A.2 | Numerical Optimization in Exploratory Factor Analysis

Having explored how orthogonal and oblique factor rotation can be used to reduce complexity among the dimensions retained as factors, we can now turn our attention back to the mathematical framework of identifying an optimal rotation. Given that achieving this goal involves finding a set of rotation angles that minimizes the overall complexity within a loading matrix, we will again rely heavily on numerical optimization. First, however, we need to establish a means of quantifying the amount of complexity in a loading matrix, something that is actually not quite as simple as we had originally indicated.

A.2.1 | Quantifying the Complexity of Loading Matrix

Recall that the loading matrix $\boldsymbol{A}$ represents the correlations or loadings between each of the original dimensions and the factors that were identified through a thorough investigation of the intrinsic dimensionality of the original data matrix.

Using an absolute loading threshold of 0.30, any dimensions that load substantially on to more than one factor are considered to be complex.

Consider again the unrotated loading matrix for the two-factor representation of our example data.

```
A <- pca(r = R, nfactors = 2, rotate = "none")$loadings[]
A
                          PC1         PC2
Hopeless        0.8168518 -0.06733209
Overwhelmed     0.5916281 -0.47691509
Exhausted       0.6358756 -0.43261569
Lonely          0.7854763 -0.16936567
Sad             0.8265731 -0.17045037
Depressed       0.8362622  0.06083607
Anxious         0.7585855 -0.17481308
SelfHarming     0.5393196  0.59271045
SuicidalThoughts 0.6186590  0.59011735
SuicidalAttempts 0.4110085  0.68894644
```

We can see that five of the ten dimensions in our example data are complex under this two-factor representation. But is there a better way to quantify the total amount of complexity across all dimensions?

One means of quantifying the complexity of each dimension, beyond simply determining if the dimension loads substantially on to more than one factor, is to use the *Hoffman's index of complexity c* that is introduced in Section 2.4:

$$c_i = \frac{\left(\sum_{j=1} a_{ij}{}^2\right)^2}{\sum_{j=1} a_{ij}{}^4} \tag{A.4}$$

Recall that this index denotes the number of factors needed to accurately represent the variability in that dimension.

We can quickly implement this equation for in R using simple arithmetic.

```
C <- rowSums(A ^ 2) ^ 2 / rowSums(A ^ 4)
C
         Hopeless       Overwhelmed         Exhausted            Lonely
         1.013588          1.913774          1.762399          1.092785
              Sad         Depressed           Anxious       SelfHarming
         1.084894          1.010584          1.105912          1.982439
SuicidalThoughts SuicidalAttempts
         1.995555          1.631779
```

For our example data, several dimensions, particularly Hopeless and Depressed, have indices of approximately 1.0, indicating that a single factor was sufficient to represent them. This is ideal in factor analysis, as it indicates an almost completely pure dimension.

Higher indices indicate higher levels of complexity within a dimension. SelfHarming and SuicidalThoughts, for example have indices of approximately 2.0, indicating that both factors are needed to sufficiently represent them. Although there is no set threshold for establishing complexity using these indices, it is common to consider any values above 1.3 as indicative of complexity.

We can use the mean of these indices to provide a single numerical measure of the complexities across all dimensions in a loading matrix.

```
mean(C)
[1] 1.459371
```

This calculation can serve as the objective function to be minimized in order to find the optimum orthogonal factor rotation, a topic we will return to in Section A.2.2. For now, let us explore alternative methods of quantifying the complexities in a loading matrix.

The most common of these methods, which was introduced in Section 2.3.3, is implemented as part of the *varimax* algorithm to orthogonal factor rotation. This algorithm uses the sum of the variances of the squared loadings in each column as a measure of complexity.

In R, we can use the apply() function to calculate the variance of the squared loadings in each column separately and then use the sum() function to find the sum of the resulting values.

```
sum(apply(X = (A ^ 2), MARGIN = 2, FUN = var))
[1] 0.06579354
```

However, unlike the mean of the Hoffman's complexity indices, this numerical summary actually measures the lack of complexity in the loading matrix. As such, the corresponding numerical optimization would be used to maximize rather than minimize this value.

