Orthogonal Factor Rotation Explained

Grant B. Morgan
Baylor University

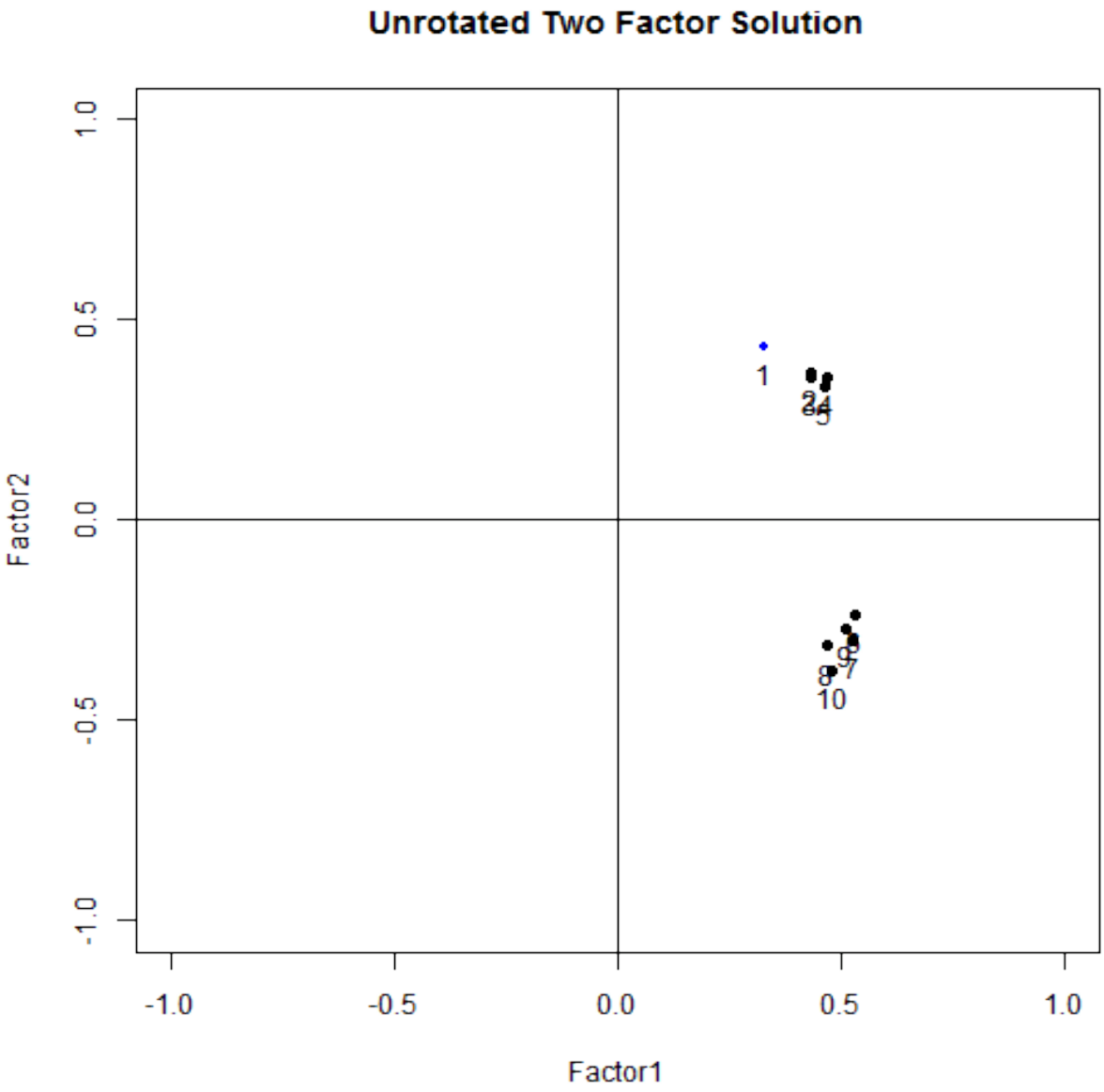
September 28, 2014

For this post, I will 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.

```
library(lavaan)

orthogonal.model <- ' f1 =~ .7*x1 + 0.7*x2 + .7*x3 + .7*x4 + .7*x5 + .1*x6 + .1*x7 + .1*x8 + .
                      f2 =~ .1*x1 + .1*x2 + .1*x3 + .1*x4 + .1*x5 + .7*x6 + .7*x7 + .7*x8 + .7
                      f1 ~ 1
                      f2 ~ 1
                      f2 ~~ 0*f1
                    '
ortho.data <- simulateData(orthogonal.model, sample.nobs=500L, seed=1028, std.lv=TRUE)
```

