Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

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

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

Abstract

Structural equation modeling (SEM) is an important research tool, especially in the social sciences. Two major uses of SEM include path-based model specification, where ease of use and graphical and table-based reporting are important for researcher productivity, and complex multi-group and multi-level models developed in fields such as behavior genetics where the model specification might often involve dozens or hundreds of lines of matrix-based code. Often these models are experienced as difficult to implement, and may present a steep learning curve before results can be obtained. This paper describes the **umx** library, which is available on CRAN. It is ideal for both beginners and for those working on complex models. **umx** supports rapid development, modification, and comparison of models, as well as both graphical and tabular reporting. High-level support is provided for several standard multi-group genetic models for twin data that involve additive genetic, common environment and specific environment variance components. These support raw and covariance data, including joint ordinal, and give solutions for ACE, ACE with covariates, Common Pathway, Independent Pathway, and Gene $\times$ Environment interaction models implemented as "one-line" functions. A tutorial site and question forum are also available.

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

1. Introduction

Structural equation modeling (Jöreskog 1969b) enables multi-group modeling with latent and measured variables, and allows researchers to realize the power of causal modeling (Pearl 2009). It is widely used: For instance Google Scholar records 762,000 articles and books published in the last 5-years that contain the term. It is not surprising, therefore, that, while closed-source commercial applications exist, (e.g. Mplus, SAS proc calis, SPSS Amos, GLAMM in STATA), there are now 3 open-source R packages for performing SEM: **sem** (Fox, Nie, and Byrnes 2014); **lavaan** (Rosseel 2012); and **OpenMx** (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).

Despite the clear utility of SEM, and the advent of modular software such as **OpenMx** required to build cumulative engineering solutions in this field (Boker *et al.* 2011; Neale *et al.* 2016), 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 present paper describes **umx** – an easy-to-use package written in response to the need for a comprehensive and coherent set of high-level and support functions for path-based SEM and matrix-based multi-group twin modeling. Practical examples of **umx** usage are given. Users wanting a transition from **OpenMx** to **umx** should start at the beginning. Users wanting to learn about twin-modeling in umx may wish to skip to the section "Twin modeling in **umx**". Users unfamiliar with SEM should consult appendix 1 which outlines (briefly) the history of SEM and the origins of this graphical-causal modelling method. Users already familiar with **OpenMx** and adding **umx** to their skill-set should read appendices 2 and 3. To support editing with **umx**, an OpenMx.tmbundle (Bates 2016) is available within TextMate 2 which supports code editing for both **OpenMx** and **umx**, and can be adapted for other text editors.

1.1. Easy-to-learn but powerful open-source modeling

OpenMx (Boker *et al.* 2011; Neale *et al.* 2016) is a modular open-source package enabling advanced SEM inside the R environment. **umx** is also open-source and is designed to make this power accessible, easy to learn, and easier to use. It is designed to appeal to people who might otherwise consider commercial packages such as Amos® or Mplus®.

OpenMx provides a sophisticated underpinning for extended structural equation modelling. It supports path-based specification in RAM (McArdle and Boker 1990) and LISREL (Jöreskog 1969a) syntax. There is also support for modeling via arbitrary matrices, constraints, and algebras. In terms of data, it accepts both summary and raw data input. **OpenMx** has full support for analysis of binary and ordinal variables, including raw data containing arbitrary mixtures of continuous, ordinal and binary data (joint-ordinal). Full-information maximum likelihood (FIML) analysis with missing data is supported, as are Weighted (WLS), un-weighted, and diagonal least-squares estimators. With raw data, models may include row-specific values (definition variables). Multiple-group models are implemented simply by embedding each group as an `mxModel` inside a container `mxModel`. Constraints and equalities are supported via a flexible system of label-based equating and algebra-based linear and non-linear constraint specification. Under the hood, **OpenMx** includes two powerful open-source optimization packages–CSOLNP and SLSQP–and can use the closed-source NPSOL optimizer.

umx builds on the **OpenMx** to help users to access the power of this platform. It is of particular benefit to novice users (especially via the `umxRAM` and `umxPath` modeling functions), and experts (via a library of 1-line solutions for twin modeling and helpers for low-level tasks such as switching optimizer and constructing complex models). It provides functions to produce attractive graphical and publication ready tables, and seeks to flatten the learning curve from beginner to expert levels. This ease-of-use is also useful for teaching, and the example code included in function help-files is oriented in this direction with ample commented examples. The functions also make coding less error-prone via a significant investment in error-checking and instructive feedback. In total, the package includes approximately 130 higher- and lower-level helpers for such tasks as data processing, fixing, freeing and reporting model parameters.

1.2. History

The **umx** package evolved over the last 5 years in response to modelling demands of research and teaching. It builds on scripts contributed over the last 25 years of teaching based on Mx and **OpenMx**, primarily at the Boulder Twin Workshops (Neale and Maes 1996). Experience in teaching SEM, and feedback while supporting **OpenMx** users in research lead to the library expanding to cover a multitude of common tasks, including path-based modeling, with an emphasis on speed (to allow its use during time-constrained lessons, but also to afford researchers the ability to rapidly prototype and capture new model ideas), graphical feedback of model construction, and conformity of reporting to the needs of peer-reviewed publication. During this process function names and parameters were intensively refined based on feedback and a drive for consistency and memorability. The functions presented here have been tested, and several papers using the library have been published (Archontaki, Lewis, and Bates 2013; Ritchie and Bates 2013). **umx** is under active development – with updates on CRAN every month or two, and major extensions both past (e.g., window-based G × E modeling (Briley, Harden, Bates, and Tucker-Drob 2015), current (sex-limitation (Maes, Sullivan, Bulik, Neale, Prescott, Eaves, and Kendler 2004) models) and planned (5-group extensions of the twin modeling functions, modeling of covariates).

In the sections below, we first discuss functions supporting path-based models such as that shown in Figure 1. The second section covers twin modeling in **umx**. We do not provide a comprehensive reference for all **umx** functions – for this readers are referred to the built-in package documentation. Instead we describe modeling using **umx**, with a small set of complete examples in the style that a working researcher would generate.

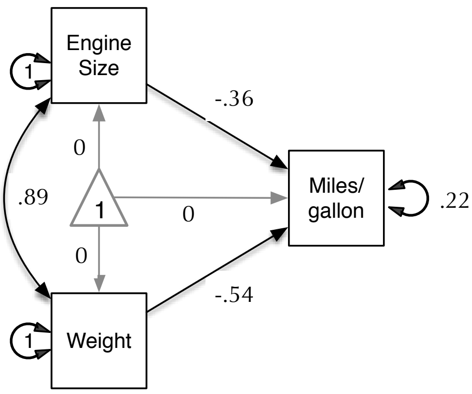


Figure 1: Example Path diagram linking miles/gallon (mpg) to vehicle weight (wt) and engine size or displacement (disp) with standardized path estimates.

2. Installing the package

Installing **umx** can be done using the R-code:

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

This document assumes **umx** version 1.2.5 or higher. The current package version can be shown with:

`packageVersion("umx")`

Users wanting to speed processing with multi-core parallel execution or try the "NPSOL" optimizer should execute:

`umx_get_OpenMx()`.

This will install a version of the package from our development website with OpenMP enabled and with NPSOL.

3. Core path-based functions

3.1. An example RAM model

Coding models in **umx** is best learned by doing. Here we implement the model shown in Figure 1) with the following **umx** script:

```
library("umx")
m1 <- umxRAM("big", data = mxData(mtcars, type = "raw"), showEstimates= "std",
      umxPath(c("disp", "wt"), to = "mpg"),
      umxPath("disp", with = "wt"),
      umxPath(v.m. = c("disp", "wt", "mpg"))
)
```

umxRAM tells the user what (if any) latent variables were created, and what manifests were used from the data. By default it then runs the model and shows a summary.

