Title

stata.com

sem path notation extensions — Command syntax for path diagrams

Description Syntax Options Remarks and examples Also see

Description

This entry concerns `sem` only.

The command syntax for describing your SEMs is fully specified by *paths*, `covariance()`, `variance()`, `covstructure()`, and `means()`. How that works is described in [SEM] **sem and gsem path notation** and [SEM] **sem and gsem option covstructure()**. See those sections before reading this section.

This entry concerns the path features unique to `sem`, and that has to do with the `group()` option for comparing different groups.

Syntax

`sem` *paths* ... [`, covariance() variance() means() group(`*varname*`)`]

`sem` *paths* ... [`, covstructure() means() group(`*varname*`)`]

paths specifies the direct paths between the variables of your model.

The model to be fit is fully described by *paths*, `covariance()`, `variance()`, `covstructure()`, and `means()`.

The syntax of these elements is modified (generalized) when the `group()` option is specified.

Options

`covariance()`, `variance()`, and `means()` are described in [SEM] **sem and gsem path notation**.

`covstructure()` is described in [SEM] **sem and gsem option covstructure()**.

`group(`*varname*`)` allows models specified with *paths*, `covariance()`, `variance()`, `covstructure()`, and `means()` to be automatically generalized (interacted) with the groups defined by *varname*; see [SEM] **Intro 6**. The syntax of *paths* and the arguments of `covariance()`, `variance()`, `covstructure()`, and `means()` gain an extra syntactical piece when `group()` is specified.

Remarks and examples

stata.com

The model you wish to fit is fully described by the *paths*, `covariance()`, `variance()`, `covstructure()`, and `means()` that you type. The `group(`*varname*`)` option,

```
. sem ..., ... group(varname)
```

specifies that the model be fit separately for the different values of *varname*. *varname* might be `sex` and then the model would be fit separately for males and females, or *varname* might be something else and perhaps take on more than two values.

Whatever *varname* is, `group(`*varname*`)` defaults to letting some of the path coefficients, covariances, variances, and means of your model vary across the groups and constraining others to be equal. Which parameters vary and which are constrained is described in [SEM] **sem group options**, but that is a minor detail right now.

In what follows, we will assume that *varname* is `mygrp` and takes on three values. Those values are 1, 2, and 3, but they could just as well be 2, 9, and 12.

Consider typing

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

and typing

```
. sem ..., ... group(mygrp)
```

Whatever the *paths*, `covariance()`, `variance()`, `covstructure()`, and `means()` are that describe the model, there are now three times as many parameters because each group has its own unique set. In fact, when you give the second command, you are not merely asking for three times the parameters, you are specifying three models, one for each group! In this case, you specified the same model three times without knowing it.

You can vary the model specified across groups.

1. Let's write the model you wish to fit as

   ```
   . sem (a) (b) (c), cov(d) cov(e) var(f)
   ```

   where *a*, *b*, . . . , *f* stand for what you type. In this generic example, we have two `cov()` options just because multiple `cov()` options often occur in real models. When you type

   ```
   . sem (a) (b) (c), cov(d) cov(e) var(f) group(mygrp)
   ```

   results are as if you typed

   ```
   . sem (1: a) (2: a) (3: a)                                  ///
        (1: b) (2: b) (3: b)                                   ///
        (1: c) (2: c) (3: c),                                  ///
                  cov(1: d) cov(2: d) cov(3: d)                ///
                  cov(1: e) cov(2: e) cov(3: e)                ///
                  var(1: f) cov(2: f) cov(3: f)  group(mygrp)
   ```

   The `1:`, `2:`, and `3:` identify the groups for which paths, covariances, or variances are being added, modified, or constrained.

   If `mygrp` contained the unique values 5, 8, and 10 instead of 1, 2, and 3, then `5:` would appear in place of `1:`; `8:` would appear in place of `2:`; and `10:` would appear in place of `3:`.

2. Consider the model

   ```
   . sem (y <- x) (b) (c), cov(d) cov(e) var(f) group(mygrp)
   ```

   If you wanted to constrain the path coefficient `(y <- x)` to be the same across all three groups, you could type

   ```
   . sem (y <- x@c1) (b) (c), cov(d) cov(e) var(f) group(mygrp)
   ```

   See item 12 in [SEM] **sem and gsem path notation** for more examples of specifying constraints. This works because the expansion of `(y <- x@c1)` is

   ```
   (1: y <- x@c1) (2: y <- x@c1) (3: y <- x@c1)
   ```

3. Consider the model

   ```
   . sem (y <- x) (b) (c), cov(d) cov(e) var(f) group(mygrp)
   ```

If you wanted to constrain the path coefficient `(y <- x)` to be the same in groups 2 and 3, you could type

```
. sem (1: y <- x) (2: y <- x@c1) (3: y <- x@c1) (b) (c),      ///
                              cov(d) cov(e) var(f) group(mygrp)
```

