Title

stata.com

intro 2 — Learning the language: Path diagrams and command language

Description Remarks and examples Reference Also see

Description

Individual structural equation models are usually described using path diagrams. Path diagrams are described here.

Path diagrams can be used in `sem`'s (`gsem`'s) GUI, known as the SEM Builder or simply the Builder, as the input to describe the model to be fit. Path diagrams differ a little from author to author, and `sem`'s and `gsem`'s path diagrams differ a little, too. For instance, we omit drawing the variances and covariances between observed exogenous variables by default.

`sem` and `gsem` also provide a command-language interface. This interface is similar to path diagrams but is typable.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Using path diagrams to specify standard linear SEMs

Specifying correlation

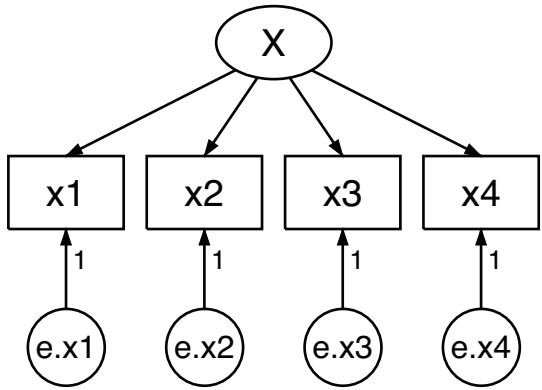
Using the command language to specify standard linear SEMs

Specifying generalized SEMs: Family and link
Specifying generalized SEMs: Family and link, multinomial logistic regression
Specifying generalized SEMs: Family and link, paths from response variables

Specifying generalized SEMs: Multilevel mixed effects (2 levels)
Specifying generalized SEMs: Multilevel mixed effects (3 levels)
Specifying generalized SEMs: Multilevel mixed effects (4+ levels)
Specifying generalized SEMs: Multilevel mixed effects with random intercepts
Specifying generalized SEMs: Multilevel mixed effects with random slopes
Specifying generalized SEMs: Latent class analysis (LCA)
Specifying generalized SEMs: Latent class analysis, class predictors
Specifying generalized SEMs: Latent class analysis, two latent variables

Using path diagrams to specify standard linear SEMs

In structural equation modeling, models are often illustrated in a path diagram such as this one:



This diagram is composed of the following:

1. Boxes and circles with variable names written inside them.
    a. Boxes contain variables that are observed in the data.
    b. Circles contain variables that are unobserved, known as latent variables.
2. Arrows, called paths, that connect some of the boxes and circles.
    a. When a path points from one variable to another, it means that the first variable affects the second.
    b. More precisely, if $s \rightarrow d$, it means to add $\beta_k s$ to the linear equation for d. β_k is called the path coefficient.
    c. Sometimes small numbers are written along the arrow connecting two variables. This means that β_k is constrained to be the value specified. (Some authors use the term "path coefficient" to mean standardized path coefficient. We do not.)
    d. When no number is written along the arrow, the corresponding coefficient is to be estimated from the data. Sometimes symbols are written along the path arrow to emphasize this and sometimes not.
    e. The same path diagram used to describe the model can be used to display the results of estimation. In that case, estimated coefficients appear along the paths.
3. There are other elements that may appear on the diagram to indicate variances and between-variable correlations. We will get to them later.

Thus the above figure corresponds to the equations

$$x_1 = \alpha_1 + \beta_1 X + e.x_1$$

$$x_2 = \alpha_2 + \beta_2 X + e.x_2$$

$$x_3 = \alpha_3 + \beta_3 X + e.x_3$$

$$x_4 = \alpha_4 + \beta_4 X + e.x_4$$

There's a third way of writing this model, namely,

```
(x1<-X) (x2<-X) (x3<-X) (x4<-X)
```

This is the way we would write the model if we wanted to use `sem`'s or `gsem`'s command syntax rather than drawing the model in the Builder. The full command we would type is

```
. sem (x1<-X) (x2<-X) (x3<-X) (x4<-X)
```

We will get to that later.

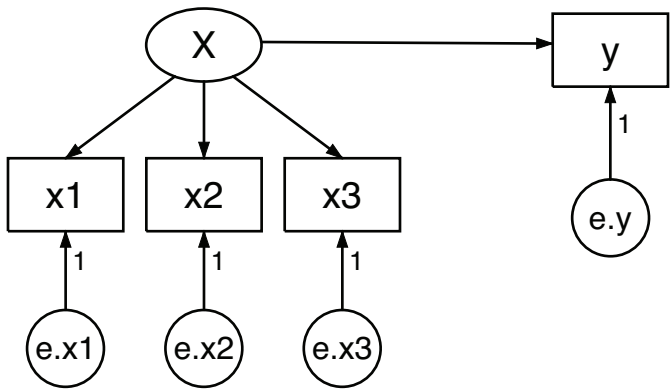
By the way, the above model is a linear single-level model. Linear single-level models can be fit by `sem` or by `gsem`. `sem` is the preferred way to fit linear single-level models because it has added features for these models that you might find useful later. Nonetheless, if you want to fit the model with `gsem`, you would type

```
. gsem (x1<-X) (x2<-X) (x3<-X) (x4<-X)
```

Whether we use `sem` or `gsem`, we obtain the same results.

However we write this model, what is it? It is a measurement model, a term loaded with meaning for some researchers. X might be mathematical ability. x_1, x_2, x_3, and x_4 might be scores from tests designed to measure mathematical ability. x_1 might be the score based on your answers to a series of questions after reading this section.

The model we have just drawn, or written in mathematical notation, or written in Stata command notation, can be interpreted in other ways, too. Look at this diagram:



Despite appearances, this diagram is identical to the previous diagram except that we have renamed x_4 to be y. The fact that we changed a name obviously does not matter substantively. The fact that we have rearranged the boxes in the diagram is irrelevant, too; paths connect the same variables in the same directions. The equations for the above diagrams are the same as the previous equations with the substitution of y for x_4:

$$x_1 = \alpha_1 + X\beta_1 + e.x_1$$

$$x_2 = \alpha_2 + X\beta_2 + e.x_2$$

$$x_3 = \alpha_3 + X\beta_3 + e.x_3$$

$$y = \alpha_4 + X\beta_4 + e.y$$

The Stata command notation changes similarly:

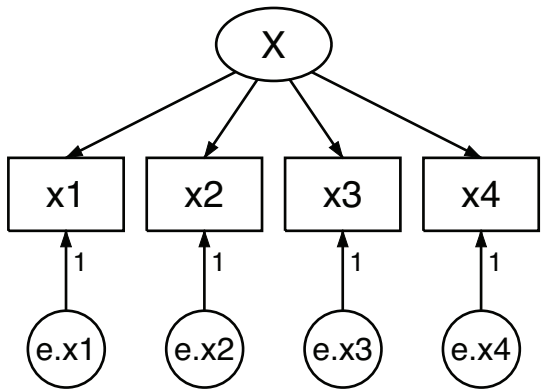
```
(x1<-X) (x2<-X) (x3<-X) (y<-X)
```

Many people looking at the model written this way might decide that it is not a measurement model but a measurement-error model. y depends on X but we do not observe X. We do observe x_1, x_2, and x_3, each a measurement of X, but with error. Our interest is in knowing β_4, the effect of true X on y.

A few others might disagree and instead see a model for interrater agreement. Obviously, we have three or four raters who each make a judgment, and we want to know how well the judgment process works and how well each of these raters performs.

Specifying correlation

One of the key features of SEM is the ease with which you can allow correlation between latent variables to adjust for the reality of the situation. In the measurement model in the previous section, we drew the following path diagram:



which corresponded to the equations

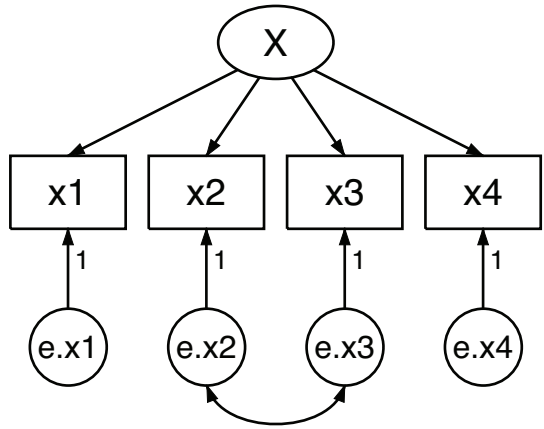
$$x_1 = \alpha_1 + X\beta_1 + e.x_1$$
$$x_2 = \alpha_2 + X\beta_2 + e.x_2$$
$$x_3 = \alpha_3 + X\beta_3 + e.x_3$$
$$x_4 = \alpha_4 + X\beta_4 + e.x_4$$

$(X, x_1, x_2, x_3, x_4, e.x_1, e.x_2, e.x_3, e.x_4) \sim$ i.i.d. with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$

where i.i.d. means that observations are independent and identically distributed.

We must appreciate that $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are estimated, just as are α_1, β_1, …, α_4, β_4. Some of the elements of $\boldsymbol{\Sigma}$, however, are constrained to be 0; which elements are constrained is determined by how we specify the model. In the above diagram, we drew the model in such a way that we assumed that error variables were uncorrelated with each other. We could have drawn the diagram differently.

