umx: Twin and Path-based Structural Equation Modeling in R

Timothy C. Bates[1], Hermine Maes[2], and Michael C. Neale[2]

Abstract: Structural equation modeling (SEM) is an important research tool, both for path- based model specification, common in the social sciences, and also matrix-based models in heavy use in behavior genetics. We developed umx to give more immediate access, concise syntax and helpful defaults for users in these two broad disciplines. umx supports development, modification, and comparison of models, as well as both graphical and tabular output. The second major focus of umx, behavior genetic models, is supported via functions implementing standard multi-group twin models. These functions support raw and covariance data, including joint ordinal data, and give solutions for ACE models including support for covariates, common- and independent-Pathway models, and Gene × Environment interaction models. A tutorial site and question forum are also available.

Keywords: structural equation modeling, OpenMx, path models, twin models, R.

1. University of Edinburgh email: tim.bates@ed.ac.uk
2. Virginia Commonwealth University

umx: Twin and Path-based Structural Equation Modeling in R

Timothy C. Bates, Hermine Maes, and Michael C. Neale

Abstract

Structural equation modeling (SEM) is an important research tool, both for path-based model specification, common in the social sciences, and also matrix-based models in heavy use in behavior genetics. We developed **umx** to give more immediate access, concise syntax and helpful defaults for users in these two broad disciplines. **umx** supports development, modification, and comparison of models, as well as both graphical and tabular output. The second major focus of **umx**, behavior genetic models, is supported via functions implementing standard multi-group twin models. These functions support raw and covariance data, including joint ordinal data, and give solutions for ACE models including support for covariates, common- and independent-Pathway models, and Gene × Environment interaction models. A tutorial site and question forum are also available.

Keywords: structural equation modeling, OpenMx, path models, twin models, R.

1. Introduction

Structural equation modeling (Jöreskog 1969b) enables modeling with latent and measured variables, and allows researchers to realize the power of causal modeling (Pearl 2009), and it has grown in importance (Hershberger 2003). Despite the utility of SEM, Learning, implementing, and interpreting these techniques has remained a bottle-neck for many researchers, especially for more complex multiple-group models common in advanced fields such as behavior genetics. The advent of modular software such as **OpenMx** has provided tools for software solutions in this field (Boker, Neale, Maes, Wilde, Spiegel, Brick, Spies, Estabrook, Kenny, Bates, Mehta, and Fox 2011; Neale, Hunter, Pritikin, Zahery, Brick, Kirkpatrick, Estabrook, Bates, Maes, and Boker 2016). The present paper describes **umx**, a package designed to give more immediate access, concise syntax and helpful defaults for path-based SEM, together with a set of high-level functions implementing matrix-based multi-group twin modeling. Practical examples of **umx** usage are given. Users wanting to learn about twin-modeling in **umx** may wish to skip to the section "Twin modeling in **umx**".

1.1. Existing SEM packages

While a number of closed-source commercial applications exist, (e.g., Mplus (Muthén and Muthén 1998-2016), SAS proc calis (SAS Institute Inc. 2003), SPSS Amos (IBM Corp 2013), and GLAMM in STATA (Stata Corp LP 2016)), there are now 3 open-source R packages for performing SEM: **sem** (Fox, Nie, and Byrnes 2014); **lavaan** (Rosseel 2012); and **OpenMx** (Boker *et al.* 2011; Neale *et al.* 2016). As is common in R, these interoperate with an ecosys-

tem of packages, such as semTools (semTools Contributors 2016), Onyx (von Oertzen, Brandmaier, and Tsang 2015), ctsem, EasyMx, ifaTools, lvnet, metaSEM, and semtree to provide additional features.

sem includes functions for fitting general linear structural equation models, including both observed and latent variables using the RAM (McArdle and Boker 1990) approach. It also allows fitting structural equations in observed-variable models by two-stage least squares. Models are input using an "Arrow specification" with paths described in an intuitively straightforward notation encompassing regression coefficients ("A -> B"), variances ("A <-> A") and covariances ("A <-> B"). Models can be optimized against a maximum-likelihood objective assuming multi-variate normality as well as multivariate-normal full-information maximum-likelihood in the presence of missing data, with alternative objectives including generalized least squares or user-specified objective functions. **sem** also implements multi-group models.

lavaan implements a similar string-based syntax for model description, and comparable multi-group capability and a range of estimators including robust ML, and variants of WLS. It outputs standard errors (SEs) including robust and bootstrap SEs, along with standard fit indices, and statistics such as Satorra-Bentler, Satterthwaite, and Bollen-Stine bootstrap. **lavaan** can handle missing data via FIML estimation. It allows the use of linear and nonlinear equality (and inequality) constraints via a string syntax, e.g., to equate model parameters "a1" and "a2", the user includes the following in their model statement: "a1 == a2". More complex statements are supported, for instance "a1 + a2 + a3 == 3". As of version 0.5, **lavaan** supports models with mixtures of binary, ordinal and continuous observed variables. Exogenous categorical variables are supported via dummy variables, with additional variables being created to represent the levels of nominal measures with more than 2 levels. Modeling of binary and ordinal endogenous categorical variables (but not nominal) is supported by a three-stage WLS approach. **lavaan** outputs results in a format familiar to Mplus and EQS users, as well as the ability to translate Mplus code into **lavaan** format via `mplus2lavaan`.

OpenMx provides a sophisticated kit of basic objects for building structural equation models including modeling via arbitrary matrices, algebras, constraints, and fit-functions. It also supports "RAM" (McArdle and Boker 1990) and LISREL (Jöreskog 1969a) path-based models. It accepts both summary and raw data and arbitrary mixtures of continuous, ordinal and binary data. Full-information maximum likelihood (FIML) analysis with missing data is supported, as is WLS. With raw data, models may include row-specific values (definition variables). Multiple-group models are supported, and constraints and equalities may be implemented via label-based equating and algebra-based linear and non-linear constraint specification. The **OpenMx** package includes two open-source optimization packages–CSOLNP and SLSQP–and can use the closed-source NPSOL optimizer. **OpenMx** has developed a strong following among geneticists and twin researchers, reflected in several hundred citations in published projects, many of which rely on testing complex models, often with constraints, using data comprising mixtures of binary, ordinal, and continuous data, with missingness, and wide-format data comprising multiple genetically related groups (in particular identical and fraternal twins, siblings, parents, grand-parents, offspring, adoptive parents), with data nested in these family structures.

1.2. Accessible modeling with concise syntax and helpful defaults

The **umx** package evolved over the last 6 years in response to modeling demands experienced

in practical path based modeling and matrix-based behavior genetics structural modeling, and in teaching SEM. Its concise syntax aids in time-constrained lessons, and affords researchers the ability to rapidly implement and modify new models, while high-level implementations of complex models increase the speed and reliability of behavior genetics modeling. Attention was paid to smart defaults – for instance setting start values and automatically labeling paths – as well as ensuring functions handle a wide range of inputs, so that users can offer up a wide range of data and get the results they expect. For instance, models can accept not only continuous data, but mixtures of ordinal and continuous data, with the **umx** functions handling the error-prone process of setting up threshold matrices, and integrating these with latent variables. To aid learning, and help users correct coding mistakes, help files contain substantial, well-commented practical code examples oriented toward being used directly in class instruction or self-directed learning. Moreover, **umx** functions include significant error-checking. Error reporting tries explain both "why" (for instance not just saying there was a type-match error, but what type was provided, and which was expected) with a "how-to-fix" orientation – where possible explaining to users what the most likely fix is for the error encountered.

Finally, **umx** provides methods for generating graphical and tabular output suitable for publication (e.g. Table 2). Function names and parameters have been refined based on feedback and a drive for consistency and memorability. Where appropriate, these are implemented as S3 methods well known to **R** users: for instance `plot` and `residuals`. A number of papers using the package have been published (Archontaki, Lewis, and Bates 2013; Ritchie and Bates 2013) and **umx** is under active development – with updates on CRAN every month or two, and a road-map of future extensions of the twin modeling functions including use of WLS, and 5-group twin models, non-Cholesky-based implementations of models, multivariate correlation-based sex-limitation models (Maes, Sullivan, Bulik, Neale, Prescott, Eaves, and Kendler 2004).

In total, the package includes approximately 140 high- and low-level helpers for such tasks as data processing, creating and editing data structures, updating model parameters, and reporting. The functions provided by **umx** may be grouped under three headings:

1. Model building and reporting functions.
2. Functions implementing behavior genetic models.
3. Wrappers and helpers to simplify or enhance model building and data wrangling.

In the next section, **umx**'s path-based functions are introduced in the context of practical models, such as that shown in Figure 1. The second major section covers behavior genetic modeling in **umx**. Not detailed here, we briefly note some of the additional helper functions available in **umx**. While the approximately 100 wrapper and helper functions make little claim to innovation, they offer increased speed of coding, reduced errors, and more readable code. For instance, `umxMatrix`, is a wrapper for `mxMatrix` that places the matrix name as the first parameter (increasing readability), and uses `umx_label` to automatically add labels in a standard format, often halving code size and effort. Other helpers simplify repetitive tasks, such as generating lists of twin-variables from base names, and residualizing family-based data.

1.3. Installing the package

Installing **umx** can be done using the standard R-code to access the CRAN version of the package.

```
install.packages("umx")
```

This document assumes **umx** version 1.8.0 or higher. The current package version can be shown with:

```
umxVersion("umx")
```

Developer versions are available on github, using:

```
install_github("tbates/umx")
```

umx also aids installing custom versions of OpenMx via the `install.OpenMx()` function.

