example 23 — Specifying parameter constraints across groups

Description Remarks and examples Also see

Description

Below we demonstrate how to constrain the parameters we want constrained to be equal across groups when using `sem` with the `group()` option.

We pick up where [SEM] **example 22** left off:

```
. use http://www.stata-press.com/data/r15/sem_2fmmby
. sem (Peer -> peerrel1 peerrel2 peerrel3 peerrel4)   ///
      (Par  -> parrel1  parrel2  parrel3  parrel4), group(grade)
. estat ginvariant
```

The `estat ginvariant` command implied that perhaps we could constrain all the variances and covariances to be equal across groups except for the variances of `e.parrel2` and `Peer`.

Remarks and examples

Remarks are presented under the following headings:

Background

We can specify which parameters we wish to allow to vary. Remember that `sem`'s `group()` option classifies the parameters of the model as follows:

Class description	Class name	
1. structural coefficients	`scoef`	
2. structural intercepts	`scons`	
3. measurement coefficients	`mcoef`	
4. measurement intercepts	`mcons`	
5. covariances of structural errors	`serrvar`	
6. covariances of measurement errors	`merrvar`	
7. covariances between structural and measurement errors	`smerrcov`	
8. means of exogenous variables	`meanex`	(*)
9. covariances of exogenous variables	`covex`	(*)
10. all the above	`all`	(*)
11. none of the above	`none`	

(*) Exogenous variables means just the latent exogenous variables unless you specify `sem` option `noxconditional` or you specify option `method(mlmv)` (which implies option `noxconditional`); see [SEM] **sem option noxconditional**.

When fitting a model with the `group()` option,

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

you may also specify the `ginvariant()` option:

```
. sem ..., ... group(varname) ginvariant(class names)
```

You may specify any of the class names as being `ginvariant()`. You may specify as many class names as you wish. When you specify `ginvariant()`, `sem` cancels its default actions on which parameters vary and which do not, and uses the information you specify. All classes that you do not mention as being `ginvariant()` are allowed to vary across groups.

By using `ginvariant()`, you can constrain, or free by your silence, whole classes of parameters. For instance, you could type

```
. sem ..., group(mygroup) ginvariant(mcoef mcons serrvar)
```

and you are constraining those parameters to be equal across groups and leaving unconstrained `scoef`, `scons`, `merrvar`, `smerrcov`, `meanex`, and `covex`.

In addition, if a class is constrained, you can still unconstrain individual coefficients. Consider the model

```
. sem ... (x1<-L) ...
```

If you typed

```
. sem ... (1: x1<-L@a1) (2: x1<-L@a2) ..., group(mygroup) ginvariant(all)
```

then all estimated parameters would be the same across groups except for the path `x1<-L`, and it would be free to vary in groups 1 and 2.

By the same token, if a class is unconstrained, you can still constrain individual coefficients. If you typed

```
. sem ... (1: x1<-L@a) (2: x1<-L@a) ..., group(mygroup) ginvariant(none)
```

then you would leave unconstrained all parameters except the path `x1<-L`, and it would be constrained to be equal in groups 1 and 2.

This is all discussed in [SEM] **intro 6**, including how to constrain and free variance and covariance parameters.

Fitting the constrained model

In our case, we wish to fit our model:

```
. sem (Peer -> peerrel1 peerrel2 peerrel3 peerrel4)   ///
      (Par  -> parrel1  parrel2  parrel3  parrel4),   ///
                 group(grade)
```