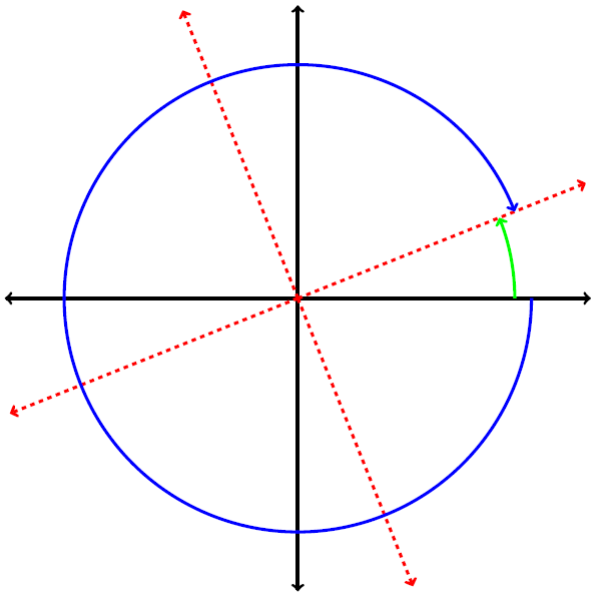
First, let's extract two factors but keep the solution unrotated to see how it looks.

