Oblique Factor Rotation Explained

Grant B. Morgan
Baylor University

September 6, 2014

A Step-by-Step Look at Promax Factor Rotation

For this post, I will continue my attempt to demistify factor rotation to the extent that I can. To keep things easier to visualize, I will only use two factors so we'll be rotating the solution in two-dimensional space. Let's get started by generating some data from a true two-factor model.

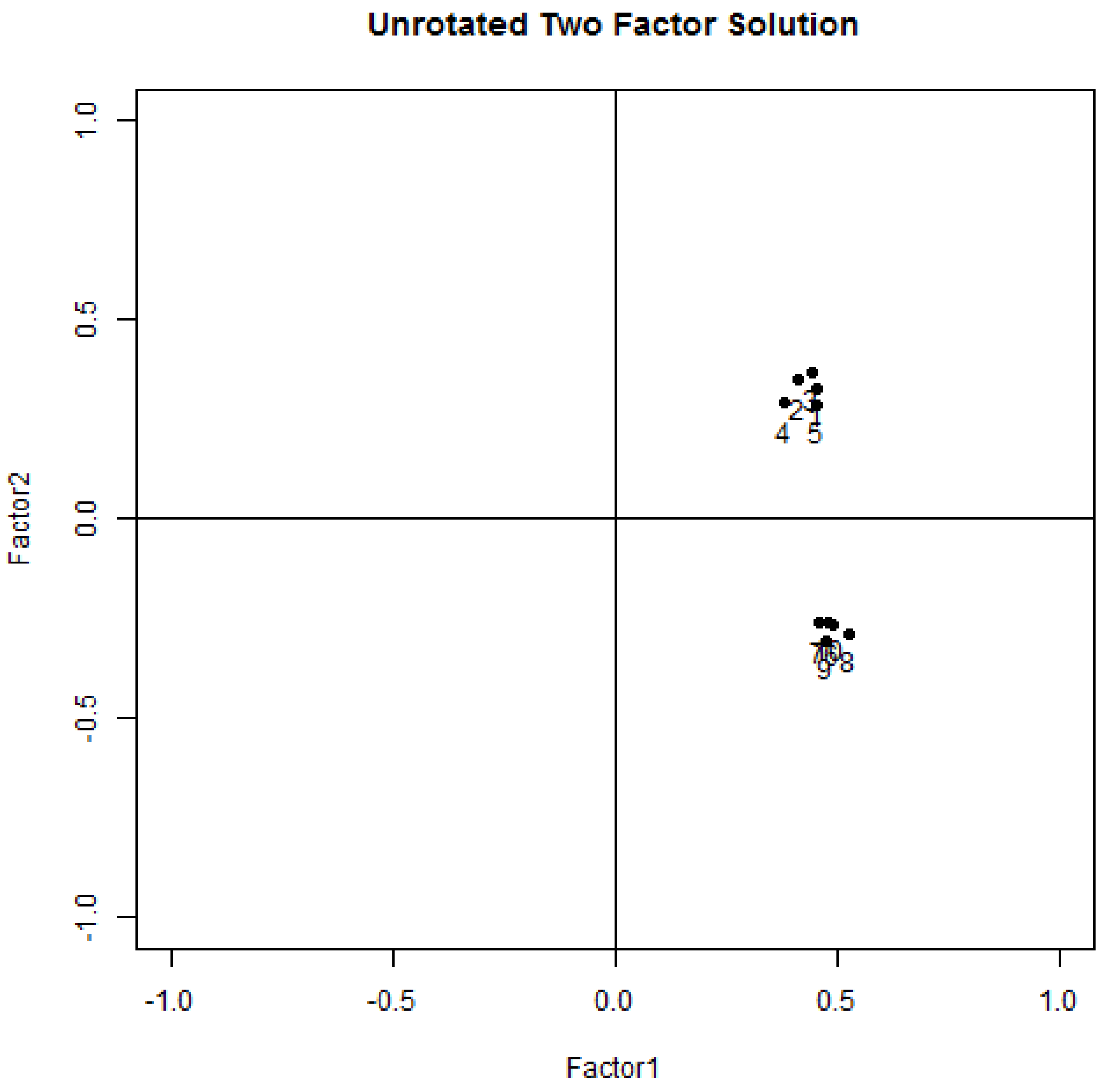
```r
library(lavaan)

oblique.model <- ' f1 =~ .7*x1 + 0.7*x2 + .7*x3 + .7*x4 + .7*x5
                   f2 =~ .7*x6 + .7*x7 + .7*x8 + .7*x9 + .7*x10
                   f1 ~ 1
                   f2 ~ 1
                   f2 ~~ .4*f1
                 '
oblique.data <- simulateData(oblique.model, sample.nobs=500L, seed=0110, std.lv=TRUE)
```