If we wish to allow for a correlation between $e.x_2$ and $e.x_3$, we add a curved path between the variables:



The curved path states that there is a correlation to be estimated between the variables it connects. The absence of a curved path—say, between $e.x_1$ and $e.x_4$—means that the variables are constrained to be uncorrelated. That is not to say that x_1 and x_4 are uncorrelated; obviously, they are correlated because both are functions of the same X. Their corresponding error variables, however, are uncorrelated. The equations for this model, in their full detail, are

$$x_1 = \alpha_1 + X\beta_1 + e.x_1$$

$$x_2 = \alpha_2 + X\beta_2 + e.x_2$$

$$x_3 = \alpha_3 + X\beta_3 + e.x_3$$

$$x_4 = \alpha_4 + X\beta_4 + e.x_4$$

$$(X, x_1, x_2, x_3, x_4, e.x_1, e.x_2, e.x_3, e.x_4) \sim \text{i.i.d. with mean } \boldsymbol{\mu} \text{ and variance } \boldsymbol{\Sigma}$$

$\boldsymbol{\Sigma}$ is constrained such that

$$\begin{aligned}
\sigma_{e.x_1,e.x_2} &= \sigma_{e.x_2,e.x_1} = 0 \\
\sigma_{e.x_1,e.x_3} &= \sigma_{e.x_3,e.x_1} = 0 \\
\sigma_{e.x_1,e.x_4} &= \sigma_{e.x_4,e.x_1} = 0 \\
\sigma_{e.x_2,e.x_4} &= \sigma_{e.x_4,e.x_2} = 0 \\
\sigma_{e.x_3,e.x_4} &= \sigma_{e.x_4,e.x_3} = 0
\end{aligned}$$

$$\begin{aligned}
\sigma_{X,e.x_1} &= \sigma_{e.x_1,X} = 0 \\
\sigma_{X,e.x_2} &= \sigma_{e.x_2,X} = 0 \\
\sigma_{X,e.x_3} &= \sigma_{e.x_3,X} = 0 \\
\sigma_{X,e.x_4} &= \sigma_{e.x_4,X} = 0
\end{aligned}$$

$\boldsymbol{\mu}$ is constrained such that

$$\begin{aligned}
\mu_X &= 0 \\
\mu_{e.x_1} &= 0 \\
\mu_{e.x_2} &= 0 \\
\mu_{e.x_3} &= 0 \\
\mu_{e.x_4} &= 0
\end{aligned}$$

The parameters to be estimated are

$$\begin{aligned}
&\alpha_1, \beta_1, \ldots, \alpha_4, \beta_4, \\
&\boldsymbol{\mu} \\
&\boldsymbol{\Sigma}
\end{aligned}$$

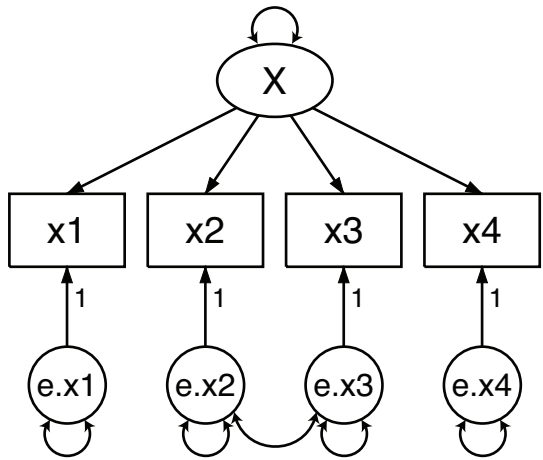
Look carefully at the above list. You will not find a line reading

$$\sigma_{e.x_2,e.x_3} = \sigma_{e.x_3,e.x_2} = 0$$

although you will find lines constraining the other covariances between error terms to be 0. The line is missing because we drew a curved path between $e.x_2$ and $e.x_3$.

There are lots of other curved arrows we could have drawn. By not drawing them, we are asserting that the corresponding covariance is 0.

Some authors would draw the above model as



A curved path from a variable to itself indicates a variance. All covariances could be shown, including those between latent variables and between observed exogenous variables.

When we draw diagrams, however, we will assume variance paths and omit drawing them, and we will similarly assume but omit drawing covariances between observed exogenous variables (there are no observed exogenous variables in this model). The Builder in `sem` mode has an option concerning the latter. Like everyone else, we will not assume correlations between latent variables unless they are shown.

In `sem`'s (`gsem`'s) command-language notation, curved paths between variables are indicated via an option:

```
(x1<-X) (x2<-X) (x3<-X) (x4<-X), cov(e.x2*e.x3)
```

Using the command language to specify standard linear SEMs

You can describe your model to `sem` by using path diagrams with the Builder, or you can describe your model by using `sem`'s command language. Here are the trade-offs:

1. If you use path diagrams, you can see the results of your estimation as path diagrams or as standard computer output.
2. If you use the command language, only standard computer output is available.
3. Typing models in the command language is usually quicker than drawing them in the Builder.
4. You can type models in the command language and store them in do-files. By doing so, you can more easily correct the errors you make.

Translating from path diagrams to command language is easy.

1. Path diagrams have squares and circles to distinguish observed from latent variables.

   In the command language, variables are assumed to be observed if they are typed in lowercase and are assumed to be latent if the first letter is capitalized. Variable `educ` is observed, while variable `Knowledge` or `KNOWLEDGE` is latent.

   If the observed variables in your dataset have uppercase names, type `rename _all, lower` to covert them to lowercase; see [D] **rename group**.

2. When typing path diagrams in the command language, remember the `///` continuation line indicator. You may type

```
. sem (x1<-X) (x2<-X) (x3<-X) (y<-X)
```

   or you may type

```
. sem (x1<-X) (x2<-X)    ///
      (x3<-X) (y<-X)
```

   or you may type

```
. sem (x1<-X)            ///
      (x2<-X)            ///
      (x3<-X)            ///
      (y<-X)
```

3. In path diagrams, you draw arrows connecting variables to indicate paths. In the command language, you type variable names and arrows. So long as your arrow points the correct way, it does not matter which variable comes first. The following mean the same thing:

```
(x1 <- X)
(X -> x1)
```

4. In the command language, you may type multiple variables on either side of the arrow:

```
(X -> x1 x2 x3 x4)
```

   The above means the same as

```
(X -> x1) (X -> x2) (X -> x3) (X -> x4)
```

   which means the same as

```
(x1 <- X) (x2 <- X) (x3 <- X) (x4 <- X)
```

   which means the same as

```
(x1 x2 x3 x4 <- X)
```

   In a more complicated measurement model, we might have

```
(X Y -> x1 x2 x3) (X -> x4 x5) (Y -> x6 x7)
```

   The above means the same as

```
(X -> x1 x2 x3 x4 x5)  ///
(Y -> x1 x2 x3 x6 x7)
```

which means

```
(X -> x1) (X -> x2) (X -> x3) (X -> x4) (X -> x5)     ///
(Y -> x1) (Y -> x2) (Y -> x3) (Y -> x6) (Y -> x7)
```

5. In path diagrams, you are required to show the error variables. In the command language, you may omit the error variables. `sem` knows that each endogenous variable needs an error variable. You can type

```
(x1 <- X) (x2 <- X) (x3 <- X) (x4 <- X)
```

and that means the same thing as

```
(x1 <- X e.x1)   ///
(x2 <- X e.x2)   ///
(x3 <- X e.x3)   ///
(x4 <- X e.x4)
```

except that we have lost the small numeric 1s we had next to the arrows from `e.x1` to `x1`, `e.x2` to `x2`, and so on. To constrain the path coefficient, you type

```
(x1 <- X e.x1@1)    ///
(x2 <- X e.x2@1)    ///
(x3 <- X e.x3@1)    ///
(x4 <- X e.x4@1)
```

It is easier to simply type

```
(x1 <- X) (x2 <- X) (x3 <- X) (x4 <- X)
```

but now you know that if you wanted to constrain the path coefficient `x2<-X` to be 2, you could type

```
(x1 <- X) (x2 <- X@2) (x3 <- X) (x4 <- X)
```

If you wanted to constrain the path coefficients `x2<-X` and `x3<-X` to be equal, you could type

```
(x1 <- X) (x2 <- X@b) (x3 <- X@b) (x4 <- X)
```

We have not covered symbolic constraints with path diagrams yet, but typing symbolic names along the paths in either the Builder or the command language is how you constrain coefficients to be equal.