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

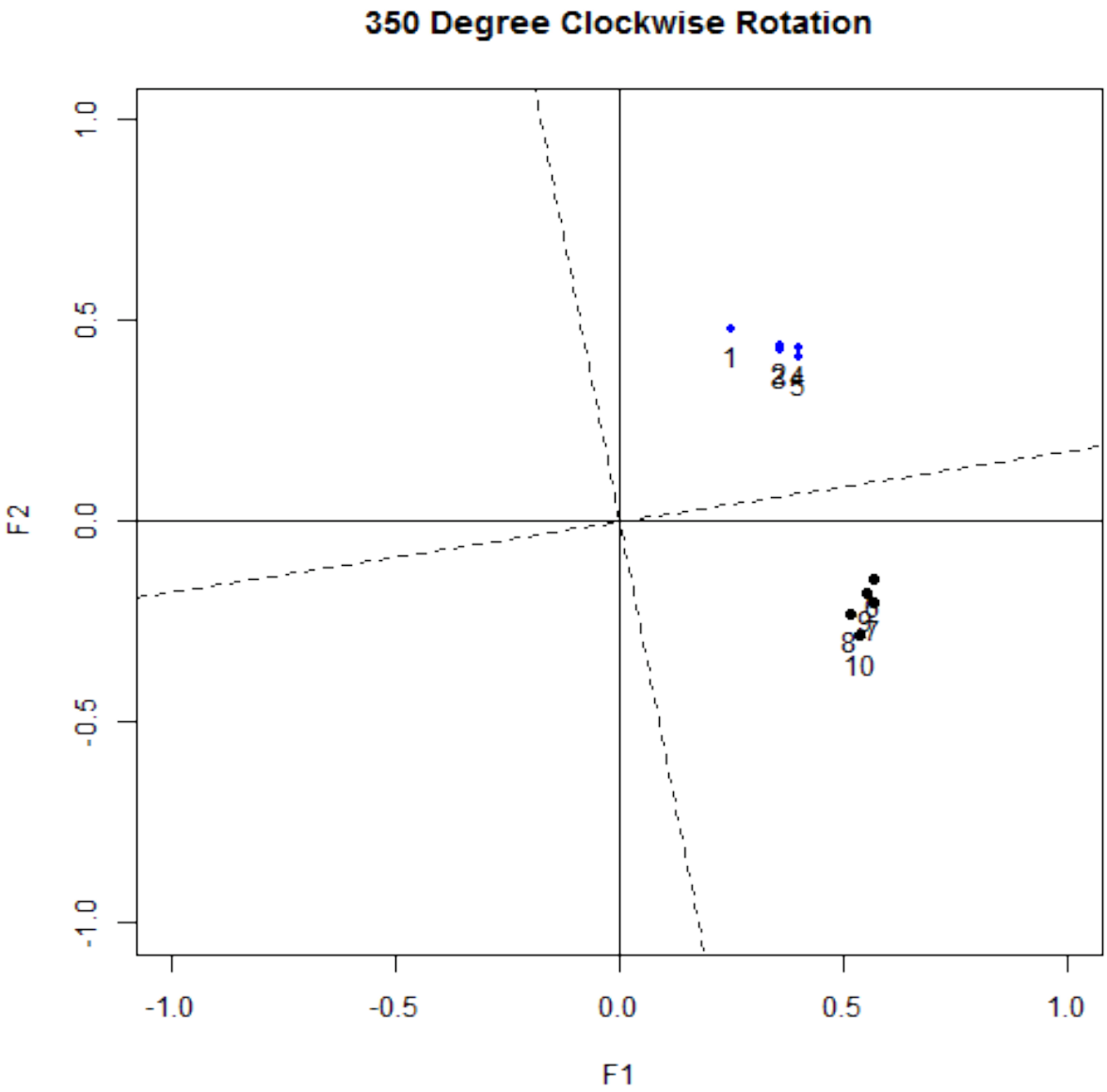


View each axis as a factor. Clearly, without rotating the solution, these clusters of items will be difficult to interpret because they are not close to either factor (i.e., axis). I'd like to start by rotating the solution by 10 degrees counterclockwise, but rotation prefers to to move clockwise! It shouldn't be news to anyone that there are 360 degrees in a circle. Rotation of 10 degrees counterclockwise is the same as rotating 320 ($360° - 40° = 320°$) degrees clockwise. My goal is to rotate all the way around because I want to keep the items as close to the upper-right quadrant as possible in order to keep the signs of the loadings positive.

Before we go any further, I want to be certain we understand that the degrees/angles of rotation are actually referring to. The angle of rotation is the angle created by any of the original axes and the same axis after rotation (see example below). The angle by which a solution is rotated is represented in the sample below by the blue line. I included the green line only because I wanted to illustrate how small counterclockwise rotation is equivalent to a large clockwise rotation.



