2.5 | Oblique Factor Rotation

The purpose of factor rotation is to improve factor interpretability by decreasing and hopefully eliminating any complexities among the original dimensions. Unfortunately, orthogonal rotation, which produces completely independent factors, may not always be sufficient to achieve this goal. In such situations, we can relax the need for orthogonality and complete independence and instead rotate the axes obliquely. Although most of the calculations and interpretations remain unchanged, there are a few more details that must be considered in oblique factor rotation.

2.4.1 | Obliquely Rotating a Loading Matrix

The easiest way to illustrate the difference between orthogonal and oblique rotation is to again visualize how well each factor aligns with the information contained in a loading matrix.

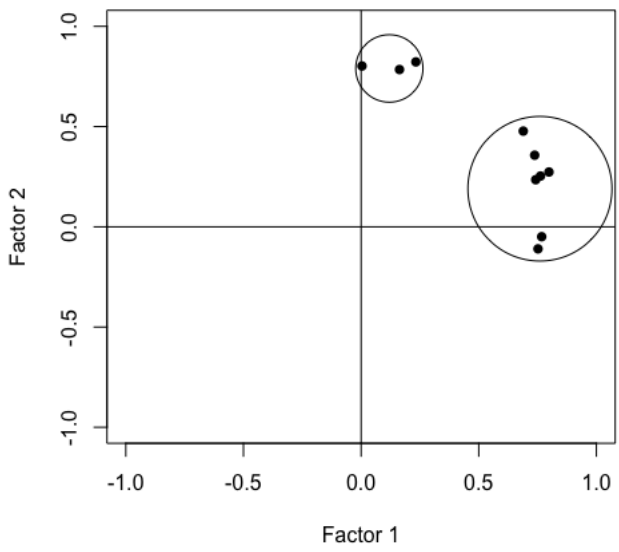
Consider again the loading matrix $\boldsymbol{A}$ for the optimally orthogonally rotated two-factor representation our example data matrix.

```
A <- pca(r = R, nfactors = 2, rotate = "varimax")$loadings[]
A
                          RC1         RC2
Hopeless          0.737684419  0.35721446
Overwhelmed       0.751913330 -0.10999159
Exhausted         0.767501553 -0.04935094
Lonely            0.762527268  0.25339680
Sad               0.798470525  0.27335188
Depressed         0.689253418  0.47745704
Anxious           0.742138215  0.23503718
SelfHarming       0.163183240  0.78456520
SuicidalThoughts  0.232827147  0.82265971
SuicidalAttempts  0.003767682  0.80222254
```

Recall that, at least for two- or three-factor solutions, we can represent the loadings of each dimension by using each row in the loading matrix as a set of cartesian coordinates. The coordinates for our first dimension, for example, are $(x_1, y_1) = (0.738, 0.357)$ and represent the loadings of our first dimension on both the first and second factor.

In R, we can use the `plot()` function to generate the corresponding factor loading plot.

```
plot(A, pch = 16, xlab = "Factor 1", ylab = "Factor 2",
     xlim = c(-1,1), ylim = c(-1,1))
abline(h = 0, v = 0)
points(0.12, 0.79, cex = 7); points(0.76, 0.19, cex = 15)
```



Note again that the varimax rotation resulted in both factor axes going directly through their respective clouds of points. This and other orthogonal optimization algorithms ensure that the absolute best alignment between factors and dimensions is found, but only so long as the factor axes remain perpendicular to each other.

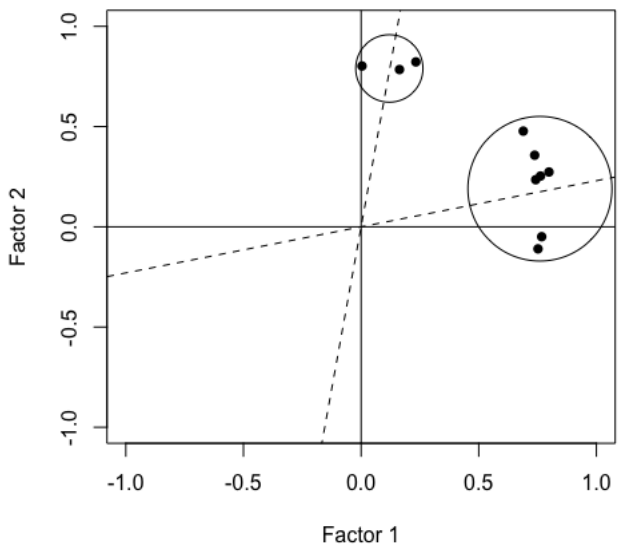
But could the alignment be improved if the requirement of orthogonality were relaxed? Using *oblique rotation*, we reduce complexity by rotating the axes freely, without necessarily keeping them perpendicular to each other, until all data points are as far from the spaces in between axes as possible.

