(), umx_trim()

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() =
# ========================================================================
namez(mtcars, "m", collapse = "vector") # Paste-able R-code for a vector

# 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_names(mtcars, "disp", value  = "grepl")  # which var matches disp
umx_names(mtcars, "^d"  , fixed  = TRUE)  # Vars containing literal '^d' (none...)

# ======================================
# = 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, "2$") # names ending in 2
umx_names(GFF, "[^12bs]$") # 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))
umx_names(mxData(na.omit(twinData[, c("wt1", "wt2")]), type = "raw"))

namez(umxMatrix("bob", "Full", 3,3)$labels)
```

---

umx_open *Open a file or folder*

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 shQuote()'d by this function.

Value

None

References

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

See Also

Other File Functions: dl_from_dropbox(), umx, umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_move_file(), umx_rename_file(), umx_write_to_clipboard()

Examples

```
## Not run:
umx_open() # Default is to open working directory getwd()
umx_open("~/bin/umx/R/misc_and_utility copy.r")

## End(Not run)
```

---

umx_open_CRAN_page *Open the CRAN page for a package*

Description

On MacOS, this function opens the CRAN page for a package. Useful for looking up documentation, checking you have an up-to-date version, showing the package to people etc.

Usage

```
umx_open_CRAN_page(package = "umx", inst = FALSE)
```

Arguments

package	An R package name.
inst	Install and load if not already installed?

Value

None

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
## Not run:
umx_open_CRAN_page("umx")

## End(Not run)
```

---

umx_pad *Pad an Object with NAs*

Description

This function pads an R object (list, data.frame, matrix, atomic vector) with NAs. For matrices, lists and data.frames, this occurs by extending each (column) vector in the object.

Usage

```
umx_pad(x, n)
```

Arguments

x	An R object (list, data.frame, matrix, atomic vector).
n	The final length of each object.

Value

- padded object

References

https://github.com/kevinushey/Kmisc/tree/master/man

See Also

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
umx_pad(1:3, 4)
umx_pad(1:3, 3)
```

umx_polychoric *FIML-based polychoric, polyserial, and Pearson correlations*

Description

Compute polychoric/polyserial/Pearson correlations with FIML.

Usage

```
umx_polychoric(
  data,
  useDeviations = TRUE,
  tryHard = c("no", "yes", "ordinal", "search")
)
```

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

Barendse, M. T., Ligtvoet, R., Timmerman, M. E., & Oort, F. J. (2016). Model Fit after Pairwise Maximum Likelihood. *Frontiers in Psychology*, **7**, 528. doi:10.3389/fpsyg.2016.00528.

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
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")])


## End(Not run)
```

`umx_polypairwise` *FIML-based Pairwise polychoric, polyserial, and Pearson correlations*

Description

Compute polychoric/polyserial/Pearson correlations with FIML in OpenMx, but pair by pair, not across the whole dataset at once.

Usage

```
umx_polypairwise(
  data,
  useDeviations = TRUE,
  printFit = FALSE,
  use = "any",
  tryHard = c("no", "yes", "ordinal", "search")
)
```

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

Barendse, M. T., Ligtvoet, R., Timmerman, M. E., & Oort, F. J. (2016). Model Fit after Pairwise Maximum Likelihood. *Frontiers in Psychology*, **7**, 528. doi:10.3389/fpsyg.2016.00528.

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
# Mixed continuous + binary (polyserial / polychoric). Ordinal FIML can be
# platform-sensitive; keep out of --run-donttest on win-builder.
tmp = mtcars
tmp$am = umxFactor(mtcars$am)
tmp$vs = umxFactor(mtcars$vs)
# Scale continuous only (leave ordered factors alone)
tmp[, c("hp", "mpg")] = umx_scale(tmp[, c("hp", "mpg")])
x = umx_polypairwise(tmp[, c("hp", "mpg", "am", "vs")], tryHard = "yes")
x$R
cor(mtcars[, c("hp", "mpg", "am", "vs")])

## End(Not run)
```

---

umx_polytriowise *FIML-based trio-based polychoric, polyserial, and Pearson correlations*

Description

Compute polychoric/polyserial/Pearson correlations with FIML in OpenMx.

Usage

```
umx_polytriowise(
  data,
  useDeviations = TRUE,
  printFit = FALSE,
  use = "any",
  tryHard = c("no", "yes", "ordinal", "search")
)
```

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

doi:10.3389/fpsyg.2016.00528

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
tmp = mtcars
tmp$am = umxFactor(mtcars$am)
tmp$vs = umxFactor(mtcars$vs)
tmp[, c("hp", "mpg")] = umx_scale(tmp[, c("hp", "mpg")])
x = umx_polytriowise(tmp[, c("hp", "mpg", "am", "vs")], tryHard = "ordinal")
x$R
cor(mtcars[, c("hp", "mpg", "am", "vs")])

## End(Not run)
```

---

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"),
  caption = NULL,
  report = c("markdown", "html"),
  file = c(NA, "tmp.html"),
  na.print = "",
  zero.print = "0",
  justify = "none",
  quote = FALSE,
  suppress = NULL,
  kableExtra = TRUE,
  append = FALSE,
  sortableDF = TRUE,
  html_font = NULL,
 style = c("paper", "material_dark", "classic", "classic_2", "minimal", "material"),
  bootstrap_options = c("hover", "bordered", "condensed", "responsive"),
  lightable_options = "striped",
  both = TRUE,
  ...
)
```

Arguments

x	A data.frame to print (matrices will be coerced to data.frame)
digits	The number of decimal places to print (getOption("digits"))
caption	Optional caption.
report	How to report the results. "html" = open in browser.
file	Whether to write to a file (defaults to NA (no file). Use "html" to open table in browser.
na.print	How to display NAs (default = "")
zero.print	How to display 0 values (default = "0") for sparse tables, using "." can produce more readable results.
justify	Parameter passed to print (defaults to "none")
quote	Whether or not to quote strings (FALSE)
suppress	Minimum numeric value to print (NULL = print all values, no matter how small)
kableExtra	Whether to print the table using kableExtra (if report="html")
append	If html, is this appended to file? (FALSE)
sortableDF	If html, is table sortable? (TRUE)
html_font	Override default font. e.g. "Times" or '"Arial Narrow", arial, helvetica, sans-s'
style	The style for the table "paper","material_dark" etc.
bootstrap_options	e.g. border etc.

lightable_options	e.g. striped
both	If html, is table also printed as markdown? (TRUE)
...	Optional parameters for print

Value

- A dataframe of text

See Also

umx_msg(), umx_set_table_format()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_scholastica_report(), umx_tail(), umx_wide2long(), umx_wide4lmer()

Examples

```
data(mtcars)
umx_print(mtcars[1:10,], digits = 2, zero.print = ".", justify = "left")
umx_print(mtcars[1,1:2], digits = 2, zero.print = "")
umx_print(mtcars[1,1:2], digits = 2, caption = "Hi: I'm the caption!")
## Not run:
umx_print(mtcars[1:10,], report = "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

file	Path to file (Default "" will read from user input)
diag	Whether data include diagonal (Default TRUE)
names	Variable names. (Default as.character(paste0("X", 1:n)))
ensurePD	Whether to coerce the resultant matrix to positive definite (Default FALSE)

Value

- matrix()

References

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

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
## Not run:
require(umx) # for umxRAM
IQtests = c("brainstorm", "matrix", "moral", "shopping", "typing")
allCols = c("C", IQtests, "avgIQ", "maxIQ", "video")

df = umx_read_lower(diag = FALSE, names = allCols)
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)

## End(Not run)
```

---

umx_rename *umx_rename*

Description

Returns a dataframe with variables renamed as desired.

Usage

```
umx_rename(
  data,
  from = NULL,
  to = NULL,
  regex = NULL,
  test = FALSE,
  old = "deprecated_from",
  replace = "deprecated_to"
)
```

Arguments

data	The dataframe in which to rename variables
from	List of existing names that will be found and replaced by the contents of replace. (optional: Defaults to NULL).
to	If used alone, a named collection of c(oldName = "newName") pairs. OR, if "from" is a list of existing names, the list of new names) OR, if "regex" is a regular expression, the replace string)
regex	Regular expression with matches will be replaced using replace as the replace string. (Optional: Defaults to NULL).
test	Whether to report a "dry run", not changing anything. (Default = FALSE).
old	deprecated: use from
replace	deprecated: use to

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: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
tmp = mtcars

tmp = umx_rename(tmp, to = c(cyl = "cylinder"))
# let's check cyl has been changed to cylinder...
namez(tmp, "c")

# Alternate style: from->to, first with a test-run
# Dry run
tmp = umx_rename(tmp, from = "disp", to = "displacement", test= TRUE)
# Actually do it
tmp = umx_rename(tmp, from = c("disp"), to = c("displacement"))
umx_check_names("displacement", data = tmp, die = TRUE)
namez(tmp, "disp")

# This will warn that "disp" does not exist (anymore)
new = c("auto", "displacement", "rear_axle_ratio")
tmp = umx_rename(tmp, from = c("am", "disp", "drat"), to = new)
namez(tmp, "a") # still updated am to auto (and rear_axle_ratio)

# Test using regex (in this case to revert "displacement" to "disp")
tmp = umx_rename(tmp, regex = "lacement", to = "", test= TRUE)
tmp = umx_rename(tmp, regex = "lacement", to = "") # revert to disp
umx_names(tmp, "^d") # all names beginning with a d

# dev: check deprecated format handled...
tmp = umx_rename(tmp, old = c("am", "disp", "drat"), replace = new)
```

---

umx_rename_file *Rename files*

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 too change names.

Usage

```
umx_rename_file(
  findStr = "old",
  replaceStr = NA,
  baseFolder = "Finder",
  test = TRUE,
  ignoreSuffix = TRUE,
  listPattern = NULL,
  overwrite = FALSE
)
```

Arguments

findStr	The pattern to find, i.e., "cats"
replaceStr	The replacement pattern "\1 are not dogs"
baseFolder	Folder to search in. Default ("Finder") will use the current front-most Finder window (on MacOS). Set to NA for a "choose folder" dialog.
test	Boolean determining whether to change files on disk, or just report on what would have happened (Defaults to test = TRUE)
ignoreSuffix	Whether to ignore (don't search in) the suffix (file-type like .mpg) TRUE.
listPattern	A pre-filter for files
overwrite	Boolean determining if an existing file will be overwritten (Defaults to the safe FALSE)

Value

None

See Also

Other File Functions: dl_from_dropbox(), umx, umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_move_file(), umx_open(), umx_write_to_clipboard()

Examples

```
## Not run:
# "Season 01" --> "S01" in current folder in MacOS Finder
umx_rename_file("[Ss]eason +([0-9]+)", replaceStr="S\\1", test = TRUE)

# move date to end of file name
umx_rename_file("^(.*) *([0-9]{2}\\.[0-9]{2}\\.[0-9]+) *(.*)", replaceStr="\\1 \\3 \\2")


## 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

```
umx_reorder(old, newOrder, force = FALSE)
```

Arguments

old	a square matrix of correlation or covariances to reorder
newOrder	Variables you want in the order you wish to have
force	Just assume input is value (default = FALSE)

Value

- the re-ordered/resized matrix

References

https://github.com/tbates/umx

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_score_scale(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
oldMatrix = cov(mtcars)
umx_reorder(oldMatrix, newOrder = c("mpg", "cyl", "disp")) # first 3
umx_reorder(oldMatrix, newOrder = c("hp", "disp", "cyl")) # subset and reordered
umx_reorder(oldMatrix, "hp") # edge-case of just 1-var
```

umx_residualize *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 residuals function:

```
residuals(lm(mpg ~ wt + am, data = mtcars, na.action = na.exclude))
```

This result could then be written over the old DV column.

umx_residualize obviates the user having to build the lm, set na.action, or replace the data. In addition, it has the powerful features 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

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

See Also

Other Twin Data functions: umx, umx_long2wide(), umx_make_TwinData(), umx_make_twin_data_nice(), umx_scale_wide_twin_data(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

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)

# ============================
# = Use the formula interface =
# ============================
r1 = umx_residualize(mpg ~ cyl + I(cyl^2) + disp, data = mtcars)

# validate against using lm
r2 = residuals(lm(mpg ~ cyl + I(cyl^2) + disp, data = mtcars, na.action = na.exclude))
all(r1$mpg == r2)

# ============================================================
# = Residualize twin data (i.e. wide or "1 family per row") =
# ============================================================
# Make some toy "twin" data to demonstrate with
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

covs = c("cyl", "disp")
tmp= umx_residualize(var="mpg", covs=covs, suffixes=c("_T1","_T2"), data = tmp)
str(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_rot *Rotate a vector*

Description

umx_rot rotates the items of a vector (1 place, by default). So: c(1,2,3) -> c(2,3,1)

Usage

```
umx_rot(vec, na.last = FALSE)
```

Arguments

vec	vector to rotate
na.last	Whether to set the last value to NA (default = FALSE)

Value

- OpenMx::mxModel()

See Also

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_str_chars(), umx_str_from_object(), umx_trim()

Examples

```
umx_rot(1:10)
umx_rot(c(3,4,5,6,7))
# [1] 4 5 6 7 3
```

---

umx_round *umx_round*

Description

A version of round() which works on dataframes that contain non-numeric data (or data that cannot be coerced to numeric) Helpful for dealing with table output that mixes numeric and string types.

Usage

```
umx_round(df, digits = getOption("digits"), coerce = FALSE)
```

Arguments

df	a dataframe to round in
digits	how many digits to round to (defaults to getOption("digits"))
coerce	whether to make the column numeric if it is not (default = FALSE)

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_scale(), umx_var()

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	Three or 4 variables forming the two pairs of columns.
alternative	A two (default) or one-sided (greater less) test.

Details

Non-overlapping (no variable in common) correlations in the same dataset. If 4 variables are provided in vars, umx_r_test conducts a test of the correlation of var 1 & 2 differs in magnitude from the correlation of var 3 with var 4. (r.jk and r.hm in cocor speak).

Overlapping (1 variable in common) correlations in the same dataset. If 3 variables are provided in vars, umx_r_test conducts a test of whether the correlation of var 1 & 2 differs in magnitude from the correlation of var 1 with var 3. (r.jk and r.jh in cocor speak).

In the future it will be expanded to handle other correlations, and to take correlations as input.

Value

cocor result.

See Also

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_round(), umx_scale(), umx_var()

Examples

```
# Is the correlation of mpg with cylinder count different from that
# obtaining between disp and hp?
vars = c("mpg", "cyl", "disp", "hp")
umx_r_test(mtcars, vars)
umx_r_test(mtcars, c("mpg", "disp", "hp"))
```

`umx_scale` *Scale data columns, skipping non-scalable columns*

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 `scale()` function adds `attributes()` ("scaled:center" and "scaled:scale", `umx_scale` removes these leaving nice numeric columns. Set `attr= TRUE` to preserve them.

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

See Also

umx_scale_wide_twin_data scale

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_var()

Examples

```
data(twinData)
# note: this example is here to remind us why scaling independently for each
# twin would be very bad! Use  umx_scale_wide_twin_data() instead!
df = umx_scale(twinData, varsToScale = c("wt1", "wt2"))
df = umx_scale(twinData,  attr= TRUE)
plot(wt1 ~ wt2, data = df)
```

---

umx_scale_reliabilities

Accumulate scale reliabilities and report them as a table

Description

umx_score_scale() with alpha = TRUE prints reliability coefficients. umx_scale_reliabilities() collects these across scales, then reports them together.

Usage

```
umx_scale_reliabilities(
  action = "show",
  scale_name = NULL,
  reliability = NULL,
  type = NULL,
  report = c("markdown", "html"),
  digits = 2
)
```

Arguments

action	One of "init" (start or clear the store), "add" (append one row), or "show" (print and return the table). "print" is accepted as an alias of "show". Default "show".
scale_name	Name of the scale (single string, required when action = "add").
reliability	Reliability value (single numeric, required when action = "add").
type	Type of reliability. When action = "add", a single string (required), e.g. "alpha", "omega_t", or "omega_h". When action = "show", an optional filter: one or more types to report (NULL, the default, reports all rows).
report	Format for the printed table: "markdown" or "html".
digits	Decimal places for the printed table (default 2). Display only: stored values keep full precision.

Details

To use, call with "init", score your scales, then call with "show".

Value

- invisible data.frame with columns scale, reliability, and type

See Also

Other Get and set: umx, umx_detect_cores(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
umx_scale_reliabilities("init") # start with an empty store
umx_scale_reliabilities("add", scale_name = "Agreeableness", reliability = 0.72, type = "alpha")
umx_scale_reliabilities("add", scale_name = "Agreeableness", reliability = 0.75, type = "omega_t")
umx_scale_reliabilities("show")
umx_scale_reliabilities("show", type = "alpha") # just the alphas
umx_scale_reliabilities("init") # clear the store
## Not run:
data(bfi, package = "psychTools")
umx_scale_reliabilities("init")
bfi = umx_score_scale("A", pos = 2:5, rev = 1, max = 6, data = bfi, name = "A", alpha = TRUE)
umx_scale_reliabilities("show")

## End(Not run)
```

---

umx_scale_wide_twin_data

Scale wide twin data

Description

Scale wide data across all twins. You offer up a list of variables to scale, e.g. c("DEP", "bmi") and the separator (e.g. sep = "_T") and twin suffixes e.g. 1:2 that paste together to make complete variable names: e.g. "DEP_T1" and "DEP_T2".

Usage

```
umx_scale_wide_twin_data(varsToScale, sep, data, twins = 1:2)
```

Arguments

varsToScale	The base names of the variables ("weight" etc.)
sep	The suffix that distinguishes each case, e.g. "_T")
data	A wide dataframe
twins	Legal digits following sep (default 1:2)

Value

- dataframe with varsToScale standardized

References

https://github.com/tbates/umx

See Also

umx_scale

Other Twin Data functions: umx, umx_long2wide(), umx_make_TwinData(), umx_make_twin_data_nice(), umx_residualize(), umx_wide2longTwinData(), umx_yj_wide_twin_data()

Examples

```
data(twinData)
df = umx_scale_wide_twin_data(data = twinData, varsToScale = c("ht", "wt"), sep = "")
plot(wt1 ~ wt2, data = df)
```

---

umx_scholastica_report

Journal processing report for an ICA Scholastica export

Description

Read a Scholastica CSV export and open an HTML report of editorial efficiency. Journal time on each version runs from that version hitting the desk to the decision on it. Time the author spends between a decision and the next version is counted on its own. The page gives means and medians for triage, review, the editor after the reviews are in, author revision, and acceptance of a revision, with the ggplot figures embedded in the page.

Usage

```
umx_scholastica_report(dataDir = "~/bin/ica/ICAdata", asOf = Sys.Date())
```

Arguments

dataDir	Directory of the Scholastica CSV export. A single string. Default is ~/bin/ica/ICAdata. The files used are manuscripts.csv, decisions.csv, reviewer-invitations.csv, and reviews.csv.
asOf	Date used to age manuscripts that are still in review or still with the author. A Date, or something base::as.Date() can parse. Default is Sys.Date().

Details

Files and keys:

The export is a set of tables joined by id, one Scholastica submission version per manuscript row.

- manuscripts.csv has one row per version. manuscript_id is unique to that version. previous_version_id is the previous version's manuscript_id (empty on the first version). submission_id is shared by every version of the same paper, so it identifies the paper, but it is not reliably the first or the latest manuscript_id. version_number is 1, 2, or 3. created_at is when that version hit the desk. workflow_state is the current state of that version. decision_created_at is when the decision on that version was made.
- decisions.csv has one row per decided version, joined on manuscript_id. decision_type is accept, reject, or revise_and_resubmit. Its created_at matches manuscripts.csv decision_created_at.
- reviewer-invitations.csv is joined on the version manuscript_id. created_at is when the invitation was sent. accepted_at is when the reviewer accepted. workflow_state is submitted (review filed), accepted (agreed, review not in), declined, expired, revoked, or hard_bounced. manuscript_version_number repeats the version.
- reviews.csv has no manuscript_id. It joins to an invitation through reviewer_invitation_id. created_at is when the review was filed. publication_recommendation is Accept, Revise and resubmit, or Reject.
- authors.csv joins on manuscript_id. articles.csv is the published article, joined on manuscript_id, with published_at. activities.csv is a prose log of the same events (submitted, invited, accepted an invitation, submitted a review, made a decision). The clocks below use the structured timestamps, not the prose log. issues.csv is journal issues. posts.csv is empty in the export this function was written against.

Trial rows are dropped: title beginning with [EXAMPLE], or editor_tags containing trial entries to ignore.

A paper is the chain walked from a version with no previous_version_id along previous_version_id. Outcomes come from the latest version's workflow_state: ready_to_publish accepted, rejected or desk_rejected rejected, revise_and_resubmit with no later version still with the author, reviewers_confirmed or reviews_submitted still in review, and withdrawn_before_decision or withdrawn_after_non_terminal_decision withdrawn.

Seconds columns already on manuscripts.csv:

Scholastica also stores durations. They are not what the report plots, because the total mixes the two clocks, and the editor buckets sometimes overlap.

- time_to_decision_in_seconds is journal time on that version only (created_at to decision_created_at). It does not include the author gap.
- time_spent_with_editor_pre_review_in_seconds is meant to be desk time before review. On many versions it matches the wait until the first reviewer accepts. On others it keeps running after reviews have arrived.

- `time_spent_with_editor_post_review_in_seconds` matches last review filed to the decision.
- On two versions, pre-review plus post-review exceeds time-to-decision, so those two buckets are not a partition.
- `time_spent_with_author_in_seconds` is stored on the new version and matches the gap from the previous decision to this version's `created_at`.
- `total_time_to_decision_across_versions_in_seconds` sums version time-to-decision and the author gaps. It is filled on the latest decided version. A paper reviewed in two days, held by the author for a year, and then accepted the next day has a total near a year.

Clocks the report uses:

Timestamps are minute resolution (`2015-01-13 10:18PM` style). Durations are in days.

- Hits the desk: that version's `created_at`.
- Goes out to a reviewer: invitation `created_at`.
- Reviewer takes it on: invitation `accepted_at`.
- Review comes back: `reviews.csv` `created_at`.
- Decision: `decision_created_at`.
- Triage, until a reviewer has it: version `created_at` to the first `accepted_at`. Time until the first invitation is sent is reported beside it.
- Review window: first `accepted_at` to the last review filed on that version.
- Editor after the reviews are in: last review to the decision.
- Round-1 journal time: first version `created_at` to its decision. Desk rejects and editorial accepts with no invitation are in this total, and are also reported on their own.
- Author revision time: decision on version N to `created_at` of version N+1. Each return is one observation. An open revise-and-resubmit is aged to `asOf` and kept out of the completed mean.
- Time to accept a revision: `created_at` to the accept decision on a version whose `decision_type` is `accept` and whose `version_number` is greater than 1.
- Journal time for a finished paper: sum over its versions of (decision - arrived). Author gaps are not in that sum.
- Calendar time: first version `created_at` to the final decision. This is journal time plus author time.

Versions that have not yet been decided are left out of the completed journal means. Their elapsed time is listed under papers still open.

Value

Invisibly, a list. `htmlFile` is the temporary HTML page that was opened. `papers` is one row per paper. `versions` is one row per version. `authorGaps` is one row per completed author return. `reviewTurns` is one row per filed review. `revisionRounds` is one row per decided version after the first.

See Also

Other Miscellaneous Utility Functions: `install.OpenMx()`, `libs()`, `qm()`, `umx`, `umxLav2RAM()`, `umxModelNames()`, `umxRAM2Lav()`, `umxVersion()`, `umx_array_shift()`, `umx_data()`, `umx_find_object()`, `umx_lower.tri()`, `umx_msg()`, `umx_open_CRAN_page()`, `umx_pad()`, `umx_print()`, `umx_tail()`, `umx_wide2long()`, `umx_wide4lmer()`

Examples

```
## Not run:
umx_scholastica_report()
umx_scholastica_report(dataDir = "~/bin/ica/ICAdata")

## End(Not run)
```

`umx_score_scale` *Score a psychometric scale by summing normal and reversed items.*

Description

Use this function to generate scores as the appropriate sum of responses to the normal and reversed items in a scale.

Items must be named on the pattern `basename + N + suffix`, where `base` is the prefix common to all item (column) names, N is item number in the scale, and suffix an optional trail (like "_T1").

`pos` and `rev` are vectors of the item numbers for the normal and reverse-scored item numbers.

To reverse items, the function uses `max` and `min` as the lowest and highest possible response scores to compute how to reverse items.

note: `min` defaults to 1. **TIP**: If you have strings, `umx_score_scale` will work (use `mapStrings = `). *BUT* if you want to make a numeric copy, use `umx_strings2numeric`

Usage

```
umx_score_scale(
  base = NULL,
  pos = NULL,
  rev = NULL,
  min = 1,
  max = NULL,
  data = NULL,
  score = c("total", "proportionCorrect", "errors", "mean", "max", "factor"),
  name = NULL,
  na.rm = TRUE,
  minManifests = NA,
  alpha = FALSE,
  mapStrings = NULL,
  correctAnswer = NULL,
  omegaNfactors = 1,
  digits = 2,
  verbose = FALSE,
  suffix = ""
)
```

Arguments

base	String common to all item names.
pos	The positive-scored item numbers.
rev	The reverse-scored item numbers.
min	Minimum legal response value (default = 1). Not implemented for values other than 1 so far...
max	Maximum legal response value (also used to compute reversed item values).
data	The data frame
score	Score total (default), proportionCorrect, errors, mean, max, or factor scores
name	The name of the scale to be returned. Defaults to "base_score"
na.rm	Whether to delete NAs when computing scores (Default = TRUE) Note: Choice affects mean!
minManifests	How many missing items to tolerate for an individual (when score = factor)
alpha	Print reliability (omega and Cronbach's alpha, default = FALSE). If a store was started with umx_scale_reliabilities() "init", the alpha and omega values are also added to that store.
mapStrings	Recoding input like "No"/"Maybe"/"Yes" into numeric values (0,1,2)
correctAnswer	Use when scoring items with one correct response (1/0).
omegaNfactors	Number of factors for the omega reliability (default = 1)
digits	Rounding for omega etc. (default 2)
verbose	Whether to print the whole omega output (FALSE)
suffix	(if dealing with, e.g. "_T1")

Details

In the presence of NAs, score= "mean" and score = "totals" both return NA unless na.rm = TRUE. score = "max", ignores NAs no matter what.

Value

- scores

References

- Revelle, W. (2022) psych: Procedures for Personality and Psychological Research, Northwestern University, Evanston, Illinois, USA, https://CRAN.R-project.org/package=psych Version = 2.2.9.
- McNeish, D. (2018). Thanks coefficient alpha, we'll take it from here. *Psychological Methods*, **23**, 412-433. doi:10.1037/met0000144.

See Also

umx_strings2numeric

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_select_valid(), umx_stack(), umx_strings2numeric()

Examples

```
library(psych)
library(psychTools)
data(bfi)

# ==============================
# = Score Agreeableness totals =
# ==============================

# Handscore subject 1
# A1(R)+A2+A3+A4+A5 = (6+1)-2 +4+3+4+4 = 20

tmp = umx_score_scale(base = "A", pos = 2:5, rev = 1, max = 6, data= bfi, name = "A")
tmp[1, namez(tmp, "A",ignore.case = FALSE)]
#  A1 A2 A3 A4 A5  A
#  2  4  3  4  4  20

# ====================
# = Request the mean =
# ====================
tmp = umx_score_scale(name = "A", base = "A",
   pos = 2:5, rev = 1, max = 6, data= bfi, score="mean")
tmp$A[1] # = 4

# ========================
# = Request factor score =
# ========================
## Not run:
tmp = umx_score_scale(name = "A", base = "A", pos = 2:5, rev = 1,
   max = 6, score = "factor", minManifests = 4, data= bfi)
#            g
# A2 0.6574826
# A3 0.7581274
# A4 0.4814788
# A5 0.6272332
# A1 0.3736021

# ==================
# = Request alpha  =
# ==================
```

```
tmp=umx_score_scale(base="A", pos=2:5, rev=1, max=6, data=bfi, alpha=TRUE)
# omega t = 0.72

## End(Not run)

# ==================
# = na.rm = TRUE ! =
# ==================
tmpDF = bfi
tmpDF[1, "A1"] = NA
tmp = umx_score_scale("A", pos = 2:5, rev = 1, max = 6, data= tmpDF, score="mean")
tmp$A_score[1] # 3.75

tmp= umx_score_scale("A", pos= 2:5, rev= 1, max = 6, data = tmpDF,
   score="mean", na.rm=FALSE)
tmp$A_score[1] # NA (reject cases with missing items)

# ===============
# = Score = max =
# ===============
tmp = umx_score_scale("A", pos = 2:5, rev = 1, max = 6,
  data = bfi, name = "A", score = "max")
tmp$A[1] # Subject 1 max = 5 (reversed) item 1

# Default scale name
tmp = umx_score_scale("E", pos = 3:5, rev = 1:2, max = 6,
   data= tmp, score = "mean", na.rm = FALSE)
tmp$E_score[1]

# 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"),
 A = c("-A1", "A2", "A3", "A4", "A5")
))
summary(scores)
scores$scores[1, ]
#  E   A
# 3.8 4.0

# Compare output
# (note, by default psych::scoreItems replaces NAs with the sample median...)
RevelleE = as.numeric(scores$scores[,"E"])
RevelleE == tmp[,"E_score"]

# ======================
# = MapStrings examples =
# ======================
mapStrings = c(
   "Very Inaccurate", "Moderately Inaccurate",
   "Slightly Inaccurate", "Slightly Accurate",
   "Moderately Accurate", "Very Accurate")
bfi$As1 = factor(bfi$A1, levels = 1:6, labels = mapStrings)
bfi$As2 = factor(bfi$A2, levels = 1:6, labels = mapStrings)
```

```
bfi$As3 = factor(bfi$A3, levels = 1:6, labels = mapStrings)
bfi$As4 = factor(bfi$A4, levels = 1:6, labels = mapStrings)
bfi$As5 = factor(bfi$A5, levels = 1:6, labels = mapStrings)
bfi= umx_score_scale(name="A" , base="A" , pos=2:5, rev=1, max=6, data = bfi)
bfi= umx_score_scale(name="As", base="As", pos=2:5, rev=1, mapStrings = mapStrings, data= bfi)
```

`umx_select_valid` *Update NA values in one column with valid entries from another*

Description

Merge valid entries from two columns

Usage

```
umx_select_valid(col1, col2, bothways = FALSE, data)
```

Arguments

col1	name of the first column
col2	name of the second column
bothways	Whether to replace from 1 to 2 as well as from 2 to 1
data	The dataframe containing the two columns.

Value

- Updated dataframe

See Also

- within()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_stack(), umx_strings2numeric()

Examples

```
tmp = mtcars
tmp$newDisp = tmp$disp
tmp$disp[c(1,3,6)] = NA
anyNA(tmp$disp) # column has NAs
tmp = umx_select_valid("disp", "newDisp", data = tmp)
anyNA(tmp$disp) # column repaired
```

umx_set_auto_plot *umx_set_auto_plot*

Description

Set autoPlot default for models like umxACE umxGxE etc.

Usage

```
umx_set_auto_plot(autoPlot = NULL, silent = FALSE)
```

Arguments

autoPlot	If TRUE, sets the umx_auto_plot option. Else returns the current value of umx_auto_plot
silent	If TRUE, no message will be printed.

Value

- Current umx_auto_plot setting
- existing value

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_run(), umx_set_checkpoint(), umx_set_condensed_slots(), umx_set_cores(), umx_set_data_variance_check(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_auto_plot() # print current state
old = umx_set_auto_plot(silent = TRUE) # store existing value
old
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 *Automatically run models?*

Description

Set autoRun default for models like umxRAM(), umxACE() 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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_plot(), umx_set_checkpoint(), umx_set_condensed_slots(), umx_set_cores(), umx_set_data_variance_check(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_auto_run() # print existing value
old = umx_set_auto_run(silent = TRUE) # store existing value
umx_set_auto_run(FALSE)  # set to FALSE
umx_set_auto_run(old)    # reinstate
```

umx_set_checkpoint *Set the checkpoint status for a model or global options*

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
)

umx_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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_plot(), umx_set_auto_run(), umx_set_condensed_slots(), umx_set_cores(), umx_set_data_variance_check(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
## Not run:
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)
manifests = names(demoOneFactor)
```

```
m1 = umxRAM("One Factor", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
m1 = umx_set_checkpoint(model = m1)
m1 = mxRun(m1)
umx_checkpoint(0)

## End(Not run)
```

---

umx_set_condensed_slots *umx_set_condensed_slots*

Description

Sets whether newly-created mxMatrices are to be condensed (set to NULL if not being used) or not.

Usage

```
umx_set_condensed_slots(state = NA, silent = FALSE)
```

Arguments

state	what state (TRUE or FALSE) to set condensed slots (default NA returns current value).
silent	If TRUE, no message will be printed.

Value

- current value of condensed slots

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_plot(), umx_set_auto_run(), umx_set_checkpoint(), umx_set_cores(), umx_set_data_variance_check(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_condensed_slots() # print
old = umx_set_condensed_slots(silent = TRUE) # store the existing state
umx_set_condensed_slots(TRUE) # update globally
umx_set_condensed_slots(old) # set back
```

---

umx_set_cores *Set the number of cores your models will use.*

Description

note: This function is intelligent: it will default to all performance cores (low power cores are typically so slow its not worth forcing the power cores to wait for them.