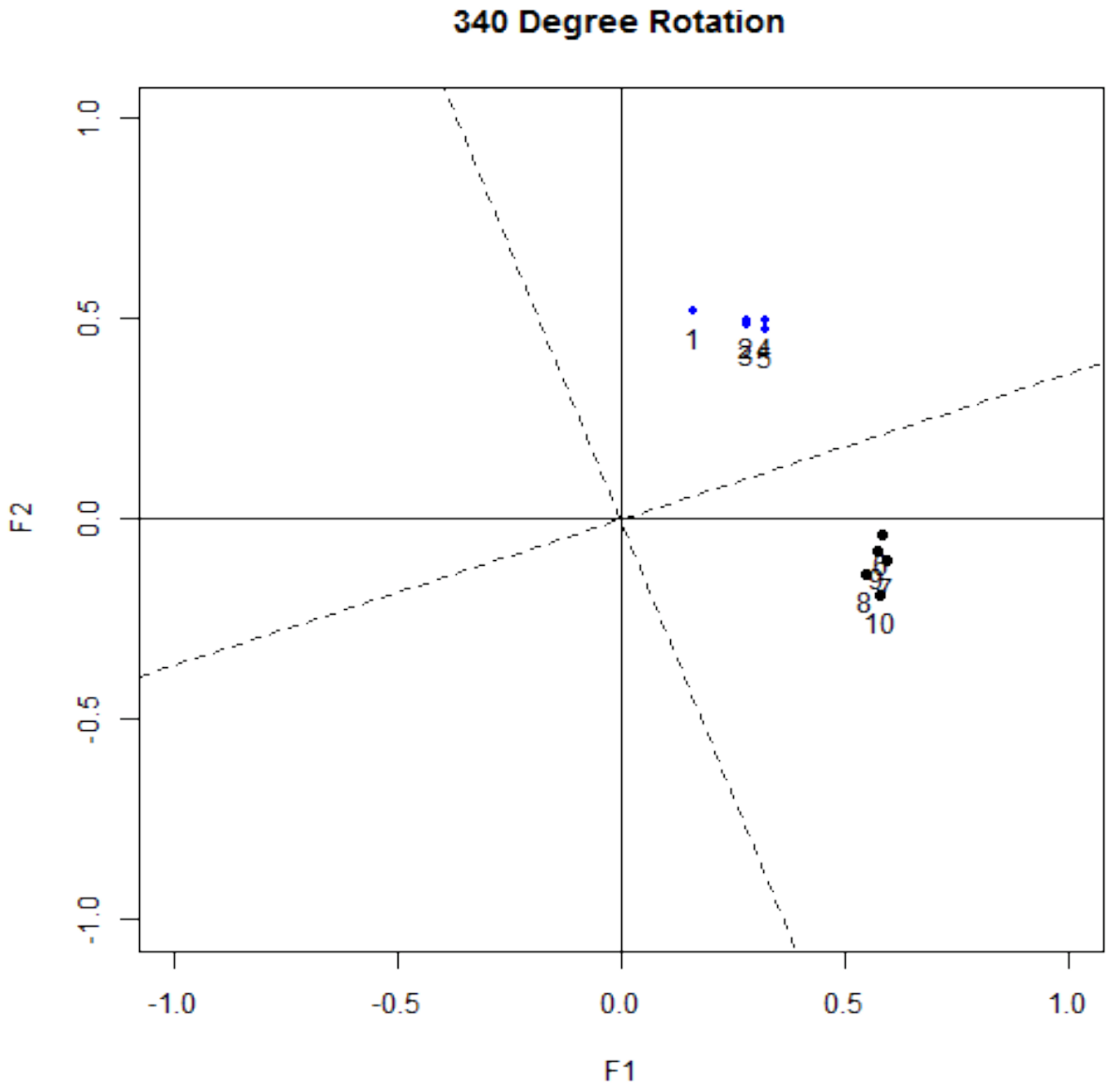
```
factor.rotate(efa.norotate, angle=350, plot=TRUE, title="350 Degree Clockwise Rotation", xlim=
```




```
##       [,1]    [,2]
## x1  0.2509  0.4814
## x2  0.3610  0.4386
## x3  0.3620  0.4256
## x4  0.4025  0.4326
## x5  0.3999  0.4077
## x6  0.5657 -0.1419
## x7  0.5689 -0.2026
## x8  0.5142 -0.2285
## x9  0.5500 -0.1791
## x10 0.5372 -0.2856
```

The dashed lines are the original axes, and the solid lines are the axes after rotation. We want to get each item cluster as close to an axis as possible. The 350 degree rotation is better than the unrotated solution, but it could be better. Let's try not rotating it quite as far.

```
factor.rotate(efa.norotate, angle=340, plot=TRUE, title="340 Degree Rotation", xlim=c(-1,1), yl
```