4. Instead of following item 3, you could type

```
. sem (y <- x) (2: y <- x@c1) (3: y <- x@c1) (b) (c),        ///
                              cov(d) cov(e) var(f) group(mygrp)
```

The part `(y <- x) (2: y <- x@c1) (3: y <- x@c1)` expands to

```
(1: y <- x) (2: y <- x) (3: y <- x) (2: y <- x@c1) (3: y <- x@c1)
```

and thus the path is defined twice for group 2 and twice for group 3. When a path is defined more than once, the definitions are combined. In this case, the second definition adds more information, so the result is as if you typed

```
(1: y <- x) (2: y <- x@c1) (3: y <- x@c1)
```

5. Instead of following item 3 or item 4, you could type

```
. sem (y <- x@c1) (1: y <- x@c2) (b) (c),      ///
                              cov(d) cov(e) var(f) group(mygrp)
```

The part `(y <- x@c1) (1: y <- x@c2)` expands to

```
(1: y <- x@c1)  (2: y <- x@c1)  (3: y <- x@c1)  (1: y <- x@c2)
```

When results are combined from repeated definitions, then definitions that appear later take precedence. In this case, results are as if the expansion read

```
(1: y <- x@c2) (2: y <- x@c1)  (3: y <- x@c1)
```

Thus coefficients for groups 2 and 3 are constrained. The group-1 coefficient is constrained to `c2`. If `c2` appears nowhere else in the model specification, then results are as if the path for group 1 were unconstrained.

6. Instead of following item 3, item 4, or item 5, you could not type

```
. sem (y <- x@c1) (1: y <- x) (b) (c),                ///
                              cov(d) cov(e) var(f) group(mygrp)
```

The expansion of `(y <- x@c1) (1: y <- x)` reads

```
(1: y <- x@c1)  (2: y <- x@c1)  (3: y <- x@c1)  (1: y <- x)
```

and you might think that `1: y <- x` would replace `1: y <- x@c1`. Information, however, is combined, and even though precedence is given to information appearing later, silence does not count as information. Thus the expanded and reduced specification reads the same as if `1: y <- x` was never specified:

```
(1: y <- x@c1)  (2: y <- x@c1)  (3: y <- x@c1)
```

7. Items 1–6, stated in terms of *paths*, apply equally to what is typed inside the `means()`, `variance()`, `covariance()`, and `covstructure()` options. For instance, if you typed

```
. sem (a) (b) (c), var(e.y@c1) group(mygrp)
```

then you are constraining the variance to be equal across all three groups.

If you wanted to constrain the variance to be equal in groups 2 and 3, you could type

```
. sem (a) (b) (c), var(e.y) var(2: e.y@c1) var(3: e.y@c1), group(mygrp)
```

You could omit typing `var(e.y)` because it is implied. Alternatively, you could type

```
. sem (a) (b) (c), var(e.y@c1) var(1: e.y@c2) group(mygrp)
```

You could not type

```
. sem (a) (b) (c), var(e.y@c1) var(1: e.y) group(mygrp)
```

because silence does not count as information when specifications are combined.

Similarly, if you typed

```
. sem (a) (b) (c), cov(e.y1*e.y2@c1) group(mygrp)
```

then you are constraining the covariance to be equal across all groups. If you wanted to constrain the covariance to be equal in groups 2 and 3, you could type

```
. sem (a) (b) (c), cov(e.y1*e.y2)                                    ///
                   cov(2: e.y1*e.y2@c1) cov(3: e.y1*e.y2@c1) ///
                   group(mygrp)
```

You could not omit `cov(e.y1*e.y2)` because it is not assumed. By default, error variables are assumed to be uncorrelated. Omitting the option would constrain the covariance to be 0 in group 1 and to be equal in groups 2 and 3.

Alternatively, you could type

```
. sem (a) (b) (c), cov(e.y1*e.y2@c1)                  ///
                   cov(1: e.y1*e.y2@c2)               ///
                   group(mygrp)
```

8. In the examples above, we have referred to the groups with their numeric values, 1, 2, and 3. Had the values been 5, 8, and 10, then we would have used those values.

   If the group variable `mygrp` has a value label, you can use the label to refer to the group. For instance, imagine `mygrp` is labeled as follows:

   ```
   . label define grpvals 1 Male  2 Female  3 "Unknown sex"
   . label values mygrp grpvals
   ```

   We could type

   ```
   . sem (y <- x) (Female: y <- x@c1) (Unknown sex: y <- x@c1) ..., ...
   ```

   or we could type

   ```
   . sem (y <- x) (2: y <- x@c1) (3: y <- x@c1) ..., ...
   ```

Also see

[SEM] **sem** — Structural equation model estimation command

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

[SEM] **Intro 2** — Learning the language: Path diagrams and command language

[SEM] **Intro 6** — Comparing groups