We impose constraints on all parameters except the variances of `e.parrel2` and `Peer`. We can do that by typing

```
. sem (Peer -> peerrel1 peerrel2 peerrel3 peerrel4)
>     (Par  -> parrel1  parrel2  parrel3  parrel4),
>                   group(grade)
>                   ginvariant(all)
>                   byparm
>                   var(1: e.parrel2@v1)
>                   var(2: e.parrel2@v2)
>                   var(1: Peer@v3)
>                   var(2: Peer@v4)
Endogenous variables
Measurement:  peerrel1 peerrel2 peerrel3 peerrel4 parrel1 parrel2 parrel3
              parrel4
Exogenous variables
Latent:       Peer Par
Fitting target model:
Iteration 0:   log likelihood = -5560.9934
Iteration 1:   log likelihood = -5552.3122
Iteration 2:   log likelihood = -5549.5391
Iteration 3:   log likelihood = -5549.3528
Iteration 4:   log likelihood = -5549.3501
Iteration 5:   log likelihood = -5549.3501
Structural equation model                       Number of obs     =        385
Grouping variable  = grade                      Number of groups  =          2
Estimation method  = ml
Log likelihood     = -5549.3501
 ( 1)  [peerrel1]1bn.grade#c.Peer = 1
 ( 2)  [peerrel2]1bn.grade#c.Peer - [peerrel2]2.grade#c.Peer = 0
 ( 3)  [peerrel3]1bn.grade#c.Peer - [peerrel3]2.grade#c.Peer = 0
 ( 4)  [peerrel4]1bn.grade#c.Peer - [peerrel4]2.grade#c.Peer = 0
 ( 5)  [parrel1]1bn.grade#c.Par = 1
 ( 6)  [parrel2]1bn.grade#c.Par - [parrel2]2.grade#c.Par = 0
 ( 7)  [parrel3]1bn.grade#c.Par - [parrel3]2.grade#c.Par = 0
 ( 8)  [parrel4]1bn.grade#c.Par - [parrel4]2.grade#c.Par = 0
 ( 9)  [/]var(e.peerrel1)#1bn.grade - [/]var(e.peerrel1)#2.grade = 0
 (10)  [/]var(e.peerrel2)#1bn.grade - [/]var(e.peerrel2)#2.grade = 0
 (11)  [/]var(e.peerrel3)#1bn.grade - [/]var(e.peerrel3)#2.grade = 0
 (12)  [/]var(e.peerrel4)#1bn.grade - [/]var(e.peerrel4)#2.grade = 0
 (13)  [/]var(e.parrel1)#1bn.grade - [/]var(e.parrel1)#2.grade = 0
 (14)  [/]var(e.parrel3)#1bn.grade - [/]var(e.parrel3)#2.grade = 0
 (15)  [/]var(e.parrel4)#1bn.grade - [/]var(e.parrel4)#2.grade = 0
 (16)  [/]cov(Peer,Par)#1bn.grade - [/]cov(Peer,Par)#2.grade = 0
 (17)  [/]var(Par)#1bn.grade - [/]var(Par)#2.grade = 0
 (18)  [peerrel1]1bn.grade - [peerrel1]2.grade = 0
 (19)  [peerrel2]1bn.grade - [peerrel2]2.grade = 0
 (20)  [peerrel3]1bn.grade - [peerrel3]2.grade = 0
 (21)  [peerrel4]1bn.grade - [peerrel4]2.grade = 0
 (22)  [parrel1]1bn.grade - [parrel1]2.grade = 0
 (23)  [parrel2]1bn.grade - [parrel2]2.grade = 0
 (24)  [parrel3]1bn.grade - [parrel3]2.grade = 0
 (25)  [parrel4]1bn.grade - [parrel4]2.grade = 0
 (26)  [peerrel1]2.grade#c.Peer = 1
 (27)  [parrel1]2.grade#c.Par = 1
```

	Coef.	OIM Std. Err.	z	P>|z|	[95% Conf.	Interval]
Measurement						
peerrel1						
Peer						
[*]	1	(constrained)				
_cons						
[*]	8.708274	.0935844	93.05	0.000	8.524852	8.891696
peerrel2						
Peer						
[*]	1.112225	.0973506	11.42	0.000	.9214217	1.303029
_cons						
[*]	7.858713	.1035989	75.86	0.000	7.655663	8.061763
peerrel3						
Peer						
[*]	1.416486	.113489	12.48	0.000	1.194052	1.638921
_cons						
[*]	7.398217	.1147474	64.47	0.000	7.173316	7.623118
peerrel4						
Peer						
[*]	1.196494	.0976052	12.26	0.000	1.005191	1.387796
_cons						
[*]	8.183148	.1021513	80.11	0.000	7.982936	8.383361
parrel1						
Par						
[*]	1	(constrained)				
_cons						
[*]	9.339558	.0648742	143.96	0.000	9.212407	9.46671
parrel2						
Par						
[*]	1.100315	.1362999	8.07	0.000	.8331722	1.367458
_cons						
[*]	9.255299	.0725417	127.59	0.000	9.11312	9.397478
parrel3						
Par						
[*]	2.051278	.2066714	9.93	0.000	1.64621	2.456347
_cons						
[*]	8.676961	.088927	97.57	0.000	8.502667	8.851255
parrel4						
Par						
[*]	1.529938	.154971	9.87	0.000	1.2262	1.833675
_cons						
[*]	9.045247	.0722358	125.22	0.000	8.903667	9.186826

```
-------------------------------------------------------------------------------
var(e.peer~1)  |
           [*] |   1.799133    .159059                    1.512898    2.139523
var(e.peer~2)  |
           [*] |   2.186953    .193911                    1.838086    2.602035
var(e.peer~3)  |
           [*] |   1.915661   .2129913                     1.54056    2.382094
var(e.peer~4)  |
           [*] |   1.767354   .1746104                     1.45622    2.144965
var(e.parr~1)  |
           [*] |   1.125082   .0901338                    .9615942    1.316366
var(e.parr~2)  |
             1 |   .9603043     .13383                     .730775    1.261927
             2 |   1.799668   .1747351                    1.487807    2.176898
var(e.parr~3)  |
           [*] |   .9606889   .1420406                    .7190021    1.283617
var(e.parr~4)  |
           [*] |   .8496935   .0933448                    .6850966    1.053835
     var(Peer) |
             1 |   1.951555   .3387796                    1.388727    2.742489
             2 |   1.361431   .2122853                    1.002927    1.848084
      var(Par) |
           [*] |   .4952527   .0927994                    .3430288    .7150281
---------------+---------------------------------------------------------------
cov(Peer,Par)  |
           [*] |   .4096197   .0708726     5.78   0.000   .2707118    .5485275
-------------------------------------------------------------------------------
Note: [*] identifies parameter estimates constrained to be equal across
      groups.
LR test of model vs. saturated: chi2(61)  =     75.25, Prob > chi2 = 0.1037
```

Notes:

1. In [SEM] **example 20**, we previously fit this model by typing

```
. sem (...) (...), group(grade)
```

   This time, we typed

```
. sem (...) (...), group(grade)        ///
                   ginvariant(all)     ///
                   byparm              ///
                   var(1: e.parrel2@v1)    ///
                   var(2: e.parrel2@v2)    ///
                   var(1: Peer@v3)     ///
                   var(2: Peer@v4)
```

2. We specified the `byparm` option so that the results are sorted by parameters rather than groups. With this option, `sem` will report a single value for a parameter estimate if it is constrained to be equal across the groups. Because most of the parameters are constrained to be equal across groups, the output is much shorter than the default output. The default output produces a separate table for each group.

3. Previously, `sem, group()` mentioned 20 constraints that it imposed because of normalization or because of assumed `ginvariant(mcoef mcons)`.

   This time, `sem, group()` mentioned 27 constraints. It applied more constraints because we specified `ginvariant(all)`.

4. After the `ginvariant(all)` option, we relaxed the following constraints:

```
var(1: e.parrel2@v1)
var(2: e.parrel2@v2)
var(1: Peer@v3)
var(2: Peer@v4)
```

`ginvariant(all)` specified, among other constraints, that

```
var(1: e.parrel2) == var(2: e.parrel2)
var(1: Peer) == var(2: Peer)
```

`ginvariant(all)` did that by secretly issuing the options

var(1: e.parrel2@*secretname1*)
var(2: e.parrel2@*secretname1*)
var(1: Peer@*secretname2*)
var(2: Peer@*secretname2*)

because that is how you impose equality constraints with the path notation. When we specified

```
var(1: e.parrel2@v1)
var(2: e.parrel2@v2)
var(1: Peer@v3)
var(2: Peer@v4)
```

our new constraints overrode the secretly issued constraints. It would not have worked to leave off the symbolic names; see [SEM] **sem path notation extensions**. We specified the symbolic names `v1`, `v2`, `v3`, and `v4`. `v1` and `v2` overrode *secretname1*, and thus the constraint that `var(e.parrel2)` be equal across the two groups was relaxed. `v3` and `v4` overrode *secretname2*, and thus the constraint that `var(Peer)` be equal across groups was relaxed.

Also see

[SEM] **example 20** — Two-factor measurement model by group

[SEM] **example 22** — Testing parameter equality across groups

[SEM] **intro 6** — Comparing groups

[SEM] **sem group options** — Fitting models on different groups