2. Path-based Models in umx

2.1. Path-based models using `umxRAM` and `umxPath`

In **umx**, a path-based structural model consists of a model container, some data, and a collection of paths. The model container is the `umxRAM` function. This specifies the `data`, a name for the model, options such as `autoRun`, and, importantly, a collection of `umxPath`s specifying the paths that make up the model. As coding is often best learned by doing, we introduce these functions via building the Confirmatory Factor Analysis (CFA) model shown in Figure 1. To build the CFA model shown in Figure 1, we will specify three things:

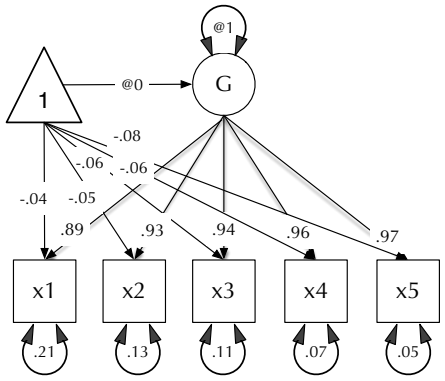


Figure 1: Example CFA path diagram, showing standardized path estimates.

1. An arbitrary **name** – "CFA"

2. The **data** (in this case a built-in dataset `demoOneFactor` containing 5 correlated variables x1:x5). In keeping with familiar R functions such as `"lm"`, the data to be modeled are provided via the `data` parameter. The `umxRAM` function can accept a data-frame, covariance matrix, or `mxData` as data.

3. The **paths** of which the model is composed. The paths required in this model are those needed to create a latent variable "G" with mean of 0 and variance of 1, the five manifest variables x1:x5, each with freely-estimated mean and variance, and 5 single-headed paths from G to each of the manifest variables.

Building, running and saving this model in its run state for possible modification or comparison, along with producing summary data and a plot, can be done in 5-lines (excluding comments, loading the library and data) of model code shown below as "CFA Code".

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 = c("x1","x2","x3","x4","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)
)
```

On execution, this code block builds the model, echoing to the console which latent and manifest variables were created. By default, `umxRAM` also runs the model, plots it graphically, and prints a brief fit statistic summary:

$\chi^2(2485) = 7.4, p = 0.193; CFI = 0.999; TLI = 0.999; RMSEA = 0.031$

Two aspects of this code are noteworthy. First, we did not explicitly specify a list of manifest and latent variables contained in the model. Instead, **umxRAM** maps these from the data, with any variable name not found being assumed to be a latent variable. As with **lm**, unused variables are excluded from the model. Secondly, we did not need to specify starting values for the parameters. **umx** generates feasible start values, with manifest variable means set to the observed means, manifest variances set to 80% of the observed variance of each variable. Single-headed paths are set to a positive value (.9). Estimation thus begins from close to an independence model.

umxPath in detail

`umxPath` creates paths in a model using a compact syntax, describing a full range of path types and settings. A complete list of `umxPath` keywords is set out in Table 1.

For example, to create a 2-headed path allowing a variable "a" to covary with variable "b", with the covariance fixed at the value 1, the user would add a call to `umxPath("a", with = "b", fixedAt = 1)`. To specify a mean for a manifest or latent variable, say the variable "b", the `umxPath` code would be `umxPath(means = "b")`. As specifying the mean and variance of variables is such a common task, `umxPath` supports doing both in one line with the `v1m0` and `v.m.` parameters we saw used in the CFA model above to specify normalized (fixed mean = 0, variance = 1) or freely-estimated variables respectively.

umxPath syntax	Result
`umxPath(means ='b')`	Add means expectation for variable b.
`umxPath(var = c('a','b'))`	Variance for "a" and for "b". Value free.
`umxPath('a', with ='b')`	Covariance of "a" with "b". Value free.
`umxPath('a', with ='b', fixedAt = 1)`	2-headed path a↔b. Value fixed at 1.
`umxPath('a', with ='b', freeAt = 1)`	2-headed path a↔b. Value free, but started at 1.
`umxPath('a', to = vars, firstAt = 1)`	Path from "a" to vars, with first path fixedAt 1, remainder free.
`umxPath(v1m0 ='g')`	Fix variance of "g" at 1, mean at 0.
`umxPath(v.m0 ='g')`	Free variance, mean fixed at 0
`umxPath(v.m.  = c('wt','mpg'))`	Estimated variance and mean.
`umxPath(v0m0 = c('wt','mpg'))`	Variance and mean fixed at zero.
`umxPath(c('a','b','c'), forms="A")`	Define a formative latent variable ("A"). with incoming paths from a, b, & c to A, variances for a, b, & c, and covariances among them.
`umxPath(unique.pairs=c('A','B'))`	Create paths A→A, B→B, A→B
`umxPath(unique.bivariate = c('A','B','C'))`	Create paths A↔B, B↔C, A↔C
`umxPath(fromEach=c('A','B'))`	Create A→B, B→A
`umxPath(Cholesky=c('A','B'), to=c('a','b'))`	Create the lower-triangle (Cholesky) paths: A→a, A→b, B→b
`umxPath(defn='A', label="age")`	Create latent variable A, var@0 mean fixed, value = data.age

Table 1: Table of all umxPath options.

labels

The **umx** package automatically adds labels to all the parameters of any model. These labels allow the user not only to get and set the values of parameters by label, but also to equate parameters by setting their labels to be the same (see Section 2.3.3 on `umxModify` below).

One-headed paths are labeled "fromVariable_to_toVariable". Thus the path `umxPath("IQ", to = "earnings")` would be labeled "IQ_to_earnings". For two-headed paths, "_to_" is replaced with "_with_". This is consistent with Onyx (von Oertzen *et al.* 2015). Future versions of **umx** may extend the labeling scheme to encode more information in each label, for instance the extra information captured in LISREL-type models.

For matrix-style models (such as the behavior genetic twin models described in Section 3),

umx uses a more general labeling scheme, in which labels are a concatenation of the matrix name, an underscore, the letter "r" for row, the row number, followed by the letter "c" for "column" and, finally, the column-number of the matrix cell. The following code returns the $labels slot of a matrix "means" to show an example of this labelling scheme:

```
x = umxLabel(mxMatrix(name="means", "Full", ncol = 2, nrow = 2))
x$labels
```

```
labels
     [,1]          [,2]
[1,] "means_r1c1"  "means_r1c2"
[2,] "means_r2c1"  "means_r2c2"
```

2.2. Controlling graphical and summary output

A parameter table (see Table 2) can be requested by setting `showEstimates` = "std" in `umxRAM`. More control and creation of summary output tables and plots is possible, however, using `umxSummary` on the model returned from `umxRAM`.

umxSummary

Output from `umxSummary` can be as little as a line of fit information (the default), or a table of standardized or raw estimates (requested with `show = "std"` or `show = "raw"`). Example output from `umxSummary(cfa1, show = "std")`) is shown in Table 2.

name	Std.Estimate	Std.SE	CI
G to x1	0.89	0.01	0.89 [0.87, 0.91]
G to x2	0.93	0.01	0.93 [0.92, 0.95]
G to x3	0.94	0.01	0.94 [0.93, 0.95]
G to x4	0.96	0.00	0.96 [0.95, 0.97]
G to x5	0.97	0.00	0.97 [0.97, 0.98]
x1 with x1	0.21	0.02	0.21 [0.17, 0.24]
x2 with x2	0.13	0.01	0.13 [0.11, 0.15]
x3 with x3	0.11	0.01	0.11 [0.09, 0.13]
x4 with x4	0.07	0.01	0.07 [0.06, 0.09]
x5 with x5	0.05	0.01	0.05 [0.04, 0.07]
G with G	1.00	0.00	1 [1, 1]

Table 2: Example output table from `umxSummary`.

This report is customizable, with parameters to `filter` non-significant ("NS") or significant ("SIG") parameters and to show or hide `SE` and `RMSEA_CI` columns.

The output-type of the summary table also be selected: e.g. markdown, latex, or html. The default output can be changed with the `umx_set_table_format` function. Markdown is easy to read in the console, or to include in a reproducible document. Html output is opened in a browser for easy copying into a word processor.

plot

`umx` includes S3 `plot` methods for RAM and twin models. These rely on the dot language, invented at Bell Laboratories (as was **S**: the ancestor of **R**) to specify graphs in a text-based format of edges and vertices, analogous to the way that the latex system separates content from layout. Plots from `umx` are displayed in the user's browser courtesy of the **DiagrammR** package.

The `plot` function has a number of options to customize output. For instance the plot in Figure 1, showed the fixed path representing the variance of "G" using `fixed = TRUE`. We could also not display the means model (paths to the "one" triangle), e.g.:

```
plot(cfa1, means = FALSE, fixed = TRUE)
```

`plot` can also standardize the model (`std = TRUE`), and control numeric precision (with the `digits` parameters). Other options include control over how residuals are drawn. By default, these appear as conventional circles, with double-headed arrows. To cope with situations where diagrams imported into other applications fail to render the residual circles, these can be set to simple one-headed labeled arrows using `resid = "line"`. This code snippet shows these options in use:

```
plot(cfa1, std = TRUE, digits = 3, resid= 'line')
```

Plotting is useful not only to display final models, but may be used during model construction to verify or test what the code build to date is specifying. To make this easier when the user is is simply "sketching out" ideas, instead of simulating data needed for the model, the user can simply offer up a list of variable names expected to be encountered, and write the code they wish to play with and visualize. For instance to explore the `unique.pairs` construction of `umxPath`, the following model could be plotted:

```
m1 = umxRAM("play", data = c("A", "B", "C"),
        umxPath(unique.pairs = c("A", "B", "C"))
)
```

For publication purposes, further processing of the diagram is often desirable. This is done by editing the file created by plot. The graph is written to a text-file, by default "<model name>.gv". This can be overridden by setting the `file =` parameter to the desired file name. This file can be edited using either open-source graph visualization software available at http://www.graphviz.org, or closed software such as Omnigraffle®, or Visio®.

Inspecting model parameters and residuals.

Often we wish to see some (but not all) estimates from a model. As shown above, `umxSummary` can filter output according to whether parameters are significant or not, i.e., `umxSummary(cfa1, show="std", filter = "SIG")`. The generic `coef` function can return a list of model coefficients. `umx` provides the convenience function `parameters`, which adds support for filtering by name and value and returns the parameters and estimates of a model as a table.

```
# Show parameters, above .5, with label containing `x2'
parameters(cfa1, "above", .5, pattern= "x2")
```

```
     name Estimate
2 G_to_x2      0.5
```

Another common need in modeling is to inspect residuals. **umx** implements a `residuals` method. Table 3 shows these for the `cfa1` model. The user can zoom in on specific values with the suppress parameter. For instance this call will hide residuals smaller than the value of .005.

```
residuals(cfa1, suppress = .005)
```

	x1	x2	x3	x4	x5
x1	.	.	0.01	.	.
x2	.	.	0.01	-0.01	.
x3	0.01	0.01	.	.	.
x4	.	-0.01	.	.	.
x5	.	.	.	.	.

Table 3: Result of the `residuals` function on model `cfa1`.

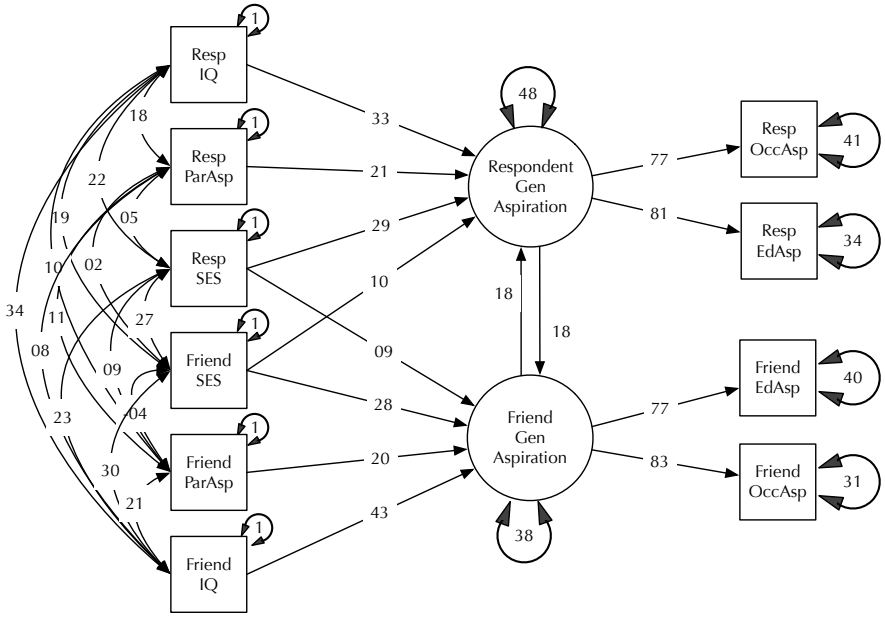


Figure 2: A model of Aspiration (modified from Duncan, Haller, and Portes 1968).

2.3. Modifying and comparing models

Model comparison and modification is a key modeling task (MacCallum 2003). Here, we introduce the **umx** functions enabling these tasks in the context of a classic example, modified from Duncan *et al.* (1968) (See Figure 2). This is a moderately complex model, often used in teaching because of the range of structural model elements it displays.

The code for this model introduces some new features, and reinforces others already discussed. It is presented in three parts. First, reading in data. This uses a helper function included in **umx** for converting lower matrices to symmetrical, full matrices. There are many dozens of helpers such as this included in **umx**, and readers who adopt the package in their work will find these documented in the package help, where they are also grouped in functional families.

```
# 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)
```

Next, we make some useful lists of variables to use when creating paths. These will help reduce errors, and increase the readability of code.

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

Finally, we build the model using the data and variable lists created above.

```
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 to influence each 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.
        # note also, this path does not display as 1 in Figure 2, because
        # Figure 2 shows the standardized paths.
        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

The model fits well ($\chi^2(22) = 24.59, p = 0.317; CFI = 0.997; TLI = 0.993; RMSEA = 0.02$). However, for theoretically interesting models, the user will typically wish to test different versions of the model, dropping or adding paths corresponding to alternative hypotheses. The `umxModify` function supports this task, updating the model, giving the model a new name, running the new model, and printing a table comparing fits of the old and new models.

In its simplest use, paths to be dropped are passed in as a vector or labels to the `update` parameter. The following code-snippet will run a modified version of our example CFA, with the paths "RespLatentAsp_to_FrndLatentAsp" and "FrndLatentAsp_to_RespLatentAsp" dropped (fixed at zero). The new model is renamed to reflect this, and fit-comparison table

printed (See Table 4)

```
# List the paths to drop
pathList = c("RespLatentAsp_to_FrndLatentAsp", "FrndLatentAsp_to_RespLatentAsp")
# Modify duncan1 model, requesting a comparison table
duncan2 = umxModify(duncan1, update = pathList,
        name = "No_influence", comparison = TRUE)
```

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with
Duncan	33				-19.411	
No influence	31	19.509	2	< 0.001	-3.902	Duncan

Table 4: Comparison of effect of dropping reciprocal influence from the Duncan model.

By default, updated paths are fixed at zero, but any value is possible via the `values`. Regular expressions can be used to pick-out parameters that match a given text-pattern. A regular expression is a search pattern with powerful features greatly exceeding normal wildcards. While regular expressions are somewhat complex to learn, they repay the user in a wide range of computing applications, and allow more compact syntax.

As an instance of using regular-expressions in updating a model, consider a user who wants to test the effect of dropping all paths from "G" in the `cfa1` model. By label, these all begin with the string "G_to_", followed by a variable name, e.g. "G_to_x1". Rather than listing each label in the update parameter, a regular expression that matches all instances could be used. Something as simple as "G_to_.*" would work in this case, (with ".*" matching any characters following "G_to_". For more explicit safety, the regular expression anchor-at-start character ("^") could be used to ensure the match starts at the first character of the label. Whereas "G_to_.*" would match labels such as "AG_to_x1", using the carat to anchor the expression at the first character "^G_to.*" prevents this. An example in **R** code would be:

```
cfa2 = umxModify(cfa1, regex = "^G_to.*")
```

`umxModify` can also be used to add or replace objects in models, for instance, if a `umxPath` is passed into `update`, the path will added to the model. Finally, `umxModify` can be used to equate two paths, by setting the `master` parameter to one of the labels, and update to the other label (or list of labels). The master and update lists will then have the same labels, equating these paths. Note: **umx** includes the `umxEquate` function dedicated to equating paths with defaults which make this easier to use during model building, rather than modification (see Section 2.3.3).

Comparing models

The table of model comparison output from `umxModify` is shown in Table 4. This can be called directly at any time using the `umxCompare` function. This takes one or more base models and one or more comparison models, and prints a table of model comparisons including a column directing the reader to the base model for the comparison, formats values in a publication style, with control over precision via the standard `digits` parameter. It can report the results to the console, or else open a browser table for pasting into a word processor. Printing to

console is by default in markdown style, (change this with `umx_set_table_format` set to one of latex, html, markdown, pandoc, or rst). If `report = "inline"` is selected, it also reports the output as a one or more sentences to help the user describing the results in the text of a paper. An example of using this function is below. In the plain-English description (the new model name is used to describe what was done, and will need some editing): For instance, the output from the code snippet below is "The hypothesis that no reciprocal influence was tested by dropping no_reciprocal_influence from Duncan. This caused a significant loss of fit ($\chi^2(2) = 19.51, p =< 0.001 : AIC = -3.902$)."

```
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

In addition to dropping or adding parameters, a second common task in modeling is to equate parameters - setting two or more paths to have the same value. 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. As of version 1.9.0 of `umx`, this can be done using the `umxModify` , which is useful for one-step modifications. However there is also a dedicated function `umxEquate` which is useful when building models as, by default, it does not re-run the updated model.

For example, based on the `duncan1` model, we might test if the effect of IQ on aspiration levels can be equated for respondent and friend. This is done by making a new model in which they are equated and comparing the two models as follows:

```
# Use parameters to quickly search the model and find the paths to equate.
parameters(duncan1, pattern = "IQ_to_")
#                      name Estimate
# RespIQ_to_RespLatentAsp     0.25
# FrndIQ_to_FrndLatentAsp     0.35

# Modify duncan1 model, requesting a comparison table
duncan3 = umxEquate(duncan1, name = "Equate IQ effect",
        master = "RespIQ_to_RespLatentAsp",
        slave = "FrndIQ_to_FrndLatentAsp"
)