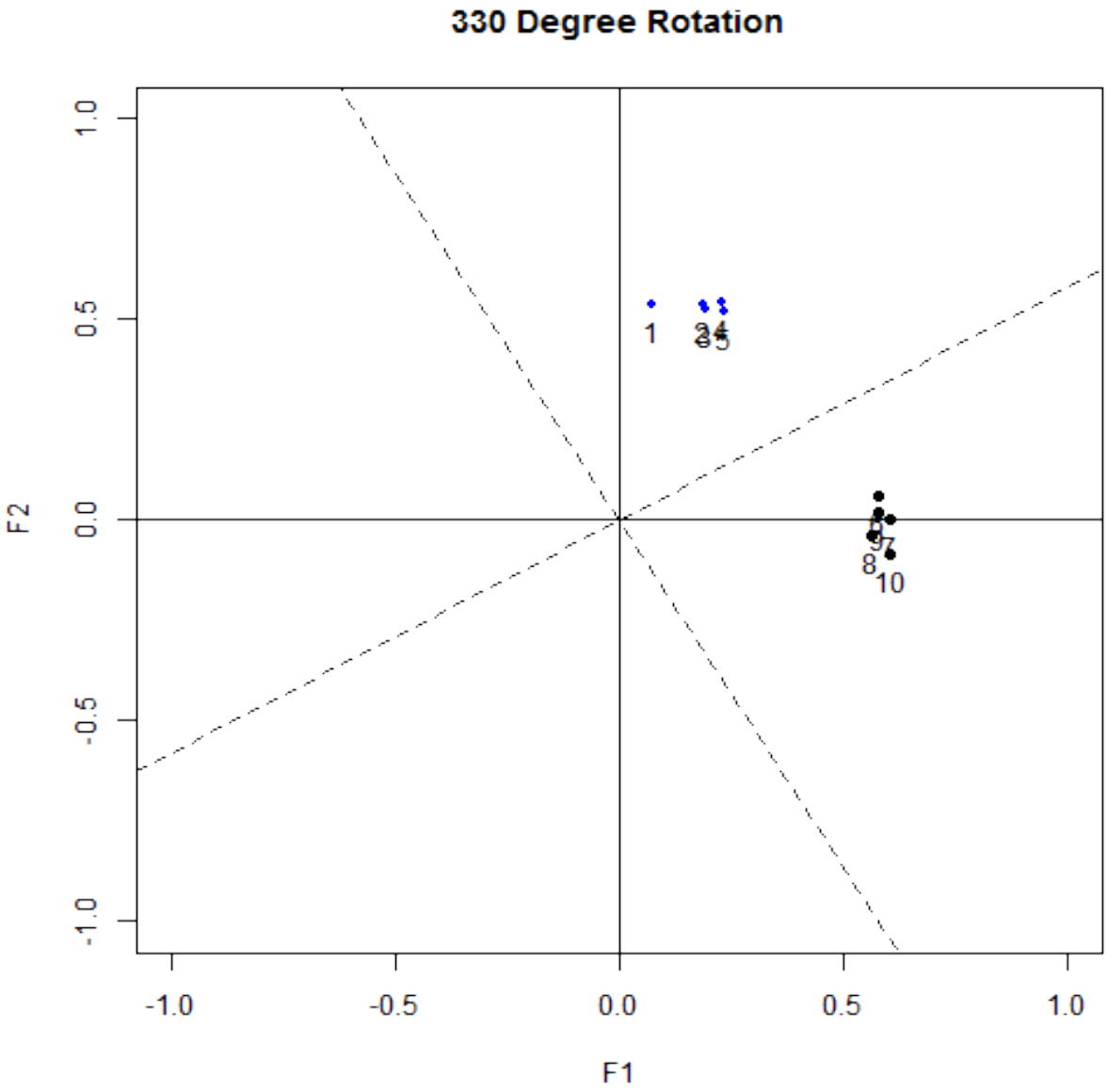



```
##       [,1]     [,2]
## x1  0.1635  0.51761
## x2  0.2794  0.49459
## x3  0.2826  0.48203
## x4  0.3212  0.49590
## x5  0.3230  0.47097
```

```
## x6  0.5817 -0.04147
## x7  0.5954 -0.10072
## x8  0.5461 -0.13572
## x9  0.5728 -0.08087
## x10 0.5787 -0.18802
```

The 340 degree rotation is better. Let's try not rotating it quite as far again.