We can visually estimate that a clockwise rotation of approximately 5 to 10 degrees should align the vertical axis with the centroid of the smaller cloud of data points. A counterclockwise rotation of approximately 10 to 15 degrees, on the other hand, is needed to align the horizontal axis with the centroid of the larger cloud of data points.

Whereas orthogonal rotation would have prevented rotating the factor axes in opposite directions or any other manipulation that resulted in anything other than perpendicular axes, oblique rotation is not bound by such limitations. This allows for a much more optimal alignment between axes and dimensions, one that is even able to ensure that each axis goes directly through the centroid of a cloud of points rather than simply bringing them closer to it.

In R, we can use a bit of trigonometry and clever plotting to display these rotated axes.

```
plot(A, pch = 16, xlab = "Factor 1", ylab = "Factor 2",
     xlim = c(-1,1), ylim = c(-1,1))
abline(h = 0, v = 0)
abline(coef = c(0, 6.47), lty = "dashed")
abline(coef = c(0, 0.23), lty = "dashed")
points(0.12, 0.79, cex = 7); points(0.76, 0.19, cex = 15)
```



Following this oblique rotation, the two clouds of data points are even more aligned with the two factor axes.

Before moving on, it is important to note that oblique rotation merely relaxes the requirement to maintain perpendicularity between axes, thereby allowing them to be rotated freely. The best alignment between factors and dimensions for a particular loading matrix, however, may end up being perpendicular even if perpendicularity is not strictly enforced. As such, care must be taken to determine if oblique rotation is truly necessary or if the simpler orthogonal rotation is sufficient, the exact process for which we will discuss in more detail in Section 2.5.3.

Another thing to note is that relaxing the requirement of perpendicularity can, under some circumstances, result in factor axes that are so close together that the corresponding factors become virtually indistinguishable. This is particularly likely if too many factors are extracted; as such, care must be taken to properly estimate the intrinsic dimensionality before proceeding with any factor analysis.

One final note is that despite the various factor validation methods introduced in Section 2.4 and the additional techniques that will be introduced below, factor loading plots remain one of the best ways to identify issues with rotations and to determine if rotation is even necessary.

As was the case with orthogonal rotations, numerous methods or algorithms have been proposed to identify the optimal angle for oblique rotation and thereby produce the least complex loading matrix possible. Again, however, if the correlation structure (a standardized version of the covariance structure) is fairly clear, most of these methods will yield very similar if not identical results and, as such, only a few are commonly used.

The most common oblique rotation algorithm by far is known as *oblimin*. This method seeks to minimize the cross products of the loadings in each column by making high loadings higher and low loadings lower for each factor.

In R, we can implement this algorithm by setting the `rotate` argument equal to `"oblimin"` in the `pca()` function.

```
A3 <- pca(r = R, nfactors = 2, rotate = "oblimin")$loadings[]
A3
                          TC1         TC2
Hopeless           0.72798915  0.21021186
Overwhelmed        0.80152412 -0.27544080
Exhausted          0.81024507 -0.21615827
Lonely             0.76704288  0.09762898
Sad                0.80219375  0.11050541
Depressed          0.66216280  0.34483718
Anxious            0.74798675  0.08305296
SelfHarming        0.07250148  0.77527910
SuicidalThoughts   0.14068165  0.79963917
SuicidalAttempts -0.09672167  0.82782741
```

Although the columns in this loading matrix now represent oblique rather than orthogonal factors, the corresponding interpretation of loadings and thresholds for complexity remain largely unchanged. The squared loadings, for example, still represent the unique proportion of variance in each dimension that is accounted for by a particular factor.