$\chi^2(87) = 0, p < 0.001; CFI = 1; TLI = 1; RMSEA = 0$

A parameter table (shown below) can be requested, or generated a-fresh with `umxSummary`. We leverage the **knitr::kable** function for table making which allows output in markdown, latex, or html styles. Output defaults to markdown and is easily set with the `umx_set_table.format` function.

```
|name           | Std.Estimate| Std.SE|CI                   |
|:--------------|------------:|------:|:--------------------|
|disp_to_mpg    |        -0.36|   0.18|-0.36 [-0.71, -0.02] |
|wt_to_mpg      |        -0.54|   0.17|-0.54 [-0.89, -0.2]  |
|mpg_with_mpg   |         0.22|   0.07|0.22 [0.08, 0.35]    |
|disp_with_disp |         1.00|   0.00|1 [1, 1]             |
|disp_with_wt   |         0.89|   0.04|0.89 [0.81, 0.96]    |
|wt_with_wt     |         1.00|   0.00|1 [1, 1]             |
```

Markdown or latex can be rendered into a formatted tables. For instance a lightly-edited version of the above table is shown in Table 1.

3.2. Building a model

In keeping with familiar R functions such as `"lm"`, `umxRAM` take a `data` parameter, providing the data to be modeled. There is no need to specify manifest and latent variables as these

Variable Name	Std Estimate	Std SE	CI
Disp to mpg	-0.36	0.18	-0.36 [-0.71, -0.02]
Wt to mpg	-0.54	0.17	-0.54 [-0.89, -0.2]
Mpg with mpg	0.22	0.07	0.22 [0.08, 0.35]
Disp with disp	1.00	0.00	1.00 [1, 1]
Disp with wt	0.89	0.04	0.89 [0.81, 0.96]
Wt with wt	1.00	0.00	1.00 [1, 1]

Table 1: Example `umxSummary` output table, formatted using a markdown processor.

are mapped from the data, and any variable not found is assumed to be a latent variable. The user is given explicit feedback about which manifests have been detected, which latent variables were created, and which data columns were dropped from the data.frame, and how the data have been interpreted (see above for example of this feedback).

`umxPath` creates paths in a model. This function provides adjectives to describe a range of path types. For example, to create a 2-headed path fixed at 1, the user would add a call to `umxPath("a", with = "b", fixedAt = 1)`. Means are added using the `means` parameter. For instance `umxPath(means = "b")` will add a means expectation for the variable "b". `umxPath` supports common tasks like specifying a variableâĂŹs mean and variance, with `v1m0` and `v.m.` parameters to specify a normalized (mean = 0, variance = 1) or freely estimated path respectively. The character `.` was chosen by analogy with the wild-card character which matches any value in regular expressions. Examples of `umxPath` include:

```
umxPath(means = 'b') # Add means expectation for variable b
umxPath(var = c('a', 'b')) # 2-head (variance) path for a and for b. Value free.
umxPath("a", with = 'b', fixedAt = 1) # 2-headed path from a to b. Value fixed at 1
umxPath(v1m0 = 'g') # create fixed variance of 1, mean = 0 for variable g.
umxPath(v.m. = c('disp', 'wt', 'mpg')) # three variances and means, freely estimated.
umxPath(unique.bivariate = c('A','B','C'))} # Create paths A<->B, B<->C, A<->C.
```

3.3. Summary and graphical Output

To request a user-configured summary, the user can call `umxSummary`. This can be as little as a line of fit information, or a table of standardized estimates (requested using `umxSummary(m1, show = "std")`). The summary table can be a markdown, latex, or html echoed to the console, or an html table opened in your web browser from where it can be copied and pasted into a word processor:

`umxSummary(m1, show = "std", report = "html")`

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

`umx` includes S3 `plot` methods for RAM and twin models. These rely on the dot language, invented at Bell Laboratories (as was RâĂŹs predecessor S) to specify graphs. Dot describes graphs in a text-based format that can be automatically laid out by graphviz applications, much as latex handles layout. The .gv file created can now be displayed in **R** using the **DiagrammR** package. Further processing of the diagram is possible outside R using free cross-platform apps (graphviz) and also closed software such as Omnigraffle®, and Visio®.

A basic usage of the command is shown below, and was used to generate Figure 1 above (rendered in Omnigraffle®).

```
plot(m1, showMeans = FALSE, showFixed = TRUE)
```

The `plot` function has a number of options to customizing output. By default, files are automatically named of files based on the model name. This can be overriden by setting the `dotFilename` parameter to the desired filename. Similarly residuals are drawn as circles by default. To cope with situations where diagrams imported into other applications fail to render the residual cirlces, these can be set to `resid="line"`. Similarly, whether means and fixed paths are shown can be customized with the `showMeans` and `showFixed` parameters respectively.

3.4. Inspecting model parameters, coefficients, and residuals.

It is convenient to access the values, labels, and freefixed state parameters in models. the **OpenMx** function can be used to view coefficients via the S3 method `coef`. `umx` also supports the `parameters` function for accessing labels, and adds support for regular expression filtering. The `umx_show` function can also be useful for inspecting models. By default this shows the values for one and two headed paths in a compact matrix layout, but allows accessing free or fixed variables only, and for specified matrices within a model. For example, to access the free state of asymmetric paths, the following snippet will give the output shown in Table 2.

```
\code{umx_show(m1, what = 'free', matrices = 'A')}
```

	mpg	disp	wt
mpg	.	TRUE	TRUE
disp	.	.	.
wt	.	.	.

Table 2: Showing free for:A matrix in m1 - model of miles per gallon.

Another common need in modeling is to inspect residuals. **umx** implements an S3 for `residuals` which for model 2 built above yields output shown in Table 3. The user can zoom in on bad values with, e.g. `suppress = .01`, which will hide residuals smaller than the value of `suppress`. The ability to filter results is also implemented elsewhere in **umx** functionality. For instance `umxSummary`.

	mpg	displacement	weight
mpg	.	-0.08	.
disp	-0.08	.	.
wt	.	.	.

Table 3: Residuals

3.5. Modify and re-run models

A common task in modelling is to re-run models with some paths fixed to values such as zero, or others free to vary. The `umxModify` function supports this task with a number of features for model modification. Along with re-running a model with updated paths (by default an updated path is fixed at zero, but of course any value is possible via setting the `values` parameter), the update parameter can take regular expressions, so it can easily find ranges of character strings that match a pattern. This function also supports the convenience of automatically printing a comparison table of model fit comparing the old and new models.

The following code-snippet will create a new model based on model `m1`, with the path "disp_to_mpg" dropped, the model renamed to reflect this, re-running the model, returning it as `m2`, and displaying a comparison table:

```
m2 = umxModify(m1, update = "disp_to_mpg", name = "no_disp", comp = TRUE)
```

As an instance of using regular-expressions in update a model, all paths from "disp" could be dropped at once using `update = "d̂isp_to.*"`, and setting `regex = TRUE` to let **umx** know to interpret the update command as a regular expression. Alternatively, if `regex` is a regular expression, this over-rides the input to `update`, and treats the contents of regex as the regular expression.

```
umxModify(m1, update = "^disp_to.*", regex = TRUE)
```

3.6. Compare models

Model comparison is done using `umxCompare`. This function is very similar to **OpenMx**âĂŹs `mxCompare()` function, with a focus on publication-ready table-layout. To this end, it returns columns most commonly used in papers, includes a column directing the reader to the base model for the comparison, and formats values in APA style with control over precision via the standard `digits` parameter. It can report the results to the console (by default in markdown style), or else open a browser table for pasting into a word processor. By default it also reports the output as a series of sentences suitable (with editing) for inclusion in a paper. An examples of using this function is `umxCompare(m1, m2)`, which yields the following fit table and approximation to a plain-English description (the new name is used to describe what was done, which might need some editing):

The hypothesis that drop_x2 was tested by dropping drop_x2 from One Factor. This caused a significant loss of fit ($\chi^2(1) = 944.11, p < 0.001$).

Model	EP	Δ -2LL	Δ df	p	AIC	Compare with Model
One Factor	10				-2.615998	
drop_x2	9	944.1	1	< 0.001	939.49	One Factor

Table 4: Example umxSummary output table, formatted using a markdown processor.