6. Curved paths are specified with the `cov()` option after you have specified your model:

```
(x1 x2 x3 x4 <- X), cov(e.x2*e.x3)
```

If you wanted to allow for correlation of `e.x2*e.x3` and `e.x3*e.x4`, you can specify that in a single `cov()` option,

```
(x1 x2 x3 x4 <- X), cov(e.x2*e.x3 e.x3*e.x4)
```

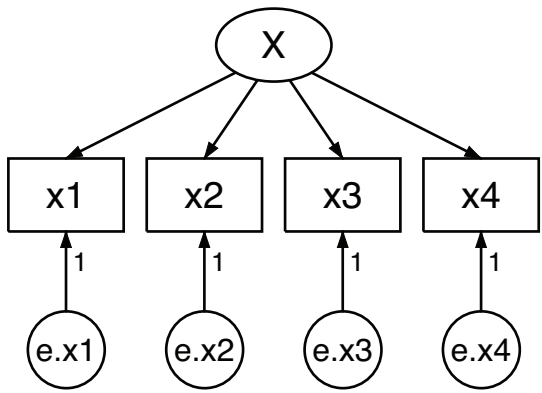
or in separate `cov()` options:

```
(x1 x2 x3 x4 <- X), cov(e.x2*e.x3) cov(e.x3*e.x4)
```

7. Nearly all the above applies equally to `gsem`. We have to say "nearly" because sometimes, in some models, some concepts simply vanish. For instance, in a logistic model, there are no error terms. For generalized responses with family Gaussian, link log, there are error terms, but they cannot be correlated. Also, for responses with family Gaussian, link identity, and censoring, there are error terms, but they cannot be correlated. `gsem` also takes observed exogenous variables as given and so cannot estimate the covariances between them.

Specifying generalized SEMs: Family and link

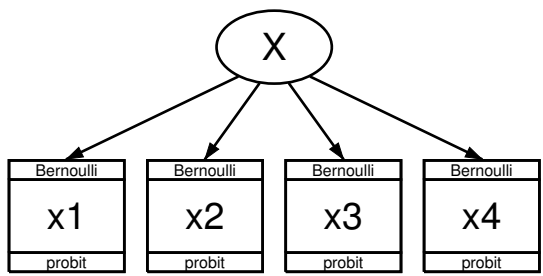
We began this discussion by showing you a linear measurement model:



In this model, we observe continuous measurements `x1`, . . . , `x4`. What if `x1`, . . . , `x4` were instead binary outcomes such as success and failure or passed and failed? That is, what if `x1`, . . . , `x4` took on values of only 1 and 0?

In that case, we would want to fit a model appropriate to binary outcomes. Perhaps we want to fit a logistic regression model or a probit model. To do either one, we will have to use `gsem` rather than `sem`. We will use a probit model.

The path diagram for the measurement model with binary outcomes is



What were plain boxes for `x1`, . . . , `x4` now say "Bernoulli" and "probit" at the top and bottom, meaning that the variable is from the Bernoulli family and is using the probit link. In addition, `e.x1`, . . . , `e.x4` (the error terms for `x1`, . . . , `x4`) have disappeared. In the generalized linear framework, there are error terms only for the Gaussian family.

Perhaps some math will clear up the issue. The generalized linear model is

$$g\{E(y\,|\,X)\} = \mathbf{x}\beta$$

and in the case of probit, $g\{E(y\,|\,X)\} = \Phi^{-1}\{E(y\,|\,X)\}$, where $\Phi(\cdot)$ is the cumulative normal distribution. Thus the equations are

$$\Phi^{-1}\{E(x_1\,|\,X)\} = \alpha_1 + X\beta_1$$

$$\Phi^{-1}\{E(x_2\,|\,X)\} = \alpha_2 + X\beta_2$$

$$\Phi^{-1}\{E(x_3\,|\,X)\} = \alpha_3 + X\beta_3$$

$$\Phi^{-1}\{E(x_4\,|\,X)\} = \alpha_4 + X\beta_4$$

Note that there are no error terms. Equivalently, the above can be written as

$$\Pr(x_1 = 1\,|\,X) = \Phi(\alpha_1 + X\beta_1)$$

$$\Pr(x_2 = 1\,|\,X) = \Phi(\alpha_2 + X\beta_2)$$

$$\Pr(x_3 = 1\,|\,X) = \Phi(\alpha_3 + X\beta_3)$$

$$\Pr(x_4 = 1\,|\,X) = \Phi(\alpha_4 + X\beta_4)$$

In `gsem`'s command language, we write this model as

```
(x1 x2 x3 x4<-X, family(bernoulli) link(probit))
```

or as

```
(x1<-X, family(bernoulli) link(probit))
(x2<-X, family(bernoulli) link(probit))
(x3<-X, family(bernoulli) link(probit))
(x4<-X, family(bernoulli) link(probit))
```

In the command language, you can simply type `probit` to mean `family(bernoulli) link(probit)`, so the model could also be typed as

```
(x1 x2 x3 x4<-X, probit)
```

or even as

```
(x1<-X, probit) (x2<-X, probit) (x3<-X, probit) (x4<-X, probit)
```

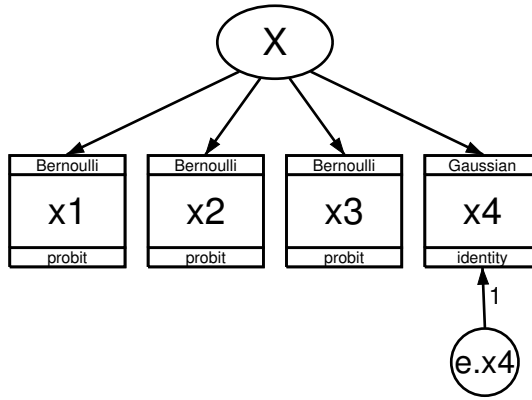
Whether you type `family(bernoulli) link(probit)` or type `probit`, when all the response variables are probit, you can type

```
(x1 x2 x3 x4<-X), probit
```

or

```
(x1<-X) (x2<-X) (x3<-X) (x4<-X), probit
```

The response variables do not have to be all from the same family and link. Perhaps `x1`, `x2`, and `x3` are pass/fail variables but `x4` is a continuous variable. Then the model would be diagrammed as



The words "Gaussian" and "identity" now appear for variable `x4` and `e.x4` is back! Just as previously, the generalized linear model is

$$g\{E(y \mid X)\} = \mathbf{x}\boldsymbol{\beta}$$

and in the case of linear regression, $g(\mu) = \mu$, so our fourth equation becomes

$$E(x_4 \mid X) = X\beta_4$$

or

$$x_4 = \alpha + X\beta_4 + e.x_4$$

and the entire set of equations to be estimated is

$$\Pr(x_1 = 1 \mid X) = \Phi(\alpha_1 + X\beta_1)$$

$$\Pr(x_2 = 1 \mid X) = \Phi(\alpha_2 + X\beta_2)$$

$$\Pr(x_3 = 1 \mid X) = \Phi(\alpha_3 + X\beta_3)$$

$$x_4 = \alpha_4 + X\beta_4 + e.x_4$$

This can be written in the command language as

```
(x1 x2 x3<-X, family(bernoulli) link(probit))
(x4      <-X, family(gaussian)  link(identity))
```

or as

```
(x1 x2 x3<-X, probit) (x4<-X, regress)
```

`regress` is a synonym for `family(gaussian) link(identity)`. Because `family(gaussian) link(identity)` is the default, we can omit the option altogether:

```
(x1 x2 x3<-X, probit) (x4<-X)
```

We demonstrated generalized linear models above by using probit (family Bernoulli, link probit), but we could just as well have used logit (family Bernoulli, link logit). Nothing changes except that in the path diagrams, where you see probit, logit would now appear. Likewise, in the command, where you see `probit`, `logit` would appear.

The same is true with almost all the other possible generalized linear models. What they all have in common is

1. There are no `e.` error terms, except for family Gaussian.
2. Response variables appear the ordinary way except that the family is listed at the top of the box and the link is listed at the bottom of the box, except for family multinomial, link logit (also known as multinomial logistic regression or mlogit).

Concerning item 1, we just showed you a combined probit and linear regression with its $e.x4$ term. Linear regression is family Gaussian.

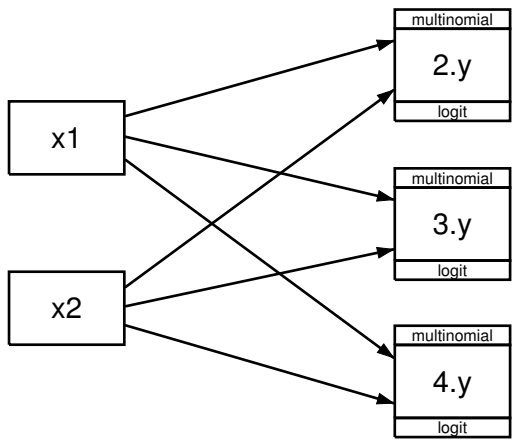
Concerning item 2, multinomial logistic regression is different enough that we need to show it to you.

Specifying generalized SEMs: Family and link, multinomial logistic regression

Let's consider a multinomial logistic model in which y takes on one of four possible outcomes and is determined by x_1 and x_2. Such a model could be fit by Stata's `mlogit` command:

```
. mlogit y x1 x2
```

The model could also be fit by `gsem`. The path diagram would be



Note that there are three boxes for y, boxes containing $2.y$, $3.y$, and $4.y$. When specifying a multinomial logistic regression model in which the dependent variables can take one of k different values, you draw $k-1$ boxes. Names like $1.y$, $2.y$, ..., mean $y=1$, $y=2$, and so on. Logically speaking, you have to omit one of them. Which you omit is irrelevant and is known as the base outcome. Estimated coefficients in the model relevant to the other outcomes will be measured as a difference from the base outcome. We said which you choose is irrelevant, but it might be easier for you to interpret your model if you chose a meaningful base outcome, such as the most frequent outcome. In fact, that is just what Stata's `mlogit` command does by default when you do not specify otherwise.

