Title

stata.com

intro 4 — Substantive concepts

Description Remarks and examples References Also see

Description

The structural equation modeling way of describing models is deceptively simple. It is deceptive because the machinery underlying structural equation modeling is sophisticated, complex, and sometimes temperamental, and it can be temperamental both in substantive statistical ways and in practical computer ways.

Professional researchers need to understand these issues.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Differences in assumptions between sem and gsem

sem fits standard linear SEMs.

gsem fits generalized SEMs, by which we mean

1. gsem fits not just linear but generalized linear models,
2. gsem fits multilevel mixed models,
3. gsem fits models with categorical latent variables,
4. gsem fits items 1 and 2 combined, and
5. gsem fits items 1 and 3 combined.

There is a difference in assumptions between standard linear SEMs and generalized SEMs.

Standard linear SEMs generally assume that the observed endogenous variables, the observed exogenous variables, the latent endogenous variables, and the latent exogenous variables are jointly distributed normally with mean μ and variance matrix $\mathbf{\Sigma}$. In this formulation, we include the $e.$ error variables among the latent exogenous variables.

Generalized SEMs drop the observed variables and categorical latent variables from the joint-normality assumption. Generalized SEMs treat the observed exogenous variables as given and produce estimates conditional on their values. This seems a minor difference, but for some researchers it has important implications.

Consider a researcher wishing to include a subject's age and age-squared among the observed exogenous variables of a model. Variables age and age-squared simply cannot be jointly normally distributed. Including age and age-squared violates the assumptions of standard linear SEMs, at least officially. You may now suspect that such researchers—perhaps including yourself—should use gsem and not sem. Because standard linear SEMs are a subset of generalized SEMs, that should not be much of an inconvenience.

It is interesting to note, however, that sem and gsem produce the same numeric solutions for the parameters and the standard errors when both can fit the same model.

All of which is to say that although it is typical to assume joint normality of all variables when deriving the standard linear SEM, joint normality is not strictly necessary. The lesser assumption of joint normality conditional on the observed exogenous variables is sufficient. Even the normality assumption can be relaxed and replaced with i.i.d., and even the i.i.d. assumption can be relaxed.

We will discuss these issues below.

Even so, some structural equation modeling statistics are dependent on the full joint normality assumption. Many goodness-of-fit tests fall into that category. Throughout this manual, we will be explicit about when the full joint-normality assumption is required.

gsem never requires the full joint-normality assumption. Standard linear SEMs have a history that includes the full joint-normality assumption; generalized SEMs have no such history. At the end of the day, both can be considered M estimation.

sem: Choice of estimation method

sem provides four estimation methods: maximum likelihood (ML; the default), quasimaximum likelihood (QML), asymptotic distribution free (ADF), and maximum likelihood with missing values (MLMV).

Strictly speaking, the assumptions one must make to establish the consistency of the estimates and their asymptotic normality is determined by the method used to estimate them. We want to give you advice on when to use each.

1. ML is the method that `sem` uses by default. In `sem`, the function being maximized formally assumes the full joint normality of all the variables, including the observed variables. But the full joint-normality assumption can be relaxed, and the substitute conditional-on-the-observed-exogenous-variables is sufficient to justify all reported estimates and statistics except the log-likelihood value and the model-versus-saturated χ^2 test.

   Relaxing the constraint that latent variables outside of the error variables are not normally distributed is more questionable. In the measurement model `(X->x1 x2 x3 x4)`, simulations with the violently nonnormal $X \sim \chi^2(2)$ produced good results except for the standard error of the estimated variance of X. Note that it was not the coefficient on X that was estimated poorly, it was not the coefficient's standard error, and it was not even the variance of X that was estimated poorly. It was the standard error of the variance of X. Even so, there are no guarantees.

   `sem` uses method ML when you specify `method(ml)` or when you omit the `method()` option altogether.

2. QML uses ML to fit the model parameters but relaxes the normality assumptions when estimating the standard errors. QML handles nonnormality by adjusting standard errors.

   Concerning the parameter estimates, everything just said about ML applies to QML because those estimates are produced by ML.

   Concerning standard errors, we theoretically expect consistent standard errors, and we practically observe that in our simulations. In the measurement model with $X \sim \chi^2(2)$, we even obtained good standard errors of the estimated variance of X. QML does not really fix the problem of nonnormality of latent variables, but it does tend to do a better job.

   `sem` uses method QML when you specify `method(ml) vce(robust)` or, because `method(ml)` is the default, when you specify just `vce(robust)`.

   When you specify `method(ml) vce(sbentler)` or just `vce(sbentler)`, the Satorra–Bentler scaled χ^2 test is reported. When observed data are nonnormal, the standard model-versus-saturated test statistic does not follow a χ^2 distribution. The Satorra–Bentler scaled χ^2 statistic uses a function of fourth-order moments to adjust the standard goodness-of-fit statistic so that it has a mean that more closely follows the reference χ^2 distribution. The corresponding robust standard errors, which are adjusted using a function of fourth-order moments, are reported as well. For details, see Satorra and Bentler (1994).