Usage

```
umx_set_cores(cores = NA, model = NULL, silent = FALSE)
```

Arguments

cores	number of cores to use. NA (the default) returns current value. "-1" will set to umx_detect_cores().
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

See Also

- umx_time()

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_plot(), umx_set_auto_run(), umx_set_checkpoint(), umx_set_condensed_slots(), umx_set_data_variance_check(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
## Not run:
library(umx)
manifests = c("mpg", "disp", "gear")
m1 = umxRAM("ind", data = mxData(mtcars[, manifests], type = "raw"),
 umxPath(v.m. = manifests)
)
```

```
umx_set_cores() # print current value
# 5/5 (P-cores / total: 11)
oldCores = umx_set_cores(silent = TRUE)  # store existing value
umx_set_cores(omxDetectCores()) # set to max
umx_set_cores(-1); umx_set_cores() # set to max (performance cores)
m1 = umx_set_cores(1, m1)  # set m1 usage to 1 core
umx_set_cores(model = m1)  # show new value for m1
umx_set_cores(oldCores)    # reinstate old global value
umx_set_cores(3) # Request use of 3 cores

## End(Not run)
```

---

umx_set_data_variance_check *umx_set_data_variance_check*

Description

Set default for data checking in models like umxACE umxGxE etc.

Usage

```
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, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), umx_set_auto_plot(), umx_set_auto_run(), umx_set_checkpoint(), umx_set_condensed_slots(), umx_set_cores(), umx_set_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
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_dollar_symbol *Set the symbol for money*

Description

Set umx_set_dollar_symbol (used in e.g. fin_value_interest()

Usage

```
umx_set_dollar_symbol(umx.dollar.symbol = NULL, silent = FALSE)
```

Arguments

umx.dollar.symbol	symbol for money calculations.
silent	If TRUE, no message will be printed.

Value

- Current umx.dollar.symbol

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_dollar_symbol() # show current state
old = umx_set_dollar_symbol(silent=TRUE) # store existing value
fin_value_interest(100)
umx_set_dollar_symbol(old)    # reinstate
```

umx_set_optimization_options *Set options that affect optimization in OpenMx*

Description

umx_set_optimization_options provides access to get and set options affecting optimization.

Usage

```
umx_set_optimization_options(
  opt = c("mvnRelEps", "mvnMaxPointsA", "Parallel diagnostics"),
  value = NULL,
  model = NULL,
  silent = FALSE
)
```

Arguments

opt	default returns current values of the options listed. Currently "mvnRelEps", "mvnMaxPointsA", and "Parallel diagnostics".
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.

Details

note: For mvnRelEps, values between .0001 to .01 are conventional. Smaller values slow optimization.

Value

- current values if no value set.

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
# show current value for selected or all options
umx_set_optimization_options() # print the existing state(s)
umx_set_optimization_options("mvnRelEps")
## Not run:
umx_set_optimization_options("mvnRelEps", .01) # update globally
umx_set_optimization_options("Parallel diagnostics", value = "Yes")

## End(Not run)
```

---

umx_set_optimizer *Set the optimizer in OpenMx*

Description

umx_set_optimizer provides an easy way to get and set the default optimizer. umx can use several optimizers: SLSQP (the default), CSOLNP, or NPSOL. SLSQP is often a good choice. CSOLNPoften works well for ordinal models. NPSOL is proprietary. It is not in the CRAN version of OpenMx. However it can outperform other optimizers, esp. on CIs.

Usage

```
umx_set_optimizer(opt = NA, model = NULL, silent = FALSE)
```

Arguments

opt	default (NA) returns current value. Current alternatives are "NPSOL" "SLSQP" and "CSOLNP".
model	A model for which to set the optimizer. Default (NULL) sets the optimizer globally.
silent	If TRUE, no message will be printed.

Details

A bare call simply shows the current optimizer:

```
umx_set_optimizer()
> Current Optimizer is: 'SLSQP'. Options are: 'CSOLNP', 'SLSQP', and 'NPSOL'
```

Set the optimizer by name. note: the old optimizer is returned, so you can stash it:

```
oldOpt = umx_set_optimizer("CSOLNP")
# ...
umx_set_optimizer(oldOpt)
```

Value

- current optimizer if nothing requested to be set.

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_optimizer() # print the existing state
old = umx_set_optimizer(silent = TRUE) # store the existing state
umx_set_optimizer("SLSQP") # update globally
umx_set_optimizer(old) # set back
```

---

umx_set_plot_file_suffix

Set output suffix used in umx SEM diagram files saved to disk.

Description

umx SEM diagram files can have a suffix of "gv" (default) or "dot". Interrogate the setting by calling with no value: it will return the current setting. To change the setting call with "gv" or "dot". Or use TRUE to toggle the setting.

Usage

```
umx_set_plot_file_suffix(umx.plot.suffix = NULL, silent = FALSE)
```

Arguments

umx.plot.suffix	The suffix for plot files (if empty current value is returned). "TRUE", toggles setting.
silent	If TRUE, no message will be printed.

Value

- Current setting

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_format(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

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::DiagrammeR()", alternatives are graphviz, svg, png, pdf). 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_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://github.com/tbates/umx

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_separator(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_plot_format() # print current state
old = umx_set_plot_format(silent = TRUE) # store current value
umx_set_plot_format("graphviz")
umx_set_plot_format("DiagrammeR")
umx_set_plot_format("png")
umx_set_plot_format("pdf")
umx_set_plot_format(old) # reinstate
```

---

umx_set_separator *Set the separator*

Description

Set umx_default_separator (used in CI[low sep high]). Default = ","

Usage

```
umx_set_separator(umx_default_separator = NULL, silent = FALSE)
```

Arguments

umx_default_separator	separator for CIs etc. (if empty, returns the current value)
silent	If TRUE, no message will be printed.

Value

- Current umx_default_separator

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_silent(), umx_set_table_format()

Examples

```
library(umx)
umx_set_separator() # show current state
old = umx_set_separator(silent=TRUE) # store existing value
umx_set_separator("|")
umxAPA(.3, .2)
umx_set_separator(old)    # reinstate
```

---

umx_set_silent *Turn off most console and summary output from umx*

Description

Running multiple analyses or simulations, it can be handy to turn off the automatic summary, graphing, and printing that umx does to help interactive sessions. umx_set_silent does this. Summary and graph output, as well as progress and durable console output will be suppressed.

Usage

```
umx_set_silent(value = NA, silent = FALSE)
```

Arguments

value	Boolean stating if umx Models should run silently (TRUE).
silent	If TRUE, this function itself will just return the state of the option, with no user message.

Details

Not every function knows about silent, but most, like umxRAM() etc do.

Under the hood, umx_set_silent sets options("umx_silent"). This can be set to either TRUE or FALSE. If TRUE, then the progress messages from model runs are suppressed. Useful for power simulations etc.

Value

- Current silent value

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_table_format()

Examples

```
library(umx)
old = umx_set_silent() # print & store existing value
umx_set_silent(FALSE, silent = TRUE) # set to FALSE
umx_set_silent(old)   # reinstate
umx_set_silent() # print existing value
```

`umx_set_table_format` *umx_set_table_format*

Description

Set knitr.table.format default (output style for tables). Legal values are "latex", "html", "markdown", "pandoc", and "rst".

Usage

```
umx_set_table_format(knitr.table.format = NULL, silent = FALSE)
```

Arguments

`knitr.table.format`	format for tables (if empty, returns the current value of knitr.table.format)
`silent`	If TRUE, no message will be printed.

Value

- Current knitr.table.format setting

References

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

See Also

Other Get and set: umx, umx_detect_cores(), umx_get_checkpoint(), umx_get_options(), umx_scale_reliabilities(), 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_dollar_symbol(), umx_set_optimization_options(), umx_set_optimizer(), umx_set_plot_file_suffix(), umx_set_plot_format(), umx_set_separator(), umx_set_silent()

Examples

```
library(umx)
umx_set_table_format() # show current state
old = umx_set_table_format() # store existing value
umx_set_table_format("latex")
umx_set_table_format("html")
umx_set_table_format("markdown")
umx_set_table_format("") # get available options
umx_set_table_format(old)    # reinstate
```

---

umx_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

umx_wide2long()

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_strings2numeric()

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
p = ggplot2::ggplot(df, ggplot2::aes(x= mpg, y= values, colour= ind))
p + ggplot2::geom_point()+ggplot2::geom_smooth()
```

---

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 OpenMx::mxModel() whose fit will be reported.
...	Other parameters.

Details

umx_standardize takes umx models, including RAM and twin models, and returns a standardized version.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(),

xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`umx_strings2numeric` *A wrapper to map columns of strings to numeric.*

Description

If you give one column name, the column is converted from string to numeric, and returned as a **vector**. If multiple column names are given (or cols is not set), each is converted, and an updated **data.frame** returned.

Usage

```
umx_strings2numeric(df, cols = NA, mapStrings = NULL)
```

Arguments

df	The df
cols	(optional) list of columns (default = use all)
mapStrings	legal strings which will be mapped in order to numbers.

Value

- df

See Also

Other Data Functions: noNAs(), prolific_anonymize(), prolific_check_ID(), prolific_read_demog(), prolific_scoring_stub(), umx, umxFactor(), umxGSEM_sim_snps(), umxHetCor(), umx_as_numeric(), umx_cont_2_quantiles(), umx_lower2full(), umx_make_GxE_data(), umx_make_MR_data(), umx_make_TwinData(), umx_make_fake_data(), umx_make_raw_from_cov(), umx_merge_randomized_columns(), umx_polychoric(), umx_polypairwise(), umx_polytriowise(), umx_read_lower(), umx_rename(), umx_reorder(), umx_score_scale(), umx_select_valid(), umx_stack()

Examples

```
tmp = data.frame(x=letters)
umx_strings2numeric(tmp, mapStrings = letters)
umx_strings2numeric(tmp, cols= "x", mapStrings = letters)
```

---

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.

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

Details

A use case is including a matrix exponent (that is A %% *A* %% A %*% A...) with a variable exponent.

Value

- OpenMx::mxAlgebra()

References

https://github.com/tbates/umx

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPath, xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMo, xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre, xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(),

xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
alg = umx_string_to_algebra(paste(rep("A", nReps), collapse = " %*% "), name = "test_case")

## End(Not run)
```

`umx_str_chars` *Select desired characters from a string*

Description

`umx_str_chars` returns desired characters of a string

Usage

```
umx_str_chars(what, which)
```

Arguments

`what`	A string
`which`	Chars to select out.

Value

- Array of selected characters

References

https://tbates.github.io, github

See Also

- umx_explode()

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_from_object(), umx_trim()

Examples

```
umx_str_chars("myFpassUword", c(3,8))
```

---

umx_str_from_object *Return variable name as a string*

Description

Utility to return an object's name as a string

Usage

```
umx_str_from_object(x)
```

Arguments

x	an object

Value

- name as string

References

https://github.com/tbates/umx

See Also

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_trim()

Examples

```
umx_str_from_object(mtcars)
# "mtcars"
```

umx_tail *Show the n most extreme rows of a data.frame*

Description

Sort data on the column wag and keep the n most extreme rows. what chooses which extreme:

- high (default): the largest values, largest first.
- low: the smallest values, smallest first.
- both: rows from both ends. An odd n gives the extra row to the high end. The high end is printed first (largest first), then the low end (smallest first).
- abs: the rows with the largest absolute deviation from the median of wag. wag must be numeric. Ties keep their original relative order.

Rows with NA in wag are left out of the ranking. The printed columns are face, in the order given, then wag if it is not already in face. Printing goes through umx_print(): markdown by default, or a browser table when report = "html".

Usage

```
umx_tail(
  data,
  wag,
  face,
  report = c("markdown", "html"),
  n = 6,
  what = c("high", "low", "both", "abs")
)
```

Arguments

data	A data.frame.
wag	Column name to rank on. A single string.
face	Column names to display. A character vector.
report	"markdown" (default) or "html". Passed to umx_print().
n	How many extreme rows to show. A single positive integer. Default is 6, the same as utils::tail().
what	Which extreme. One of "high", "low", "both", "abs".

Value

The selected rows and columns, invisibly. A data.frame.

See Also

umx_print()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_wide2long(), umx_wide4lmer()

Examples

```
data(mtcars)
umx_tail(mtcars, wag = "mpg", face = c("cyl", "hp"), n = 3, what = "high")
umx_tail(mtcars, wag = "mpg", face = c("cyl", "hp"), n = 4, what = "both")
## Not run:
umx_tail(mtcars, wag = "mpg", face = c("cyl", "hp"), n = 3, report = "html")

## End(Not run)
```

---

umx_time *umx_time*

Description

A function to compactly report how long a model took to execute. Comes with some preset styles User can set the format with C-style string formatting.

Usage

```
umx_time(
  x = NA,
  formatStr = c("simple", "std", "custom %H %M %OS3"),
  tz = "GMT",
  autoRun = TRUE
)
```

Arguments

x	A OpenMx::mxModel() or list of models for which to display elapsed time, or 'start' or 'stop'
formatStr	A format string, defining how to show the time (defaults to human readable)
tz	time zone in which the model was executed (defaults to "GMT")
autoRun	If TRUE (default), run the model if it appears not to have been.

Details

The default time format is "simple", which gives only the biggest unit used. i.e., "x seconds" for times under 1 minute. "std" shows time in the format adopted in OpenMx 2.0 e.g. "Wall clock time (HH:MM:SS.hh): 00:00:01.16"

If a list of models is provided, time deltas will also be reported.

If instead of a model the key word "start" is given in x, a start time will be recorded. "stop" gives the time since "start" was called (and clears the timer)

If a model has not been run, umx_time will run it for you.

Value

- invisible time string

References

https://github.com/tbates/umx

See Also

- umx_set_cores(), umx_check_parallel()

Other Reporting Functions: aic_weights(), plot.umx_GWAS(), umx, umxAPA(), umxFactorScores(), umxGetLatents(), umxGetManifests(), umxGetModel(), umxGetParameters(), umxParameters(), umx_aggregate()

Examples

```
## Not run:
require(umx)
umx_time('stop') # alert user stop called when not yet started...
umx_time('stop')
umx_time('start')
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
myData = mxData(cov(demoOneFactor), type = "cov", numObs=500)
m1 = umxRAM("umx_time_example", data = myData,
 umxPath(from = latents, to = manifests),
 umxPath(var = manifests),
 umxPath(var = latents, fixedAt = 1)
)
umx_time(m1) # report time from mxModel
m2 = umxRun(m1)
umx_time(c(m1, m2)) # print comparison table
umx_time('stop') # report the time since timer last started, and restart
umx_time('stop') # report the time since timer was restarted.

## End(Not run)
```

umx_trim *Trim whitespace surrounding a string.*

Description

Returns string without leading or trailing whitespace, like the php function. See also built-in base::trimws() does the same.

Usage

```
umx_trim(string, removeThis = NULL)
```

Arguments

string	to trim
removeThis	if not NULL then this regular expression is removed wherever found in 'string'

Value

- string

See Also

base::trimws()

Other String Functions: tvars(), umx, umx_explode(), umx_explode_twin_names(), umx_grep(), umx_names(), umx_rot(), umx_str_chars(), umx_str_from_object()

Examples

```
umx_trim(" dog") # "dog"
trimws(" dog ", "l") # added by R in v 3.3.0
umx_trim("dog ") # "dog"
umx_trim("\t dog \n") # "dog"
umx_trim("xlsx dog.xlsx", "\\.?xlsx ?") # "dog"
```

umx_tutorials *Easily open vignettes (tutorials) for a package*

Description

Present the user with a list of available vignettes (tutorials) for a package, prompt them to select one by number, and open the selected vignette.

Usage

```
umx_tutorials(package = "umx")
```

Arguments

package	The name of the package. Can be a character string or an unquoted name (default = "umx").

Value

None

See Also

- vignette()

Examples

```
## Not run:
# Open a tutorial from umx
umx_tutorials()

# Open a tutorial from another package, e.g. ggplot2
umx_tutorials(ggplot2)

## End(Not run)
```

---

umx_var *Get variances from a df that might contain some non-numeric columns*

Description

Pass in any dataframe and get variances despite some non-numeric columns. Cells involving these non-numeric columns are set to ordVar (default = 1).

Usage

```
umx_var(
  df,
  format = c("full", "diag", "lower"),
  use = c("complete.obs", "pairwise.complete.obs", "everything", "all.obs",
    "na.or.complete"),
  ordVar = 1,
  digits = NULL,
  strict = TRUE,
  allowCorForFactorCovs = FALSE
)
```

Arguments

df	A dataframe of raw data from which to get variances.
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).
ordVar	The value to return at any ordinal columns (defaults to 1).
digits	digits to round output to (Ignored if NULL). Set for easy printing.
strict	Whether to allow non-ordered factors to be processed (default = FALSE (no)).
allowCorForFactorCovs	When ordinal data are present, use heterochoric correlations in affected cells, in place of covariances.

Value

- OpenMx::mxModel()

References

https://tbates.github.io

See Also

Other Miscellaneous Stats Functions: FishersMethod(), SE_from_p(), geometric_mean(), harmonic_mean(), oddsratio(), reliability(), umx, umxCov2cor(), umxHetCor(), umxParan(), umxWeightedAIC(), umx_apply(), umx_cor(), umx_means(), umx_r_test(), umx_round(), umx_scale()

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")

data(myFADataRaw)
df = myFADataRaw[,c("z1", "z2", "z3")]
df$z1 = mxFactor(df$z1, levels = c(0, 1))
df$z2 = mxFactor(df$z2, levels = c(0, 1))
df$z3 = mxFactor(df$z3, levels = c(0, 1, 2))
umx_var(df, format = "diag")
umx_var(df, format = "full", allowCorForFactorCovs=TRUE)

# Ordinal/continuous mix
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[, "bmi1"], probs = c(.5, .8), 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] = umxFactor(twinData[, ordDVs])
varStarts = umx_var(twinData[, c(ordDVs, "wt1", "wt2")],
  format= "diag", ordVar = 1, use = "pairwise.complete.obs")
```

---

umx_wide2long *umx_wide2long*

Description

Makes wide data long using reshape Hopefully a more robust interface to `reshape()` For twin data, this calls `umx_wide2longTwinData(data =data, sep = sep, verbose = verbose)`

Usage

```
umx_wide2long(
  data = df,
  timevar = list(condition = c("control", "expt")),
 repeated = list(example = c("easyexample", "hardexample"), grade = c("grd1", "grd2")),
  covs = c("Age", "Sex"),
  idvar = "PID",
  sep = "_T",
  verbose = FALSE
)
```

Arguments

data	A data.frame to make long.
timevar	A list of the conditions individuals are in that generate repeated measures, list(condition = c("control", "expt"))
repeated	A list of varied inputs and their levels: i.e., list(exam = c("easy", "hard"), ...)
covs	A vector of variables that do not vary, e.g., c("Age", "Sex", "IQ").
idvar	The column containing the unique ID of the subjects "PID".
sep	For twin data - calls = umx_wide2longTwinData default "_T"
verbose	Whether to be verbose (FALSE)

Details

This is for processing data in which subjects (identified by a `idvar` column, have repeated measures on one or more outcomes. The goal is to make the data into long format, for passing to functions like `lmer`. You must:

1. Set `timevar`. This is a list of the conditions that you repeated. The name becomes a column in the long output. e.g., `list(difficulty = c("easy", "hard"))`
2. Set `repeated` This is a list of the measured outcomes, e.g. `list(DV = c("NASA1_frust", "NASA2_frust"), effort = c("NASA1_eff", "NASA2_eff"))`
3. Set `covs` This is vector of non-repeated non-varying IVs c("age", "sex", "IQ").

Table: The resulting output is like this:

idvar	condition	Age	DV
001	"easy"	45	10
001	"hard"	45	75
002	"easy"	19	54
002	"hard"	19	74

Value

- a long version of the df.

See Also

- `reshape()`

Other Miscellaneous Utility Functions: `install.OpenMx()`, `libs()`, `qm()`, `umx`, `umxLav2RAM()`, `umxModelNames()`, `umxRAM2Lav()`, `umxVersion()`, `umx_array_shift()`, `umx_data()`, `umx_find_object()`, `umx_lower.tri()`, `umx_msg()`, `umx_open_CRAN_page()`, `umx_pad()`, `umx_print()`, `umx_scholastica_report()`, `umx_tail()`, `umx_wide4lmer()`

Examples

```
## Not run:
timevar  = list(difficulty = c("easy", "hard"))
repeated = list(
   frustration = c("NASA1_frustration", "NASA2_frustration"),
   effort      = c("NASA1_effort", "NASA2_effort")
)
df.l = umx_long2wide(data, timevar, repeated, covs = c("Age"), idvar = "PID")

## End(Not run)
```

umx_wide2longTwinData *Change twin 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.

Note, zygosity codings differ among labs. One scheme uses 1 = MZFF, 2 = MZMM, 3 = DZFF, 4 = DZMM, 5 = DZOS or DZFM, 6 = DZMF, with 9 = unknown, and then 50, 51,... for siblings.

Typically, OS twins are ordered Female/Male.

Usage

```
umx_wide2longTwinData(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

reshape(), umx_wide2long(), umx_merge_randomized_columns(), umx_select_valid()

Other Twin Data functions: umx, umx_long2wide(), umx_make_TwinData(), umx_make_twin_data_nice(), umx_residualize(), umx_scale_wide_twin_data(), umx_yj_wide_twin_data()

Examples

```
long = umx_wide2longTwinData(data = twinData, sep = "")
long = umx_wide2longTwinData(data = twinData, sep = "", verbose = TRUE)
str(long)
str(twinData)
```

umx_wide4lmer *Take a long dataframe and make it wide for repeated measures and multi-level analysis*

Description

umx_wide4lmer Transform data from wide to long format for repeated measures and multi-level modeling in R.

Wraps reshape stats::reshape()

Usage

```
umx_wide4lmer(
  repeated = list(y = c("y1", "y2")),
  timevar = list(cond = c("cont", "expt")),
  covs = c("Age", "Sex", "Conscientiousness"),
  data = df,
  idvar = "PID"
)
```

Arguments

repeated	list of repeated measures each in list(y = c("y1", "y2")) form
timevar	list of conditions in a list, e.g., list(condition = c("cont", "expt")),
covs	vector of covariates e.g., c("Age", "Sex", "Conscientiousness")
data	A (long-format) data file
idvar	The variable which links repeated measures, e.g., "ID"

See Also

- umx_long2wide()

Other Miscellaneous Utility Functions: install.OpenMx(), libs(), qm(), umx, umxLav2RAM(), umxModelNames(), umxRAM2Lav(), umxVersion(), umx_array_shift(), umx_data(), umx_find_object(), umx_lower.tri(), umx_msg(), umx_open_CRAN_page(), umx_pad(), umx_print(), umx_scholastica_report(), umx_tail(), umx_wide2long()

Examples

```
## Not run:
covs = c("Age", "Sex", "Conscientiousness")
timevar  = list(Difficulty = c("short", "long"))
repeated = list(
 Frustration = c("NASA_Frustration1", "NASA_Frustration2"),
 Effort      = c("NASA_Effort1"     , "NASA_Effort2"),
 Efficacy    = c("NASA_Performance1", "NASA_Performance2"),
 Howmany     = c("howmany_30secs"   , "howmany_60secs")
```

```
)
df.l = umx_wide4lmer(repeated = repeated, timevar = timevar, covs = covs, data = df, idvar = "PID")

## End(Not run)
```

---

umx_write_to_clipboard *umx_write_to_clipboard*

Description

umx_write_to_clipboard writes data to the clipboard

Usage

```
umx_write_to_clipboard(x)
```

Arguments

x	something to paste to the clipboard

Details

Works on Mac. Let me know if it fails on windows or Unix.

Value

None

See Also

Other File Functions: dl_from_dropbox(), umx, umx_file_load_pseudo(), umx_make_sql_from_excel(), umx_move_file(), umx_open(), umx_rename_file()

Examples

```
## Not run:
umx_write_to_clipboard("hello")

## End(Not run)
```

`umx_yj_wide_twin_data` *Yeo-Johnson transform wide twin data (Non-destructive)*

Description

`umx_yj_wide_twin_data` applies the Yeo-Johnson transformation to wide twin data. It "stacks" the data across twins (T1 and T2) to estimate a single optimal Maximum Likelihood lambda parameter. This ensures that the transformation is identical for both twins, preserving the twin covariance structure.

Usage

```
umx_yj_wide_twin_data(
  data,
  varsToTransform,
  sep = "_T",
  twins = 1:2,
  suffix = "_yj",
  verbose = TRUE
)
```

Arguments

`data`	A wide dataframe
`varsToTransform`	The base names of the variables (e.g. "CAQ")
`sep`	The separator (e.g. "_T")
`twins`	Suffixes for twins (default 1:2)
`suffix`	The suffix for the new transformed columns (default "_yj")
`verbose`	Whether to print parameters and plot distributions (default TRUE)

Details

The Yeo-Johnson transformation is a power transform that handles zero and negative values natively. Lambda is estimated with `car::powerTransform(..., family = "yjPower")` on the stacked twin columns (one lambda for T1 and T2). It is often superior to `log(x+1)` because it uses MLE to find the mathematically optimal power to minimize skewness.

When `verbose = TRUE`, the function reports the lambda value and provides a diagnostic plot comparing the raw and transformed distributions.

Value

- dataframe with original and new transformed variables

References

- Yeo, I. K., & Johnson, R. A. (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4), 954-959.
- Cragg, J. G. (1971). Some Statistical Models for Limited Dependent Variables with Application to the Demand for Durable Goods. *Econometrica*, 39(5), 829-844.

See Also

Other Twin Data functions: umx, umx_long2wide(), umx_make_TwinData(), umx_make_twin_data_nice(), umx_residualize(), umx_scale_wide_twin_data(), umx_wide2longTwinData()

Examples

```
# df = umx_yj_wide_twin_data(data = df, varsToTransform = c("CAQ"), sep = "_T")
```

---

us_skinfold_data *Anthropometric data on twins*

Description

A dataset containing height, weight, BMI, and skin-fold fat measures in several hundred US twin families participating in the MCV Cardiovascular Twin Study (PI Schieken). 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

```
data(us_skinfold_data)
```

Format

A data frame with 53940 twin families (1 per row) each twin measured on 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

Moskowitz, W. B., Schwartz, P. F., & Schieken, R. M. (1999). Childhood passive smoking, race, and coronary artery disease risk: the MCV Twin Study. Medical College of Virginia. *Archives of Pediatrics and Adolescent Medicine*, **153**, 446-453. doi:10.1001/archpedi.153.5.446.

See Also

Other datasets: `Anthro_LDSC`, `Fischbein_wt`, `GFF`, `HSwls`, `Psych_LDSC`, `docData`, `iqdat`, `umx`

Examples

```
## Not run:
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

## End(Not run)
```

---

xgh_check_base_name *Check branch address is valid.*

Description

`xgh_check_base_name` takes a full git branch address, validates it, and returns it as a list. For example, "textmatelives/textmate:fix/browser_sorting" would return `list(owner = "textmatelives", repo = "textmate", branchname = "fix/browser_sorting")`.

Usage

```
xgh_check_base_name(base = "textmatelives/textmate:main")
```

Arguments

base	A three-component branch name. Default "textmatelives/textmate:main"

Value

list of components

References

https://tbates.github.io, tutorials

See Also

gh_feature_branch()

Other github: gh_feature_branch(), gh_message_search(), gh_open_PR_url(), xgh_get_current_branch()

Examples

```
xgh_check_base_name("textmatelives/textmate:fix/browser_sorting")
```

---

xgh_get_current_branch

Get the current git branch

Description

xgh_get_current_branch takes a local_path to return a current branch

Usage

```
xgh_get_current_branch(local_path = ".")
```

Arguments

local_path	Default ".".

Value

- current_branch name

See Also

- gh_open_PR_url()

Other github: gh_feature_branch(), gh_message_search(), gh_open_PR_url(), xgh_check_base_name()

Examples

```
## Not run:
xgh_get_current_branch(local_path)

## End(Not run)
```

`xmuCalculateSRMR` *xmuCalculateSRMR*

Description

`xmuCalculateSRMR` calculates the Standardized Root Mean Square Residual (SRMR) for an OpenMx model.

Usage

```
xmuCalculateSRMR(model)
```

Arguments

`model`	An evaluated OpenMx model.

Value

The SRMR value as a numeric scalar, or NA if calculation fails.

`xmuHasSquareBrackets` *xmuHasSquareBrackets*

Description

Tests if an input has square brackets

Usage

```
xmuHasSquareBrackets(input)
```

Arguments

`input`	an input to test

Value

- TRUE/FALSE

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPath, xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod, xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre, xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor, xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmuHasSquareBrackets("A[1,2]")
```

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

Description

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

Usage

```
xmuLabel(
  obj,
```

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

Arguments

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

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuValues()

Examples

```
## Not run:
# ============================================================
# = Show how OpenMx models are not labeled, and then add labels =
# ============================================================
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 = mxModel("One Factor", type = "RAM",
 manifestVars = manifests, latentVars = latents,
 mxPath(from = latents  , to = manifests),
 mxPath(from = manifests, arrows = 2),
 mxPath(from = latents  , arrows = 2, free = FALSE, values = 1.0),
 mxData(cov(demoOneFactor), type = "cov", numObs=500)
)

umxGetParameters(m1) # Default "matrix address" labels, i.e "One Factor.S[2,2]"
m1 = xmuLabel(m1)
umxGetParameters(m1, free = TRUE) # Informative labels: "G_to_x1", "x4_with_x4", etc.