In path diagrams, you may implicitly or explicitly specify the base outcome. We implicitly specified the base by omitting $1.y$ from the diagram. We could have included it by drawing a box for $1.y$ and labeling it 1b.y. Stata understands the b to mean base category. See [SEM] **example 37g** for an example.

Once you have handled specification of the base category, you draw path arrows from the predictors to the remaining outcome boxes. We drew paths from x_1 and x_2 to all the outcome boxes, but if we wanted to omit x_1 to $3.y$ and $4.y$, we could have omitted those paths.

Our example is simple in that y is the final outcome. If we had a more complex model where y's outcome affected another response variable, arrows would connect all or some of $2.y$, $3.y$, and $4.y$ to the other response variable.

The command syntax for our simple example is

```
(2.y 3.y 4.y<-x1 x2), mlogit
```

`2.y`, `3.y`, and `4.y` are examples of Stata's factor-variable syntax. The factor-variable syntax has some other features that can save typing in command syntax. `i.y`, for instance, means `1b.y`, `2.y`, `3.y`, and `4.y`. It is especially useful because if we had more levels, say, 10, it would mean `1b.y`, `2.y`, ..., `10.y`. To fit the model we diagrammed, we could type

```
(i.y<-x1 x2), mlogit
```

If we wanted to include some paths and not others, we need to use the more verbose syntax. For instance, to omit paths from x_1 to $3.y$ and $4.y$, we would type

```
(2.y<-x1 x2) (3.y 4.y<-x2), mlogit
```

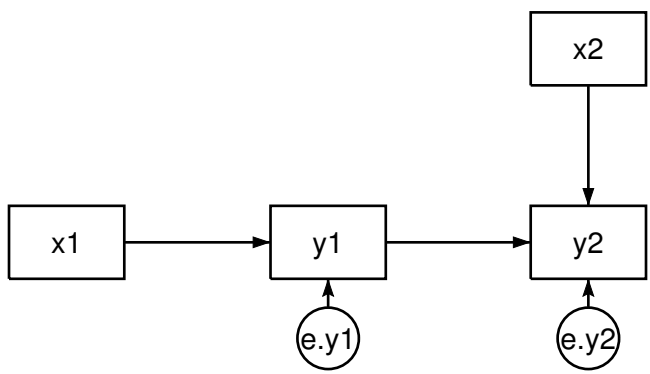
or

```
(i.y<-x2) (2.y<-x1), mlogit
```

For more information on specifying mlogit paths, see [SEM] **intro 3**, [SEM] **example 37g**, and [SEM] **example 41g**.

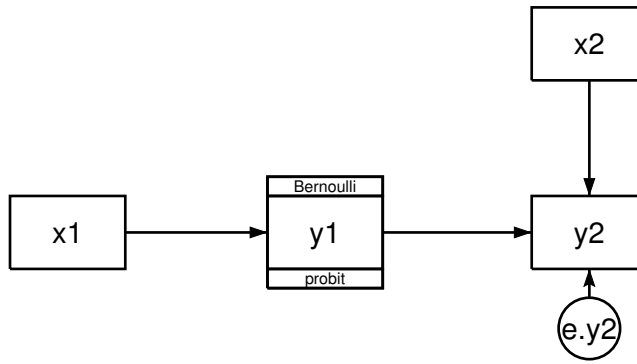
Specifying generalized SEMs: Family and link, paths from response variables

When we draw a path from one response variable to another, we are stating that the first endogenous variable is a predictor of the other endogenous variable. The diagram might look like this:



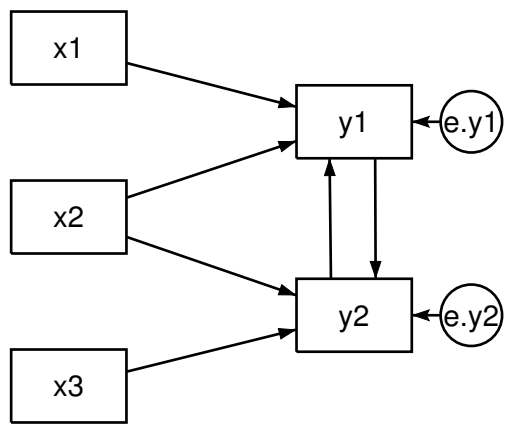
The command syntax might look like `(y1<-x1) (y2<-y1 x2)`.

The response variables in the model are linear, and note that there is a path from y_1 to y_2. Could we change y_1 to be a member of the generalized linear family, such as probit, logit, and so on? It turns out that we can:



In the command syntax, we could write this model as `(y1<-x1, probit) (y2<-y1 x2)`. (In this case, the observed values and not the expectations are used to fit the `y1->y2` coefficient. In general, this is true for all generalized responses that are not family Gaussian, link identity.)

We can make the substitution from linear to generalized linear if the path from y_1 appears in a recursive part of the model. We will define recursive shortly, but trust us that the above model is recursive all the way through. The substitution would not have been okay if the path had been in a nonrecursive portion of the model. The following is a through-and-through nonrecursive model:



It could be written in command syntax as `(y1<-y1 x1 x2) (y2<-y2 x2 x3)`.

In this model, we could not change y_1 to be family Bernoulli and link probit or any other generalized linear response variable. If we tried to fit the model with such a change, we would get an error message:

```
invalid path specification;
a loop among the paths between 'y1' and 'y2' is not allowed
r(198);
```

The software will spot the problem, but you can spot it for yourself.

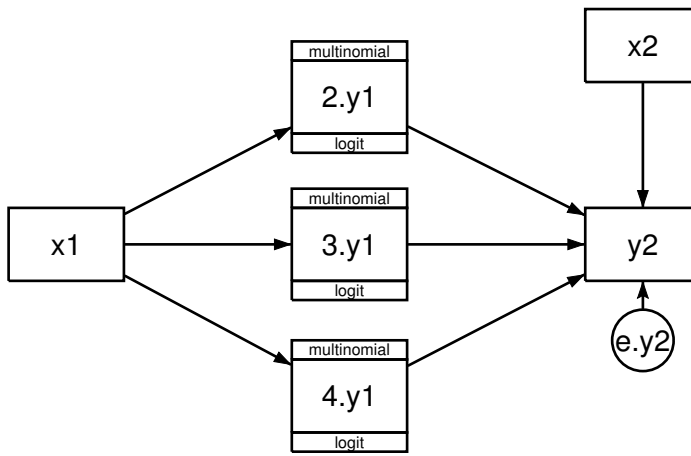
Nonrecursive models have loops. Do you see the loop in the above model? You will if you work out the total effect of a change in y_2 from a change in y_1. Assume a change in y_1. Then that change directly affects y_2, the new value of y_2 affects y_1, which in turn indirectly affects y_2 again, which affects y_1, and on and on.

Now follow the same logic with either the probit or the continuous recursive models above. The change in y_1 affects y_2 and it stops right there.

We sympathize if you think that we have interchanged the terms recursive and nonrecursive. Remember it this way: total effects in recursive models can be calculated nonrecursively because the model itself is recursive, and total effects in nonrecursive models must be calculated recursively because the model itself is nonrecursive.

Anyway, you may draw paths from generalized linear response variables to other response variables, whether linear or generalized linear, as long as no loops are formed.

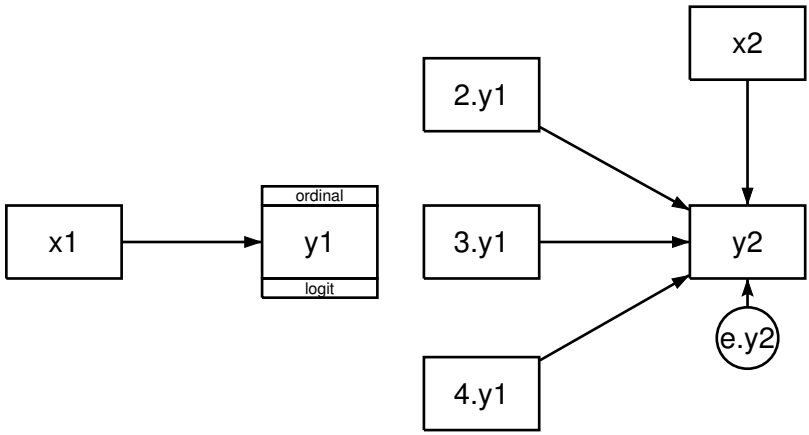
We gave special mention to multinomial logistic regression in the previous section because those models look different from the other generalized linear models. Multinomial logistic regression has a plethora of response variables. In the case of multinomial logistic regression, a recursive model with a path from the multinomial logistic outcome y_1 to (linear) y_2 would look like this:



In the command syntax, the model could be written as

```
(2.y1 3.y1 4.y1<-x1, mlogit) (y2<-2.y1 3.y1 4.y1 x2)
```

