Description Remarks and examples Also see

Description

It can be devilishly difficult for software to obtain results for SEMs. Here is what can happen:

```
. sem ...
Variables in structural equation model
  (output omitted)
Fitting target model:
initial values not feasible
r(1400);
```

or,

```
. gsem ...
Fitting fixed-effects model:
Iteration 0:  log likelihood = -914.65237
Iteration 1:  log likelihood = -661.32533
Iteration 2:  log likelihood = -657.18568
  (output omitted)
Refining starting values:
Grid node 0:  log likelihood = .
Grid node 1:  log likelihood = .
Grid node 2:  log likelihood = .
Grid node 3:  log likelihood = .
Fitting full model:
initial values not feasible
r(1400);
```

or,

```
. sem ...
Endogenous variables
  (output omitted)
Fitting target model:
Iteration  1:  log likelihood = ...
.
.
.
Iteration 50:  log likelihood = -337504.44  (not concave)
Iteration 51:  log likelihood = -337503.52  (not concave)
Iteration 52:  log likelihood = -337502.13  (not concave)
.
.
.
Iteration 101: log likelihood = -337400.69  (not concave)
--Break--
r(1);
```

In the first two cases, `sem` and `gsem` gave up. The error message is perhaps informative if not helpful. In the last case, `sem` (it could just as well have been `gsem`) iterated and iterated while producing little improvement in the log-likelihood value. We eventually tired of watching a process that was going nowhere slowly and pressed *Break*.

Now what?

Remarks and examples

Remarks are presented under the following headings:

Is your model identified?
Convergence solutions generically described
Temporarily eliminate option reliability()
Use default normalization constraints
Temporarily eliminate feedback loops
Temporarily simplify the model
Try other numerical integration methods (gsem only)
Get better starting values (sem and gsem)
Get better starting values (gsem)

Is your model identified?

You may not know whether your model is identified. Infinite iteration logs are an indication of lack of identification or of poor starting values:

```
. sem ...
Endogenous variables
  (output omitted)
Fitting target model:
Iteration  1:  log likelihood = ...
.
.
.
Iteration 50:  log likelihood = -337504.44  (not concave)
Iteration 51:  log likelihood = -337503.52  (not concave)
Iteration 52:  log likelihood = -337502.13  (not concave)
.
.
.
Iteration 101: log likelihood = -337400.69  (not concave)
--Break--
r(1);
```

If the problem is lack of identification, the criterion function being optimized (the log likelihood in this case) will eventually stop improving at all and yet `sem` or `gsem` will continue iterating.

If the problem is poor starting values, the criterion function will continue to increase slowly.

So if your model might not be identified, do not press *Break* too soon.

There is another way to distinguish between identification problems and poor starting values. If starting values are to blame, it is likely that a variance estimate will be heading toward 0. If the problem is lack of identification, you are more likely to see an absurd path coefficient.

To distinguish between those two alternatives, you will need to rerun the model and specify the `iterate()` option:

```
. sem ..., ... iterate(100)
  (output omitted)
```

We omitted the output, but specifying `iterate(100)` allowed us to see the current parameter values at the point. We chose to specify `iterate(100)` because we knew that the likelihood function was changing slowly by that point.

If you are worried about model identification, you have a choice: Sit it out and do not press *Break* too soon, or press *Break* and rerun.

If you discover that your model is not identified, see *Identification 1: Substantive issues* in [SEM] **Intro 4**.

Convergence solutions generically described

There are three generic solutions to convergence problems that we will use:

G1. The improved-starting-values procedure:
Obtain the current parameter values from a failed attempt, modify those lousy values to make them better, use the improved values as starting values to try to fit the model again, and repeat as necessary.

G2. The alternative-starting-values procedure:
Simplify the model to produce an easier-to-fit model, fit the simplified model, use the simplified model's solution as starting values to fit the original, more complicated model, and repeat as necessary.

G3. The alternative-software-logic procedure:
Specify strange options that make the software behave differently in hopes that a different approach will produce a solution. `sem` does not have any such strange options, but `gsem` does. In following this approach, it does not matter whether we in fact understand what we are doing because, once we find a solution, we can obtain the parameter values from the successful model and use those values as starting values to fit the model without the strange and confusing options.