To open the output as an html table in a browser, say:

`umxCompare(m1, m2, report = "html") # Open table in browser`

Multiple comparisons are supported via the ability of the function to take more than one base and/or more than one comparison model, e.g.: `umxCompare(c(m1, m2), c(m2, m3), all = TRUE)`

3.7. Equating model parameters

In addition to dropping or adding parameters, a second common task in modelling is to equate parameters. umx provides a convenience function to equate parameters by setting one or more parameters (the "slave" set) equal to one or more "master" parameters. These parameters are picked out via their labels, and setting two or more parameters to have the same value is accomplished by setting the slave(s) to have the same label(s) as the master parameters, thus constraining them to take the same value during model fitting. For example, if m1 is set to our standard example of a 1-factor RAM model, we can test equating the paths from G to x1 and to x2 as follows:

```
m2 = umxEquate(m1, master = "G_to_x1", slave = "G_to_x2", name = "Eq")
m2 = mxRun(m2) # have to run the model again...
umxCompare(m1, m2) # not good :-)
```

3.8. Using labels to set parameter values

As seen above modeling, reducing or updating model, specifying two parameters to be equated involves labels. The user can access model parameters by label, and use labels to set the values of parameters. Much of this relies on the function umxLabel, which automatically labels all paths in the built-in umxRAM and twin models. Partnering with umxLabel, the function umxLabel intellligently sets start values for parameters. These are discussed next, along with the labeling scheme and an applicaiton of equating paths using labels.

By design **OpenMx** does not add labels to paths (matrix cells in the $labels matrix of an mxMatrix), nor does it presume to set start values for free paths (cells in the $values matrix). Both labels and start values are of course important, however. Indeed, crucial in the case of values, enabling models to be fit. In the case of labels, these unlock much of the power of **OpenMx** for label-based modifications, and to labeling paths for printing and graphic display. **umx** automates these labeling and start value tasks for several common situations using the functions umxLabel and umxValues. Both functions are polymorphic (respond contextually to the type of input they are called with), allowing each function to handle RAM models, matrix models, paths, and matrices as input. umxLabel will create appropriate unique labels for each element of the object offered up as input.

In the case of a RAM model or mxPath, it labels cells with the "from" and "to" variables, separated by underscores and either "to" or "with" based on the number of arrow heads on the path. Thus umxLabel(mxPath("IQâĂ▝, "earnings")) would create the label "IQ_to_earnings". This is consistent with Onyx (von Oertzen, Brandmaier, and Tsang 2015). Future versions of (umx) may extend the labeling scheme to encode more information in each label. Specifically it is proposed to set labels created using the "var" and "resid" parameters in umxPath to take the form "var_VarName" and "resid_VarName" to encode this information in the model.

By contrast, for matrices, which can contain arbitrary contents, labels are based on the matrix name, an underscore, the letter "r" for row, the row number, followed by the letter "c" and the cell column number. The statement following code will return the $labels slot of a matrix "means":

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

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

The `umxValues` function will attempt to set feasible start-values for the free parameters in its input. Currently, this can be a RAM model, a matrix-based model, or a single mxMatrix. the function tries to be intelligent in guessing good start values based on the input type, and on values in any data in the model. Thus for RAM models, manifest variable means are set to the observed means. Manifest variances (the diagonal of the S (symmetric or two-head path) matrix, filtered by the F matrix) are set to 80% of the observed variance of each variable. Free off-diagonal values in the S matrix are set to 0 by default (so estimation begins from an independence model). Single-head paths (free values in the A matrix) are set to a modest positive value (.9). These defaults may change in subsequent versions as improved solutions are found.

3.9. An example: Using labels to drop paths

In the following example, we modify the common pathway model constructed earlier. We may wish to update this model by dropping the specific C paths, and also the common-pathway paths for the C common factor, to examine the effects of shared or family-level environment. The `umxCP` labeling scheme follows a systematic pattern. The common factor paths are stored in matrices `a_cp`, `c_cp`, and `e_cp`. Specific a, c, and e effects are stored in matrices `as`, `cs`, and `es`.

We can see which parameters beginning with `c` are free in the model with a call to `umxGetParameters`.

```
parameters(m1, "^c", free = TRUE)}
```

This reveals the following 5 matching labels:

"c_cp_r1c1"

"cs_r1c1"

"cs_r2c2"

"cp_loadings_r1c1"

"cp_loadings_r2c1"

If the user runs this command, they will see that the labels take the form: matrix name, `_r` a row number, followed by `c` and a column number. While we could list all the labels in the *cs* (c specific-factor) and *c_cp* (c common pathway factors) matrices, **umx** supports regular-expressions which means we can select a subset of labels matching a pattern. To capture all the paths in the cs matrix, one would use the pattern `cs_.*`. This will match any label containing `cs_` followed by any characters. It would therefore match labels such as `cs_r1c1` which are our target. Similarly to target labels in the `c_cp` matrix, we can use the pattern `^cp_`. The caret (ˆ) symbol means "anchor this to the beginning of the string". We can join these patterns allowing either to match with a pipe | symbol. Wrapping each search string in round brackets forms them into groups, and separating with a pipe allows either group to match. This is a very sophisticated example, and users unfamiliar with regular expressions might wish to spell out each matched path completely and leave regular expressions for later learning.

```
pat = "(^cs_)|(^c_cp_)"
m2 = umxModify(m1, update = pat, regex = TRUE, name = "dropC")
umxSummary(m2, comparison = m1)
```

Equivalently, and more straightforwardly for users not expert in regular expressions:

```
paths = c("c_cp_r1c1", "cs_r1c1", "cs_r2c2")
m2 = umxModify(m1, update = paths, name = "dropC")
```

3.10. Modification Indices

In addition to locating the source of residual variance, users often wish to ask what modifications would improve model fit. Modification indices are supported in **OpenMx** (via `mxMI`). `umxMI` is a wrapper around mxMI allowing this function to behave intelligently for RAM-type models, to operate silently, and to return a table of either the significant modifications, or a list of user-chosen modifications, sorted by effect size.

Note: As the help for `mxMI`, admonishes, such modifications must be reported as post-hoc exploration. It is also wise to remember that this list of modifications simply reports the model fit changes in response to single changed paths: it is as un-creative and as pregnant with risk as p-hacking.

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

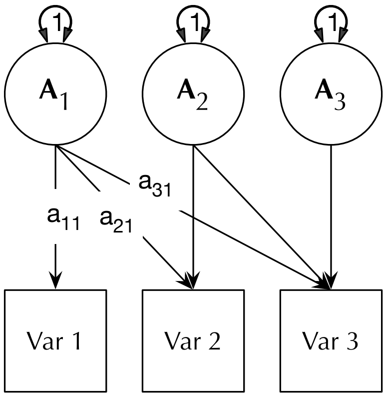
4. Behavior Genetic Twin Modeling

A major goal of **umx** was to provide support for common twin-models, including ACE, common-pathway, independent-pathway, and GxE, including tabular and graphical output, model comparison, and sophisticated control over model parameterization. These functions are outlined below.

Twin and family modeling takes advantage of different classes of genetic and environmental covariance present in nature. For example, identical (monozygotic) twins who share 100% of their genes, and fraternal (dizygotic) twins who share on average half their genes, siblings, who share 50% of their genes, but differ in age, adoptees who do not share genetic material with their rearing family, but who share the 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 (Neale and Maes 1996).

4.1. The ACE Cholesky model

`umxACE` supports a core model in behavior genetics, known as the ACE Cholesky model



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 2: The tri-variate Cholesky ACE model genetic components (C and E not shown) in graphical (left) and matrix forms.

(Neale and Maes 1996). This 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). This latter restriction emerges due to confounding of C and D when data are available from only MZ and DZ twin pairs. 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 (A, C, D or E) 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 2).

The `umxACE` function flexibly accepts both raw data and summary covariance data (in which case the user must also supply numbers of observations for the summary statistics). 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.

`umxACE` also supports weighting of individual data rows. In this case, the model is estimated for each row individually, then each rowâĂŹs likelihood 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 negative log-likelihood). This feature is currently used in non-linear GxE model functions. 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.

When it comes to interpretation and graphing, models built by `umxACE` are able to 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 built-in plot() method is extended by **umx** to handle graphical reporting of ACE models, laying out models as seen in Figure 2.

4.2. ACE Examples

We first set up data for a summary-data ACE analysis of weight data (using a built-in example dataset from Nick MartinâĂŹs Australian twin sample (Martin, Eaves, Heath, Jardine, Feingold, and Eysenck 1986; Martin and Jardine 1986).