In multinomial logistic regression models, outcomes are numbered 1, 2, ..., k. The outcomes might correspond to walk, take public transportation, drive a car, and fly. In ordered probit and logistic models, outcomes are also numbered 1, 2, ..., k, and the outcomes are otherwise similar to multinomial logistic models except that the outcomes are also naturally ordered. The outcomes might be did poorly, did okay, did fairly well, and did well. After taking account of the ordering, you would probably want to model your remaining response variables in the same way you did for multinomial logistic regression. In that case, you would diagram the model like this:



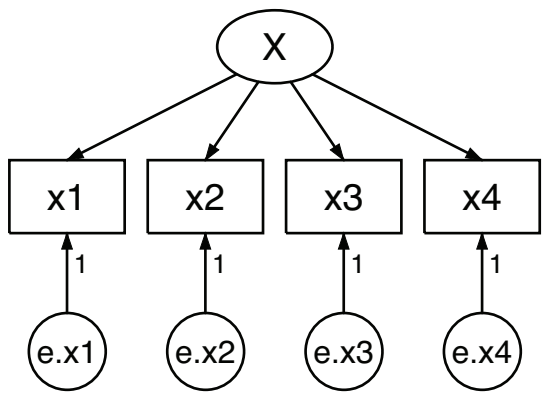
In the command syntax, the model could be written as

```
(y1<-x1, ologit) (y2<-2.y1 3.y1 4.y1 x2)
```

Unlike multinomial logistic regression, in which the k outcomes result in the estimation of $k-1$ equations, in ordered probit and logistic models, only one equation is estimated, and thus the response variable is specified simply as y_1 rather than $2.y_1$, $3.y_1$, and $4.y_1$. Even so, you probably will want the different outcomes to have separate coefficients in the y_2 equation so that the effects of being in groups 1, 2, 3, and 4 are β_0, $\beta_0+\beta_1$, $\beta_0+\beta_2$, and $\beta_0+\beta_3$, respectively. If you drew the diagram with a single path from y_1 to y_2, the effects of being in the groups would be $\beta_0+1\beta_1$, $\beta_0+2\beta_1$, $\beta_0+3\beta_1$, and $\beta_0+4\beta_1$, respectively.

Specifying generalized SEMs: Multilevel mixed effects (2 levels)

The models above are all single-level models or, equivalently, observational-level models. Let's reconsider our original measurement model:



The data for this model would look something like this:

```
. list in 1/5
```

	x1	x2	x3	x4
1.	96	82	96	43
2.	106	81	93	40
3.	127	95	96	134
4.	123	98	94	109
5.	119	99	100	108

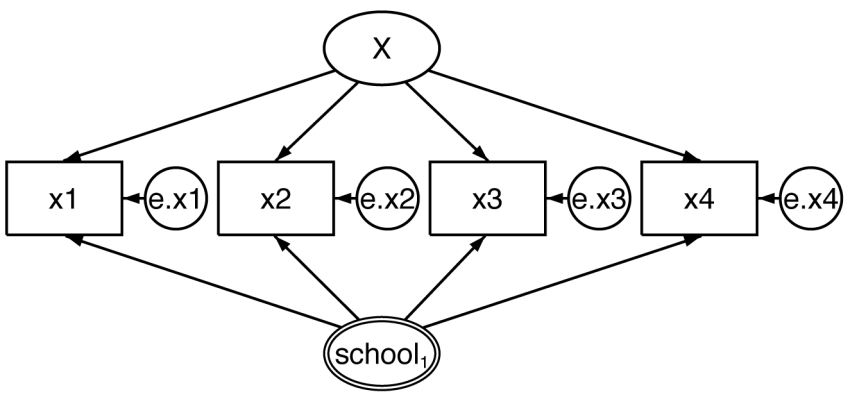
Let's pretend that the observations are students and that `x1`, . . . , `x4` are four test scores.

Now let's pretend that we have new data with students from different schools. A part of our data might look like this:

```
. list in 1/10
```

	school	x1	x2	x3	x4
1.	1	116	128	125	127
2.	1	92	104	101	103
3.	1	105	117	114	116
4.	1	117	129	126	128
5.	1	102	114	111	113
6.	2	116	122	120	121
7.	2	95	101	99	100
8.	2	80	86	84	85
9.	2	96	102	100	101
10.	2	107	113	111	112

We have four test scores from various students in various schools. Our corresponding model might be



The `school1` inside double circles in the figure tells us, "I am a latent variable at the `school` level—meaning that I am constant within school and vary across schools—and I correspond to the latent variable named `M1`." That is, double circles denote a latent variable named `M#`, where `#` is the subscript of the variable name inside the double circles. Meanwhile, the variable name inside the double circles specifies the level of the `M#` variable.

The equivalent command-language form for this model is

```
(x1 x2 x3 x4<-X M1[school])
```

The `M1` part of `M1[school]` is the latent variable's name, while the `[school]` part means "at the `school` level". Thus `M1[school]` denotes latent variable `M1`, which varies at the `school` level or, equivalently, which is constant within `school`.

By the way, we gave the latent variable the name `M1` just to match what the Builder does by default. In real life, we would probably give it a different name, such as `S`.

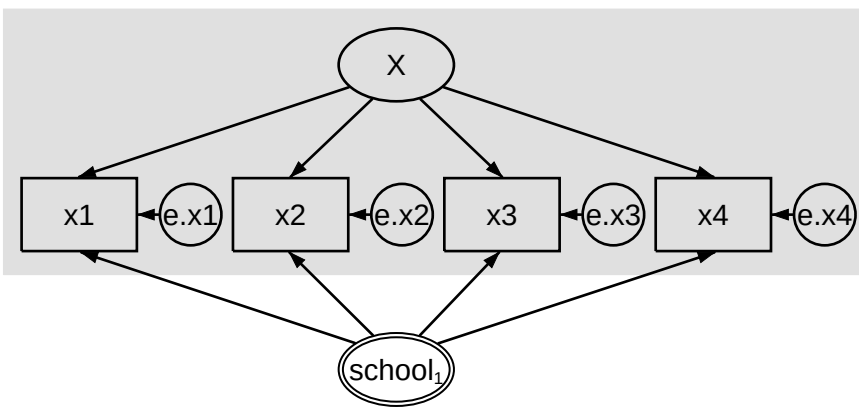
The mathematical way of writing this model is

$$x_1 = \alpha_1 + \beta_1 X + \gamma_1 M_{1,S} + e.x_1$$
$$x_2 = \alpha_2 + \beta_2 X + \gamma_2 M_{1,S} + e.x_2$$
$$x_3 = \alpha_3 + \beta_3 X + \gamma_3 M_{1,S} + e.x_3$$
$$x_4 = \alpha_4 + \beta_4 X + \gamma_4 M_{1,S} + e.x_4$$

where $S =$ school number.

Thus we have three different ways of referring to the same thing: `school1` inside double circles in the path diagram corresponds to `M1[school]` in the command language, which in turn corresponds to $M_{1,S}$ in the mathematical notation.

Rabe-Hesketh, Skrondal, and Pickles (2004) use boxes to identify different levels of the model. Our path diagrams are similar to theirs, but they do not use double circles for multilevel latent variables. We can put a box around the individual-level part of the model, producing something that looks like this:



You can do that in the Builder, but the box has no special meaning to `gsem`; however, adding the box does make the diagram easier to understand in presentations.

However you diagram the model, this model is known as a two-level model. The first level is the student or observational level, and the second level is the school level.

Specifying generalized SEMs: Multilevel mixed effects (3 levels)

A three-level model adds another latent variable at yet another level. For instance, perhaps we have data on four test scores for individual students, who are nested within school, which are nested within counties. The beginning of our data might look like this:

```
. list in 1/10
```

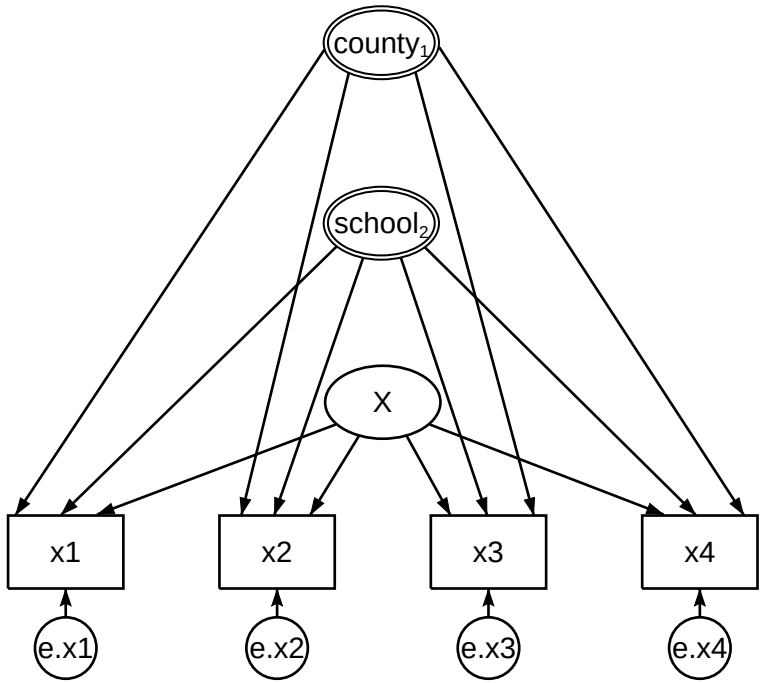
	county	school	x1	x2	x3	x4
1.	1	1	103	122	117	120
2.	1	1	88	107	102	105
3.	1	1	106	125	120	123
4.	1	1	108	127	122	125
5.	1	1	100	119	114	117
6.	1	2	90	99	97	98
7.	1	2	87	96	94	95
8.	1	2	121	130	128	129
9.	1	2	100	109	107	108
10.	1	2	88	97	95	96

Further down, the data might look like this:

```
. list in 991/1l
```

	county	school	x1	x2	x3	x4
991.	2	1	100	96	97	97
992.	2	1	90	86	87	87
993.	2	1	99	95	96	96
994.	2	1	68	64	65	65
995.	2	1	79	75	76	76
996.	2	2	94	100	98	99
997.	2	2	104	110	108	109
998.	2	2	90	96	94	95
999.	2	2	105	111	109	110
1000.	2	2	99	105	103	104

We might diagram the student-within-school-within-county model as



Now we have two variables in double circles: `county1` and `school2`.