Sometimes we will use all three of these procedures. We will talk about convergence and these procedures substantively below, but before we do that, we want to show you how to use a tool that you will need.

The generic solutions share a mechanical step in common, namely, obtaining parameter values from one attempt at fitting the model to use as starting values in a subsequent attempt. Here is how you do that:

```
. sem ..., ...                    // fit one model
. matrix b = e(b)                 // save parameter estimates in b
. ...                             // optionally modify b
. sem ..., ... from(b)            // fit same model or different model
```

If the first `sem` or `gsem` command fails because you pressed *Break* or if the command issued an error message, you must reissue the command adding option `noestimate` or `iterate(#)`. Specify `noestimate` if the failure came early before the iteration log started, and otherwise specify `iterate(#)`, making # the iteration number close to but before the failure occurred:

```
. sem ..., ... noestimate
```

or

```
. sem ..., ... iterate(50)
```

Once you have obtained the parameter values in b, you can list b,

```
. matrix b = e(b)
. matrix list b
b[1,10]
            x1:         x1:         x2:         x2:         x3:         x3:
             L       _cons           L       _cons           L       _cons
y1           1      99.518   1.1207418      99.954   1.1149982      99.052
             /:          /:          /:          /:
      var(e.x1)   var(e.x2)   var(e.x3)      var(L)
y1   123.31427   97.675646   100.72204   82.311401
```

and you can modify it:

```
. matrix b[1, 10] = 500
. matrix list b
b[1,10]
            x1:         x1:         x2:         x2:         x3:         x3:
             L       _cons           L       _cons           L       _cons
y1           1      99.518   1.1207418      99.954   1.1149982      99.052
             /:          /:          /:          /:
      var(e.x1)   var(e.x2)   var(e.x3)      var(L)
y1   123.31427   97.675646   100.72204         500
```

And, whether you modify it or not, you can use b as the starting values for another attempt of the same model or for an attempt of a different model:

```
. sem ..., ... from(b)
```

Temporarily eliminate option reliability()

If you specified sem's or gsem's reliability() option, remove it and try fitting the model again. If the model converges, then your estimate of the reliability is too low; see *What can go wrong* in [SEM] **sem and gsem option reliability()**.

Use default normalization constraints

Let sem and gsem provide their default normalization constraints. By default, sem and gsem constrain all latent exogenous variables to have mean 0; constrain all latent endogenous variables to have intercept 0; and constrain the paths from latent variables to the first observed endogenous variable to have coefficient 1.

Replacing any of the above defaults can cause problems, but problems are most likely to arise if you replace the last listed default. Do not constrain path coefficients merely to obtain model identification. Let sem choose those constraints. It is possible that the default-coefficient-1 constraint is inappropriate for how you want to interpret your model. Relax it anyway and reimpose it later.

If default constraints solve the problem, you are done unless you want to reimpose your original, alternative constraints. That you do by typing

```
. sem ..., ...            // model with default constraints
. matrix b = e(b)
. sem ..., ... from(b)    // model with desired constraints
```

If you have multiple constraints that you want to reimpose, you may need to do them in sets.

Temporarily eliminate feedback loops

Check whether your model has any feedback loops, such as

```
. sem ... (y1<-y2 x2) (y2<-y1 x3) ...
```

In this example, variable y1 affects y2 affects y1. Models with such feedback loops are said to be nonrecursive. Assume you had a solution to the above model. The results might be unstable in a substantive sense; see *nonrecursive (structural) model (system)* in [SEM] **Glossary**. The problem is that finding such truly unstable solutions is often difficult and the stability problem manifests itself as a convergence problem.

If you have convergence problems and you have feedback loops, that is not proof that the underlying values are unstable.

Regardless, temporarily remove the feedback loop,

```
. sem ... (y1<-y2 x2) (y2<-   x3) ...
```

and see whether the model converges. If it does, save the parameter estimates and refit the original model with the feedback, but using the saved parameter estimates as starting values.

```
. matrix b = e(b)
. sem ... (y1<-y2 x2) (y2<-y1 x3) ..., ... from(b)
```

If the model converges, the feedback loop is probably stable. If you are using sem, you can check for stability with estat stable. If the model does not converge, you now must find which other variables need to have starting values modified.

Temporarily simplify the model

At this point, it is difficult to know whether you should temporarily simplify your model or simply proceed with the subsequent steps and come back to simplification later should it be necessary. So proceed in whatever order seems best to you.

The idea of temporarily simplifying the model is to simplify the model, get that model to converge, and use the simplified model's solution as starting values for the more complicated model.

The more orthogonal and independent you can make the pieces of the model, the better. Remove covariances. If you have measurement, fit it separately. Basically, remove whatever you hold most dear, because you are probably looking for subtle and correlated effects.

Once you find a simplified model that converges, do the following:

```
. sem ..., ...           // fit simplified model
. matrix b = e(b)
. sem ..., ... from(b)   // fit original model
```

Try other numerical integration methods (gsem only)

For models with continuous latent variables, gsem provides four numerical integration methods just so you can try them. The intmethod() option specifies the integration method.

1. intmethod(mvaghermite) is the default and performs mean-and-variance adaptive Gauss–Hermite quadrature. It is fast and accurate.

2. `intmethod(mcaghermite)` performs mode-and-curvature adaptive Gauss–Hermite quadrature. It is accurate but not as fast. If you are fitting a multilevel model, there are cases where method 1 will not work but this method will work.
3. `intmethod(ghermite)` performs nonadaptive Gauss–Hermite quadrature. It is less accurate but quicker, and the calculation it makes converges more readily than either of the above methods.
4. `intmethod(laplace)` performs the Laplacian approximation instead of quadrature. It is the least accurate, sometimes the fastest, and the calculation it makes has no convergence issues whatsoever.

Try all of them. We view methods 1 and 2 as completely trustworthy. If your model will only converge with method 3 or 4, we recommend using the result as starting values for method 1 or 2. Thus you might type

```
. gsem ..., ... intmethod(ghermite)
. matrix b = e(b)
. gsem ..., ... from(b)
```

There is another option we should mention, namely, `intpoints(#)`. Methods 1, 2, and 3 default to using seven integration points. You can change that. A larger number of integration points produces more accurate results but does not improve model convergence. You might reduce the number of integration points to, say, 3. A lower number of integration points slightly improves convergence of the model, and it certainly makes model fitting much quicker. Obviously, model results are less accurate. We have been known to use fewer integration points, but mainly because of the speed issue. We can experiment more quickly. At the end, we always return to the default number of integration points.

Get better starting values (sem and gsem)

If you observe,

```
. sem ..., ...
Variables in structural equation model
  (output omitted)
Fitting target model:
initial values not feasible
r(1400);
```

and you are using `sem`, we direct you back to *Temporarily simplify the model*. The problem that we discuss here seldom reveals itself as "initial values not feasible" with `sem`. If you are using `gsem`, we direct you to the next section, *Get better starting values (gsem)*. It is not impossible that what we discuss here is the solution to your problem, but it is unlikely.

We discuss here problems that usually reveal themselves by producing an infinite iteration log:

```
. sem ..., ...
Endogenous variables
  (output omitted)
Fitting target model:
Iteration  1:  log likelihood = ...
.
.
.
Iteration 50:  log likelihood = -337504.44  (not concave)
Iteration 51:  log likelihood = -337503.52  (not concave)
Iteration 52:  log likelihood = -337502.13  (not concave)
.
.
.
Iteration 101: log likelihood = -337400.69  (not concave)
--Break--
r(1);
```

The first thing to do is look at the parameter values. To do that, type

```
. sem ..., ... iterate(100)
```

We specified 100; you should specify an iteration value based on your log.

In most cases, you will discover that you have a variance of a latent exogenous variable going to 0, or you have a variance of an error (`e.`) variable going to 0.

Based on what you see, say that you suspect the problem is with the variance of the error of a latent endogenous variable `F` going to 0, namely, `e.F`. You need to give that variance a larger starting value, which you can do by typing

```
. sem ..., ... var(e.F, init(1))
```

or

```
. sem ..., ... var(e.F, init(2))
```

or

```
. sem ..., ... var(e.F, init(10))
```

We recommend choosing a value for the variance that is larger than you believe is necessary.

To obtain that value,

1. If the variable is observed, use `summarize` to obtain the summary statistics for the variable, square the reported standard deviation, and then increase that by, say, 20%.
2. If the variable is latent, use `summarize` to obtain a summary of the latent variable's anchor variable and then follow the same rule: use `summarize`, square the reported standard deviation, and then increase that by 20%. (The anchor variable is the variable whose path is constrained to have coefficient 1.)
3. If the variable is latent and has paths only to other latent variables so that its anchor variable is itself latent, follow the anchor's paths to an observed variable and follow the same rule: use `summarize`, square the reported standard deviation, and then increase that by 20%.
4. If you are using `gsem` to fit a multilevel model and the latent variable is at the observational level, follow advice 2 above. If the latent variable is at a higher level—say `school`—and its anchor is `x`, a Gaussian response with the identity link, type

```
. sort school
. by school: egen avg = mean(x)
. by school: generate touse = _n==1 if school<.
. summarize avg if touse==1
```

Square the reported standard deviation and add 20%.

5. If you are using gsem and the anchor of the latent variable is not Gaussian with the identity link, see the next section.

Do not dismiss the possibility that the bad starting values concern estimated parameters other than variances of latent exogenous or error variables, although variances of those kinds of variables is the usual case. Covariances are rarely the problem because covariances can take on any value and, whether too small or too large, usually get themselves back on track as the iterative process proceeds. If you need to specify an initial value for a covariance, the syntax is

```
. sem ..., ... cov(e.F*e.G, init(-25))
```

Substitute for −25 the value you consider reasonable.

The other possibility is that a path coefficient needs a better starting value, which is as unlikely as a covariance being the problem, and for the same reasons. To set the initial value of a path coefficient, add the `init()` option where the path is specified. Say the original `sem` command included `y<-x1`:

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

If you wanted to set the initial value of the path from `x1` to 3, modify the command to read

```
. sem ... (y<-(x1, init(3)) x2) ...
```

Get better starting values (gsem)

What we say below applies regardless of how the convergence problem revealed itself. You might have seen the error message "initial values not feasible", or some other error message, or you might have an infinite iteration log.

`gsem` provides two options to help you obtain better starting values: `startvalues()` and `startgrid()`.

For models with continuous latent variables, `startvalues(`*svmethod*`)` allows you to specify one of five starting-value calculation methods: `zero`, `constantonly`, `fixedonly`, `ivloadings`, and `iv`. By default, `gsem` uses `ivloadings`. Evidently, that did not work for you. Try the others, starting with `iv`:

```
. gsem ..., ... startvalues(iv)
```

If that does not solve the problem, proceed through the others in the following order: `fixedonly`, `constantonly`, and `zero`.

For models with categorical latent variables, `startvalues(`*svmethod*`)` allows you to specify one of seven starting-value calculation methods: `factor`, `classid`, `randomid`, `classpr`, `randompr`, `jitter`, and `zero`. By default, `gsem` uses `factor`.

If you have a variable, say, *guessid*, that provides a guess for the latent class membership of each observation, try

```
. gsem ..., ... startvalues(classid guessid)
```

You might even use another command, such as `cluster kmeans` with the `generate()` option to create a new variable to specify in place of *guessid*.

The option `startvalues(randomid)` provides a function similar to `startvalues(classid` *guessid*`)` but uses random guesses for the latent class membership. You may find it helpful to try multiple random guesses and use the one with the best log likelihood after a given number of EM iterations for starting values. To do this, specify the `draws(#)` suboption. For instance, to try 10 random guesses, you can specify `startvalues(randomid, draws(10))`. You can also specify the number of EM iterations that should be taken before deciding which model is best. To do this, specify the `emopts(iterate(#))` option.

If you have a set of variables, say, *guessprlist*, that provide guesses for the probabilities for latent class membership in each observation, try

```
. gsem ..., ... startvalues(classpr guessprlist)
```

The option `startvalues(randompr)` provides a similar function but uses random guesses for the latent class probabilities. You can also specify the `draws(#)` suboption here to specify the number of random guesses that should be tried. For instance, `startvalues(randompr, draws(10))` specifies that 10 random guesses for class probabilities should be tried and that the one with the best log likelihood after the EM iterations should be used as starting values.

The option `startvalues(jitter)` applies random adjustments to starting values computed from a normal approximation to each outcome.

The option `startvalues(zero)` is really only useful if you have your own set of starting values and you want `gsem` to use zero for any model parameter without a specified value.

Whether you have continuous or categorical latent variables, if you have starting values for some parameters but not others—perhaps you fit a simplified model to get them—you can combine the options `startvalues()` and `from()`:

```
. gsem ..., ...                              // simplified model
. matrix b = e(b)
. gsem ..., ... from(b) startvalues(iv)      // full model
```

You can combine `startvalues()` with the `init()` option, too. We described `init()` in the previous section.

The other special option `gsem` provides is `startgrid()`. This option is mostly helpful for models with continuous latent variables, but not so much for models with categorical latent variables. `startgrid()` can be used with or without `startvalues()`. `startgrid()` is a brute-force approach that tries various values for variances and covariances and chooses the ones that work best.

1. You may already be using a default form of `startgrid()` without knowing it. If you see `gsem` displaying Grid node 1, Grid node 2, … following Grid node 0 in the iteration log, that is `gsem` doing a default search because the original starting values were not feasible.

   The default form tries 0.1, 1, and 10 for all variances of all latent variables, by which we mean the variances of latent exogenous variables and the variances of errors of latent endogenous variables.

2. `startgrid(`*numlist*`)` specifies values to try for variances of latent variables.

3. `startgrid(`*covspec*`)` specifies the particular variances and covariances in which grid searches are to be performed. Variances and covariances are specified in the usual way. `startgrid(e.F e.F*e.L M1[school] G*H e.y e.y1*e.y2)` specifies that 0.1, 1, and 10 be tried for each member of the list.

4. `startgrid(`*numlist covspec*`)` allows you to combine the two syntaxes, and you can specify multiple `startgrid()` options so that you can search the different ranges for different variances and covariances.

Our advice to you is this:

1. If you got an iteration log and it did not contain Grid node 1, Grid node 2, …, then specify `startgrid(.1 1 10)`. Do that whether the iteration log was infinite or ended with some other error. In this case, we know that `gsem` did not run `startgrid()` on its own because it did not report Grid node 1, Grid node 2, etc. Your problem is poor starting values, not infeasible ones.

   A synonym for `startgrid(.1 1 10)` is just `startgrid` without parentheses.

   Be careful, however, if you have a large number of continuous latent variables. `startgrid` could run a long time because it runs all possible combinations. If you have 10 latent variables, that means $3^10 = 59{,}049$ likelihood evaluations.

   If you have a large number of continuous latent variables, rerun your difficult `gsem` command including option `iterate(#)` and look at the results. Identify the problematic variances and search across them only. Do not just look for variances going to 0. Variances getting really big can be a problem, too, and even reasonable values can be a problem. Use your knowledge and intuition about the model.

   Perhaps you will try to fit your model by specifying `startgrid(.1 1 10 e.F L e.X)`. Because values 0.1, 1, and 10 are the default, you could equivalently specify `startgrid(e.F L e.X)`.

   Look at covariances as well as variances. If you expect a covariance to be negative and it is positive, try negative starting values for the covariance by specifying `startgrid(-.1 -1 -10 G*H)`.

   Remember that you can have multiple `startgrid()` options, and thus you could specify `startgrid(e.F L e.X) startgrid(-.1 -1 -10 G*H)`.

2. If you got "initial values not feasible", you know that `gsem` already tried the default `startgrid`.

   The default `startgrid` only tried the values 0.1, 1, and 10, and only tried them on the variances of latent variables. You may need to try different values or try the same values on covariances or variances of errors of observed endogenous variables.

   We suggest you first rerun the model causing difficulty including the `noestimate` option. We also direct you back to the idea of first simplifying the model; see *Temporarily simplify the model*.

   If, looking at the results, you have an idea of which variance or covariance is a problem, or if you have few variances and covariances, we would recommend running `startgrid()` first. On the other hand, if you have no idea as to which variance or covariance is the problem and you have a large number of them, you will be better off if you first simplify the model. If, after doing that, your simplified model does not include all the variances and covariances, you can specify a combination of `from()` and `startgrid()`.

Also see

[SEM] **Intro 11** — Fitting models with summary statistics data (sem only)