(nb: this code requires version 2.5.0 or better of **OpenMx**).

```
require(umx); data(twinData)
selDVs = c("wt1", "wt2")
tmpTwin = twinData[twinData$cohort == "younger"]
dz = tmpTwin[tmpTwin$zyg == "DZFF", selDVs]
mz = tmpTwin[tmpTwin$zyg == "MZFF", selDVs]
```

The next example shows how `umxACE` allows the user to easily build an ACE model with a single function call. **umx** will give some feedback, noting that the variables are continuous and that the data have been treated as raw. We could conduct this same modeling using only covariance data, offering up suitable covariance matrices to `mzData` and `dzData`, and entering the number of subjects in each via `numObsDZ` and `numObsMZ`. (see Figure 3 for `plot` output). Small differences may be seen in the solutions if the covariance matricesâĂŹ numerical precision is limited to a few decimal places, such as could happen by entering the matrices by hand.

```
m1 = umxACE(selDVs = selDVs, dzData = dz, mzData = mz)
plot(m1)
```

The output can also be summarized in table form as shown below. By default the report table is written to the console in markdown. By setting `report` = "html", the user can request results in html and opened in the default browser. Whether the parameter table is standardized or not is set using the `showStd = TRUE` parameter (the default). The user can request the genetic correlations with `showRg = TRUE` (the default is FALSE). If Confidence intervals have been computed, these can be displayed with `CIs = TRUE`.

```
umxSummary(m1) # Create a tabular summary of the model
```

Which yields the following fit string and table of standardized loadings:

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

	a1	c1	e1
weight	-0.92	.	-0.39

Table 5: Standardized path loadings for ACE model.

The user can control output precision using the `digits` parameter. The `umxSummary` function can also call the plot in line by adding `dotFilename = name`. More advanced features include that the function can also return the standardized model (`returnStd = TRUE`). A model fit comparison can also be requested by offering up the comparison model in comparison = model. Help (`?umxACE`) gives extensive examples, including for binary, ordinal, and joint-ordinal cases.

It is often desirable to include covariates within twin models. We currently support ACE models with covariates via the `umxACEcov` function. This handles covariates by including them in the model as additional variables, allowed to covary with the main variables of

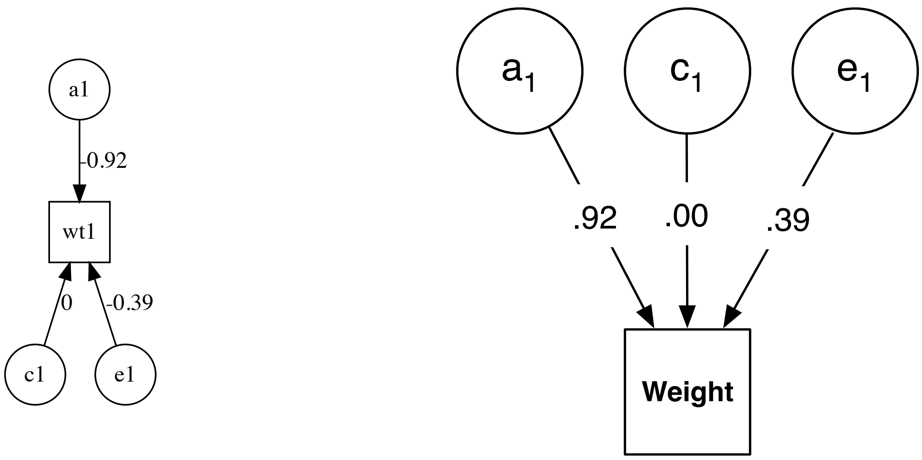


Figure 3: Output from plot(m1) of univariate ACE model of Australian weight data, rendered as default in **DiagrammR** (left) and after editing in Omnigraffle (right graphic).

interest (Neale and Martin 1989). In the next major version of **umx** this functionality will be enhanced to allow modelling covariances as definition variables on the means model (as is done currently in `umxGxE`), and these options will be included in each of the twin model functions.

4.3. Common Pathways model

The common-pathway (CP)model provides a powerful tool for theory-based decomposition of genetic and environmental differences (Neale and Maes 1996)). This is a valuable theoretical tool, allowing 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.

Like the ACE 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, 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 4).

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 can be seen Figure 4, each phenotype (by default) has A, C, and E influences specific to that phenotype. As with `umxACE`, `umxCP` can transparently handle mixtures of ordinal (binary or multi-level ordered factor data) inputs. Similar parameters are available for controlling parameters such as the DZ genetic correlation, and `plot` and `umxSummary` implement comprehensive model reporting and graphical output.

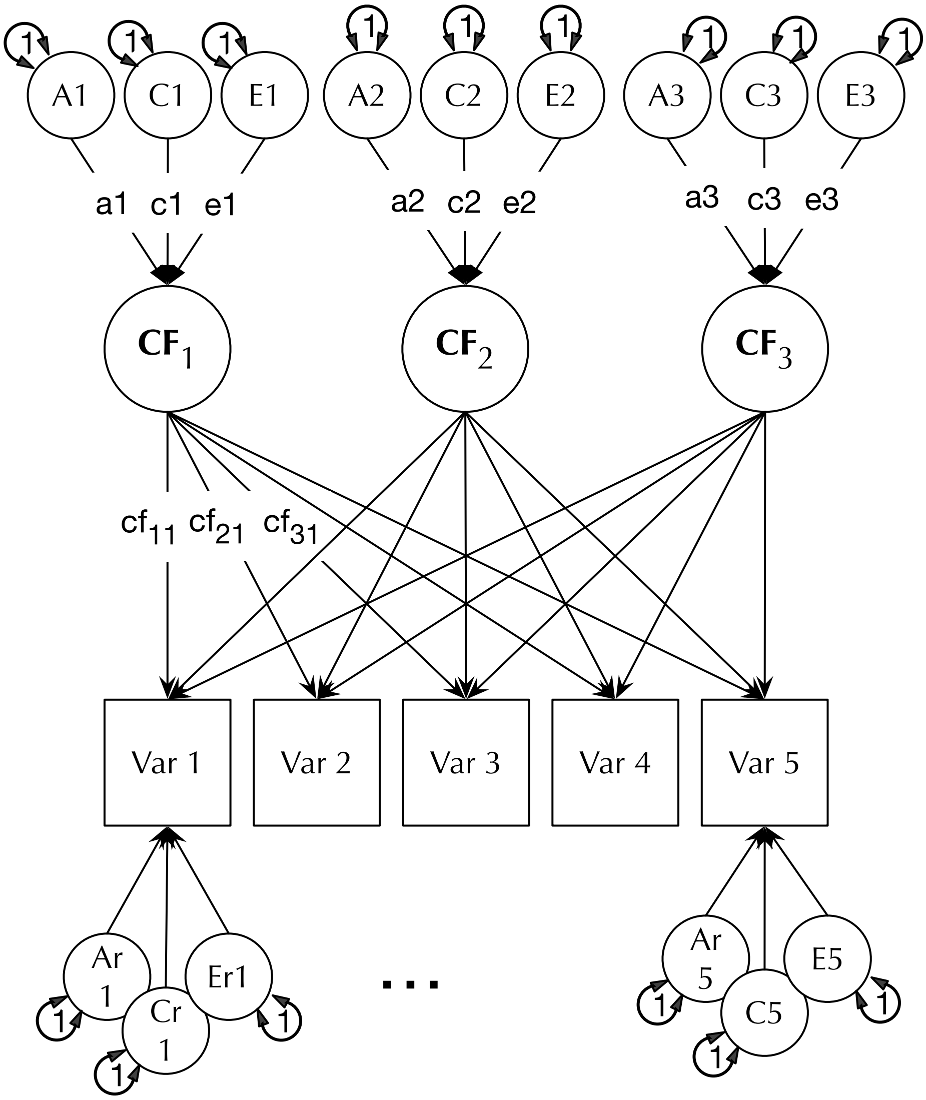


Figure 4: Common Pathway twin model with three common factors (CF1, CF2, and CF3), for five measured variables (phenotypes) var 1 thru 5. The specific ACE residual structure is shown at the base of the figure (drawn for only first and last phenotypes)

4.4. Example CP model

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

```
require(umx)
data(twinData)
selDVs = c("ht", "wt")
varNames = umx_paste_names(selDVs, "", 1:2)
zygList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = zygList)
mzData = subset(twinData, ZYG == "MZFF", varNames)
dzData = subset(twinData, ZYG == "DZFF", varNames)
```

The next section shows how `umxCP` allows the user to build the CP model in one line, followed by calls to `umxSummary`, and `plot` to show the fit, parameter estimates (shown in Tables 6 thru 9), and render and display in **DiagrammR**, graphviz or as pdf (Figure 5).

```
# Build and run a Common pathway model
m1 = umxCP(selDVs = selDVs, dzData = dzData, mzData = mzData, suffix = "")
m1 = umxRun(m1)
umxSummary(m1)
```

	A	C	E
Common factor 1	-0.98	.	0.21

Table 6: CP model common-factor path loadings.