Although varimax, and the underlying numerical summary of complexity, is the most common algorithm used to achieve orthogonal factor rotation, several alternatives have been devised. The *quartimax* algorithm, for example, uses the sum of the variances of the squared loadings in each row (as opposed to each column) as the measure of complexity.

```
sum(apply(X = (A ^ 2), MARGIN = 1, FUN = var))
[1] 1.077262
```

Note that we have set the `MARGIN` argument of the `apply()` function equal to 1 to specify that we want to apply the calculation of variances across each row rather than each column.

Conceptually, quartimax does for variables what varimax does for factors, ensuring that dimensions are easier to interpret. However, given that the main purpose of factor rotation is to simplify the factors, this algorithm is not used as frequently as the varimax one.

Another alternative is the *equamax* algorithm, which provides a compromise between the varimax and quartimax algorithms. This compromise is achieved by defining a weighted average between the sum of the variances of the squared loadings in each column and those in each row. Although there are an infinite number of possible weighting schemes that we could employ, the simplest by far is to weigh the two numerical summaries equally.

```
0.5 * sum(apply(X = (A ^ 2), MARGIN = 2, FUN = var)) +
  0.5 * sum(apply(X = (A ^ 2), MARGIN = 1, FUN = var))
[1] 0.5715277
```

Although the equamax algorithm would also seek to maximize this numerical summary, there have been reports that this approach behaves rather erratically if the intrinsic dimensionality is not fairly clear. Given that identifying intrinsic dimensionality is heuristic in nature and that different criteria often yield different results, this algorithm is rarely used in factor analysis.

A.2.2 | Numerical Optimization of Orthogonal Factor Rotation

Now that we have a better understanding of how to summarize the complexities in a loading matrix into a single numerical value, let's turn our attention back to how we can use numerical optimization to find either the maximum or the minimum of this values based on some input values.

Recall that for a two-factor representation of a data matrix, we can achieve orthogonal factor rotation by using a *projection* or *rotation matrix* $\boldsymbol{\Psi}$:

$$\boldsymbol{\Psi} = \begin{bmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{bmatrix} \tag{A.5}$$

This rotation matrix contains only sines and cosines of a *rotation angle* ψ, which are used to project the loadings on to a new set of factor axes. The new factor axes are rotated from the original by a counterclockwise rotation of ψ.

From the perspective of numerical optimization, this rotation angle serves as the input value. Any of the numerical summaries of complexity within the resulting loading matrix can then be used as the objective function. We can then systematically evaluate different rotation angles until the absolute smallest measure of complexity is achieved.

In order to demonstrate this computational approach, we first need to define an appropriate function in R to return the measure of complexity following an orthogonal rotation based on a single rotation angle to, for instance, the unrotated loading matrix for the two-factor representation of our example data.

```
rotatedComplexity <- function(psi = c(0)) {
   psi <- psi * pi / 180
   PSI <- matrix(c(cos(psi), -sin(psi),
                   sin(psi),  cos(psi)),
                 nrow = 2, ncol = 2, byrow = T)

   A1 <- A %*% PSI

   complexity <- sum(apply(X = (A1 ^ 2), MARGIN = 2, FUN = var))

   return(complexity)
}
```

Note that we have used the used the numerical summary employed by the varimax algorithm to quantify the (technically, lack of) complexity in the rotated loading matrix.

We can now use this function to compare, for example, whether a rotation of 10 degrees counterclockwise provides a less complex loading matrix than a rotation of 20 degrees.

```
rotatedComplexity(10)
[1] 0.04126817

rotatedComplexity(20)
[1] 0.04078508
```

Given that this function quantifies the lack of complexity, which implies that higher values are preferred, we can see that the 10-degree rotation provides a better solution than does the 20-degree rotation. The question then becomes how we can find the angle that provides the absolute largest lack of complexity possible.