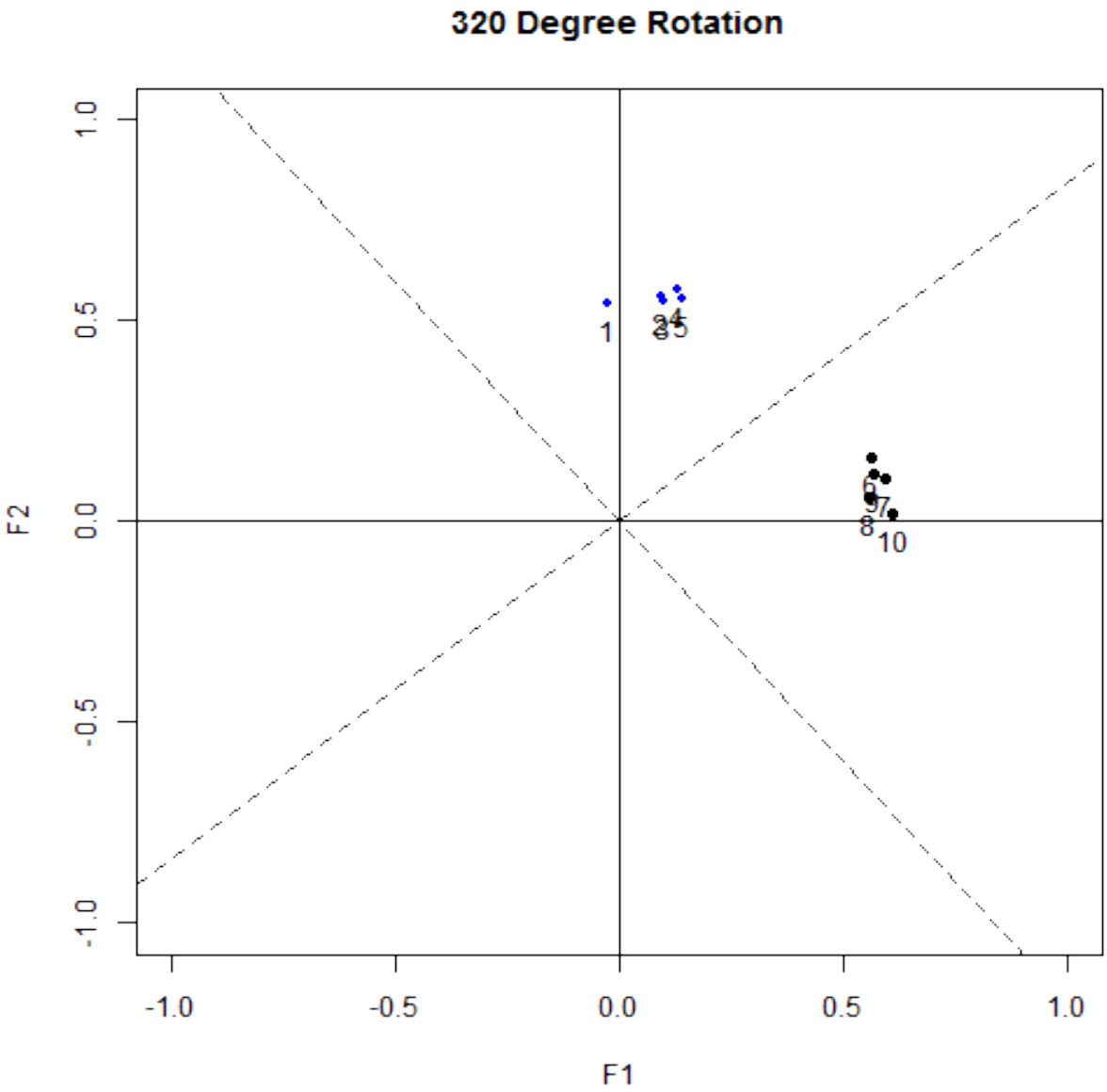
```
factor.rotate(efa.norotate, angle=330, plot=TRUE, title="330 Degree Rotation", xlim=c(-1,1), yl
```



```
##        [,1]      [,2]
## x1  0.07111  0.538134
## x2  0.18924  0.535584
## x3  0.19457  0.523777
## x4  0.23024  0.544143
## x5  0.23633  0.519910
## x6  0.58010  0.060177
## x7  0.60387  0.004201
## x8  0.56135 -0.038835
## x9  0.57813  0.019820
## x10 0.60252 -0.084678
```

The 330 degree rotation is better. Let's backing off a little more still.

```
factor.rotate(efa.norotate, angle=320, plot=TRUE, title="320 Degree Rotation", xlim=c(-1,1), yl
```




```
##          [,1]    [,2]
## x1  -0.02342 0.54231
## x2   0.09336 0.56031
## x3   0.10066 0.54961
## x4   0.13226 0.57586
## x5   0.14246 0.55305
## x6   0.56083 0.16000
## x7   0.59396 0.10900
## x8   0.55956 0.05923
## x9   0.56590 0.11991
## x10  0.60808 0.02124
```

The 320 degree rotation is even better. We could keep playing with the angle, but this solution looks pretty good.