# equivalently, with autoRun and plot by default
duncan3 = umxModify(duncan1, name = "Equate IQ effect", comparison = TRUE,
        master = "RespIQ_to_RespLatentAsp",
        update = "FrndIQ_to_FrndLatentAsp"
)
```

There are numerous additional functions in the **umx** library facilitating model interrogation, some are discussed at the end of this paper. For a complete listing, however, we direct the

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with Model
Duncan	33				-19.41	
Equate friend's IQ effect	32	2.265	1	0.132	-19.145	Duncan

Table 5: Comparison of equating IQ effects in respondents and friends.

reader to the package help "?umx", and to the tutorial site for **umx**: http://tbates.github.io Next, we turn to twin modeling.

3. Behavior genetic twin modeling

A major goal of **umx** was to provide support for common twin-models, including Cholesky, Common-Pathway, Independent-Pathway, and Gene ×Environment moderation models, with full support for tables and graphical output suitable for publication, as well as support for model comparison and modification. These functions are outlined below, beginning with a brief introduction to a common twin model (see Figure 3).

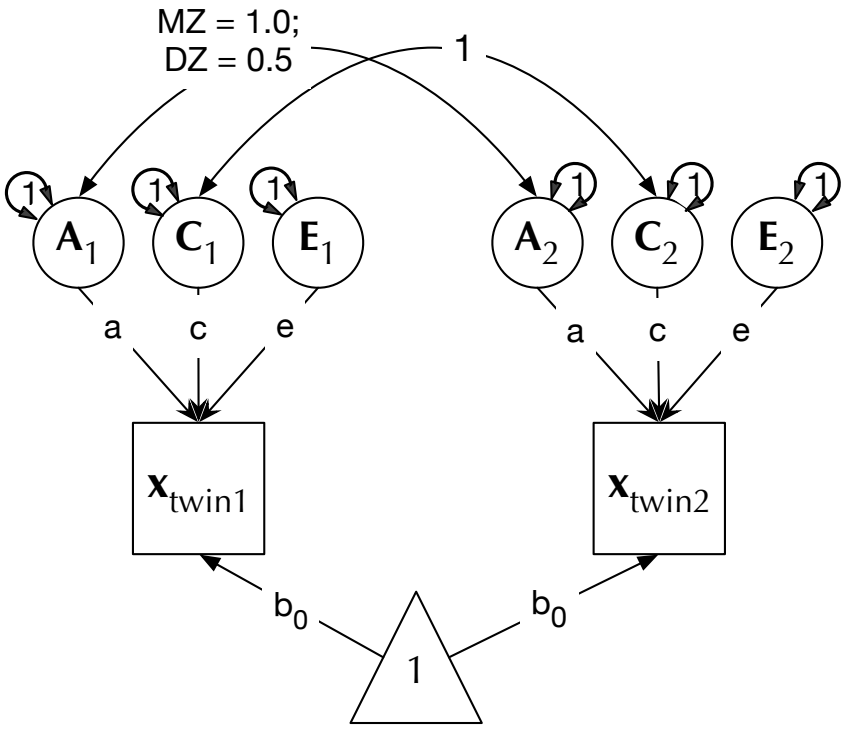


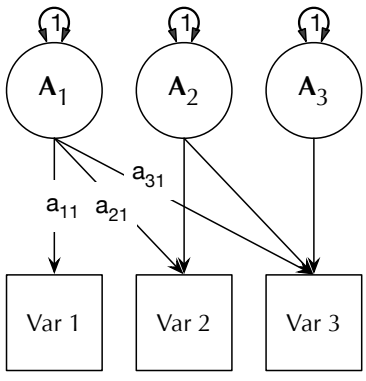
Figure 3: Cholesky decomposition (ACE model) of variance in behavior (x) in twin-1 and twin-2, decomposed into A (additive genetic), C (Shared environmental) and E (uniques environmental) components. There are two groups in the model: Identical (Monozygotic) twins and Fraternal (Dizygotic) twins.

3.1. Twin models and matrix-based modeling

Twin and family modeling takes advantage of classes of genetic and environmental covariance present in nature. For example, "identical" or monozygotic (MZ) twins who share 100% of their genes, and fraternal (DZ: dizygotic) twins who share, on average, half of their genes, siblings, who also share 50% of their genes, but differ in year of birth, adoptees who are unrelated genetically to their rearing family, but who share that family environment. These classes of relatedness allow researchers to specify proposed structural and measurement models of their phenotype(s) of interest and to model many types of relatedness using multiple-group models, which are fitted simultaneously to arrive at estimates of genetic and environmental variance consistent with the covariances found among the variables in the different group of relatedness (Yong-Kyu Kim 2009; Knopik, Neiderhiser, DeFries, and Plomin 2016; Neale and Maes 1996). A classic approach to modeling data such as these is shown in Figure 3). This shows the decomposition of variance in behavior "x" measured in two related individuals – either Monozygotic twins or Dizygotic twins – into additive genetic (A), shared environmental (C) and unique environmental (E) components. There are two groups in the model, one for each of the datasets. The correlation between A_1 and A_2 is fixed at 1 (all variable genes shared) in the MZ group, and at .5 for the DZ group (reflecting their 50% sharing of variable genes).

For efficiency, twin models in **umx** are implemented using a matrix based approach (rather than the path-approach used in Section 2. Each of the A, C, and E components are modeled using square matrices, with the same number of rows and columns as there are variables under analysis. These in turn are formed from the product of lower-triangle matrices a, c, and e respectively, multiplied by their transpose to form the variance component matrices (e.g. $A = aa'$). The utility of the Cholesky factorization is that the product of a lower triangular matrix and its transpose, e.g. aa' is guaranteed to be positive definite. The total phenotypic (observed) covariance is modeled as the sum of these three components: $V_p = A + C + E$.

Figure 4 shows how the path diagram is mapped onto matrices in the case of the Additive genetic (A) matrix. As shown, the latent additive genetic variables form the columns of the A matrix. The variables are mapped to rows of this matrix, and paths from a latent variable to a manifest variable appear in the appropriate cell, for instance the value of the path from A_2 to var_2 appears in cell $A[2, 2]$ of matrix A.



3×3 matrix-form of the genetic (A matrix) paths, with labels as applied by `umxLabel`.

	A1	A2	A3
Var 1	a_r1c1		
Var 2	a_r2c1	a_r2c2	
Var 3	a_r3c1	a_r3c2	a_r3c3

Figure 4: Genetic components of a tri-variate ACE model (C and E not shown) in graphical (left) and matrix forms.

3.2. The ACE Cholesky model

`umxACE` supports a basic model in behavior genetics, known as the multivariate ACE or Cholesky model (Neale and Maes 1996). The ACE model decomposes phenotypic variance into additive genetic (A), unique environmental (E) and, one of either shared-environment (C) or non-additive genetic effects (D). This latter restriction emerges due to confounding of C and D when data are available from only MZ and DZ twin pairs reared together. The Cholesky or lower-triangle decomposition allows a model that is both sure to be solvable and it provides a saturated model against which models with fewer parameters can be compared. This model creates as many latent A C and E variables as there are phenotypes, and, moving from left to right, decomposes the variance in each component into successively restricted factors (see Figure 4).

This function can be used to fit an ACE model to a single variable to decompose its variance as shown in the example in Section 3.2.1. It can also be applied to two (bivariate) or more (multivariate) models, decomposing not only the variance but also the covariance of the traits.

The `umxACE` function flexibly accepts both raw data and summary covariance data, offering up suitable covariance matrices to `mzData` and `dzData`, and entering the number of subjects in each via `numObsDZ` and `numObsMZ`. 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. This involves setting up threshold matrices for binary and ordinal data, which are modeled as thresholds applied to underlying latent variables.

It is often desirable to include covariates within twin models. We currently support ACE models with fixed covariates (covariates included in the means model) for continuous variables, and as random effects (i.e., modeled in the covariance matrix, allowed to covary with the main variables of interest (Neale and Martin 1989)). In the next major version of **umx** this functionality will be enhanced to allow modeling covariates in ordinal and mixed data across all twin models.,

`umxACE` also supports weighting of individual data rows. In this case, the model is estimated for each row individually, the likelihood of each row is multiplied by its weight, and the logarithm is taken. These weighted log-likelihoods are then summed to form the model log-likelihood, which is to be maximized (by minimizing the $-2log(Likelihood)$). In addition, `umxACE` 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. This weighting feature is used in Section 3.6.

When it comes to interpretation and graphing, models built by `umxACE` can be plotted and summarized using `plot` and `umxSummary` methods. `umxSummary` can report summary A, C, and E multivariate path-coefficients, along with model fit indices, and genetic correlations. The **umx** package provides custom `plot` methods to handle graphical reporting of twin models, including ACE models, and other models discussed below. This provides output as seen in Figure 4.

ACE examples

We first set up data for a summary-data ACE analysis of weight data (using a built-in example dataset from the Australian twin sample of Professor Nick Martin (Martin, Eaves, Heath,

Jardine, Feingold, and Eysenck 1986; Martin and Jardine 1986).

```
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", ]
```

The next code block uses `umxACE` to build and run the model.

```
ACE1 = umxACE(selDVs = selDVs, dzData = dz, mzData = mz, sep = "")
```

umxACE prints feedback to the console, noting that the variables are continuous and that the data have been treated as raw. It then prints the fit

$$-2 \times log(Likelihood) = 12186.28(df = 4)$$

and outputs a plot of the fitted model (See Figure 5) and a table of the fitted parameters (Table 6). By default the report table is written to the console in the format set by `umx_set_table_format`.

	a1	c1	e1
weight	0.92	.	0.39

Table 6: Standardized path loadings for ACE model.

The tabular output can also be requested at any time with `umxSummary`. By setting `report` = "html", the user can request the results in an html table, opened in the default browser for pasting into a word processor or slide. Whether the parameter table is standardized or not is set using `std = TRUE`. The user can request the genetic and environmental correlations with `showRg = TRUE`. If Confidence intervals have been computed, these can be displayed with `CIs = TRUE`. The user can control output precision using the `digits` parameter. The following snippet creates a tabular summary of the unstandardized model (note, by default it will also plot the model. This can be controlled with `umx_set_auto_plot(FALSE)`). The function help (`?umxACE`) gives extensive examples, including for binary, ordinal, and joint-ordinal cases.

```
# An example (not run) using more control features of umxSummary
# This would print a table of raw parameters to the console in markdown,
# open the table in the browser, set rounding to 3-digits,
# and print a table showing the comparative fit of ACE1 and ACE2

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

umxSummary(ACE1, std = FALSE, report = 'html', digits = 3, comparison = ACE2)
```

Using labels to drop paths in twin models

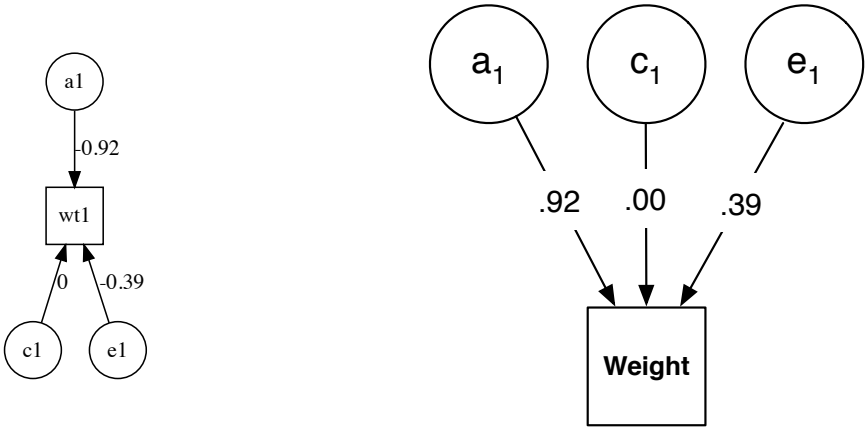


Figure 5: Output from `plot(m1)` for univariate ACE model of Australian weight data, rendered as default in **DiagrammR** (left) and after editing in Omnigraffle (right graphic). note: for simplicity, the unit-variance of A, C, and E are assumed, and not drawn in this figure.

We noted above (Section 2.3) how labels can be used to update a model. For twin models, such model reduction by dropping paths is routine. For instance, to examine the effects of shared or family-level environment, we may wish to update this model by dropping the C (shared environment) paths. The matrix-based model labeling scheme (used in all the twin models) follows a systematic pattern, with the path coefficients underlying the A, C, and E factors stored in matrices named `a`, `c`, and `e`. We can view the `c` parameters with a call to `parameters`.

```
# Show free parameters in model ACE1
parameters(ACE1)
```

This reveals the following 4 parameter labels (and their current (unstandardized) estimated values) (See Table 7). It shows the four matrices for free parameters – a, c, e, and $expMean$ – each with 1 row and 1 column as this is a univariate model.

	name	Estimate
1	expMean_r1c1	58.80
2	a_r1c1	8.19
3	c_r1c1	0.00
4	e_r1c1	4.55

Table 7: Free paths loadings for ACE model.

These labels take the form: matrix name (a, c, or e in this case), `_r` followed by a row number, then a `c` for column followed by the column number of the matrix cell containing the parameter.

A straightforward way to drop the shared environment path we wish to test is to list it in the update option of `umxModify`:

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

As shown in the resulting table of fit comparison table (Table 8), this parameter could be dropped without significant loss of fit.

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with
ACE	4				19515.23	
dropC	3	0	1	1.000	19513.23	ACE

Table 8: Fit comparison of full ACE model and AE model.

We next discuss a nested model, the common pathway model.

3.3. Common-pathway model

The common-pathway (CP) model provides a powerful tool for theory-based decomposition of genetic and environmental differences (Neale and Maes 1996)). This allows one to test, for instance if genes and environment work through a common latent personality trait (Lewis and Bates 2014), or to test claims regarding the specificity or generality of a theorized latent psychological or other construct (Lewis and Bates 2010). `umxCP` supports common-pathway modeling for pairs of MZ and DZ twins reared together to model the genetic and environmental structure of multiple phenotypes according to one or more common pathways. As can be seen at the bottom of Figure 6, each phenotype also has A, C, and E influences specific to that phenotype.

Like the ACE model, the common pathway 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, however, these factors do not act directly on the phenotype. Instead latent A, C, and E impact on latent factors (by default 1) which then account for variance in the phenotypes (see Figure 6).

Note: Often researchers use only a single common pathway. Such models seldom provide a good fit to multivariate data, and `umxCP` supports the more theoretically plausible situation of multiple common pathways simply by setting the `nFac` parameter from its default (1) to the desired number of common pathways to be modeled.

As with `umxACE`, `umxCP` can transparently handle mixtures of continuous and ordinal (binary or multi-level ordered factor data) inputs. Similar options are available for controlling parameters such as the DZ genetic correlation, and `plot` and `umxSummary` implement comprehensive model reporting and graphical output. Note for comparison of this common-pathway model with the independent pathway model to be discussed next, one would set the number of common factors (`nFac`) to 3.

We endeavored to keep the matrix names used in the behavior genetic models memorable (thus, *expMean*, *a*, *c*, and *e* in the ACE model). For the common pathway model, the loadings of a, c, and e, on the common pathway factors are stored in matrices *a_cp*, *c_cp*, and *e_cp*, and the specific loadings in the diagonals of matrices *as*, *ce*, and *es* respectively. The loadings of the common factors onto variables stored in matrix *cp_loadings*. Thus for when the researcher wishes to drop paths, it is in these matrices that they would find the labels to set to zero. For instance, to drop the shared environmental effect specific to variable 2 in a common pathway model, the user would modify the model, updating *cs_r2c2* to be fixed at zero.

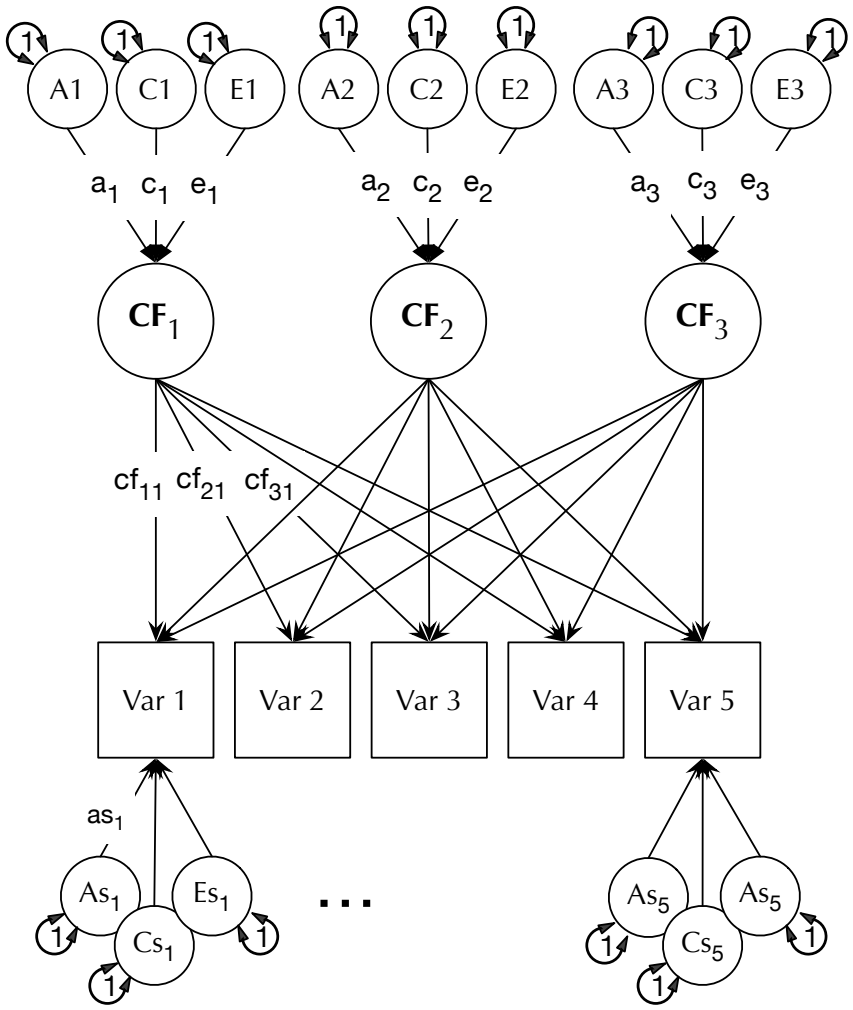


Figure 6: Common pathway twin model with three common factors (CF1, CF2, and CF3), for five measured variables (phenotypes) var 1 thru 5. The specific ACE specific structure is shown at the base of the figure (drawn for only first and last phenotypes.)

Example Common Pathway model

In this example Common Pathway model, we first set up the data for an analysis of height and weight using the built-in twinData data.frame:

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

# Selecting the 'ht' and 'wt' variables
selDVs = c("ht", "wt")
# create dataset consisting of MZ and DZ female twins respectively.
mzData = subset(twinData, zygosity == "MZFF",)
dzData = subset(twinData, zygosity == "DZFF",)
```

The next section shows how `umxCP` allows the user to build the Common Pathway model in one line, followed by calls to `umxSummary`, and `plot` to show the fit, parameter estimates (shown in Tables 9 thru 12), and render and display in **DiagrammR**, graphviz or as pdf (Figure 7). The following code will build and run a Common Pathway model:

```
# Run and report a common-pathway model
CP1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = "")
```

	A	C	E
Common factor 1	0.98	.	0.21

Table 9: Common Pathway model common-factor path loadings.

	CP1
Height	0.85
Weight	0.55

Table 10: Common Pathway model common factor path-loadings for each trait.

	As1	As2	Cs1	Cs2	Es1	Es2
ht1	-0.44		.		-0.29	
wt1	.	0.75	.	.	.	0.37

Table 11: Common Pathway model standardized specific-factor loadings.

A straightforward way to test dropping all the shared environment paths from this model is shown below, with the comparative fit shown in Table 13.

```
# make a list of paths to drop
paths = c("c_cp_r1c1", "cs_r1c1", "cs_r2c2")
CP2 = umxModify(CP1, update = paths, name = "dropC", comparison = TRUE)
```

For users who understand the syntax of regular expression, we can select the same subset of labels using a pattern match:

	rA1	rA2	rC1	rC2	rE1	rE2
height	1.00	0.51	1.00	0.93	1.00	0.16
weight	0.51	1.00	0.93	1.00	0.16	1.00

Table 12: Common Pathway model genetic and environmental correlations.

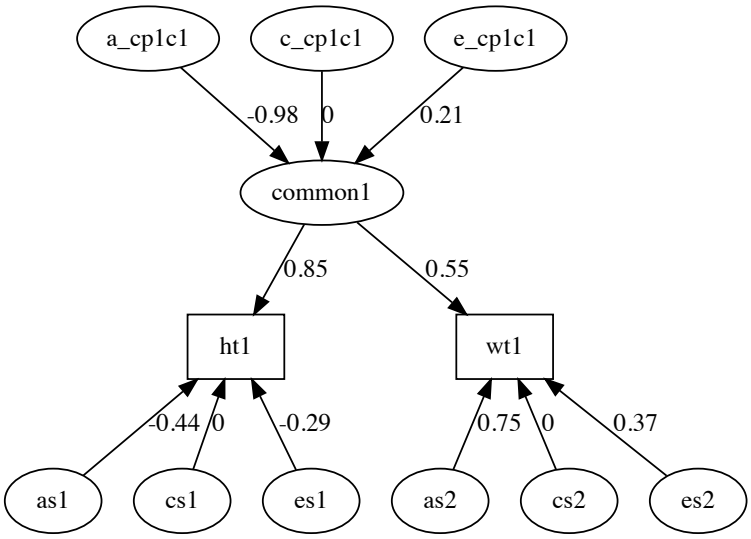


Figure 7: Common Pathway model for height and weight `plot` output.

```
CP2 = umxModify(CP1, regex = "(^cs_)|(^c_cp_)", name = "dropC")
umxSummary(CP2, comparison = CP1)
```

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with Model
CP	13				-751.0117	
dropC	10	2.0414365	3	0.564	-754.9703	CP

Table 13: Fit comparison dropping shared environment effects from Common Pathway model CP1.

3.4. Independent-pathway model

The basic Independent Pathway (IP) model is nested within the 3-factor Common Pathway model (it is essentially a Common Pathway model with each of A, C, and E acting on only one factor. The Independent Pathway models are created using `umxIP`. In this model, one or more latent A, C, and E factors are proposed, each influencing all manifests. In addition, each manifest (phenotype) has A, C, and E influences specific to itself (See Figure 8).

Data input and additional control parameters for `umxIP` closely reflect those available for `umxCP` and `umxACE`, making it easier to move between these functions. Likewise the `plot` and `umxSummary` transparently handle model reporting and graphical output functionality identically to how it is implemented for other models, again lowering the learning curve and increasing productivity. Users can of course implement ACE, CP, and IP models, then submit these as nested comparisons using `umxCompare` to test which of these models is preferred.

3.5. Gene x Environment models

`umxGxE` implements the (Purcell 2002) gene-environment interaction single-phenotype model. In this model, a standard ACE model is modified to include a moderator variable, measured for each subject. This moderator (known as a definition variable because it is defined for each subject), is represented on the path diagram as a diamond (or, in this case, we write the path-formula on the path, including the definition-variable moderator, rather than drawing paths from a diamond). In G×E models, the moderator is included in the means model, removing any heritable effects it has on the DV of interest, and also moderates the A, C, and E path values (See Figure 9). A common application of this type of model has been to examine changes in heritability (and environmentality) across a range of values of a moderator such as, in human twin research, developmental stress or parental socio-economic status (Bates, Lewis, and Weiss 2013; Bates, Hansell, Martin, and Wright 2016).

As with all **umx** functions, examples of this type of analysis are included in the help documents linked to each function. As the moderator is crucial to the estimated model, all rows much have the moderator present, and rows with NA in the moderator are excluded (`umxGxE` will do this for the user if necessary, reporting the quantity of data loss).

Example Gene x Environment model

As usual, we first setup the input data. Because G×E models use definition variables (variables with a value for each subject in the data), rows must not contain NA for any definition variable

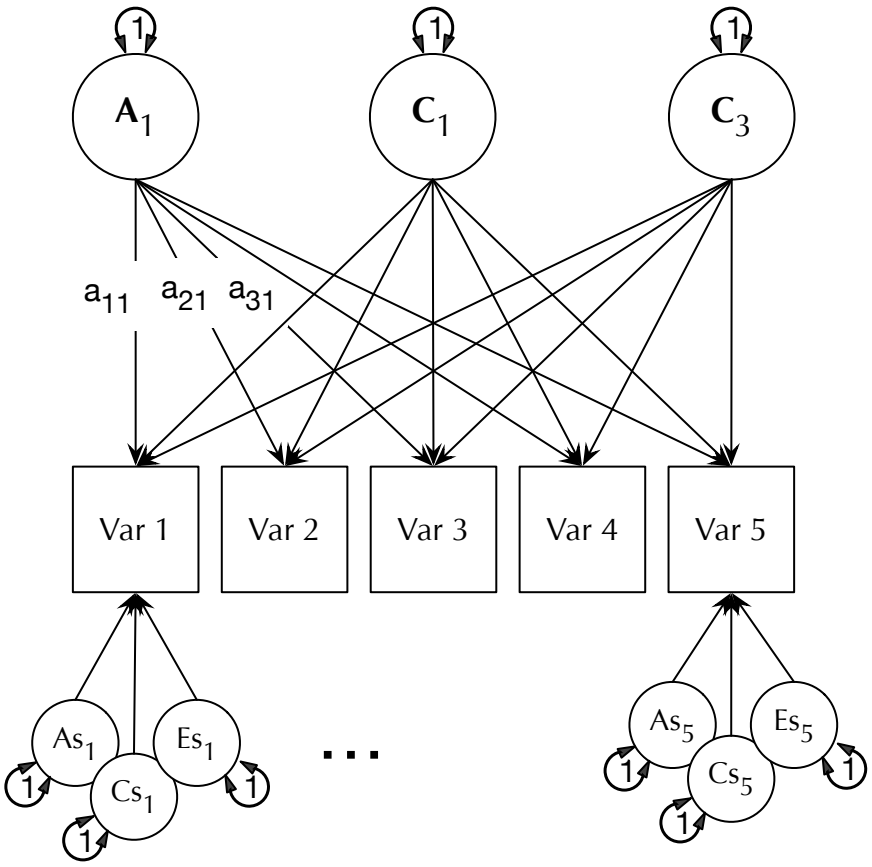


Figure 8: Independent pathways model with a single independent general factor for each of A, C or D and E loading on all phenotypes (Var 1 thru Var 5), and showing the residual specific ACE structure modeled for each phenotype (drawn for variables 1 and 5 only for clarity).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.3354v1 | CC BY 4.0 Open Access | rec: 19 Oct 2017, publ: 19 Oct 2017

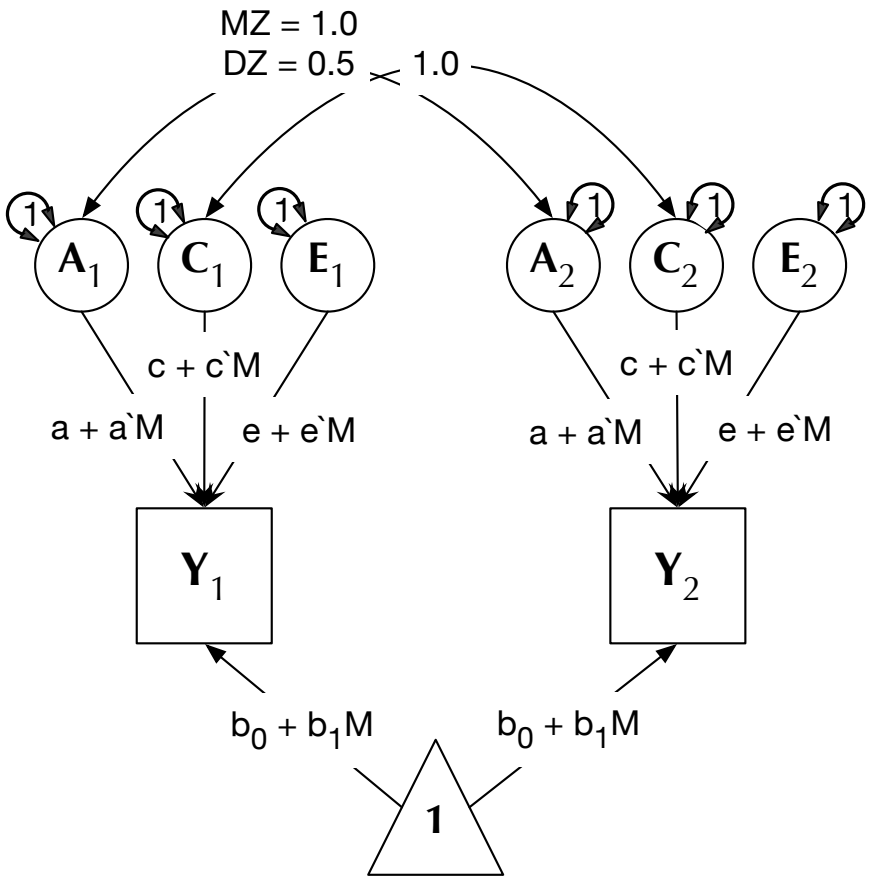


Figure 9: Univariate Gene x measured shared-environment twin model.

used in the model. `umxGxE` takes care of this by removing these rows (reporting explicitly to the user what it has done for MZ and DZ data separately).

```
data("twinData")

# create age variables for twin 1 and twin 2
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")
```

With setup out of the way, what remains is to call `umxGxE`, allowing the model to auto-run. The user can request a custom `umxSummary` if desired. In this case, the summary is reported as a plot, which may be either the raw or standardized output, in two side-by side plots, or in separate plots (see Figure 10).

```
# 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)
```

As with all **umx** functions, all parameters are consistently labeled, and `umxModify` can be used to, for instance, drop the moderated additive genetic path by label, and requesting a test of change in likelihood for significance:

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

Note likelihood ratio tests for dropping parameters are not always asymptotically distributed as chi-squared with df equal to the difference in the number of parameters. For single variance components, in the univariate case the p-values can be divided by 2, but this does not extend to the multivariate case (Dominicus, Skrondal, Gjessing, Pedersen, and Palmgren 2006; Visscher 2006; Wu and Neale 2013). Moderating parameters, being unbounded, do not typically suffer from this problem.

In order to facilitate theory-driven model reduction, umx implements the `umxReduce` function which, if it knows about the type of model input, can intelligently reduce the model, outputting

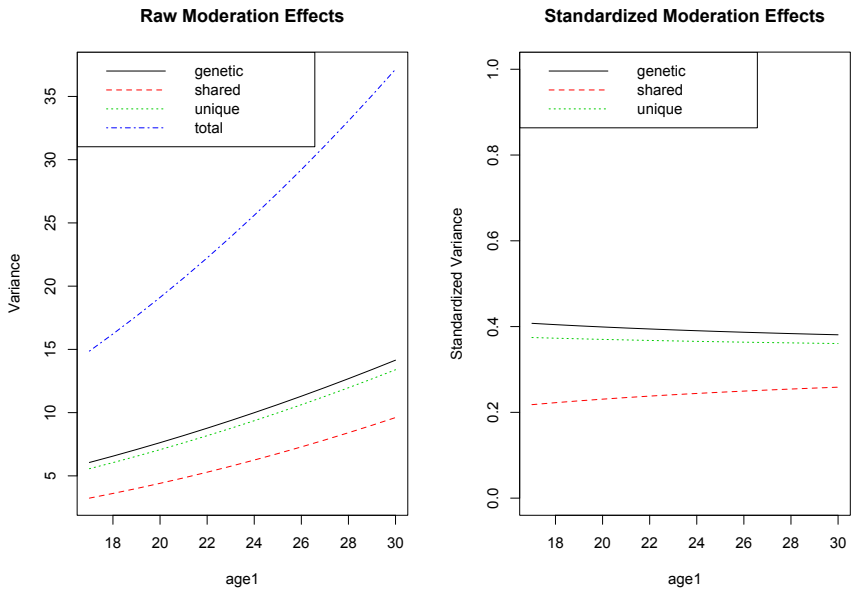


Figure 10: GxE analysis default plot output.

a table of model comparisons. In the case of `umxGxE` models, there is a standard set of comparisons, and these have been implemented in `umxReduce`. Functionality continues to improve for other model types. This is shown for the example G×E model in Table 14.

```
# Reduce the model and output a comparison table
umxReduce(GE1)
```

3.6. Window-based GxE

`umxGxE_window` (Briley, Harden, Bates, and Tucker-Drob 2015) also implements a gene-environment interaction model. It does this not by imposing a particular function (linear or otherwise) on the interaction, but by estimating the model sequentially on windows of the data. In this way, it generates a spline-style interaction function that can take arbitrary forms (See Figure 11). The function linking genetic influence and context is not necessarily linear, but may react more steeply at extremes of the moderator, take the form of known growth functions of age, or take other, unknown forms. To avoid obscuring the underlying shape of the interaction effect, local structural equation modeling (LOSEM) may be used, and `umxGxE_window` implements this model. LOSEM is 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 G×E (or C×E or E×E) interaction forms.

Example GxE windowed analysis

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with
G by E	9				1000958.352	
No lin mean	8	-44684.14	1	1.000	956272.216	G by E
No quad mean	8	-999320.62	1	1.000	1635.730	G by E
No means moderation	7	-999097.00	2	1.000	1857.350	G by E
DropA	8	149404.86	1	< 0.001	1150361.208	G by E
DropC	8	15192.63	1	< 0.001	1016148.987	G by E
No mod on A	8	692656.81	1	< 0.001	1693613.165	G by E
No mod on C	8	520231.54	1	< 0.001	1521187.896	G by E
No mod on E	8	286226.09	1	< 0.001	1287182.445	G by E
No moderation	*6*	*-999259.71*	*3*	*1.000*	*1692.639*	*G by E*
No A no mod on A	7	732979.50	2	< 0.001	1733933.851	G by E
No C no mod on C	7	520664.23	2	< 0.001	1521618.585	G by E
No c no ce mod	6	1325976.22	3	< 0.001	2326928.580	G by E
No c no moderation	5	-999259.71	4	1.000	1690.639	G by E

Table 14: Model reduction table, generated for the example G×E model using `umxReduce`.

Again we need to setup the data correctly for the analysis. `umxGxE_window` takes a `data.frame` consisting of a moderator and two DV columns: one for each twin. The model also assumes two groups: MZ and DZ. Moderator cannot be missing, so to be explicit, we delete cases with missing moderator prior to analysis. The first three lines open the built-in "twinData" dataset of Australian twins, and define the name of the moderator column in the dataset ("age" in this case), along with the DV ("bmi").

```
require(umx);
data("twinData")
mod     = "age"
selDVs  = c("bmi1", "bmi2")
```

We next pull out the younger cohort from the data, remove rows where the moderator is missing, and generate the MZ and DZ subsets of the data:

```
# 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))
```

Next, we run the analysis.

```
# toggle auto-plot off, so we don't plot every
# level of the moderator
umx_set_auto_plot(FALSE)
# Run the GxE analyses across all the windows
umxGxE_window(selDVs = selDVs, moderator = mod, mzData = mzData, dzData = dzData)
umx_set_auto_plot(TRUE)
```

The software reports to the user as it works through each level of the moderator encountered, and produces a graph at the end of this run, plotting the A, C, and E windowed estimates at each level of the moderator (see Figure 11). It is possible to run the function at only a single level or chosen range of moderator values, and of course the model results may be subjected to additional tests (Briley *et al.* 2015).

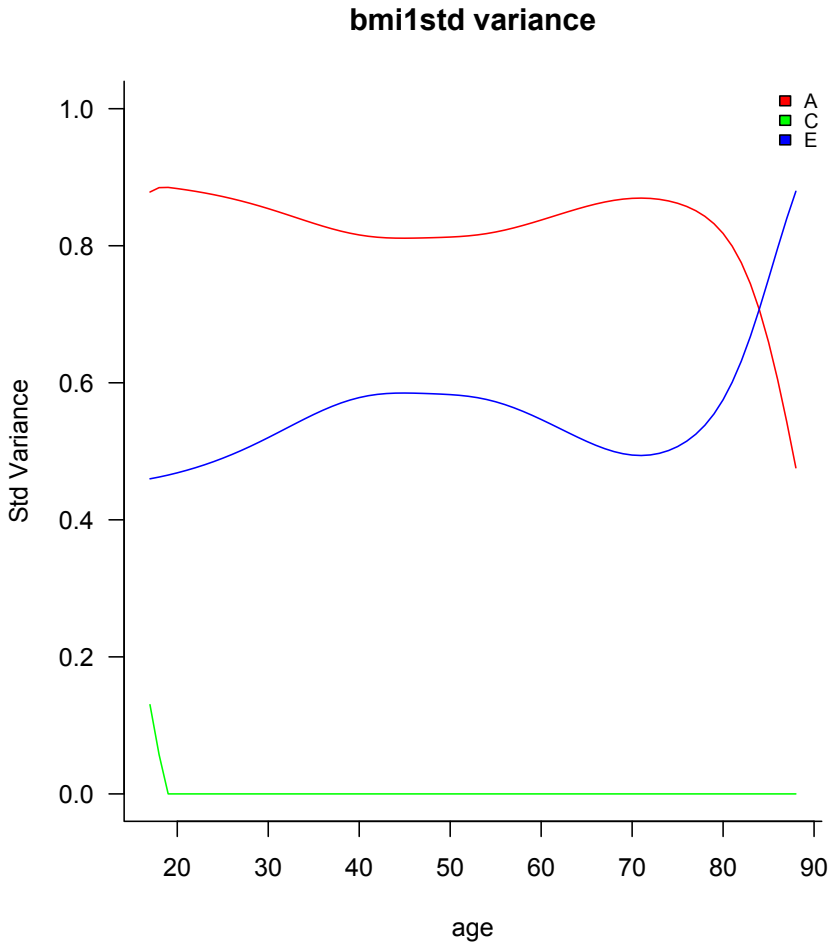


Figure 11: Output graphic from a windowed or "LOSEM" G×Age analysis.

4. Summary

umx offers a variety of functions for rapid path-based modeling, a growing set of twin models, and helpful plotting and reporting routines. It makes available a set of data-processing functions, especially suitable for twin or wide-format data. Helping to lower the learning curve, a tutorial blog site operates at `http://tbates.github.io`. In addition, a help forum for users of the package is provided at the **OpenMx** website `http://openmx.ssri.psu.edu/forums/`

third-party-software/umx.

It is hoped that the package is useful to those learning and undertaking behavior genetics, but also to the wider set of users seeking to utilize the power of structural modeling in their work and who seek approachable but powerful open-source solutions for this need.

References

Archontaki D, Lewis G, Bates T (2013). "Genetic Influences on Psychological Well-being: a Nationally Representative Twin Study." *Journal of Personality*, **81**(2), 221–30. ISSN 1467-6494 (Electronic) 0022-3506 (Linking). doi:10.1111/j.1467-6494.2012.00787.x. J Pers. 2012 Mar 20. doi: 10.1111/j.1467-6494.2012.00787.x., URL http://www.ncbi.nlm.nih.gov/pubmed/22432931.

Bates T, Lewis G, Weiss A (2013). "Childhood Socioeconomic Status Amplifies Genetic Effects on Adult Intelligence." *Psychological Science*, **24**(10), 2111–6. ISSN 1467-9280 (Electronic) 0956-7976 (Linking). doi:10.1177/0956797613488394. URL http://www.ncbi.nlm.nih.gov/pubmed/24002887.

Bates TC, Hansell NK, Martin NG, Wright MJ (2016). "When does socioeconomic status (SES) moderate the heritability of IQ? No evidence for g × SES interaction for IQ in a representative sample of 1176 Australian adolescent twin pairs." *Intelligence*, **56**, 10–15. doi:10.1016/j.intell.2016.02.003.

Boker S, Neale M, Maes H, Wilde M, Spiegel M, Brick T, Spies J, Estabrook R, Kenny S, Bates TC, Mehta P, Fox J (2011). "OpenMx: An Open Source Extended Structural Equation Modeling Framework." *Psychometrika*, **76**(2), 306–317. doi:10.1007/s11336-010-9200-6.

Briley DA, Harden KP, Bates TC, Tucker-Drob EM (2015). "Nonparametric Estimates of Gene x Environment Interaction Using Local Structural Equation Modeling." *Behavior Genetics*, **45**(5), 581–96. ISSN 1573-3297 (Electronic) 0001-8244 (Linking). doi:10.1007/s10519-015-9732-8. URL http://www.ncbi.nlm.nih.gov/pubmed/26318287.

Dominicus A, Skrondal A, Gjessing HK, Pedersen NL, Palmgren J (2006). "Likelihood Ratio Tests in Behavioral Genetics: Problems and Solutions." *Behav Genet*, **36**(2), 331–40. ISSN 0001-8244 (Print) 0001-8244 (Linking). doi:10.1007/s10519-005-9034-7. URL http://www.ncbi.nlm.nih.gov/pubmed/16474914.

Duncan OD, Haller AO, Portes A (1968). "Peer Influences on Aspirations: a Reinterpretation." *American Journal of Sociology*, pp. 119–137.

Fox J, Nie Z, Byrnes J (2014). "sem: Structural Equation Models." URL http://CRAN.R-project.org/package=sem.

Hershberger SL (2003). "The Growth of Structural Equation Modeling: 1994-2001." *Structural Equation Modeling: A Multidisciplinary Journal*, **10**(1), 35–46. doi:10.1207/s15328007sem1001_2.

IBM Corp (2013). *IBM SPSS Statistics for Macintosh, Version 22.0*. Armonk, NY. URL http://www.sas.com/.

Jöreskog KG (1969a). "A General Approach to Confirmatory Maximum Likelihood Factor Analysis." *Psychometrika*, **34**(2P1), 183–202. ISSN 0033-3123. doi:10.1007/Bf02289343.

Jöreskog KG (1969b). "A General Method for Analysis of Covariance Structures." *Biometrics*, **25**(4), 794–. ISSN 0006-341X.

Knopik VS, Neiderhiser JM, DeFries JC, Plomin R (2016). *Behavioral Genetics*. 7 edition. Worth Publishers, London.

Lewis G, Bates T (2010). "Genetic Evidence for Multiple Biological Mechanisms Underlying Ingroup Favoritism." *Psychological Science*, **21**(11), 1623–1628. doi:10.1177/0956797610387439. URL http://www.ncbi.nlm.nih.gov/pubmed/20974715.

Lewis G, Bates TC (2014). "How Genes Influence Personality: Evidence from Multi-facet Twin Analyses of the HEXACO Dimensions." *Journal of Research in Personality*, **51**, 9–17. ISSN 00926566. doi:10.1016/j.jrp.2014.04.004.

MacCallum RC (2003). "Working with imperfect models." *Multivariate Behavioral Research*, **38**, 113–139. doi:10.1207/S15327906MBR3801_5.

Maes HH, Sullivan PF, Bulik CM, Neale MC, Prescott CA, Eaves LJ, Kendler KS (2004). "A Twin Study of Genetic and Environmental Influences on Tobacco Initiation, Regular Tobacco use and Nicotine Dependence." *Psychol Med*, **34**(7), 1251–61. ISSN 0033-2917. URL http://www.ncbi.nlm.nih.gov/pubmed/15697051.

Martin N, Jardine R (1986). "Eysenck's Contributions to Behaviour Genetics." *Hans Eysenck: consensus and controversy*, pp. 13–47.

Martin NG, Eaves LJ, Heath AC, Jardine R, Feingold LM, Eysenck HJ (1986). "Transmission of Social Attitudes." *Proceedings of the National Academy of Sciences of the United States of America*, **83**(12), 4364–8. ISSN 0027-8424 (Print) 0027-8424 (Linking). URL http://www.ncbi.nlm.nih.gov/pubmed/3459179.

McArdle JJ, Boker SM (1990). *RAMpath*. Lawrence Erlbaum, Hillsdale, NJ.

Muthén L, Muthén B (1998-2016). *Mplus UserâĂŹs Guide*. 7 edition. Muthén Muthén, Los Angeles, CA.

Neale MC, Hunter MD, Pritikin JN, Zahery M, Brick TR, Kirkpatrick RM, Estabrook R, Bates TC, Maes HH, Boker SM (2016). "OpenMx 2.0: Extended Structural Equation and Statistical Modeling." *Psychometrika*, p. in press. ISSN 1860-0980 (Electronic) 0033-3123 (Linking). doi:10.1007/s11336-014-9435-8. URL http://www.ncbi.nlm.nih.gov/pubmed/25622929.

Neale MC, Maes HH (1996). *Methodology for Genetics Studies of Twins and Families*. 6th edition. Kluwer, Dordrecht, The Netherlands.

Neale MC, Martin NG (1989). "The Effects of age, sex, and Genotype on Self-report Drunkenness Following a Challenge Dose of Alcohol." *Behavior Genetics*, **19**(1), 63–78. ISSN 0001-8244. doi:10.1007/BF01065884. URL http://www.ncbi.nlm.nih.gov/pubmed/2712814.

Pearl J (2009). *Causality: Models, Reasoning and Inference*. 2 edition. Cambridge University Press, Oxford.

Purcell S (2002). "Variance Components Models for Gene-environment Interaction in Twin Analysis." *Twin Res*, **5**(6), 554–571. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12573187.

Ritchie S, Bates T (2013). "Enduring Links from Childhood Mathematics and Reading Achievement to Adult Socioeconomic Status." *Psychological Science*, **24**(7), 1301–8. ISSN 1467-9280 (Electronic) 0956-7976 (Linking). doi:10.1177/0956797612466268. URL http://www.ncbi.nlm.nih.gov/pubmed/23640065.

Rosseel Y (2012). "lavaan: An R Package for Structural Equation Modeling." *Journal of Statistical Software*, **48**(2), 1–36. URL http://www.jstatsoft.org/v48/i02.

SAS Institute Inc (2003). *SAS/STAT Software, Version 9.1*. Cary, NC. URL http://www.sas.com/.

semTools Contributors (2016). *semTools: Useful tools for structural equation modeling*. R package version 0.4-11, URL http://cran.r-project.org/package=semTools.

Stata Corp LP (2016). *Stata Statistical Software: Release 14*. College Station, TX. URL http://www.stata.com/.

Visscher PM (2006). "A Note on the Asymptotic Distribution of Likelihood Ratio Tests to Test Variance Components." *Twin Res Hum Genet*, **9**(4), 490–5. ISSN 1832-4274 (Print) 1832-4274 (Linking). doi:10.1375/183242706778024928. URL http://www.ncbi.nlm.nih.gov/pubmed/16899155.

von Oertzen T, Brandmaier AM, Tsang S (2015). "Structural Equation Modeling With Îl'nyx." *Structural Equation Modeling: A Multidisciplinary Journal*, **22**(1), 148–161. ISSN 1070-5511.

Wu H, Neale MC (2013). "On the Likelihood Ratio Tests in Bivariate ACDE Models." *Psychometrika*, **78**(3), 441–63. ISSN 1860-0980 (Electronic) 0033-3123 (Linking). doi:10.1007/s11336-012-9304-2. URL http://www.ncbi.nlm.nih.gov/pubmed/25106394.

Yong-Kyu Kim E (2009). *Handbook of Behavior Genetics*. 1 edition. Springer, New York.

Affiliation:

Timothy C. Bates
University of Edinburgh
7 George Square, EH8 9JZ
UK
E-mail: tim.bates@ed.ac.uk
URL: http://www.psy.ed.ac.uk/people/view.php?name=timothy-bates