Something to note is that in oblique rotation, the loading matrix becomes the *pattern matrix*. However, this distinction is rarely used in analysis and as such will not be expanded upon here.

Comparing the correlations in this obliquely loading matrix to those in the orthogonally rotated loading matrix, we can see that feeling `Hopeless` has been successfully purified, leaving only a single dimension (feeling `Depressed`) as complex. Given that the goal of rotation is to decrease complexity, this result would indicate that oblique rotation was more effective than orthogonal. Whether this effectiveness was warranted, however, is something that we will return to in Section 2.5.3.

2.4.2 | Extracting Oblique Factor Scores

After an appropriate oblique rotation has been performed and the resulting loading matrix has been extracted, the next step is again to calculate the numerical value associated with this factor for each case in the original data matrix.

Thankfully, given the similarities between how loading matrices are used and interpreted between oblique and orthogonal rotation, we can simply use equations 2.23 and 2.24 to calculate the standardized scores on each factor (see Section 2.2.4).

To review, this begins by first calculating the matrix of regression coefficients $\boldsymbol{B}$, which will quantify how much each dimension should be weighted in the linear combination that will produce each factor.

```
B <- solve(R) %*% A3
B
                         TC1         TC2
Hopeless           0.1671172  0.01742994
Overwhelmed        0.2837717 -0.24115004
Exhausted          0.2746004 -0.21178667
Lonely             0.2004625 -0.04525038
Sad                0.2079951 -0.04300849
Depressed          0.1217655  0.09476099
Anxious            0.1978740 -0.05036682
SelfHarming       -0.1318391  0.38917242
SuicidalThoughts  -0.1171073  0.39320470
SuicidalAttempts  -0.1906476  0.43720707
```

These values can again be interpreted as coefficients in a multiple linear regression model. To calculate the standardized score on the first oblique factor for a given individual, for example, we would take their value on feeling Hopeless and multiply it by approximately 0.167, their value on feeling Overwhelmed by 0.284, and so on, and then add the resulting values together.

As before, however, these regression coefficients can only be applied to a standardized version of the original data. Given that the data matrix corresponding to our example data is not publicly available, we will instead use a fictional data matrix to demonstrate these results, as well as those related to factor correlations in Section 2.5.3.

Consider a fictional data matrix of the responses of four students corresponding to the ten dimensions in our example data.

```
D <- matrix(c(7.3,3.1,3.2,7.9,7.3,8.9,7.7,6.4,6.4,7.9,
              1.7,1.4,1.3,0.1,0.0,0.4,0.1,3.3,1.9,5.4,
              5.0,0.0,0.0,5.3,4.3,4.5,0.3,0.1,0.0,1.1,
              5.2,6.2,5.7,6.0,5.7,4.6,9.0,1.3,0.4,2.1),
            nrow = 4, ncol = 10, byrow = T,
            dimnames = list(c("1","2","3","4"),
                            c("Hopeless","Overwhelmed","Exhausted",
                              "Lonely","Sad","Depressed","Anxious",
                              "SelfHarming","SuicidalThoughts",
                              "SuicidalAttempts")))
D
  Hopeless Overwhelmed Exhausted Lonely Sad Depressed Anxious
1      7.3         3.1       3.2    7.9 7.3       8.9     7.7
2      1.7         1.4       1.3    0.1 0.0       0.4     0.1
3      5.0         0.0       0.0    5.3 4.3       4.5     0.3
4      5.2         6.2       5.7    6.0 5.7       4.6     9.0
  SelfHarming SuicidalThoughts SuicidalAttempts
1         6.4              6.4              7.9
2         3.3              1.9              5.4
3         0.1              0.0              1.1
4         1.3              0.4              2.1
```

The first individual in our fictional data matrix, for example, quantified their feeling Hopeless over the past year as a 7.3 on a 10-point scale, but feeling Overwhelmed as a 3.1.