Perform EFA without rotation

First, let's extract two factors but keep the solution unrotated to see how it looks.

```r
library(psych)
fa.norotate <- factanal(factors = 2, covmat=cov(oblique.data), rotation="none")
factor.plot(fa.norotate, xlim=c(-1,1), ylim=c(-1,1), title="Unrotated Two Factor Solution")
```



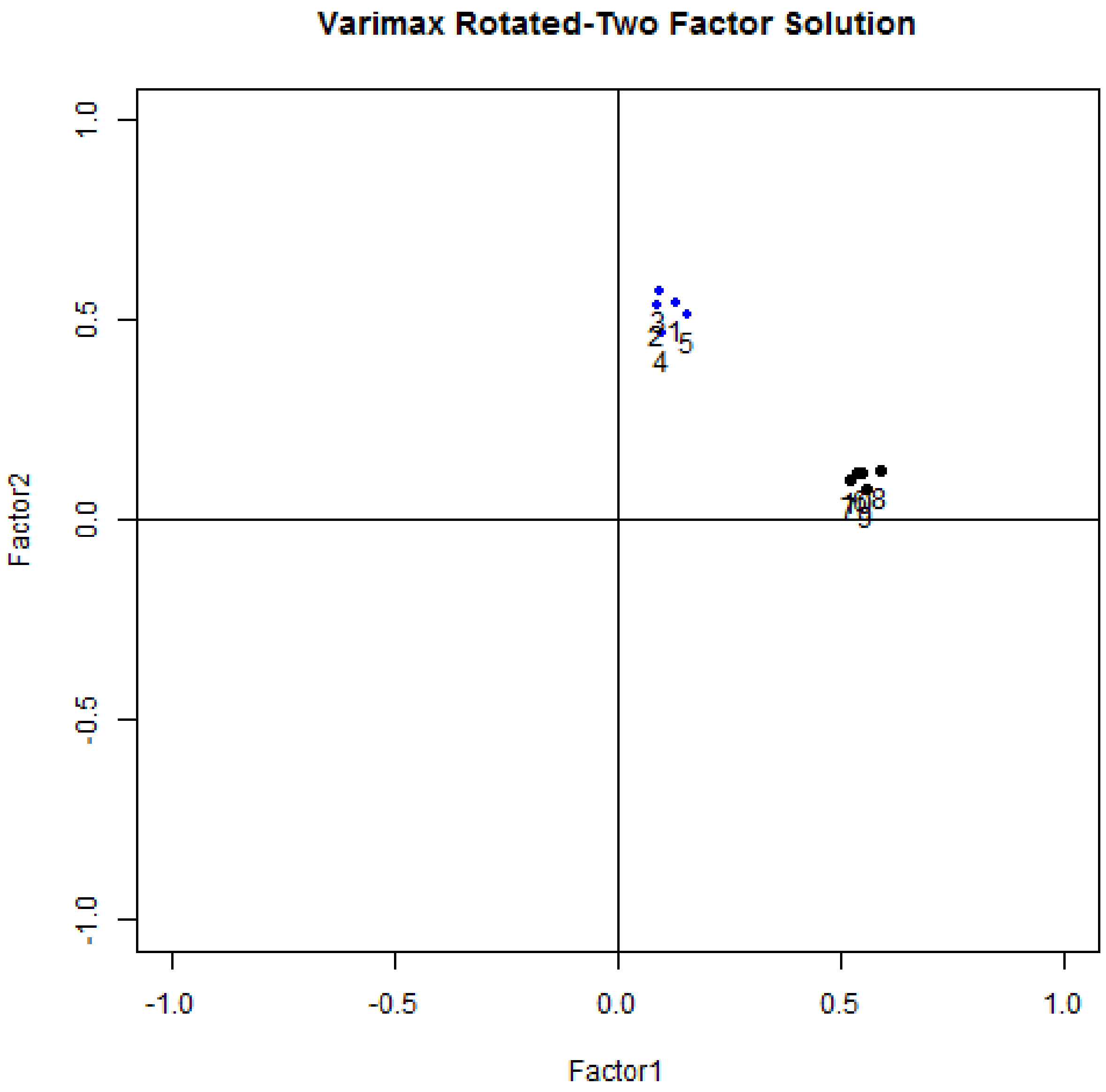
View each axis as a factor. Clearly, without rotating the solution, these clusters of items will be difficult to interpret because they are not close to either factor (i.e., axis). Rotation is clearly needed again. With orthogonal rotation, we kept both axes perpendicular to each other because we could get reasonably close the item clusters despite leaving the axes perpendicular. That isn't the case here.