In R, we can again use the psoptim() function in the pso package to perform particle swarm optimization to find this optimal solution.

```
library(pso)
psoptim(par = c(0), fn = rotatedComplexity, lower = 0, upper = 90,
        control = list(maxit = 10000, fnscale = -1))
$par
[1] 60.18413

$value
[1] -0.1475917

$counts
 function iteration  restarts 
   120000     10000         0 

$convergence
[1] 2

$message
[1] "Maximal number of iterations reached"
```

Note that in addition to increasing the maximum number of iterations by setting the maxi sub-argument of the control argument to 10000, we have also set the fnscale sub-argument to -1 to specify that, in this instance, we want to find the global maximum of this objective function rather than the global minimum.

Also note that rather than setting the lower and upper arguments, which provide bounded constrains to the optimization, to any ridiculously low and high number, we have set them equal to 0 and 90 to account for the fact that only a single quadrant in a full two-dimensional rotation provides a unique solution. Essentially, this avoids the complication that, for example, a rotation of 30 degrees is mathematically equivalent to a rotation of 210 degrees.

Recall that the $par aspect of the output identifies the value that the algorithm determined would produce the absolute largest function result, which in turn is contained in $value. Using our example output, we can see that a counterclockwise orthogonal rotation of approximately 60.2 degrees produces a loading matrix with a numerical lack of complexity of almost 0.15. Assuming that the algorithm was successful, then there is no angle that this loading matrix could be rotated by that would produce a more optimal result than that seen above.

It is important to note that in order to perform maximization rather than minimization of an objective function, the pso() function essentially attempts to find the minimum of the negative of the output value. As such, the result contained in $value is actually the negative of the objective function at the optimized input value, as we can confirm by using the optimized $par value in the original function.

```
rotatedComplexity(60.18413)
[1] 0.1475917
```

Now that we have identified the optimal rotation angle, we can use matrix algebra to derive the corresponding optimally orthogonally rotated loading matrix.

```
psi <- 60.18413 * pi / 180
PSI <- matrix(c(cos(psi), -sin(psi),
                sin(psi),  cos(psi)), nrow = 2, ncol = 2, byrow = T)

A1 <- A %*% PSI
A1
                        [,1]        [,2]
Hopeless           0.34773121 -0.74220178
Overwhelmed       -0.11961883 -0.75044196
Exhausted         -0.05918293 -0.76680606
Lonely             0.24360370 -0.76571210
Sad                0.26309650 -0.80190814
Depressed          0.46858459 -0.69531575
Anxious            0.22550688 -0.74508943
SelfHarming        0.78240947 -0.17322457
SuicidalThoughts   0.81960831 -0.24335096
SuicidalAttempts   0.80210838 -0.01404839
```

Comparing the correlations in this rotated loading matrix to those in our original unrotated loading matrix, we can see that the number of complex dimensions has decreased from five to two, with only feeling Hopeless and Depressed remaining complex. Although this is still not ideal in terms of interpreting and naming factors, it is a substantial improvement from the original unrotated two-factor solution.

A.2.3 | Orthogonal Factor Rotation in Higher Dimensionalities

Both the numerical optimization of orthogonal factor rotation in Section A.2.2 and the geometric and numerical representations of factor rotation originally introduced in Sections 2.3.1 and 2.3.3 were based on a two-factor solution. Although this solution was based on the commonly used Kaiser's criterion for identifying intrinsic dimensionality, other criteria may have suggested a higher number of factors.

Although the pca() function introduced in Section 2.3.3 can easily accommodate higher factor solutions, it is still important to understand how to expand our geometric and numerical understanding of factor rotation into these higher dimensionalities.

Geometrically, an orthogonal rotation of a three-factor solution is very similar to that of a two-factor solution. The only difference is that instead of two perpendicular factor axes there are now three perpendicular factor axes that can rotated about the origin in any direction. Unfortunately, expanding this visualization into a fourth dimension is simply not feasible given our inherent grounding in a three-dimensional world.

Expanding the numerical aspects of orthogonal factor rotation into three or more dimensions, on the other hand, is relatively straightforward. However, because these expansions become increasingly tedious as the intrinsic dimensionality continues to increase, we will focus our attention only on the three-dimensional representation of orthogonal factor rotation.

Consider the unrotated loading matrix for the three-factor representation of our example data, which corresponds to an intrinsic dimensionality of three as indicated by Jolliffe's criterion.

```
A <- pca(r = R, nfactors = 3, rotate = "none")$loadings[]
A
                       PC1         PC2         PC3
Hopeless         0.8168518 -0.06733209 -0.24766512
Overwhelmed      0.5916281 -0.47691509  0.51586452
Exhausted        0.6358756 -0.43261569  0.48496059
Lonely           0.7854763 -0.16936567 -0.22527293
Sad              0.8265731 -0.17045037 -0.22582291
Depressed        0.8362622  0.06083607 -0.28483875
Anxious          0.7585855 -0.17481308 -0.11202108
SelfHarming      0.5393196  0.59271045  0.20510537
SuicidalThoughts 0.6186590  0.59011735  0.08441262
SuicidalAttempts 0.4110085  0.68894644  0.27414105
```

We can orthogonally rotate this loading matrix by using an expanded rotation matrix $\boldsymbol{\Psi}$:

$$\boldsymbol{\Psi} = \begin{bmatrix} \cos\psi_1 \cos\psi_2 & -\cos\psi_1 \sin\psi_2 & \sin\psi_1 \\ \sin\psi_2 & \cos\psi_2 & 0 \\ -\sin\psi_1 \cos\psi_2 & \sin\psi_1 \sin\psi_2 & -\cos\psi_1 \end{bmatrix} \tag{A.6}$$

This rotation matrix contains only sines and cosines of two separate rotation angle ψ_1 and ψ_2, which represent rotations around the horizontal and vertical planes.

Once this rotation matrix is constructed in R, we can use the same approach as before to calculate the rotated loading matrix based on a set of two rotation angles. By implementing this series of calculations within a function that returns some corresponding measure of complexity, we can again use particle swarm optimization to find the optimized set of rotation angles that would produce the least complex loading matrix possible.

Although functions such as `pca()` can always be used to quickly identify these and other higher order optimal solutions to orthogonal and even oblique factor rotations, a thorough understanding of how they operate numerically is still quite useful!

Exercises for Section A.2

A.03 Cavity Trees 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	-0.73	-0.09	-0.38
Type	-0.73	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 and find the unrotated loading matrix corresponding to a two-factor representation of these data (as is supported by Kaiser's criterion)
b. Use function() to create a function that returns the mean Hoffman's index of complexity of the loading matrix following a two-dimensional orthogonal factor rotation based on a single rotation angle
c. Use this function to find the mean index of complexity for both a 20-degree and a 70-degree orthogonal rotation
d. Explain which of the two rotations provided a more pure alignment between factor axes and the clouds of points from the original dimensions
e. Use particle swarm optimization via psoptim() to find the orthogonal rotation that produces the absolute smallest mean index of complexity possible

A.04 Cavity Trees 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	-0.73	-0.09	-0.38
Type	-0.73	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 and find the unrotated loading matrix corresponding to a three-factor representation of these data
b. Use `function()` to create a function that returns the mean Hoffman's index of complexity of the loading matrix following a three-dimensional orthogonal factor rotation based on a concatenated set of two rotation angles
c. Use particle swarm optimization via `psoptim()` to find the orthogonal rotation that produces the absolute smallest mean index of complexity possible
d. Find the corresponding optimally orthogonally rotated loading matrix
e. Explain if there appears to be any remaining complexity among the original dimensions following this rotation