We can now use R to find the standardized form of this fictional data matrix.

```
Z <- scale(D)
round(Z, 3)
  Hopeless Overwhelmed Exhausted Lonely    Sad Depressed Anxious
1    1.081       0.159     0.262  0.922  0.950     1.239   0.723
2   -1.340      -0.478    -0.505 -1.416 -1.380    -1.210  -0.882
3    0.086      -1.002    -1.029  0.142 -0.008    -0.029  -0.839
4    0.173       1.320     1.272  0.352  0.439     0.000   0.998
  SelfHarming SuicidalThoughts SuicidalAttempts
1       1.316            1.441            1.212
2       0.191           -0.094            0.409
3      -0.971           -0.742           -0.971
4      -0.536           -0.605           -0.650
attr(,"scaled:center")
        Hopeless      Overwhelmed        Exhausted           Lonely
           4.800            2.675            2.550            4.825
             Sad        Depressed          Anxious      SelfHarming
           4.325            4.600            4.275            2.775
SuicidalThoughts SuicidalAttempts
           2.175            4.125
attr(,"scaled:scale")
        Hopeless      Overwhelmed        Exhausted           Lonely
        2.313727         2.670050         2.477230         3.336041
             Sad        Depressed          Anxious      SelfHarming
        3.133023         3.470831         4.735944         2.753634
SuicidalThoughts SuicidalAttempts
        2.933002         3.115954
```

One benefit of standardizing data in this way is that now the dimensions are directly comparable, regardless of the scale or units that were originally used to measure them. The first individual's measure of feeling Hopeless, for example, is 1.1 standard deviations above the mean.

Now we can use simple matrix multiplication in R to calculate the corresponding matrix of standardized factor scores.

```
F1 <- Z %*% B
F1
          TC1         TC2
1  0.4008062  1.5317185
2 -1.4829798  0.4680911
3 -0.2952258 -0.5995335
4  1.3773994 -1.4002761
```

Because we standardized our fictional data matrix prior to calculating the matrix of factor scores, the factor scores contained therein can now also be interpreted as standardized scores. Our first individual, for example, has a score on the first oblique factor that is 0.40 standard deviations above the mean. If, as before, we name this factor to be psychological distress, then this fictional student's psychological distress is somewhat above average.

2.5.3 | Investigating the Factor Correlation Matrix

Now that the matrix of oblique factor scores has been calculated, we return to the question of whether or not oblique rotation was necessary in the first place.

Recall that the difference between oblique and orthogonal rotation is that oblique rotation simply relaxes the requirement of orthogonality. Mathematically, this allows the resulting oblique factors to be correlated with each other. As such, one way to determine if oblique rotation was necessary is to investigate the correlation structure between the resulting factors.

We can use the *factor correlation matrix* $\boldsymbol{\Phi}$ to compare the correlations between factors:

$$\boldsymbol{\Phi} = \frac{1}{n-1} \times \boldsymbol{F}^{\mathrm{T}} \times \boldsymbol{F} \tag{2.34}$$

In R, we can use matrix multiplication to calculate this factor correlation matrix or simply use the `cor()` function to arrive at the same answer.

```
PHI <- 1 / (nrow(D) - 1) * t(F1) %*% F1
PHI
            TC1         TC2
TC1  1.4480874 -0.6106631
TC2 -0.6106631  1.6284948

cor(F1)
            TC1         TC2
TC1  1.0000000 -0.3976592
TC2 -0.3976592  1.0000000
```

Given that the data matrix corresponding to our example data is not publicly available and the fictional data matrix cannot perfectly capture the correlation structure in the real data, these correlation matrices are not accurate and do not even match. That aside, we will continue to demonstrate how to interpret the values contained in this matrix and present a more accurate factor correlation matrix later in this section.

Similar to the correlations in a loading matrix, a common threshold for determining whether the correlation between factors is sufficiently strong to warrant oblique rotation is based on whether or not it exceeds an absolute value of 0.30. As before, this value is chosen because the corresponding coefficient of determination R^2 between the factors (which is simply equal to the square of the correlation) is just below 10%. Note again that this value is a much lower threshold than what was used to define a strong correlation in exploratory data analysis.