`county1` tells us, "I am a latent variable at the `county` level—meaning that I am constant within county and vary across counties—and I correspond to the latent variable named `M1`."

`school2` tells us, "I am a latent variable at the `school` level—meaning that I am constant within school and vary across schools—and I correspond to the latent variable named `M2`."

Do not read anything into the fact that the latent variables are named `M1` and `M2` rather than `M2` and `M1`. When we diagrammed the figure in the Builder, we diagrammed `county` first and then we added `school`. Had we diagrammed them the other way around, the latent variable names would have been reversed. We can edit the names after the fact, but we seldom bother.

The equivalent command-language form for this three-level model is

```
(x1 x2 x3 x4<-X M1[county] M2[county>school])
```

or

```
(x1 x2 x3 x4<-X M1[county] M2[school<county])
```

The `[school<county]` part is usually spoken aloud as "school within county", and `[county>school]` is spoken aloud as "county containing school". When we write `[county>school]` or `[school<county]`, we are saying that the counties contain schools or, equivalently, that school is nested within county. Thus the model we are specifying is in fact a three-level nested model.

The order in which we specify the level effects is irrelevant; we could just as well type

```
(x1 x2 x3 x4<-X M2[county>school] M1[county])
```

or

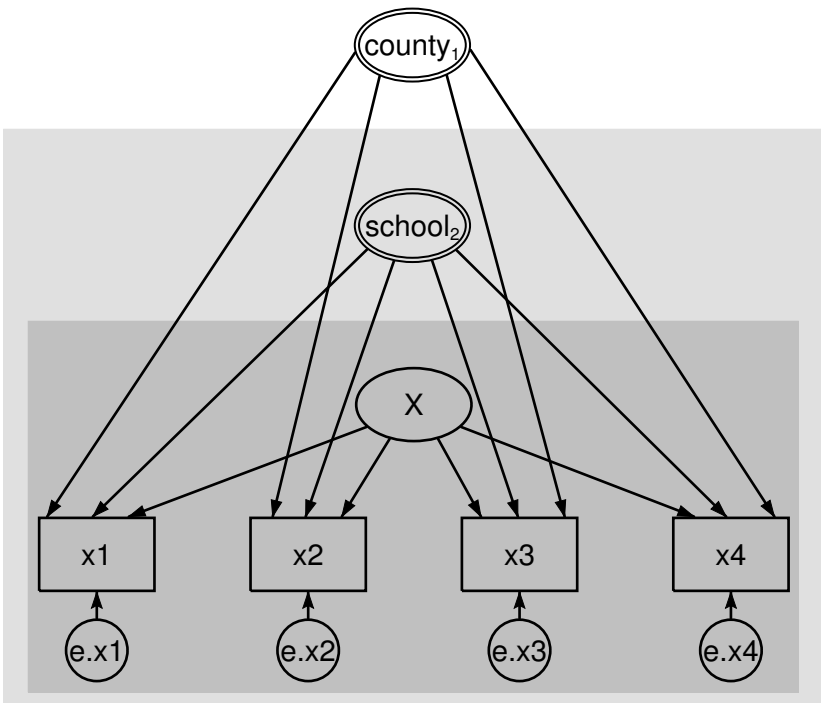
```
(x1 x2 x3 x4 <- X M2[school<county] M1[county])
```

The mathematical way of writing this model is

$$x_1 = \alpha_1 + \beta_1 X + \gamma_1 M_{1,C} + \delta_1 M_{2,S} + e.x_1$$
$$x_2 = \alpha_2 + \beta_2 X + \gamma_2 M_{1,C} + \delta_2 M_{2,S} + e.x_2$$
$$x_3 = \alpha_3 + \beta_3 X + \gamma_3 M_{1,C} + \delta_3 M_{2,S} + e.x_3$$
$$x_4 = \alpha_4 + \beta_4 X + \gamma_4 M_{1,C} + \delta_4 M_{2,S} + e.x_4$$

where $C =$ county number and $S =$ school number.

Three-level nested models are often double-boxed (or double-shaded) in presentation:



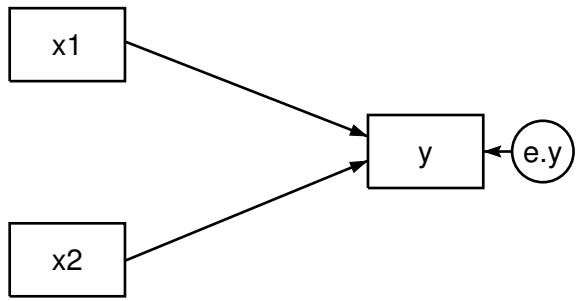
The darker box highlights the first level (student, in our case), and the lighter box highlights the second level (school, in our case). As we previously mentioned, you can add these boxes using the Builder, but they are purely aesthetic and have no meaning to `gsem`.

Specifying generalized SEMs: Multilevel mixed effects (4+ levels)

We have now diagrammed one-, two-, and three-level nested models. You can draw with the Builder and fit with `gsem` higher-level models, but you will usually need lots of data to be successful in fitting those models and, even so, you may run into other estimation problems.

Specifying generalized SEMs: Multilevel mixed effects with random intercepts

Let's change gears and consider with the following simple model a linear regression of y on x_1 and x_2:



In the command language, this model is specified as

```
(x1 x2->y)
```

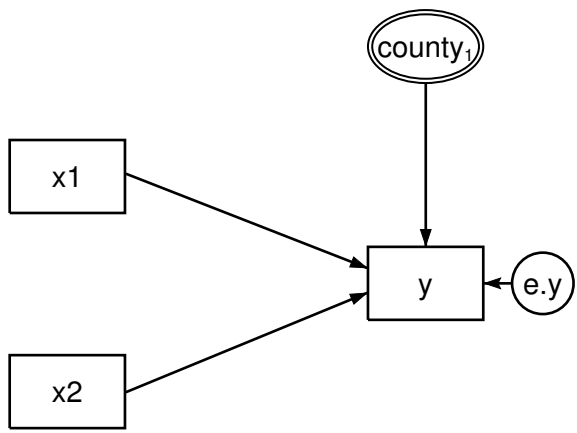
or

```
(y<-x1 x2)
```

and the mathematical representation is

$$y = \alpha + \beta x_1 + \gamma x_2 + e.y$$

Now assume that we have data not just on y, x_1, and x_2, but also on county of residence. If we wanted to add a random intercept—a random effect—for county, the diagram becomes



The command-language equivalent is

```
(x1 x2 M1[county]->y)
```

or

```
(y<-x1 x2 M1[county])
```

and the mathematical representation is

$$y = \alpha + \beta x_1 + \gamma x_2 + M_{1,C} + e.y$$

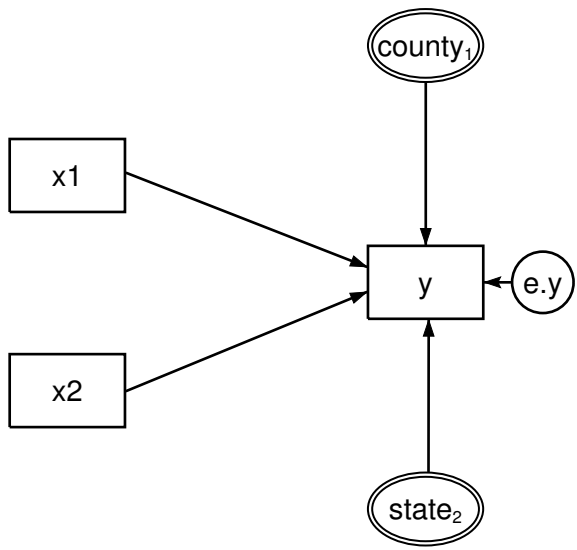
where $C =$ county number. Actually, the model is

$$y = \alpha + \beta x_1 + \gamma x_2 + \delta M_{1,C} + e.y$$

but δ is automatically constrained to be 1 by `gsem`. The software is not reading our mind; consider a solution for $M_{1,C}$ with $\delta = 0.5$. Then another equivalent solution is $M_{1,C}/2$ with $\delta = 1$, and another is $M_{1,C}/4$ with $\delta = 2$, and on and on, because in all cases, $\delta M_{1,C}$ will equal the same value. The fact is that δ is unidentified. Whenever a latent variable is unidentified because of such scaling considerations, `gsem` (and `sem`) automatically set the coefficient to 1 and thus set the scale.