	CP1
Height	0.85
Weight	0.55

Table 7: CP model path loadings on the Common Factor(s) for each trait.

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

Table 8: CP model standardized specific-factor loadings.

4.5. Independent Pathway model

The basic independent pathway (IP) model is nested within the 3-factor CP model (it is essentially a CP model with each of A, C, and E acting on only one factor. The IP models are created using `umxIP`. In this model (as can be seen Figure 6), 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 alone. Data input and additional control parameters for `umxIP` closely reflect those available for `umxCP` and `umxACE`, making

	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 9: CP model genetic and environmental correlations

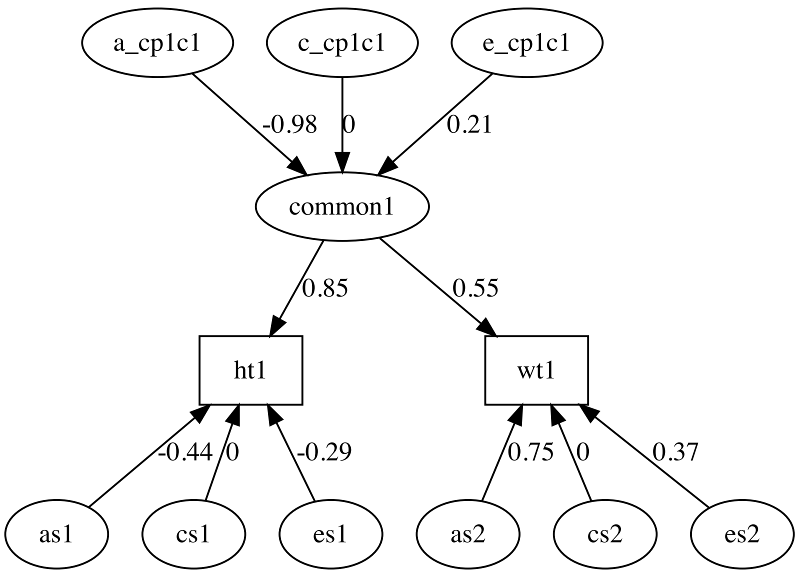


Figure 5: Common-pathway model for height and weight, as rendered automatically in GraphViz.

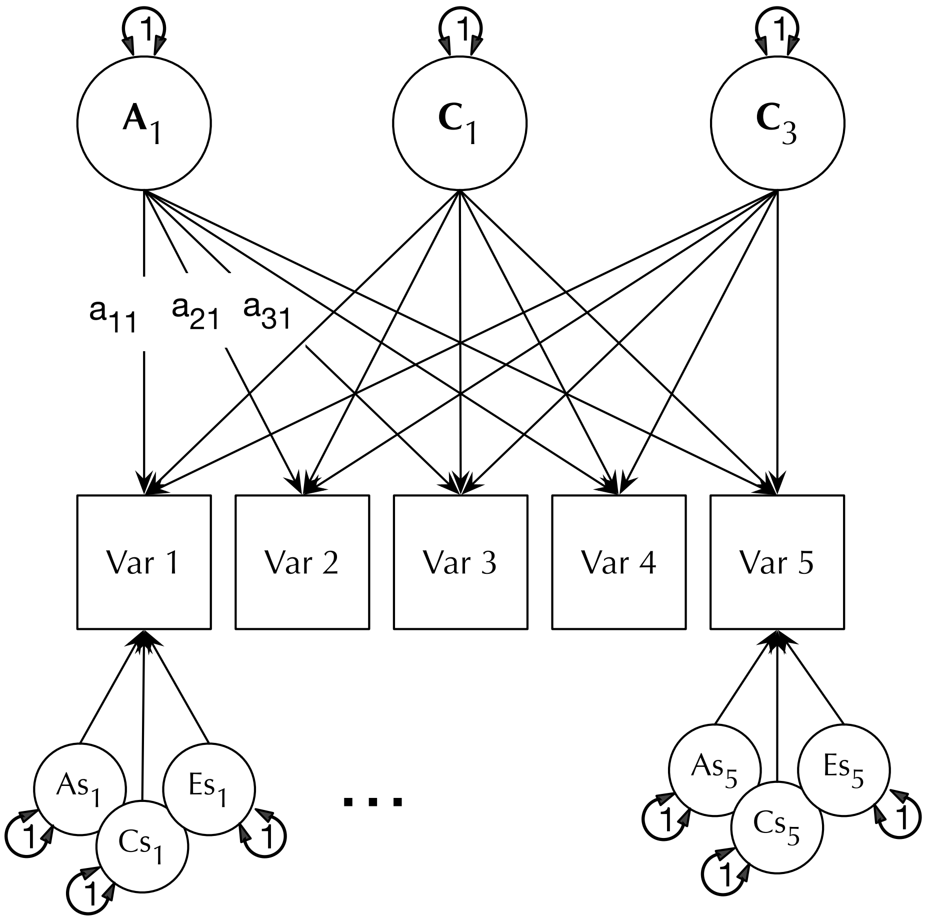


Figure 6: Independent Pathways model with a single independent general factor for each of A, C/D and E loading on all phenotypes (Var 1 thru Var 5), and showing the residual specific ACE structure modelled for each phenotype (drawn for variables 1 and 5 only for clarity).

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 lower the learning curve and increasing productivity. Users can of course implement ACE, CP, and IP models, then submit these as nested comparisons using `umxCompare`.

4.6. Gene x Environment interaction models

`umxGxE` implements the (Purcell 2002) gene-environment interaction single-phenotype model (See Figure 7). A common use for this type of model is examining changes in heritability (or 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).

As with all **umx** functions, examples of this type of analysis are included in the help documents linked to each function. As is often the case once using **umx**, most of a working script is taken up with data setup, including exclusion of rows with missing moderator data (`umxGxE` will do this for the user if desired). The user can make quick wrappers around both the function call and the setup to speed this for their particular column naming and other custom setup.

```
require(umx)
data(twinData)
zygList = c("MZFF", "MZMM", "DZFF", "DZMM", "DZOS")
twinData$ZYG = factor(twinData$zyg, levels = 1:5, labels = zygList)
twinData$age1 = twinData$age2 = twinData$age
selDVs  = c("bmi1", "bmi2")
selDefs = c("age1", "age2")
selVars = c(selDVs, selDefs)
mzData  = subset(twinData, ZYG == "MZFF", selVars)
dzData  = subset(twinData, ZYG == "DZFF", selVars)
mzData <- mzData[!is.na(mzData[selDefs[1]]) & !is.na(mzData[selDefs[2]]),]
dzData <- dzData[!is.na(dzData[selDefs[1]]) & !is.na(dzData[selDefs[2]]),]
```

With setup out of the way, the analysis is a simple call to `umxGxE`, allowing the model to auto-run, and requesting 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 8).

```
m1 = umxGxE(selDVs = selDVs, selDefs = selDefs, dzData = dzData, mzData = mzData)
umxSummaryGxE(m1)
umxSummary(m1, location = "topright")
umxSummary(m1, separateGraphs = FALSE)
```