3. ADF makes no assumption of joint normality or even symmetry, whether for observed or latent variables. Whereas QML handles nonnormality by adjusting standard errors and not point estimates, ADF produces justifiable point estimates and standard errors under nonnormality.

   For many researchers, this is most important for relaxing the assumption of normality of the errors, and because of that, ADF is sometimes described that way. ADF in fact relaxes the normality assumption for all latent variables.

   Along the same lines, it is sometimes difficult to be certain exactly which normality assumptions are being relaxed when reading other sources. It sometimes seems that ADF uniquely relaxes the assumption of the normality of the observed variables, but that is not true. Other methods, even ML, can handle that problem.

   ADF is a form of weighted least squares (WLS). ADF is also a generalized method of moments (GMM) estimator. In simulations of the measurement model with $X \sim \chi^2(2)$, ADF produces

excellent results, even for the standard error of the variance of X. Be aware, however, that ADF is less efficient than ML when latent variables can be assumed to be normally distributed. If latent variables (including errors) are not normally distributed, on the other hand, ADF will produce more efficient estimates than ML or QML.

`sem` uses method ADF when you specify `method(adf)`.

4. MLMV aims to retrieve as much information as possible from observations containing missing values.

   In this regard, `sem` methods ML, QML, and ADF do a poor job. They are known as listwise deleters. If variable `x1` appears someplace in the model and if `x1` contains a missing value in observation 10, then observation 10 simply will not be used. This is true whether `x1` is endogenous or exogenous and even if `x1` appears in some equations but not in others.

   Method MLMV, on the other hand, is not a deleter at all. Observation 10 will be used in making all calculations.

   For method MLMV to perform what might seem like magic, joint normality of all variables is assumed and missing values are assumed to be missing at random (MAR). MAR means either that the missing values are scattered completely at random throughout the data or that values more likely to be missing than others can be predicted by the variables in the model.

   Method MLMV formally requires the assumption of joint normality of all variables, both observed and latent. If your observed variables do not follow a joint normal distribution, you may be better off using ML, QML, or ADF and simply omitting observations with missing values.

   `sem` uses method MLMV when you specify `method(mlmv)`. See [SEM] **example 26**.

gsem: Choice of estimation method

`gsem` provides only two estimation methods: maximum likelihood (ML; the default) and quasi-maximum likelihood (QML).

1. ML is the method that `gsem` uses by default. This is the same ML used by `sem` but applied to a different likelihood function. The `sem` likelihood function assumes and includes joint normality of all variables. The `gsem` likelihood function assumes only conditional normality.

   Because likelihood functions are different, the likelihood values reported by `sem` and `gsem` are not comparable except in the case where there are no observed exogenous variables.

   ML formally assumes conditional normality, and thus continuous latent variables are still assumed to be normally distributed. What we said in the `sem` case about relaxing the assumption of normality of the latent variables applies equally in the `gsem` case.

   `gsem` uses method ML when you specify `method(ml)` or when you omit the `method()` option altogether.

2. QML uses ML to fit the model but relaxes the conditional normality assumptions when estimating the standard errors. QML handles nonnormality by adjusting standard errors.

   Everything said about `sem`'s QML applies equally to `gsem`'s QML.

   `gsem` uses method QML when you specify `vce(robust)`.

Because the choice of method often affects convergence with `sem`, in the `gsem` case there is a tendency to confuse choice of integration method with maximization method. However, there are no issues related to assumptions about integration method; choice of integration method is purely a mechanical issue. This is discussed in [SEM] **intro 12**.

Treatment of missing values

sem, sem with method MLMV, and gsem treat missing values differently.

1. sem by default is a listwise deleter.

   If variable x1 appears in the model and if x1 contains missing in observation 10, then observation 10 will not be used. This is true whether x1 is endogenous or exogenous and even if x1 appears in some equations but not in others.

2. sem with method MLMV is not a deleter at all; it uses all observations.

   If variable x1 appears in the model and if x1 contains missing in observation 10, then observation 10 will still be used. Doing this formally requires assuming the joint normality of all observed variables and was discussed in item 4 of *sem: Choice of estimation method*.

3. gsem by default is an equationwise deleter for models without categorical latent variables.

   The abridged meaning is that gsem will often be able to use more observations from the data than sem will, assuming you do not use sem with method MLMV.

   The full meaning requires some setup. Consider a model of at least five equations. Assume that observed exogenous variable x1 appears in the first equation but not in equations 2–4; that equation 1 predicts y1; that y1 appears as a predictor in equations 2–4; and that x1 contains missing in observation 10.

   If endogenous variable y1 is latent or observed and of family Gaussian, link identity, but without censoring, then

   3.1 Observation 10 will be ignored in making calculations related to equation 1.

   3.2 Observation 10 will also be ignored in making calculations related to equations 2–4 because y1, a function of x1, appears in them.

   3.3 The calculations for the other equation(s) will include observation 10.

   Alternatively, if y1 is observed and not family Gaussian, link identity, or has censoring, then item 3.2 changes:

   3.2 Observation 10 will be used in making calculations related to equations 2–4 even though y1, a function of x1, appears in them.

   As we said at the outset, the result of all of this is that gsem often uses more observations than does sem (excluding method MLMV).

4. gsem has an option, listwise, that duplicates the sem rules. This is used in testing of Stata. There is no reason you would want to specify the option.

5. gsem uses listwise deletion for models with categorical latent variables.

Variable types: Observed, latent, endogenous, exogenous, and error

Structural equation models can contain four different types of variables:

1. observed exogenous
2. latent exogenous
3. observed endogenous
4. latent endogenous

As a software matter, it is useful to think as though there is a fifth type, too:

5. error

Errors are in fact a special case of latent exogenous variables, but there will be good reason to consider them separately.

As a language matter, it is sometimes useful to think of there being yet another type of variable, namely,

6. measure or measurement

Measurement variables are a special case of observed endogenous variables.

Let us explain:

Observed.
A variable is observed if it is a variable in your dataset. In this documentation, we often refer to observed variables with `x1`, `x2`, ..., `y1`, `y2`, and so on, but in reality observed variables have names such as `mpg`, `weight`, `testscore`, and so on.

Latent.
A variable is latent if it is not observed. A variable is latent if it is not in your dataset but you wish it were. You wish you had a variable recording the propensity to commit violent crime, or socioeconomic status, or happiness, or true ability, or even income. Sometimes, latent variables are imagined variants of real variables, variables that are somehow better, such as being measured without error. At the other end of the spectrum are latent variables that are not even conceptually measurable.

In this documentation, latent variables usually have names such as `L1`, `L2`, `F1`, ..., but in real life the names are more descriptive such as `VerbalAbility`, `SES`, and so on. The `sem` and `gsem` commands assume that variables are latent if the first letter of the name is capitalized, so we will always capitalize our latent variable names.

Endogenous.
A variable is endogenous (determined within the system) if any path points to it.

Exogenous.
A variable is exogenous (determined outside the system) if paths only originate from it or, equivalently, no path points to it.

Now that we have the above definitions, we can better understand the five types of variables:

1. *Observed exogenous.*
   A variable in your dataset that is treated as exogenous in your model.
2. *Latent exogenous.*
   An unobserved variable that is treated as exogenous in your model.
3. *Observed endogenous.*
   A variable in your dataset that is treated as endogenous in your model.
4. *Latent endogenous.*
   An unobserved variable that is treated as endogenous in your model.
5. *Error.*
   Mathematically, error variables are just latent exogenous variables. In `sem` and `gsem`, however, errors are different in that they have defaults different from the other latent exogenous variables.

   Errors are named `e`. So, for example, the error variable associated with observed endogenous variable `y1` has the full name `e.y1`; the error variable associated with latent endogenous variable `L1` has the full name `e.L1`.

In sem, each endogenous variable has a corresponding e. variable.

In gsem, observed endogenous variables associated with family Gaussian have corresponding e. variables, but other observed endogenous variables do not. All continuous latent endogenous variables have an associated e. variable, but that is not a special case because all continuous latent endogenous variables are assumed to be family Gaussian.

In the Builder, when you create an endogenous variable, the variable's corresponding error variable instantly springs into existence. The same happens in the command language, you just do not see it. In addition, error variables automatically and inalterably have their path coefficient constrained to be 1.

The covariances between the different variable types are given in the table below. The latent variables discussed here are continuous latent variables. In the table,

1. **0** means 0 and there are no options or secret tricks to change the value from 0.
2. **i** means as implied by your model and beyond your control.
3. (#) means to see the numbered note below the table.

If an entry in the matrix below is **0** or **i**, then you may not draw curved paths between variables of the specified types.

	Observed exogenous	Latent exogenous	Observed endogenous	Latent endogenous	Error
Observed exogenous	(1) & (2)				
Latent exogenous	(3)	(4) & (5)			
Observed endogenous	**i**	**i**	**i**		
Latent endogenous	**i**	**i**	**i**	**i**	
Error	**0**	**0**	**i**	**i**	(6) & (7)

1. Variances of observed exogenous variables:
   - 1.1 Builder, sem mode: Taken as given, assuming not using method MLMV. Can be estimated or constrained; if so, you are assuming normality of the observed exogenous variables unless using method ADF.
   - 1.2 sem command: Same as item 1.1. Use var() option to estimate or constrain.
   - 1.3 Builder, gsem mode: Taken as given. Cannot be estimated or constrained.
   - 1.4 gsem command: Same as item 1.3.
2. Covariances between observed exogenous variables:
   - 2.1 Builder, sem mode: Taken as given, assuming not using method MLMV. Can be estimated or constrained by drawing curved paths between variables; if so, you are assuming normality of the observed exogenous variables unless using method ADF.
   - 2.2 sem command: Same as item 2.1. Use cov() option to estimate or constrain.
   - 2.3 Builder, gsem mode: Taken as given. Cannot be estimated or constrained. Path may not be drawn.
   - 2.4 gsem command: Same as item 2.3.
3. Covariances between latent exogenous and observed exogenous variables:
   - 3.1 Builder, sem mode: Constrained to be 0 unless a curved path is drawn between variables, in which case it is estimated. Can be constrained.

3.2 `sem` command: Assumed to be nonzero and therefore estimated by default. Can be constrained (even to 0) using `cov()` option.

3.3 Builder, `gsem` mode: Assumed to be nonzero. Cannot be estimated or constrained because this covariance is not among the identified parameters of the generalized SEM.

3.4 `gsem` command: Same as item 3.3.

4. Variances of latent exogenous variables:

4.1 Builder, `sem` mode: Variances are estimated and can be constrained.

4.2 `sem` command: Variances are estimated and can be constrained using `var()` option.

4.3 Builder, `gsem` mode: Almost the same as item 4.1 except variances cannot be constrained to 0.

4.4 `gsem` command: Almost the same as item 4.2 except variances cannot be constrained to 0.

5. Covariances between latent exogenous variables:

5.1 Builder, `sem` mode: Assumed to be 0 unless a curved path is drawn between variables. Path may include constraints.

5.2 `sem` command: Assumed to be nonzero and estimated, the same as if a curved path without a constraint were drawn in the Builder. Can be constrained (even to 0) using `cov()` option.

5.3 Builder, `gsem` mode: Same as item 5.1.

5.4 `gsem` command: Same as item 5.2.

6. Variances of errors:

6.1 Builder, `sem` mode: Estimated. Can be constrained.

6.2 `sem` command: Estimated. Can be constrained using `var()` option.

6.3 Builder, `gsem` mode: Almost the same as item 6.1 except variances cannot be constrained to 0.

6.4 `gsem` command: Almost the same as item 6.2 except variances cannot be constrained to 0.

7. Covariances between errors:

7.1 Builder, `sem` mode: Assumed to be 0. Can be estimated by drawing curved paths between variables. Can be constrained.

7.2 `sem` command: Assumed to be 0. Can be estimated or constrained using `cov()` option.

7.3 Builder, `gsem` mode: Almost the same as item 7.1 except covariances between errors cannot be estimated or constrained if one or both of the error terms correspond to a generalized response with family Gaussian, link log, or link identity with censoring.

7.4 `gsem` command: Almost the same as item 7.2 except covariances between errors cannot be estimated or constrained if one or both of the error terms correspond to a generalized response with family Gaussian, link log, or link identity with censoring.

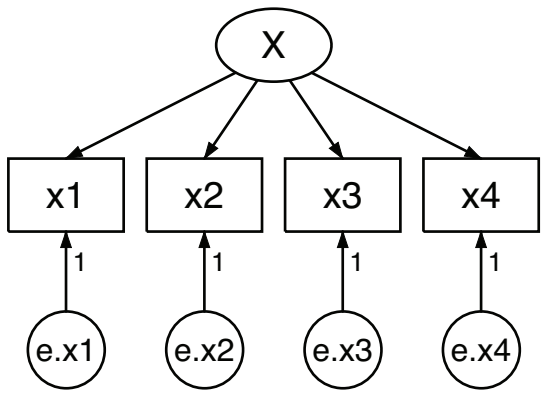
The properties of categorical latent variables differ from those of the continuous latent variables that we have discussed to this point. Categorical latent variables cannot be included in models with other types of latent variables. They do not covary by default. To specify a covariance, rather than using the `cov()` option, you add an intercept or a predictor for a cell of the interaction of the latent variables. For instance, you might type `2.C# 3.D <- _cons`. Categorical latent variables do not covary with any other types of variables in the model. Instead, parameters related to other variables are allowed to vary across the classes of the categorical latent variable.

Finally, there is a sixth variable type that we sometimes find convenient to talk about:

Measure or *measurement*.

A measure variable is an observed endogenous variable with a path from a latent variable. We introduce the word "measure" not as a computer term or even a formal modeling term but as a convenience for communication. It is a lot easier to say that `x1` is a measure of `X` than to say that `x1` is an observed endogenous variable with a path from latent variable `X` and so, in a real sense, `x1` is a measurement of `X`.

In our measurement model,



the variables are

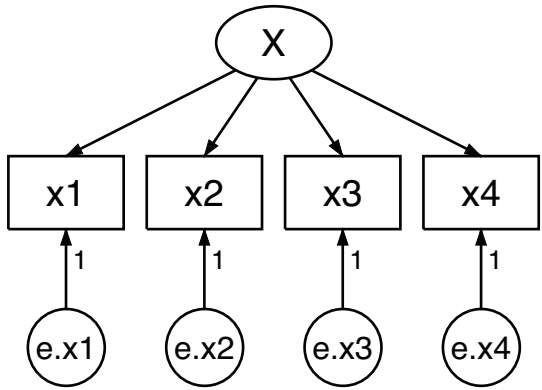
latent exogenous:	`X`
error:	`e.x1`, `e.x2`, `e.x3`, `e.x4`
observed endogenous:	`x1`, `x2`, `x3`, `x4`

All the observed endogenous variables in this model are measures of `X`.

Constraining parameters

Constraining path coefficients to specific values

If you wish to constrain a path coefficient to a specific value, you just write the value next to the path. In our measurement model without correlation of the residuals,



we indicate that the coefficients `e.x1`, ..., `e.x4` are constrained to be 1 by placing a small 1 along the path. We can similarly constrain any path in the model.

If we wanted to constrain $\beta_2 = 1$ in the equation

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

we would write a 1 along the path between X and x_2. If we were instead using `sem`'s or `gsem`'s command language, we would write

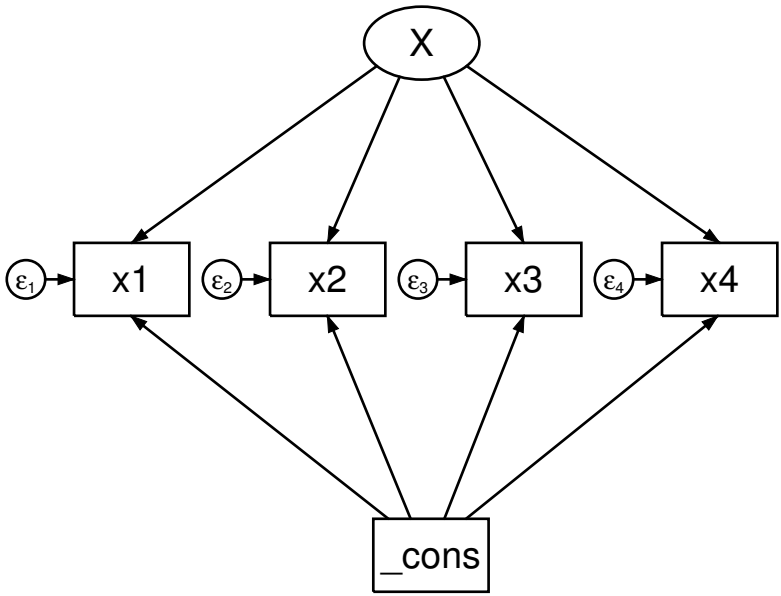
```
(x1<-X) (x2<-X@1) (x3<-X) (x4<-X)
```

That is, you type an `@` symbol immediately after the variable whose coefficient is being constrained, and then you type the value.

Constraining intercepts to specific values (suppressing the intercept)

Constraining path coefficients is common. Constraining intercepts is less so, and usually when the situation arises, you wish to constrain the intercept to 0, which is often called "suppressing the intercept".

Although it is unusual to draw the paths corresponding to intercepts in path diagrams, they are assumed, and you could draw them if you wish. A more explicit version of our path diagram for the measurement model is



The path coefficient of `_cons` to `x1` corresponds to α_1 in

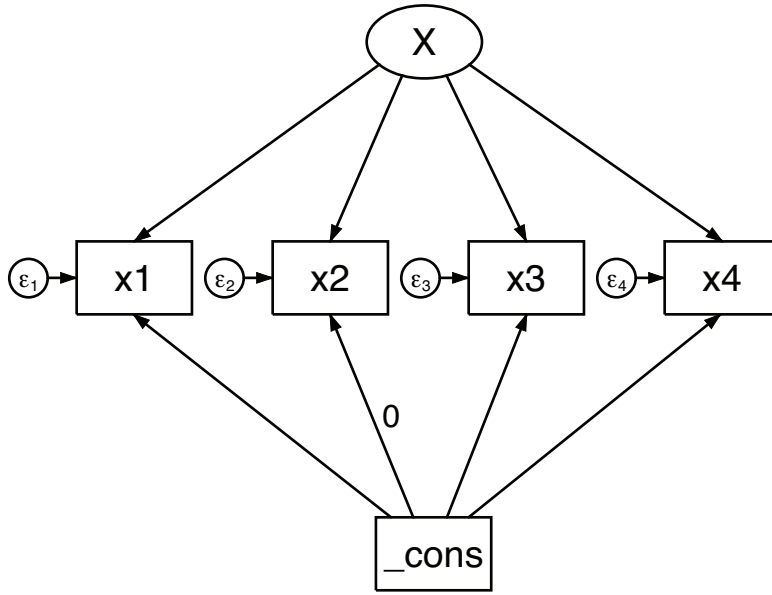
$$x_1 = \alpha_1 + X\beta_1 + e.x_1$$

and the path coefficient of `_cons` to `x2` corresponds to α_2 in

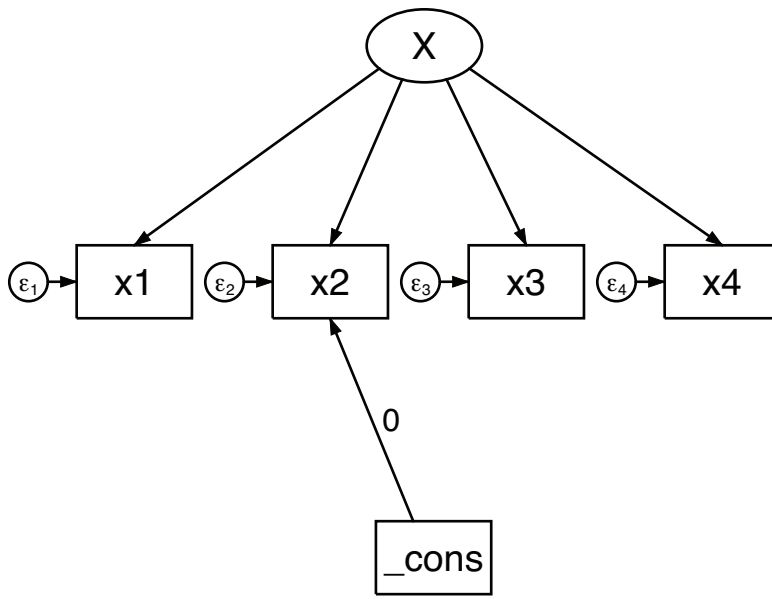
$$x_2 = \alpha_2 + X\beta_2 + e.x_2$$

and so on.

Obviously, if you wanted to constrain a particular intercept to a particular value, you would write the value along the path. To constrain $\alpha_2 = 0$, you could draw



Because intercepts are assumed, you could omit drawing the paths from `_cons` to `x1`, `_cons` to `x3`, and `_cons` to `x4`:



Just as with the Builder, the command language assumes paths from _cons to all endogenous variables, but you could type them if you wished:

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

If you wanted to constrain $\alpha_2 = 0$, you could type

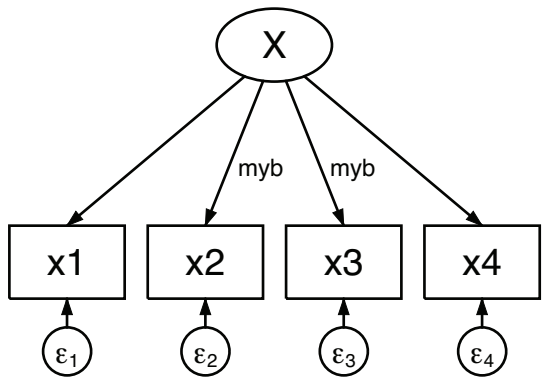
```
(x1<-X _cons) (x2<-X _cons@0) (x3<-X _cons) (x4<-X _cons)
```

or you could type

```
(x1<-X) (x2<-X _cons@0) (x3<-X) (x4<-X)
```

Constraining path coefficients or intercepts to be equal

If you wish to constrain two or more path coefficients to be equal, place a symbolic name along the relevant paths:



In the diagram above, we constrain $\beta_2 = \beta_3$ because we stated that $\beta_2 =$ myb and $\beta_3 =$ myb.

You follow the same approach in the command language:

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

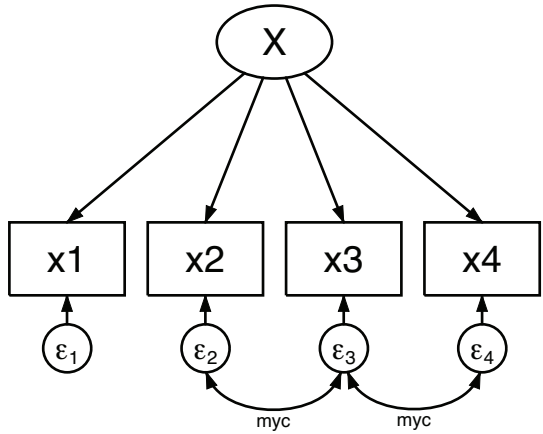
This works the same way with intercepts. Intercepts are just paths from _cons, so to constrain intercepts to be equal, you add symbolic names to their paths. In the command language, you constrain $\alpha_1 = \alpha_2$ by typing

```
(x1<-X _cons@c) (x2<-X _cons@c) (x3<-X) (x4<-X)
```

See [SEM] **example 8**.

Constraining covariances to be equal (or to specific values)

If you wish to constrain covariances, usually you will want to constrain them to be equal instead of to a specific value. If we wanted to fit our measurement model and allow correlation between e.x2 and e.x3 and between e.x3 and e.x4, and we wanted to constrain the covariances to be equal, we could draw



If you instead wanted to constrain the covariances to specific values, you would place the value along the paths in place of the symbolic names.

In the command language, covariances (curved paths) are specified using the `cov()` option. To allow covariances between `e.x2` and `e.x3` and between `e.x3` and `e.x4`, you would type

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

To constrain the covariances to be equal, you would type

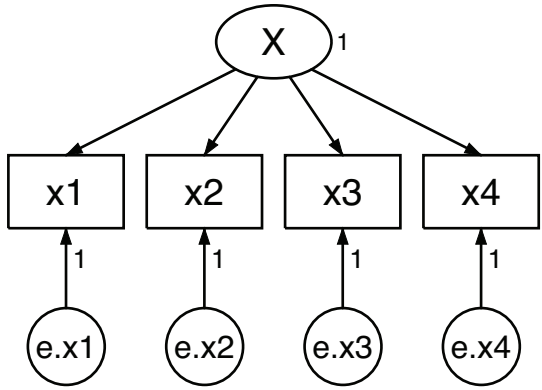
```
(x1<-X) (x2<-X) (x3<-X) (x4<-X), cov(e.x2*e.x3@myc) cov(e.x3*e.x4@myc)
```

Constraining variances to specific values (or to be equal)

Variances are like covariances except that in path diagrams drawn by some authors, variances curve back on themselves. In the Builder, variances appear inside or beside the box or circle. Regardless of how they appear, variances may be constrained to normalize latent variables, although normalization is handled by `sem` and `gsem` automatically (something we will explain in *How sem (gsem) solves the problem for you* under *Identification 2: Normalization constraints (anchoring)* below).

In the Builder, you constrain variances by clicking on the variable and using the lock box to specify the value, which can be a number or a symbol. In the command language, variances are specified using the `var()` option as we will explain below.

Let's assume that you want to normalize the latent variable `X` by constraining its variances to be 1. You do that by drawing



In the command language, we specify this model as

```
(x1<-X) (x2<-X) (x3<-X) (x4<-X), var(X@1)
```

Constraining latent exogenous variables to have unit variance as an identifying restriction may be desirable when you wish simultaneously to constrain their correlations with other latent exogenous variables. `sem` and `gsem` allow you to constrain covariances, not correlations. Covariances are equal to correlations when variances are 1.

In more complicated models, you may wish to constrain variances of latent exogenous variables to be equal. You can do that by specifying a symbolic name.

Identification 1: Substantive issues

Not all models are identified

Just because you can draw the path diagram for a model, write its equations, or write it in Stata's command syntax, does not mean the model is identified. Identification refers to the conceptual constraints on parameters of a model that are required for the model's remaining parameters to have a unique solution. A model is said to be unidentified if these constraints are not supplied. These constraints are of two types: substantive constraints and normalization constraints. We will begin by discussing substantive constraints because that is your responsibility; the software provides normalization constraints automatically.

How to count parameters

If you are fitting a model with `sem` and your model has K observed variables, then your data contain $K(K+1)/2$ second-order moments, and thus p, the number of parameters based on second-order moments that can be estimated, cannot exceed $K(K+1)/2$.

Every path in your model contributes 1 to p unless the parameter is constrained to a specific value, and then it does not contribute at all. If two parameters are constrained to be equal, the two parameters count as one. In counting p, you must remember to count the curved paths from latent variables back to themselves, which is to say, the variances. Just counting the number of parameters can be challenging. And even if $p \leq K(K+1)/2$, your model may not be identified. Identification depends not only on the number of paths but also on their locations.

Counting parameters can be even more difficult in the case of certain generalized linear (gsem) models. For a discussion of this, see Skrondal and Rabe-Hesketh (2004, chap. 5).

Even in the non-gsem case, books have been written on this subject, and we will refer you to them. A few are Bollen (1989), Brown (2015), Kline (2016), and Kenny (1979). We will refer you to them, but do not be surprised if they refer you back to us. Brown (2015, 179) writes, "Because latent variable software programs are capable of evaluating whether a given model is identified, it is often most practical to simply try to estimate the solution and let the computer determine the model's identification status." That is not bad advice.

What happens when models are unidentified

So what happens when you attempt to fit an unidentified model? In some cases, sem (gsem) will tell you that your model is unidentified. If your model is unidentified for subtle substantive reasons, however, you will see

```
initial values not feasible
r(1400);
```

or

```
Iteration 50:   log likelihood = -337504.44  (not concave)
Iteration 51:   log likelihood = -337504.44  (not concave)
Iteration 52:   log likelihood = -337504.44  (not concave)
.
.
.
Iteration 101:   log likelihood = -337504.44  (not concave)
.
.
.
```

In the latter case, sem (gsem) will iterate forever, reporting the same criterion value (such as log likelihood) and saying "not concave" over and over again.

Observing periods of the "not concave" message is not concerning, so do not overreact at the first occurrence. Become concerned when you see "not concave" and the criterion value is not changing, and even then, stay calm for a short time because the value might be changing in digits you are not seeing. If the iteration log continues to report the same value several times, however, press *Break*. Your model is probably not identified.

How to diagnose and fix the problem

You must find and fix the problem.

If you get the "initial values not feasible" message, your goal is to find feasible initial values and thereby either solve the problem or convert it into the infinitely long iteration log problem; see [SEM] **intro 12**.

If you get the infinitely long log, rerun the model specifying sem's (gsem's) iterate(#) option, where # is large enough to reach the "not concave" messages with constant criterion value. sem (gsem) will iterate that many times and then report the results it has at that point. Look at the output for missing standard errors. Those parameters are unidentified, and you need to think about either changing your model so that they become identified or placing constraints on them.

Identification 2: Normalization constraints (anchoring)

Normalization constraints (anchoring) are provided automatically by `sem` and `gsem`. It is rare indeed that the automatically provided constraints will be the source of convergence problems.

Models with continuous latent variables require normalization constraints because these latent variables have no natural scale. If constraints were not provided, the model would appear to the software the same as a model with a substantive lack of identification; the estimation routine would iterate forever and never arrive at a solution.

The `sem` and `gsem` commands automatically provide normalization constraints to prevent that from happening.

Below we explain why the normalization constraints are required, which normalization constraints `sem` and `gsem` automatically supply, how to override those automatic choices, and how to substitute your own constraints should you desire.

Why the problem arises

Imagine a latent variable for propensity to be violent. Your imagination might supply a scale that ranges from 0 to 1 or 1 to 100 or over other values, but regardless, the scale you imagine is arbitrary in that one scale works as well as another.

Scales have two components: mean and variance. If you imagine a latent variable with mean 0 and your colleague imagines the same variable with mean 100, the difference can be accommodated in the parameter estimates by an intercept changing by 100. If you imagine a standard deviation of 1 (variance $1^2 = 1$) and your colleague imagines a standard deviation of 10 (variance $10^2 = 100$), the difference can be accommodated by a path coefficient differing by a multiplicative factor of 10. You might measure an effect as being 1.1 and then your colleague would measure the same effect as being 0.11, but either way you both will come to the same substantive conclusions.

How the problem would manifest itself

The problem is that different scales all work equally well, and the software will iterate forever, jumping from one scale to another.

Another way of saying that the means and variances of latent variables are arbitrary is to say that they are unidentified. That's important because if you do not specify the scale you have in mind, results of estimation will look just like substantive lack of identification.

`sem` (`gsem`) will iterate forever and never arrive at a solution.

How sem (gsem) solves the problem for you

You usually do not need to worry about this problem because `sem` (`gsem`) solves it for you. `sem` (`gsem`) solves the unidentified scale problem for continuous latent variables by

1. Assuming that latent exogenous variables have mean 0.
2. Assuming that latent endogenous variables have intercept 0.
3. Setting the coefficients on paths from latent variables to the first observed endogenous variable to be 1.
4. Setting the coefficients on paths from latent variables to the first latent endogenous variable to be 1 if rule 3 does not apply—if the latent variable is measured by other latent variables only.

Rules 3 and 4 are also known as the unit-loading rules. The variable to which the path coefficient is set to 1 is said to be the anchor for the latent variable.

Applying those rules to our measurement model, when we type

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

sem (gsem) acts as if we typed

```
(X@1->x1) (X->x2) (X->x3) (X->x4), means(X@0)
```

The above four rules are sufficient to provide a scale for latent variables for all models.

Overriding sem's (gsem's) solution

sem (gsem) automatically applies rules 1 through 4 to produce normalization constraints. There are, however, other normalization constraints that would work as well. In what follows, we will assume that you are well versed in deriving normalization constraints and just want to know how to bend sem (gsem) to your will.

Before you do this, however, let us warn you that substituting your normalization rules for the defaults can result in more iterations being required to fit your model. Yes, one set of normalization constraints are as good as the next, but sem's (gsem)'s starting values are based on its default normalization rules, which means that when you substitute your rules for the defaults, the required number of iterations sometimes increases.

Let's return to the measurement model:

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

As we said previously, type the above and sem (gsem) acts as if you typed

```
(X@1->x1) (X->x2) (X->x3) (X->x4), means(X@0)
```

If you wanted to assume instead that the mean of X is 100, you could type

```
(X->x1) (X->x2) (X->x3) (X->x4), means(X@100)
```

The means() option allows you to specify mean constraints, and you may do so for latent or observed variables.

Let's leave the mean at 0 and specify that we instead want to constrain the second path coefficient to be 1:

```
(X->x1) (X@1->x2) (X->x3) (X->x4)
```

We did not have to tell sem (gsem) not to constrain X->x1 to have coefficient 1. We just specified that we wanted to constrain X->x2 to have coefficient 1. sem (gsem) takes all the constraints that you specify and then adds whatever normalization constraints are needed to identify the model. If what you have specified is sufficient, sem (gsem) does not add its constraints to yours.

Obviously, if we wanted to constrain the mean to be 100 and the second rather than the first path coefficient to be 1, we would type

```
(X->x1) (X@1->x2) (X->x3) (X->x4), means(X@100)
```

If we wanted to constrain the standard deviation of X to be 10 instead of constraining a path coefficient to be 1, we would type

```
(X->x1) (X->x2) (X->x3) (X->x4), means(X@100) var(X@100)
```

Standard deviations are specified in variance units when you use the var() option.

References

Acock, A. C. 2013. *Discovering Structural Equation Modeling Using Stata*. Rev. ed. College Station, TX: Stata Press.

Bollen, K. A. 1989. *Structural Equations with Latent Variables*. New York: Wiley.

Brown, T. A. 2015. *Confirmatory Factor Analysis for Applied Research*. 2nd ed. New York: Guilford Press.

Kenny, D. A. 1979. *Correlation and Causality*. New York: Wiley.

Kline, R. B. 2016. *Principles and Practice of Structural Equation Modeling*. 4th ed. New York: Guilford Press.

Li, C. 2013. Little's test of missing completely at random. *Stata Journal* 13: 795–809.

Satorra, A., and P. M. Bentler. 1994. Corrections to test statistics and standard errors in covariance structure analysis. In *Latent Variables Analysis: Applications for Developmental Research*, ed. A. von Eye and C. C. Clogg, 399–419. Thousand Oaks, CA: Sage.

Skrondal, A., and S. Rabe-Hesketh. 2004. *Generalized Latent Variable Modeling: Multilevel, Longitudinal, and Structural Equation Models*. Boca Raton, FL: Chapman & Hall/CRC.

Also see

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

[SEM] **intro 5** — Tour of models

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

[SEM] **sem and gsem option covstructure( )** — Specifying covariance restrictions