This is a two-level model: the first level is the observational level and the second level is county.

Just as we demonstrated previously with the measurement model, we could have a three-level nested model. We could imagine the observational level nested within the county level nested within the state level. The path diagram would be



The command-language equivalent is

```
(y<-x1 x2 M1[county<state] M2[state])
```

and the mathematical representation is

$$y = \alpha + \beta x_1 + \gamma x_2 + M_{1,C} + M_{2,S} + e.y$$

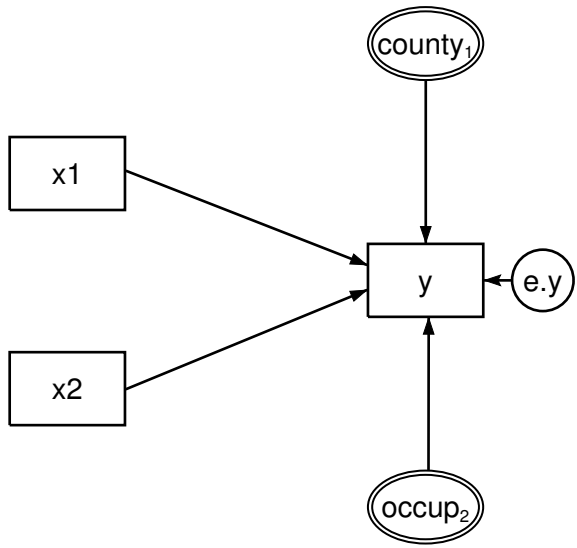
where $C =$ county number and $S =$ state number. Just as previously, the actual form of the equation is

$$y = \alpha + \beta x_1 + \gamma x_2 + \delta M_{1,C} + \zeta M_{2,S} + e.y$$

but the constraints $\delta = \zeta = 1$ are automatically applied by the software.

You can specify higher-level models, but just as we mentioned when discussing the higher-level measurement models, you will need lots of data to fit them successfully and you still may run into other estimation problems.

You can also fit crossed models, such as county and occupation. Unlike county and state, where a particular county appears only in one state, the same occupation will appear in more than one county, which is to say, occupation and county are crossed, not nested. Except for a change in the names of the variables, the path diagram for this model looks identical to the diagram for the state and county-within-state model:



When we enter the model into the Builder, however, we will specify that the two effects are crossed rather than nested.

The command-language way of expressing this crossed model is

```
(y<-x1 x2 M1[county] M2[occupation])
```

and the mathematical representation is

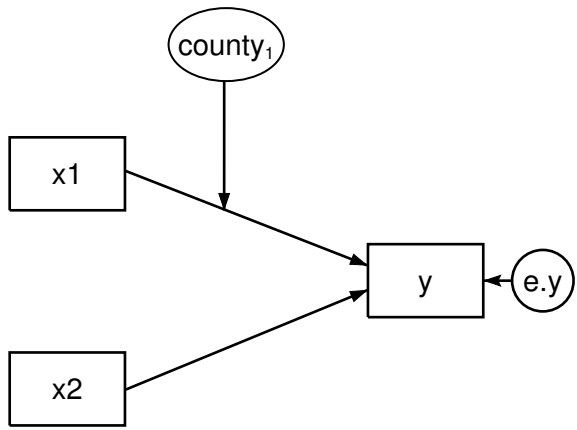
$$y = \alpha + \beta x_1 + \gamma x_2 + M_{1,C} + M_{2,O} + e.y$$

where C = county number and O = occupation number.

Higher-order crossed models are theoretically possible, and you will find `gsem` game to try to fit them. However, you are unlikely to be successful unless you have lots of data so that all the effects can be identified. In fact, you can specify crossed models anywhere you can specify nested models, but the same comment applies.

Specifying generalized SEMs: Multilevel mixed effects with random slopes

Perhaps we feel that county does not affect the intercept, but it affects the slope of `x1`. In that case, we just shift the path from `county1` to instead point to the path between y and x_1.



The command-language equivalent for this model is

```
(y<-x1 c.x1#M1[county]) (y<-x2)
```

or

```
(y<-x1 c.x1#M1[county] x2)
```

To include a random slope (coefficient) on a variable in the command language, include the variable on which you want the random slope just as you ordinarily would:

```
(y<-x1
```

Then, before closing the parenthesis, type

`c.`*variable*`#`*latent_variable*`[`*grouping_variable*`]`

In our case, the *variable* on which we want the random coefficient is `x1`, the *latent_variable* we want to create is named `M1`, and the *grouping_variable* within which the latent variable will be constant is `county`, so we type

```
(y<-x1 c.x1#M1[county]
```

Finally, finish off the command by typing the rest of the model:

```
(y<-x1 c.x1#M1[county] x2)
```

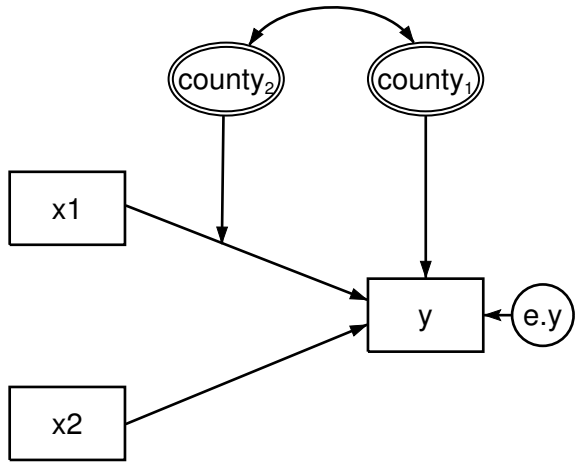
This is another example of Stata's factor-variable notation.

The mathematical representation of the random-slope model is

$$y = \alpha + \beta x_1 + \gamma x_2 + \delta M_{1,C} x_1 + e.y$$

where $C =$ county number and $\delta = 1$.

You can simultaneously specify both random slope and random intercept by putting together what we have already done. The first time you see the combined path diagram, you may think there is a mistake:



County appears twice in two different double circles, once with a subscript of 1 and the other with a subscript of 2! We would gamble at favorable odds that you would have included `county` in double circles once and then would have drawn two paths from it, one to the path between `x1` and `y`, and the other to `y` itself. That model, however, would make an extreme assumption.

Consider what you would be saying. There is one latent variable `M1`. The random intercepts would be equal to `M1`. The random slopes would be equal to γ`M1`, a scalar replica of the random intercepts! You would be constraining the random slopes and intercepts to be correlated 1.

What you usually want, however, is one latent variable for the random slopes and another for the random intercepts. They might be correlated, but they are not related by a multiplicative constant. Thus we also included a covariance between the two random effects.

The command-language equivalent of the correct path diagram (the one shown above) is

```
(y<-x1 x2 c.x1#M2[county] M1[county])
```

You will learn later that the command language assumes covariances exist between latent exogenous variables unless an option is specified. Meanwhile, the Builder assumes those same covariances are 0 unless a curved covariance path is drawn.

The mathematical representation of the model is

$$y = \alpha + \beta x_1 + \gamma x_2 + \delta M_{2,C} x_1 + \zeta M_{1,C} + e.y$$

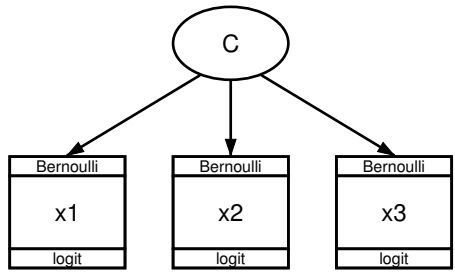
where C = county number and $\delta = \zeta = 1$.

Specifying generalized SEMs: Latent class analysis (LCA)

All the latent variables discussed in the previous sections are continuous latent variables. We can also fit models with categorical latent variables using `gsem`. The unobserved levels of a categorical latent variable are known as latent classes. The classes represent groups in a population such as groups of consumers with different buying preferences. In this manual, we refer to models with categorical latent variables as latent class models, and we refer to an analysis involving categorical latent variables as a latent class analysis (LCA).

Latent class models have distinct parameters for each class. We also estimate the probability of being in each class.