# ============================================================================
# = Create a new model, with suffixes added to paths, and model renamed =
# ============================================================================
m2 = xmuLabel(m1, suffix= "_male", overRideExisting= TRUE, name = "male")
umxGetParameters(m2, free = TRUE) # suffixes added

# ============================
# = Example Labeling a matrix =
# ============================
a = xmuLabel(mxMatrix(name = "a", "Full", 3, 3, values = 1:9))
a$labels
a = xmuLabel(mxMatrix(name = "a", "Full", 3, 3, values = 1:9), baseName="bob")
a$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
xmuLabel(a, verbose = TRUE)
xmuLabel(a, verbose = TRUE, overRideExisting = FALSE)
xmuLabel(a, verbose = TRUE, overRideExisting = TRUE)

## End(Not run)
```

---

xmuLabel_Matrix *xmuLabel_Matrix (not a user function)*

Description

This function will label all the free parameters in an OpenMx::mxMatrix()

Usage

```
xmuLabel_Matrix(
  mx_matrix = NA,
  baseName = NA,
  setfree = FALSE,
  drop = 0,
  jiggle = NA,
  boundDiag = NA,
  suffix = "",
  verbose = TRUE,
  labelFixedCells = FALSE,
  overRideExisting = FALSE
)
```

Arguments

mx_matrix	an mxMatrix
baseName	A base name for the labels NA
setfree	Whether to set free cells FALSE
drop	What values to drop 0
jiggle	= whether to jiggle start values
boundDiag	set diagonal element lbounds to this numeric value (default = NA = ignore)
suffix	a string to append to each label
verbose	how much feedback to give
labelFixedCells	= FALSE
overRideExisting	Whether to overRideExisting (Default FALSE)

Details

Model developers should just call `xmuLabel()`

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 xmuLabel umx:::xmuLabel_Matrix(mxMatrix("Lower", 3, 3, values = 1, name = "a", byrow = TRUE), jiggle = .05, boundDiag = NA); umx:::xmuLabel_Matrix(mxMatrix("Full" , 3, 3, values = 1, name = "a", byrow = TRUE)); umx:::xmuLabel_Matrix(mxMatrix("Symm" , 3, 3, values = 1, name = "a", byrow = TRUE), jiggle = .05, boundDiag = NA); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "data.a")); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "data.a"), overRideExisting = TRUE); umx:::xmuLabel_Matrix(mxMatrix("Full" , 1, 1, values = 1, name = "a", labels= "test"), overRideExisting = TRUE); See also: fit2 = omxSetParameters(fit1, labels = "a_r1c1", free = FALSE, value = 0, name = "drop_a_row1_c1")

Value

- The labeled `OpenMx::mxMatrix()`

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFro xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmuLabel_MATRIX_Model *xmuLabel_MATRIX_Model (not a user function)*

Description

This function will label all the free parameters in a (non-RAM) OpenMx OpenMx::mxModel() nb: We don't assume what each matrix is for. Instead, the function just sticks labels like "a_r1c1" into each cell i.e., matrix-name + _ + r + rowNumber + c + colNumber

Usage

```
xmuLabel_MATRIX_Model(model, suffix = "", verbose = TRUE)
```

Arguments

model	a matrix-style mxModel to label
suffix	a string to append to each label
verbose	how much feedback to give

Details

Model developers should just call xmuLabel()

Value

- The labeled OpenMx::mxModel()

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathL xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
require(umx)
data(demoOneFactor)
m2 <- mxModel("label_ex",
 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)
```

---

xmuLabel_RAM_Model *xmuLabel_RAM_Model (not a user function)*

Description

This function will label all the free parameters in a RAM OpenMx::mxModel()

Usage

```
xmuLabel_RAM_Model(
  model,
  suffix = "",
  labelFixedCells = TRUE,
  overRideExisting = FALSE,
  verbose = FALSE,
  name = NULL
)
```

Arguments

model	a RAM mxModel to label
suffix	a string to append to each label
labelFixedCells	Whether to labelFixedCells (Default TRUE)
overRideExisting	Whether to overRideExisting (Default FALSE)
verbose	how much feedback to give
name	Add optional name parameter to rename returned model (default = leave it along)

Details

Model developers should just call xmuLabel()

Value

- The labeled OpenMx::mxModel()

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuMI(), xmuMakeOneHeadedPathsFromPa xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMo xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
require(umx); data(demoOneFactor)
# raw but no means
m1 <- mxModel("label_ex", 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)
```

---

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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod, xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre, xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor, xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(, xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

xmuMakeTwoHeadedPathsFromPathList
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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateMod xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thre xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check()

xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmuMaxLevels *xmuMaxLevels*

Description

Get the max levels from df

Usage

```
xmuMaxLevels(df, what = c("value", "name"))
```

Arguments

df	Dataframe to search through
what	Either "value" or "name" (of the max-level column)

Value

- max number of levels in frame

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor

xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmuMaxLevels(mtcars) # NA = no ordinal vars
xmuMaxLevels(umxFactor(mtcars))
xmuMaxLevels(umxFactor(mtcars), what = "name")
```

---

xmuMI *xmuMI (not for end users)*

Description

A function to compute and report modifications which would improve fit. You will probably use umxMI() instead

Usage

```
xmuMI(model, vector = TRUE)
```

Arguments

model	an OpenMx::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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(),

xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmuMinLevels` *xmuMinLevels*

Description

Get the min levels from df

Usage

```
xmuMinLevels(df, what = c("value", "name"))
```

Arguments

`df`	Dataframe to search through
`what`	Either "value" or "name" (of the min-level column)

Value

- min number of levels in frame

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(),

xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmuMinLevels(mtcars) # NA = no ordinal vars
xmuMinLevels(umxFactor(mtcars))
xmuMinLevels(umxFactor(mtcars), what = "name")
```

---

xmuPropagateLabels *xmuPropagateLabels (not a user function)*

Description

You should be calling xmuLabel(). This function is called by xmuLabel_MATRIX_Model

Usage

```
xmuPropagateLabels(model, suffix = "", verbose = TRUE)
```

Arguments

model	a model to label
suffix	a string to append to each label
verbose	whether to say what is being done

Value

- OpenMx::mxModel()

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
require(umx)
data(demoOneFactor)
latents  = c("G")
manifests = names(demoOneFactor)
m1 = mxModel("propage_example", 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")
```

xmuRAM2Ordinal *xmuRAM2Ordinal*

Description

xmuRAM2Ordinal: Convert a RAM model whose data contain ordinal variables to a threshold-based model

Usage

```
xmuRAM2Ordinal(model, verbose = FALSE, name = NULL)
```

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

- OpenMx::mxModel()

See Also

- umxRAM()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_da
xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(),

xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## 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",]
m1 = umxRAM("tim", data = mzData,
 umxPath("bmi1", with = "bmi2"),
 umxPath(v.m.= c("bmi1", "bmi2"))
)

m1 = umxRAM("tim", data = mzData,
 umxPath("obese1", with = "obese2"),
 umxPath(v.m.= c("obese1", "obese2"))
)

## End(Not run)
```

---

xmuTwinSuper_Continuous

Create core of twin model for all-continuous data.

Description

Sets up top, MZ and DZ submodels with a means model, data, and expectation for all-continuous data. called by xmu_make_TwinSuperModel().

Usage

```
xmuTwinSuper_Continuous(
  name = NULL,
  fullVars,
  fullCovs = NULL,
  sep,
  mzData,
```

```
  dzData,
  equateMeans,
  type,
  allContinuousMethod,
  nSib
)
```

Arguments

name	The name of the supermodel
fullVars	Full Variable names (wt_T1)
fullCovs	Full Covariate names (age_T1)
sep	default "_T"
mzData	An mxData object containing the MZ data
dzData	An mxData object containing the DZ data
equateMeans	Whether to equate the means across twins (default TRUE)
type	type
allContinuousMethod	allContinuousMethod
nSib	nSib

Value

- A twin model

See Also

- xmu_make_TwinSuperModel()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(),

xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
 xmuTwinSuper_Continuous(name="twin_super", selVars = selVars, selCovs = selCovs,
    mzData = mzData, dzData = dzData, equateMeans = TRUE, type = type,
    allContinuousMethod = allContinuousMethod, nSib= nSib, sep = "_T" )

## End(Not run)
```

---

xmuTwinSuper_NoBinary *xmuTwinSuper_NoBinary*

Description

xmuTwinSuper_NoBinary

Usage

```
xmuTwinSuper_NoBinary(
  name = NULL,
  fullVars,
  fullCovs = NULL,
  mzData,
  dzData,
  sep,
  nSib,
  equateMeans = TRUE,
  verbose = FALSE
)
```

Arguments

name	= NULL
fullVars	full names of variables
fullCovs	full names of covariates

mzData	mzData
dzData	dzData
sep	sep
nSib	nSib
equateMeans	T/F
verbose	(Default FALSE)

Value

- twin model

Handle 1 or more ordinal variables (no binary)

Means ordinal, but no binary Means: all free, start cont at the measured value, ordinals @0

Notes: Ordinal requires:

1. Variable set to mxFactor
2. For Binary variables:
3. Latent means of binary variables fixedAt 0 (or by data.def?)
4. Latent variance (A + C + E) constrained == 1
5. For Ordinal variables, first 2 thresholds fixed

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(),

xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
#
```

`xmuTwinUpgradeMeansToCovariateModel`

Not for end-users: Add a means model with covariates to a twin model

---

Description

Does the following to `model` (i.e., a umx top/MZ/DZ supermodel):

1. Change `top.expMeans` to `top.intercept`.
2. Create `top.meansBetas` for beta weights in rows (of covariates) and columns for each variable.
3. Add matrices for each twin's data.cov vars (matrixes are called `T1DefVars`).
4. Switch `mxExpectationNormal` in each data group to point to the local `expMean`.
5. Add "expMean" algebra to each data group.

- `grp.expMean` sums `top.intercept` and `grp.DefVars` `%*%` `top.meansBetas` for each twin.

Usage

```
xmuTwinUpgradeMeansToCovariateModel(model, fullVars, fullCovs, nSib, sep)
```

Arguments

`model`	The `umxSuperModel()` we are modifying (must have `MZ` `DZ` and `top` submodels)
`fullVars`	the FULL names of manifest variables
`fullCovs`	the FULL names of definition variables
`nSib`	How many siblings
`sep`	How twin variable names have been expanded, e.g. "_T".

Details

In umx models with no covariates, means live in `top$expMean`

Value

- model, now with means model extended to covariates.

See Also

- called by xmuTwinSuper_Continuous()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(twinData) # ?twinData from Australian twins.
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 10
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
# m1 = umxACE(selDVs= "ht", sep= "", dzData= dzData, mzData= mzData, autoRun= FALSE)
# m2 = xmuTwinUpgradeMeansToCovariateModel(m1, fullVars = c("ht1", "ht2"),
#  fullCovs = c("age1", "sex1", "age2", "sex2"), sep = "")


## End(Not run)
```

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

Description

For models to be estimated, it is essential that path values start at credible values. xmuValues takes on that task for you.

Usage

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

Arguments

obj	The RAM or matrix OpenMx::mxModel(), or OpenMx::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 have starts (useful to speed umxModify())

Details

xmuValues can set start values for the free parameters in both RAM and Matrix OpenMx::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.

note: If you give xmuValues a numeric input, it will use obj as the mean, and return a list of length n, with sd = sd.

Value

- OpenMx::mxModel() with updated start values

References

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

See Also

- Core functions:

Other Advanced Model Building Functions: umx, umxAlgebra(), umxFixAll(), umxJiggle(), umxRAM_GLM(), umxRun(), umxThresholdMatrix(), umxUnexplainedCausalNexus(), xmuLabel()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
latents = c("G")
manifests = names(demoOneFactor)

# ======================================================================
# = Make an OpenMx model (which will lack start values and labels..) =
# ======================================================================
m1 = mxModel("One Factor", type = "RAM",
 manifestVars = manifests, latentVars = latents,
 mxPath(from = latents  , to = manifests),
 mxPath(from = manifests, arrows = 2),
 mxPath(from = latents  , arrows = 2, free = FALSE, values = 1.0),
 mxData(cov(demoOneFactor), type = "cov", numObs=500)
)
mxEval(S, m1) # default variances are jiggled away from near-zero
# Add start values to the model
m1 = xmuValues(m1)
mxEval(S, m1) # plausible variances
umx_print(mxEval(S,m1), 3, zero.print = ".") # plausible variances
xmuValues(14, sd = 1, n = 10) # Return vector of length 10, with mean 14 and sd 1


## End(Not run)
```

---

xmu_ace_de_apply_censor_thresholds

Apply double-entry mean/variance/threshold identification (post-supermodel)

Description

For every DE pair in `contByCens`: release binary `Vtot==1` (continuous variance must not be forced to 1), and equate free means cont=cens. For names in `fixedCuts`, fix the twin-shared threshold at the known cut; other DE pairs keep a free threshold (cut estimated in data units).

Usage

```
xmu_ace_de_apply_censor_thresholds(
  model,
  fixedCuts,
  contByCens,
  selDVs,
  sep,
  nSib = 2,
  equateMeansWithCont = TRUE,
```

```
  freeVariance = TRUE,
  equateMeansForAllPairs = TRUE,
  freeTauStarts = NULL
)
```

Arguments

model	Twin ACE model with top$deviations_for_thresh and top$expMean.
fixedCuts	named numeric cuts (may be empty numeric(0)).
contByCens	named character, cens base -> cont base (all DE pairs).
selDVs	base names per individual (Cholesky / Vtot row order).
sep	twin separator.
nSib	number of sibs.
equateMeansWithCont	free and label-equate means for pairs in fixedCuts.
freeVariance	drop DE cens traits from constrain_Bin_var_to_1.
equateMeansForAllPairs	if TRUE, mean-equate every pair in contByCens (not only fixed cuts).
freeTauStarts	optional named numeric starts for free thresholds (data units); default continuous mean.

Value

modified model.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censo
xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat
xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(),

xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_ace_de_parse_censor_meta` *Resolve which double-entry pairs get fixed censor thresholds*

Description

Combines `umxACE_DE()` arguments with the `umxDoubleEntry` data-frame attribute.

Usage

```
xmu_ace_de_parse_censor_meta(
  mzData,
  dzData,
  doubleEntryPairs,
  fixCensorThresholds,
  censorCuts = NULL,
  doubleEntrySuffix = c("_cont", "_cens")
)
```

Arguments

`mzData`	MZ data frame (may carry `umxDoubleEntry`).
`dzData`	DZ data frame.
`doubleEntryPairs`	list of character length-2 vectors `c(contBase, censBase)`.
`fixCensorThresholds`	`"no"`, `"yes"`, or `"auto"`.
`censorCuts`	named numeric cuts or NULL.
`doubleEntrySuffix`	c("_cont","_cens").

Value

list with `fixedCuts` (named by cens base), `contByCens`, `sideByCens`.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_ace_de_parse_censor_rule` *Parse a double-entry censoring rule into cut, side, and fixable flag*

Description

Character cuts use a strict regex grammar (no `eval`). Indicator evaluation for arbitrary expressions remains in umx_make_double_entry_data().

Usage

```
xmu_ace_de_parse_censor_rule(rule)
```

Arguments

`rule`	Numeric scalar/length-2, character comparison, or function.

Value

list with `side`, `cut`, `fixable`, `ruleRepr`.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_da`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_polycor_blocks()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`, `xmu_show_fit_or_comparison()`, `xmu_simplex_corner()`, `xmu_standardize_ACE()`, `xmu_standardize_ACEcov()`, `xmu_standardize_ACEv()`, `xmu_standardize_CP()`, `xmu_standardize_IP()`, `xmu_standardize_LISREL()`, `xmu_standardize_RAM()`, `xmu_standardize_SexLim()`, `xmu_standardize_Simplex()`, `xmu_start_value_list()`, `xmu_starts()`, `xmu_summary_RAM_group_parameters()`, `xmu_threshold_id_RAM()`, `xmu_threshold_id_twin_check()`, `xmu_topo_sort()`, `xmu_twin_add_WeightMatrices()`, `xmu_twin_check()`, `xmu_twin_get_var_names()`, `xmu_twin_make_def_means_mats_and_alg()`, `xmu_twin_upgrade_selDvs2SelVars()`, `xmu_update_covar()`, `xmu_wls_display_chi()`, `xmu_wls_extract_WV()`

`xmu_ace_de_release_binary_v1` *Drop selected traits from binary Vtot==1 constraint (DE known-cut ID)*

Description

Post-hoc edit of supermodel `binLabels` / `constrain_Bin_var_to_1` only. Does not change `xmuTwinSuper_SomeBinary`.

Usage

```
xmu_ace_de_release_binary_v1(model, vtotIndices)
```

Arguments

model	Twin model with top$binLabels (optional if already absent).
vtotIndices	Integer diagonal indices into Vtot to stop constraining at 1.

Value

modified model.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_bracket_address2rclabel

Convert a bracket address into an A_rXcX-style label.

Description

Takes a label like A[1,1] and returns "A_r1c1".

Usage

```
xmu_bracket_address2rclabel(label, keepPrefix = TRUE)
```

Arguments

label	A bracket label
keepPrefix	Keep any prefix found e.g. "model.top"

Value

- label e.g. "ai_r1c1"

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_bracket_address2rclabel(label = "A[1,1]")
xmu_bracket_address2rclabel(label = "top.A[1,1]")
xmu_bracket_address2rclabel(label = "A_std[1,1]")
```

`xmu_catml_df3_diagnostic` *Polycorrelation-block Jacobian rank degrees of freedom (diagnostic)*

Description

Computes polycorrelation-subsystem rank deficits $df_3 = p^* - \mathrm{rank}(\Delta_{poly})$ and the analogous null rank. **Not used for final Savalei reporting** in `xmu_robust_WLS_fit()`, which takes target df_3 from the model test df and null $df_{3,null}$ from the polycorrelation moment count. Retained for diagnostics and future extensions.

Usage

```
xmu_catml_df3_diagnostic(jacPoly, jacIndPoly)
```

Arguments

`jacPoly`	Polycorrelation rows of the implied Jacobian.
`jacIndPoly`	Independence Jacobian polycorrelation rows.

Value

List with `df3` and `df3Null`.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`,

xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat
xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(),
xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(),
xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(),
xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(),
xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(),
xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(),
xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(),
xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(),
xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(),
xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check()
xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(),
xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(),
xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_catml_discrepancy_at_WLS` *Evaluate catML discrepancy (XX_3) at fixed WLS estimates*

Description

Plugs converged ordinal WLS parameters into the correlation-matrix ML scaffold from `xmu_catml_eval_model()` and computes likelihood-ratio components relative to saturated and independence correlation reference models via `OpenMx::mxRefModels()`.

Usage

```
xmu_catml_discrepancy_at_WLS(model)
```

Arguments

`model`	A fitted ordinal WLS `OpenMx::mxModel()`.

Details

Output components:

`fMlTarget` Target catML discrepancy XX_3 (target minus saturated correlation fit). This is the misfit measure entering robust CFI, TLI, and RMSEA for ordinal WLS.

`fMlNull` Independence catML discrepancy $XX_{3,null}$ (independence minus saturated). Used in robust CFI/TLI denominators.

Degrees of freedom for indices are taken from the WLS model test df in `xmu_robust_WLS_fit()`, not from this function.

Value

List with `fMlTarget`, `fMlNull`, `dfCatMl`, `dfNull`, or `NULL` on failure.

See Also

xmu_catml_eval_model(), xmu_catml_wls_v()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_catml_eval_model *Build correlation-matrix ML evaluation model at fixed WLS estimates*

Description

Constructs a correlation-matrix ML OpenMx::mxModel() with all parameters fixed at the converged ordinal WLS solution. This is the umx/OpenMx analogue of a **catML evaluation scaffold**: a correlation-structure ML model used to evaluate discrepancy at fixed estimates, not to re-fit ordinal data under ML.

Usage

```
xmu_catml_eval_model(model)
```

Arguments

model	A fitted ordinal WLS OpenMx::mxModel().

Details

The model uses mxData(type = "cor") with the observed polychoric correlation matrix from observedStats$cov, and RAM paths with free = FALSE for loadings, latent variances, and residual variances taken from the WLS solution. Thresholds and means are not part of this correlation scaffold; they enter the WLS fit separately.

Value

Fitted correlation ML model (silent run), or NULL on failure.

See Also

xmu_catml_discrepancy_at_WLS(), xmu_catml_implied_correlation()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_catml_implied_correlation` *Model-implied correlation matrix at fixed WLS estimates*

Description

Extracts Σ from the correlation-matrix ML evaluation model built by `xmu_catml_eval_model()`. Parameters are fixed at the converged ordinal WLS solution; no re-optimization is performed. This implied matrix feeds `xmu_catml_wls_v()`.

Usage

```
xmu_catml_implied_correlation(model)
```

Arguments

`model`	A fitted ordinal WLS `OpenMx::mxModel()`.

Value

Model-implied correlation matrix, or `NULL` on failure.

See Also

`xmu_catml_eval_model()`, `xmu_catml_discrepancy_at_WLS()`

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_dat`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_polycor_blocks()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`,

xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_catml_wls_v` *catML expected information matrix (V) for correlation moments*

Description

Computes the **catML** `wls.v` matrix used in Savalei (2021) mean-and-variance corrections. This is *not* the OpenMx WLS `asymCov` block and *not* the WLS fitting weight matrix.

Usage

```
xmu_catml_wls_v(impliedCor)
```

Arguments

`impliedCor`	Numeric symmetric correlation matrix ($p \times p$).

Details

The matrix is the expected Fisher information for the $p^*(p-1)/2$ off-diagonal correlation parameters under a multivariate normal correlation model, evaluated at the **model-implied** correlation matrix from the catML evaluation scaffold (correlation-matrix ML at fixed WLS estimates). Algebraically,

$$V = \tfrac{1}{2} D'(\Sigma^{-1} \otimes \Sigma^{-1}) D$$

where D is the duplication map from unique correlation elements to $\mathrm{vec}(\Sigma)$ and Σ is the implied correlation matrix.

Value

Symmetric $p^* \times p^*$ information matrix for polycorrelation moments, where $p^* = p(p-1)/2$.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(),

xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_cell_is_on` *Return whether a cell is in a set location of a matrix*

Description

Helper to determine is a cell is in a set location of a matrix or not. Left is useful for, e.g. twin means matrices.

Usage

```
xmu_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

- OpenMx::mxModel()

References

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

See Also

- xmuLabel()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_check_levels_identical(xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_cell_is_on(r = 3, c = 3, "lower")
xmu_cell_is_on(r = 3, c = 3, "lower_inc")
xmu_cell_is_on(r = 3, c = 3, "upper")
xmu_cell_is_on(r = 3, c = 3, "upper_inc")
xmu_cell_is_on(r = 3, c = 3, "diag")
xmu_cell_is_on(r = 2, c = 3, "diag")
xmu_cell_is_on(r = 3, c = 3, "any")
```

```
a_cp = umxMatrix("a_cp", "Lower", 3, 3, free = TRUE, values = 1:6)
xmu_cell_is_on(r = 3, c = 3, "left", mat = a_cp)
```

`xmu_check_levels_identical` *xmu_check_levels_identical*

Description

Just checks that the factor levels for twins 1 and 2 are the same

Usage

```
xmu_check_levels_identical(df, selDVs, sep, action = c("stop", "ignore"))
```

Arguments

df	data.frame containing the data
selDVs	base names of variables (without suffixes)
sep	text-constant separating base variable names the twin index (1:2)
action	if unequal levels found: c("stop", "ignore")

Value

None

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(),

xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
require(umx)
data(twinData)
baseNames = c("bmi")
selDVs = umx_paste_names(baseNames, "", 1:2)
tmp = twinData[, selDVs]
tmp$bmi1[tmp$bmi1 <= 22] = 22
tmp$bmi2[tmp$bmi2 <= 22] = 22
xmu_check_levels_identical(umxFactor(tmp, sep = ""), selDVs = baseNames, sep = "")
## Not run:
xmu_check_levels_identical(umxFactor(tmp), selDVs = baseNames, sep = "")

## End(Not run)
```

---

xmu_check_needs_means *Check data to see if model needs means.*

Description

Check data to see if model needs means.

Usage

```
xmu_check_needs_means(
  data,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals")
)
```

Arguments

data	OpenMx::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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_check_needs_means(mtcars, type = "Auto")
xmu_check_needs_means(mtcars, type = "FIML")
# xmu_check_needs_means(mtcars, type = "cov")
# xmu_check_needs_means(mtcars, type = "cor")

# TRUE - marginals means means
xmu_check_needs_means(mtcars, type = "WLS", allContinuousMethod= "marginals")
xmu_check_needs_means(mtcars, type = "ULS", allContinuousMethod= "marginals")
xmu_check_needs_means(mtcars, type = "DWLS", allContinuousMethod= "marginals")

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

```
# = Provided as an mxData object =
# ================================
tmp = mxData(mtcars, type="raw")
xmu_check_needs_means(tmp, type = "FIML") # TRUE
xmu_check_needs_means(tmp, type = "ULS", allContinuousMethod= "cumulants") #FALSE
# TRUE - means with marginals
xmu_check_needs_means(tmp, type = "WLS", allContinuousMethod= "marginals")
tmp = mxData(cov(mtcars), type="cov", numObs= 100)
# Should catch this can't be type FIML
xmu_check_needs_means(tmp) # FALSE
tmp = mxData(cov(mtcars), means = umx_means(mtcars), type="cov", numObs= 100)
xmu_check_needs_means(tmp) # TRUE

# ======================
# = One var is a factor =
# ======================
tmp = mtcars
tmp$cyl = factor(tmp$cyl)
xmu_check_needs_means(tmp, allContinuousMethod= "cumulants") # TRUE
xmu_check_needs_means(tmp, allContinuousMethod= "marginals") # TRUE - always has means
```

`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 = umx_set_data_variance_check(silent = TRUE)$minVar,
  maxVarRatio = umx_set_data_variance_check(silent = TRUE)$maxVarRatio
)
```

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

None

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

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_CI_merge *xmu_CI_merge*

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

```
xmu_CI_merge(m1, m2)
```

Arguments

m1	first copy of the model
m2	second copy of the model

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCo, xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
xmu_CI_merge(m1, m2)

## End(Not run)
```

`xmu_CI_stash` *Stash the CI values of a model as strings in the values of the model*

Description

Stash formatted CIs (e.g. ".1 [-.1, .3]") as strings, *overwriting* the parameter values of the model.

Usage

```
xmu_CI_stash(model, digits = 3, dropZeros = FALSE, stdAlg2mat = TRUE)
```

Arguments

model	An OpenMx::mxModel() to get CIs from.
digits	rounding.
dropZeros	makes strings for failed CIs?
stdAlg2mat	treat std as algebra: stash in non std matrix.

Details

I might change this to a lookup-function that gets a CI string if one exists.

Value

- OpenMx::mxModel()

References

https://github.com/tbates/umx

See Also

- umxConfint(), xmu_get_CI()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_DF_to_mxData_TypeCo, xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(),

xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`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

- xmuLabel()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_compare_constraint_not xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_clean_label("data.var", replace = "_")
xmu_clean_label("my.var.lab", replace = "_")
```

---

xmu_compare_constraint_note

Lines reconciling raw free parameters with equality constraints.

Description

OpenMx stores equality OpenMx::mxConstraint() rows in summary(model)$constraints and adds that total to observedStatistics. estimatedParameters stays the raw free-parameter count, which is also the EP column of umxCompare().

Usage

```
xmu_compare_constraint_note(models)
```

Arguments

models	A list of OpenMx::mxModel() objects.

Value

A character vector, one line per model whose equality-constraint row count is not 0. An empty character vector when no model qualifies. Unrun models are skipped.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

xmu_compare_robust_ML *xmu_compare_robust_ML*

Description

Computes Satorra-Bentler scaled difference tests (Satorra & Bentler 2001, 2010) for nested ML models.

Usage

```
xmu_compare_robust_ML(model1, model2)
```

Arguments

model1	First OpenMx::mxModel().
model2	Second OpenMx::mxModel().

Value

A list containing the robust difference fit statistic (diffFit), change in df (diffdf), p-value (p), and comparison scaling factor (scalingFactor).

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(),

xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_compare_WLS *xmu_compare_WLS*

Description

xmu_compare_WLS is a helper function for umxCompare, called for WLS models.

Usage

```
xmu_compare_WLS(baseModel, comparisonModel = NULL)
```

Arguments

baseModel	a model
comparisonModel	a model

Details

xmu_compare_WLS intercepts WLS models when they are being compared, and handles comparison statistics using modern robust fit indices.

Value

- a table

References

tutorials, tutorials

See Also

- umxCompare() OpenMx::mxCompare()

Other Model Summary and Comparison: umx, umxCompare(), umxEquate(), umxMI(), umxReduce(), umxSetParameters(), umxSummary(), xmu_robust_WLS_fit()

Examples

```
## Not run:
# TODO

## End(Not run)
```

xmu_custom_completer *xmu_custom_completer*

Description

A custom autocompletion function for R console. It intercepts the tab-completion process and suggests default values/options (from `c(...)` or logical vectors) for function arguments when completing an argument value.

Usage

```
xmu_custom_completer(env)
```

Arguments

`env`	The completion environment (utils internal CompletionEnv).

Value

NULL (called for its side effects of modifying `env[["comps"]]`).

xmu_data_missing *Drop rows with missing definition variables*

Description

Definition variables can't be missing. This function helps with that.

Usage

```
xmu_data_missing(
  data,
  selVars,
  sep = NULL,
  dropMissingDef = TRUE,
  hint = "data"
)
```

Arguments

`data`	The dataframe to check for missing variables
`selVars`	The variables to check for missingness
`sep`	A sep if this is twin data and selVars are baseNames (default NULL)
`dropMissingDef`	Whether to drop the rows, or just stop (TRUE)
`hint`	info for message to user ("data")

Value

- data with missing rows dropped

See Also

- complete.cases()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_swap_a_block(), xmu_describe_data_WLS(, xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
tmp = mtcars;
tmp[1,]; tmp[1, "wt"] = NA
tmp = xmu_data_missing(tmp, selVars = "wt", sep= NULL, dropMissingDef= TRUE, hint= "mtcars")
dim(mtcars)
dim(tmp)

## Not run:
tmp = xmu_data_missing(tmp, selVars = "wt", sep= NULL, dropMissingDef= FALSE, hint= "mtcars")

## End(Not run)
```

`xmu_data_swap_a_block` *Data helper function to swap blocks of data from one set of columns to another.*

Description

Swap a block of rows of a dataset between two sets of variables (typically twin 1 and twin 2)

Usage

```
xmu_data_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 xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_dat`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`,

xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

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)
xmu_data_swap_a_block(test, rowSelector = c(1,2,3,6), T1Names = "b", T2Names = "c")
xmu_data_swap_a_block(test, rowSelector = c(1,2,3,6), T1Names = c("a","c"), T2Names = c("b","d"))
```

---

xmu_describe_data_WLS *Determine if a dataset will need statistics for the means if used in a WLS model.*

Description

Given either a data.frame or raw mxData, this function determines whether OpenMx::mxFitFunctionWLS() will generate expectations for means.

Usage

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

Arguments

data	The raw data being used in a OpenMx::mxFitFunctionWLS() model.
allContinuousMethod	the method used to process data when all columns are continuous (default = "cumulants")
verbose	Whether or not to report diagnostics.

Details

All-continuous models processed using the "cumulants" method LACK means, while all continuous processed with allContinuousMethod = "marginals" will HAVE means.

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

Value

- list describing the data.

See Also

- `OpenMx::mxFitFunctionWLS()`, See OpenMx internals for `omxAugmentDataWithWLSSummary` (used in WLS).

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_dat`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_polycor_blocks()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`, `xmu_show_fit_or_comparison()`, `xmu_simplex_corner()`, `xmu_standardize_ACE()`, `xmu_standardize_ACEcov()`, `xmu_standardize_ACEv()`, `xmu_standardize_CP()`, `xmu_standardize_IP()`, `xmu_standardize_LISREL()`, `xmu_standardize_RAM()`, `xmu_standardize_SexLim()`, `xmu_standardize_Simplex()`, `xmu_start_value_list()`, `xmu_starts()`, `xmu_summary_RAM_group_parameters()`, `xmu_threshold_id_RAM()`, `xmu_threshold_id_twin_check()`, `xmu_topo_sort()`, `xmu_twin_add_WeightMatrices()`, `xmu_twin_check()`, `xmu_twin_get_var_names()`, `xmu_twin_make_def_means_mats_and_alg()`, `xmu_twin_upgrade_selDvs2SelVars()`, `xmu_update_covar()`, `xmu_wls_display_chi()`, `xmu_wls_extract_WV()`

Examples

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

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

# ==========================
# = mxData object as input =
# ==========================
tmp = mxData(mtcars, type="raw")
xmu_describe_data_WLS(tmp, allContinuousMethod= "cumulants", verbose = TRUE)$hasMeans # FALSE
xmu_describe_data_WLS(tmp, allContinuousMethod= "marginals")$hasMeans  # TRUE

# ======================================
# = One var is a factor: Means modeled =
# ======================================
tmp = mtcars
tmp$cyl = factor(tmp$cyl)
xmu_describe_data_WLS(tmp, allContinuousMethod= "cumulants")$hasMeans # TRUE - always has means
xmu_describe_data_WLS(tmp, allContinuousMethod= "marginals")$hasMeans # TRUE
```

---

xmu_DF_to_mxData_TypeCov

Convert a dataframe into a cov mxData object

Description

xmu_DF_to_mxData_TypeCov converts a dataframe into OpenMx::mxData() with type="cov" and nrow = numObs and optionally adding means.

Usage

```
xmu_DF_to_mxData_TypeCov(
  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

- OpenMx::mxData() of type = cov

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

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

---

xmu_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 defining the latent, manifest, and means and their shapes

Usage

```
xmu_dot_define_shapes(latents, manifests, preOut = "")
```

Arguments

latents	list of latent variables (including "one")
manifests	list of manifest variables
preOut	existing output string (pasted in front of this: "" by default).

Value

string

See Also

- xmu_dot_rank()

Other Graphviz: xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_mat2dot(), xmu_dot_rank()

Examples

```
xmu_dot_define_shapes(c("as1"), c("E", "N"))
```

---

xmu_dot_maker *Internal umx function to help plotting graphviz*

Description

Helper to print a digraph to file and open it. Safe with options(umx_auto_plot = FALSE) / TRUE: logical and NULL file values are coerced before any cat(..., file =) or DiagrammeR call.

Usage

```
xmu_dot_maker(model, file, digraph, strip_zero = TRUE)
```

Arguments

model	An OpenMx::mxModel() to get the name from
file	Plot destination: "name" (temp/model name), a path string, NA/NULL (print digraph only, no file), or logical (TRUE -> "name", FALSE -> off). Often passed as getOption("umx_auto_plot").
digraph	Graphviz code for a model
strip_zero	Whether to remove the leading "0." in digits in the diagram

Value

- optionally returns the digraph text.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Other Graphviz: xmu_dot_define_shapes(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_mat2dot(), xmu_dot_rank()

`xmu_dot_make_paths` *xmu_dot_make_paths (not for end users)*

Description

Makes graphviz paths

Usage

```
xmu_dot_make_paths(
  mxMat,
  stringIn,
  heads = NULL,
```

```
  fixed = TRUE,
  comment = "More paths",
  showResiduals = TRUE,
  labels = "labels",
  digits = 2,
  pt = NULL,
  std = FALSE,
  uncertainty = c("none", "SE", "MLR", "CI"),
  cis = NULL
)
```

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
labels	show labels on the path? ("none", "labels", "both")
digits	how many digits to report
pt	Parameter table for lookup of estimates and SEs/CIs (defaults to NULL)
std	Whether values should be standardized (defaults to FALSE)
uncertainty	Type of uncertainty to display (defaults to "none")
cis	Matrix of confidence intervals (defaults to NULL)

Value

- string

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(),

xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Other Graphviz: xmu_dot_define_shapes(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_mat2dot(), xmu_dot_rank()

`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"),
  pt = NULL,
  std = FALSE,
  uncertainty = c("none", "SE", "MLR", "CI"),
  cis = NULL
)
```

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"
pt	Parameter table for lookup of estimates and SEs/CIs (defaults to NULL)
std	Whether values should be standardized (defaults to FALSE)
uncertainty	Type of uncertainty to display (defaults to "none")
cis	Matrix of confidence intervals (defaults to NULL)

Value

- list of variance names and variances

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Other Graphviz: xmu_dot_define_shapes(), xmu_dot_make_paths(), xmu_dot_maker(), xmu_dot_mat2dot(), xmu_dot_rank()

`xmu_dot_mat2dot` *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.:

`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

```
xmu_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,
  SEstyle = FALSE,
  p = list(str = "", latents = c(), manifests = c()),
  showCIs = TRUE
)
```

Arguments

`x`	a `umxMatrix()` to make paths from.
`cells`	which cells to process: "any" (default), "diag", "lower", "upper". "left" is the left half (e.g. in a twin means matrix)

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())
showCIs	Whether to show confidence intervals (default = TRUE)

Value

- list(str = "", latents = c(), manifests = c())

See Also

- plot()

Other Graphviz: xmu_dot_define_shapes(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_rank()

Examples

```
# test with a 1 * 1
a_cp = umxMatrix("a_cp", "Lower", 1, 1, free = TRUE, values = pi)
out = xmu_dot_mat2dot(a_cp, cells = "lower_inc", from = "cols", arrows = "both")
cat(out$str) # a_cp -> a_cp [dir = both label="2"];
out = xmu_dot_mat2dot(a_cp, cells = "lower_inc", from = "cols", arrows = "forward",
 fromLabel = "fromMe", toLabel = "toYou",
 fromType  = "latent", toType  = "manifest", digits = 3, SEstyle = TRUE
 )
cat(out$str) # fromMe -> toYou [dir = forward label="3.142"];
cat(out$latent) # fromMe
cat(out$manifest) # toYou

# 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 = xmu_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 = xmu_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 = xmu_dot_mat2dot(a_cp, toLabel= paste0("v", 1:3), cells = "lower", from = "cols")
umx_msg(out$str) # a_cp1 -> v2 [dir = forward label="2"] ...

# First call also inits the plot struct
out = xmu_dot_mat2dot(a_cp, from = "rows", cells = "lower", arrows = "both", fromType = "latent")
out = xmu_dot_mat2dot(a_cp, from = "rows", cells = "diag",
  toLabel= "common", toType = "manifest", p = out)
umx_msg(out$str); umx_msg(out$manifests); umx_msg(out$latents)

# ================================
# = Add found sinks to manifests =
# ================================
out = xmu_dot_mat2dot(a_cp, from= "rows", cells= "diag",
  toLabel= c('a','b','c'), toType= "manifest");
umx_msg(out$manifests)

# ================================
# = Add found sources to latents =
# ================================
out = xmu_dot_mat2dot(a_cp, from= "rows", cells= "diag",
  toLabel= c('a','b','c'), fromType= "latent");
umx_msg(out$latents)


# ========================
# = Label a means matrix =
# ========================

tmp = umxMatrix("expMean", "Full", 1, 4, free = TRUE, values = 1:4)
out = xmu_dot_mat2dot(tmp, cells = "left", from = "rows",
 fromLabel= "one", toLabel= c("v1", "v2")
)
cat(out$str)

## Not run:
# ============================================
# = Get a string which includes CI information =
# ============================================
data(demoOneFactor)
latents = c("g"); manifests = names(demoOneFactor)
m1 = umxRAM("xmu_dot", data = demoOneFactor, type = "cov",
 umxPath(latents, to = manifests),
 umxPath(var = manifests),
 umxPath(var = latents, fixedAt = 1.0)
)
m1 = umxCI(m1, run= "yes")
out = xmu_dot_mat2dot(m1$A, from = "cols", cells = "any",
```

```
      toLabel= paste0("x", 1:5), fromType = "latent", model= m1);
umx_msg(out$str); umx_msg(out$latents)


## End(Not run)
```

---

xmu_dot_move_ranks *xmu_dot_move_ranks (not for end users)*

Description

Variables will be moved from any existing rank to the new one. Setting a rank to "" will clear it.