As we have discussed, simple structure is always preferred in factor solutions to aid in interpreting the model. As rotation methods go, orthogonal rotations provide solutions that are easier to interpret. Varimax rotation, an orthogonal rotation procedure, was specifically designed to maximize simple structure. You might think of Varimax rotation as a best case scenario...so why don't we start with a Varimax solution and see if we can force (i.e., transform) the data to fit the Varimax solution? If you enjoy Greek mythology as much I do, this sounds very Procrustean, right? These procedures are actually classified as a variant of Procrustes analysis (seriously, look it up).

Step 1 - Perform EFA with Varimax Rotation

Let's use Varimax rotation and see what we come up with.

```r
library(GPArotation)
fa.varimax <- factanal(factors = 2, covmat=cov(oblique.data), rotation="Varimax")
factor.plot(fa.varimax, xlim=c(-1,1), ylim=c(-1,1), title="Varimax Rotated-Two Factor Solution"
```



The items are still in limbo between the axes, which is evidence that we need to break the orthogonality of the axes to better account for the data structure.

Step 2 - Create a Target Matrix from the Varimax-rotated Solution

Let's look at the loadings in the Varimax solution.

```
print(fa.varimax$loadings, cutoff=.01)

##
## Loadings:
##     Factor1 Factor2
## x1  0.131   0.543
## x2  0.087   0.534
## x3  0.097   0.569
## x4  0.099   0.467
## x5  0.156   0.513
## x6  0.548   0.117
## x7  0.519   0.101
## x8  0.587   0.125
## x9  0.560   0.080
## x10 0.535   0.116
##
##                Factor1 Factor2
## SS loadings      1.583   1.445
## Proportion Var   0.158   0.144
## Cumulative Var   0.158   0.303
```

It looks like items 1 to 5 load most strongly on Factor 2, and items 6 to 10 load most strongly onto Factor 1. If we wanted to strengthen simple structure, we would therefore want to loadings for items 1 to 5 to be closer to zero for Factor 1 and for items 6 to 10 for Factor 2. We would also want to do this without moving the primary loadings to zero. Can you think of way of transforming the loadings to make this happen? What if we squared all of the loadings? Let's try and take a look.

```
print((fa.varimax$loadings)**2, cutoff=.01)

##
## Loadings:
##     Factor1 Factor2
## x1  0.017   0.295
## x2          0.285
## x3          0.323
## x4          0.218
## x5  0.024   0.263
## x6  0.300   0.014
## x7  0.270   0.010
## x8  0.345   0.016
## x9  0.314
## x10 0.286   0.013
##
##                Factor1 Factor2
## SS loadings      0.463   0.391
## Proportion Var   0.046   0.039
## Cumulative Var   0.046   0.085
```

That looks better, right? Let's see if we can get the nonprimary loadings to be zero out to two decimal places. Let's try raising the loadings to the 4th power.

```
print((fa.varimax$loadings)**4, cutoff=.01)

##
## Loadings:
##     Factor1 Factor2
## x1          0.087
## x2          0.081
## x3          0.105
## x4          0.048
## x5          0.069
## x6  0.090
## x7  0.073
## x8  0.119
## x9  0.098
## x10 0.082
##
##                Factor1 Factor2
## SS loadings      0.044   0.032
## Proportion Var   0.004   0.003
## Cumulative Var   0.004   0.008
```

That does it! We know this because we used the cutoff command to suppress any loading less than .01. So this becomes the loading matrix that we want to target. Therefore, let's call it the target matrix and denote it as $\mathbf{\Lambda_T}$.

Step 3 - Find the Initial Transformation Matrix

Using the target matrix ($\mathbf{\Lambda_T}$) and the unrotated loading matrix ($\mathbf{\Lambda_0}$), we can find a initial transformation matrix using ordinary least squares. That would look like this:

$$\mathbf{T_i} = \left(\mathbf{\Lambda_0}'\mathbf{\Lambda_0}\right)^{-1}\mathbf{\Lambda_0'}\mathbf{\Lambda_T}$$

Don't let that scare you though...it's actually the same matrix operation as when we perform multiple regression. Remember that regression coefficients are estimated:

$$\mathbf{\hat{B}} = \left(\mathbf{X}'\mathbf{X}\right)^{-1}\mathbf{X}'\mathbf{y}$$

Using R we can compute the initial transformation matrix.

```
Ti <- solve(t(fa.norotate$loadings) %*% fa.norotate$loadings) %*% (t(fa.norotate$loadings) %*%
    (fa.varimax$loadings)^4)
Ti

##         Factor1 Factor2
## Factor1   0.109 0.07881
## Factor2  -0.143 0.13907
```

Step 4 - Find the Final Transformation Matrix

We can't stop here though. In promax rotation, the factor correlation matrix ($\mathbf{\Phi}$) is computed using ($(\mathbf{T}'\mathbf{T})^{-1} = \mathbf{\Phi}$). A correlation matrix must have 1s on the diagonal so if we stopped here, we wouldn't get 1s on the diagonal (see below).

```
solve(t(Ti)%*%Ti)

##         Factor1 Factor2
## Factor1   36.58   16.17
## Factor2   16.17   46.28
```

The diagonal of the matrix does not have 1s so we need to further transform the transformation matrix and hope it doesn't cause a black hole. First, we need to take the inverse of $\mathbf{T}'\mathbf{T}$. Second, we need to take the square root of the diagonal elements. Third, we'll create a new matrix that contains those roots on the diagonal and 0s off the diagonal. Let's do it.

```
Ti_inv<-solve(t(Ti)%*%Ti)
D<-matrix(0,nrow=nrow(Ti_inv), ncol=ncol(Ti_inv))
D[1,1]<-sqrt(Ti_inv[1,1])
D[2,2]<-sqrt(Ti_inv[2,2])
Ti_inv

##         Factor1 Factor2
## Factor1   36.58   16.17
## Factor2   16.17   46.28

D

##       [,1]  [,2]
## [1,] 6.048 0.000
## [2,] 0.000 6.803
```

Multiplying our intial transformation matrix by this new matrix, **D**, will give us the final transformation matrix (**T**). Thus:

$$\mathbf{T} = \mathbf{T_i D}$$

Let's take a look at **T**.

```
T<-Ti%*%D
T

##            [,1]   [,2]
## Factor1  0.6593 0.5362
## Factor2 -0.8648 0.9461
```

Step 4a - Compute the Factor Correlation Matrix

We went through all of this to get a factor correlation matrix that has 1s on the diagonal and factor correlations off the diagonal. Let's see if $\mathbf{T}'\mathbf{T}$ now has 1s on the diagonal.

```
solve(t(T)%*%T)

##       [,1]  [,2]
## [1,] 1.000 0.393
## [2,] 0.393 1.000
```

Yes! We successfully computed the factor correlation matrix. Not only do we see that the matrix has 1s on the diagonal, but we also see the correlation between the factors - $\phi_{2,1} = .39$ (Remember that we generated the data from a population where the factor correction was .4). The factor correlation matrix, $\mathbf{\Phi}$, is:

$$\mathbf{\Phi} = \begin{bmatrix} 1.000 & 0.393 \\ 0.393 & 1.000 \end{bmatrix}$$

Step 5 - Rotate the Factor Loadings

All that's left is to postmultiply the unrotated loading matrix ($\mathbf{\Lambda_0}$) by the final transformation matrix ($\mathbf{T}$) to get the rotated loading (i.e., pattern) matrix ($\mathbf{\Lambda}^*$). That is:

$$\mathbf{\Lambda_0 T} = \mathbf{\Lambda}^*$$

```
fa.norotate$loadings%*%T

##           [,1]     [,2]
## x1   0.016221  0.55224
## x2  -0.028644  0.55172
## x3  -0.025801  0.58652
## x4  -0.000014  0.47777
## x5   0.050125  0.51455
## x6   0.556540  0.00992
## x7   0.529757 -0.00179
## x8   0.596491  0.01036
## x9   0.577797 -0.03270
## x10  0.542398  0.01153
```

That's it! Now just for fun, let's see what we would get if we used the built-in promax rotation function.

Verifying Using Built-in Promax Rotation Procedure

```
fa(r = cor(oblique.data), fm = "ml", rotate="promax", nfactors=2)$loadings

## Loading required package:  MASS
## Loading required package:  parallel

##
## Loadings:
##     ML1    ML2
## x1          0.552
## x2          0.552
## x3          0.586
## x4          0.478
## x5          0.515
## x6   0.557
## x7   0.530
## x8   0.597
## x9   0.578
## x10  0.542
##
##                  ML1   ML2
## SS loadings    1.579 1.448
## Proportion Var 0.158 0.145
## Cumulative Var 0.158 0.303
```

```
fa(r = cor(oblique.data), fm = "ml", rotate="promax", nfactors=2)$Phi
```

```
##       [,1]  [,2]
## [1,] 1.000 0.393
## [2,] 0.393 1.000
```

Thankfully, it looks familiar! Promax rotation = Done!