As with all **umx** functions, all parameters are consistently labelled, 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:

```
m2 = umxModify(m1, "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

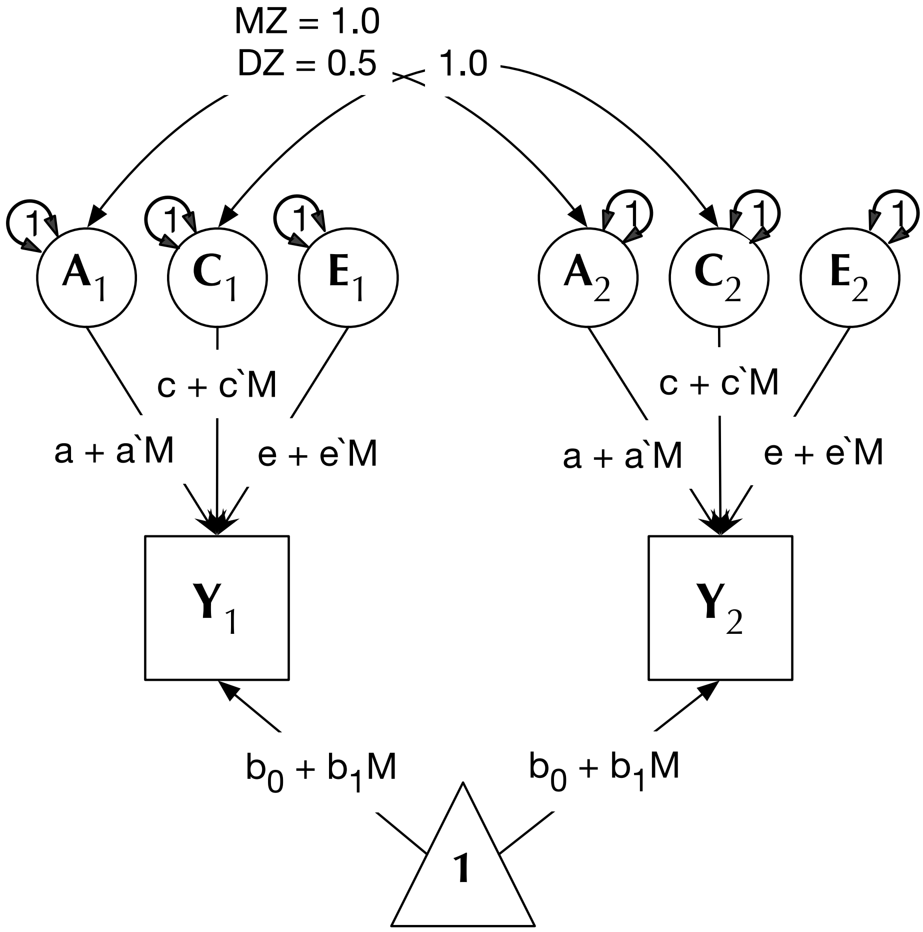


Figure 7: Univariate Gene x Measured shared-environment Twin Model

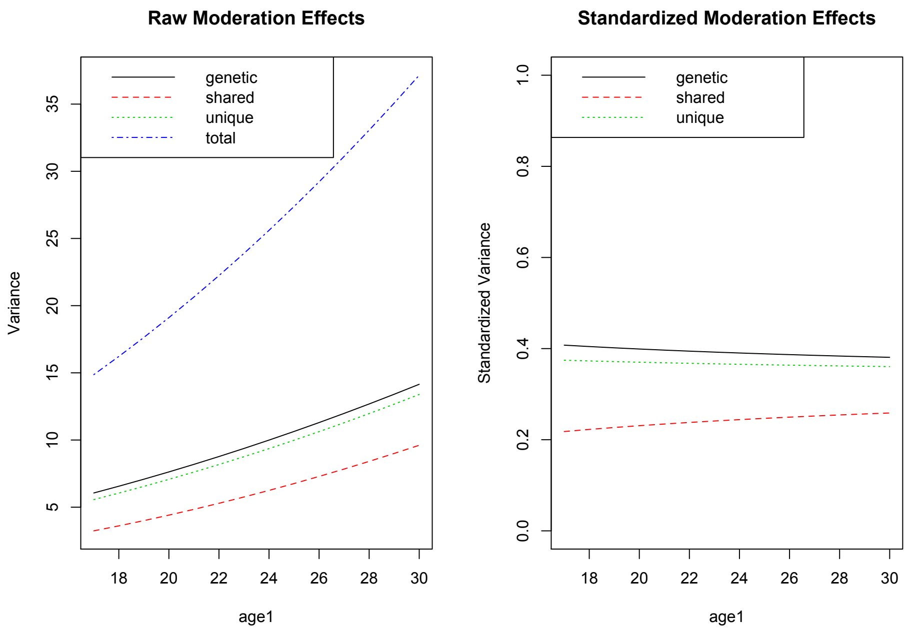


Figure 8: GxE analysis default plot output.

2006; Wu and Neale 2013). Moderating parameters, being unbounded, do not typically suffer from this problem.

4.7. Window-based GxE

`umxGxE_window` (Briley *et al.* 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 9). 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$\times$E (or C or E$\times$E) interaction forms.

4.8. Example GxE windowed analysis

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.