Before moving on, one thing to note is that in addition to calculating the loading matrix and numerous measures used in factor validation, the `pca()` function also includes the factor correlation matrix as part of the output.

```
pca(r = R, nfactors = 2, rotate = "oblimin")
Principal Components Analysis
Call: principal(r = r, nfactors = nfactors, residuals = residuals,
    rotate = rotate, n.obs = n.obs, covar = covar, scores = scores,
    missing = missing, impute = impute,
    oblique.scores = oblique.scores,
    method = method, use = use, cor = cor, correct = 0.5,
    weight = NULL)
Standardized loadings (pattern matrix) based upon correlation matrix
                   TC1   TC2   h2   u2 com
Hopeless          0.73  0.21 0.67 0.33 1.2
Overwhelmed       0.80 -0.28 0.58 0.42 1.2
Exhausted         0.81 -0.22 0.59 0.41 1.1
Lonely            0.77  0.10 0.65 0.35 1.0
Sad               0.80  0.11 0.71 0.29 1.0
Depressed         0.66  0.34 0.70 0.30 1.5
Anxious           0.75  0.08 0.61 0.39 1.0
SelfHarming       0.07  0.78 0.64 0.36 1.0
SuicidalThoughts  0.14  0.80 0.73 0.27 1.1
SuicidalAttempts -0.10  0.83 0.64 0.36 1.0

                       TC1  TC2
SS loadings           4.19 2.34
Proportion Var        0.42 0.23
Cumulative Var        0.42 0.65
Proportion Explained  0.64 0.36
Cumulative Proportion 0.64 1.00

 With component correlations of
     TC1  TC2
TC1 1.00 0.32
TC2 0.32 1.00

Mean item complexity =  1.1
Test of the hypothesis that 2 components are sufficient.

The root mean square of the residuals (RMSR) is  0.08

Fit based upon off diagonal values = 0.97
```

Using the highlighted factor correlation matrix in this output, we can see that the correlation between our oblique factors is actually equal to 0.32. Given that this value is beyond the common threshold of 0.30, we can conclude that oblique rotation was indeed necessary.

One final note is that, unlike orthogonally rotated factors, oblique factors are not statistically independent of each other. This poses a challenge because many statistical and predictive models, including most forms of regression, require independence or a lack of collinearity among variables in order to function properly.

Thankfully, as we will explore in Section 3.1, one of the main benefits of structural equation modeling is that it provides an effective means of explicitly incorporating collinearity and other dependencies among variables, thereby making it quite simple to including oblique factors, as well as any dimensions whose complexity could not be fully eliminated through factor rotation.

Exercises for Section 2.5

2.17 Cavity Trees, Dens, and Fishers (continued). Exercise 2.13 introduced data on the characteristics of eight cavity trees used as dens by fishers in Minnesota that were published in 2020 in the *Canadian Journal of Forest Research*, which are recreated below:

	DBH	Type	Slope	Aspect
DBH	1.00	-0.72	-0.09	-0.38
Type	-0.72	1.00	0.23	0.49
Slope	-0.09	0.23	1.00	-0.46
Aspect	-0.38	0.49	-0.46	1.00

The correlation matrix above was calculated using the DBH (diameter at breast height in cm) and Type (0 for coniferous and 1 for deciduous) of the tree itself, as well as the Slope (in degrees from horizontal) and Aspect (in degrees from due South) of the ground the tree is on for eight fisher dens throughout the Camp Ripley military facility in Minnesota.

a. Construct this correlation matrix in R with the proper row and column names
b. Find the orthogonally rotated loading matrix using the "varimax" algorithm for an intrinsic dimensionality of two
c. Create a factor loading plot of this loading matrix
d. Explain if it appears that oblique rotation would be effective in further decreasing any remaining complexities among the original dimensions

2.18 Stock Portfolios (continued). Exercise 1.13 introduced summary data on the performance of a diverse stock portfolio that spanned five different crypto currency stocks, which are recreated below:

	SOL	BNB	ETH	BTC	DOGE
SOL	3805	4980	42330	468665	0.354
BNB	4980	8293	64715	700917	1.401
ETH	42330	64715	559182	6294123	14.990
BTC	468665	700917	6294123	80834339	249.131
DOGE	0.354	1.401	14.990	249.131	0.005

The covariance matrix above was calculated using the daily adjusted closing values of five different crypto currency stocks over the course of a single year.

a. Construct this covariance matrix in R with the proper row and column names
b. Use matrix algebra to find the corresponding correlation matrix
c. Find the obliquely rotated loading matrix using the "oblimin" algorithm for an intrinsic dimensionality of two
d. Use matrix algebra to find the matrix of regression coefficients

e. Find the standardized score on the first oblique factor for a day where the performance of `SOL` was 1.3 standard deviations above average, that of `BNB` and `ETH` was 0.6 below average, that of `BTC` was 0.8 below average, and that of `DOGE` was 2.5 above average
f. Interpret the standardized performance of stocks on this day based on the factor score

2.19 Facework in Intercultural Communication (continued). Exercise 2.04 introduced data on the mediating effects of situations factors on the use of avoiding and corrective facework during intercultural miscommunication in face-to-face contexts, which are recreated below:

	1	2	3	4	5	6	7	8	9	10
1. Self-positive face	1.0	.35	.36	.19	.12	.18	.21	.23	-.15	-.13
2. Self-negative face	.35	1.0	.28	.41	.17	.03	.06	.24	.01	-.12
3. Other-positive face	.36	.28	1.0	.40	.23	.27	.25	.32	-.21	-.07
4. Other-negative face	.19	.41	.40	1.0	.24	.24	.27	.21	-.09	.05
5. Severity	.12	.17	.23	.24	1.0	.75	.74	-.01	.34	.29
6. Threatened positive face	.18	.03	.27	.24	.75	1.0	.78	.12	.12	.26
7. Threatened negative face	.21	.06	.25	.27	.74	.78	1.0	.17	.06	.24
8. Unintentional attribution	.23	.24	.32	.21	-.01	.12	.17	1.0	-.35	-.27
9. Intentional attribution	-.15	.01	-.21	-.09	.34	.12	.06	-.35	1.0	.27
10. Incidental attribution	-.13	-.12	-.07	.05	.29	.26	.24	-.27	.27	1.0

The correlation matrix above includes the covariability between ten different situational factors that were measured on 103 undergraduate students who were U.S. citizens.

a. Construct this correlation matrix in R with the proper row and column names
b. Find the intrinsic dimensionality using Kaiser's criterion
c. Use the `pca()` function to find the corresponding factor correlation matrix following an oblique rotation using the `"oblimin"` algorithm
d. Explain if oblique rotation was necessary or if an orthogonal rotation would have sufficed

2.20 Residential Property Values in Ames, Iowa (continued). Exercise 1.16 introduced a sample of data collected by the Ames Assessor's Office on residential properties, which are recreated below:

	Area	Price	Subclass	Lot Frontage	Lot Area	Quality	Built	Remodel
526301100	1656	215000	20	141	31770	6	1960	1960
526350040	896	105000	20	80	11622	5	1961	1961
526351010	1329	172000	20	81	14267	6	1958	1958
526353030	2110	244000	20	93	11160	7	1968	1968
527105010	1629	189900	60	74	13830	5	1997	1998
527105030	1604	195500	60	78	9978	6	1998	1998
527127150	1338	213500	120	41	4920	8	2001	2001
527145080	1280	191500	120	43	5005	8	1992	1992
527146030	1616	236500	120	39	5389	8	1995	1996
527162130	1804	189000	60	60	7500	7	1999	1999

527163010	1655	175900	60	75	10000	6	1993	1994
527165230	1187	185000	20	NA	7980	6	1992	2007
527166040	1465	180400	60	63	8402	6	1998	1998
527180040	1341	171500	20	85	10176	7	1990	1990
527182190	1502	212000	120	NA	6820	8	1985	1985
527216070	3279	538000	60	47	53504	8	2003	2003
527225035	1752	164000	50	152	12134	8	1988	2005
527258010	1856	394432	20	88	11394	9	2010	2010
527276150	864	141000	20	140	19138	4	1951	1951
527302110	2073	210000	20	85	13175	6	1978	1988

The data above include information on the first 20 rows of residential properties sold in Ames, IA from 2006 to 2010, including the price they sold for.

a. Construct this data matrix in R with the proper row and column names
b. Find the obliquely rotated loading matrix using the `"oblimin"` algorithm for a four-factor representation derived from a pairwise complete correlation matrix
c. Use matrix algebra to find the matrix of regression coefficients
d. Use matrix algebra to find the matrix of factor scores
e. Use the `cor()` function to find the corresponding factor correlation matrix using pairwise complete observations
f. Explain if oblique rotation was necessary or if an orthogonal rotation would have sufficed