Let's consider a simple latent class model with a categorical latent variable `C` that has two classes and with three observed binary variables, `x1`, `x2`, and `x3`, that are modeled using logistic regression. Some textbooks draw a path diagram for this model as a measurement model. For instance, you may see something like



When we fit this model, we will actually estimate one intercept for `x1` in class 1 and one intercept for `x1` in class 2. Likewise, we will estimate class-specific intercepts for `x2` and for `x3`. Mathematically, the logistic equations for class 1 are

$$\Pr(x_1 = 1 \,|\, C = 1) = \frac{\exp(\alpha_{11})}{1 + \exp(\alpha_{11})}$$

$$\Pr(x_2 = 1 \,|\, C = 1) = \frac{\exp(\alpha_{21})}{1 + \exp(\alpha_{21})}$$

$$\Pr(x_3 = 1 \,|\, C = 1) = \frac{\exp(\alpha_{31})}{1 + \exp(\alpha_{31})}$$

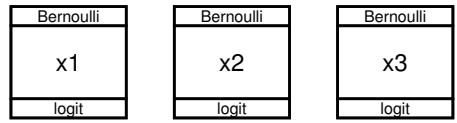
and the equations for class 2 are

$$\Pr(x_1 = 1 \,|\, C = 2) = \frac{\exp(\alpha_{12})}{1 + \exp(\alpha_{12})}$$

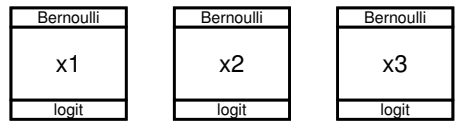
$$\Pr(x_2 = 1 \,|\, C = 2) = \frac{\exp(\alpha_{22})}{1 + \exp(\alpha_{22})}$$

$$\Pr(x_3 = 1 \,|\, C = 2) = \frac{\exp(\alpha_{32})}{1 + \exp(\alpha_{32})}$$

In a sense, `C` has an effect on `x1`, `x2`, and `x3`, but we do not estimate any coefficients on `C`. Rather, the two classes of `C` partition our model into two component models, each with the same `x` variables but with different coefficients (intercepts) on those `x` variables. In Stata's path diagrams, arrows always represent coefficients, so the path diagram above does not actually correspond with the model we fit. A more accurate description would be to draw the diagram as



for the first class and again as



for the second class. Now, we could place the six estimated intercepts in the six boxes.

However, these two diagrams do not fully describe the model. We also estimate the probability of being in each class using multinomial logistic regression,

$$\Pr(C = 1) = \frac{e^{\gamma_1}}{e^{\gamma_1} + e^{\gamma_2}}$$
$$\Pr(C = 2) = \frac{e^{\gamma_2}}{e^{\gamma_1} + e^{\gamma_2}}$$

where γ_1 and γ_2 are intercepts in the multinomial logit model. By default, the first class will be treated as the base, so $\gamma_1 = 0$. We do not have a place for the γ parameter estimates on our diagrams above. Thus, we cannot draw path diagrams in the Builder to fit latent class models. Instead, we fit latent class models using the command language.

We specify this model using the command language as

```
(x1 x2 x3 <- ), logit lclass(C 2)
```

`lclass(C 2)` says that our categorical latent variable is named `C` and has two classes. Because we are only estimating an intercept in each equation, we do not include any variables on the right side of the arrow. We could directly specify that a constant, `_cons`, is included by typing

```
(x1 x2 x3 <- _cons), logit lclass(C 2)
```

We could also be more explicit about having separate models for the two classes by including `1:` at the beginning of the path specification for class 1 and including `2:` at the beginning of the path specification for class 2.

```
(1: x1 x2 x3 <- _cons) (2: x1 x2 x3 <- _cons), logit lclass(C 2)
```

We could even write a separate path specification for each class and outcome variable combination.

```
(1: x1 <- _cons) (1: x2 <- _cons) (1: x3 <- _cons)
(2: x1 <- _cons) (2: x2 <- _cons) (2: x3 <- _cons),
logit lclass(C 2)
```

In any case, the model is the same.

We can build on this to fit different models for different classes and for different outcome variables. For instance, if x3 is ordinal, we could use an ordinal logit model for this variable by typing

```
(1: x1 <- _cons, logit) (1: x2 <- _cons, logit)
(1: x3 <- _cons, ologit) (2: x1 <- _cons, logit)
(2: x2 <- _cons, logit) (2: x3 <- _cons, ologit),
lclass(C 2)
```

Or we can add a constraint that the intercepts for x1 and x2 are equal in class 2.

```
(1: x1 <- _cons) (1: x2 <- _cons) (1: x3 <- _cons)
(2: x1 <- _cons@c) (2: x2 <- _cons@c) (2: x3 <- _cons),
logit lclass(C 2)
```

Other than including continuous latent variables, all command-language features of `gsem` discussed above can be used to modify and extend this latent class model.

Specifying generalized SEMs: Latent class analysis, class predictors

We can include variables in a latent class model that predict class membership. Extending the model above, we simply add exogenous variables to the multinomial logit model for `C`. If we believe that `z` predicts class membership, our multinomial logit model becomes

$$\Pr(C=1) = \frac{e^{\gamma_1+z\beta_1}}{e^{\gamma_1+z\beta_1}+e^{\gamma_2+z\beta_2}}$$
$$\Pr(C=2) = \frac{e^{\gamma_2+z\beta_2}}{e^{\gamma_1+z\beta_1}+e^{\gamma_2+z\beta_2}}$$

where $\gamma_1 = 0$ and $\beta_1 = 0$ when `C` $= 1$ is the base class.

To fit this model using the command language, we type

```
(x1 x2 x3 <- _cons, logit) (C <- z), lclass(C 2)
```

We can also allow different predictors for different classes. For instance, if `C` has three levels, we can type

```
(x1 x2 x3 <- _cons, logit)
(1.C <- z1 z2 z3)
(2.C <- z1 z2)
(3.C <- z1 z2 z3),
lclass(C 3)
```

where `z1`, `z2`, and `z3` are predictors of class membership, but only `z1` and `z2` are predictors of `C` $= 2$.

Specifying generalized SEMs: Latent class analysis, two latent variables

Latent class models can include more than one categorical latent variable. Let's consider an example where `C` has two classes and `D` has three classes.

We could extend our model from *Specifying generalized SEMs: Latent class analysis (LCA)* to include `D` by typing

```
(x1 x2 x3 <- _cons), logit lclass(C 2) lclass(D 3)
```

Now, if we want to refer to a specific class, we use factor-variable notation before the colon. The most verbose specification of this model is

```
(1.C#1.D: x1 <- _cons) (1.C#1.D: x2 <- _cons) (1.C#1.D: x3 <- _cons)
(2.C#1.D: x1 <- _cons) (2.C#1.D: x2 <- _cons) (2.C#1.D: x3 <- _cons)
(1.C#2.D: x1 <- _cons) (1.C#2.D: x2 <- _cons) (1.C#2.D: x3 <- _cons)
(2.C#2.D: x1 <- _cons) (2.C#2.D: x2 <- _cons) (2.C#2.D: x3 <- _cons)
(1.C#3.D: x1 <- _cons) (1.C#3.D: x2 <- _cons) (1.C#3.D: x3 <- _cons)
(2.C#3.D: x1 <- _cons) (2.C#3.D: x2 <- _cons) (2.C#3.D: x3 <- _cons),
logit lclass(C 2) lclass(D 3)
```

As before, this allows us to specify different models for different outcomes or for different classes. It also allows us to specify constraints. For instance, if we want equal intercepts for `x3` when `C` = 2 and `D` = 2 and when `C` = 2 and `D` = 3, we type

```
(1.C#1.D: x1 <- _cons) (1.C#1.D: x2 <- _cons) (1.C#1.D: x3 <- _cons)
(2.C#1.D: x1 <- _cons) (2.C#1.D: x2 <- _cons) (2.C#1.D: x3 <- _cons)
(1.C#2.D: x1 <- _cons) (1.C#2.D: x2 <- _cons) (1.C#2.D: x3 <- _cons)
(2.C#2.D: x1 <- _cons) (2.C#2.D: x2 <- _cons) (2.C#2.D: x3 <- _cons@a)
(1.C#3.D: x1 <- _cons) (1.C#3.D: x2 <- _cons) (1.C#3.D: x3 <- _cons)
(2.C#3.D: x1 <- _cons) (2.C#3.D: x2 <- _cons) (2.C#3.D: x3 <- _cons@a),
logit lclass(C 2) lclass(D 3)
```

We can allow predictors of class membership just as we did with a single categorical latent variable. For specifying `z` as a predictor of the classes of C, we type

```
(x1 x2 x3 <- _cons, logit) (C <- z), lclass(C 2) lclass(D 3)
```

We can also add predictors of specific cells in the interaction between `C` and `D`. For instance,

```
(x1 x2 x3 <- _cons, logit) (1.C#3.D <- z), lclass(C 2) lclass(D 3)
```

Adding predictors or even an intercept (by specifying `_cons`) to an individual cell induces correlation between the categorical latent variables `C` and `D`.

And now you, too, are an expert on Stata's path diagrams and command language for SEM.

Reference

Rabe-Hesketh, S., A. Skrondal, and A. Pickles. 2004. Generalized multilevel structural equation modeling. *Psychometrika* 69: 167–190.

Also see

[SEM] **intro 1** — Introduction

[SEM] **intro 3** — Learning the language: Factor-variable notation (gsem only)

[SEM] **Builder** — SEM Builder

[SEM] **Builder, generalized** — SEM Builder for generalized models

[SEM] **sem and gsem path notation** — Command syntax for path diagrams