```
require(umx);
data(twinData) # Dataset of Australian twins, built into OpenMx
mod = "age" # The name of the moderator column in the dataset
selDVs = c("bmi1", "bmi2") # The DV for twin 1 and twin 2
tmpTwin = twinData[twinData$cohort == "younger"]
tmpTwin = tmpTwin[!is.na(tmpTwin[mod]),]
mzData = subset(tmpTwin, ZYG == "MZFF", c(selDVs, mod))
dzData = subset(tmpTwin, ZYG == "DZFF", c(selDVs, mod))
```

Next, we run the analysis. By default, FIML will be used, but analyses on covariance data are supported also.

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

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 9). 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).

5. 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.psyc.virginia.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.

6. Appendix 1: A (very) Brief History of Path analysis and SEM

Early in the last century, Wright (1918) developed path analysis as a tool for modeling sources of variance–initially in a dataset of rabbit skeletal dimensions. Wright saw that "*The correlation between two variables can be shown to equal the sum of the products of the chains of path coefficients along all of the paths by which they are connected*" (Wright 1920, p. 115). This insight lead Wright to propose a graphical method for thinking about causation which, he realized, could be generalized, enabling decomposition of associations among variables into modeled sources of variance: for instance environmental and genetic variance (Wright 1920). His now-classic and influential path diagram from this paper is shown in Figure 10.

Surprisingly, this graphical method for thinking about causation, so well suited to variables measured with error, and to modeling of multiple outcomes and causal effects, where each variable may have multiple influences, lay fallow for 30 years until its importance was recognized once more in the 1960s and 1970s and sociologist (Duncan 1966) and economist (Goldberger

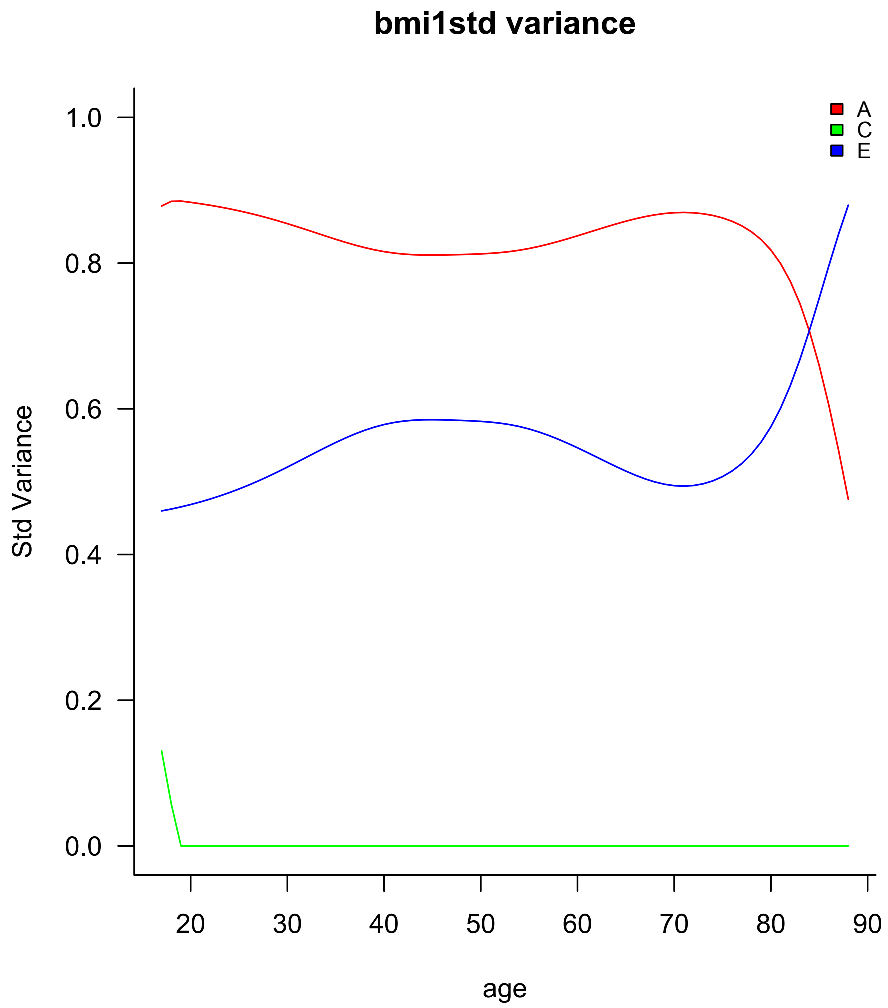


Figure 9: Output graphic from a windowed or "LOSEM" G×Age analysis.

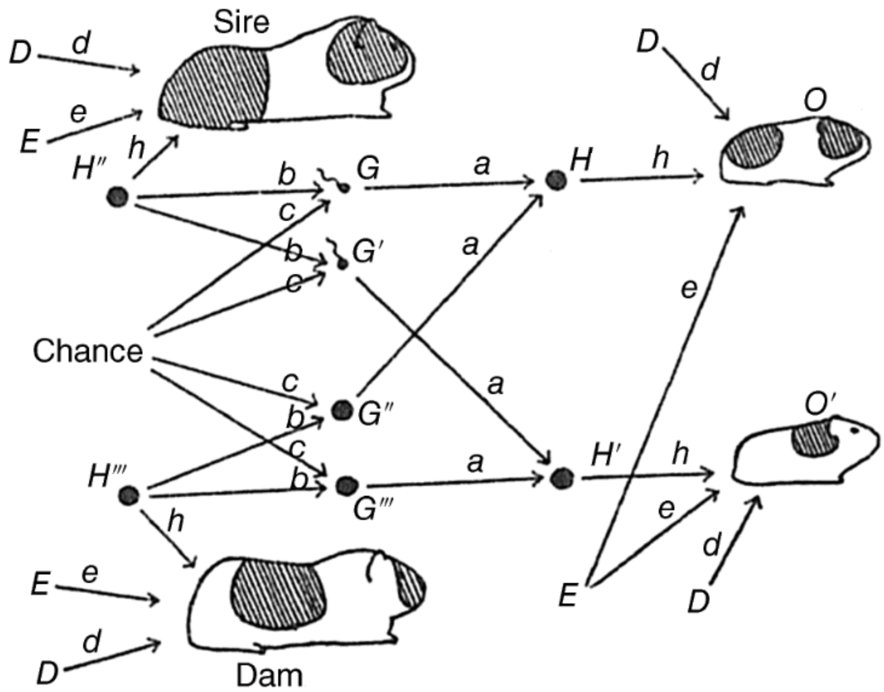


Figure 10: The first path diagram, showing inheritance of piebald pattern in the Guinea Pig (Wright 1920)

.

1971) alerted their fields to the value of structural equation modeling. Importantly, this alert was accompanied by the advent of accessible and powerful computer resources, which made practical widespread use of such models and saw the emergence of application-level software support for structural equation modeling (Jöreskog 1969b). Enhanced by developments continuing to the present day, modern extended SEM allows causal modeling of data (Pearl 2009), including continuous, ordinal and binary measures.

Models of these data types can include not only measured variables, but unmeasured "latent" constructs and relations among multiple variables. These measurement and structural models are conveniently represented graphically, with a standard set of symbols consisting of boxes (measured or manifest variables), circles (latent, or unmeasured variables), triangles (means) and diamonds (subject-specific or definition variables) and connections or paths among these objects consisting of single-headed directional paths representing directed causal influences, and two-headed curved paths representing covariance between variables. The model, once run, yields parameter estimates of these paths that may be extracted, used to form scores, plotted, etc. Models themselves can be compared using goodness of fit indicators. A moderately complex model, modified from (Duncan, Haller, and Portes 1968), and using modern symbols is shown in Figure 11. In Figure 1), a much simpler model shows how one might begin to graphically model the relationship of fuel efficiency (miles per gallon) of a car to its weight and the size (cubic capacity) of its engine in a path diagram. In this model, engine size and weight correlate (.89) and negatively influence mpg.

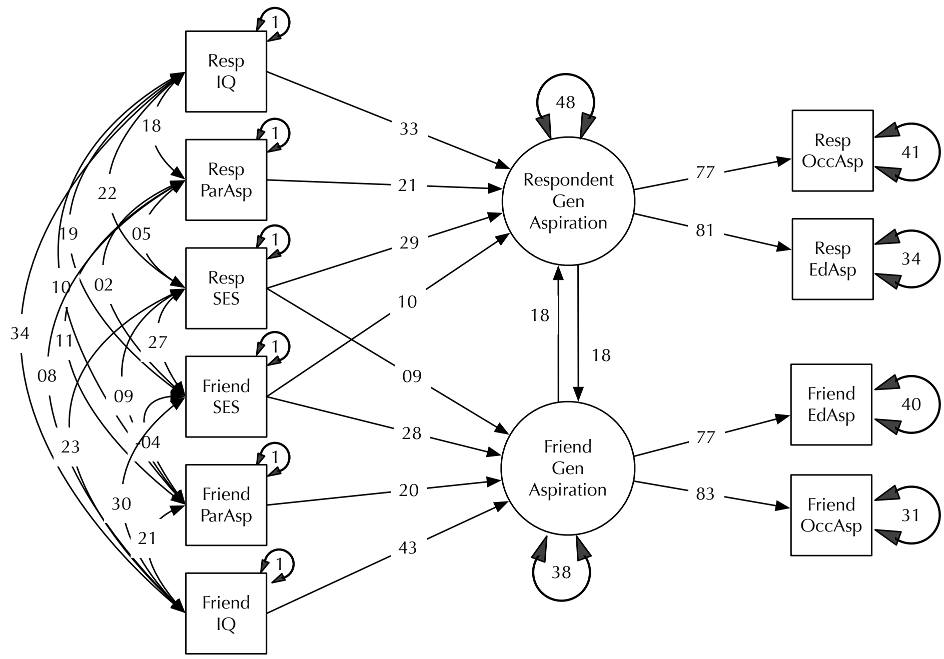


Figure 11: A model of Aspiration, modified from (modified from Duncan *et al.* 1968).

7. Appendix 2: Miscellaneous help functions

In this short article, it is not appropriate to cover all the miscellaneous functions of **OpenMx**.

Others that the user may find helpful and wish to explore include support for setting options. These include `umx_set_optimizer` to get and set the optimizer (including a check for valid optimizer names) and `umx_set_cores` to get and set the number of processor cores that **OpenMx** will exploit on multi-core machines, and a convenience function `umx_check_parallel` which runs a parallel script on a user-determined number of cores to test if the installed version of **OpenMx** is running with OpenMP support and is using multiple cores where available. This also informs the user how to obtain a multi-core version of **OpenMx** from the **OpenMx** web-site. To facilitate check-pointing of scripts which may fail to converge, **umx** includes the `umx_checkpoint`, and `umx_set_checkpoint` functions.

Condition and type checking

Code resilience is supported by robust input checking, and **umx** provides a suite of input checking functions. Thes are also used to provide feedback to the user, enabling them to more rapidly correct inputs (for instance by telling them what was expected and also showing them what they provided to specific parameters when functions fail). Checking functions include checking what OS the user is running, whether an object is an **OpenMx** model or a `RAM` etc.. Additional check functions test if a model has been run, whether it has confidence intervals or whether data are included in the model, checking whether data are ordered (as factors must be in ordinal analyses). In the next code snippet, we assume the following model exists:

```
data(demoOneFactor)
df = demoOneFactor
manifests = names(df)
myData    = mxData(cov(df), type = "cov", numObs = 500)
m1 <- umxRAM("not_run", run = F, data = myData,
    umxPath("g", to = manifests),
    umxPath(var = manifests),
    umxPath(var = "g", fixedAt = 1)
)
```

Given this, the following checks are possible:

```
umx_check_OS("OSX")  # TRUE
umx_is_MxModel(m1)   # TRUE
umx_is_RAM(m1)       # TRUE
umx_has_been_run(m1) # FALSE
umx_has_been_run(m2) # TRUE
umx_has_CIs(m1)      # FALSE
umx_is_mxMatrix(m1)  # FALSE (not a matrix)
```

Because binary and ordinal columns distinctly from continuous columns must be treated differently from continuous variables and from each other, **umx** includes the `umx_is_ordinal` function to test this status:

```
tmp = mtcars
tmp$cyl = ordered(mtcars$cyl) # ordered factor
tmp$vs  = ordered(mtcars$vs) # binary factor
umx_is_ordered(tmp) # numeric indices
```

The user can also request the names of variables matching the criteria:

```
umx_is_ordered(tmp, names = TRUE)
```

Moreover it is possible to select specific types of data:

```
umx_is_ordered(tmp, binary.only  = TRUE)
umx_is_ordered(tmp, ordinal.only = TRUE)
umx_is_ordered(tmp, continuous.only = TRUE)
```

`mxFactor` objects are required to be ordered. By default if the function finds unordered factors, it alerts the user to the fact that one or more columns are not ordered, and which they are:

```
tmp$gear = factor(mtcars$gear) # unordered factor
umx_is_ordered(tmp)
# Dataframe contains at least 1 unordered factor. Set strict = FALSE to allow this.
# "gear"
```

Ancillary functions

umx evolved in the context of conducting twin- and experimental-research using **R**. As such, it includes additional functions not restricted to structural model in **OpenMx**, and which the authors have found useful enough to include in the package. These include functions for returning a heterochoric correlation matrix (`umx_hetcor`), computing column means on data.frames with mixed data types (`umx_means`), functions for flexibly residualising (`umx_residualize`) wide-format twin data, as well as for simulating twin data.

Another common task is the construction of variable name lists which repeat for each twin, expanding base names for variables (e.g. “bmi”) into fully specified family-wise row names for variables c(“bmi_T1”, “bmi_T2”). umx simplifies this task with the `umx_paste_names` helper. This allows the user to list just the handful of base names, an optional string dividing this base name from the twin-specific suffix, and list of twin suffixes. So, to produce the desired output list. So:

`umx_paste_names(c("bmi", "IQ"), "_T", 1:2)`

yields the vector:

`c("bmi_T1", "IQ_T1", "bmi_T2", "IQ_T2")`

Twin and cohort datasets often contain many hundreds or even thousands of variables. Users often also import data from SPSS with short variable names. To ease searching these, **umx** includes grep-enabled search function `umx_names`. This functions like the built-in `names` function, but allows the user to include a regular expression to filter the list of found names. Thus in a dataframe “df” containing many names where the user may only want variables beginning with“DSM_”, this call would achieve the goal: `umx_names(df, pattern = "D̂SM_")`

umx also contains functions to aid processing strings (`umx_rot`; `umx_trim`, `umx_explode`, `umx_grep`), printing (`umx_print`; `umx_msg`), find and changing variable names (`umx_names`, `umx_grep`, `umx_rename`). Useful functions are included for APA-style summary-report tables for `"lm"` models (`summaryAPA`), and lower-level functions for formatting values (`umx_APA_pval`, `umx_round`).

A feature which may be of interest to those running time-consuming models is the `umx_pb_note` function which will push a notification (for instance when model has completed) to your smartphone or desktop using the free PushBullet®service.

8. Appendix 3: Migrating from OpenMx for Path-based models

For users who have already learned base **OpenMx**, `umxRAM` and `umxPath` will be familiar, but distinct in around seven important ways. In **OpenMx**, path-based models are specified by passing `type = "RAM"` to the `mxModel` function, along with a vector of manifest variable names, another of latent variable names, a series of `mxPaths`, and an `mxData` object. The following code implements the model shown in Figure 1 in base **OpenMx** functions:

```
library("OpenMx")
m1 <- mxModel("big", type= "RAM",
    latentVars = NULL,
    manifestVars = c("disp", "wt", "mpg"),
    mxPath(from = c("disp", "wt"), to = "mpg"),
    mxPath(from = "disp", to = "wt", arrows= 2),
    mxPath(from = c("disp", "wt", "mpg"), arrows = 2),
    mxPath(from = "one", to = c("disp", "wt", "mpg")),
    mxData(mtcars, type = "raw")
)
m1 = mxRun(m1)
summary(m1)
```

By contrast, this is the same model using `umx`:

```
library("umx")
m1 <- umxRAM("big", data = mxData(mtcars, type = "raw"),
    umxPath(c("disp", "wt"), to = "mpg"),
    umxPath("disp", with = "wt"),
    umxPath(v.m. = c("disp", "wt", "mpg"))
)
plot(m1)
```

The first change from **OpenMx** to **umx**, is that a dedicated path-based model function is provided: `umxRAM`. This frees the user from having to specify model type. As with R functions such as `"lm"`, `umxRAM` provides a `data= parameter` for the model data.

Second, `umxRAM` detects which variable names have been used in creating the model, and maps these to the data. This saves the user having to specify the OpenMx objects `manifestVars` and `latentVars` because `umxRAM` automatically adds names not found in the data to the latent variable list. The user is given feedback including which manifests have been detected, which latent variables were created, and which data columns were dropped from the data.frame, and how the data have been interpreted (see above for example of this feedback). `umxRAM` also automatically drops unused variables from the modelled data.

Third, `umxRAM` automatically labels paths in a consistent fashion. A path from `a` to `b` for instance, is labeled `a_to_b`. Two-headed paths are labeled in a similar consistent fashion, e.g. `a_with_b` (name order is sorted to avoid unambiguity). Fourth, `umx` sets starting values that are likely to allow model estimation to begin successfully. This can save experts valuable time.

Fifth, as a convenience, by default `umxRAM` automatically runs the model, and prints a user summary. Sixth, `umx` implements S3 functions supporting graphical output and table-based reporting via `plot()` and `umxSummary`, as well as generic and core R S3 functions including `confint`, and `residuals` that users are used to using e.g. with `lm` objects.

Finally, but importantly, `umxPath` extends `mxPath` in several ways. `umxPath` provides additional adjectives supporting more explicit path descriptions that involve fewer keystrokes. For example, to create a 2-headed path fixed at 1, `with` and `fixedAt` allow the user to say: `umxPath("a", with= "b", fixedAt = 1)`. This is easier to type, to read, and to maintain than:

`mxPath(from = "a", to = "b", arrows = 2, free = FALSE, values = 1)`

Similarly, to aid readability, rather than specifying a means model thus: `mxPath(from = "one", to = "b")`, the user can say:

`umxPath(means = "b")`

Other parameters compress multiple statements into a single call. For instance a common task is specifying variable means and variances. Common cases include making a normalized latent variable (mean == 0, variance == 1) or allowing these parameters to be freely estimated . `umxPath`, the user can specify these using the `v1m0` and `v.m.` parameters respectively. The character `.` was chosen by analogy with the wild-card character which matches any value in regular expressions. As an example, the following shows the 2-lines of **umx** code, followed by the four lines of **OpenMx** code which they replicate:

```
# umx
umxPath(v1m0 = "g") # fixed at variance of 1, mean = 0
umxPath(v.m. = c("disp", "wt", "mpg")) # freely estimated

# OpenMx
mxPath(from = "g", arrows = 2, free = FALSE, values = 1),
mxPath(from = "one", to =  "g", free = FALSE, values = 0),
mxPath(from = c("disp", "wt", "mpg"), arrows=2, free = TRUE, values = .1),
mxPath(from = "one", to =  c("disp", "wt", "mpg"), free = TRUE, values = 10)
```

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 (2016). "OpenMx.tmbundle Textmate support bundle for umx and OpenMx." doi:10.5281/zenodo.46448. URL http://dx.doi.org/10.5281/zenodo.46448.

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.

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 O (1966). "Path analysis: sociological examples." *The American Journal of Sociology*, **72**(1), 1–16.

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.

Goldberger A (1971). "Econometrics and psychometrics: a survey of communalities." *Econometrica*, **36**(6), 841–868.

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.

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.

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 (Print) 0033-2917 (Linking). 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.

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.

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 Ωnyx." *Structural Equation Modeling: A Multidisciplinary Journal*, **22**(1), 148–161. ISSN 1070-5511.

Wright S (1918). "On the nature of size factors." *The Annals of Mathematical Statistics*, **3**, 367–374.

Wright S (1920). "The relative importance of heredity and environment in determining the piebald pattern of guinea-pigs." *Proceedings of the National Academy of Sciences of the United States of America*, **6**, 320–332.

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.

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

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd