Bi-Factor Models and Exploratory Bifactor Rotation
A Quick Introduction

A bi-factor model combines the notion of a general factor with the idea of simple structure. For example, suppose six items have a general factor, but two groups of 3 items each also have a specific dimension associated with them. Then the factor pattern would be of the classic *bi-factor form,*

$$\Lambda = \begin{bmatrix} x & x & 0 \\ x & x & 0 \\ x & x & 0 \\ x & 0 & x \\ x & 0 & x \\ x & 0 & x \end{bmatrix} \tag{1.1}$$

in which x represents a free parameter to be estimated, while other elements are constrained to be zero.

The simple idea of a *bi-factor model* dates back to the early days of factor analysis, at least as early as 1937, but it is the subject of renewed interest in at least two areas. First, it is a model that makes sense for many data sets on substantive grounds, and second, it tends to arise in certain testing situations due to “testlet effects.”

Bi-factor models can be exploratory or confirmatory. Jennrich and associates at UCLA have developed methods for *exploratory bi-factor analysis* by modifying classic rotation methods to seek out a bi-factor structure automatically, rather than the traditional “simple structure.” This can be very useful in exploratory factor analysis, as it allows one to deal with the existence of a general factor, while discovering previously unanticipated dimensional structures.

Errors of inference can result when items that are assumed to be essentially unidimensional are plagued by what Sun-Joo Cho calls “nuisance dimensions.” As an example, consider the multidimensionality induced in a set of items within a test due to “testlet” effects. *Testlets* are groups of items based on the same stimulus. For example, on a reading test, a reading passage may be followed by several questions. The dependency among the testlet items induced by the fact that they refer to the same passage will cause the assumption of unidimensionality to be violated for this group of items, resulting in turn in errors of estimation.

A *bi-factor model* can be used to help account for additional testlet dimensions in a set of items. Reise and associates, among others, have used the bi-factor model to model unwanted dependencies in an item response framework. So, for example, Λ above could represent the factor pattern for two testlets composed of 3 items each.

So the general factor in a *bi-factor* model can be a key general factor, or a nuisance dimension. To ponder which is which, one must be able to find a bi-factor solution by rotation if one exists.

Unfortunately, until very recently, there was no reliable technology for rotating a solution to bi-factor form in the context of simple exploratory factor analysis. In a truly exploratory situation, one could obtain a factor pattern, then rotate it via *target* ("Procrustes") rotation to a particular pattern, but the problem remains --- what pattern? Ideally, one should have a rotation criterion that produces a bifactor pattern like the one in Equation (1.1).

Unfortunately, the closes thing to automatic bifactor rotation was the Schmid-Leiman (1957) method. Schmid and Leiman discussed a situation in which the factors have a *hierarchical* configuration. For example, suppose one obtains a simple structure from 3 factors, but the factors are correlated. Then these 3 factors have a correlation matrix **P**. One can then factor analyze that correlation matrix, to obtain a "higher order" solution.

For example, the first order solution is

$$\mathbf{y} = \mathbf{F}\mathbf{x} + \mathbf{e} \tag{1.2}$$

With the standard assumptions, this leads to

$$\mathbf{\Sigma} = \mathbf{FPF}' + \mathbf{U}^2 \tag{1.3}$$

But we also have

$$\mathbf{x} = \mathbf{g}h + \mathbf{d} \tag{1.4}$$

In this case, *h* is a single higher order factor, and we get, by a second application of the fundamental theorem,

$$Var(\mathbf{x}) = \mathbf{P} = \mathbf{gg}' + \mathbf{\Delta}^2 \tag{1.5}$$

Combining Equations (1.3) and (1.5), we get

$$\mathbf{\Sigma} = \mathbf{F}(\mathbf{gg}' + \mathbf{\Delta}^2)\mathbf{F}' + \mathbf{U}^2 \tag{1.6}$$

Notice that this can be written in the following form

$$\mathbf{\Sigma} = \mathbf{\Lambda\Lambda}' + \mathbf{U}^2 \tag{1.7}$$

where

$$\mathbf{\Lambda} = \begin{bmatrix} \mathbf{Fg} & \mathbf{F\Delta} \end{bmatrix} \tag{1.8}$$

Notice that there is an added factor, and since **Fg** is a linear combination of all the factors, it often will have non-negligible loadings in all positions. Since $\mathbf{F\Delta}$ is a diagonal rescaling of the original **F,** it will still have simple structure if the original **F** did.

In general, the Schmid-Leiman method fails on many examples. The problem is that, if there is a general factor, the *m* factor solution will not be close to simple structure to begin with.

Jennrich and Bentler (2011, 2012) have produced a very clever method that recovers a bi-factor solution by exploratory rotation in many situations where the Schmid-Leiman method fails. The technical details of their method would require more time to cover than I wish to in our abbreviated treatment, but some simple examples should suffice to demonstrate how potentially important and useful this method can be in exploring factor analytic data.

Download the *AdvancedFactorFunctions.txt* file to your working directory.

Open up R and load the code in the *AdvancedFactorFunctions.txt* file.

```
> source("AdvancedFactorFunctions.txt")
```

This simple matrix exhibits nearly perfect bifactor structure.

```
> A = matrix(c(1,1,0,2,1,0,1,1,0,2,0,1,1,0,1,2,0,1),6,3,byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    2    1    0
[3,]    1    1    0
[4,]    2    0    1
[5,]    1    0    1
[6,]    2    0    1
```

Next, we rotate **A** into a position in which **A'A** is diagonal, and the bifactor structure is clearly unrecognizeable.

```
> T <- eigen(t(A) %*% A)$vectors
> A.r <- A%*%T
> A.r
         [,1]       [,2]        [,3]
[1,] 1.165945 -0.7808688 -0.17554252
[2,] 2.083487 -0.7808688  0.22209836
[3,] 1.165945 -0.7808688 -0.17554252
[4,] 2.145588  0.6246950  0.07880251
[5,] 1.228046  0.6246950 -0.31883837
[6,] 2.145588  0.6246950  0.07880251
```

Next we do a varimax rotation using the GPForth function with a random start. In order to get the optimal rotation, several random starts may be needed. Notice that we could/would reverse the signs in the first column.

```
GPForth(A,method="varimax",Tmat=Random.Start(3))

$Lh
           [,1]      [,2]        [,3]
[1,] -1.2799326 0.5272179  0.28950641
[2,] -1.8107501 1.3115611 -0.03148967
[3,] -1.2799326 0.5272179  0.28950641
[4,] -0.6653278 2.1332148  0.08205772
[5,] -0.1345103 1.3488716  0.40305380
[6,] -0.6653278 2.1332148  0.08205772
```

Varimax doesn't produce a useful solution. It (of course) does not recover the bifactor solution, and the structure is nothing close to the classic "independent cluster" form characteristic of simple structure.

Now we try the bifactor rotation method. My service routine `FindBifactor` runs as many random starts as you want (25 in this case) and returns the best result.

```
> FindBifactor(A.r,25)
$Lh
           [,1]          [,2]          [,3]
[1,] -1.0000039  1.844015e-06 -9.999961e-01
[2,] -2.0000039 -2.839338e-06 -9.999922e-01
[3,] -1.0000039  1.844015e-06 -9.999961e-01
[4,] -1.9999953 -1.000009e+00  1.239751e-06
[5,] -0.9999953 -1.000005e+00 -2.643818e-06
[6,] -1.9999953 -1.000009e+00  1.239751e-06
```

To clean things up further, I generated another service routine that automatically flips the signs to eliminate negative loadings, and also rounds off the pattern. As you can see from the code, by default, it returns the pattern rounded to 2 digits.

```
> FindBifactorPattern(A.r,25)
     [,1] [,2] [,3]
[1,]    1    0    1
[2,]    2    0    1
[3,]    1    0    1
[4,]    2    1    0
[5,]    1    1    0
[6,]    2    1    0
```

In class, we will explore another example from Thurstone, in which a dramatic bifactor solution is revealed. In this example, we will use some advanced utility functions to help us decide on the number of factors and to examine various rotational position.