Now that we've rotated the solution, the items have different loadings (i.e., coordinates). We need to find a way to figure out what the new coordinates are based on the 320 degree-rotated solution. I know that R gave us the new coordinates, but how did R find them? The trigonometric functions can help us (stop here if you want to Google the trigonometric functions).

Factor rotation is a geometric transformation that involves multiplying the unrotated loadings by some sort of matrix that transforms the loadings. There are two factors here so the rotation matrix will have two rows and two columns. Let's call the matrix **T**.

$$T = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

In this case, the angle (θ) is equal to 43. We can just plug these values in.

$$T = \begin{bmatrix} cos(320) & -sin(320) \\ sin(320) & cos(320) \end{bmatrix}$$

In R:

```
library(aspace)
T<-matrix(c(cos_d(320), -sin_d(320), sin_d(320), cos_d(320)), nrow=2, ncol=2, byrow=TRUE)
T
```

```
##         [,1]   [,2]
## [1,]  0.7660 0.6428
## [2,] -0.6428 0.7660
```

This matrix is what we multiple the original loading matrix by.

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

```
##         [,1]    [,2]
## x1  -0.02342 0.54231
## x2   0.09336 0.56031
## x3   0.10066 0.54961
## x4   0.13226 0.57586
## x5   0.14246 0.55305
## x6   0.56083 0.16000
## x7   0.59396 0.10900
## x8   0.55956 0.05923
## x9   0.56590 0.11991
## x10  0.60808 0.02124
```

Do these look familiar? They should because they're the same as the loadings produced when we used R to rotate the solution by 320 degrees.

0.1 Rotation Methods

You may be wondering at this point, how the computer determines which angle is the best angle for rotation. For illustrative purposes, let's focus on the Quartimax rotation procedure. The Quartimax procedures rotates the solution and computes a criterion for each rotation. The "best" solution according to Quartimax is the one that maximizes the criterion, which is:

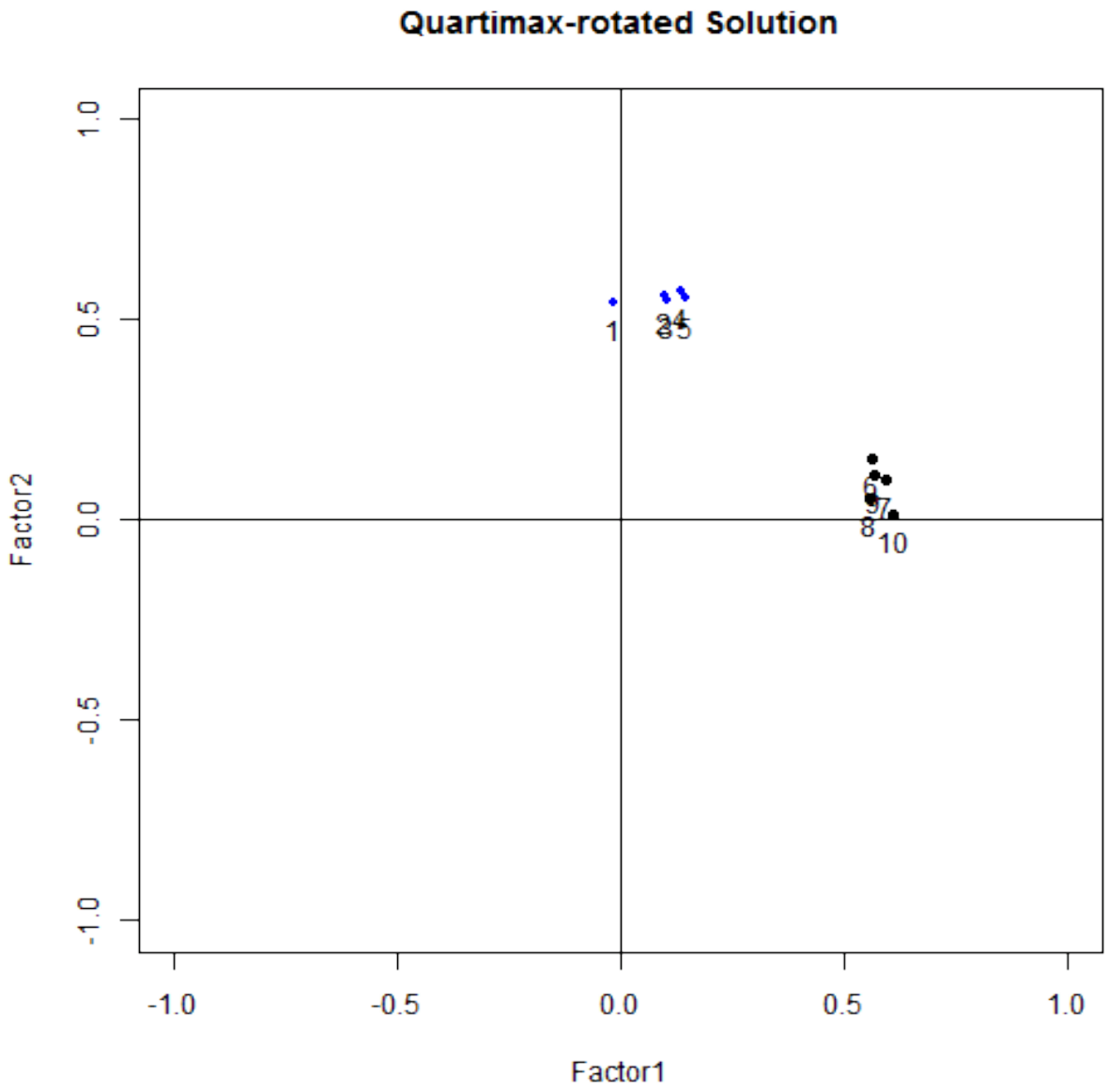
$$Q = \sum_{j=1}^{F} \sum_{k=1}^{V} \lambda_{jk}^4$$

Let's use Quartimax rotation and see how much the rotation matrix differs from what we computed from just eyeballing it.

```
library(GPArotation)
fa.quartimax <- factanal(factors = 2, covmat=cov(ortho.data), rotation="quartimax", cutoff=.01)
fa.quartimax$rotmat

##         [,1]    [,2]
## [1,] 0.7730 -0.6344
## [2,] 0.6344  0.7730

print(fa.quartimax$loadings, cutoff=.01)

##
## Loadings:
##     Factor1 Factor2
## x1  -0.018   0.543
## x2   0.099   0.559
## x3   0.107   0.548
## x4   0.138   0.574
## x5   0.148   0.551
## x6   0.563   0.154
## x7   0.595   0.103
## x8   0.560   0.053
## x9   0.567   0.114
## x10  0.608   0.015
##
##                Factor1 Factor2
## SS loadings      1.739   1.592
## Proportion Var   0.174   0.159
## Cumulative Var   0.174   0.333
```

It looks remarkably similar, right? Let's plot it also.

```
factor.plot(fa.quartimax, xlim=c(-1,1), ylim=c(-1,1), title="Quartimax-rotated Solution")
```



To compute the Quartimax criterion, we could do this with:

```
sum(fa.quartimax$rotmat**4)

## [1] 1.038
```

For any other two-factor solution, the Quartimax criterion would be smaller. Finally, let's see which angle Quartimax identified as the optimal solution.

```
library(aspace)
acos_d(fa.quartimax$rotmat[1,1])

## [1] 39.38
```

Quartimax rotated the solution by about 39.4 counterclockwise degrees, which is equivalent to saying 320.6 degrees clockwise. Compared against our 320 degrees. We did an awesome job!

Varimax rotation is a more popular orthogonal rotation method, and it rotates the solution in a very similar manner to Quartimax except Varimax maximizes the criterion below instead:

$$Q = \sum_{j=1}^{F} \sum_{k=1}^{K} \lambda_{jk}^4 - \frac{1}{k} \sum_{j=1}^{F} (\sum_{k=1}^{K} \lambda_{jk}^2)^2,$$

where k is the number of variables.

That's it. Orthogonal rotation = Done.