Usage

```
xmu_dot_move_ranks(
  min = NULL,
  same = NULL,
  max = NULL,
  old_min,
  old_same,
  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)

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(),

xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat
xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(),
xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check()
xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
old_min = c("min1", "min2")
old_same = c("s1", "s2")
old_max = paste0("x", 1:3)

# Add L1 to min
xmu_dot_move_ranks(min = "L1", old_min= old_min, old_same= old_same, old_max= old_max)

# Move min1 to max
xmu_dot_move_ranks(max = "min1", old_min= old_min, old_same= old_same, old_max= old_max)

# Clear min
xmu_dot_move_ranks(min = "", old_min= old_min, old_same= old_same, old_max= old_max)
```

---

xmu_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

```
xmu_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

- xmu_dot_define_shapes()

Other Graphviz: xmu_dot_define_shapes(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_mat2dot()

Examples

```
xmu_dot_rank(c("as1"), "^[ace]s[0-9]+$", "same")
```

---

xmu_dot_rank_str *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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_dot_rank_str(min = "L1", same = c("x1", "x2"), max = paste0("e", 1:3))
```

---

xmu_equate_threshold_values

Equate Threshold Values Across Columns in a Model

Description

This function sets the threshold values for multiple columns in a model to be equal to the threshold values of the first specified column. It is useful in contexts where consistent threshold values are needed across different variables for statistical modeling.

Usage

```
xmu_equate_threshold_values(model, x_cols)
```

Arguments

model	A model object that contains threshold values in its deviations_for_thresh slot.
x_cols	A character vector specifying the names of the columns whose thresholds will be equated.

Value

The modified model object with equated threshold values across the specified columns.

Examples

```
## Not run:
# Assumes `my_model` is a previously defined threshold model
# and has columns "var1", "var2", and "var3" in deviations_for_thresh$values
updated_model = xmu_equate_threshold_values(my_model, x_cols = c("var1", "var2", "var3"))

## End(Not run)
```

---

xmu_extract_column *Get one or more columns from mzData or regular data.frame*

Description

same effect as df[, col] but works for OpenMx::mxData() and check the names are present

Usage

```
xmu_extract_column(data, col, drop = FALSE)
```

Arguments

data	mxData or data.frame
col	the name(s) of the column(s) to extract
drop	whether to drop the structure of the data.frame when extracting one column

Value

- column of data

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_extract_column(mtcars, "wt")
xmu_extract_column(mxData(mtcars, type = "raw"), "wt")
xmu_extract_column(mxData(mtcars, type = "raw"), "wt", drop=TRUE)
xmu_extract_column(mxData(mtcars, type = "raw"), c("wt", "mpg"))
```

---

xmu_get_CI *Look up and report CIs for free parameters*

Description

Look up CIs for free parameters in a model, and return as APA-formatted text string. If std are available, then these are reported.

Usage

```
xmu_get_CI(
  model,
  label,
  prefix = "top.",
  suffix = "_std",
  digits = 2,
  SEstyle = FALSE,
  verbose = FALSE
)
```

Arguments

model	an OpenMx::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	Rounding digits.
SEstyle	If TRUE, report "b(se)" instead of b CI95[l,u] (default = FALSE) If "mxSE" compute these.
verbose	= FALSE

Value

- the CI string, e.g. ".73[-.20, .98]" or .73(.10)

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_has_summary_mxData(),

xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
require(umx); data(demoOneFactor)
manifests = names(demoOneFactor)

tmp = umxRAM("get_CI_example", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1)
)
tmp = umxCI(tmp, run= "yes")

# Get CI by parameter label
xmu_get_CI(model= tmp, "x1_with_x1")
xmu_get_CI(model= tmp, "x1_with_x1", SEstyle = TRUE, digits = 3)

# prefix (submodel) and suffix (e.g. std) are ignored if not needed
xmu_get_CI(model= tmp, "x1_with_x1", prefix = "top.", suffix = "_std")

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_gsem_expand_snp *Expand SNP Summary Statistics into a Full Covariance Matrix*

Description

Takes a baseline covariance structure (`covstruc`) of traits and incorporates the summary statistics of a single SNP to produce a combined $(k+1) \times (k+1)$ covariance matrix (`S`) and corresponding asymptotic covariance matrix (`V`).

Usage

```
xmu_gsem_expand_snp(
  covstruc,
  beta_i,
  se_i,
  varSNP_i,
  varSNPSE2,
  GC = "standard",
  coords = NULL
)
```

Arguments

covstruc	A list containing the base trait covariance matrix $S and its asymptotic covariance $V.
beta_i	Numeric vector of length k. The GWAS betas (effects) of the SNP on the k traits.
se_i	Numeric vector of length k. The GWAS standard errors for the SNP on the k traits.
varSNP_i	Numeric. The variance of the SNP (typically $2 \times MAF \times (1 - MAF)$).
varSNPSE2	Numeric. The asymptotic variance of the SNP's variance estimate.
GC	Character. Genomic control method applied to the cross-trait sampling covariances. Options are "standard", "conserv", or "none".
coords	Optional matrix mapping coordinates for the asymptotic covariance matrix.

Value

A list containing:

- S: The expanded $(k + 1) \times (k + 1)$ covariance matrix.
- V: The expanded $((k + 1)(k + 2)/2) \times ((k + 1)(k + 2)/2)$ asymptotic covariance matrix.

---

xmu_gsem_extract_snp_path

Extract SNP Effect from a Fitted Model

Description

Inspects a fitted MxModel (or model summary) and extracts the estimate and standard error for the specified SNP path. Used internally by umxGSEM_GWAS during the fallback/OpenMx standard loop.

Usage

```
xmu_gsem_extract_snp_path(fit, traits, snpEffect = "SNP_to_F1")
```

Arguments

fit	A fitted MxModel or its summary object.
traits	Character vector of the base trait names.
snpEffect	Character string indicating the name of the parameter to extract. Defaults to "SNP_to_F1".

Value

A list containing:

- est: The parameter estimate.
- se: The standard error of the estimate.
- se_source: Character indicating the source of the SE (e.g., "openmx").
- lab: The parameter label.

---

xmu_gsem_subset_covstruc

Subset a GenomicSEM Covariance Structure to Selected Traits

Description

Filters a covstruc object containing genetic covariance (S), sampling covariance (V), and optionally genomic control (I) matrices down to a specified subset of traits. It ensures proper alignment of the vech-ordered elements of the sampling covariance matrix V.

Usage

```
xmu_gsem_subset_covstruc(covstruc, keep_vars)
```

Arguments

covstruc	A list containing the genetic covariance matrix $S, the sampling covariance matrix $V, and optionally the genomic control matrix $I.
keep_vars	Character vector of trait names to retain in the subset.

Value

A list in the same format as covstruc containing only the subsetted and aligned elements for the traits in keep_vars.

xmu_has_summary_mxData *Check if this version of OpenMx supports modern WLS (mxData of type = "summary")*

Description

Probe whether the installed OpenMx build accepts modern summary WLS data (`mxData(..., type = "summary", observedStats = list(cov=, useWeight=, asymCov=))`). CRAN OpenMx historically did not; the GenomicMx OpenMx build does (`install.OpenMx("GenomicMx")`, binaries from umx GitHub Releases). Result is cached in `options("umx.xmu_has_summary_mxData")` after the first probe.

Usage

```
xmu_has_summary_mxData(force = FALSE)
```

Arguments

force	If TRUE, re-probe and refresh the cache (default FALSE).

Value

Logical.

See Also

xmu_require_summary_mxData(), xmu_mxData_summary()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(),

xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_invert_matrix` *Invert a matrix with chol, then ginv, then solve fallbacks*

Description

Invert a matrix with chol, then ginv, then solve fallbacks

Usage

```
xmu_invert_matrix(x)
```

Arguments

x	Numeric matrix.

Value

Matrix inverse.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat

xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_is_ordinal_WLS` *Detect ordinal or categorical WLS models*

Description

Returns `TRUE` when the fitted model uses raw data and at least one manifest variable is stored as `ordered` or `factor`. This gate controls whether `xmu_robust_WLS_fit()` applies the Savalei (2021) branch rather than the Satorra-Bentler (2010) branch used for continuous WLS.

Usage

```
xmu_is_ordinal_WLS(model)
```

Arguments

`model`	A fitted `OpenMx::mxModel()`.

Value

Logical scalar.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`,

xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_is_wls` *Check if a model is a WLS model*

Description

`xmu_is_wls` is an internal function to check if a model (or any of its submodels) uses WLS/DWLS/ULS.

Usage

```
xmu_is_wls(model)
```

Arguments

`model`	An `OpenMx::mxModel()`

Value

- Boolean

xmu_lavaan_process_group *Process table of paths 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

```
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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(),

xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
tab = lavaan::lavaanify("y~x")
xmu_lavaan_process_group(tab, groupNum = 1)
xmu_lavaan_process_group(tab, groupNum = 0)


## End(Not run)
```

---

xmu_make_bin_cont_pair_data

Make pairs of bin & continuous columns to represent censored data

---

Description

Takes a dataframe of right or left-censored variables (vars with a floor/ceiling 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. The second variable receives a suffix "cont".
2. By default, in each existing variable, it sets all instances of min for that var to NA

Usage

```
xmu_make_bin_cont_pair_data(
  data,
  vars = NULL,
  suffixes = NULL,
  censp = NULL,
  type = "low"
)
```

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)

censp	Optional censoring point
type	Either "low" or "high" (default = "low"). Low for left-censored, high for right-censored.

Value

- copy of the dataframe with new binary variables and censoring

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
df = xmu_make_bin_cont_pair_data(mtcars, vars = c("mpg"))
str(df)
df[order(df$mpgcont), c("mpgcont","mpgbin")]

# Introduce a floor effect
tmp = mtcars; tmp$mpg[tmp$mpg<=15]=15
tmp$mpg_T1 = tmp$mpg_T2 = tmp$mpg
```

```
df = xmu_make_bin_cont_pair_data(tmp, vars = c("mpg"), suffixes = c("_T1", "_T2"))
df[order(df$mpgcont_T1), c("mpgcont_T2", "mpgcont_T1", "mpgbin_T1", "mpgbin_T2")]
```

---

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 and rows 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,
  numObs = NULL,
  weight = NULL,
  fullCovs = NULL,
  dropMissingDef = TRUE,
  verbose = FALSE,
  use = "pairwise.complete.obs"
)
```

Arguments

data	A data.frame() or OpenMx::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.
numObs	Only needed if you pass in a cov/cor matrix wanting this to be upgraded to mxData
weight	Passes weight values to mxData
fullCovs	Covariate names if any (NULL = none) These are checked by dropMissingDef
dropMissingDef	Whether to automatically drop missing def var rows for the user (default = TRUE). You get a polite note.
verbose	If verbose, report on columns kept and dropped (default FALSE)
use	When type = cov or cor, should this drop NAs? (use = "pairwise.complete.obs" by default, with a polite note)

Value

- OpenMx::mxData()

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
# ========================
# = Continuous ML example =
# ========================
data(mtcars)
tmp = xmu_make_mxData(data= mtcars, type = "Auto"); # class(tmp); # "MxDataStatic"
# names(tmp$observed) # "mpg"  "cyl"  "disp"
manVars = c("mpg", "cyl", "disp")
tmp = xmu_make_mxData(data= mtcars, type = "Auto", manifests = manVars);
tmp$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)

## Not run:
# ==========================
# = already mxData example =
# ==========================
m1 = umxRAM("auto", data = mxData(mtcars, type = "raw"),
umxPath(var= "wt"),
umxPath(mean=  "wt")
)


## End(Not run)

# ========================
# = Cov and cor examples =
# ========================
tmp = xmu_make_mxData(data= mtcars, type = "cov", manifests = c("mpg", "cyl"))
tmp = xmu_make_mxData(data= mtcars, type = "cor", manifests = c("mpg", "cyl"))
tmp = xmu_make_mxData(data= cov(mtcars[, c("mpg", "cyl")]),
        type = "cov", manifests = c("mpg", "cyl"), numObs=200)

# mxData input examples
tmp = mxData(cov(mtcars[, c("mpg", "cyl")]), type = "cov", numObs= 100)
xmu_make_mxData(data= tmp, type = "cor", manifests = c("mpg", "cyl")) # consume mxData
xmu_make_mxData(data= tmp, type = "cor", manifests = c("mpg"))      # trim existing mxData
xmu_make_mxData(data= tmp, type = "cor") # no manifests specified (use all)
xmu_make_mxData(data= tmp, manifests = c("mpg", "cyl")) # auto

# ======================
# = Pass string through =
# ======================
xmu_make_mxData(data= c("a", "b", "c"), type = "Auto")
```

---

xmu_make_TwinSuperModel

Helper to make a basic top, MZ, and DZ model.

Description

xmu_make_TwinSuperModel makes basic twin model containing top, MZ, and DZ models. It intelligently handles thresholds for ordinal data, and means model for covariates matrices in the twin models if needed.

Usage

```
xmu_make_TwinSuperModel(
  name = "twin_super",
  mzData,
  dzData,
  selDVs,
  selCovs = NULL,
  sep = NULL,
  type = c("Auto", "FIML", "cov", "cor", "WLS", "DWLS", "ULS"),
  allContinuousMethod = c("cumulants", "marginals"),
  numObsMZ = NULL,
  numObsDZ = NULL,
  nSib = 2,
  equateMeans = TRUE,
  weightVar = NULL,
  bVector = FALSE,
  dropMissingDef = TRUE,
  verbose = FALSE
)
```

Arguments

name	for the supermodel
mzData	Dataframe containing the MZ data
dzData	Dataframe containing the DZ data
selDVs	List of manifest base names (e.g. BMI, NOT 'BMI_T1') (OR, you don't set "sep", the full variable names)
selCovs	List of covariate base names (e.g. age, NOT 'age_T1') (OR, you don't set "sep", the full variable names)
sep	string used to expand selDVs into selVars, i.e., "_T" to expand BMI into BMI_T1 and BMI_T2 (optional but STRONGLY encouraged)
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.
numObsMZ	Number of MZ observations contributing (for summary data only)
numObsDZ	Number of DZ observations contributing (for summary data only)
nSib	Number of members per family (default = 2)
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).
dropMissingDef	Whether to automatically drop missing def var rows for the user (default = TRUE). You get a polite note.
verbose	(default = FALSE)

Details

xmu_make_TwinSuperModel is used in twin models (e.g.umxCP(), umxACE() and umxACEv() and will be added to the other models: umxGxE(), umxIP(), simplifying code maintenance.

It takes mzData and dzData, a list of the selDVs to analyse and optional selCovs (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.

If covariates are passed in these are included in the means model (via a call to xmuTwinUpgradeMeansToCovariateModel.

Modeling

Matrices created

top model

For raw and WLS data, top contains a expMeans 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()).

For covariates, top stores the intercepts matrix and a betaDef matrix. These are then used to make expMeans in MZ and DZ.

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 OpenMx::mxExpectationNormal() and OpenMx::mxFitFunctionML(). For WLS these the fit function is switched to OpenMx::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, these columns are used to create a row-weighted MZ and DZ models.

If equateMeans is TRUE, then the Twin-2 vars in the mean matrix are equated by label with Twin-1.

Decent starts are guessed from the data. 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.

Where needed, e.g. continuous raw data, top adds a means matrix "expMean". For ordinal data, top adds a 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_TwinSuperModel 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 OpenMx::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.

Value

- OpenMx::mxModel()s for top, MZ and DZ.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
# ==============
# = Continuous =
# ==============
library(umx)
data(twinData)
twinData = umx_scale(twinData, varsToScale= c('ht1','ht2'))
mzData = twinData[twinData$zygosity %in%  "MZFF",]
dzData = twinData[twinData$zygosity %in%  "DZFF",]
m1= xmu_make_TwinSuperModel(mzData=mzData, dzData=dzData, selDVs=c("wt","ht"), sep="", nSib=2)
names(m1) # "top" "MZ"  "DZ"
class(m1$MZ$fitfunction)[[1]] == "MxFitFunctionML"
```

```
# ====================
# = With a covariate =
# ====================

m1= xmu_make_TwinSuperModel(mzData=mzData, dzData=dzData,
  selDVs= "wt", selCovs= "age", sep="", nSib=2)
m1$top$intercept$labels
m1$MZ$expMean

# ===============
# = WLS example =
# ===============
m1=xmu_make_TwinSuperModel(mzData=mzData, dzData=dzData,selDVs=c("wt","ht"),sep="",type="WLS")
class(m1$MZ$fitfunction)[[1]] == "MxFitFunctionWLS"
m1$MZ$fitfunction$type =="WLS"
# Check default all-continuous method
m1$MZ$fitfunction$continuousType == "cumulants"

# Choose non-default type (DWLS)
m1= xmu_make_TwinSuperModel(mzData= mzData, dzData= dzData,
 selDVs= c("wt","ht"), sep="", type="DWLS")
m1$MZ$fitfunction$type =="DWLS"
class(m1$MZ$fitfunction)[[1]] == "MxFitFunctionWLS"

# Switch WLS method
m1 = xmu_make_TwinSuperModel(mzData= mzData, dzData= dzData, selDVs= c("wt","ht"), sep= "",
  type = "WLS", allContinuousMethod = "marginals")
m1$MZ$fitfunction$continuousType == "marginals"
class(m1$MZ$fitfunction)[[1]] == "MxFitFunctionWLS"


# ==========================================
# = 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",]
m1 = xmu_make_TwinSuperModel(mzData= mzData, dzData= dzData, selDVs= selDVs, sep="", nSib= 2)
names(m1) # "top" "MZ"  "DZ"

# ==============
# = 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",]
m1 = xmu_make_TwinSuperModel(mzData= mzData, dzData= dzData, selDVs= selDVs, sep= "", nSib= 2)

# ========================================
# = Cov data (calls xmuTwinSuper_CovCor) =
# ========================================

data(twinData)
mzData =cov(twinData[twinData$zygosity %in% "MZFF", tvars(c("wt","ht"), sep="")], use="complete")
dzData =cov(twinData[twinData$zygosity %in% "DZFF", tvars(c("wt","ht"), sep="")], use="complete")
m1 = xmu_make_TwinSuperModel(mzData= mzData, dzData= dzData, selDVs= "wt", sep= "",
 nSib= 2, numObsMZ = 100, numObsDZ = 100, verbose=TRUE)
class(m1$MZ$fitfunction)[[1]] =="MxFitFunctionML"
dimnames(m1$MZ$data$observed)[[1]]==c("wt1", "wt2")
```

`xmu_match.arg` *Select first item in list of options, while being flexible about choices.*

Description

Like a smart version of `match.arg()`: Handles selecting parameter options when default is a list. Unlike `match.arg()` `xmu_match.arg` allows items not in the list.

Usage

```
xmu_match.arg(x, option_list, check = TRUE)
```

Arguments

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

Value

- one validated option

References

https://github.com/tbates/umx

See Also

- match.arg()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
option_list = c("default", "par.observed", "empirical")

xmu_match.arg("par.observed", option_list)
xmu_match.arg("allow me", option_list, check = FALSE)
xmu_match.arg(option_list, option_list)
option_list = c(NULL, "par.observed", "empirical")
 # fails with NULL!!!!!
xmu_match.arg(option_list, option_list)
option_list = c(NA, "par.observed", "empirical")
xmu_match.arg(option_list, option_list) # use NA instead
option_list = c(TRUE, FALSE, NA)
xmu_match.arg(option_list, option_list) # works with non character
# An example of checking a bad item and stopping
## Not run:
 tmp <- function(x= c("one", "two", "three")) {
```

```
  xmu_match.arg(x, option_list = c("one", "two", "three"))
 }
testthat::expect_true(tmp() == "one")
testthat::expect_error(tmp("bad"))
 tmp <- function(x= c("one", "two", "three")) {
  xmu_match.arg(x, option_list = c("one", "two", "three"), check = FALSE)
 }
testthat::expect_true(tmp("OK") == "OK")
testthat::expect_error(tmp(), NA)

## End(Not run)
```

`xmu_mxData_summary` *Create mxData(type = "summary") with an OpenMx capability guard*

Description

Wrapper around `OpenMx::mxData(..., type = "summary", observedStats = ...)` that calls `xmu_require_summary_mxD` first so CRAN OpenMx fails with a clear message instead of a low-level OpenMx error.

Usage

```
xmu_mxData_summary(numObs, observedStats, ...)
```

Arguments

`numObs`	Number of observations.
`observedStats`	Named list, typically `list(cov = S, useWeight = W, asymCov = V)`.
`...`	Reserved for future `mxData` arguments (currently unused).

Value

An `OpenMx::mxData()` object.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`,

xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_mxRefModels` *Generate reference models safely across OpenMx versions*

Description

`xmu_mxRefModels` wraps `OpenMx::mxRefModels()`, conditionally passing `beginMessage` and `silent` parameters only if supported by the installed version of `OpenMx`.

Usage

```
xmu_mxRefModels(
  x,
  run = FALSE,
  ...,
  beginMessage = FALSE,
  silent = FALSE,
  distribution = "default",
  equateThresholds = TRUE
)
```

Arguments

`x`	An `OpenMx::mxModel()` to generate reference models for.
`run`	Logical. Whether to run the reference models (default = TRUE).
`...`	Extra arguments passed to modern `OpenMx::mxRefModels()`.
`beginMessage`	Logical. Whether to show `Running Saturated...` messages (default = FALSE).
`silent`	Logical. Whether to run reference models silently (default = FALSE).
`distribution`	passed along
`equateThresholds`	passed along

Value

- Reference models list

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_name_from_lavaan_str

Find name for model

Description

Use name if provided. If first line contains a #, uses this line as name. Else use default.

Usage

```
xmu_name_from_lavaan_str(lavaanString = NULL, name = NA, default = "m1")
```

Arguments

lavaanString	A model string, possibly with # model name on line 1.
name	A desired model name (optional).
default	A default name if nothing else found.

Value

- A name string

References

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

See Also

- umxRAM()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thre
xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check()
xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
"m1" == xmu_name_from_lavaan_str("x~~x")
"bob" == xmu_name_from_lavaan_str(name = "bob")
"my_model" == xmu_name_from_lavaan_str("# my model")
```

`xmu_openmx_engine_status` *Detect whether loaded OpenMx is a GenomicMx-capable build*

Description

Capability (not version alone): `options(OpenMx.genomicMx)`, DESCRIPTION field `GenomicMx: yes`, or presence of GenomicMx WLS API markers.

Usage

```
xmu_openmx_engine_status()
```

Value

List with `ok`, `version`, `reason`, `install_msg`.

See Also

`install.OpenMx()`, `xmu_openmx_install_message()`

`xmu_openmx_install_message` *Canonical install recipe when GenomicMx OpenMx is required*

Description

Canonical install recipe when GenomicMx OpenMx is required

Usage

```
xmu_openmx_install_message(feature = NULL)
```

Arguments

`feature`	Optional short name of the feature that failed (for the message header).

Value

Character scalar (multi-line).

See Also

install.OpenMx(), xmu_openmx_engine_status()

---

xmu_PadAndPruneForDefVars *Where all data are missing for a twin, add default values for definition variables, allowing the row to be kept*

Description

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

Usage

```
xmu_PadAndPruneForDefVars(
  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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(),

xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(twinData)
sum(is.na(twinData$ht1))
df = xmu_PadAndPruneForDefVars(twinData, varNames = "ht", defNames = "wt", c("1", "2"))

## End(Not run)
```

---

xmu_parse_completion_context

xmu_parse_completion_context

Description

Parse the current console line buffer to extract the function name, argument name, and argument value prefix under completion.

Usage

```
xmu_parse_completion_context(line, end)
```

Arguments

line	The current command line buffer string.
end	The integer position of the cursor in the line buffer.

Value

A list containing the parsed context (funName, argName, valPrefix, hasQuote, quoteChar), or NULL if not in an argument value completion context.

---

xmu_path2twin *Re-name variables in umxPaths to twin versions*

Description

xmu_path2twin takes a collection of paths that use base variable names, and returns a model with twin names.

Usage

```
xmu_path2twin(paths, thisTwin = 1, sep = "_T")
```

Arguments

paths	A collection of paths using base variable names.
thisTwin	The twin we are making (i.e., "_T1", or "_T2")
sep	The separator (default "_T")

Details

A path like a to b will be returned as a_T1 to b_T1.

Value

- list of relabeled paths

See Also

- umxTwinMaker(), umxRAM()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(),

xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
twin1PathList = c(
umxPath(v1m0 = c("a1", 'c1', "e1")),
umxPath(fromEach = c("a1", 'c1', "e1"), to = "NFC3", values=.2)
)
xmu_path2twin(twin1PathList, thisTwin = 2)
```

---

xmu_path_regex *Re-name variables umxPaths to twin versions*

Description

xmu_path2twin takes a collection of umxPath()s (use base variable names), and returns a model for both twins (and using the expanded variable names).

Usage

```
xmu_path_regex(input, pattern = NA, replacement = NA, ignore = "one")
```

Arguments

input	vector of path labels
pattern	= pattern to match and replace
replacement	= replacement string
ignore	Labels to ignore (reserved words like "one")

Details

A path like `a to b` will be returned as `a_T1 to b_T1`.

Value

- renamed paths

References

tutorials, github

See Also

- `xmu_path2twin()`, `umxTwinMaker()`

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_polycor_blocks()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`, `xmu_show_fit_or_comparison()`, `xmu_simplex_corner()`, `xmu_standardize_ACE()`, `xmu_standardize_ACEcov()`, `xmu_standardize_ACEv()`, `xmu_standardize_CP()`, `xmu_standardize_IP()`, `xmu_standardize_LISREL()`, `xmu_standardize_RAM()`, `xmu_standardize_SexLim()`, `xmu_standardize_Simplex()`, `xmu_start_value_list()`, `xmu_starts()`, `xmu_summary_RAM_group_parameters()`, `xmu_threshold_id_RAM()`, `xmu_threshold_id_twin_check()`, `xmu_topo_sort()`, `xmu_twin_add_WeightMatrices()`, `xmu_twin_check()`, `xmu_twin_get_var_names()`, `xmu_twin_make_def_means_mats_and_alg()`, `xmu_twin_upgrade_selDvs2SelVars()`, `xmu_update_covar()`, `xmu_wls_display_chi()`, `xmu_wls_extract_WV()`

Examples

```
xmu_path_regex(c("a", "one", "b"), pattern = "$", replacement = "_T1")
# "a_T1" "one"  "b_T1"
```

`xmu_print_algebras` *Print algebras from a umx model*

Description

`xmu_print_algebras` adds the results of algebras to a summary

Usage

```
xmu_print_algebras(model, digits = 3, verbose = FALSE)
```

Arguments

`model`	A umx model from which to print algebras.
`digits`	rounding (default = 3)
`verbose`	tell user if no algebras found

Details

Non-user function called by `umxSummary()`

Value

- nothing

See Also

- `umxSummary()`

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_ram_de_apply_censo`
`xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`,

xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
library(mlbench)
data(BostonHousing2)
BostonHousing2$log_crim = log2(BostonHousing2$crim)
BostonHousing2$nox      = BostonHousing2$nox*100
m2 = umxRAM(data = BostonHousing2, "#crime_model
 cmedv ~ log_crim + b1*nox;
 nox   ~ a1*rad + a2*log_crim
i_1 := a1*b1
i_2 := a2*b1"
)
m3 = mxRun(mxModel(m1, mxAlgebra(name= "rtwo", rbind(i_1, i_2))))
m3 = mxRun(mxModel(m3, mxAlgebra(name= "ctwo", cbind(i_1, i_2))))
xmu_print_algebras(m3)

## End(Not run)
```

---

xmu_ram_de_apply_censor_thresholds

Apply RAM DE latent-trait identification

Description

After xmuRAM2Ordinal() (which may set binary mean@0 / resid@1), enforce the latent-trait measurement model: each DEvars base x has A[x_cont, x] = A[x_cens, x] = 1, indicator residuals and means at 0, trait mean/variance on the latent. Fix or free the _cens threshold.

Usage

```
xmu_ram_de_apply_censor_thresholds(
  model,
  fixedCuts,
  contByCens,
  DEvars,
  doubleEntrySuffix = c("_cont", "_cens"),
  sep = ""
)
```

Arguments

model	RAM model after xmuRAM2Ordinal.
fixedCuts	Named numeric cuts (may be empty).
contByCens	Named character cens->cont (unused for equate; kept for call signature).
DEvars	Base names (latents).
doubleEntrySuffix	Suffixes.
sep	Separator (unused but kept for parity).

Value

Modified model.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_ram_de_parse_censor_meta` *Parse censor cuts for single-group RAM DE*

Description

Mirrors `xmu_ace_de_parse_censor_meta` but for a single data.frame and `DEvars`.

Usage

```
xmu_ram_de_parse_censor_meta(
  data,
  DEvars,
  doubleEntrySuffix = c("_cont", "_cens"),
  fixCensorThresholds = c("yes", "auto", "no"),
  censorCuts = NULL
)
```

Arguments

`data`	Data frame with `umxDoubleEntry` attr.
`DEvars`	Base names.
`doubleEntrySuffix`	Suffixes.
`fixCensorThresholds`	`c("yes","auto","no")`.
`censorCuts`	Named numeric cuts or `NULL`.

Value

List with `fixedCuts` (named by cens col), `contByCens`, `sideByCens`.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`,

xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_rclabel_2_bracket_address` *Convert an "A_r1c1"-style label to a bracket address.*

Description

Takes a label like "A_r1c1" and returns "A[1,1]"

Usage

```
xmu_rclabel_2_bracket_address(label, dotprefix = "", suffix = "")
```

Arguments

label	A umx style row col label
dotprefix	Dot address prefix for label (e.g., "ai"
suffix	e.g. "_std" default = "")

Value

- label e.g. "ai[1,1]"

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_rclabel_2_bracket_address(label = "A_r1c1") #A[1,1]
xmu_rclabel_2_bracket_address(label = "A_r10c1")
xmu_rclabel_2_bracket_address(label = "A_r1c1", dotprefix = "model.top")
xmu_rclabel_2_bracket_address("A_r1c1", suffix= "_std")
xmu_rclabel_2_bracket_address("A_r1c1", dotprefix="myModel", suffix="_std")
```

`xmu_relevel_factors` *Relabel Factor Columns in a Data Frame*

Description

This function modifies the levels of specified factor columns in a data.frame where the specified factor columns have potentially collapsed levels based on the criteria provided.

Levels that make up less than a specified proportion of total observations are collapsed into the previous level, providing that a minimum number of levels remains.

The levels of the remaining factor columns are synchronized with the updated levels of the first specified column. Variables named in `cols` must be factors. Note too that prop uses e.g., .1 to stand for 10 percent.

Usage

```
xmu_relevel_factors(df, cols, prop = 0.1, min = 8)
```

Arguments

`df`	A data frame containing the factor columns to be modified.
`cols`	A character vector specifying the names of the factor columns to relabel.
`prop`	A numeric value indicating the minimum proportion of observations for a level (default = .1)
`min`	Integer bounding the minimum remaining number of levels (Default 8).

Value

data.frame with the same structure as the input

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_polycor_blocks()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`, `xmu_show_fit_or_comparison()`, `xmu_simplex_corner()`, `xmu_standardize_ACE()`, `xmu_standardize_ACEcov()`, `xmu_standardize_ACEv()`, `xmu_standardize_CP()`, `xmu_standardize_IP()`, `xmu_standardize_LISREL()`,

xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
df = data.frame(
    group = factor(c("A", "B", "B", "C", "D", "E", "E", "E")),
    score = c(10, 15, 15, 20, 25, 30, 30, 30)
)

# Relabel factor columns
df_releveled = xmu_relevel_factors(df, cols = c("group"), prop = 0.2, min=2)
df_releveled
```

---

xmu_require_summary_mxData

Require OpenMx support for type = "summary" mxData

Description

Stops with a polite install hint if xmu_has_summary_mxData() is FALSE. Call at the entry of any feature that builds modern summary WLS data. Do **not** fall back to legacy type = "none" / "acov" - that API is refused forever in umx.

Usage

```
xmu_require_summary_mxData(where = "This feature")
```

Arguments

where	Short label for the call site (default "This feature").

Value

Invisibly TRUE when supported.

See Also

xmu_has_summary_mxData(), xmu_mxData_summary()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(),

xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_robust_ML_fit` *xmu_robust_ML_fit*

Description

Computes Satorra-Bentler (1994) robust scaling corrections and Brosseau-Liard & Savalei (2012) robust fit indices (CFI, TLI, RMSEA) for Maximum Likelihood models with raw continuous complete or incomplete (FIML) data.

Usage

```
xmu_robust_ML_fit(model, refModels = NULL)
```

Arguments

`model`	An `OpenMx::mxModel()` fitted with ML.
`refModels`	Reference models (Independence and Saturated models) to calculate incremental fit indices.

Details

The robust scaling correction factor c is computed using the difference between the trace of the saturated model and the trace of the fitted model, scaled by the degrees of freedom:

$$c = \frac{\mathrm{tr}(J_{\mathrm{saturated}}^{-1} B_{\mathrm{saturated}}) - \mathrm{tr}(J_{\mathrm{model}}^{-1} B_{\mathrm{model}})}{d}$$

where $J = H/2$ is the observed information matrix (reconciled from the Hessian H of $-2 \ln L$) and $B = G^T G$ is the outer product of the individual log-likelihood row gradients. Because OpenMx optimizes $-2 \ln L$ and G contains the row gradients of $-2 \ln L_i$ (which is 2 times the score s_i), the matrix product $(H/2)^{-1} B$ scales to $4J^{-1}K$. Thus, dividing by $4 \times d$ scales the difference of these traces back to the Satorra-Bentler / Yuan-Bentler expectation scale.

Robust RMSEA is computed using Equation (5) of Brosseau-Liard, Savalei, and Li (2012):

$$\mathrm{RMSEA}_{\mathrm{robust}} = \sqrt{\max\left(0, \frac{\chi^2_{\mathrm{raw}}/N}{df} - \frac{c_{\mathrm{model}}}{N}\right)}$$

Value

A list containing scaling factors and adjusted robust fit statistics, or NULL if robust adjustments are not supported.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(),

xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check()
xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(),
xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(),
xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_robust_WLS_fit` *Robust WLS fit statistics: Satorra-Bentler (2010) and Savalei (2021)*

Description

Computes robust fit indices and scaled test statistics for Weighted Least Squares (WLS / DWLS) models fitted in OpenMx. This is the engine behind WLS routing in `umxSummary()` and `umxCompare()`.

The function **intentionally uses two statistic families**:

- **Display omnibus test** (`Chi`, `ChiDoF`, `p`): always Satorra-Bentler (2010) scaled WLS χ^2, regardless of whether manifests are continuous or ordinal.
- **Robust incremental/absolute indices** (`CFI`, `TLI`, `RMSEA`): Satorra-Bentler (2010) for continuous WLS; Savalei (2021) catML mean-and-variance corrections for ordinal/categorical WLS.

Independence (null) baseline fit is computed **natively in R** (not via `mxRun`) so that asymptotic-covariance-only inputs (e.g. Genomic SEM / summary-statistic pipelines) do not trigger OpenMx validation errors.

Usage

```
xmu_robust_WLS_fit(model)
```

Arguments

`model`	A fitted WLS `OpenMx::mxModel()` with `implied_jacobian`.

Details

Prerequisites. The fitted model must expose `model@output$implied_jacobian`, plus WLS `asymCov` and `useWeight` (or equivalent) with consistent moment rownames. Without the Jacobian, the function stops with an error.

Shared pipeline (both branches).

1. Extract raw target WLS χ^2_{raw}, test df, and implied Jacobian Δ_{target}.
2. Align Δ, Γ (`asymCov`), and W (`useWeight`) to a common moment ordering via `xmu_WLS_align_jacobian()` and `xmu_WLS_align_weight()`.
3. Build independence Jacobian Δ_{ind} with `xmu_build_independence_jacobian()` on the same moment set.
4. Compute native independence discrepancy $\chi^2_{ind,raw} = d'_{ind} W d_{ind}$, where d_{ind} is the observed summary vector with means, variances, and thresholds zeroed (independence pattern).
5. Scale matrices to sample size n for raw-data models.

6. Compute Satorra-Bentler trace scaling factors

$$U = \Gamma W - \Gamma W \Delta (\Delta' W \Delta)^{-1} \Delta' W$$

$$c = \mathrm{tr}(U)/df$$

for target (c_{model}) and independence (c_{null}) models.

7. Form display statistics: $\chi^2_{SB} = \chi^2_{raw}/c_{model}$, with $p = 1 - F_{\chi^2_{df}}(\chi^2_{SB})$.

Branch A - Continuous WLS (`correction = "SB2010"`). Detected when no manifest is `ordered` or `factor` on raw data (`xmu_is_ordinal_WLS()` returns `FALSE`). Robust `CFI`, `TLI`, and `RMSEA` are derived from SB-scaled target and independence χ^2 and their dfs. Attributes `c_model` and `c_null` hold the SB trace factors.

Branch B - Ordinal / categorical WLS (`correction = "Savalei2021"`). Detected when at least one manifest is `ordered` or `factor`. Robust indices use the Brosseau-Liard / Savalei noncentrality form with catML ingredients evaluated at **fixed** converged WLS estimates (no re-optimization):

catML discrepancy XX_3. `xmu_catml_discrepancy_at_WLS()` builds a correlation-matrix ML scaffold (`xmu_catml_eval_model()`), runs reference models, and returns:

- `fMlTarget` = XX_3 (target minus saturated correlation fit)
- `fMlNull` = $XX_{3,null}$ (independence minus saturated)

Savalei scaling $\hat{c}_3$. `xmu_savalei_polycor_blocks()` assembles OpenMx-native polycor blocks:

V catML expected information at model-implied correlation (`xmu_catml_wls_v()`); not `asymCov` and not `solve(useWeight)`.

Γ_{poly} Sample moment covariance: $n^2\times$ polycor block of per-observation `asymCov`.

W_{poly} WLS weight polycor block at sample scale: $n\times$ per-observation `useWeight`.

Δ_{poly} Polycor rows of aligned `implied_jacobian` (threshold rows excluded).

E^{-1} $(\Delta'_{full} W_{full} \Delta_{full})^{-1}$ from the identically scaled full weight matrix.

Then `xmu_savalei_scaling_factor()` computes

$$\hat{c}_3 = \frac{\mathrm{tr}(W_i U' V W_i U \, \Gamma_{poly})}{df_3}$$

with $W_i U = I - \Delta_{poly} E^{-1} \Delta'_{poly} W_{poly}$, and `xmu_savalei_null_scaling_factor()` computes $\hat{c}_{3,null} = \mathrm{tr}(\Gamma_{poly})/df_{3,null}$.

Degrees of freedom: df_3 = WLS model test df; $df_{3,null}$ = number of polycorrelation moments (`length(polyNames)`). Jacobian-rank shortcuts (`xmu_catml_df3_diagnostic()`) are diagnostic only.

Robust indices: `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_tli()`, `xmu_savalei_fit_rmsea()` with XX_3, $\hat{c}_3$, $\hat{c}_{3,null}$. CFI is clamped to $[0, 1]$; TLI may slightly exceed 1 with excellent fit. If Savalei scaling fails, $\hat{c}_3$ falls back to SB c_{model} (and null analogously).

Cutoff guidance. For ordinal WLS, Hu & Bentler (1999) conventional cutoffs (e.g. CFI ≥ 0.95, RMSEA ≤ 0.06) apply to the **robust** `CFI`/`TLI`/`RMSEA` reported here-not to raw WLS χ^2. Display χ^2 and p remain SB-scaled WLS omnibus tests.

Software note. Numeric values of $\hat{c}_3$ may differ from other SEM packages because Jacobians and parameterizations differ across engines; umx uses OpenMx `implied_jacobian` and native `asymCov`/`useWeight` throughout. The goal is statistically correct evaluation under OpenMx, not bitwise replication of external implementations.

Value

A named list designed to patch directly into umxSummary() slots:

CFI, TLI, RMSEA Robust fit indices (SB2010 or Savalei2021 branch).

Chi SB-scaled target χ^2 (display omnibus statistic).

ChiDoF Target model test degrees of freedom.

p Two-sided p-value for Chi under χ^2_{df}.

scalingFactor, scalingFactorNull SB trace factors c_{model} and c_{null} (always computed).

Attributes (read by umxSummary() for footnotes):

correction "SB2010" or "Savalei2021".

c_model Scaling factor used for robust target noncentrality (SB c or Savalei $\hat{c}_3$).

c_null Scaling factor for null/independence noncentrality.

fMlTarget, fMlNull catML XX_3 components when Savalei branch runs; NA_real_ otherwise.

References

- Satorra, A., & Bentler, P. M. (2010). Ensuring positiveness of the scaled difference chi-square test statistic. *Psychometrika*, **75**(2), 243–269.
- Savalei, V. (2021). Improving fit indices in SEM with categorical data. *Multivariate Behavioral Research*, **56**(3), 390–407.
- Brosseau-Liard, P. E., & Savalei, V. (2012). Adjusting incremental fit indices for nonnormality. *Multivariate Behavioral Research*, **47**(5), 647–677.
- Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis. *Structural Equation Modeling*, **6**, 1–55.

See Also

umxSummary(), umxCompare(), xmu_build_independence_jacobian, xmu_is_ordinal_WLS(), xmu_catml_discrepancy_at_WLS(), xmu_savalei_polycor_blocks()

Other Model Summary and Comparison: umx, umxCompare(), umxEquate(), umxMI(), umxReduce(), umxSetParameters(), umxSummary(), xmu_compare_WLS()

---

xmu_safe_run_summary *Safely run and summarize a model*

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", "ordinal", "search"),
  summary = !umx_set_silent(silent = TRUE),
  std = "default",
  comparison = TRUE,
  digits = 3,
  intervals = FALSE,
  returning = c("model", "summary"),
  refModels = NULL
)
```

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: "ordinal", "search"
summary	Whether to print model summary (default = autoRun).
std	What to print in summary. "default" = the object's summary default. FALSE = raw, TRUE = standardize, NULL = omit parameter table.
comparison	Toggle to allow not making comparison, even if second model is provided (more flexible in programming).
digits	Rounding precision in tables and plots
intervals	whether to run intervals or not (default FALSE)
returning	What to return (default, the run model)
refModels	whether to run refModels or not (default NULL)

Value

- OpenMx::mxModel()

See Also

- OpenMx::mxTryHard()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(),

xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
tmp = mtcars
tmp$disp = tmp$disp/100
m1 = umxRAM("tim", data = tmp,
 umxPath(c("wt", "disp"), to = "mpg"),
 umxPath("wt", with = "disp"),
 umxPath(v.m. = c("wt", "disp", "mpg"))
)
m2 = umxModify(m1, "wt_to_mpg")

# Summary ignored if run is false
xmu_safe_run_summary(m1, autoRun = FALSE, summary = TRUE)
# Run, no summary
xmu_safe_run_summary(m1, autoRun = TRUE, summary = FALSE)
# Default summary is just fit string
xmu_safe_run_summary(m1, autoRun = TRUE, summary = TRUE)
# Show std parameters
xmu_safe_run_summary(m1, autoRun = TRUE, summary = TRUE, std = TRUE)
# Run + Summary + comparison
xmu_safe_run_summary(m1, m2, autoRun = TRUE, summary = TRUE, intervals = TRUE)
# Run + Summary + no comparison
xmu_safe_run_summary(m1, m2, autoRun = TRUE, summary = TRUE, std = TRUE, comparison= FALSE)

## End(Not run)
```

`xmu_savalei_fit_cfi` *Robust CFI from mean-and-variance corrected noncentrality (Savalei 2021 form)*

Description

Implements the Brosseau-Liard / Savalei noncentrality formulation for robust CFI using catML discrepancy XX_3 at fixed WLS estimates and scaling factors $\hat{c}_3$, $\hat{c}_{3,null}$. Used for ordinal WLS in `xmu_robust_WLS_fit()`; continuous WLS falls back to SB-scaled χ^2.

Usage

```
xmu_savalei_fit_cfi(x2, df, x2Null, dfNull, cHat = 1, cHatNull = 1)
```

Arguments

`x2`	Target catML discrepancy (or SB-scaled χ^2 for continuous).
`df`	Target model degrees of freedom.
`x2Null`	Independence/null catML discrepancy.
`dfNull`	Null model degrees of freedom.
`cHat`	Target scaling factor $\hat{c}_3$.
`cHatNull`	Null scaling factor $\hat{c}_{3,null}$.

Value

Robust CFI in $[0, 1]$ (values slightly outside may occur for TLI).

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`,

xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_savalei_fit_rmsea` *Robust RMSEA from mean-and-variance corrected noncentrality (Savalei 2021 form)*

Description

$$\text{RMSEA}_{robust} = \sqrt{\max\left(\frac{XX_3/n}{df_3} - \frac{\hat{c}_3}{n}, 0\right)}$$

Uses catML XX_3 and $\hat{c}_3$ for ordinal WLS; continuous WLS uses SB-scaled target χ^2 and SB trace scaling.

Usage

```
xmu_savalei_fit_rmsea(x2, df, nVal, cHat = 1, gGroups = 1L)
```

Arguments

`x2`	Target discrepancy (XX_3 or SB-scaled χ^2).
`df`	Target degrees of freedom.
`nVal`	Sample size n.
`cHat`	Scaling factor $\hat{c}_3$ (or SB c for continuous).
`gGroups`	Number of groups (default 1).

Value

Robust RMSEA (0 if $df = 0$).

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_bl xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_savalei_fit_tli` *Robust TLI from mean-and-variance corrected noncentrality (Savalei 2021 form)*

Description

Companion to xmu_savalei_fit_cfi(). TLI is not bounded above by 1 under this correction; values slightly above 1 can occur with excellent fit.

Usage

```
xmu_savalei_fit_tli(x2, df, x2Null, dfNull, cHat = 1, cHatNull = 1)
```

Arguments

x2	Target catML discrepancy (or SB-scaled χ^2 for continuous).
df	Target model degrees of freedom.
x2Null	Independence/null catML discrepancy.
dfNull	Null model degrees of freedom.
cHat	Target scaling factor $\hat{c}_3$.
cHatNull	Null scaling factor $\hat{c}_{3,null}$.

Value

Robust TLI.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_savalei_null_scaling_factor` *Savalei (2021) independence/null scaling factor* $\hat{c}_3, null$

Description

Computes the null-model mean-and-variance correction used alongside $\hat{c}_3$ in robust CFI and TLI:

$$\hat{c}_{3,null} = \frac{\mathrm{tr}(\Gamma_{poly})}{df_{3,null}}$$

where $df_{3,null}$ is the number of polycorrelation moments (all off-diagonal correlations free under the independence baseline in the correlation subsystem).

Usage

```
xmu_savalei_null_scaling_factor(gammaPoly, dfNullCatMl)
```

Arguments

`gammaPoly`	Polycorrelation sample-moment covariance Γ_{poly}.
`dfNullCatMl`	Null degrees of freedom $df_{3,null}$ (typically `length(polyNames)`).

Value

Numeric $\hat{c}_{3,null}$, or `NA_real_` on failure.

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`,

xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_savalei_polycor_blocks` *Build OpenMx-native polycorrelation blocks for the Savalei sandwich*

Description

Assembles the Δ, Γ, and W ingredients for the polycorrelation subsystem of Savalei (2021) $\hat{c}_3$ (`c.hat3`), using matrices extracted from the **same** fitted OpenMx WLS model at mutually consistent scales.

Usage

```
xmu_savalei_polycor_blocks(
  asymCovAlignedPerObs,
  weightMatAlignedPerObs,
  jacTargetAligned,
  polyNames,
  nVal
)
```

Arguments

`asymCovAlignedPerObs`	Aligned per-observation asymptotic covariance (`asymCov` subset to common moments, before n scaling).
`weightMatAlignedPerObs`	Aligned per-observation WLS weight (before n scaling).
`jacTargetAligned`	Full aligned implied Jacobian (all moment rows).
`polyNames`	Character vector of polycorrelation moment names.
`nVal`	Sample size n (from `model@data@numObs`).

Details

Why a separate builder? Savalei's correction is evaluated on polycorrelations only (threshold rows excluded). Each matrix must refer to the same moment ordering (rownames from aligned `asymCov`) and the same sample size scaling.

Objects returned:

`jacPoly` (Δ_{poly}) Rows of `model@output$implied_jacobian` corresponding to polycorrelation moments (names not ending in `t1`, `t2`, etc.). Columns are model parameters in OpenMx order. This is $\partial s/\partial\theta$ for the polycor block.

`gammaPoly` (Γ_{poly}) Asymptotic covariance of the sample polycorrelation moments at sample scale: $\Gamma_{poly} = n^2 \times V_{poly,per-obs}$, where $V_{poly,per-obs}$ is the polycorrelation block of OpenMx `observedStats$asymCov` *before* the global n scaling applied to the full aligned matrices. This is the moment sampling covariance; it is **not** `solve(useWeight)`.

`wPolyWiU` (W_{poly} **in** W_iU) Polycorrelation block of the WLS weight matrix at sample scale: $W_{poly} = n \times W_{poly,per-obs}$, from `useWeight`. This is paired with $E^{-1} = (\Delta'_{full} W_{full} \Delta_{full})^{-1}$ computed from the identically scaled full weight matrix.

Value

List with `gammaPoly`, `wPolyWiU`, and `jacPoly`, or `NULL` if any block is empty.

See Also

`xmu_savalei_scaling_factor()`, `xmu_WLS_polycor_names()`

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`, `xmu_dot_move_ranks()`, `xmu_dot_rank_str()`, `xmu_extract_column()`, `xmu_get_CI()`, `xmu_has_summary_mxData()`, `xmu_invert_matrix()`, `xmu_is_ordinal_WLS()`, `xmu_lavaan_process_group()`, `xmu_make_TwinSuperModel()`, `xmu_make_bin_cont_pair_data()`, `xmu_make_mxData()`, `xmu_match.arg()`, `xmu_mxData_summary()`, `xmu_mxRefModels()`, `xmu_name_from_lavaan_str()`, `xmu_path2twin()`, `xmu_path_regex()`, `xmu_print_algebras()`, `xmu_ram_de_apply_censor_thresholds()`, `xmu_ram_de_parse_censor_meta()`, `xmu_rclabel_2_bracket_address()`, `xmu_relevel_factors()`, `xmu_require_summary_mxData()`, `xmu_robust_ML_fit()`, `xmu_safe_run_summary()`, `xmu_savalei_fit_cfi()`, `xmu_savalei_fit_rmsea()`, `xmu_savalei_fit_tli()`, `xmu_savalei_null_scaling_factor()`, `xmu_savalei_scaling_factor()`, `xmu_set_sep_from_suffix()`, `xmu_show_fit_or_comparison()`, `xmu_simplex_corner()`, `xmu_standardize_ACE()`, `xmu_standardize_ACEcov()`, `xmu_standardize_ACEv()`,

xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_savalei_scaling_factor` *Savalei (2021) target scaling factor ĉ_3 for ordinal WLS*

Description

Computes the mean-and-variance correction used in robust CFI, TLI, and RMSEA for ordinal/categorical WLS models (Savalei, 2021; Brosseau-Liard & Savalei, 2012 noncentrality form).

Usage

```
xmu_savalei_scaling_factor(
  jacFull,
  jacPoly,
  vCatMlPoly,
  gammaPoly,
  wPolyWiU,
  wFullScaled,
  dfCatMl
)
```

Arguments

`jacFull`	Full aligned WLS Jacobian Δ_{full} (all moments).
`jacPoly`	Polycorrelation rows Δ_{poly}.
`vCatMlPoly`	catML V matrix (polycor block).
`gammaPoly`	Sample moment covariance Γ_{poly}.
`wPolyWiU`	WLS weight polycor block W_{poly} for the W_iU sandwich.
`wFullScaled`	Full aligned weight matrix at sample scale ($n\times$ per-obs).
`dfCatMl`	Target model test degrees of freedom df_3.

Details

Formula. Let $W_iU = I - \Delta_{poly}E^{-1}\Delta'_{poly}W_{poly}$. Then

$$\hat{c}_3 = \frac{\mathrm{tr}\big(W_iU'\,V\,W_iU\,\Gamma_{poly}\big)}{df_3}$$

where:

- V = catML expected information (xmu_catml_wls_v())
- Γ_{poly} = sample moment covariance from OpenMx asymCov
- W_{poly} = WLS useWeight polycor block at sample scale
- E^{-1} = $(\Delta'_{full} W_{full} \Delta_{full})^{-1}$ from the same fitted model and weight scaling
- df_3 = target model test df (not a polycor Jacobian rank shortcut)

Design note. V comes from the catML correlation scaffold; Γ and W come from the ordinal WLS fit. They are different statistical objects serving different roles in Savalei's correction. Do not substitute asymCov for V, or solve(useWeight) for Γ.

Value

Numeric $\hat{c}_3$, or NA_real_ if computation fails.

References

Savalei, V. (2021). Improving fit indices in SEM with categorical data. *Multivariate Behavioral Research*, **56**(3), 390–407.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(),

xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor

xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_set_sep_from_suffix(sep = "_T", suffix = "deprecated")
```

`xmu_show_fit_or_comparison`

Show model logLik of model or print comparison table

Description

Just a helper to show the fit of a model or print a comparison table.

Usage

```
xmu_show_fit_or_comparison(model, comparison = NULL, digits = 2)
```

Arguments

model	an OpenMx::mxModel() to report on
comparison	If not NULL, used as comparison model
digits	(default = 2)

Value

None

See Also

- umxSummary()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(),

xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
xmu_show_fit_or_comparison(model, comparison, digits=3)

## End(Not run)
```

`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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
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_standardize_ACE *xmu_standardize_ACE*

Description

Standardize an ACE model *BUT* you probably want umx_standardize().

Usage

```
xmu_standardize_ACE(model, ...)
```

Arguments

model	an umxACE() model to standardize
...	Other options

Value

- Standardized ACE umxACE() model

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
require(umx)
data(twinData)
selDVs = c("bmi1", "bmi2")
mzData = twinData[twinData$zygosity %in% "MZFF", selDVs]
dzData = twinData[twinData$zygosity %in% "DZFF", selDVs]
m1     = umxACE(selDVs = selDVs, dzData = dzData, mzData = mzData)
std    = xmu_standardize_ACE(m1)

## End(Not run)
```

`xmu_standardize_ACEcov`

xmu_standardize_ACEcov

Description

Standardize an ACE model with covariates

Usage

```
xmu_standardize_ACEcov(model, ...)
```

Arguments

`model`	an `umxACEcov()` model to standardize
`...`	Other options

Value

- Standardized `umxACEcov()` model

References

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

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`,

xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
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 = xmu_standardize_ACEcov(m1)

## End(Not run)
```

---

xmu_standardize_ACEv *Standardize an ACE variance components model (ACEv)*

Description

xmu_standardize_ACE allows umx_standardize to standardize an ACE variance components model.

Usage

```
xmu_standardize_ACEv(model, ...)
```

Arguments

model	An umxACEv() model to standardize.
...	Other parameters.

Value

- A standardized umxACEv() model.

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
require(umx)
data(twinData)
```

```
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
m1  = umxACEv(selDVs = "bmi", sep="", dzData = dzData, mzData = mzData)
std = umx_standardize(m1)

## End(Not run)
```

`xmu_standardize_CP` *Function to standardize a common pathway model*

Description

You probably want `umx_standardize()`. This function simply inserts the standardized CP components into the ai ci ei and as cs es matrices

Usage

```
xmu_standardize_CP(model, ...)
```

Arguments

`model`	an `umxCP()` model to standardize
`...`	Other options

Value

- standardized `umxCP()` model

References

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

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`, `xmuTwinSuper_NoBinary()`, `xmuTwinUpgradeMeansToCovariateModel()`, `xmu_CI_merge()`, `xmu_CI_stash()`, `xmu_DF_to_mxData_TypeCov()`, `xmu_PadAndPruneForDefVars()`, `xmu_WLS_align_jacobian()`, `xmu_WLS_align_weight()`, `xmu_WLS_polycor_names()`, `xmu_ace_de_apply_censor_thresholds()`, `xmu_ace_de_parse_censor_meta()`, `xmu_ace_de_parse_censor_rule()`, `xmu_ace_de_release_binary_v1()`, `xmu_bracket_address2rclabel()`, `xmu_catml_df3_diagnostic()`, `xmu_catml_discrepancy_at_WLS()`, `xmu_catml_eval_model()`, `xmu_catml_implied_correlation()`, `xmu_catml_wls_v()`, `xmu_cell_is_on()`, `xmu_check_levels_identical()`, `xmu_check_needs_means()`, `xmu_check_variance()`, `xmu_clean_label()`, `xmu_compare_constraint_note()`, `xmu_compare_robust_ML()`, `xmu_data_missing()`, `xmu_data_swap_a_block()`, `xmu_describe_data_WLS()`, `xmu_dot_make_paths()`, `xmu_dot_make_residuals()`, `xmu_dot_maker()`,

xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
selDVs = c("gff", "fc", "qol", "hap", "sat", "AD")
m1 = umxCP(selDVs = selDVs, nFac = 3, data=GFF, zyg="zyg_2grp")
m2 = xmu_standardize_CP(m1)

## End(Not run)
```

---

xmu_standardize_IP *non-user: Standardize an IP model*

Description

You probably want umx_standardize(). This function simply copies the standardized IP components into the ai ci ei and as cs es matrices

Usage

```
xmu_standardize_IP(model, ...)
```

Arguments

model	an umxIP() model to standardize
...	Other options

Value

- standardized IP umxIP() model

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

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

## End(Not run)
```

---

xmu_standardize_LISREL

xmu_standardize_LISREL

Description

Standardize a LISREL model

Usage

```
xmu_standardize_LISREL(model, ...)
```

Arguments

model	The OpenMx::mxModel() you wish to standardize
...	Other options

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_standardize_RAM *Standardize a Structural Model (not for end users)*

Description

You probably want umx_standardize(), not this.

Usage

```
xmu_standardize_RAM(model, ...)
```

Arguments

model	The OpenMx::mxModel() you wish to standardize
...	Other options

Details

xmu_standardize_RAM takes a RAM-style model, and returns standardized version.

References

https://github.com/tbates/umx

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
require(umx)
data(demoOneFactor)
manifests = names(demoOneFactor)

m1 = umxRAM("std_ex", data = demoOneFactor, type = "cov",
 umxPath("G", to = manifests),
 umxPath(var = manifests),
 umxPath(var = "G", fixedAt = 1.0)
)

m1 = xmu_standardize_RAM(m1)
m1 = umx_standardize(m1)
umxSummary(m1)


## End(Not run)
```

`xmu_standardize_SexLim` *Standardize a SexLim model*

Description

`xmu_standardize_SexLim` would move standardized Sexlim values into raw cells, but can't as these are algebras.

Usage

```
xmu_standardize_SexLim(model, ...)
```

Arguments

`model`	an `umxSexLim()` model to standardize
`...`	Other options

Value

- standardized `umxSexLim()` model

See Also

Other xmu internal not for end user: `mx_make()`, `umxModel()`, `umxRenameMatrix()`, `umx_APA_pval()`, `umx_fun_mean_sd()`, `umx_get_bracket_addresses()`, `umx_make()`, `umx_standardize()`, `umx_string_to_algebra()`, `xmuHasSquareBrackets()`, `xmuLabel_MATRIX_Model()`, `xmuLabel_Matrix()`, `xmuLabel_RAM_Model()`, `xmuMI()`, `xmuMakeOneHeadedPathsFromPathList()`, `xmuMakeTwoHeadedPathsFromPathList()`, `xmuMaxLevels()`, `xmuMinLevels()`, `xmuPropagateLabels()`, `xmuRAM2Ordinal()`, `xmuTwinSuper_Continuous()`,

xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

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

## End(Not run)
```

`xmu_standardize_Simplex`

Standardize a Simplex twin model

Description

xmu_standardize_Simplex

Usage

```
xmu_standardize_Simplex(model, ...)
```

Arguments

model	an umxSimplex() model to standardize
...	Other options

Value

- Standardized Simplex umxSimplex() model

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(iqdat)
mzData = subset(iqdat, zygosity == "MZ")
dzData = subset(iqdat, zygosity == "DZ")
m1  = umxSimplex(selDVs = paste0("IQ_age", 1:4), sep = "_T",
   dzData = dzData, mzData = mzData, tryHard = "yes")
std = xmu_standardize_Simplex(m1)
```

```
## End(Not run)
```

`xmu_starts` *Helper providing boilerplate start values for means and variance in twin models*

Description

`xmu_starts` can handle several common/boilerplate situations in which means and variance start values are used in twin models.

Usage

```
xmu_starts(
  mzData,
  dzData,
  selVars = selVars,
  sep = NULL,
  equateMeans = NULL,
  nSib,
  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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

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)
xmu_starts(mzData, dzData, selVars=selDVs, nSib= 2, sep="", equateMeans=TRUE, varForm="Cholesky")

# Variance instead of SD
round(var(dzData[,tvars(selDVs, "")], na.rm=TRUE)/3,3)
xmu_starts(mzData, dzData, selVars = selDVs, nSib = 2, sep= "",
 equateMeans= TRUE, varForm= "Cholesky", SD= FALSE)

# one variable
xmu_starts(mzData, dzData, selVars= "wt", nSib = 2, sep="", equateMeans = TRUE)
```

```
# Ordinal/continuous mix
data(twinData)
twinData= umx_scale_wide_twin_data(data=twinData,varsToScale="wt",sep= "")
# Cut BMI column to form ordinal obesity variables
cuts     = quantile(twinData[, "bmi1"], probs = c(.5, .8), na.rm = TRUE)
obLevels = c('normal', 'overweight', 'obese')
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] = umxFactor(twinData[, ordDVs])
mzData = twinData[twinData$zygosity %in% "MZFF",]
dzData = twinData[twinData$zygosity %in% "DZFF",]
xmu_starts(mzData, dzData, selVars = c("wt","obese"), sep= "",
 nSib= 2, equateMeans = TRUE, SD= FALSE)

xmu_starts(mxData(mzData, type="raw"), mxData(mzData, type="raw"),
   selVars = c("wt","obese"), sep= "", nSib= 2, equateMeans = TRUE, SD= FALSE)

# ==============
# = Three sibs =
# ==============
data(twinData)
twinData$wt3 = twinData$wt2
twinData$ht3 = twinData$ht2
selDVs = c("wt", "ht")
mzData = twinData[twinData$zygosity %in%  "MZFF", ]
dzData = twinData[twinData$zygosity %in%  "DZFF", ]

xmu_starts(mzData, dzData, selVars=selDVs, sep="", nSib=3, equateMeans=TRUE)
xmu_starts(mzData, dzData, selVars=selDVs, sep="", nSib=3, equateMeans=FALSE)
```

`xmu_start_value_list` *Make start values*

Description

Purpose: Create startvalues for OpenMx paths use cases umx:::xmuStart_value_list(1) xmuValues(1) # 1 value, varying around 1, with sd of .1 xmuValues(1, n=letters) # length(letters) start values, with mean 1 and sd .1 xmuValues(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: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_summary_RAM_group_parameters

Order and group the parameters in a RAM summary

Description

Makes understanding complex model output easier by grouping parameters are type: residuals, latent variance, factor loading etc.

Usage

```
xmu_summary_RAM_group_parameters(
  model,
  paramTable,
  means = FALSE,
  residuals = FALSE
)
```

Arguments

model	the model containing the parameters.
paramTable	The parameter table.
means	Whether to show the means (FALSE)
residuals	Whether to show the residuals (FALSE)

Value

- Sorted parameter table

See Also

- umxSummary()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(),

xmu_starts(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(demoOneFactor)
manifests = names(demoOneFactor)
m1 = umxRAM("One Factor", data = demoOneFactor,
 umxPath("G", to = manifests),
 umxPath(v.m. = manifests),
 umxPath(v1m0 = "G")
)
tmp = umxSummary(m1, means=FALSE, residuals = FALSE)
xmu_summary_RAM_group_parameters(m1, paramTable = tmp,  means= FALSE, residuals= FALSE)

## End(Not run)
```

---

xmu_threshold_id_RAM *Enforce or check Mehta/binary identification for threshold RAM models*

Description

For binary manifests: latent mean fixed at 0 and residual variance fixed at 1. For ordinal manifests (>2 levels, Mehta): mean and residual variance free. Continuous manifests are left unchanged. Does not rewrite path labels.

Usage

```
xmu_threshold_id_RAM(model, action = c("fix", "check"), verbose = TRUE)
```

Arguments

model	An OpenMx RAM OpenMx::mxModel() with data and (typically) thresholds.
action	"fix" (default) corrects free/values; "check" only warns.
verbose	If TRUE (default), emit one line when parameters are fixed/freed or when check finds problems. Silent when already correct.

Value

The model (modified if action = "fix").

See Also

umxThresholdMatrix(), xmuRAM2Ordinal(), umxRAM()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_threshold_id_twin_check` *Verify twin-model threshold identification (means / binary Vtot constraint)*

Description

Does not modify the model or rewrite twin mean labels. Emits at most one warning if binary means are free, ordinal means are fixed, or the binary Vtot==1 machinery is missing.

Usage

```
xmu_threshold_id_twin_check(model, fullVars, verbose = TRUE)
```

Arguments

model	A twin super-model with top (typically from xmu_make_TwinSuperModel()).
fullVars	Character vector of full twin variable names (e.g. wt_T1, wt_T2).
verbose	If TRUE, warn on problems; silent when OK.

Value

Invisibly, a character vector of problem strings (empty if OK).

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_topo_sort *Topologically sort a graph based on from and to paths*

Description

Performs a topological sort on a set of nodes based on directional paths. In structural equation modeling, processing sources (exogenous variables) first and sinks (endogenous variables) last often ensures the A matrix is strictly lower triangular (Recursive (acyclic) models are those whose path matrix can be arranged lower triangular). This both speeds optimization (a triangular A inverts as a finite series (Bollen, 1989; Kahn, 1962; Neale et al., 2016). OpenMx detects when A is strictly lower-triangular and uses a finite series to invert (I - A) rather than (slower) general inversion.

However, it also improves discovery: small but real path weights (such as we encounter in umxGSEM()) get clean gradient signal instead of stalling at local minima in the dense coupled inverse. umx implements this rearrangement in umxRAM() and umxRAM_DE().

Usage

```
xmu_topo_sort(from, to, nodes)
```

Arguments

from	Character vector of source nodes.
to	Character vector of destination nodes.
nodes	Character vector of all nodes to sort.

Value

A character vector of the nodes sorted topologically. Nodes not involved in paths are appended.

References

- Bollen, K. A. (1989). *Structural Equations with Latent Variables*. Wiley.
- Kahn, A. B. (1962). Topological sorting of large networks. *Communications of the ACM*, **5**, 558-562. doi:10.1145/368996.369025
- Neale, M. C., Hunter, M. D., Pritikin, J. N., Zahery, M., Brick, T. R., Kirkpatrick, R. M., Estabrook, R., Bates, T. C., Maes, H. H., & Boker, S. M. (2016). OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika*. doi:10.1007/s1133601494358.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(),

xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_twin_add_WeightMatrices` *Add weight matrices to twin models.*

Description

Add weight models (MZw, DZw) with matrices (e.g. mzWeightMatrix) to a twin model, and update `mxFitFunctionMultigroup`. This yields a weighted model with vector objective.

To weight objective functions in OpenMx, you specify a container model that applies the weights m1 is the model with no weights, but with "vector = TRUE" option added to the FIML objective. This option makes FIML return individual likelihoods for each row of the data (rather than a single -2LL value for the model) You then optimize weighted versions of these likelihoods by building additional models containing weight data and an algebra that multiplies the likelihoods from the first model by the weight vector.

Usage

```
xmu_twin_add_WeightMatrices(model, mzWeights = NULL, dzWeights = NULL)
```

Arguments

`model`	umx-style twin model
`mzWeights`	data for MZ weights matrix
`dzWeights`	data for DZ weights matrix

Value

- model

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
tmp = umx_make_twin_data_nice(data=twinData, sep="", zygosity="zygosity", numbering= 1:2)
m1  = umxACE(selDVs = "wt", data = tmp, dzData = "DZFF", mzData = "MZFF", autoRun= FALSE)
m1$MZ$fitfunction$vector= TRUE

tmp = xmu_twin_add_WeightMatrices(m1,
 mzWeights= rnorm(nrow(m1$MZ$data$observed)),
 dzWeights= rnorm(nrow(m1$DZ$data$observed))
)
```

---

xmu_twin_check *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

None

References

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

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(),

xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
library(umx)
data(twinData)
mzData = subset(twinData, zygosity == "MZFF")
dzData = subset(twinData, zygosity == "MZFF")
xmu_twin_check(selDVs = c("wt", "ht"), dzData = dzData, mzData = mzData,
 sep = "", enforceSep = TRUE)
xmu_twin_check(selDVs = c("wt", "ht"), dzData = dzData, mzData = mzData,
 sep = "", enforceSep = FALSE)
xmu_twin_check(selDVs = 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(selDVs = c("wt", "ht"), dzData = dzData, mzData = mzData, enforceSep = TRUE)
# 3. stop on a factor with sep = NULL

## End(Not run)
```

---

xmu_twin_get_var_names

Not for user: pull variable names from a twin model

Description

Barely useful, but justified perhaps by centralizing trimming the "_T1" off, and returning just twin 1.

Usage

```
xmu_twin_get_var_names(
  model,
  source = c("expCovMZ", "observed"),
  trim = TRUE,
  twinOneOnly = TRUE
)
```

Arguments

model	A model to get the variables from
source	Whether to access the dimnames of the "expCovMZ" or the names of the "observed" data (will include covariates)
trim	Whether to trim the suffix (TRUE)
twinOneOnly	Whether to return on the names for twin 1 (i.e., unique names)

Value

- variable names from twin model

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(

xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
## Not run:
data(twinData) # ?twinData from Australian twins.
twinData[, c("ht1", "ht2")] = twinData[, c("ht1", "ht2")] * 10
mzData = twinData[twinData$zygosity %in% "MZFF", ]
dzData = twinData[twinData$zygosity %in% "DZFF", ]
m1 = umxACE(selDVs= "ht", sep= "", dzData= dzData, mzData= mzData, autoRun= FALSE)
selVars = xmu_twin_get_var_names(m1, source = "expCovMZ", trim = TRUE, twinOneOnly = TRUE) # "ht"
umx_check(selVars == "ht")
xmu_twin_get_var_names(m1, source= "expCovMZ", trim= FALSE, twinOneOnly= FALSE) # "ht1" "ht2"
selVars = xmu_twin_get_var_names(m1, source= "observed", trim= TRUE, twinOneOnly= TRUE)# "ht"
nVar = length(selVars)
umx_check(nVar == 1)

## End(Not run)
```

---

xmu_twin_make_def_means_mats_and_alg

Make the matrices and algebras for definition-based means models

---

Description

not-for-end-user helper for means in twin models. Returns matrices for each definition variable, and an algebra to compute means.

Usage

```
xmu_twin_make_def_means_mats_and_alg(baseCovs, fullVars, nSib, sep)
```

Arguments

baseCovs	base names of the DVs, e.g. "age"
fullVars	full names of the DVs, e.g. "E_T1"
nSib	how many siblings - typically 2
sep	in twin variable, i.e., "_T"

Value

matrices and an algebra

See Also

- xmuTwinUpgradeMeansToCovariateModel()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
# xmu_twin_make_def_means_mats_and_alg(baseCovs= baseCovs,
#   fullVars = fullVars, nSib = nSib, sep= sep)
```

---

xmu_twin_upgrade_selDvs2SelVars

Upgrade selDVs to selVars

Description

Just a helper to go from "wt" to "wt_T1" contingent on sep not being null

Usage

```
xmu_twin_upgrade_selDvs2SelVars(selDVs, sep, nSib)
```

Arguments

selDVs	with wt or wt_T1
sep	either "" etc., or NULL
nSib	wideness of data

Value

list of wt_T1 wt_T2 etc.

See Also

- umx

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
xmu_twin_upgrade_selDvs2SelVars("wt", NULL, 2)
```

`xmu_update_covar` *Update covariates in twin data sets to 99999 if missing and corresponding twin phenotype to NA*

Description

Takes a dataframe with twin data and updates the covariates to 99999 if missing and the corresponding twin phenotype to NA. This avoids removing rows with missing data in the covariates and does not affect the estimation.

Usage

```
xmu_update_covar(data, covar, pheno, sep = "_T")
```

Arguments

data	A data.frame() to convert
covar	The covariates.
pheno	The phenotypes affected by covariates.
sep	The separator used in the column names (default = "_T")

Value

- dataframe with updated covariates and phenotypes

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(),

xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Examples

```
# data(docData)
# df = docData
# Add some missing data
# df$varA1_T1[1:5] <- NA
# df <- xmu_update_covar(df, covar = "varA1", pheno = "varB1")
# head(df)
```

---

xmu_WLS_align_jacobian

Align WLS implied Jacobian to asymptotic-covariance moment order

---

Description

OpenMx may store `implied_jacobian` rows in a different order than `observedStats$asymCov` (means after covariances in the Jacobian; mixed order in `asymCov`). This helper reorders Jacobian rows to match `asymCov` rownames and returns the intersection set used by all SB and Savalei matrix algebra in `xmu_robust_WLS_fit()`.

Usage

```
xmu_WLS_align_jacobian(jacMat, asymCovMat, numCovsVal)
```

Arguments

`jacMat`	Implied Jacobian Δ from `model@output$implied_jacobian`.
`asymCovMat`	Asymptotic covariance matrix Γ with moment rownames.
`numCovsVal`	Number of covariance/polycorrelation moments (for mean/cov block reordering when Jacobian row count equals full moment count).

Value

List with `jac` (aligned Jacobian) and `commonNames` (moment labels shared across Δ, Γ, and W).

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(, xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

`xmu_WLS_align_weight` *Subset and align WLS weight matrix to common moment set*

Description

Extracts the W block corresponding to `commonNames` from OpenMx `useWeight` (or a diagonal vector). Accepts either a full matrix (with or without dimnames) or a named diagonal vector (expanded to `diag()`).

Usage

```
xmu_WLS_align_weight(weightMat, commonNames, fullRowNames)
```

Arguments

weightMat	WLS weight matrix or diagonal vector from MxData.
commonNames	Moment names shared by aligned Δ and Γ.
fullRowNames	Full moment names from the asymptotic covariance matrix.

Value

Weight matrix W subset to commonNames.

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat, xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

---

xmu_wls_display_chi *Display chi-square and AFIs for a WLS model (shared by summary and compare)*

Description

When output$implied_jacobian is present, returns Satorra-Bentler (2010) scaled statistics from xmu_robust_WLS_fit() (including saturated df=0 convention: Chi=0, CFI/TLI=1, RMSEA=0). Otherwise falls back to OpenMx summary() Browne residual chi-square.

Usage

```
xmu_wls_display_chi(model)
```

Arguments

model	A run MxModel (typically WLS/DWLS).

Value

Named list with Chi, ChiDoF, p, CFI, TLI, RMSEA, and source one of "SB2010", "saturated", "OpenMx".

See Also

xmu_robust_WLS_fit(), xmu_compare_WLS(), umxSummary()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(),

xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_extract_WV()

---

xmu_wls_extract_WV *Extract WLS useWeight (W) and asymCov (Gamma) from an MxModel*

Description

Modern only: reads observedStats = list(cov=, useWeight=, asymCov=). OpenMx legacy type="acov" / "none" / MxDataLegacyWLS is refused forever.

Usage

```
xmu_wls_extract_WV(model, stop_if_missing = TRUE)
```

Arguments

model	An OpenMx::mxModel() with data (or multigroup children with data).
stop_if_missing	If TRUE, stop when either matrix is missing.

Details

For multigroup models with data only on submodels, returns **block-diagonal** useWeight and asymCov across groups (row/column names prefixed by group name when more than one group contributes).

Value

List with useWeight, asymCov, optional cov, and nGroups (integer).

See Also

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_WLS_polycor_names(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(), xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(),

xmu_make_bin_cont_pair_data(), xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check(), xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi()

`xmu_WLS_polycor_names` *Polycorrelation moment names from WLS asymptotic-covariance labels*

Description

Filters OpenMx WLS moment rownames to the polycorrelation subsystem used in Savalei (2021) corrections. Threshold rows (names ending in `t1`, `t2`, ...) are excluded; means and variances are excluded by the naming pattern (they do not match the polycor regex filter).

Usage

```
xmu_WLS_polycor_names(momentNames)
```

Arguments

`momentNames`	Character vector of `asymCov` rownames (aligned set).

Value

Character vector of polycorrelation-only moment names.

See Also

xmu_savalei_polycor_blocks()

Other xmu internal not for end user: mx_make(), umxModel(), umxRenameMatrix(), umx_APA_pval(), umx_fun_mean_sd(), umx_get_bracket_addresses(), umx_make(), umx_standardize(), umx_string_to_algebra(), xmuHasSquareBrackets(), xmuLabel_MATRIX_Model(), xmuLabel_Matrix(), xmuLabel_RAM_Model(), xmuMI(), xmuMakeOneHeadedPathsFromPathList(), xmuMakeTwoHeadedPathsFromPathList(), xmuMaxLevels(), xmuMinLevels(), xmuPropagateLabels(), xmuRAM2Ordinal(), xmuTwinSuper_Continuous(), xmuTwinSuper_NoBinary(), xmuTwinUpgradeMeansToCovariateModel(), xmu_CI_merge(), xmu_CI_stash(), xmu_DF_to_mxData_TypeCov(), xmu_PadAndPruneForDefVars(), xmu_WLS_align_jacobian(), xmu_WLS_align_weight(), xmu_ace_de_apply_censor_thresholds(), xmu_ace_de_parse_censor_meta(),

xmu_ace_de_parse_censor_rule(), xmu_ace_de_release_binary_v1(), xmu_bracket_address2rclabel(), xmu_catml_df3_diagnostic(), xmu_catml_discrepancy_at_WLS(), xmu_catml_eval_model(), xmu_catml_implied_correlation(), xmu_catml_wls_v(), xmu_cell_is_on(), xmu_check_levels_identical(), xmu_check_needs_means(), xmu_check_variance(), xmu_clean_label(), xmu_compare_constraint_note(), xmu_compare_robust_ML(), xmu_data_missing(), xmu_data_swap_a_block(), xmu_describe_data_WLS(), xmu_dot_make_paths(), xmu_dot_make_residuals(), xmu_dot_maker(), xmu_dot_move_ranks(), xmu_dot_rank_str(), xmu_extract_column(), xmu_get_CI(), xmu_has_summary_mxData(), xmu_invert_matrix(), xmu_is_ordinal_WLS(), xmu_lavaan_process_group(), xmu_make_TwinSuperModel(), xmu_make_bin_cont_pair_dat xmu_make_mxData(), xmu_match.arg(), xmu_mxData_summary(), xmu_mxRefModels(), xmu_name_from_lavaan_str(), xmu_path2twin(), xmu_path_regex(), xmu_print_algebras(), xmu_ram_de_apply_censor_thresholds(), xmu_ram_de_parse_censor_meta(), xmu_rclabel_2_bracket_address(), xmu_relevel_factors(), xmu_require_summary_mxData(), xmu_robust_ML_fit(), xmu_safe_run_summary(), xmu_savalei_fit_cfi(), xmu_savalei_fit_rmsea(), xmu_savalei_fit_tli(), xmu_savalei_null_scaling_factor(), xmu_savalei_polycor_blocks(), xmu_savalei_scaling_factor(), xmu_set_sep_from_suffix(), xmu_show_fit_or_comparison(), xmu_simplex_corner(), xmu_standardize_ACE(), xmu_standardize_ACEcov(), xmu_standardize_ACEv(), xmu_standardize_CP(), xmu_standardize_IP(), xmu_standardize_LISREL(), xmu_standardize_RAM(), xmu_standardize_SexLim(), xmu_standardize_Simplex(), xmu_start_value_list(), xmu_starts(), xmu_summary_RAM_group_parameters(), xmu_threshold_id_RAM(), xmu_threshold_id_twin_check() xmu_topo_sort(), xmu_twin_add_WeightMatrices(), xmu_twin_check(), xmu_twin_get_var_names(), xmu_twin_make_def_means_mats_and_alg(), xmu_twin_upgrade_selDvs2SelVars(), xmu_update_covar(), xmu_wls_display_chi(), xmu_wls_extract_WV()

Index

∗ **Advanced Model Building Functions**
umx, 103
umxAlgebra, 133
umxFixAll, 187
umxJiggle, 224
umxRAM_GLM, 290
umxRun, 300
umxThresholdMatrix, 346
umxUnexplainedCausalNexus, 354
xmuLabel, 494
xmuValues, 516
∗ **CLPM Functions**
umxCLPM, 144
∗ **Check or test**
umx, 103
umx_check_names, 365
umx_is_class, 388
umx_is_endogenous, 390
umx_is_exogenous, 391
umx_is_numeric, 396
umx_is_ordered, 397
∗ **Core Model Building Functions**
umx, 103
umxLISREL, 229
umxMatrix, 232
umxModify, 240
umxPath, 248
umxRAM, 278
umxRAM_DE, 287
umxSuperModel, 343
∗ **Data Functions**
noNAs, 61
prolific_anonymize, 79
prolific_check_ID, 80
prolific_read_demog, 81
prolific_scoring_stub, 83
umx, 103
umx_as_numeric, 362
umx_cont_2_quantiles, 370
umx_lower2full, 403
umx_make_fake_data, 409
umx_make_GxE_data, 411
umx_make_MR_data, 413
umx_make_raw_from_cov, 414
umx_make_TwinData, 417
umx_merge_randomized_columns, 423
umx_polychoric, 432
umx_polypairwise, 433
umx_polytriowise, 434
umx_read_lower, 437
umx_rename, 439
umx_reorder, 442
umx_score_scale, 453
umx_select_valid, 457
umx_stack, 472
umx_strings2numeric, 474
umxFactor, 182
umxGSEM_sim_snps, 205
umxHetCor, 219
∗ **File Functions**
dl_from_dropbox, 16
umx, 103
umx_file_load_pseudo, 376
umx_make_sql_from_excel, 415
umx_move_file, 424
umx_open, 429
umx_rename_file, 440
umx_write_to_clipboard, 488
∗ **GSEM**
Anthro_LDSC, 12
plot.umx_GWAS, 71
Psych_LDSC, 84
umxGSEM, 193
umxGSEM_dl_tutorial_files, 196
umxGSEM_GWAS, 197
umxGSEM_label_ldsc, 200
umxGSEM_ldsc, 201
umxGSEM_munge, 203

umxGSEM_std, 206
umxGSEM_sumstats, 207
umxSummary.list, 313
umxSummary.MxDataStatic, 315
∗ **Get and set**
umx, 103
umx_detect_cores, 373
umx_get_checkpoint, 381
umx_get_options, 382
umx_scale_reliabilities, 448
umx_set_auto_plot, 458
umx_set_auto_run, 459
umx_set_checkpoint, 459
umx_set_condensed_slots, 461
umx_set_cores, 462
umx_set_data_variance_check, 463
umx_set_dollar_symbol, 464
umx_set_optimization_options, 465
umx_set_optimizer, 466
umx_set_plot_file_suffix, 467
umx_set_plot_format, 468
umx_set_separator, 469
umx_set_silent, 470
umx_set_table_format, 471
∗ **Graphviz**
xmu_dot_define_shapes, 550
xmu_dot_make_paths, 552
xmu_dot_make_residuals, 554
xmu_dot_maker, 551
xmu_dot_mat2dot, 556
xmu_dot_rank, 560
∗ **Miscellaneous Functions**
deg2rad, 15
rad2deg, 86
umx_check_parallel, 368
umx_complete_dollar, 369
umxBrownie, 138
∗ **Miscellaneous Stats Functions**
FishersMethod, 40
geometric_mean, 41
harmonic_mean, 50
oddsratio, 62
reliability, 87
SE_from_p, 92
umx, 103
umx_apply, 360
umx_cor, 371
umx_means, 422
umx_r_test, 446
umx_round, 445
umx_scale, 447
umx_var, 482
umxCov2cor, 152
umxHetCor, 219
umxParan, 247
umxWeightedAIC, 356
∗ **Miscellaneous Utility Functions**
install.OpenMx, 53
libs, 56
qm, 85
umx, 103
umx_array_shift, 361
umx_data, 373
umx_find_object, 377
umx_lower.tri, 402
umx_msg, 425
umx_open_CRAN_page, 430
umx_pad, 431
umx_print, 435
umx_scholastica_report, 450
umx_tail, 478
umx_wide2long, 484
umx_wide4lmer, 487
umxLav2RAM, 225
umxModelNames, 239
umxRAM2Lav, 286
umxVersion, 355
∗ **Model Comparison**
umxMeasurementInvariance, 235
∗ **Model Summary and Comparison**
umx, 103
umxCompare, 146
umxEquate, 173
umxMI, 236
umxReduce, 292
umxSetParameters, 301
umxSummary, 313
xmu_compare_WLS, 543
xmu_robust_WLS_fit, 605
∗ **Model Summary**
umxSummary.list, 313
umxSummary.MxDataStatic, 315
umxSummary.MxLISRELModel, 316
∗ **Plotting functions**
ggAddR, 45
plot.MxLISRELModel, 63

plot.MxModel, 65
plot.MxModelTwinMaker, 68
umx, 103
umxPlot, 252
umxPlotACE, 254
umxPlotACE_DE, 258
umxPlotACE_GLM, 259
umxPlotACEcov, 255
umxPlotACEv, 257
umxPlotCP, 260
umxPlotDoC, 262
umxPlotFun, 264
umxPlotGxE, 266
umxPlotGxEbiv, 267
umxPlotIP, 269
umxPlotPredict, 270
umxPlotSexLim, 271
umxPlotSimplex, 273
* **Reporting Functions**
aic_weights, 11
plot.umx_GWAS, 71
umx, 103
umx_aggregate, 357
umx_time, 479
umxAPA, 134
umxFactorScores, 184
umxGetLatents, 188
umxGetManifests, 189
umxGetModel, 190
umxGetParameters, 191
umxParameters, 245
* **Reporting functions**
extractAIC.MxModel, 18
loadings, 57
loadings.MxModel, 58
residuals.MxModel, 88
RMSEA, 89
RMSEA.MxModel, 90
RMSEA.summary.mxmodel, 91
tmx_show, 96
tmx_show.MxMatrix, 97
umxCI, 138
umxCI_boot, 141
umxConfint, 150
umxExpCov, 180
umxExpMeans, 181
umxFitIndices, 185
umxRotate, 297
umxSummaryACE_DE, 327
* **String Functions**
tvars, 101
umx, 103
umx_explode, 374
umx_explode_twin_names, 375
umx_grep, 382
umx_names, 427
umx_rot, 444
umx_str_chars, 476
umx_str_from_object, 477
umx_trim, 481
* **Summary functions**
umxSummary.MxModel, 317
umxSummaryACEcov, 324
umxSummaryCP, 330
umxSummaryGxE, 333
umxSummaryIP, 337
umxSummaryMRDoC, 338
* **Super-easy helpers**
umx, 103
umxEFA, 169
umxTwoStage, 352
* **Teaching and Testing functions**
tmx_show.MxModel, 99
umxDiagnose, 158
umxPower, 275
* **Teaching and testing Functions**
tmx_genotypic_effect, 93
tmx_is.identified, 95
umx, 103
* **Test**
umx_check, 363
umx_check_model, 364
umx_check_names, 365
umx_check_OS, 367
umx_has_been_run, 384
umx_has_CIs, 385
umx_has_means, 386
umx_has_square_brackets, 387
umx_is_cov, 389
umx_is_GSEM, 392
umx_is_LISREL, 393
umx_is_MxData, 394
umx_is_MxMatrix, 394
umx_is_MxModel, 395
umx_is_RAM, 398
* **Twin Data functions**

umx, 103
umx_long2wide, 400
umx_make_twin_data_nice, 421
umx_make_TwinData, 417
umx_residualize, 443
umx_scale_wide_twin_data, 449
umx_wide2longTwinData, 486
umx_yj_wide_twin_data, 489
* **Twin Modeling Functions**
power.ACE.test, 72
umx, 103
umx_make_double_entry_data, 407
umxACE, 107
umxACE_DE, 124
umxACE_GLM, 130
umxACEcov, 115
umxACEv, 118
umxCloud, 142
umxCP, 153
umxDiffMZ, 160
umxDiscTwin, 162
umxDoC, 164
umxDoCp, 167
umxGxE, 210
umxGxE_window, 216
umxGxEbiv, 214
umxIP, 220
umxMRDoC, 243
umxReduce, 292
umxReduceACE, 293
umxReduceGxE, 295
umxRotate.MxModelCP, 298
umxSexLim, 303
umxSimplex, 308
umxSummarizeTwinData, 311
umxSummaryACE, 322
umxSummaryACE_GLM, 329
umxSummaryACEv, 326
umxSummaryDoC, 331
umxSummaryGxEbiv, 335
umxSummarySexLim, 339
umxSummarySimplex, 342
umxTwinMaker, 349
* **Utility Functions**
umx_tutorials, 481
* **datasets**
Anthro_LDSC, 12
docData, 17
Fischbein_wt, 39
GFF, 43
HSwls, 51
iqdat, 55
Psych_LDSC, 84
umx, 103
us_skinfold_data, 490
* **financial functions**
fin_options_Greeks, 19
fin_options_LeapSimulate, 20
fin_options_plotGreeks, 21
fin_options_teach, 22
fin_stock_CAGR, 24
fin_stock_justifiedPE, 25
fin_stock_target, 27
fin_stock_ticker, 28
fin_stock_valuation, 29
fin_tax_FIF, 30
fin_tax_NI, 31
fin_value_CAGR, 32
fin_value_CarryCost, 33
fin_value_interest, 35
fin_value_NPV, 37
fin_value_percent, 38
* **github**
gh_feature_branch, 46
gh_message_search, 47
gh_open_PR_url, 48
xgh_check_base_name, 491
xgh_get_current_branch, 492
* **umx S3 functions**
plot.MxLISRELModel, 63
* **xmu internal functions**
xmu_is_wls, 572
* **xmu internal not for end user**
mx_make, 59
umx_APA_pval, 359
umx_fun_mean_sd, 378
umx_get_bracket_addresses, 379
umx_make, 405
umx_standardize, 473
umx_string_to_algebra, 475
umxModel, 238
umxRenameMatrix, 296
xmu_ace_de_apply_censor_thresholds, 517
xmu_ace_de_parse_censor_meta, 519
xmu_ace_de_parse_censor_rule, 520

xmu_ace_de_release_binary_v1, 521
xmu_bracket_address2rclabel, 522
xmu_catml_df3_diagnostic, 524
xmu_catml_discrepancy_at_WLS, 525
xmu_catml_eval_model, 526
xmu_catml_implied_correlation, 528
xmu_catml_wls_v, 529
xmu_cell_is_on, 530
xmu_check_levels_identical, 532
xmu_check_needs_means, 533
xmu_check_variance, 535
xmu_CI_merge, 536
xmu_CI_stash, 538
xmu_clean_label, 539
xmu_compare_constraint_note, 540
xmu_compare_robust_ML, 542
xmu_data_missing, 544
xmu_data_swap_a_block, 546
xmu_describe_data_WLS, 547
xmu_DF_to_mxData_TypeCov, 549
xmu_dot_make_paths, 552
xmu_dot_make_residuals, 554
xmu_dot_maker, 551
xmu_dot_move_ranks, 559
xmu_dot_rank_str, 561
xmu_extract_column, 563
xmu_get_CI, 564
xmu_has_summary_mxData, 569
xmu_invert_matrix, 570
xmu_is_ordinal_WLS, 571
xmu_lavaan_process_group, 573
xmu_make_bin_cont_pair_data, 574
xmu_make_mxData, 576
xmu_make_TwinSuperModel, 578
xmu_match.arg, 583
xmu_mxData_summary, 585
xmu_mxRefModels, 586
xmu_name_from_lavaan_str, 587
xmu_PadAndPruneForDefVars, 590
xmu_path2twin, 592
xmu_path_regex, 593
xmu_print_algebras, 595
xmu_ram_de_apply_censor_thresholds, 596
xmu_ram_de_parse_censor_meta, 598
xmu_rclabel_2_bracket_address, 599
xmu_relevel_factors, 600
xmu_require_summary_mxData, 602
xmu_robust_ML_fit, 603
xmu_safe_run_summary, 607
xmu_savalei_fit_cfi, 610
xmu_savalei_fit_rmsea, 611
xmu_savalei_fit_tli, 612
xmu_savalei_null_scaling_factor, 614
xmu_savalei_polycor_blocks, 615
xmu_savalei_scaling_factor, 617
xmu_set_sep_from_suffix, 619
xmu_show_fit_or_comparison, 620
xmu_simplex_corner, 621
xmu_standardize_ACE, 622
xmu_standardize_ACEcov, 624
xmu_standardize_ACEv, 625
xmu_standardize_CP, 627
xmu_standardize_IP, 628
xmu_standardize_LISREL, 629
xmu_standardize_RAM, 630
xmu_standardize_SexLim, 632
xmu_standardize_Simplex, 633
xmu_start_value_list, 637
xmu_starts, 635
xmu_summary_RAM_group_parameters, 638
xmu_threshold_id_RAM, 640
xmu_threshold_id_twin_check, 641
xmu_topo_sort, 642
xmu_twin_add_WeightMatrices, 644
xmu_twin_check, 645
xmu_twin_get_var_names, 647
xmu_twin_make_def_means_mats_and_alg, 649
xmu_twin_upgrade_selDvs2SelVars, 650
xmu_update_covar, 652
xmu_WLS_align_jacobian, 653
xmu_WLS_align_weight, 654
xmu_wls_display_chi, 655
xmu_wls_extract_WV, 657
xmu_WLS_polycor_names, 658
xmuHasSquareBrackets, 493
xmuLabel_Matrix, 496
xmuLabel_MATRIX_Model, 498
xmuLabel_RAM_Model, 500
xmuMakeOneHeadedPathsFromPathList, 501
xmuMakeTwoHeadedPathsFromPathList,

503
xmuMaxLevels, 504
xmuMI, 505
xmuMinLevels, 506
xmuPropagateLabels, 507
xmuRAM2Ordinal, 509
xmuTwinSuper_Continuous, 510
xmuTwinSuper_NoBinary, 512
xmuTwinUpgradeMeansToCovariateModel, 514
∗ **zAdvanced Helpers**
umx, 103

aggregate(), *50*, *357, 358*
AIC(), *12*, *18*, *357*
aic_weights, 11
aic_weights(), *72*, *104*, *136*, *185*, *188, 189*, *191, 192*, *246*, *358*, *480*
Anthro_LDSC, 12, *17*, *40*, *45*, *52*, *55*, *72*, *84*, *105*, *195–198*, *201, 202*, *204*, *206*, *209*, *314*, *316*, *491*
apply(), *360*
attributes(), *447*

base::as.Date(), *451*
base::colMeans(), *360*
base::rowSums(), *360*
base::trimws(), *481*
bquote(), *253*
bucks, 13

calculateStrictSb, 14
complete.cases(), *247*, *545*
cor.test(), *134*, *372*
cov(), *483*
cov2cor(), *152*, *415*
cowplot::draw_label(), *253*
cumsum(), *360*

data.frame, *80*, *82*, *136*
data.frame(), *62*, *219*, *362*, *383*, *397*, *421, 422*, *574*, *576*, *652*
deg2rad, 15
deg2rad(), *86*, *138*, *369*
DiagrammeR::DiagrammeR(), *63*, *65*, *67*, *468*
dimnames(), *365*
dl_from_dropbox, 16
dl_from_dropbox(), *105*, *377*, *416*, *425*, *430*, *441*, *488*

docData, *13*, 17, *40*, *45*, *52*, *55*, *84*, *105*, *491*

extractAIC.MxModel, 18
extractAIC.MxModel(), *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*

factanal(), *57, 58*, *170*, *172*
factor(), *183*
file.rename(), *425*
fin_options_Greeks, 19
fin_options_Greeks(), *21–24*, *26, 27*, *29–34*, *36*, *38, 39*
fin_options_LeapSimulate, 20
fin_options_LeapSimulate(), *20*, *22–24*, *26, 27*, *29–34*, *36*, *38, 39*
fin_options_plotGreeks, 21
fin_options_plotGreeks(), *20, 21*, *23, 24*, *26, 27*, *29–34*, *36*, *38, 39*
fin_options_teach, 22
fin_options_teach(), *20–22*, *24*, *26, 27*, *29–34*, *36*, *38, 39*
fin_resolveVolatility, 23
fin_stock_CAGR, 24
fin_stock_CAGR(), *20–23*, *26, 27*, *29–34*, *36*, *38, 39*
fin_stock_justifiedPE, 25
fin_stock_justifiedPE(), *20–24*, *27*, *29–34*, *36*, *38, 39*
fin_stock_target, 27
fin_stock_target(), *20–24*, *26*, *29–34*, *36*, *38, 39*
fin_stock_ticker, 28
fin_stock_ticker(), *20–24*, *26, 27*, *30–34*, *36*, *38, 39*
fin_stock_valuation, 29
fin_stock_valuation(), *20–24*, *26, 27*, *29*, *31–34*, *36*, *38, 39*
fin_tax_FIF, 30
fin_tax_FIF(), *20–24*, *26, 27*, *29, 30*, *32–34*, *36*, *38, 39*
fin_tax_NI, 31
fin_tax_NI(), *20–24*, *26, 27*, *29–31*, *33, 34*, *36*, *38, 39*
fin_value_CAGR, 32
fin_value_CAGR(), *20–24*, *26, 27*, *29–32*, *34*, *36*, *38, 39*
fin_value_CarryCost, 33

fin_value_CarryCost(), *20–24*, *26, 27*, *29–33*, *36*, *38, 39*
fin_value_interest, 35
fin_value_interest(), *13, 14*, *20–24*, *26, 27*, *29–34*, *38, 39*, *464*
fin_value_NPV, 37
fin_value_NPV(), *20–24*, *26, 27*, *29–34*, *36*, *39*
fin_value_percent, 38
fin_value_percent(), *14*, *20–24*, *26, 27*, *29–34*, *36*, *38*, *70, 71*, *77*
Fischbein_wt, *13*, *17*, 39, *45*, *52*, *55*, *84*, *105*, *491*
FishersMethod, 40
FishersMethod(), *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
formula(), *443*

geometric_mean, 41
geometric_mean(), *41*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
GFF, *13*, *17*, *40*, 43, *52*, *55*, *84*, *105*, *491*
ggAddR, 45
ggAddR(), *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*, *274*
ggplot2::annotate(), *253*
ggplot2::geom_point(), *252*
ggplot2::geom_smooth(), *252*
ggplot2::ggplot(), *252*, *254*, *271*
ggplot2::labs(), *252*
ggplot2::stat_function(), *264, 265*
ggplot2::theme_gray(), *253*
gh_feature_branch, 46
gh_feature_branch(), *48, 49*, *492*
gh_message_search, 47
gh_message_search(), *47*, *49*, *492*
gh_open_PR_url, 48
gh_open_PR_url(), *47, 48*, *492*
glm(), *136*
grep, *427*
grep(), *383*, *428*

harmonic_mean, 50
harmonic_mean(), *41, 42*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
HSwls, *13*, *17*, *40*, *45*, 51, *55*, *84*, *105*, *491*

install.OpenMx, 53
install.OpenMx(), *56*, *59, 60*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*, *589, 590*
install.packages(), *54*, *56*
iqdat, *13*, *17*, *40*, *45*, *52*, 55, *84*, *105*, *491*

lavaan string syntax, *287*
lavaan::HolzingerSwineford1939, *51*
library(), *56*
libs, 56
libs(), *54*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*
lm(), *136*, *253*, *281*
loadings, *57*, 57
loadings(), *18*, *58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
loadings.MxModel, *57*, 58
loadings.MxModel(), *18*, *57*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
logLik(), *18*
lower.tri(), *402, 403*

MASS::mvrnorm(), *414, 415*
match.arg(), *583, 584*
matrix(), *402*, *438*
Matrix::nearPD(), *194*
mean(), *41, 42*
merge(), *401*
mx_make, 59
mx_make(), *53, 54*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
mxConstraint(), *291*

mxData, *316*
mxData(), *291*, *365*
mxFitFunctionWLS, *316*
mxMatrix(), *291*
MxModel(), *291*
mxPath(), *291*
mxRefModels(), *290*

names, *427*
namez, *440*
namez *(*umx_names*)*, 427
namez(), *103*, *246*, *365*, *383*
nlme::lme(), *136*, *162*
nlme::nlme(), *162*
noNAs, 61
noNAs(), *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*

oddsratio, 62
oddsratio(), *41, 42*, *50*, *76, 77*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
OpenMx::HS.ability.data, *51, 52*
OpenMx::imxRobustSE(), *319*
OpenMx::mxAlgebra(), *134*, *227*, *475*
OpenMx::mxCheckIdentification(), *95*, *320*, *355*
OpenMx::mxCI(), *138, 139*, *150, 151*
OpenMx::mxCompare(), *543*
OpenMx::mxConstraint(), *148*, *227*, *540*
OpenMx::mxData, *315*
OpenMx::mxData(), *280*, *288*, *300*, *397*, *533*, *549*, *563*, *576*, *585*
OpenMx::mxExpectationNormal(), *580*
OpenMx::mxFactor(), *182, 183*, *370*
OpenMx::mxFactorScores(), *172*, *185*
OpenMx::mxFitFunctionML(), *580*
OpenMx::mxFitFunctionMultigroup(), *343, 344*
OpenMx::mxFitFunctionWLS(), *547, 548*, *580*
OpenMx::mxGenerateData(), *410*
OpenMx::mxMatrix(), *224*, *233, 234*, *395*, *403, 404*, *494–497*, *516*
OpenMx::mxMI(), *237*
OpenMx::mxModel(), *18*, *64*, *66*, *88*, *90, 91*, *110*, *117*, *121*, *127*, *139–141*, *146, 147*, *150*, *156*, *159*, *166*, *171*, *173, 174*, *180, 181*, *185*, *187*, *192, 193*, *195*, *212*, *215*, *223*, *229–231*, *237–242*, *244*, *246*, *254*, *256*, *259*, *272*, *281, 282*, *289*, *292, 293*, *300–302*, *305*, *310*, *313*, *317, 318*, *322, 323*, *325–328*, *330*, *332*, *334*, *336, 337*, *340*, *342–344*, *353*, *356*, *384–386*, *390–393*, *395*, *398*, *445*, *473*, *479*, *483*, *494, 495*, *498–500*, *505*, *507*, *509*, *516*, *525–528*, *531*, *537, 538*, *541, 542*, *551*, *565*, *571, 572*, *581*, *586*, *603*, *605*, *608*, *620*, *630, 631*, *640*, *657*
OpenMx::mxPath(), *250, 251*, *494, 495*
OpenMx::mxPower(), *74*
OpenMx::mxRefModels(), *525*, *586*
OpenMx::mxRename(), *239*
OpenMx::mxRun(), *150*, *241*, *300*
OpenMx::mxSE(), *151*
OpenMx::mxThreshold(), *347*
OpenMx::mxTryHard(), *608*
OpenMx::omxAssignFirstParameters(), *302*
OpenMx::omxBrownie(), *138*
OpenMx::omxGetParameters(), *192*
OpenMx::omxSetParameters(), *494*
OpenMx::summary.MxModel(), *185*

packageVersion(), *356*
parallel::detectCores(), *374*
parameters *(*umxParameters*)*, 245
parameters(), *192*, *282*, *290*
paste0, *427*
plot *(*plot.MxModel*)*, 65
plot(), *65*, *67*, *69*, *127*, *156*, *215*, *223*, *255, 256*, *259*, *261*, *267, 268*, *270*, *274*, *282*, *290*, *310*, *331, 332*, *334*, *336*, *338, 339*, *409*, *557*
plot.MxLISRELModel, 63
plot.MxLISRELModel(), *46*, *67*, *69*, *105*, *231*, *254–256*, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*, *274*, *317*
plot.MxModel, 65
plot.MxModel(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*, *274*
plot.MxModelACE *(*umxPlotACE*)*, 254
plot.MxModelACE(), *324*

plot.MxModelACE_DE *(*umxPlotACE_DE*)*, 258
plot.MxModelACE_GLM *(*umxPlotACE_GLM*)*, 259
plot.MxModelACEcov *(*umxPlotACEcov*)*, 255
plot.MxModelACEv *(*umxPlotACEv*)*, 257
plot.MxModelCP *(*umxPlotCP*)*, 260
plot.MxModelDoC *(*umxPlotDoC*)*, 262
plot.MxModelDoC(), *17*, *332*
plot.MxModelGxE *(*umxPlotGxE*)*, 266
plot.MxModelGxEbiv *(*umxPlotGxEbiv*)*, 267
plot.MxModelIP *(*umxPlotIP*)*, 269
plot.MxModelSexLim *(*umxPlotSexLim*)*, 271
plot.MxModelSimplex *(*umxPlotSimplex*)*, 273
plot.MxModelTwinMaker, 68
plot.MxModelTwinMaker(), *46*, *65*, *67*, *105*, *254–256*, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*, *274*
plot.percent, 70
plot.umx_GWAS, 71
plot.umx_GWAS(), *12, 13*, *84*, *104*, *136*, *185*, *188, 189*, *191, 192*, *195*, *197, 198*, *201, 202*, *204*, *206*, *209*, *246*, *314*, *316*, *358*, *480*
plotmath(), *136*
power.ACE.test, 72
power.ACE.test(), *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *276*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*, *417*
print *(*bucks*)*, 13
print(), *77–79*
print.oddsratio, 76
print.percent, 77
print.reliability, 78
print.reliability(), *87*
print.RMSEA, 78
prolific_anonymize, 79
prolific_anonymize(), *62*, *81–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
prolific_check_ID, 80
prolific_check_ID(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
prolific_read_demog, 81
prolific_read_demog(), *62*, *80, 81*, *83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
prolific_scoring_stub, 83
prolific_scoring_stub(), *62*, *80–82*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
Psych_LDSC, *12, 13*, *17*, *40*, *45*, *52*, *55*, *72*, 84, *105*, *193–198*, *201–204*, *206*, *209*, *313, 314*, *316*, *491*

qm, 85
qm(), *54*, *56*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*

rad2deg, 86
rad2deg(), *15*, *138*, *369*
reformulate(), *253*
regex(), *246*, *425*
regular expressions, *439*
reliability, 87
reliability(), *41, 42*, *50*, *63*, *78*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
remove.packages(), *56*
replacement, *427*
reshape(), *484–486*
residuals(), *88*
residuals.MxModel, 88
residuals.MxModel(), *18*, *57, 58*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
RMSEA, 89
RMSEA(), *18*, *57, 58*, *78, 79*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
RMSEA.MxModel, 90
RMSEA.MxModel(), *18*, *57, 58*, *88–90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
RMSEA.summary.mxmodel, 91

RMSEA.summary.mxmodel(), *18*, *57, 58*, *88*, *90*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
round(), *360*

scale(), *447*
scales::dollar(), *14*
SE_from_p, 92
SE_from_p(), *41, 42*, *50*, *63*, *87*, *105*, *136*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
shQuote(), *429*
sin(), *15*, *86*
stack(), *472*
stats::confint(), *140*, *151*
stats::glm(), *134*
stats::lm(), *134*
stats::reshape(), *487*
subset(), *546*
summary(), *318*
summaryAPA *(*umxAPA*)*, 134
Sys.Date(), *451*

t.test(), *134*, *136*
tmx_genotypic_effect, 93
tmx_genotypic_effect(), *95*, *105*
tmx_is.identified, 95
tmx_is.identified(), *94*, *105*
tmx_show, 96
tmx_show(), *18*, *57, 58*, *88*, *90*, *92*, *98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
tmx_show.MxMatrix, 97
tmx_show.MxMatrix(), *18*, *57, 58*, *88*, *90*, *92*, *97*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *328*
tmx_show.MxModel, 99
tmx_show.MxModel(), *101*, *159*, *276*
tmx_show.MxModelGSEM, 101
tryCatch(), *607*
tvars, 101
tvars(), *105*, *375*, *383*, *428*, *445*, *477*, *481*

umx, *12, 13*, *16, 17*, *40–42*, *45, 46*, *50*, *52*, *54–56*, *62, 63*, *65*, *67*, *69*, *72*, *74*, *80–85*, *87*, *93–95*, *103*, 103, *110*, *117*, *121*, *127*, *132*, *134*, *136*, *144*, *149*, *152*, *156*, *161*, *164*, *166*, *168*, *172*, *174*, *183*, *185*, *187–189*, *191, 192*, *205*, *212*, *216*, *218*, *220*, *223*, *225*, *227*, *231*, *233*, *237*, *239*, *242*, *245–247*, *251*, *254–256*, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*, *274*, *282*, *287*, *290, 291*, *293, 294*, *296*, *299*, *301*, *303*, *305*, *310*, *312, 313*, *324*, *327*, *329*, *332*, *336*, *340*, *343, 344*, *347*, *350*, *353*, *355–358*, *361–363*, *366*, *371–375*, *377*, *381–383*, *388*, *390, 391*, *396*, *398*, *401*, *403, 404*, *409, 410*, *412*, *414–416*, *419*, *422–426*, *428*, *430–432*, *434, 435*, *437, 438*, *440–443*, *445, 446*, *448–450*, *452*, *455*, *457–465*, *467–472*, *474*, *477*, *479–481*, *483*, *485–488*, *490, 491*, *495*, *516*, *543*, *607*, *651*
umx(), *175*
umx-package *(*umx*)*, 103
umx2ord *(*umx_cont_2_quantiles*)*, 370
umx_aggregate, 357
umx_aggregate(), *12*, *72*, *104*, *136*, *185*, *188, 189*, *191, 192*, *246*, *361*, *383*, *480*
umx_APA_pval, 359
umx_APA_pval(), *60*, *238*, *296*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
umx_apply, 360
umx_apply(), *41, 42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357, 358*, *372*, *423*, *445, 446*, *448*, *483*
umx_array_shift, 361
umx_array_shift(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*
umx_as_numeric, 362
umx_as_numeric(), *62*, *80–83*, *105*, *183*, *205*, *220*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*,

442, *455*, *457*, *472*, *474*
umx_check, 363
umx_check(), *365–367*, *384–387*, *389*, *392–396*, *399*
umx_check_model, 364
umx_check_model(), *364*, *366, 367*, *384–387*, *389*, *392–396*, *399*
umx_check_names, 365, *440*
umx_check_names(), *105*, *364, 365*, *367*, *384–396*, *398, 399*, *428*
umx_check_OS, 367
umx_check_OS(), *364–366*, *384–387*, *389*, *392–396*, *399*
umx_check_parallel, 368
umx_check_parallel(), *15*, *86*, *138*, *369*, *480*
umx_checkpoint *(*umx_set_checkpoint*)*, 459
umx_complete_dollar, 369
umx_complete_dollar(), *15*, *86*, *138*, *369*
umx_cont_2_quantiles, 370
umx_cont_2_quantiles(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_cor, 371
umx_cor(), *41, 42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *423*, *445, 446*, *448*, *483*
umx_data, 373
umx_data(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*
umx_detect_cores, 373
umx_detect_cores(), *105*, *381, 382*, *449*, *458–465*, *467–471*
umx_explode, 374
umx_explode(), *103*, *105*, *375*, *383*, *428*, *445*, *477*, *481*
umx_explode_twin_names, 375
umx_explode_twin_names(), *103*, *105*, *375*, *383*, *428*, *445*, *477*, *481*
umx_factor *(*umxFactor*)*, 182
umx_file_load_pseudo, 376
umx_file_load_pseudo(), *16*, *105*, *416*, *425*, *430*, *441*, *488*
umx_find_object, 377
umx_find_object(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*
umx_fun_mean_sd, 378
umx_fun_mean_sd(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
umx_get_bracket_addresses, 379
umx_get_bracket_addresses(), *60*, *238*, *296*, *360*, *378*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
umx_get_checkpoint, 381
umx_get_checkpoint(), *105*, *374*, *382*, *449*, *458–465*, *467–471*
umx_get_options, 382
umx_get_options(), *105*, *374*, *381*, *449*, *458–465*, *467–471*
umx_grep, 382
umx_grep(), *103*, *105*, *375*, *428*, *445*, *477*, *481*
umx_has_been_run, 384
umx_has_been_run(), *364–367*, *385–387*, *389*, *392–396*, *399*
umx_has_CIs, 385
umx_has_CIs(), *364–367*, *384*, *386, 387*, *389*, *392–396*, *399*
umx_has_means, 386
umx_has_means(), *364–367*, *384, 385*, *387*, *389*, *392–396*, *399*
umx_has_square_brackets, 387

umx_has_square_brackets(), *364–367*, *384–386*, *389*, *392–396*, *399*
umx_is_class, 388
umx_is_class(), *105*, *366*, *390*, *391*, *396*, *398*
umx_is_cov, 389
umx_is_cov(), *364–367*, *384–387*, *392–396*, *399*
umx_is_endogenous, 390
umx_is_endogenous(), *105*, *366*, *388*, *391*, *396*, *398*
umx_is_exogenous, 391
umx_is_exogenous(), *105*, *366*, *388*, *390*, *396*, *398*
umx_is_GSEM, 392
umx_is_GSEM(), *364–367*, *384–387*, *389*, *393–396*, *399*
umx_is_LISREL, 393
umx_is_LISREL(), *364–367*, *384–387*, *389*, *392*, *394–396*, *399*
umx_is_MxData, 394
umx_is_MxData(), *364–367*, *384–387*, *389*, *392*, *393*, *395*, *396*, *399*
umx_is_MxMatrix, 394
umx_is_MxMatrix(), *364–367*, *384–387*, *389*, *392–394*, *396*, *399*
umx_is_MxModel, 395
umx_is_MxModel(), *364–367*, *384–387*, *389*, *392–395*, *399*
umx_is_numeric, 396
umx_is_numeric(), *105*, *366*, *388*, *390*, *391*, *398*
umx_is_ordered, 397
umx_is_ordered(), *105*, *159*, *366*, *388*, *390*, *391*, *396*
umx_is_RAM, 398
umx_is_RAM(), *364–367*, *384–387*, *389*, *392–396*
umx_log_wide_twin_data, 399
umx_long2wide, 400
umx_long2wide(), *105*, *419*, *422*, *424*, *443*, *450*, *486*, *487*, *490*
umx_lower.tri, 402
umx_lower.tri(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *426*, *430*, *431*, *437*, *452*, *479*, *485*, *487*
umx_lower2full, 403
umx_lower2full(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_make, 405
umx_make(), *59*, *60*, *238*, *296*, *360*, *378*, *380*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
umx_make_double_entry_data, 407
umx_make_double_entry_data(), *74*, *105*, *110*, *117*, *121*, *125–127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *278*, *287*, *293*, *294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *520*
umx_make_fake_data, 409
umx_make_fake_data(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_make_GxE_data, 411
umx_make_GxE_data(), *62*, *80–83*, *105*, *183*, *205*, *212*, *220*, *363*, *371*, *404*, *410*, *414*, *415*, *417*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_make_MR_data, 413
umx_make_MR_data(), *62*, *80–83*, *105*, *183*, *205*, *220*, *353*, *363*, *371*, *404*, *410*, *412*, *415*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_make_raw_from_cov, 414
umx_make_raw_from_cov(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_make_sql_from_excel, 415
umx_make_sql_from_excel(), *16*, *105*, *377*, *425*, *430*, *441*, *488*
umx_make_twin_data_nice, 421

umx_make_twin_data_nice(), *105*, *401*, *419*, *443*, *450*, *486*, *490*
umx_make_TwinData, 417
umx_make_TwinData(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *401*, *404*, *410*, *412*, *414*, *415*, *422*, *424*, *432*, *434*, *435*, *438*, *440*, *442*, *443*, *450*, *455*, *457*, *472*, *474*, *486*, *490*
umx_means, 422
umx_means(), *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *445*, *446*, *448*, *483*
umx_merge_randomized_columns, 423
umx_merge_randomized_columns(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *432*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*, *486*
umx_move_file, 424
umx_move_file(), *16*, *105*, *377*, *416*, *430*, *441*, *488*
umx_msg, 425
umx_msg(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *430*, *431*, *437*, *452*, *479*, *485*, *487*
umx_names, 427
umx_names(), *103*, *105*, *375*, *383*, *445*, *477*, *481*
umx_open, 429
umx_open(), *16*, *105*, *377*, *416*, *425*, *441*, *488*
umx_open_CRAN_page, 430
umx_open_CRAN_page(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *431*, *437*, *452*, *479*, *485*, *487*
umx_pad, 431
umx_pad(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430*, *437*, *452*, *479*, *485*, *487*
umx_paste_names *(*tvars*)*, 101
umx_paste_names(), *375*
umx_polychoric, 432
umx_polychoric(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *434*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_polypairwise, 433
umx_polypairwise(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_polytriowise, 434
umx_polytriowise(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umx_print, *313*, *314*, 435
umx_print(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430*, *431*, *452*, *478*, *479*, *485*, *487*
umx_r_test, 446
umx_r_test(), *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445*, *448*, *483*
umx_read_lower, 437
umx_read_lower(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *435*, *440*, *442*, *455*, *457*, *472*, *474*
umx_rename, 439
umx_rename(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *435*, *438*, *442*, *455*, *457*, *472*, *474*
umx_rename_file, 440
umx_rename_file(), *16*, *105*, *377*, *416*, *425*, *430*, *488*
umx_reorder, 442
umx_reorder(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414*, *415*, *419*, *424*, *432*, *434*, *435*, *438*, *440*, *455*, *457*, *472*, *474*
umx_residualize, 443
umx_residualize(), *105*, *109*, *401*, *419*, *422*, *450*, *486*, *490*
umx_rot, 444
umx_rot(), *103*, *105*, *375*, *383*, *428*, *477*, *481*
umx_round, 445
umx_round(), *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *446*, *448*, *483*
umx_scale, 447
umx_scale(), *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445*, *446*, *483*
umx_scale_reliabilities, 448
umx_scale_reliabilities(), *105*, *374*, *381*,

382, *454*, *458–465*, *467–471*
umx_scale_wide_twin_data, 449
umx_scale_wide_twin_data(), *105*, *401*, *419*, *422*, *443*, *486*, *490*
umx_scholastica_report, 450
umx_scholastica_report(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *479*, *485*, *487*
umx_score_scale, 453
umx_score_scale(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *448*, *457*, *472*, *474*
umx_select_valid, 457
umx_select_valid(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *472*, *474*, *486*
umx_set_auto_plot, 458
umx_set_auto_plot(), *65*, *105*, *374*, *381, 382*, *449*, *459–465*, *467–471*
umx_set_auto_run, 459
umx_set_auto_run(), *105*, *291*, *374*, *381, 382*, *449*, *458*, *460–465*, *467–471*
umx_set_checkpoint, 459
umx_set_checkpoint(), *105*, *374*, *381, 382*, *449*, *458, 459*, *461–465*, *467–471*
umx_set_condensed_slots, 461
umx_set_condensed_slots(), *105*, *374*, *381, 382*, *449*, *458–460*, *462–465*, *467–471*
umx_set_cores, 462
umx_set_cores(), *105*, *369*, *374*, *381, 382*, *449*, *458–461*, *463–465*, *467–471*, *480*
umx_set_data_variance_check, 463
umx_set_data_variance_check(), *105*, *374*, *381, 382*, *449*, *458–462*, *464, 465*, *467–471*
umx_set_dollar_symbol, 464
umx_set_dollar_symbol(), *36*, *105*, *374*, *381, 382*, *449*, *458–463*, *465*, *467–471*
umx_set_optimization_options, 465
umx_set_optimization_options(), *105*, *153*, *221*, *374*, *381, 382*, *449*, *458–464*, *467–471*
umx_set_optimizer, 466
umx_set_optimizer(), *105*, *374*, *381, 382*, *449*, *458–465*, *468–471*
umx_set_plot_file_suffix, 467
umx_set_plot_file_suffix(), *105*, *374*, *381, 382*, *449*, *458–465*, *467*, *469–471*
umx_set_plot_format, 468
umx_set_plot_format(), *65*, *67*, *69*, *105*, *374*, *381, 382*, *449*, *458–465*, *467–471*
umx_set_separator, 469
umx_set_separator(), *105*, *374*, *381, 382*, *449*, *458–465*, *467–471*
umx_set_silent, 470
umx_set_silent(), *105*, *374*, *381, 382*, *449*, *458–465*, *467–469*, *471*
umx_set_table_format, 471
umx_set_table_format(), *105*, *292*, *295*, *374*, *381, 382*, *435*, *437*, *449*, *458–465*, *467–470*
umx_stack, 472
umx_stack(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *474*
umx_standardize, 473
umx_standardize(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
umx_str_chars, 476
umx_str_chars(), *103*, *105*, *375*, *383*, *428*, *445*, *477*, *481*
umx_str_from_object, 477
umx_str_from_object(), *103*, *105*, *375*, *383*, *428*, *445*, *477*, *481*
umx_string_to_algebra, 475
umx_string_to_algebra(), *60*, *238*, *296*,

360, *378*, *380*, *406*, *473*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*

umx_strings2numeric, 474

umx_strings2numeric(), *62*, *80–83*, *105*, *183*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*

umx_tail, 478

umx_tail(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *485*, *487*

umx_time, 479

umx_time(), *12*, *72*, *104*, *136*, *185*, *188, 189*, *191, 192*, *246*, *358*, *369*, *462*

umx_trim, 481

umx_trim(), *103*, *105*, *375*, *383*, *428*, *445*, *477*

umx_tutorials, 481

umx_update_OpenMx *(*install.OpenMx*)*, 53

umx_var, 482

umx_var(), *41, 42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*

umx_wide2long, 484

umx_wide2long(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *422*, *426*, *430, 431*, *437*, *452*, *472*, *479*, *486, 487*

umx_wide2longTwinData, 486

umx_wide2longTwinData(), *105*, *401*, *419*, *422*, *443*, *450*, *490*

umx_wide4lmer, 487

umx_wide4lmer(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*

umx_write_to_clipboard, 488

umx_write_to_clipboard(), *16*, *105*, *377*, *416*, *425*, *430*, *441*

umx_yj_wide_twin_data, 489

umx_yj_wide_twin_data(), *105*, *401*, *419*, *422*, *443*, *450*, *486*

umxACE, 107, *144*

umxACE(), *74*, *104*, *115*, *117*, *121*, *125*, *127*, *130–132*, *144*, *155, 156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *222, 223*, *245*, *254–256*, *292–294*, *296*, *299*, *305*, *310*, *312*, *322*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*, *419*, *459*, *580*, *623*

umxACE_DE, 124

umxACE_DE(), *74*, *104*, *110*, *117*, *121*, *132*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *258, 259*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327–329*, *332*, *336*, *340*, *343*, *350*, *408, 409*, *519*

umxACE_GLM, 130

umxACE_GLM(), *74*, *104*, *110*, *117*, *121*, *127*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *260*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*

umxACEcov, 115

umxACEcov(), *74*, *102*, *104*, *110*, *121*, *127*, *132*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324, 325*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*, *624*

umxACEv, 118

umxACEv(), *74*, *104*, *110*, *117*, *127*, *132*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *257*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *326, 327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*, *580*, *626*

umxAlgebra, 133

umxAlgebra(), *105*, *187*, *225*, *291*, *301*, *347*, *355*, *495*, *516*

umxAPA, 134

umxAPA(), *12*, *72*, *93*, *104*, *185*, *188, 189*, *191, 192*, *246*, *312*, *358–360*, *480*

umxBrownie, 138

umxBrownie(), *15*, *86*, *369*

umxCI, 138

umxCI(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*,

142, *151*, *181, 182*, *186*, *298*, *328*
umxCI_boot, 141
umxCI_boot(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *151*, *181, 182*, *186*, *298*, *328*
umxCloud, 142
umxCloud(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxCLPM, 144
umxCompare, 146
umxCompare(), *18*, *104*, *127*, *149*, *174*, *236, 237*, *293*, *303*, *313*, *319*, *321*, *329*, *355*, *409*, *540*, *543*, *605*, *607*
umxConfint, 150
umxConfint(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *181, 182*, *186*, *298*, *328*, *538*
umxCov2cor, 152
umxCov2cor(), *41, 42*, *50*, *63*, *87*, *93*, *105*, *220*, *247*, *357*, *361*, *372*, *423*, *445, 446*, *448*, *483*
umxCP, 153, *331*
umxCP(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *161*, *164*, *166*, *168*, *212*, *216*, *218*, *220*, *223*, *245*, *261*, *293, 294*, *296, 297*, *299*, *305*, *310*, *312*, *324*, *327*, *329, 330*, *332*, *336*, *340*, *343*, *350*, *409*, *580*, *627*
umxDiagnose, 158
umxDiagnose(), *100*, *276*
umxDiffMZ, 160
umxDiffMZ(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *263*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *353*, *409*
umxDiscTwin, 162
umxDiscTwin(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *161*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *263*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *353*, *409*
umxDoC, 164, *339*
umxDoC(), *17*, *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *161*, *164*, *168*, *212*, *216*, *218*, *223*, *245*, *262, 263*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *331, 332*, *336*, *339, 340*, *343*, *350*, *353*, *409*
umxDoCp, 167
umxDoCp(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156*, *161*, *164*, *166*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxEFA, 169
umxEFA(), *104*, *353*
umxEquate, 173
umxEquate(), *104*, *149*, *237*, *293*, *303*, *313*, *494*, *543*, *607*
umxExampleCode_TRHGpaper *(*umxExamples*)*, 175
umxExamples, 175
umxExpCov, 180
umxExpCov(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *182*, *186*, *298*, *328*
umxExpMeans, 181
umxExpMeans(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181*, *186*, *298*, *328*
umxFactanal *(*umxEFA*)*, 169
umxFactanal(), *183*
umxFactor, 182
umxFactor(), *62*, *80–83*, *105*, *205*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umxFactorScores, 184
umxFactorScores(), *12*, *72*, *104*, *136*, *188, 189*, *191, 192*, *246*, *358*, *480*
umxFitIndices, 185
umxFitIndices(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *298*, *328*
umxFixAll, 187
umxFixAll(), *105*, *134*, *225*, *291*, *301*, *347*, *355*, *495*, *516*
umxGetLatents, 188
umxGetLatents(), *12*, *72*, *104*, *136*, *185*, *189*, *191, 192*, *246*, *358*, *480*
umxGetManifests, 189
umxGetManifests(), *12*, *72*, *104*, *136*, *185*, *188, 189*, *191, 192*, *246*, *358*, *480*
umxGetModel, 190

umxGetModel(), *12*, *72*, *104*, *136*, *185*, *188, 189*, *192*, *246*, *358*, *480*
umxGetParameters, 191
umxGetParameters(), *12*, *72*, *104*, *136*, *174*, *185*, *188, 189*, *191*, *246*, *358*, *480*
umxGSEM, 193, *316*
umxGSEM(), *12, 13*, *72*, *84*, *197, 198*, *201, 202*, *204*, *206*, *209*, *314*, *316*, *392*, *643*
umxGSEM_dl_tutorial_files, 196
umxGSEM_dl_tutorial_files(), *13*, *72*, *84*, *194, 195*, *198*, *201–204*, *206*, *209*, *314*, *316*
umxGSEM_GWAS, *71*, 197, *205*
umxGSEM_GWAS(), *12, 13*, *72*, *84*, *193*, *195*, *197*, *201, 202*, *204*, *206–209*, *314*, *316*
umxGSEM_label_ldsc, 200
umxGSEM_label_ldsc(), *13*, *72*, *84*, *195*, *197, 198*, *202*, *204*, *206*, *209*, *314*, *316*
umxGSEM_ldsc, 201
umxGSEM_ldsc(), *13*, *72*, *84*, *193*, *195*, *197, 198*, *201*, *203, 204*, *206*, *209*, *313, 314*, *316*
umxGSEM_munge, 203
umxGSEM_munge(), *13*, *72*, *84*, *195–198*, *201, 202*, *206*, *209*, *314*, *316*
umxGSEM_sim_snps, 205
umxGSEM_sim_snps(), *62*, *80–83*, *105*, *183*, *220*, *363*, *371*, *404*, *410*, *412*, *414, 415*, *419*, *424*, *432*, *434, 435*, *438*, *440*, *442*, *455*, *457*, *472*, *474*
umxGSEM_std, 206
umxGSEM_std(), *13*, *72*, *84*, *195*, *197, 198*, *201, 202*, *204*, *209*, *314*, *316*
umxGSEM_sumstats, 207
umxGSEM_sumstats(), *13*, *72*, *84*, *195*, *197, 198*, *201, 202*, *204*, *206*, *314*, *316*
umxGxE, 210
umxGxE(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156, 157*, *161*, *164*, *166*, *168*, *215, 216*, *218*, *223*, *245*, *266, 267*, *292–296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332–334*, *336*, *340*, *343*, *350*, *409*, *411, 412*, *417*, *419*, *580*
umxGxE_window, 216
umxGxE_window(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxGxEbiv, 214, *336*
umxGxEbiv(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *215*, *218*, *223*, *245*, *268*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *335, 336*, *340*, *343*, *350*, *409*, *417*, *419*
umxHetCor, 219
umxHetCor(), *41, 42*, *50*, *62, 63*, *80–83*, *87*, *93*, *105*, *152*, *183*, *205*, *247*, *357*, *361*, *363*, *371, 372*, *404*, *410*, *412*, *414, 415*, *419*, *423, 424*, *432*, *434, 435*, *438*, *440*, *442*, *445, 446*, *448*, *455*, *457*, *472*, *474*, *483*
umxIP, 220, *338*
umxIP(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156, 157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *245*, *269, 270*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336, 337*, *340*, *343*, *350*, *409*, *580*, *628*
umxJiggle, 224
umxJiggle(), *105*, *134*, *187*, *291*, *301*, *347*, *355*, *495*, *516*
umxLav2RAM, 225
umxLav2RAM(), *54*, *56*, *85*, *105*, *239*, *278*, *282*, *287*, *290*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*, *573*
umxLISREL, 229
umxLISREL(), *104*, *233*, *242*, *251*, *282*, *290*, *317*, *344*
umxMatrix, 232
umxMatrix(), *96, 97*, *99*, *104*, *134*, *231*, *234, 235*, *242*, *251*, *282*, *290*, *296*, *344*, *556*, *621, 622*
umxMatrixFree, 234
umxMeasurementInvariance, 235
umxMI, 236
umxMI(), *104*, *149*, *174*, *293*, *303*, *313*, *505*, *543*, *607*
umxModel, 238
umxModel(), *60*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*,

540–542, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
umxModelNames, 239
umxModelNames(), *54*, *56*, *85*, *105*, *227*, *287*, *356*, *362*, *373*, *377*, *403*, *426*, *430*, *431*, *437*, *452*, *479*, *485*, *487*
umxModify, 240
umxModify(), *17*, *104*, *110*, *127*, *140*, *174*, *231*, *233*, *251*, *263*, *282*, *290*, *301*, *303*, *324*, *332*, *344*, *409*, *516*
umxMR *(*umxTwoStage*)*, 352
umxMR(), *161*, *164*, *166*, *263*
umxMRDoC, 243
umxMRDoC(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *293*, *294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *338*, *340*, *343*, *350*, *409*
umxParameters, 245
umxParameters(), *12*, *72*, *104*, *136*, *185*, *188*, *189*, *191*, *192*, *245*, *358*, *480*
umxParan, 247
umxParan(), *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *357*, *361*, *372*, *423*, *445*, *446*, *448*, *483*
umxPath, 248
umxPath(), *104*, *167*, *168*, *227*, *231*, *233*, *242*, *278*, *282*, *287*, *290*, *291*, *344*, *350*, *593*
umxPlot, 252
umxPlot(), *46*, *65*, *67*, *69*, *105*, *255*, *256*, *258–260*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotACE, 254
umxPlotACE(), *46*, *65*, *67*, *69*, *105*, *110*, *254*, *256*, *258–260*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotACE_DE, 258
umxPlotACE_DE(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258*, *260*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*, *328*
umxPlotACE_GLM, 259
umxPlotACE_GLM(), *46*, *65*, *67*, *69*, *105*, *132*, *254–256*, *258*, *259*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotACEcov, 255
umxPlotACEcov(), *46*, *65*, *67*, *69*, *105*, *254*, *255*, *258–260*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotACEv, 257
umxPlotACEv(), *46*, *65*, *67*, *69*, *105*, *254–256*, *259*, *260*, *262*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotCP, 260
umxPlotCP(), *46*, *65*, *67*, *69*, *105*, *156*, *254–256*, *258–260*, *263*, *265*, *267*, *268*, *270–272*, *274*
umxPlotDoC, 262
umxPlotDoC(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *265*, *267*, *268*, *270–272*, *274*
umxPlotFun, 264
umxPlotFun(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *267*, *268*, *270–272*, *274*
umxPlotGxE, 266
umxPlotGxE(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *265*, *268*, *270–272*, *274*
umxPlotGxEbiv, 267
umxPlotGxEbiv(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *265*, *267*, *270–272*, *274*
umxPlotIP, 269
umxPlotIP(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *265*, *267*, *268*, *271*, *272*, *274*
umxPlotMxModelTwinMaker *(*plot.MxModelTwinMaker*)*, 68
umxPlotPredict, 270
umxPlotPredict(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *265*, *267*, *268*, *270*, *272*, *274*
umxPlotSexLim, 271
umxPlotSexLim(), *46*, *65*, *67*, *69*, *105*, *254–256*, *258–260*, *262*, *263*, *265*, *267*, *268*, *270*, *271*, *274*, *305*, *340*
umxPlotSimplex, 273
umxPlotSimplex(), *46*, *65*, *67*, *69*, *105*,

254–256, *258–260*, *262, 263*, *265*, *267, 268*, *270–272*
umxPower, 275
umxPower(), *74*, *100*, *159*, *417*
umxPrint_double_entry_data, 278
umxRAM, 278
umxRAM(), *52*, *96, 97*, *99*, *103, 104*, *149*, *188–191*, *193, 194*, *225*, *227*, *231*, *233*, *236*, *238*, *242*, *251*, *276*, *282*, *287*, *289–291*, *321*, *344*, *349, 350*, *459*, *470*, *509*, *588*, *592*, *607*, *641*, *643*
umxRAM2Lav, 286
umxRAM2Lav(), *54*, *56*, *85*, *105*, *227*, *239*, *356*, *362*, *373*, *377*, *403*, *426*, *430, 431*, *437*, *452*, *479*, *485*, *487*
umxRAM_DE, 287
umxRAM_DE(), *104*, *231*, *233*, *242*, *251*, *282*, *344*, *409*, *643*
umxRAM_GLM, 290
umxRAM_GLM(), *105*, *131, 132*, *134*, *187*, *225*, *301*, *347*, *355*, *495*, *516*
umxReduce, 292
umxReduce(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *149*, *157*, *162*, *164*, *166*, *168*, *174*, *212*, *215, 216*, *218*, *223*, *237*, *245*, *294*, *296*, *299*, *303*, *305*, *310*, *312, 313*, *324*, *327*, *329*, *332*, *334*, *336*, *340*, *343*, *350*, *409*, *543*, *607*
umxReduceACE, 293
umxReduceACE(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *292, 293*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxReduceGxE, 295
umxReduceGxE(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *292–294*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxRenameMatrix, 296
umxRenameMatrix(), *60*, *238*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *590*, *592*, *594, 595*, *597, 598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–632*, *634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*
umxRotate, 297
umxRotate(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *328*
umxRotate.MxModelCP, 298
umxRotate.MxModelCP(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *156, 157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296, 297*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxRun, *143, 144*, 300
umxRun(), *105*, *134*, *181*, *187*, *225*, *291*, *347*, *355*, *495*, *516*
umxSetParameters, 301
umxSetParameters(), *104*, *149*, *174*, *237*, *293*, *313*, *543*, *607*
umxSexLim, 303
umxSexLim(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *272*, *293, 294*, *296*, *299*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *339, 340*, *343*, *350*, *409*, *632*
umxSimplex, 308
umxSimplex(), *55*, *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *274*, *293, 294*, *296*, *299*, *305*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *342, 343*, *350*, *409*, *633, 634*
umxSummarizeTwinData, 311
umxSummarizeTwinData(), *74*, *104*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxSummary, 313
umxSummary(), *104*, *127*, *148, 149*, *156*, *170*, *174*, *212*, *215*, *223*, *237*, *246*, *255*,

256, *259*, *261, 262*, *267, 268*, *270*, *274*, *278*, *282*, *290*, *293*, *303*, *310*, *321*, *323*, *327, 328*, *331, 332*, *336*, *338–340*, *343*, *409*, *543*, *595*, *605*, *607*, *620*, *639*, *656*
umxSummary.list, 313
umxSummary.list(), *13*, *72*, *84*, *195*, *197, 198*, *201, 202*, *204*, *206*, *209*, *315–317*
umxSummary.MxDataStatic, 315
umxSummary.MxDataStatic(), *13*, *72*, *84*, *195*, *197, 198*, *201, 202*, *204*, *206*, *209*, *314*, *317*
umxSummary.MxLISRELModel, 316
umxSummary.MxLISRELModel(), *314*, *316*
umxSummary.MxModel, 317
umxSummary.MxModel(), *313*, *325*, *331*, *334*, *338, 339*
umxSummary.MxModelACE *(*umxSummaryACE*)*, 322
umxSummary.MxModelACEcov *(*umxSummaryACEcov*)*, 324
umxSummary.MxModelACEv *(*umxSummaryACEv*)*, 326
umxSummary.MxModelCP *(*umxSummaryCP*)*, 330
umxSummary.MxModelDoC *(*umxSummaryDoC*)*, 331
umxSummary.MxModelDoC(), *17*, *263*
umxSummary.MxModelGLM, 321
umxSummary.MxModelGxE *(*umxSummaryGxE*)*, 333
umxSummary.MxModelGxEbiv *(*umxSummaryGxEbiv*)*, 335
umxSummary.MxModelIP *(*umxSummaryIP*)*, 337
umxSummary.MxModelMRDoC *(*umxSummaryMRDoC*)*, 338
umxSummary.MxModelSexLim *(*umxSummarySexLim*)*, 339
umxSummary.MxModelSimplex *(*umxSummarySimplex*)*, 342
umxSummary.MxRAMModel *(*umxSummary.MxModel*)*, 317
umxSummaryACE, 322
umxSummaryACE(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312, 313*, *327*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxSummaryACE_DE, 327
umxSummaryACE_DE(), *18*, *57, 58*, *88*, *90*, *92*, *97, 98*, *140*, *142*, *151*, *181, 182*, *186*, *298*, *313*
umxSummaryACE_GLM, 329
umxSummaryACE_GLM(), *74*, *105*, *110*, *117*, *121*, *127*, *131, 132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *259, 260*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *332*, *336*, *340*, *343*, *350*, *409*
umxSummaryACEcov, 324
umxSummaryACEcov(), *321*, *331*, *334*, *338, 339*
umxSummaryACEv, 326
umxSummaryACEv(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312, 313*, *324*, *329*, *332*, *336*, *340*, *343*, *350*, *409*
umxSummaryCP, 330
umxSummaryCP(), *156*, *313*, *321*, *325*, *334*, *338, 339*
umxSummaryDoC, 331
umxSummaryDoC(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *336*, *340*, *343*, *350*, *409*
umxSummaryGxE, 333
umxSummaryGxE(), *313*, *321*, *325*, *331*, *338, 339*
umxSummaryGxEbiv, 335
umxSummaryGxEbiv(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293, 294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *340*, *343*, *350*, *409*
umxSummaryIP, 337
umxSummaryIP(), *313*, *321*, *325*, *331*, *334*, *339*
umxSummaryMRDoC, 338
umxSummaryMRDoC(), *321*, *325*, *331*, *334*, *338*
umxSummarySexLim, 339
umxSummarySexLim(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *272*,

293, *294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *343*, *350*, *409*
umxSummarySimplex, 342
umxSummarySimplex(), *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293*, *294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *350*, *409*
umxSuperModel, 343
umxSuperModel(), *104*, *188*, *189*, *191*, *225*, *231*, *233*, *239*, *242*, *251*, *282*, *290*, *349*, *350*, *514*
umxThresholdMatrix, 346
umxThresholdMatrix(), *105*, *134*, *187*, *225*, *291*, *301*, *355*, *495*, *516*, *580*, *641*
umxTwinMaker, 349
umxTwinMaker(), *69*, *74*, *105*, *110*, *117*, *121*, *127*, *132*, *144*, *157*, *162*, *164*, *166*, *168*, *212*, *216*, *218*, *223*, *245*, *293*, *294*, *296*, *299*, *305*, *310*, *312*, *324*, *327*, *329*, *332*, *336*, *340*, *343*, *409*, *592*, *594*
umxTwoStage, 352
umxTwoStage(), *104*, *172*
umxUnexplainedCausalNexus, 354
umxUnexplainedCausalNexus(), *105*, *134*, *187*, *225*, *291*, *301*, *347*, *495*, *516*
umxVersion, 355
umxVersion(), *54*, *56*, *85*, *105*, *227*, *239*, *287*, *362*, *373*, *377*, *403*, *426*, *430*, *431*, *437*, *452*, *479*, *485*, *487*
umxWeightedAIC, 356
umxWeightedAIC(), *12*, *41*, *42*, *50*, *63*, *87*, *93*, *105*, *152*, *220*, *247*, *361*, *372*, *423*, *445*, *446*, *448*, *483*
us_skinfold_data, *13*, *17*, *40*, *45*, *52*, *55*, *84*, *105*, 490
utils::tail(), *478*

vcov(), *181*
vcov.MxModel *(*umxExpCov*)*, 180
vignette(), *482*

within(), *457*

xgh_check_base_name, 491
xgh_check_base_name(), *47–49*, *492*
xgh_get_current_branch, 492
xgh_get_current_branch(), *47–49*, *492*
xmu_ace_de_apply_censor_thresholds, 517
xmu_ace_de_apply_censor_thresholds(), *60*, *238*, *296*, *360*, *379*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *520–524*, *526–528*, *530–532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *592*, *594*, *595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmu_ace_de_parse_censor_meta, 519
xmu_ace_de_parse_censor_meta(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *521–524*, *526–528*, *530–532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *593–595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmu_ace_de_parse_censor_rule, 520
xmu_ace_de_parse_censor_rule(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520*, *522–524*, *526–528*, *530–532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *593–595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–657*, *659*

xmu_ace_de_release_binary_v1, 521
xmu_ace_de_release_binary_v1(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *475*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520, 521*, *523, 524*, *526–528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–657*, *659*
xmu_bracket_address2rclabel, 522
xmu_bracket_address2rclabel(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *475*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–522*, *524*, *526–528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*
xmu_catml_df3_diagnostic, 524
xmu_catml_df3_diagnostic(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *475*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–523*, *526–528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *606*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*
xmu_catml_discrepancy_at_WLS, 525
xmu_catml_discrepancy_at_WLS(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *475*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *527, 528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *606, 607*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*
xmu_catml_eval_model, 526
xmu_catml_eval_model(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–526*, *528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *606*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*
xmu_catml_implied_correlation, 528
xmu_catml_implied_correlation(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526, 527*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*
xmu_catml_wls_v, 529
xmu_catml_wls_v(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *531, 532*, *534*, *536, 537*, *539–542*, *545, 546*,

548, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *606*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*

xmu_cell_is_on, 530

xmu_cell_is_on(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530*, *532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645*, *647, 648*, *650–652*, *654–657*, *659*

xmu_check_levels_identical, 532

xmu_check_levels_identical(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530, 531*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569, 570*, *572, 573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650–652*, *654–657*, *659*

xmu_check_needs_means, 533

xmu_check_needs_means(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569, 570*, *572, 573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650–652*, *654–657*, *659*

xmu_check_variance, 535

xmu_check_variance(), *60*, *238*, *297*, *360*, *379, 380*, *406*, *473*, *476*, *494*, *498, 499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *537*, *539–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569, 570*, *572, 573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591*, *593–595*, *597, 598*, *600, 601*, *603, 604*, *609, 610*, *612–614*, *616*, *618, 619*, *621–623*, *625–627*, *629–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650–652*, *654–657*, *659*

xmu_CI_merge, 536

xmu_CI_merge(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536*, *538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591, 592*, *594, 595*, *597, 598*, *600, 601*, *603, 604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654–658*

xmu_CI_stash, 538

xmu_CI_stash(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536, 537*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *559*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591, 592*, *594, 595*, *597, 598*, *600, 601*, *603, 604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*,

629–631, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmu_clean_label, 539

xmu_clean_label(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *476*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539*, *541*, *542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *593–595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_compare_constraint_note, 540

xmu_compare_constraint_note(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *476*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539*, *540*, *542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_compare_robust_ML, 542

xmu_compare_robust_ML(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *476*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–541*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_compare_WLS, 543

xmu_compare_WLS(), *104*, *149*, *174*, *237*, *293*, *303*, *313*, *607*, *656*

xmu_custom_completer, 544

xmu_data_missing, 544

xmu_data_missing(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *476*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_data_swap_a_block, 546

xmu_data_swap_a_block(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *473*, *476*, *494*, *498*, *499*, *501–505*, *507–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_describe_data_WLS, 547

xmu_describe_data_WLS(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_DF_to_mxData_TypeCov, 549
xmu_DF_to_mxData_TypeCov(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *592*, *594*, *595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmu_dot_define_shapes, 550
xmu_dot_define_shapes(), *552*, *554*, *555*, *557*, *561*
xmu_dot_make_paths, 552
xmu_dot_make_paths(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550–552*, *555*, *557*, *560–562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_dot_make_residuals, 554
xmu_dot_make_residuals(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550–554*, *557*, *560–562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_dot_maker, 551
xmu_dot_maker(), *60*, *238*, *297*, *360*, *379*, *380*, *406*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *551*, *553–555*, *557*, *560–562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625–627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_dot_mat2dot, 556
xmu_dot_mat2dot(), *551*, *552*, *554*, *555*, *561*
xmu_dot_move_ranks, 559
xmu_dot_move_ranks(), *60*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *562*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_dot_rank, 560
xmu_dot_rank(), *551*, *552*, *554*, *555*, *557*
xmu_dot_rank_str, 561
xmu_dot_rank_str(), *60*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *564*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*

xmu_equate_threshold_values, 562
xmu_extract_column, 563
xmu_extract_column(), *60*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *565*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_get_CI, 564
xmu_get_CI(), *60*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *569*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_gsem_expand_snp, 566
xmu_gsem_extract_snp_path, 567
xmu_gsem_subset_covstruc, 568
xmu_has_summary_mxData, 569
xmu_has_summary_mxData(), *60*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *565*, *570*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_invert_matrix, 570
xmu_invert_matrix(), *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *569*, *572*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_is_ordinal_WLS, 571
xmu_is_ordinal_WLS(), *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–523*, *525–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *569*, *570*, *573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *606*, *607*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_is_wls, 572
xmu_lavaan_process_group, 573
xmu_lavaan_process_group(), *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *511*, *513*, *515*, *518*, *520–523*, *525–528*, *530–532*, *534*, *536*, *537*, *539–542*, *545*, *546*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *569*, *570*, *572*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599–601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650–652*, *654–657*, *659*
xmu_make_bin_cont_pair_data, 574
xmu_make_bin_cont_pair_data(), *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*,

494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 570, 572, 573, 577, 581, 584, 586–588, 591, 593–595, 597, 599–601, 603, 604, 609, 610, 612–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_make_mxData, 576

xmu_make_mxData(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 581, 584, 586–588, 591, 593–595, 597, 599–601, 603, 604, 609, 610, 612–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_make_TwinSuperModel, 578

xmu_make_TwinSuperModel(), 61, 156, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 570, 572, 573, 575, 577, 584, 586–588, 591, 593–595, 597, 599–601, 603, 604, 609, 610, 612–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–657, 659

xmu_match.arg, 583

xmu_match.arg(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 586–588, 591, 593–595, 597, 599–601, 603, 604, 609, 610, 612–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_mxData_summary, 585

xmu_mxData_summary(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 587, 588, 591, 593–595, 597, 599–604, 609, 610, 612–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_mxRefModels, 586

xmu_mxRefModels(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586, 588, 591, 593–595, 597, 599–601, 603, 604, 609, 611–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_name_from_lavaan_str, 587

xmu_name_from_lavaan_str(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586, 587, 591, 593–595, 597, 599–601, 603, 604, 609, 611–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644,

645, 647, 648, 650–652, 654–656, 658, 659
xmu_openmx_engine_status, 589
xmu_openmx_engine_status(), 54, 60, 590
xmu_openmx_install_message, 589
xmu_openmx_install_message(), 589
xmu_PadAndPruneForDefVars, 590
xmu_PadAndPruneForDefVars(), 60, 238, 296, 360, 378, 380, 406, 473, 475, 494, 498, 499, 501–506, 508, 509, 511, 513, 515, 518, 520–524, 526–529, 531, 532, 534, 536–538, 540–542, 545, 546, 548, 550, 552, 553, 555, 560, 562, 564, 565, 569–571, 573, 575, 577, 581, 584, 585, 587, 588, 592, 594, 595, 597, 598, 600, 601, 603, 604, 609, 610, 612–614, 616, 618–620, 622–624, 626, 627, 629–631, 633, 634, 636, 638, 639, 641–643, 645, 646, 648, 650–652, 654–658
xmu_parse_completion_context, 591
xmu_path2twin, 592
xmu_path2twin(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586–588, 591, 594, 595, 597, 599–601, 603, 604, 609, 611–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659
xmu_path_regex, 593
xmu_path_regex(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586–588, 591, 593, 595, 597, 599–601, 603, 604, 609, 611–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659
xmu_print_algebras, 595
xmu_print_algebras(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586–588, 591, 593–595, 597, 599–601, 603, 604, 609, 611–614, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659
xmu_ram_de_apply_censor_thresholds, 596
xmu_ram_de_apply_censor_thresholds(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 511, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586–588, 591, 593–595, 599–601, 603, 604, 609, 611–613, 615, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659
xmu_ram_de_parse_censor_meta, 598
xmu_ram_de_parse_censor_meta(), 61, 238, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–504, 506–509, 512, 513, 515, 518, 520–523, 525–528, 530–532, 534, 536, 537, 539–542, 545, 546, 548, 550, 552, 554, 555, 560, 562, 564, 566, 569, 571–573, 575, 577, 581, 584, 586–588, 591, 593–595, 597, 600, 601, 603, 604, 609, 611–613, 615, 616, 618, 619, 621–623, 625, 626, 628–631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 648, 650–652, 654–656, 658, 659

xmu_rclabel_2_bracket_address, 599
xmu_rclabel_2_bracket_address(), *61*, *238*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *569*, *571–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599*, *601*, *603, 604*, *609*, *611–613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650–652*, *654–656*, *658, 659*
xmu_relevel_factors, 600
xmu_relevel_factors(), *61*, *238*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530–532*, *534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *569*, *571–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593–595*, *597*, *599, 600*, *603, 604*, *609*, *611–613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650, 651*, *653–656*, *658, 659*
xmu_require_summary_mxData, 602
xmu_require_summary_mxData(), *61*, *238*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530, 531*, *533, 534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *569–573*, *575*, *577*, *581*, *584–588*, *591*, *593, 594*, *596, 597*, *599–601*, *604*, *609*, *611–613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650, 651*, *653–656*, *658, 659*
xmu_robust_ML_fit, 603
xmu_robust_ML_fit(), *61*, *238*, *297*, *319*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530, 531*, *533, 534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599–601*, *603*, *609*, *611–613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650, 651*, *653–656*, *658, 659*
xmu_robust_WLS_fit, 605
xmu_robust_WLS_fit(), *104*, *149*, *174*, *237*, *293*, *303*, *313*, *319*, *321*, *524, 525*, *543*, *571*, *610*, *653*, *656*
xmu_safe_run_summary, 607
xmu_safe_run_summary(), *61*, *132*, *238*, *291*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530, 531*, *533, 534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599–601*, *603, 604*, *611–613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650, 651*, *653–656*, *658, 659*
xmu_savalei_fit_cfi, 610
xmu_savalei_fit_cfi(), *61*, *238*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501–504*, *506–509*, *512, 513*, *515*, *518*, *520–523*, *525–528*, *530, 531*, *533, 534*, *536, 537*, *539–542*, *545, 546*, *548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599–601*, *603, 604*, *606*, *609*, *612, 613*, *615, 616*, *618, 619*, *621–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638, 639*, *641, 642*, *644, 645*, *647, 648*, *650, 651*, *653–656*, *658, 659*
xmu_savalei_fit_rmsea, 611
xmu_savalei_fit_rmsea(), *61*, *238*, *297*,

360, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *606*, *609*, *611*, *613*, *615*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650*, *651*, *653–656*, *658*, *659*

`xmu_savalei_fit_tli`, 612

`xmu_savalei_fit_tli()`, *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *606*, *609*, *611*, *612*, *615*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650*, *651*, *653–656*, *658*, *659*

`xmu_savalei_null_scaling_factor`, 614

`xmu_savalei_null_scaling_factor()`, *61*, *238*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–504*, *506–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *606*, *609*, *611–613*, *616*, *618*, *619*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *648*, *650*, *651*, *653–656*, *658*, *659*

`xmu_savalei_polycor_blocks`, 615

`xmu_savalei_polycor_blocks()`, *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *606*, *607*, *609*, *611–613*, *615*, *618*, *620–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

`xmu_savalei_scaling_factor`, 617

`xmu_savalei_scaling_factor()`, *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *606*, *609*, *611–613*, *615*, *616*, *620–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

`xmu_set_sep_from_suffix`, 619

`xmu_set_sep_from_suffix()`, *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–509*, *512*, *513*, *515*, *519–523*, *525–528*, *530*, *531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–573*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *621–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

`xmu_show_fit_or_comparison`, 620

`xmu_show_fit_or_comparison()`, *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*,

570–572, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *620*, *622*, *623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_simplex_corner, 621

xmu_simplex_corner(), *61*, *233*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *620*, *621*, *623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_standardize_ACE, 622

xmu_standardize_ACE(), *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *620–622*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_standardize_ACEcov, 624

xmu_standardize_ACEcov(), *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–542*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *620–623*, *626*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_standardize_ACEv, 625

xmu_standardize_ACEv(), *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–541*, *543*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *616*, *618*, *620–623*, *625*, *628–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_standardize_CP, 627

xmu_standardize_CP(), *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–541*, *543*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *617*, *618*, *620–623*, *625*, *626*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*, *649–651*, *653–656*, *658*, *659*

xmu_standardize_IP, 628

xmu_standardize_IP(), *61*, *239*, *297*, *360*, *379*, *380*, *407*, *474*, *476*, *494*, *498*, *499*, *501–503*, *505–508*, *510*, *512*, *514*, *515*, *519–523*, *525–527*, *529–531*, *533*, *534*, *536*, *537*, *539–541*, *543*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599–601*, *603*, *604*, *609*, *611–613*, *615*, *617*, *618*, *620–623*, *625*, *626*, *628*, *630*, *631*, *633*, *634*, *636*, *638*, *639*, *641*, *642*, *644*, *645*, *647*,

649–651, 653–656, 658, 659
xmu_standardize_LISREL, 629
xmu_standardize_LISREL(), 61, 231, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599–601, 603, 604, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628, 629, 631, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_standardize_RAM, 630
xmu_standardize_RAM(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602–604, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–630, 633, 634, 636, 638, 639, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_standardize_SexLim, 632
xmu_standardize_SexLim(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602–604, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 634, 636, 638, 639, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_standardize_Simplex, 633
xmu_standardize_Simplex(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602–604, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 636, 638, 639, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_start_value_list, 637
xmu_start_value_list(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602–604, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 639, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_starts, 635
xmu_starts(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 638, 640–642, 644, 645, 647, 649–651, 653–656, 658, 659
xmu_summary_RAM_group_parameters, 638
xmu_summary_RAM_group_parameters(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533,

534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 641, 642, 644, 645, 647, 649–651, 653–656, 658, 659

xmu_threshold_id_RAM, 640

xmu_threshold_id_RAM(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640, 642, 644, 645, 647, 649–651, 653–656, 658, 659

xmu_threshold_id_twin_check, 641

xmu_threshold_id_twin_check(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501–503, 505–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640, 641, 644, 645, 647, 649–651, 653–656, 658, 659

xmu_topo_sort, 642

xmu_topo_sort(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501, 502, 504–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640–642, 645, 649–651, 653–656, 658, 659

xmu_twin_add_WeightMatrices, 644

xmu_twin_add_WeightMatrices(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501, 502, 504–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640–642, 644, 647, 649–651, 653–656, 658, 659

xmu_twin_check, 645

xmu_twin_check(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501, 502, 504–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640–642, 644, 645, 649–651, 653–656, 658, 659

xmu_twin_get_var_names, 647

xmu_twin_get_var_names(), 61, 239, 297, 360, 379, 380, 407, 474, 476, 494, 498, 499, 501, 502, 504–508, 510, 512, 514, 515, 519–523, 525–527, 529–531, 533, 534, 536, 537, 539–541, 543, 545, 547, 548, 550, 552, 554, 555, 560, 562, 564, 566, 570–572, 574, 575, 577, 581, 584, 586–588, 591, 593, 594, 596, 597, 599, 600, 602, 603, 605, 609, 611–613, 615, 617, 618, 620–623, 625, 626, 628–631, 633, 634, 636, 638, 640–642, 644, 645, 647, 650,

651, *653–656*, *658, 659*
xmu_twin_make_def_means_mats_and_alg, 649
xmu_twin_make_def_means_mats_and_alg(), *61*, *239*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501, 502*, *504–508*, *510*, *512*, *514, 515*, *519–523*, *525–527*, *529–531*, *533, 534*, *536, 537*, *539–541*, *543*, *545*, *547, 548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–572*, *574, 575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599, 600*, *602, 603*, *605*, *609*, *611–613*, *615*, *617*, *619–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638*, *640–642*, *644, 645*, *647*, *649*, *651*, *653–655*, *657–659*
xmu_twin_upgrade_selDvs2SelVars, 650
xmu_twin_upgrade_selDvs2SelVars(), *61*, *239*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501, 502*, *504–508*, *510*, *512*, *514, 515*, *519–523*, *525–527*, *529–531*, *533, 534*, *536, 537*, *539–541*, *543*, *545*, *547, 548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–572*, *574, 575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599, 600*, *602, 603*, *605*, *609*, *611–613*, *615*, *617*, *619–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638*, *640–642*, *644, 645*, *647*, *649, 650*, *653–655*, *657–659*
xmu_update_covar, 652
xmu_update_covar(), *61*, *239*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501, 502*, *504–508*, *510*, *512*, *514, 515*, *519–523*, *525–527*, *529–531*, *533, 534*, *536, 537*, *539–541*, *543*, *545*, *547, 548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–572*, *574, 575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599, 600*, *602, 603*, *605*, *609*, *611–613*, *615*, *617*, *619–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638*, *640–642*, *644, 645*, *647*, *649–651*, *654, 655*, *657–659*
xmu_WLS_align_jacobian, 653
xmu_WLS_align_jacobian(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531, 532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591, 592*, *594, 595*, *597, 598*, *600, 601*, *603–605*, *609, 610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *655–658*
xmu_WLS_align_weight, 654
xmu_WLS_align_weight(), *60*, *238*, *296*, *360*, *379, 380*, *406*, *473*, *475*, *494*, *498, 499*, *501–506*, *508, 509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536–538*, *540–542*, *545, 546*, *548*, *550*, *552, 553*, *555*, *560*, *562*, *564, 565*, *569–571*, *573*, *575*, *577*, *581*, *584, 585*, *587, 588*, *591, 592*, *594, 595*, *597, 598*, *600, 601*, *603–605*, *609, 610*, *612–614*, *616*, *618–620*, *622–624*, *626, 627*, *629–631*, *633, 634*, *636*, *638, 639*, *641–643*, *645, 646*, *648*, *650–652*, *654*, *656–658*
xmu_wls_display_chi, 655
xmu_wls_display_chi(), *61*, *239*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501, 502*, *504–508*, *510*, *512*, *514, 515*, *519–523*, *525–527*, *529–531*, *533, 534*, *536, 537*, *539–541*, *543*, *545*, *547, 548*, *550*, *552*, *554, 555*, *560*, *562*, *564*, *566*, *570–572*, *574, 575*, *577*, *581*, *584*, *586–588*, *591*, *593, 594*, *596, 597*, *599, 600*, *602, 603*, *605*, *609*, *611–613*, *615*, *617*, *619–623*, *625, 626*, *628–631*, *633, 634*, *636*, *638*, *640–642*, *644, 645*, *647*, *649–651*, *653–655*, *658, 659*
xmu_wls_extract_WV, 657
xmu_wls_extract_WV(), *61*, *239*, *297*, *360*, *379, 380*, *407*, *474*, *476*, *494*, *498, 499*, *501, 502*, *504–508*, *510*, *512*, *514, 515*, *519–523*, *525–527*,

529–531, *533*, *534*, *536*, *537*, *539–541*, *543*, *545*, *547*, *548*, *550*, *552*, *554*, *555*, *560*, *562*, *564*, *566*, *570–572*, *574*, *575*, *577*, *581*, *584*, *586–588*, *591*, *593*, *594*, *596*, *597*, *599*, *600*, *602*, *603*, *605*, *609*, *611–613*, *615*, *617*, *619–623*, *625*, *626*, *628–631*, *633*, *634*, *636*, *638*, *640–642*, *644*, *645*, *647*, *649–651*, *653–655*, *657*, *659*

xmu_WLS_polycor_names, 658

xmu_WLS_polycor_names(), *60*, *238*, *296*, *360*, *379*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–528*, *530–532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *560*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *592*, *594*, *595*, *597*, *598*, *600*, *601*, *603*, *604*, *609*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–657*

xmuCalculateSRMR, 493

xmuHasSquareBrackets, 493

xmuHasSquareBrackets(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuLabel, 494

xmuLabel(), *105*, *134*, *187*, *225*, *233*, *291*, *301*, *303*, *347*, *355*, *497*, *499*, *500*, *507*, *516*, *531*, *540*

xmuLabel_Matrix, 496

xmuLabel_Matrix(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *499*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuLabel_MATRIX_Model, 498

xmuLabel_MATRIX_Model(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *501–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuLabel_RAM_Model, 500

xmuLabel_RAM_Model(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *502–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuMakeOneHeadedPathsFromPathList, 501

xmuMakeOneHeadedPathsFromPathList(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501*, *503–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*,

564, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuMakeTwoHeadedPathsFromPathList, 503

xmuMakeTwoHeadedPathsFromPathList(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501*, *502*, *504–506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuMaxLevels, 504

xmuMaxLevels(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–503*, *505*, *506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuMI, 505

xmuMI(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–504*, *506*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuMinLevels, 506

xmuMinLevels(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–505*, *508*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuPropagateLabels, 507

xmuPropagateLabels(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *509*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuRAM2Ordinal, 509

xmuRAM2Ordinal(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *511*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594–598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*

xmuTwinSuper_Continuous, 510

xmuTwinSuper_Continuous(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*,

498, *499*, *501–506*, *508*, *509*, *513*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *590*, *592*, *594*, *595*, *597*, *598*, *600–602*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–632*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmuTwinSuper_NoBinary, 512
xmuTwinSuper_NoBinary(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *515*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *592*, *594*, *595*, *597*, *598*, *600*, *601*, *603*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmuTwinUpgradeMeansToCovariateModel, 514
xmuTwinUpgradeMeansToCovariateModel(), *60*, *238*, *296*, *360*, *378*, *380*, *406*, *473*, *475*, *494*, *498*, *499*, *501–506*, *508*, *509*, *511*, *513*, *518*, *520–524*, *526–529*, *531*, *532*, *534*, *536–538*, *540–542*, *545*, *546*, *548*, *550*, *552*, *553*, *555*, *559*, *562*, *564*, *565*, *569–571*, *573*, *575*, *577*, *581*, *584*, *585*, *587*, *588*, *591*, *592*, *594*, *595*, *597*, *598*, *600*, *601*, *603*, *604*, *608*, *610*, *612–614*, *616*, *618–620*, *622–624*, *626*, *627*, *629–631*, *633*, *634*, *636*, *638*, *639*, *641–643*, *645*, *646*, *648*, *650–652*, *654–658*
xmuValues, 516
xmuValues(), *105*, *134*, *187*, *225*, *291*, *301*, *347*, *355*, *495*