Chapter 2

Exploratory Factor Analysis

In experimental design, the term "factor" refers to a categorical explanatory variable. In structural equation modeling and in the sub-field of factor analysis, a *factor* is a latent variable, period. Factor analysis may be said to originate with a 92-page article [60] by Charles Spearman in the 1901 *American Journal of Psychology*, entitled "General intelligence, objectively determined and measured." If you believe that some people are generally smarter than others, the basic idea is quite natural. True intelligence cannot be directly observed, so it's a latent variable. However, we can observe performance on various tests and puzzles. Spearman proposed that the correlations among observable variables arise from their connection to a common "g" factor — general intelligence.

The early history of factor analysis is described masterfully in Harman's (1960, 1967, 1976) classic *Modern factor analysis* [28]. Though Harman brings relative clarity to this murky literature, his book is almost guaranteed to be frustrating for a statistician to read. Lawley and Maxwell's (1971) *Factor analysis as a statistical method* is a welcome antidote. Bastlevsky's (1994) *Statistical factor analysis and related methods* [2] is a strong and more recent treatment of the topic.

Factor analysis may be divided into two types, commonly called *exploratory* factor analysis and *confirmatory* factor analysis. The books cited above are about exploratory factor analysis, which came first historically. While both types of factor analysis are special cases of structural equation models, it is confirmatory factor analysis that provides a useful measurement model. Exploratory factor analysis is helpful for understanding confirmatory factor analysis. Another good reason to learn about exploratory factor analysis is that some people still do it, or may ask you to do it.

2.1 Principal Components Analysis

Before describing what factor analysis is, it will be helpful to describe what it is not. Principal components analysis is not factor analysis. Factors are unobservable latent variables. Principal components are linear combinations of the sample data. The very existence of factors depends on one's acceptance of a fairly elaborate statistical model, while the statistical model underlying principal components is quite minimal, if there is

one at all. Still, principal components analysis and factor analysis have a similar flavour, and some of the ideas from principal components are used in factor analysis.

The main application of principal components analysis is data reduction. Suppose you have a large number of variables that are correlated with one another. Principal components analysis allows you to find a smaller set of linear combinations of the variables, linear combinations that contain most of the variation in the original set. It may be that little is lost by using the linear combinations in place of the original variables, and there can be substantial advantages in terms of storage and processing.

In the most relevant version of principal components, there are k observable variables that are standardized[1], by subtracting off their means and dividing by their standard deviations. Collect the variables into a k-dimensional random vector $\mathbf{z} = [z_j]$, with $E(\mathbf{z}) = \mathbf{0}$ and $cov(\mathbf{z}) = \mathbf{\Sigma}$. Because of standardization, $\mathbf{\Sigma}$ is a correlation matrix.

Recall the spectral decomposition $\mathbf{\Sigma} = \mathbf{CDC}^\top$ (see Section A.2 in Appendix A), where $\mathbf{D}$ is a diagonal matrix containing the k eigenvalues of $\mathbf{\Sigma}$ in descending order, and the columns of the $k \times k$ matrix $\mathbf{C} = [c_{ij}]$ contain the corresponding eigenvectors. The eigenvectors are orthonormal, so $\mathbf{CC}^\top = \mathbf{C}^\top\mathbf{C} = \mathbf{I}$.

Let $\mathbf{y} = \mathbf{C}^\top\mathbf{z} = [y_j]$. The transformed variables in $\mathbf{y}$ will be called the *principal components* of $\mathbf{z}$. Immediately, we have $E(\mathbf{y}) = \mathbf{0}$ and

$$\begin{aligned} cov(\mathbf{y}) &= cov(\mathbf{C}^\top\mathbf{z}) \\ &= \mathbf{C}^\top cov(\mathbf{z})\mathbf{C} \\ &= \mathbf{C}^\top\mathbf{\Sigma}\mathbf{C} \\ &= \mathbf{C}^\top\,\mathbf{CDC}^\top\,\mathbf{C} \\ &= \mathbf{D}, \end{aligned} \tag{2.1}$$

so that the elements of $\mathbf{y}$ are uncorrelated, and their variances are the eigenvalues of $\mathbf{\Sigma}$, sorted from largest to smallest.

Since $\mathbf{y} = \mathbf{C}^\top\mathbf{z}$, we can also write the original variables in terms of the principal components as $\mathbf{z} = \mathbf{Cy}$. In scalar form,

$$\begin{aligned} z_1 &= c_{11}y_1 + c_{12}y_2 + \cdots + c_{1k}y_k \\ z_2 &= c_{21}y_1 + c_{22}y_2 + \cdots + c_{2k}y_k \\ \vdots & \qquad\qquad \vdots \\ z_k &= c_{k1}y_1 + c_{k2}y_2 + \cdots + c_{kk}y_k. \end{aligned}$$

Because the elements of $\mathbf{y}$ are uncorrelated, the variance of variable j is

$$\begin{aligned} Var(z_j) &= Var(c_{j1}y_1 + c_{j2}y_2 + \cdots + c_{jk}y_k) \\ &= c_{j1}^2 Var(y_1) + c_{j2}^2 Var(y_2) + \cdots + c_{jk}^2 Var(y_k) \\ &= c_{j1}^2\lambda_1 + c_{j2}^2\lambda_2 + \cdots + c_{jk}^2\lambda_k = 1. \end{aligned} \tag{2.2}$$

[1] In the other main version of principal components, the variables are not standardized. The development is very similar.

Thus, the variance of z_j is decomposed into the part explained by y_1, the part explained by y_2, and so on. Specifically, y_1 explains $c_{j1}^2\lambda_1$ of the variance, y_2 explains $c_{j2}^2\lambda_2$ of the variance, etc.. Because z_j is standardized, these are *proportions* of variance.

They are also squared correlations. Correlation is covariance divided by the product of standard deviations. Using the fact that $cov(y_i, y_j) = 0$ for $i \neq j$,

$$\begin{aligned} Cov(z_i, y_j) &= Cov(c_{i1}y_1 + c_{i2}y_2 + \cdots + c_{ij}y_j + \cdots + c_{jk}y_k, y_j) \\ &= c_{ij}Cov(y_j, y_j) \\ &= c_{ij}\lambda_j. \end{aligned}$$

Then,

$$\begin{aligned} Corr(z_i, y_j) &= \frac{Cov(z_i, y_j)}{SD(z_i)SD(y_j)} \\ &= \frac{c_{ij}\lambda_j}{1\ \sqrt{\lambda_j}} = c_{ij}\sqrt{\lambda_j}, \end{aligned} \tag{2.3}$$

and the *squared* correlation between z_i and y_j is $c_{ij}^2\lambda_j$.

Looking at the variances of all the original variables,

$$\begin{aligned} Var(z_1) &= c_{11}^2\lambda_1 + c_{12}^2\lambda_2 + \cdots + c_{1k}^2\lambda_k \\ Var(z_2) &= c_{21}^2\lambda_1 + c_{22}^2\lambda_2 + \cdots + c_{2k}^2\lambda_k \\ \vdots & \qquad\qquad \vdots \\ Var(z_k) &= c_{k1}^2\lambda_1 + c_{k2}^2\lambda_2 + \cdots + c_{kk}^2\lambda_k. \end{aligned} \tag{2.4}$$

The pieces of variance being added up are the squared correlations between the original variables and the principal components.

Imagine a $k \times k$ matrix of these squared correlations, with the original variables corresponding to rows, and the principal components corresponding to columns. The layout is the same as the equations (2.4). If you add the entries in any row, you get one. If you add the entries in a column, you get the total amount of variance in the original variables that is explained by that principal component. The sum of entries in column j is

$$\begin{aligned} \sum_{i=1}^{k} c_{ij}^2\lambda_j &= \lambda_j \sum_{i=1}^{k} c_{ij}^2 \\ &= \lambda_j \cdot 1 = \lambda_j, \end{aligned} \tag{2.5}$$

where the squared weights add to one because the eigenvectors are of unit length. This means that the eigenvalues are both the variances of the principal components and the amounts of variance in the original variables that are explained by the respective principal components. The total variance in the original variables is the trace of $\mathbf{\Sigma}$, which equals k. The trace of a symmetric matrix is the sum of its eigenvalues, and everything adds up.

It's actually even better than that. There is a well-known theorem saying that y_1 has the greatest possible variance of any linear combination whose squared weights add up to

one. In addition, y_2 is the linear combination that has the greatest variance subject to the constraints that it's orthogonal to y_1 and its squared weights add to one. Continuing, y_3 is the linear combination that has the greatest variance subject to the constraints that it's orthogonal to y_1 and y_2, and its squared weights add to one — and so on. This means that the principal components are optimal in the sense that the first one explains the greatest possible amount of variance, and all the succeeding components explain the greatest possible amounts of the variance that remains unexplained by the earlier ones.

If the correlations among the original variables are substantial, the first few eigenvalues will be relatively large. The data reduction idea is to retain only the first several principal components, the ones that contain most of the variation in the original variables. The expectation is that they will capture most of the *meaningful* variation.

To apply this method to actual data, suppose you have n observations on k variables. First standardize all the variables, by subtracting off sample means and dividing by sample standard deviations. Assemble the standardized data into an $n \times k$ matrix $\mathbf{Z} = [z_{ij}]$. The true correlation matrix $\boldsymbol{\Sigma}$ is unknown, so use the sample correlation matrix $\widehat{\boldsymbol{\Sigma}}$. Based upon the spectral decomposition $\widehat{\boldsymbol{\Sigma}} = \widehat{\mathbf{C}}\widehat{\mathbf{D}}\widehat{\mathbf{C}}^\top$, calculate $\widehat{\mathbf{Y}} = \mathbf{Z}\widehat{\mathbf{C}}$. The rows of $\mathbf{Z}$ contain standardized data vectors, and the rows of $\widehat{\mathbf{Y}}$ contain the corresponding vectors of principal component values. $\widehat{\mathbf{Y}}$ has a hat because it is a matrix of the *sample* principal components. It can be informative to look at a matrix of squared sample correlations between the original variables and the components, because the entries are estimated proportions of variance in each variable that are explained by each component.

A nice feature of principal components is that the formulas given earlier in this section are exactly correct for sample principal components. This is because most of the rules for variances and covariances are also true for the sample versions[2]. As a result, it is possible to present principal components analysis as a purely descriptive procedure, without assuming any sampling model at all. Some textbooks do it this way; it's a matter of taste.

In any case, the main application of principal components is data reduction. The data reduction strategy is to retain just a few columns of $\widehat{\mathbf{Y}}$, because those principal components account for most of the variance in the original variables. But where do you draw the line? How many principal components should you preserve? A standard answer is to keep the components with eigenvalues greater than one, because one is the amount of variance in a single original variable. After that point, the principal components explain no more variance than the original variables.

Example 2.1.1 *The Body-Mind Data*

[2]This statement is true and it's good enough, but here is an another way of thinking about it. The formulas developed for principal components are true for any distribution of the observed data. In particular, they are true for the rather peculiar discrete multivariate distribution that puts probability $\frac{1}{n}$ on each observed data vector. Think of the observed data vectors as strings of beads in an urn. We are sampling from this urn with replacement. It's the re-sampling model that is used in the bootstrap! For this distribution, the population mean, variance, covariance and so on may be calculated using usual formulas for the corresponding sample moments – provided that one uses the variance and covariance formulas with n in the denominator rather than $n-1$. Consequently, all the formulas derived here apply directly to sample principal components.

The Body-Mind data are a set of educational test scores and physical measurements for a sample of high school students[3]. The variables are

- `sex`: F or M.
- `progmat`: Progressive matrices (puzzle) score.
- `reason`: Reasoning score.
- `verbal`: Verbal (reading and vocabulary) score.
- `headlng`: Head Length in mm.
- `headbrd`: Head Breadth in mm.
- `headcir`: Head Circumference in mm.
- `bizyg`: Bizygomatic breadth in mm, basically how far apart the eyes are.
- `weight`: In pounds.
- `height`: In cm.

These data will be used to illustrate true factor analysis as well as principal components. We begin by reading the data, and looking at basic descriptive statistics and the correlation matrix.

```
> rm(list=ls())
> bodymind = read.table('http://www.utstat.toronto.edu/~brunner/openSEM/data/bodymind.data.txt')
> head(bodymind)
  sex progmat reason verbal headlng headbrd headcir bizyg weight height
1   M     108    128    136     182     162     553   140    144   1769
2   F      81    110     94     192     156     571   143    144   1633
3   F     110    134    132     186     145     549   131    135   1672
4   F      95     88     83     189     139     536   124    109   1700
5   M      83     94    100     180     163     549   141    124   1679
6   M     105     77     92     195     148     560   134    126   1651
> dim(bodymind) # Number of rows,columns
[1] 80 10
dat = as.matrix(bodymind[,2:10]) # Omit sex, make dat a matrix rather than a data frame.
> # summary(dat)
> Sigma_hat = cor(dat); round(Sigma_hat,3)
        progmat reason verbal headlng headbrd headcir bizyg weight height
progmat   1.000  0.514  0.539   0.323   0.099   0.315 0.200  0.132  0.197
reason    0.514  1.000  0.728   0.203   0.053   0.322 0.291  0.171  0.207
verbal    0.539  0.728  1.000   0.260   0.139   0.354 0.337  0.236  0.199
headlng   0.323  0.203  0.260   1.000   0.255   0.821 0.475  0.506  0.554
```

[3]This is a modified subset of data reported in the journal *Human Biology* [17]. The data are used here without permission, but I believe they have been sufficiently hacked so that the original copyright no longer applies, and they can be protected under a Creative Commons license. Good luck trying to recover the original data values.

```
headbrd   0.099  0.053  0.139   0.255   1.000   0.604 0.692  0.368  0.362
headcir   0.315  0.322  0.354   0.821   0.604   1.000 0.713  0.641  0.591
bizyg     0.200  0.291  0.337   0.475   0.692   0.713 1.000  0.589  0.614
weight    0.132  0.171  0.236   0.506   0.368   0.641 0.589  1.000  0.599
height    0.197  0.207  0.199   0.554   0.362   0.591 0.614  0.599  1.000
```

The R functions `princomp` and `prcomp` will do principal components analysis, but we'll use spectral decomposition directly at first for illustrative purposes. The `eigen` function returns a list with two elements. The first element is a vector of eigenvalues, and the second element is the matrix $\mathbf{C}$ in $\mathbf{A} = \mathbf{CDC}^\top$. Column j of the matrix $\mathbf{C}$ is the eigenvector corresponding to λ_j.

```
> eigenSigma = eigen(Sigma_hat); eigenSigma
eigen() decomposition
$values
[1] 4.28768216 1.77444482 0.87126975 0.64039055 0.47989427 0.40504511 0.26315906
[8] 0.21010253 0.06801175

$vectors
            [,1]        [,2]        [,3]       [,4]        [,5]        [,6]        [,7]
 [1,] -0.2274301 -0.47286949  0.10386693 -0.5037581  0.59758999 -0.29003259 -0.08152051
 [2,] -0.2405745 -0.54632083 -0.12052776  0.2965707 -0.17975676  0.26454653 -0.63108107
 [3,] -0.2665589 -0.51874379 -0.16430737  0.1996142 -0.24123089 -0.07990715  0.71935768
 [4,] -0.3622340  0.08683821  0.53544154 -0.3586357 -0.34767275  0.16737858  0.08869681
 [5,] -0.2933333  0.27697281 -0.66373737 -0.3094155  0.04112189 -0.03633303 -0.01019606
 [6,] -0.4377198  0.12657178  0.08577647 -0.2525368 -0.33524350  0.01081660 -0.14979213
 [7,] -0.4007471  0.17219323 -0.34669164  0.1054639  0.05971130  0.13277579  0.01297777
 [8,] -0.3513037  0.20963075  0.18723810  0.4548388  0.02650610 -0.74034211 -0.13982797
 [9,] -0.3556358  0.18698153  0.24048299  0.3351036  0.55960504  0.49428994  0.16579073
             [,8]         [,9]
 [1,]  0.10288021  0.040581025
 [2,] -0.11345554 -0.166563741
 [3,] -0.09839457  0.035693597
 [4,]  0.07347486 -0.532701450
 [5,] -0.44524651 -0.315589722
 [6,] -0.14639593  0.751580920
 [7,]  0.81059576 -0.002188321
 [8,] -0.07609981 -0.128655279
 [9,] -0.28094584  0.067358086
```

Since only the first two eigenvalues are greater than one, the conventional choice for data reduction would be to retain only the first two sample principal components. Dividing the eigenvalues by the number of variables yields the proportions of the total variance explained by each component.

```
> lambda_hat = eigenSigma$values
> lambda_hat/9         # Proportions of explained variance
[1] 0.476409129 0.197160535 0.096807750 0.071154506 0.053321586 0.045005012 0.029239896
[8] 0.023344726 0.007556861
> cumsum(lambda_hat/9) # Cumulative sum
[1] 0.4764091 0.6735697 0.7703774 0.8415319 0.8948535 0.9398585 0.9690984 0.9924431
[9] 1.0000000
```

It seems that the first two components account for around 67% of the total variance in the observed variables, and five components would account for about 90%.

Calculating $\mathbf{Z}$ and then $\widehat{\mathbf{Y}} = \mathbf{Z}\widehat{\mathbf{C}}$, we verify (2.1), which says $cov(\mathbf{y}) = \mathbf{D}$.

```
> > Z = scale(dat) # Standardize
> C_hat = eigenSigma$vectors #$
> Y_hat = Z %*% C_hat # Sample principal components
> # Looking at the variance-covariance matrix of the principal components,
> round(var(Y_hat), 4) # Should equal D
       [,1]   [,2]   [,3]   [,4]   [,5]  [,6]   [,7]   [,8]  [,9]
 [1,] 4.2877 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.000
 [2,] 0.0000 1.7744 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.000
 [3,] 0.0000 0.0000 0.8713 0.0000 0.0000 0.000 0.0000 0.0000 0.000
 [4,] 0.0000 0.0000 0.0000 0.6404 0.0000 0.000 0.0000 0.0000 0.000
 [5,] 0.0000 0.0000 0.0000 0.0000 0.4799 0.000 0.0000 0.0000 0.000
 [6,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.405 0.0000 0.0000 0.000
 [7,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.2632 0.0000 0.000
 [8,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.2101 0.000
 [9,] 0.0000 0.0000 0.0000 0.0000 0.0000 0.000 0.0000 0.0000 0.068
```

There it is: a diagonal matrix with the eigenvalues on the diagonal.

Based on the eigenvalues, let's retain just the first two components and estimate how much variance they explain. First, look at the correlations.

```
> y = Y_hat[,1:2] # Just the first two components
> zy = cor(Z,y); zy
              [,1]       [,2]
progmat -0.4709330 -0.6299014
reason  -0.4981509 -0.7277446
verbal  -0.5519561 -0.6910097
headlng -0.7500678  0.1156757
headbrd -0.6073970  0.3689507
headcir -0.9063741  0.1686041
bizyg   -0.8298157  0.2293757
weight  -0.7274347  0.2792455
height  -0.7364050  0.2490749
```

All of the large correlations are negative, so they are a bit harder to look at. If this is a problem, the signs of a principal component can be flipped, reversing the signs of the correlation between that component and any variable. To see why this is true, recall the definition of an eigenvalue and associated eigenvector: $\mathbf{Ax} = \lambda\mathbf{x}$. Clearly if $\mathbf{x}$ is an eigenvector corresponding to λ, so is $-\mathbf{x}$. Since a principal component is a linear combination of variables whose weights are the elements of an eigenvector, the sign is arbitrary.

Now we will check Equation (2.3), which says $Corr(z_i, y_j) = c_{ij}\sqrt{\lambda_j}$. We should be able to reproduce the matrix of correlations between $\mathbf{Z}$ and the first two components by multiplying the first two columns of $\widehat{\mathbf{C}}$ by the matrix $\begin{pmatrix} \sqrt{\widehat{\lambda}_1} & 0 \\ 0 & \sqrt{\widehat{\lambda}_2} \end{pmatrix}$.

```
> A = rbind(c( sqrt(lambda_hat[1]), 0 ),
```

```
+           c(0, sqrt(lambda_hat[2]) )  )
> C_hat[,1:2] %*% A
            [,1]       [,2]
 [1,] -0.4709330 -0.6299014
 [2,] -0.4981509 -0.7277446
 [3,] -0.5519561 -0.6910097
 [4,] -0.7500678  0.1156757
 [5,] -0.6073970  0.3689507
 [6,] -0.9063741  0.1686041
 [7,] -0.8298157  0.2293757
 [8,] -0.7274347  0.2792455
 [9,] -0.7364050  0.2490749
```

Okay, it worked: Estimated $Corr(z_i, y_j)$ is $\widehat{c}_{ij}\sqrt{\widehat{\lambda}_j}$.

The squared correlations are components of variance. The `addmargins` function is used below to add row and column sums. It's easier to look at the output rounded to three decimal places.

```
> zy2 = zy^2
> round( addmargins(zy2, margin = c(1,2), FUN = sum) , 3)

Margins computed over dimensions
in the following order:
1:
2:
                      sum
progmat 0.222 0.397 0.619
reason  0.248 0.530 0.778
verbal  0.305 0.477 0.782
headlng 0.563 0.013 0.576
headbrd 0.369 0.136 0.505
headcir 0.822 0.028 0.850
bizyg   0.689 0.053 0.741
weight  0.529 0.078 0.607
height  0.542 0.062 0.604
sum     4.288 1.774 6.062
```

This shows, for example, that the first principal component explains 54.2% of the variance in height, and the second principal component explains an additional 6.2%. The first two principal components explain around 85% of the variance in head circumference, but only about 50.5% of the variance in head breadth. Also, the column totals are the eigenvalues, as in (2.5). These are all *estimated* values, of course.

Principal components the easy way It's a bit easier to use a specialized R function for principal components analysis, rather than relying on `eigen`. I prefer `prcomp` over `princomp`, because `princomp` has some unfortunate features that have been retained for compatibility with the defunct commercial software S-plus.

In the `prcomp` function, the `scale = T` option divides variables by their sample standard deviations. The option `center` is true by default, so the data are converted to z-scores. This is what we want.

```
> # Principal components the easy way
> # help(prcomp)
> pc = prcomp(dat, scale = T)
```

The object `pc` is a list. The `ls` function shows its elements.

```
> ls(pc)
[1] "center"   "rotation" "scale"    "sdev"     "x"
```

The element `pc$center` contains the sample means of the variables before standardization; `pc$scale` contains the standard deviations. `sdev` has the standard deviations of the components. Squaring the `sdev` vector yields the eigenvalues of the sample correlation matrix.

```
> pc$sdev^2 # Eigenvalues
[1] 4.28768216 1.77444482 0.87126975 0.64039055 0.47989427 0.40504511 0.26315906
[8] 0.21010253 0.06801175
> lambda_hat # For comparison
[1] 4.28768216 1.77444482 0.87126975 0.64039055 0.47989427 0.40504511 0.26315906
[8] 0.21010253 0.06801175
```

The list element `pc$rotation` corresponds to the $\widehat{\mathbf{C}}$ matrix produced by the spectral decomposition. Since $\widehat{\mathbf{C}}$ is an orthogonal matrix, it is indeed a rotation.

```
> pc$rotation
                PC1         PC2         PC3         PC4         PC5         PC6         PC7
progmat -0.2274301 -0.47286949  0.10386693 -0.5037581  0.59758999 -0.29003259  0.08152051
reason  -0.2405745 -0.54632083 -0.12052776  0.2965707 -0.17975676  0.26454653  0.63108107
verbal  -0.2665589 -0.51874379 -0.16430737  0.1996142 -0.24123089 -0.07990715 -0.71935768
headlng -0.3622340  0.08683821  0.53544154 -0.3586357 -0.34767275  0.16737858 -0.08869681
headbrd -0.2933333  0.27697281 -0.66373737 -0.3094155  0.04112189 -0.03633303  0.01019606
headcir -0.4377198  0.12657178  0.08577647 -0.2525368 -0.33524350  0.01081660  0.14979213
bizyg   -0.4007471  0.17219323 -0.34669164  0.1054639  0.05971130  0.13277579 -0.01297777
weight  -0.3513037  0.20963075  0.18723810  0.4548388  0.02650610 -0.74034211  0.13982797
height  -0.3556358  0.18698153  0.24048299  0.3351036  0.55960504  0.49428994 -0.16579073
                PC8          PC9
progmat -0.10288021  0.040581025
reason   0.11345554 -0.166563741
verbal   0.09839457  0.035693597
headlng -0.07347486 -0.532701450
headbrd  0.44524651 -0.315589722
headcir  0.14639593  0.751580920
bizyg   -0.81059576 -0.002188321
weight   0.07609981 -0.128655279
height   0.28094584  0.067358086
```

```
> C_hat # For comparison
            [,1]        [,2]        [,3]       [,4]        [,5]        [,6]        [,7]
 [1,] -0.2274301 -0.47286949  0.10386693 -0.5037581  0.59758999 -0.29003259 -0.08152051
 [2,] -0.2405745 -0.54632083 -0.12052776  0.2965707 -0.17975676  0.26454653 -0.63108107
 [3,] -0.2665589 -0.51874379 -0.16430737  0.1996142 -0.24123089 -0.07990715  0.71935768
 [4,] -0.3622340  0.08683821  0.53544154 -0.3586357 -0.34767275  0.16737858  0.08869681
 [5,] -0.2933333  0.27697281 -0.66373737 -0.3094155  0.04112189 -0.03633303 -0.01019606
```

```
 [6,] -0.4377198  0.12657178  0.08577647 -0.2525368 -0.33524350  0.01081660 -0.14979213
 [7,] -0.4007471  0.17219323 -0.34669164  0.1054639  0.05971130  0.13277579  0.01297777
 [8,] -0.3513037  0.20963075  0.18723810  0.4548388  0.02650610 -0.74034211 -0.13982797
 [9,] -0.3556358  0.18698153  0.24048299  0.3351036  0.55960504  0.49428994  0.16579073
             [,8]         [,9]
 [1,]  0.10288021  0.040581025
 [2,] -0.11345554 -0.166563741
 [3,] -0.09839457  0.035693597
 [4,]  0.07347486 -0.532701450
 [5,] -0.44524651 -0.315589722
 [6,] -0.14639593  0.751580920
 [7,]  0.81059576 -0.002188321
 [8,] -0.07609981 -0.128655279
 [9,] -0.28094584  0.067358086
```

Finally, `pc$x` has the principal components themselves.

```
> dim(pc$x) # x is a matrix of the principal components Y_hat = Z %*% C_hat
[1] 80  9
> head(pc$x) # Just the first 6 rows
         PC1        PC2         PC3         PC4        PC5         PC6         PC7
1 -2.9056790 -0.8163483 -2.05648959  0.89345100  1.0826163  0.09581676  0.09097201
2 -1.8420248  1.6868136 -1.11946332  0.61460425 -1.7388326  0.20651893  0.68366132
3 -1.1270571 -2.3088592  0.08809617  0.75714079  0.1575711 -0.19017485  0.51153249
4  1.6221315  0.2340440  1.62777485  0.07639917  0.3896938  0.65365783 -0.33948607
5 -0.6431189  1.8507668 -2.60883792  0.58933649 -0.1899104  0.47035165 -0.33536456
6 -0.5757390  0.9010777  0.79544134 -1.28687495  0.1150836 -0.39632242 -0.59876963
          PC8         PC9
1  0.66523233 -0.13093412
2 -0.60878367  0.09346307
3  0.34061367  0.13503816
4  0.44387949  0.16416604
5  0.06955952  0.02486900
6 -0.48127952  0.34279834
> head(Y_hat) # For comparison
        [,1]       [,2]        [,3]        [,4]       [,5]        [,6]        [,7]
1 -2.9056790 -0.8163483 -2.05648959  0.89345100  1.0826163  0.09581676 -0.09097201
2 -1.8420248  1.6868136 -1.11946332  0.61460425 -1.7388326  0.20651893 -0.68366132
3 -1.1270571 -2.3088592  0.08809617  0.75714079  0.1575711 -0.19017485 -0.51153249
4  1.6221315  0.2340440  1.62777485  0.07639917  0.3896938  0.65365783  0.33948607
5 -0.6431189  1.8507668 -2.60883792  0.58933649 -0.1899104  0.47035165  0.33536456
6 -0.5757390  0.9010777  0.79544134 -1.28687495  0.1150836 -0.39632242  0.59876963
         [,8]        [,9]
1 -0.66523233 -0.13093412
2  0.60878367  0.09346307
3 -0.34061367  0.13503816
4 -0.44387949  0.16416604
5 -0.06955952  0.02486900
6  0.48127952  0.34279834
```

A useful feature of `prcomp` is that it's easy to specify the number of components you want to extract. This is accomplished by specifying `rank` in the call to `prcomp`.

```
> pc2 = prcomp(dat, scale = T, rank = 2) # Retain two principal components
```

```
> pc2$rotation
               PC1         PC2
progmat -0.2274301 -0.47286949
reason  -0.2405745 -0.54632083
verbal  -0.2665589 -0.51874379
headlng -0.3622340  0.08683821
headbrd -0.2933333  0.27697281
headcir -0.4377198  0.12657178
bizyg   -0.4007471  0.17219323
weight  -0.3513037  0.20963075
height  -0.3556358  0.18698153
```

Only the first two columns of $\widehat{\mathbf{C}}$ are returned. Post-multiplying this matrix by the matrix of standardized data in $\mathbf{Z}$ yields an 80×2 matrix of just the first two principal components.

```
> head(pc2$x) # There should be 2 columns
         PC1        PC2
1 -2.9056790 -0.8163483
2 -1.8420248  1.6868136
3 -1.1270571 -2.3088592
4  1.6221315  0.2340440
5 -0.6431189  1.8507668
6 -0.5757390  0.9010777
```

This is all very nice, but it's not factor analysis. Principal components analysis and factor analysis are frequently confused, especially by social scientists. In a consulting situation, suppose your client claims to have done a factor analysis. You should ask "What kind of factor analysis?" If the client doesn't know, ask "What software did you use?" If it's SAS or SPSS, ask "Did you use the default options?" If the answer is yes, it was a principal components analysis. We now turn to true factor analysis.

2.2 True Factor Analysis

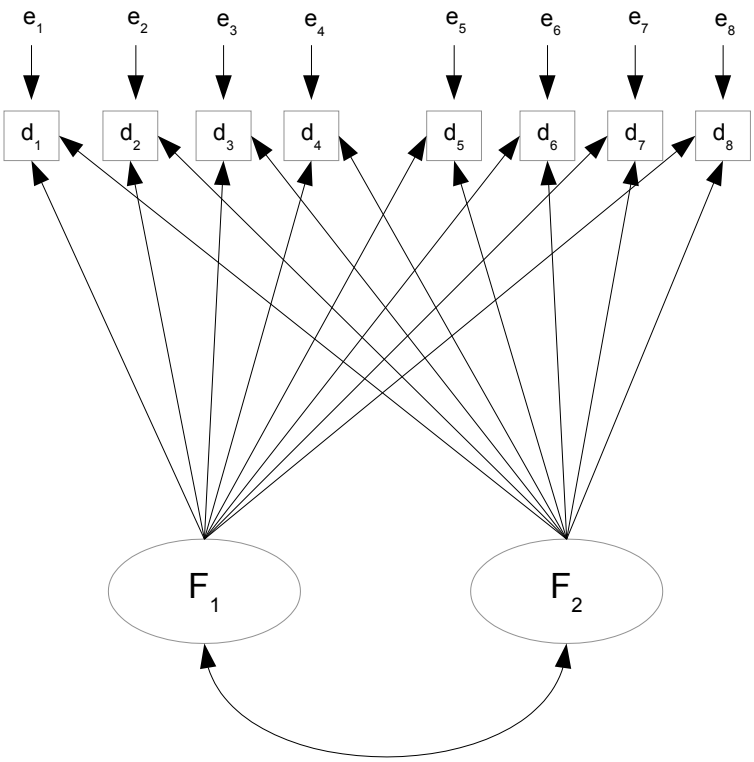
In exploratory factor analysis, the goal is to describe and summarize a data set by explaining a set of observed variables in terms of a smaller number of latent variables (factors). The factors are the reason the observable variables have the correlations they do. Figure 2.1 shows the path diagram of a model with two factors and eight observable variables. A common rule is at least three observable variables for each factor. In general, the more variables for each factor, the better.

The general factor analysis model may be written as follows. Independently for $i = 1, \ldots, n$, let

$$\mathbf{d}_i = \mathbf{\Lambda}\mathbf{F}_i + \mathbf{e}_i, \tag{2.6}$$

where $\mathbf{d}_i$ is a $k \times 1$ observable random vector, $\mathbf{\Lambda}$ is a $k \times p$ matrix of constants, and $\mathbf{F}_i$ (F for factor) is a $p \times 1$ latent random vector with covariance matrix $\mathbf{\Phi}$. The $k \times 1$ vector of error terms $\mathbf{e}_i$ is independent of $\mathbf{F}_i$; it has expected value zero and covariance matrix $\mathbf{\Omega}$,

Figure 2.1: A Two-factor Model



which is almost always assumed to be diagonal[4]. There are no intercepts, and $E(\mathbf{F}_i) = \mathbf{0}$. This is a centered surrogate model (see Section A.6.1). The notation here is consistent with the general two-stage model of Section 1.2, except that there, the dimension of $\mathbf{F}_i$ would be $(p+q) \times 1$. A multivariate normal assumption for $\mathbf{F}_i$ and $\mathbf{e}_i$ is common.

[4]The assumption that $\mathbf{\Omega}$ is diagonal helps with identifiability, and may be traced to what Spearman [60] (1904, p. 273) calls the "Law of the Universal Unity of the Intellective Function," to wit: *Whenever branches of intellectual activity are at all dis-similar, then their correlations with one another appear wholly due to their being all variously saturated with some common fundamental Function (or group of Functions* as well as positive definite. Note that in Figure 2.1, $\mathbf{\Omega}$ being diagonal corresponds to a lack of any curved, double-headed arrows connecting $e_1, \ldots, e_8$. This means that any correlations between observable variables must come from the factors.

To clarify the notation, the model equations for Figure 2.1 are

$$\begin{array}{ccccccc} \mathbf{d}_i & = & \mathbf{\Lambda} & & \mathbf{F}_i & + & \mathbf{e}_i \end{array}$$

$$\begin{pmatrix} d_{i,1} \\ d_{i,2} \\ d_{i,3} \\ d_{i,4} \\ d_{i,5} \\ d_{i,6} \\ d_{i,7} \\ d_{i,8} \end{pmatrix} = \begin{pmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \\ \lambda_{31} & \lambda_{32} \\ \lambda_{41} & \lambda_{42} \\ \lambda_{51} & \lambda_{52} \\ \lambda_{61} & \lambda_{62} \\ \lambda_{71} & \lambda_{27} \\ \lambda_{81} & \lambda_{82} \end{pmatrix} \begin{pmatrix} F_{i,1} \\ F_{i,2} \end{pmatrix} + \begin{pmatrix} e_{i,1} \\ e_{i,2} \\ e_{i,3} \\ e_{i,4} \\ e_{i,5} \\ e_{i,6} \\ e_{i,7} \\ e_{i,8} \end{pmatrix}. \tag{2.7}$$

The λ_{ij} values will be called *factor loadings*. They are essentially regression coefficients linking the factors to the observed variables[5]. The factors $F_{i,1}$ and $F_{i,2}$ are sometimes called *common factors*, because they influence all the observed variables; all the observed variables have them in common. The error terms $e_{i,1}, \ldots, e_{i,8}$ are sometimes called *unique factors*, because each one influences only a single observed variable.

The defining feature of exploratory factor analysis is that it tries to be as unconstrained as possible. The method really wants the data to speak. In Figure 2.1 and in general, there are arrows from all factors to all observed variables.

Number of factors The number of factors (symbolized here by p) is a fundamental property of a factor analysis model. For example, it determines the number of parameters. It's typically very important to subject matter experts, too. You can always get their attention by asking if something they are talking about is uni-dimensional. For example, is creativity uni-dimensional? Are political attitudes uni-dimensional (primarily just left-right)? In market research, how about attitudes toward a particular product category? Is it just positive-negative? Their eyes will light up.

Of course, there can be lots of factors. For example, Cattell's Sixteen Personality Factor Questionnaire [16] (documented in a 1970 paper by Cattell, Eber and Tatsuoka) is based on factor analyses of a large number of personality test items. They came up with 16 factors.

In a classical factor analysis, the number of common factors is generally not known in advance; it is determined in an exploratory manner. The first guiding principle is a piece of wisdom [39] from Kaiser (1960), who pointed out that for the typical problem involving human behavior or any other complex system, there are probably hundreds of common factors. Including them all in the model is out of the question. The objective should be to come up with a model that includes the most important factors for the variables in the study, and captures the essence of what is going on. Simplicity is important. Other things being more or less equal, the fewer factors the better. I have already mentioned a

[5] In some books, the term "factor loading" is reserved for the correlations between factors and observed variables. When the factors are uncorrelated, the λ_{ij} in (2.7) are indeed correlations, and the two common uses of the term coincide.

widely accepted rule of thumb[6] that says there should be at least three observed variables per factor [25]. This sets practical soft upper bound for the number of factors.

To narrow the search for the number of factors, quite a few methods are available. If the parameters are estimated by maximum likelihood, perhaps the most natural approach is to test goodness of fit using the likelihood ratio test (1.18) on page 171, increasing the number of factors until the model fits. This idea has quite a pedigree. It was essentially proposed by Lawley [41] in 1940[7], though he derived a slightly different large-sample chi-squared test. The reasoning is that if we really insist that the error terms are independent of the factors and have a diagonal covariance matrix, the only way that the model can be incorrect is that it does not have enough factors. Thus, any test for goodness of fit is also a test for number of factors.

Hypothesis testing may be attractive, but one thing to bear in mind is Kaiser's observation that in reality, there are probably hundreds of factors. Suppose the true number of factors is very large. Because the power of the likelihood ratio test increases with the sample size, significant lack of fit may be expected for any model with a modest number of factors, even if that model explains most of the non-error variance in an elegant and useful way. Statistically, rejecting the null hypothesis is a correct decision, because the model is wrong. Scientifically, it would be unfortunate. This suggests that while formal tests for lack of fit may be useful, one should not rely on them exclusively.

Another common method [39], and one that continues to be the default in some popular statistical software, is due to Kaiser (1960). Kaiser proposed estimating number of factors by the number of eigenvalues of the correlation matrix that are greater than one. The idea is that even though factor analysis and principal components analysis are different, still, if the correlations among the observed variables arise from p common factors, then the optimality of principal components in explaining variance suggests that p principal components will explain at least as much variance. And then, as in principal components, adding an additional factor that explains less variance than a single variable will not improve the model as a summary of the data.

A variation, called *parallel* analysis [31] is to test whether each eigenvalue is significantly larger than one would expect by chance. The meaning of "chance" is the probability distribution of an (ordered) eigenvalue under the null hypothesis that the variables are uncorrelated. These distributions are approximated by randomly independently permuting the observed data values a large number of times, and calculating the eigenvalues of the correlation matrix for each permutation. A factor is retained if the corresponding ordered eigenvalue is larger than the 95th percentile of the random values.

A graphical alternative called the *scree plot* [15] was proposed by Cattell (1966). Scree is a term from geology. It refers to the pile of rock and debris often found at the foot of a

[6] A rule of thumb is a rule that comes from experience and expert opinion, but is not backed up by hard evidence. The term apparently comes from brewing beer. In the early days before thermometers, the master brewer would stick a thumb in the vat of fermenting hops and stuff, and if the temperature felt right then it was on to the next stage.

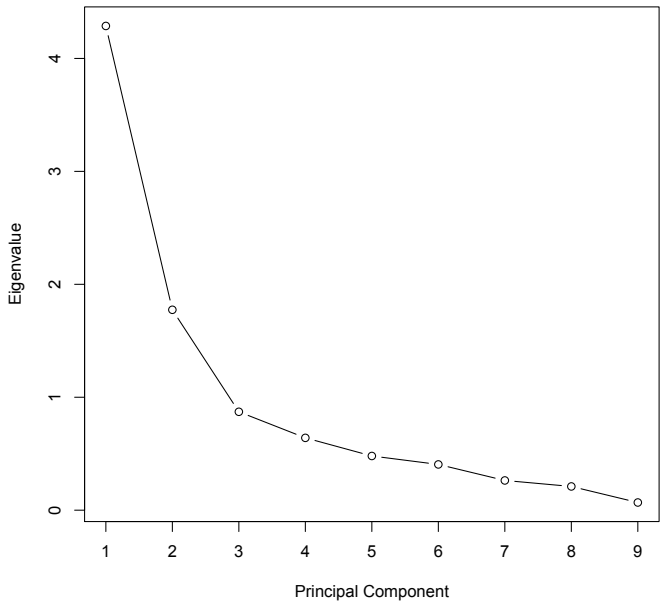
[7] This is the same article where Lawley proposed estimating factor loadings by maximum likelihood. Like many of the procedures that are now standard in multivariate analysis, maximum likelihood factor analysis became practical for most real data sets only after the invention of electronic computers.

mountain cliff or volcano. Scree slopes tend to be concave up, steepest near the cliff and then tailing off. In factor analysis, a scree plot shows the eigenvalues of the correlation matrix, sorted in order of magnitude. It has the numbers $1, \ldots, k$ (k = the number of principal components as well as variables) on the x axis, and the eigenvalues on the y axis. The largest eigenvalue goes with 1, the second largest with 2, and so on. It is very common for the graph to decrease rapidly at first, and then straighten out with a small negative slope for the rest of the way. The point at which the linear trend begins is the estimated number of factors.

Figure 2.2 show a scree plot for the Mind-body data, described in Example 2.1 on page 207. Reading the data and creating the object `pc` with `prcomp` has already been illustrated.

```
> Eigenvalue = pc$sdev^2
> plot(1:9,Eigenvalue,type='b',xlab='Principal Component',xaxp=c(1,9,8))
```

Figure 2.2: Scree Plot for the Mind-Body Data



The linear part of decreasing trend appears to begin with the third eigenvalue, suggesting three factors. There are only nine variables, so the rule of at least three variables per factor would limit us to three factors at most, anyway. Two of the eigenvalues are greater than one, suggesting two factors. There is no requirement that these any of these criteria coincide, and in fact it is reassuring that they are this close.

A final criterion for number of factors is interpretability. What do the factors seem to represent? Typically, the answer is more clear for models with fewer factors. With more and more factors, explanation tends to become increasingly difficult, and the wise factor

analyst will stop at a point where there is still a convincing story to tell. This process is subjective, but reasonable and widely accepted. In a professional paper, one might read something like "A maximum likelihood factor analysis extracted four interpretable factors, accounting for an estimated 72% of the variance in the attitude scales. Table 3 shows the factor loadings . . . "

Identifiability The parameters of the general factor analysis model are massively non-identified. This is true even when, as in the example of Figure 2.1, the model passes the test of the parameter count rule. To see this, first observe that the parameters are the unique contents of the matrices $\mathbf{\Phi}$, $\mathbf{\Lambda}$ and $\mathbf{\Omega}$. If two distinct triples $(\mathbf{\Phi}, \mathbf{\Lambda}, \mathbf{\Omega})$ yield the same covariance matrix $\mathbf{\Sigma} = cov(\mathbf{d}_i)$, then the parameters cannot be identified from $\mathbf{\Sigma}$. In practice, that means they can't be identified at all. Calculating,

$$\begin{aligned} cov(\mathbf{d}_i) = \mathbf{\Sigma} &= cov(\mathbf{\Lambda}\mathbf{F}_i + \mathbf{e}_i) \\ &= \mathbf{\Lambda}\mathbf{\Phi}\mathbf{\Lambda}^\top + \mathbf{\Omega}. \end{aligned}$$

The square root matrix of a symmetric matrix is also symmetric, so

$$\begin{aligned} \mathbf{\Lambda}\mathbf{\Phi}\mathbf{\Lambda}^\top + \mathbf{\Omega} &= \mathbf{\Lambda}\,\mathbf{\Phi}^{1/2}\mathbf{I}\mathbf{\Phi}^{1/2}\,\mathbf{\Lambda}^\top + \mathbf{\Omega} \\ &= (\mathbf{\Lambda}\mathbf{\Phi}^{1/2})\mathbf{I}(\mathbf{\Phi}^{1/2\top}\mathbf{\Lambda}^\top) + \mathbf{\Omega} \\ &= (\mathbf{\Lambda}\mathbf{\Phi}^{1/2})\mathbf{I}(\mathbf{\Lambda}\mathbf{\Phi}^{1/2})^\top + \mathbf{\Omega} \\ &= \mathbf{\Lambda}_2\mathbf{I}\mathbf{\Lambda}_2^\top + \mathbf{\Omega} \end{aligned}$$

Unless $\mathbf{\Phi} = cov(\mathbf{F}_i)$ was equal to the identity in the first place, the triple $(\mathbf{I}, \mathbf{\Lambda}_2, \mathbf{\Omega})$ is different from $(\mathbf{\Phi}, \mathbf{\Lambda}, \mathbf{\Omega})$, yet it yields the same $\mathbf{\Sigma}$. This shows that the parameters are not identifiable.

Actually, $\mathbf{\Sigma}$ is produced by infinitely many parameter sets. Let $\mathbf{Q}$ be an arbitrary positive definite covariance matrix for $\mathbf{F}_i$. Then

$$\begin{aligned} \mathbf{\Sigma} &= \mathbf{\Lambda}_2\mathbf{I}\mathbf{\Lambda}_2^\top + \mathbf{\Omega} \\ &= \mathbf{\Lambda}_2\mathbf{Q}^{-\frac{1}{2}}\mathbf{Q}\mathbf{Q}^{-\frac{1}{2}}\mathbf{\Lambda}_2^\top + \mathbf{\Omega} \\ &= (\mathbf{\Lambda}_2\mathbf{Q}^{-\frac{1}{2}})\mathbf{Q}(\mathbf{Q}^{-\frac{1}{2}\top}\mathbf{\Lambda}_2^\top) + \mathbf{\Omega} \\ &= (\mathbf{\Lambda}_2\mathbf{Q}^{-\frac{1}{2}})\mathbf{Q}(\mathbf{\Lambda}_2\mathbf{Q}^{-\frac{1}{2}})^\top + \mathbf{\Omega} \\ &= \mathbf{\Lambda}_3\mathbf{Q}\mathbf{\Lambda}_3^\top + \mathbf{\Omega} \end{aligned} \tag{2.8}$$

No matter what the truth might be, one can make the covariance matrix of the factors absolutely anything, and then adjust the factor loadings to yield exactly the same $\mathbf{\Sigma}$ that is produced by the true parameter values. Note that for multivariate normal data with expected value zero (the usual assumption), all one can ever get from increasing amounts of data is a closer and closer approximation of $\mathbf{\Sigma}$. This means that empirical data cannot help us learn the model parameters. It's not a good situation.

The classical way out of this dilemma is to regard the covariance matrix of the factors as essentially arbitrary, and fix $\mathbf{\Phi} = \mathbf{I}$. The factors are said to be "orthogonal" (at right angles, uncorrelated). They are also standardized, meaning that the (scalar) expected value of each factor is zero, and its variance equals one. This is justified on the grounds of simplicity and ease of interpretation.

Of course, the assumption of uncorrelated factors may be difficult to justify. Furthermore, it is untestable given model (2.6), since all possible covariance matrices for the factors are equally compatible with any set of data. In exploratory factor analysis, the possibility of correlated factors is addressed by transforming the estimates from a model with orthogonal factors into estimates for a model in which the factors are oblique – that is, not at right angles. Accordingly, we will proceed with the orthogonal factor model for the present.

Again, setting $\mathbf{\Phi} = \mathbf{I}$ standardizes the factors as well as making them uncorrelated. The observed variables are standardized as well. For $j = 1, \ldots, k$ and (almost) independently for $i = 1, \ldots, n$ the data we work with are $z_{ij} = \frac{d_{ij} - \bar{d}j}{s_j}$. Thus, each observed variable has variance one as well as mean zero.

In the revised exploratory factor analysis model below, the subscripts i on $\mathbf{z}_i$, $\mathbf{F}_i$ and $\mathbf{e}_i$ have been dropped to reduce notational clutter. Implicitly, everything applies independently for $i = 1, \ldots, n$. The model is

$$\mathbf{z} = \mathbf{\Lambda F} + \mathbf{e}, \text{ where} \tag{2.9}$$

- $\mathbf{z}$ is a $k \times 1$ observable random vector. Each element of $\mathbf{z}$ has expected value zero and variance one.
- $\mathbf{\Lambda}$ is a $k \times p$ matrix of constants.
- $\mathbf{F}$ (F for factor) is a $p \times 1$ latent random vector with expected value zero and covariance matrix $\mathbf{I}_p$.
- The $k \times 1$ vector of error terms $\mathbf{e}$ has expected value zero and covariance matrix $\mathbf{\Omega}$, which is diagonal.

For this model, everything emerges in terms in terms of correlations rather than covariances. This is a virtue, because correlations are easier to interpret. First, $cov(\mathbf{z}_i) = \mathbf{\Sigma} = \mathbf{\Lambda\Lambda}^\top + \mathbf{\Omega}$ is a correlation matrix; correspondingly, estimation and inference will be based on the sample correlation matrix.

Factor Loadings Next, consider the matrix of correlations between the factors and the observed variables. Because all the variables are standardized,

$$\begin{aligned} corr(\mathbf{z}, \mathbf{F}) &= cov(\mathbf{z}, \mathbf{F}) \\ &= cov(\mathbf{\Lambda F} + \mathbf{e}, \mathbf{F}) \\ &= \mathbf{\Lambda} cov(\mathbf{F}, \mathbf{F}) + cov(\mathbf{e}, \mathbf{F}) \\ &= \mathbf{\Lambda} cov(\mathbf{F}) + \mathbf{0} \\ &= \mathbf{\Lambda I} \\ &= \mathbf{\Lambda} \end{aligned} \tag{2.10}$$

Thus, the factor loadings are correlations between the observable variables and the factors. In particular, the correlation between observed variable i and factor j is λ_{ij}. The square of λ_{ij} is the reliability[8] of observed variable i as a measure of factor j.

Communality and Uniqueness Observed variable i (an element of $\mathbf{z}$; the index i goes from $1, \ldots, k$) may be written in scalar form as

$$\begin{aligned} z_i &= \lambda_{i1} F_1 + \cdots + \lambda_{ip} F_p + e_i \\ &= \sum_{j=1}^{p} \lambda_{ij} F_j + e_i, \end{aligned}$$

so that

$$\begin{aligned} Var(z_i) &= Var\left(\sum_{j=1}^{p} \lambda_{ij} F_j + e_i \right) \\ &= \sum_{j=1}^{p} \lambda_{ij}^2 Var(F_j) + Var(e_i) \\ &= \sum_{j=1}^{p} \lambda_{ij}^2 + \omega_i, \end{aligned} \tag{2.11}$$

where $\omega_i = Var(e_i)$ is the ith diagonal element of $\mathbf{\Omega}$. Since the observed variables are standardized, we have $1 = \sum_{j=1}^{p} \lambda_{ij}^2 + \omega_i$.

The variance of the observed variable has been split into two components. $\sum_{j=1}^{p} \lambda_{ij}^2$ is the proportion of variance in observed variable i that comes from the common factors. It is called the *communality*. To get the communality of a variable, add up the squares of the factor loadings in the corresponding row of $\mathbf{\Lambda}$. The other component is $\omega_i = 1 - \sum_{j=1}^{p} \lambda_{ij}^2$. It is is what's left over, the part that comes from error. It is called the *uniqueness* of the variable.

It may seem a bit peculiar for the variance of the error term to "know" about the factor loadings, but that's what you get when you standardize the observed variables.

[8] Psychometric reliability. See page 41.

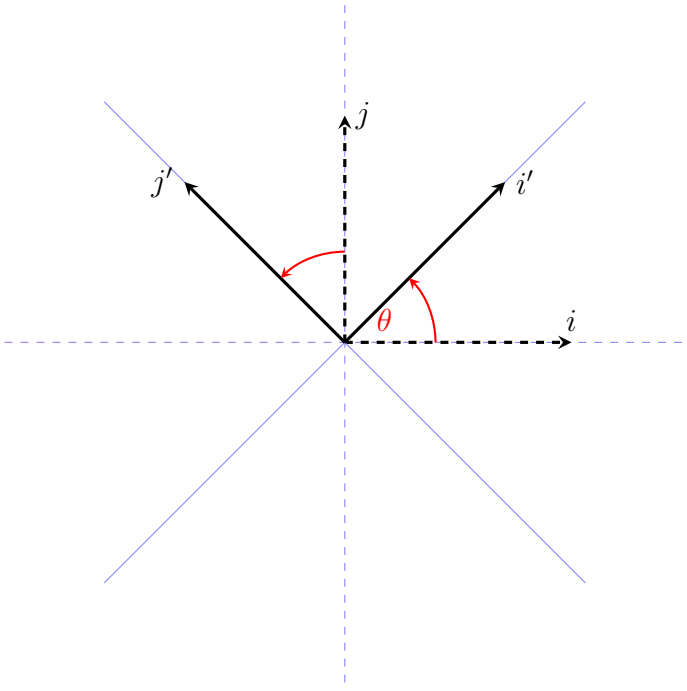
More important is that since the matrix $\mathbf{\Omega}$ is diagonal and its diagonal elements are functions of the λ_{ij}, the only parameters it contains are factor loadings that are already in $\mathbf{\Lambda}$. The role of $\mathbf{\Omega}$ is to make the diagonal elements of $\mathbf{\Sigma}$ equal one — that is, to make $\mathbf{\Sigma}$ a proper correlation matrix. In the standardized factor analysis model, the only unknown parameters are the factor loadings.

This really is quite nice. Since factor loadings are the correlations between the observable variables and the factors, they could be very informative about the processes driving the data. Squared factor loadings are reliabilities, another important feature of the measurement model. One could also use estimated factor loadings to estimate how much of the variance in each observable variable comes from each factor. All this could reveal what the underlying factors are, and what they mean.

2.3 Orthogonal Rotations

Unfortunately, the factor loadings are still not identifiable, so meaningful estimation is still out of the question. This part of the story depends on the idea of a rotation matrix. In Figure 2.3, a basis for $\mathbb{R}^2$ is provided by the unit vectors $\vec{\imath}$ and $\vec{\jmath}$, which are at right angles. These basis vectors are rotated through an angle θ, yielding $\vec{\imath}\,'$ and $\vec{\jmath}\,'$. If a point

Figure 2.3: Rotation



on the plane is denoted in terms of $\vec{\imath}$ and $\vec{\jmath}$ by (x, y), its position in terms of the rotated

basis vectors is

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta. \end{aligned}$$

These are the well-known *equations of rotation*. They may be written in matrix form as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \mathbf{R} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.12}$$

Using the identities $\cos(-\theta) = \cos\theta$ and $\sin(-\theta) = -\sin\theta$, one obtains a matrix that rotates the axes back to their original position.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} = \mathbf{R}^\top \begin{pmatrix} x' \\ y' \end{pmatrix}. \tag{2.13}$$

As the notation indicates, the matrix that reverses the rotation is the transpose of the original rotation matrix. Verifying that it's also the inverse,

$$\begin{aligned} \mathbf{R}\mathbf{R}^\top &= \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \\ &= \begin{pmatrix} \cos^2\theta + \sin^2\theta & -\cos\theta\sin\theta + \sin\theta\cos\theta \\ -\sin\theta\cos\theta + \cos\theta\sin\theta & \sin^2\theta + \cos^2\theta \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \mathbf{I}. \end{aligned}$$

So in two dimensions, the transpose of a rotation matrix is also its inverse. This fact holds in higher dimension as well. A $p \times p$ matrix $\mathbf{R}$ satisfying $\mathbf{R}^{-1} = \mathbf{R}^\top$ is called an *orthogonal matrix*, because the columns and rows are orthonormal vectors. Geometrically, pre-multiplication by an orthogonal matrix corresponds to a rotation or possibly a reflection in p-dimensional space. If you think of a set of factors $\mathbf{F}$ as a set of axes or underlying dimensions, then $\mathbf{RF}$ is a rotation (or reflection) of the factors. Call it an *orthogonal* rotation, because the factors remain uncorrelated — at right angles.

Rotation matrices are another source of non-identifiability. Returning to the standardized factor model, the covariance matrix of the observed data vector $\mathbf{z}$ is

$$\begin{aligned} \boldsymbol{\Sigma} &= \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top + \boldsymbol{\Omega} \\ &= \boldsymbol{\Lambda}\mathbf{R}^\top\mathbf{R}\boldsymbol{\Lambda}^\top + \boldsymbol{\Omega} \\ &= (\boldsymbol{\Lambda}\mathbf{R}^\top)(\boldsymbol{\Lambda}\mathbf{R}^\top)^\top + \boldsymbol{\Omega} \\ &= \boldsymbol{\Lambda}_2\boldsymbol{\Lambda}_2^\top + \boldsymbol{\Omega} \end{aligned}$$

That is, infinitely many rotation matrices produce the same $\boldsymbol{\Sigma}$, even though the factor loadings in $\boldsymbol{\Lambda}_2 = \boldsymbol{\Lambda}\mathbf{R}^\top$ can be very different for different $\mathbf{R}$ matrices.

Post-multiplication of $\mathbf{\Lambda}$ by $\mathbf{R}^\top$ is often called "rotation of the factors," for the following reason.

$$\begin{aligned} \mathbf{z} &= \mathbf{\Lambda F} + \mathbf{e} \\ &= (\mathbf{\Lambda R}^\top)(\mathbf{RF}) + \mathbf{e} \\ &= \mathbf{\Lambda}_2 \mathbf{F}' + \mathbf{e}. \end{aligned} \tag{2.14}$$

$\mathbf{F}' = \mathbf{RF}$ is a set of *rotated* factors. All rotations of the factors produce the same covariance matrix of the observable data.

In addition, all sets of rotated factors account for the same proportion of variance. To see this, recall that $\sum_{j=1}^{p} \lambda_{ij}^2$, the formula for the communality of observed variable i, instructs us to add up the squares of the factor loadings in row i of $\mathbf{\Lambda}$. This equals the ith diagonal of element of $\mathbf{\Lambda\Lambda}^\top$. Applying a rotation,

$$\begin{aligned} \mathbf{\Lambda}_2\mathbf{\Lambda}_2^\top &= (\mathbf{\Lambda R}^\top)(\mathbf{\Lambda R}^\top)^\top \\ &= \mathbf{\Lambda R}^\top \mathbf{R\Lambda}^\top \\ &= \mathbf{\Lambda\Lambda}^\top, \end{aligned} \tag{2.15}$$

so that rotation does not affect the proportions of variance explained by the common factors.

Confronted with this unpleasant situation, the exploratory factor analyst asks a question. Since all rotations of the factors explain the data equally well, why not just pick a good one? Here's an outline of the strategy.

1. Place some restrictions on the factor loadings, so that the only rotation matrix that preserves the restrictions is the identity matrix[9]. For example, $\lambda_{ij} = 0$ for $j > i$. There are other sets of restrictions that work — for example, forcing $\mathbf{\Lambda}^\top \mathbf{\Omega}^{-1} \mathbf{\Lambda}$ to be diagonal.

2. Generally, the restricted factor loadings may not make sense in terms of the data. Don't worry about it.

3. Estimate the loadings, perhaps by maximum likelihood. Other methods are available, but less commonly used than in the past.

4. Now apply a rotation, without any restriction on the resulting factor loadings. All (orthogonal) rotations result in the same maximum value of the likelihood function. That is, the maximum is not unique. Again, don't worry about it.

5. Pick a rotation that results in a simple pattern in the factor loadings, one that is easy to interpret.

The first and last steps require further discussion. The first step is to place restrictions on the factor loadings. Consider the restriction $\lambda_{ij} = 0$ for $j > i$. This means that observed

[9] This statement will require a bit of qualification, but it's the right idea.

variable one comes only from factor one, observed variable two comes only from factors one and two, observed variable three comes only from factors one, two and three – and so on. This pattern might be plausible for some sets of variables, but not in general. Carry on.

As an illustration, consider the case of two factors. In the path diagram of Figure 2.1, the straight arrow from F_2 to d_1 is missing. Also, the curved, double-headed arrow between F_1 and F_2 is missing, because the factors are orthogonal. In the model equations (2.7), the only restriction is $\lambda_{12} = 0$. Maintaining that restriction under rotation means

$$\begin{pmatrix} \lambda_{11} & 0 \\ \lambda_{21} & \lambda_{22} \\ \lambda_{31} & \lambda_{32} \\ \lambda_{41} & \lambda_{42} \\ \lambda_{51} & \lambda_{52} \\ \lambda_{61} & \lambda_{62} \\ \lambda_{71} & \lambda_{27} \\ \lambda_{81} & \lambda_{82} \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} \lambda'_{11} & 0 \\ \lambda'_{21} & \lambda'_{22} \\ \lambda'_{31} & \lambda'_{32} \\ \lambda'_{41} & \lambda'_{42} \\ \lambda'_{51} & \lambda'_{52} \\ \lambda'_{61} & \lambda'_{62} \\ \lambda'_{71} & \lambda'_{27} \\ \lambda'_{81} & \lambda'_{82} \end{pmatrix}$$

Focusing on the zero in the right-hand side, we have

$$\begin{aligned} & \lambda_{11}\sin\theta + 0\cos\theta = 0 \\ \Rightarrow \quad & \lambda_{11}\sin\theta = 0 \\ \Rightarrow \quad & \sin\theta = 0 \text{ (provided } \lambda_{11} \neq 0). \end{aligned}$$

Therefore, the angle of rotation θ equals 0, or π, or 2π, or 3π, or .... For $\theta = 0$ or any even multiple of π, $\cos\theta = 1$, and the rotation matrix is the identity. For $\theta = \pi$ or any odd multiple of π, $\cos\theta = -1$, and the rotation matrix is minus the identity. This reverses the signs of all the factor loadings.

There are two more orthogonal matrices that preserve the constraint $\lambda_{12} = 0$. They are $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. The first matrix reverses the signs of the first column of $\mathbf{\Lambda}$, but leaves the second column alone. The second matrix reverses the signs of the second column of $\mathbf{\Lambda}$ while leaving the first column alone. These represent reflections. The set of orthogonal matrices corresponds to the set of all possible reflections and rotations about the origin.

This shows that the restriction $\lambda_{12} = 0$ does not quite make the remaining factor loadings identifiable from the correlation matrix. We have located four distinct sets of parameter values that yield exactly the same correlation matrix for the observed data vector. On the other hand, these multiple solutions will not produce trouble in the numerical search for the MLE, because they are separated in the parameter space. The search will find just one of them, or it will wander off into nowhere, depending on the starting value and the topography of the likelihood function. It does not really matter which one we find. The plan is to apply a rotation later to find a more interpretable set of factor loadings, so the meaning of the parameter estimates is not an issue at this point.

To see what happens in higher dimension, it is enough to examine the case of $p = 3$. Denoting the orthogonal matrix by $\mathbf{R} = [r_{ij}]$ and insisting that it preserve the constraints $\lambda_{ij} = 0$ for $j > i$, we require

$$\begin{pmatrix} \lambda_{11} & 0 & 0 \\ \lambda_{21} & \lambda_{22} & 0 \\ \lambda_{31} & \lambda_{32} & \lambda_{33} \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} = \begin{pmatrix} \lambda'_{11} & 0 & 0 \\ \lambda'_{21} & \lambda'_{22} & 0 \\ \lambda'_{31} & \lambda'_{32} & \lambda'_{33} \\ \vdots & \vdots & \vdots \end{pmatrix} \tag{2.16}$$

Carrying out the row by column multiplications that yield the three zeros, conclude $r_{12} = r_{13} = r_{23} = 0$. Then use the fact that $\mathbf{R}\mathbf{R}^\top = \mathbf{0}$. Conclude that $r_{21} = r_{31} = r_{32} = 0$, and that

$$\begin{pmatrix} r_{11}^2 & 0 & 0 \\ 0 & r_{22}^2 & 0 \\ 0 & 0 & r_{33}^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

So, the off-diagonal elements of $\mathbf{R}$ are zero, and the diagonal elements are either plus or minus one, with entries of minus one representing reflections. This is how it goes in general, with 2^p different orthogonal matrices preserving the restriction $\lambda_{ij} = 0$ for $j > i$. The result is 2^p distinct minima of the minus log-likelihood function, all with the same value at the local minimum. Again, no numerical difficulties are created, because the multiple minima are separated in the parameter space, and the search for the MLE will only go down one of the holes.

The restriction $\lambda_{ij} = 0$ for $j > i$ is fairly easy to understand, but the restriction most used in practice is for $\mathbf{J} = \mathbf{\Lambda}^\top \mathbf{\Omega}^{-1} \mathbf{\Lambda}$ to be diagonal. In *Factor analysis as a statistical method* [42], Lawley and Maxwell (1971) show how this way of restricting $\mathbf{\Lambda}$ allows an efficient iterative solution of the equations obtained by differentiating the log likelihood and setting all the derivatives to zero.

Full details of Lawley's method will not be given here, but a few remarks are in order. First, since $\mathbf{\Lambda}$ is $k \times p$, the matrix $\mathbf{J}$ is $p \times p$. It is also symmetric, so insisting it be diagonal places $p(p-1)/2$ restrictions on $\mathbf{\Lambda}$. The restriction $\lambda_{ij} = 0$ for $j > i$ also induces a little triangle of zeros, as in (2.16); there are $p(p-1)/2$ of them, so the two methods impose the same number of restrictions. This is useful when it comes to counting degrees of freedom.

Second, let the $p \times p$ matrix $\mathbf{R}$ be a restricted kind of orthogonal matrix, a diagonal matrix, with values of plus or minus one on the diagonal. Any diagonal element of $\mathbf{R}$ equal to minus one reverses the signs of all the loadings in the corresponding column of $\mathbf{\Lambda}$. That's a reflection.

Replacing $\mathbf{\Lambda}$ with $\mathbf{\Lambda R}$,

$$\begin{aligned} (\mathbf{\Lambda R})^\top \mathbf{\Omega}^{-1} \mathbf{\Lambda R} &= \mathbf{R}^\top \mathbf{\Lambda}^\top \mathbf{\Omega}^{-1} \mathbf{\Lambda R} \\ &= \mathbf{R}^\top \mathbf{J R} \\ &= \mathbf{J}, \end{aligned}$$

since $\mathbf{J}$ is diagonal. Therefore, as in the simpler case of $\lambda_{ij} = 0$ for $j > i$, there are 2^p

different $\mathbf{\Lambda}$ matrices that satisfy the constraint, and also produce the same $\mathbf{\Sigma} = corr(\mathbf{z})$. Again, there are 2^p corresponding minima of the minus log likelihood function.

We need some notation. The initial (restricted) maximum likelihood estimates of the factors will be denoted by $\widetilde{\lambda}_{ij}$, while $\widehat{\lambda}_{ij}$ will be reserved for the final estimates after applying a rotation. In matrix form, $\widehat{\mathbf{\Lambda}} = \widetilde{\mathbf{\Lambda}}\mathbf{R}^\top$.

The Heywood case It is by no means guaranteed that the numerical search for the MLE will stop at a point that is in the parameter space. In fact, it is surprisingly common for the estimates to violate the inequality constraints of the model, as in the negative variance Example 1.5.1. Because the observed variables are standardized, an application of invariance to (2.11) yields $Var(z_i) = 1 = \sum_{j=1}^{p} \widetilde{\lambda}_{ij}^2 + \widetilde{\omega}_i$. A negative $\widetilde{\omega}_i$ would thus induce $\sum_{j=1}^{p} \widetilde{\lambda}_{ij}^2 > 1$, an estimated communality greater than one. Since the communality is the proportion of variance that comes from the common factors, this is a bit of a problem. It is sometimes called a *Heywood case*. Or sometimes, $\sum_{j=1}^{p} \widetilde{\lambda}_{ij}^2 = 1$ is called a Heywood case, and $\sum_{j=1}^{p} \widetilde{\lambda}_{ij}^2 > 1$ is called an *ultra-Heywood* case. You have to feel sorry for the user, and also for Mr. Heywood, since his name has been so often cursed[10]. Rotation will not solve this problem, because communality is unaffected by rotation (2.15).

Provided that an acceptable MLE has been located, the result is a set of estimated factor loadings that might be interpretable if the restrictions on $\mathbf{\Lambda}$ made sense in terms of the problem, but not otherwise. With respect to the original parameter space (without the restrictions), the set of estimated factor loadings we have found is only one of an uncountable infinity, all with the same value of the (minus log) likelihood function. There is one such set of factor loadings for every $p \times p$ orthogonal matrix. The last step in the 5-step recipe given earlier is to pick a good one, and go with that.

In the final step, the factors are rotated, so that $\widehat{\mathbf{\Lambda}} = \widetilde{\mathbf{\Lambda}}\mathbf{R}^\top$ has a "simple structure" that is easy to interpret. The concept of simple structure is not precisely defined, which in the past made factor analysis a bit subjective. There were many fruitless arguments in which researchers came to different conclusions because they used different rotations, even though they all claimed to have rotated to "simple structure."

It is helpful to lift the criteria for simple structure from Harman [28], 1976, p. 98; Harman takes them from Thurstone's highly influential (1947) book [64], which I cannot get my hands on right now[11]. Here are Thurstone's criteria for simple structure, using our notation.

1. Each row of $\widehat{\mathbf{\Lambda}}$ should have at least one zero.

[10] Heywood [29] gets the blame because of a 1931 paper in which he proves, among other things, that there can be legitimate correlation matrices that would imply a communality greater than one. It's one of the "cases" he considers, so I assume that's why they call it a Heywood case. From the perspective of this book, the factor analysis model implies inequality constraints that are not true of all positive definite correlation matrices. There is no mystery here.

[11] I am writing this in the Spring of 2021. The covid-19 pandemic is going strong, and the library is closed. One could not ask for a better excuse.

2. Each column of $\widehat{\mathbf{\Lambda}}$ should have at least p zeros, where p is the number of factors.

3. For every pair of columns of $\widehat{\mathbf{\Lambda}}$, there should be several variables with loadings that vanish in one column but not in the other.

4. For every pair of columns of $\widehat{\mathbf{\Lambda}}$, a large proportion of the variables should have loadings in both columns that are small in absolute value, when there are four or more factors.

5. For every pair of columns of $\widehat{\mathbf{\Lambda}}$, there should be only a small number of variables with non-vanishing loadings in both columns.

There are various ways of trying to approximate these goals in an objective manner. The methods are all iterative, taking a number of steps to approach some criterion. The most popular rotation method is *varimax* rotation. As described by Harman [28], the initial version of varimax was based on the following reasonable idea. To move the loadings in a particular column of $\widehat{\mathbf{\Lambda}}$ toward zero or ± 1, maximize the sample variance of the squared factor loadings. That is, maximize

$$\frac{1}{k}\sum_{i=1}^{k}\left(\widehat{\lambda}_{ij}^2\right)^2 - \frac{1}{k^2}\left(\sum_{i=1}^{k}\widehat{\lambda}_{ij}^2\right)^2$$

for column j. Adding up the columns yields the criterion

$$\frac{1}{k}\sum_{j=1}^{p}\sum_{i=1}^{k}\widehat{\lambda}_{ij}^4 - \frac{1}{k^2}\sum_{j=1}^{p}\left(\sum_{i=1}^{k}\widehat{\lambda}_{ij}^2\right)^2 .$$

In empirical tests, maximizing this criterion often yielded results that were less pleasing than a subjective rotation. In particular, the loadings near plus and minus one tended to be concentrated in just a few columns, which is inconsistent with properties three through five of simple structure given above. Not bothering with the intuitive justification (see Harman [28], p. 291), the work-around was to give somewhat less weight to factor loadings from variables with higher communality. This is accomplished by dividing by the communalities. The whole expression is also multiplied by k^2, which does not affect the point where the maximum occurs. The resulting criterion is

$$V = k\sum_{j=1}^{p}\sum_{i=1}^{k}\left(\frac{\widehat{\lambda}_{ij}}{\widehat{h}_i}\right)^4 - \sum_{j=1}^{p}\left(\sum_{i=1}^{k}\frac{\widehat{\lambda}_{ij}^2}{\widehat{h}_i^2}\right)^2 , \tag{2.17}$$

where $\widehat{h}_i^2 = \sum_{j=1}^{p}\widehat{\lambda}_{ij}^2$. That's the communality of variable i, the proportion of variance explained by the common factors. Another way to express (2.17) is to say the squared (estimated) factor loadings are adjusted so that each row adds to one. This is sometimes called "Kaiser normalization" after the guy who came up with the idea of varimax.

Expression (2.17) is not directly maximized over the factor loadings. Rather, the process starts with an initial set of estimated loadings (say, from constrained maximum

likelihood), and then rotates the factors two at a time as in Figure 2.3, picking the angle of rotation θ that maximizes V at each step. An iteration consists of going through $p-1$ steps, rotating factors 1 and 2, factors 2 and 3, and so on[12]. The process contines to iterate until V does not increase any more, to some specified number of decimal places. You might see a message like "Varimax converged in 5 iterations."

Varimax solutions are not unique. Suppose the rotation matrix $\mathbf{R}$ yields a solution $\widehat{\mathbf{\Lambda}} = \widetilde{\mathbf{\Lambda}}\mathbf{R}^\top$ that minimizes the varimax criterion (2.17). Let $\mathbf{M}$ be a $p \times p$ diagonal matrix, with each diagonal element equal to plus or minus one. $\mathbf{M}$ is an orthogonal matrix, and so is $\mathbf{R}^\top\mathbf{M}$. Therefore, $\widehat{\mathbf{\Lambda}}\mathbf{M} = \widetilde{\mathbf{\Lambda}}\mathbf{R}^\top\mathbf{M}$ is another orthogonal rotation/reflection. In $\widehat{\mathbf{\Lambda}}\mathbf{M}$, the columns of $\widehat{\mathbf{\Lambda}}$ are multiplied by the corresponding diagonal elements of $\mathbf{M}$. Potentially, this reverses the signs of the coefficients in one or more columns of $\widehat{\mathbf{\Lambda}}$. There is no effect on the value of the varimax criterion (2.17), because the varimax criterion is based on *squared* factor loadings. With p factors, the varimax criterion has 2^p minima, as each element of $\mathbf{M}$ switches between ± 1. The solution obtained from software will depend on where the numerical search happens to start.

Perhaps surprisingly, this does not make interpretation of results more difficult.. Reflecting a factor (multiplying by minus one) reverses the signs of the correlations between that factor and all the observable variables. It also directly reverses the *meaning* of the factor. So for example (recalling that the factors are standardized), if a factor represents wealth, then minus the factor represents poverty. After a varimax rotation, factors may be reflected at will if that makes it easier to think about the results.

In practice, varimax rotation tends to maximize the squared loading of each observable variable with just one underlying factor. In the typical varimax solution, each variable has a big loading on (correlation with) just one of the factors, and small loadings on the rest. It's usually not hard to look at the loadings and decide what the factors mean. Naming the factors is a fun game that is easy to play. In fact, the whole exercise is so satisfying that many casual users of exploratory factor analysis do not go beyond an orthogonal solution with a varimax rotation. Even the most casual class of users, who carry out a principal components analysis thinking it's factor analysis, often apply a varimax rotation to the correlations between variables and components, and are very happy with the result. Later, it will be seen that applying a rotation to principal components is really not such a bad idea, since the rotated components explain the same total amount of variance as the original set, and are easier to talk about.

Exploratory factor analysis of the Mind-body data We will start by re-reading the Mind-body data for the described in Example 2.1.

```
> # Factor analysis with orthogonal rotation
> rm(list=ls())
> bodymind = read.table('http://www.utstat.toronto.edu/~brunner/openSEM/data/bodymind.data.txt')
> dat = as.matrix(bodymind[,2:10]) # Omit sex. dat is now a numeric matrix.
> help(factanal)
```

[12]The result would seem to depend on the order in which the factors are sorted. I don't know of any proof that all orderings of factors yield the same varimax solution, but I expect that they are all pretty similar.

The built-in `factanal` function does maximum likelihood factor analysis with orthogonal factors. The first argument is an input data matrix, covariance matrix or correlation matrix. The second argument is the number of factors. How many factors should we have? We know from the principal components analysis that two eigenvalues of the correlation matrix are greater than two. That's one reason to try fitting a two-factor model. Another reason is that some of the variables are educational measurements (mental), while the rest are physical measures. Since the input comes from two distinct domains, I would expect two factors[13]. We'll start with two factors. Because there are only nine variables, the guideline of at least three variables per factor implies a maximum of three factors. The scree plot in Figure 2.2 suggests three factors, so we'll definitely consider a three-factor model after this.

```
> # Maximum likelihood, varimax, 2 factors
> fit2 = factanal(dat,factors=2) # rotation='varimax' is the default
> fit2

Call:
factanal(x = dat, factors = 2)

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.616   0.274   0.264   0.324   0.618   0.016   0.473   0.577   0.633

Loadings:
        Factor1 Factor2
progmat 0.181   0.592
reason  0.124   0.843
verbal  0.160   0.843
headlng 0.806   0.161
headbrd 0.618
headcir 0.963   0.238
bizyg   0.687   0.236
weight  0.638   0.129
height  0.588   0.144

               Factor1 Factor2
SS loadings      3.257   1.948
Proportion Var   0.362   0.216
Cumulative Var   0.362   0.578

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 87.55 on 19 degrees of freedom.
The p-value is 8.97e-11
```

First, look at the (estimated) factor loadings. We'll go over other details later. Notice that the loading for head breadth on Factor 2 appears to be missing. This happens because the matrix of factor loadings is a special kind of R object with its own elaborate print

[13]This kind of reasoning often works. To steal a joke from Tom Lehrer, factor analysis is like a sewer. What you get out of it depends on what you put into it.

method. By default, loadings below 0.1 in absolute value are not displayed. The objective is to make the loadings easier to understand by hiding trivial ones. As an SPSS jock in a past life, I am more used to loadings under 0.3 being blanked out, which works better in the present case. The cutoff is controlled by the `cutoff` option on `print`, as shown below.

```
> L2 = fit2$loadings
> print(L2,cutoff=0.3)

Loadings:
        Factor1 Factor2
progmat         0.592
reason          0.843
verbal          0.843
headlng 0.806
headbrd 0.618
headcir 0.963
bizyg   0.687
weight  0.638
height  0.588

               Factor1 Factor2
SS loadings      3.257   1.948
Proportion Var   0.362   0.216
Cumulative Var   0.362   0.578
```

Looking at this, it's a little difficult to believe that `L2` is just a matrix.

```
> is.matrix(L2)
[1] TRUE
> dim(L2)
[1] 9 2
```

So `L2` really just a 9×2 matrix. The little table under the loadings is produced automatically by the print method. It will be discussed presently.

With the small loadings hidden, it is easy to see that the mental measurements (`progmat`, `reason` and `verbal`) load primarily on the second factor, while the other variables (all physical) load on the first factor. One could name Factor One "Physical" and Factor Two "Mental." Or perhaps they could me named "Size" and "Smarts." This is a typical case. Often, the meaning of the factors jumps out at you, and they are easy to name. This is because of the varimax rotation. Unrotated factor loadings are often very difficult to interpret.

At the bottom of the output displayed for the `fit2` object, there is a chi-squared test for goodness of fit. The p-value is very small, indicating that the model does not fit well at all. For this reason and also for other reasons mentioned earlier, we need to look at a three-factor model. First, however, let's back up and look at some details, to clarify what the software is doing.

We will begin with an unrotated two-factor model, displaying all the factor loadings[14]. Note how the `cutoff=0` option on `print(fit2a)` is passed down to the printing of the factor loadings.

```
> fit2a = factanal(dat,factors=2,rotation='none')
> print(fit2a,cutoff=0)

Call:
factanal(x = dat, factors = 2, rotation = "none")

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.616   0.274   0.264   0.324   0.618   0.016   0.473   0.577   0.633

Loadings:
        Factor1 Factor2
progmat  0.335   0.521
reason   0.348   0.778
verbal   0.383   0.768
headlng  0.820  -0.064
headbrd  0.600  -0.149
headcir  0.992  -0.033
bizyg    0.725   0.040
weight   0.649  -0.049
height   0.605  -0.021

               Factor1 Factor2
SS loadings      3.708   1.497
Proportion Var   0.412   0.166
Cumulative Var   0.412   0.578

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 87.55 on 19 degrees of freedom.
The p-value is 8.97e-11
```

For an orthogonal factor model, squared factor loadings are components of explained variance. If you square the factor loadings and add, the row totals are commonalities, or proportions of variance explained by the common factors. The column totals are amounts of variance explained by each factor. The `addmargins` function is a convenient way to add row and column totals to a matrix.

```
> L2a = fit2a$loadings
> CompVar = addmargins(L2a^2) # Squared factor loadings are components of variance
> round(CompVar,3)
        Factor1 Factor2   Sum
progmat   0.112   0.271 0.384
reason    0.121   0.605 0.726
verbal    0.147   0.589 0.736
headlng   0.672   0.004 0.676
```

[14] They are estimated factor loadings, of course. Everything here is an estimate.

```
headbrd   0.360   0.022 0.382
headcir   0.983   0.001 0.984
bizyg     0.526   0.002 0.527
weight    0.421   0.002 0.423
height    0.366   0.000 0.367
Sum       3.708   1.497 5.205
```

Factor One explains a whopping 98.3% of the variance in head circumference, and 52.6% of the variance head length. Maybe the unrotated version it could be called "Head size" rather than just "Size." Anyway, the last column of numbers contains the commonalities. Checking that communality plus uniqueness equals one,

```
> fit2a$uniquenesses + CompVar[1:9,3] # Should equal ones
  progmat    reason    verbal   headlng   headbrd   headcir     bizyg    weight    height
0.9999884 0.9999994 1.0000003 0.9999984 0.9999912 1.0000000 1.0000001 1.0000041 1.0000124
```

Close enough. The column totals of `CompVar` are the amounts of variance explained by each factor, and indeed they match `SS loadings` in the display of `fit2a`. To convert these amounts of explained variance to proportions, divide by the number of variables (since the variables are standardized, the total amount of variance to explain is k, the number of variables). This yields the `Proportion Var` line. `Cumulative Var` is self-explanatory.

Notice that the `Proportion Var` lines are different for `fit2` (the rotated solution) and `fit2a` (unrotated). Rotation affects the amounts of variance explained by the factors. However, rotation does not affect the commonalities. So, it does not affect the uniquenesses or the total amount of variance explained.

To obtain the unrotated solution by maximum likelihood, `factanal` uses Lawley's [41] constraint that $\widetilde{\mathbf{\Lambda}}^\top \widetilde{\mathbf{\Omega}}^{-1} \widetilde{\mathbf{\Lambda}}$ must be diagonal[15]. Checking that the unrotated solution obeys this restriction,

```
> Omegahat = diag(fit2a$uniquenesses) # Diagonal matrix of uniquenesses little-omega-hat
> J = t(L2a) %*% solve(Omegahat) %*% L2a
> round(J,10)
         Factor1  Factor2
Factor1 69.10492 0.000000
Factor2  0.00000 5.002347
```

It's diagonal, as advertised. There is no reason to expect the rotated loadings to obey this constraint. Using the fact that $\widehat{\mathbf{\Omega}}$ is unaffected by rotation,

```
> J = t(L2) %*% solve(Omegahat) %*% L2; round(J,10)
         Factor1   Factor2
Factor1 64.36786 16.769564
Factor2 16.76956  9.739412
```

[15] Remember that $\widetilde{\mathbf{\Lambda}}$ and $\widetilde{\mathbf{\Omega}}$ are the initial estimates before rotation, obtained by constrained maximum likelihood. Of course, $\widetilde{\mathbf{\Omega}} = \widehat{\mathbf{\Omega}}$, because rotation does not affect the uniquenesses.

It is standard to specify the rotation when fitting the model, as in `fit2`. However, one may also fit a model without rotation as we have done here, and then rotate the factors as a separate step. R has a built-in `varimax` function (and also `promax`, which will not be discussed).

```
> varimax(L2a)
$loadings

Loadings:
        Factor1 Factor2
progmat 0.181   0.592
reason  0.124   0.843
verbal  0.160   0.843
headlng 0.806   0.161
headbrd 0.618
headcir 0.963   0.238
bizyg   0.687   0.236
weight  0.638   0.129
height  0.588   0.144

               Factor1 Factor2
SS loadings      3.257   1.948
Proportion Var   0.362   0.216
Cumulative Var   0.362   0.578

$rotmat
           [,1]      [,2]
[1,]  0.9623418 0.2718422
[2,] -0.2718422 0.9623418
```

The loadings are identical to the rotated factor matrix from `fit2` on page 229. The `varimax` function returns a list with two items, the factor loadings and the rotation matrix that maximizes the varimax criterion (2.17). The same matrix is also available as `fit2$rotmat`. Note that in our notation, `rotmat` is $\mathbf{R}^\top$, not $\mathbf{R}$.

More factors Next, we will try a model with three factors, as suggested by the scree plot and the highly significant chi-squared test for the the two-factor model. The `sort=TRUE` option re-orders the variables in the table of factor loadings, in an attempt to make the output easier to read.

```
> # Try a 3-factor model
> fit3 = factanal(dat,factors=3)
> print(fit3,cutoff=0.30, sort=TRUE)

Call:
factanal(x = dat, factors = 3)

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.606   0.215   0.309   0.005   0.268   0.094   0.256   0.560   0.565
```

```
Loadings:
        Factor1 Factor2 Factor3
headbrd  0.852
bizyg    0.787
weight   0.523           0.387
progmat          0.583
reason           0.879
verbal           0.811
headlng                  0.959
headcir  0.631           0.669
height   0.465           0.445

               Factor1 Factor2 Factor3
SS loadings      2.318   1.945   1.859
Proportion Var   0.258   0.216   0.207
Cumulative Var   0.258   0.474   0.680

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 30.89 on 12 degrees of freedom.
The p-value is 0.00205
```

This is more challenging. Factor 2 still definitely represents the mental measurements, while Factors 1 and 3 seem to reflect different aspects of head size. Factor 1 loads most highly on head breadth, followed closely by bizygomatic breadth, which is basically how far apart the eyes are. One could call Factor 1 "Face width." Factor 3 loads primarily on head length, and that's what it appears to be. Head circumference, which includes both face width and led length, loads about equally on the two factors. This makes pretty good sense. Height and weight, aspects of overall body size, also load on both of the head factors, though not as highly. We can live with this.

The chi-squared test for lack of fit is still significant, though the p-value of 0.00205 is a lot closer to 0.05 than 8.97e-11 is. Strictly speaking, the model still does not fit. Let's check the degrees of freedom. There are nine observed variables, so the correlation matrix $\mathbf{\Sigma}$ has 9(9-1)/2 = 36 unique elements. There would be 36 covariance structure equations in $9 \times 3 = 27$ unknown parameters, except that some of the unknown factor loadings are functions of the others, because of the constraint that $\mathbf{\Lambda}^\top \mathbf{\Omega}^{-1} \mathbf{\Lambda}$ is diagonal. There are $p(p-1)/2 = 3$ such functional connections among the factor loadings. Thus, the degrees of freedom for the test of fit should be $36 - 27 + 3 = 12$. That's what the printout says; okay.

Which model is better, the two-factor or the three-factor? The two-factor model explains an estimated 58% of the total variance, while the three-factor model explains an estimated 68%. Since there are nine observed variables, that 10% gain is worth about one variable. It's borderline. The two-factor model is a bit easier to talk about, but the three-factor model makes sense too. The three-factor model fits better, but it still does not fit in an absolute sense. How about a four-factor model? We would be violating the reasonable rule of at least three variables per factor, and we are almost running out of degrees of freedom, but it's worth a try.

```
> # A four-factor model?!
> print( factanal(dat,factors=4), cutoff=0.30, sort=TRUE)

Call:
factanal(x = dat, factors = 4)

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.580   0.216   0.305   0.005   0.005   0.109   0.248   0.356   0.437

Loadings:
        Factor1 Factor2 Factor3 Factor4
bizyg    0.633           0.527
weight   0.761
height   0.672
progmat          0.599
reason           0.872
verbal           0.813
headbrd                  0.957
headlng  0.423                   0.886
headcir  0.555           0.418   0.582

               Factor1 Factor2 Factor3 Factor4
SS loadings      2.037   1.946   1.433   1.321
Proportion Var   0.226   0.216   0.159   0.147
Cumulative Var   0.226   0.443   0.602   0.749

Test of the hypothesis that 4 factors are sufficient.
The chi square statistic is 8.98 on 6 degrees of freedom.
The p-value is 0.175
```

Now it seems that Factor 1 is overall body size, Factor 2 is educational test performance (or "intelligence," if you want to walk down that dark path), Factor 3 is face width, and Factor 4 is head length. Furthermore, the model technically fits. As for choice among the models, it's really a judgement call. As I see it, the clearest part of the picture is that the mental measurements form one cluster, and the physical measurements form another cluster, but one that may be more differentiated. I'm really torn between the two-factor model (appealing because of its simplicty), and the four-factor model, which may reveal the most detail. But is that detail real, or is it the result of over-fitting? If I had to choose, I suppose I would choose the two-factor model. It does not fully fit the data, but it tells a simple story that makes sense.

If you disagree, it does not mean that you are wrong. In the end, the choice of a model is quite subjective, though the way these analyses are written up, the semi-arbitrary final choice will probably seem like the only possibility. This is especially true because only one set of factor loadings will be presented. If you were looking for the TRUTH here, I'm sorry to disappoint you. This is in the nature of the beast called exploratory factor analysis.

In spite of all the uncertainty, this enterprise has been blessed with apparent success. There are many hundreds of published factor analytic studies in the social sciences, especially in psychology. For example, in their book *The measurement of meaning* [50],

Osgood Suci and Tannenbaum (1957) describe a series of investigation into how people describe objects, using 7-point scales ranging from Ugly to Beautiful, Strong to Weak, Fast to Slow and so on. Exploratory factor analysis revealed the same three factors across many different domains. One of the factors had high factor loadings for Good-Bad, Beautiful-Ugly, and similar adjective pairs. The investigators named the factor *evaluative*. Similar considerations led them to identify the other two main factors as *potency* and *activity*. Osgood et al. proposed that these are the main dimensions of connotative (as opposed to denotative) meaning in the English language.

In another famous application [24], Hans Eysenck[16] (1947) factor analyzed questions from a large number of personality scales, arriving at two factors, *neuroticism* and *extraversion*. It's a bit interesting that in order to get a high score on neuroticism, you have to be willing to say bad things about yourself, while if you say mostly good things you will get a low neuroticism score. Perhaps it's just Osgood et al.'s evaluative factor, reversed. In any case, there are hordes of other examples, including Cattell's Sixteen Personality Factor Questionnaire [16] mentioned earlier. The earlier work, including the examples cited here, tended to use estimation methods that are less computationally demanding than maximum likelihood. Varimax rotation also caught on gradually, as computing equipment became more available. Rotation to a "simple structure" used to be graphical and more than a little subjective.

2.4 Oblique Rotations

Correlated Factors Naturally, not everybody is comfortable with uncorrelated factors. The question of whether factors are correlated seems like something that should be decided based on the data, and not simply assumed. The problem is that by the calculation (2.8), any correlation matrix of the factors is equally compatible with any data set. This means that estimating $\mathbf{\Phi} = cov(\mathbf{F})$ is futile. However, there is almost no limit to human ingenuity.

An early subjective method (as usual, see Harman [28]) for the history) is well adapted to a setting in which there are several clusters of variables, highly correlated within sets, and much less so between sets. Compare the formula for the sample correlation coefficient to the formula for the cosine of the angle between two vectors.

$$\cos\theta = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}|\,|\vec{y}|} \qquad\qquad r = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\overline{y})^2}} \tag{2.18}$$

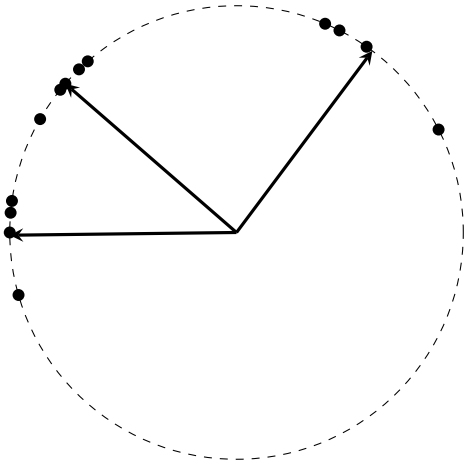
Now consider the vector of n values for a variable as a point in $\mathbb{R}^n$. Suppose that the data are centered by subtracting off sample means, as they are in the standardized case we are considering. Then the correlation between two variables equals the cosine of the angle between the two data vectors. This means that considered as points in $\mathbb{R}^n$, a set of highly correlated variables are physically clustered together. To estimate the factor that

[16] Eminent research psychologist, racist scum, running dog of the tobacco companies, fabricator of data and student of Sir Cyril Burt, who was also racist scum and a fabricator of data. See the Wikipedia article.

gives rise to them, run a vector through the center of the cluster. The natural choice is to have the estimated factor pass through the centroid — that is, through the multivariate sample mean of the data vectors belonging to that particular cluster. Then the estimated factor is normalized, giving it variance one.

Figure 2.4 shows a hypothetical example in two dimensions. Since the variables are standardized, they all have length one. This means that in $\mathbb{R}^n$, the data points lie on the surface of a hyper-sphere of radius one, centered at the origin. Since Figure 2.4 is in two dimensions, all the points are on the unit circle.

Figure 2.4: Correlated factors estimated by centroids



The estimated correlations between factors are the cosines of the angles between the arrows, and the correlations of variables with factors are the cosines of the angles between data points and the arrows. It all makes sense, and looking at this example, it is hard to see why the parameters cannot be estimated successfully by this method. The trick is that by calculating the arrows based only on the points in a single cluster, we are implicitly assuming that the points in that cluster arise from only one common factor (plus random error). Under this assumption, lots of the λ_{ij} values are zero, and in fact the remaining factor loadings and the correlations between factors are uniquely identifiable — provided there are at least three variables in each cluster. Chapter 3 treats confirmatory factor analysis models in which the parameters are identifiable, including the one just indicated.

The informal centroid method just described does work under some circumstances, but the big problem is cluster membership. When the variables form distinct, highly correlated clusters then everything is fine. More often, it will not be really clear how many clusters there are, and some variables will be difficult to classify. This uncertainty makes the method subjective, and led the developers of factor analysis to look for something more objective.

Oblique Rotations An *oblique* rotation is one in which the axes[17] need not remain at right angles. Starting with an initial orthogonal solution, the axes are rotated separately so as to achieve a simple structure in the factor loadings. There are various criteria for what "simple" means, leading to various flavours of the method.

The following account leads to the classical results, by a route that statisticians should be able to follow. The original explanations are much more complicated. Everything here is based on a model with equations $\mathbf{z} = \mathbf{\Lambda F} + \mathbf{e}$. The factors are standardized, and they are potentially correlated. Because the variance of each factor equals one, $cov(\mathbf{F}) = \mathbf{\Phi}$ is a correlation matrix. All other model specifications are the same as in Model (2.9) on page 219.

In an orthogonal factor model, the factor loadings in $\mathbf{\Lambda}$ are also the correlations between the observed variables and the factors. This is no longer true when the factors are correlated. With correlated factors, the calculations in (2.10) lead to

$$corr(\mathbf{z}, \mathbf{F}) = cov(\mathbf{z}, \mathbf{F}) = \mathbf{\Lambda\Phi}.$$

It is common to call the matrix of coefficients $\mathbf{\Lambda}$ the *factor pattern matrix*, while the matrix of correlations between variables and factors in $\mathbf{\Lambda\Phi}$ is called the *factor structure matrix*. In the factor analysis literature, these terms are applied to both the true parameter matrices and to their estimates.

When factors are correlated, some of the pleasing simplicity of the orthogonal model disappears. In particular, the explained variance of an observed variable no longer neatly splits itself into the variance explained by each factor. In scalar terms,

$$\begin{aligned} Var(z_i) &= var\left(\lambda_{i1}F_1 + \cdots + \lambda_{ip}F_p + e_i\right) \\ &= \sum_{j=1}^{p} \lambda_{ij}^2 Var(F_j) + \sum_{\ell \neq j} \lambda_{ij}\lambda_{i\ell} cov(F_\ell, F_j) + Var(e_i) \\ &= \sum_{j=1}^{p} \lambda_{ij}^2 + \sum_{\ell \neq j} \lambda_{ij}\lambda_{i\ell}\phi_{\ell j} + \omega_i. \end{aligned}$$

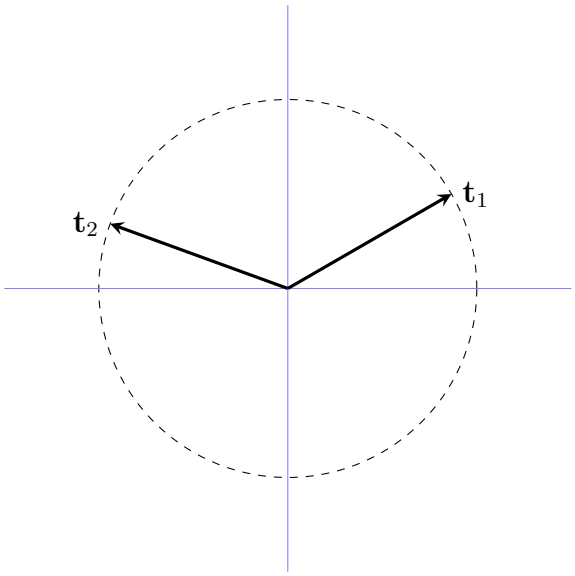
So, while the variance of z_i is still decomposed into an explained part and an unexplained part, the explained variance includes terms that come from each pair of factors, with the contribution governed by the correlation between factors as well as the factor loadings. Notice that while the factor loadings and correlations between factors may be mutually adjusted as in the re-parameterizations (2.8), the amount of unexplained variance ω_i is not affected. The choice of an oblique rotation is one such re-parameterization, and we will presently see that oblique rotations do not affect estimates of the uniqueness (explained variance) for any variable.

Oblique rotations are carried out using a $p \times p$ transformation matrix $\mathbf{T} = [t_{ij}]$ satisfying $\mathbf{T}^\top\mathbf{T} = \mathbf{\Phi}$. Denote column j of $\mathbf{T}$ by $\mathbf{t}_j$, so that $\mathbf{T} = (\mathbf{t}_1|\mathbf{t}_2|\cdots|\mathbf{t}_p)$. Because $\mathbf{\Phi}$ is a correlation matrix, $\mathbf{t}_j^\top\mathbf{t}_j = 1$. Thinking of $\mathbf{t}_1, \ldots, \mathbf{t}_p$ as vectors in $\mathbb{R}^p$ and using the formula in (2.18), the cosine of the angle between $\mathbf{t}_i$ and $\mathbf{t}_j$ is $\mathbf{t}_i^\top\mathbf{t}_j = Corr(F_i, F_j)$.

The matrix $\mathbf{T}$ is not unique. For $p = 2$, we have the picture in Figure 2.5. Spin the

[17] Think of the factors as dimensions, or axes of a co-ordinate system.

Figure 2.5: Columns of the $\mathbf{T}$ matrix



vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ around the unit circle[18] while keeping the angle between them constant. The cosine of the angle remains constant too, so there are infinitely many transformation matrices $\mathbf{T}$ that yield the same $\mathbf{\Phi}$. The square root matrix $\mathbf{\Phi}^{1/2}$ is just one of them. By the way, based on the similarity of Figure 2.5 to Figure 2.4, it would be easy to mistake the arrows in Figure 2.5 for factors. They are not. They are columns of the $\mathbf{T}$ matrix.

For a general number of factors p, the same spinning idea applies. Let $\mathbf{R}$ be a $p \times p$ orthogonal matrix. Then $(\mathbf{RT})^\top \mathbf{RT} = \mathbf{T}^\top \mathbf{R}^\top \mathbf{RT} = \mathbf{T}^\top \mathbf{T} = \mathbf{\Phi}$, and $\mathbf{RT}$ is another transformation matrix that produces $\mathbf{\Phi}$.

The next theorem says that, as Figure 2.5 suggests, *all* the transformation matrices for a given $\mathbf{\Phi}$ arise from spinning or reflecting a set of column vectors.

Theorem 2.1 *Let* $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *be square matrices satisfying* $\mathbf{T}_1^\top \mathbf{T}_1 = \mathbf{\Phi} = \mathbf{T}_2^\top \mathbf{T}_2$*, where* $\mathbf{\Phi}$ *is symmetric and positive definite. Then* $\mathbf{T}_2 = \mathbf{R}\,\mathbf{T}_1$*, where* $\mathbf{R}$ *is an orthogonal matrix.*

Proof. Because $\mathbf{\Phi}$ is positive definite, $\mathbf{T}_1$ and $\mathbf{T}_2$ are both full rank, and have inverses.

$$\begin{aligned} & \mathbf{T}_2^\top \mathbf{T}_2 = \mathbf{\Phi} \mathbf{T}_1^{-1} \mathbf{T}_1 \\ \Longrightarrow \quad & \mathbf{T}_2 = \left((\mathbf{T}_2^\top)^{-1} \mathbf{\Phi} \mathbf{T}_1^{-1} \right) \mathbf{T}_1 = \mathbf{R}\,\mathbf{T}_1 \end{aligned}$$

[18] If the axes were being rotated, the rotation matrix $\mathbf{R}$ in (2.12) would be employed. Here, the axes are remaining in position, while the points are being rotated through an angle θ. From the perspective of one of the points, it looks like the axes are being rotated through an angle of $-\theta$. So, to rotate the points, one would use the matrix $\mathbf{R}^\top$ in (2.13). Actually, in this case it does not matter which direction you spin the points.

Showing that $\mathbf{R}$ is an orthogonal matrix,

$$\begin{aligned}
\mathbf{R}^\top\mathbf{R} &= \left((\mathbf{T}_2^\top)^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1}\right)^\top\left((\mathbf{T}_2^\top)^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1}\right) \\
&= \mathbf{T}_1^{-1\top}\mathbf{\Phi}^\top\mathbf{T}_2^{\top\,-1\top}(\mathbf{T}_2^\top)^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1} \\
&= \mathbf{T}_1^{\top\,-1}\mathbf{\Phi}\mathbf{T}_2^{-1}(\mathbf{T}_2^\top)^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1} \\
&= \mathbf{T}_1^{\top\,-1}\mathbf{\Phi}\left(\mathbf{T}_2^\top\mathbf{T}_2\right)^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1} \\
&= \mathbf{T}_1^{\top\,-1}\mathbf{\Phi}\mathbf{\Phi}^{-1}\mathbf{\Phi}\mathbf{T}_1^{-1} \\
&= \mathbf{T}_1^{\top\,-1}\mathbf{\Phi}\mathbf{T}_1^{-1} \\
&= \mathbf{T}_1^{\top\,-1}\left(\mathbf{T}_1^\top\mathbf{T}_1\right)\mathbf{T}_1^{-1} \\
&= \mathbf{I}\cdot\mathbf{I} = \mathbf{I} \quad \blacksquare
\end{aligned}$$

You might be thinking that representing a set of unknown parameters in a way that is not unique will just make estimation more difficult. In fact, estimation of $\mathbf{\Phi}$ cannot be successful by conventional standards anyway, because $\mathbf{\Phi}$ is not identifiable. As you will see, the matrix $\mathbf{T}$ will be chosen to yield a nice simple factor structure. The fact that $\mathbf{T}$ is not unique just provides a wider range of options.

In the meantime, consider a standard orthonormal basis for $\mathbb{R}^p$, with basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_p$, where $\mathbf{b}_i$ has a one in position i, and zeros elsewhere. Noting that

$$\mathbf{t}_j = \begin{pmatrix} t_{1j} \\ t_{2j} \\ \vdots \\ t_{pj} \end{pmatrix},$$

the cosine of the angle between $\mathbf{b}_i$ and $\mathbf{t}_j$ is $\mathbf{b}_i^\top\mathbf{t}_j = t_{ij}$. Now suppose we were to adopt $\mathbf{t}_1, \ldots, \mathbf{t}_p$ as an alternative basis for $\mathbb{R}^p$. Column j of the transformation matrix $\mathbf{T}$ contains the cosines of the angles between $\mathbf{t}_j$ and the original basis vectors.

Geometrically, changing to the basis $\mathbf{t}_1, \ldots, \mathbf{t}_p$ corresponds to rotating each of the original basis vectors through a set of angles satisfying the cosines in $\mathbf{T}$. It is an *oblique* rotation rather than an orthogonal rotation, because the new basis vectors need not be at right angles. The operation can be represented as a matrix multiplication:

$$\mathbf{T}^\top\mathbf{b}_j = \mathbf{t}_j.$$

This rotation can be applied to $\mathbf{a} = [a_j]$, a general point in $\mathbb{R}^p$. We have

$$\mathbf{a} = a_1\mathbf{b}_1 + \cdots + a_p\mathbf{b}_p,$$

so that

$$\begin{aligned}
\mathbf{T}^\top\mathbf{a} &= \mathbf{T}^\top(a_1\mathbf{b}_1 + \cdots + a_p\mathbf{b}_p) \\
&= a_1\mathbf{T}^\top\mathbf{b}_1 + \cdots + a_p\mathbf{T}^\top\mathbf{b}_p \\
&= a_1\mathbf{t}_1 + \cdots + a_p\mathbf{t}_p,
\end{aligned}$$

representing the point $\mathbf{a}$ in terms of the new co-ordinate system. The main point here is that it makes sense to describe pre-multiplication by $\mathbf{T}^\top$ as a rotation, one that is not necessarily orthogonal.

Here is how oblique rotation may be used[19] to estimate the unknown parameters $\mathbf{\Lambda}$ and $\mathbf{\Phi}$. Returning to the model equations, we start by applying a change of variables to the factors.

$$\begin{aligned}\mathbf{z} &= \mathbf{\Lambda F} + \mathbf{e} \\ &= \mathbf{\Lambda T}^\top(\mathbf{T}^\top)^{-1}\mathbf{F} + \mathbf{e} \\ &= \mathbf{AF}' + \mathbf{e},\end{aligned}$$

where $\mathbf{A} = \mathbf{\Lambda T}^\top$ and $\mathbf{F}' = (\mathbf{T}^\top)^{-1}\mathbf{F}$. We have

$$\begin{aligned}cov(\mathbf{F}') &= cov\left((\mathbf{T}^\top)^{-1}\mathbf{F}\right) \\ &= (\mathbf{T}^\top)^{-1} cov(\mathbf{F}) \left((\mathbf{T}^\top)^{-1}\right)^\top \\ &= (\mathbf{T}^\top)^{-1}\mathbf{\Phi}\mathbf{T}^{-1} \\ &= (\mathbf{T}^\top)^{-1}\mathbf{T}^\top\mathbf{T}\mathbf{T}^{-1} \\ &= \mathbf{I},\end{aligned}$$

so the change of variables and the accompanying re-parameterization results in an orthogonal factor model. The new parameter matrix $\mathbf{A} = [a_{ij}]$ is not identifiable, but it can be estimated up to an orthogonal rotation, perhaps by constrained maximum likelihood. This yields $\widehat{\mathbf{A}}$. (In Section 2.3, the symbol $\widetilde{\mathbf{A}}$ was employed for the constrained MLE. Here, we return to a more standard notation.)

Now perform another change of variables, to return to a version of the original model with correlated factors.

$$\begin{aligned}\mathbf{z} &= \mathbf{AF}' + \mathbf{e} \\ &= \mathbf{A}\,(\mathbf{T}^\top)^{-1}\mathbf{T}^\top\,\mathbf{F}' + \mathbf{e} \\ &= \mathbf{A}(\mathbf{T}^\top)^{-1}\,\mathbf{T}^\top(\mathbf{T}^\top)^{-1}\mathbf{F} + \mathbf{e} \\ &= \mathbf{A}(\mathbf{T}^\top)^{-1}\,\mathbf{F} + \mathbf{e}\end{aligned}$$

Instead of expanding $\mathbf{A}$ and simplifying back to the original model, we will use our earlier estimate of $\mathbf{A}$, which is an estimate of $\mathbf{\Lambda T}^\top$. Symbolically, $\widehat{\mathbf{A}} = \widehat{\mathbf{\Lambda T}^\top}$. The matrix of original factor loadings $\mathbf{\Lambda}$ (the factor pattern matrix) is estimated by

$$\widehat{\mathbf{\Lambda}} = \widehat{\mathbf{\Lambda T}^\top}(\mathbf{T}^\top)^{-1} = \widehat{\mathbf{A}}(\mathbf{T}^\top)^{-1}. \tag{2.19}$$

The factor structure matrix $corr(\mathbf{z}, \mathbf{F}) = \mathbf{\Lambda\Phi}$ is estimated by

$$\begin{aligned}\widehat{\mathbf{\Lambda\Phi}} &= \widehat{\mathbf{A}}(\mathbf{T}^\top)^{-1}\mathbf{\Phi} \\ &= \widehat{\mathbf{A}}(\mathbf{T}^\top)^{-1}\mathbf{T}^\top\mathbf{T} \\ &= \widehat{\mathbf{A}}\mathbf{T}\end{aligned} \tag{2.20}$$

[19] I say "may be" used, because this is not the typical way of describing the process. However, it is clear to me and it leads to the usual estimates.

The problem is that the estimates (2.19) and (2.20) both depend on the transformation matrix $\mathbf{T}$, which unknown and un-knowable[20]. The solution, as in the case of orthogonal rotation, is to choose a $\mathbf{T}$ matrix that results in a nice simple structure, – either in the factor pattern (2.19) or the factor structure (2.20). As a by-product, the choice of $\mathbf{T}$ yields an estimate of $\mathbf{\Phi}$. Using $\widehat{\mathbf{T}}$ to denote the chosen $\mathbf{T}$ matrix, $\widehat{\mathbf{\Phi}} = \widehat{\mathbf{T}}^\top \widehat{\mathbf{T}}$.

The way in which $\mathbf{T}$ is chosen does not affect the estimated uniqueness, the portion of the variance in an observed variable that comes from the factors. From $cov(\mathbf{z}) = \mathbf{\Lambda\Phi\Lambda}^\top + \mathbf{\Omega}$, the estimated explained variances of the observed variables are the diagonal elements of

$$\begin{aligned}
\widehat{\mathbf{\Lambda}}\widehat{\mathbf{\Phi}}\widehat{\mathbf{\Lambda}}^\top &= \widehat{\mathbf{A}}\left(\widehat{\mathbf{T}}^\top\right)^{-1}\widehat{\mathbf{T}}^\top\widehat{\mathbf{T}}\left(\widehat{\mathbf{A}}\left(\widehat{\mathbf{T}}^\top\right)^{-1}\right)^\top \\
&= \widehat{\mathbf{A}}\widehat{\mathbf{T}}\left(\left(\widehat{\mathbf{T}}^\top\right)^{-1}\right)^\top\widehat{\mathbf{A}}^\top \\
&= \widehat{\mathbf{A}}\widehat{\mathbf{T}}\widehat{\mathbf{T}}^{-1}\widehat{\mathbf{A}}^\top \\
&= \widehat{\mathbf{A}}\widehat{\mathbf{A}}^\top,
\end{aligned}$$

which does not depend on the oblique rotation $\widehat{\mathbf{T}}$.

The choice of $\mathbf{T}$ depends on what criterion is optimized in search of "simple structure." A variety of criteria have been proposed, each with its own impressive name and cadre of enthusiastic supporters — most of whom, sad to say, are no longer with us. Harman [28] describes the oblimax, oblimin (including quartimin, covarimin, and biquartimin), direct oblimin, binormamin and orthoblique methods, and I may have missed some. Confronted with this wealth of alternatives, I have decided to present the oblimin family, mostly because of its connection to varimax.

Oblimin rotation Initially, oblimin rotation sought to simplify the factor structure matrix, while later work focused on simplifying the factor pattern. Logically but not chronologically, the story begins with the *covarimin* method. Consider any two columns of the estimated factor structure matrix in Expression 2.20, but square all the elements in the matrix. Suppose that all the squared correlations in the matrix are either close to one or close to zero, and that large squared correlations in one column are beside near-zero squared correlations in the other column. If this could be achieved for every pair of columns, it would be a nice simple structure in which each observed variable has a large correlation (positive or negative) with just one factor, and near zero correlations with the others. In other words, we want negative relationships between the squared correlations in all the columns.

Accordingly, square all the estimated correlations in Expression 2.20, and think of the resulting $k \times p$ matrix as a kind of data file, with k observations on p "variables."

[20]The matrix $\mathbf{T}$ is constrained by the fact the its columns are vectors of length one, and also by $\mathbf{T}^\top\mathbf{T} = \mathbf{\Phi}$. This does not help us get at $\mathbf{T}$, because the correlation matrix $\mathbf{\Phi}$ is not just unknown, it is not even identifiable. In addition, it has previously been shown that uncountably many $\mathbf{T}$ matrices produce a given $\mathbf{\Phi}$. Therefore, even if $\mathbf{\Phi}$ were known exactly, recovery of the "true" $\mathbf{T}$ would be impossible.

Calculate the $p \times p$ sample covariance matrix for these "data." The covarmin criterion is the sum of unique off-diagonal elements (multiplied by k^2):

$$\sum_{i=1}^{p} \sum_{j=i+1}^{p} \left(k \sum_{\ell=1}^{k} c_{\ell i}^2 c_{\ell j}^2 - \sum_{\ell=1}^{k} c_{\ell i}^2 \sum_{\ell=1}^{k} c_{\ell j}^2 \right), \tag{2.21}$$

where c is an estimated correlation. Minimize (2.21) over the elements of the matrix $\mathbf{T}$. This can be done one column (axis) at a time, literally rotating the axes. As an option, it is possible to adjust for communalities as in (2.17). Again, one divides squared correlations by the communality, that is, by the total amount of variance in the variable that is explained by the common factors.

Covarimin is similar in approach to varimax, and in fact they are both described in the same 1958 paper by H. F. Kaiser [38]. Both methods treat a matrix of squared estimated correlations as data. Varimax maximizes the sum of sample variances of the columns, and covarimin minimizes the sum of sample covariances of the columns.

Covarimin was a nice idea, but based on application to real data sets, it did not yield satisfactory results. The problem was that it tended to produce solutions that were "too orthogonal." That is, the estimated correlation matrix of the factors $\widehat{\mathbf{\Phi}} = \mathbf{T}^\top \mathbf{T}$ tended to be quite close to the identity, regardless of the data. Perhaps as a way of reducing how negative the covariances were, a modification was to drop the negative part of (2.21). This yielded a criterion called *quartimin*, which had been proposed some years earlier:

$$\sum_{i=1}^{p} \sum_{j=i+1}^{p} \left(\sum_{\ell=1}^{k} c_{\ell i}^2 c_{\ell j}^2 \right). \tag{2.22}$$

The quartimin criterion tended to yield solutions that were "too oblique." As a compromise, putting back the k that was omitted from (2.22) and then averaging the two criteria yielded the *biquartimin* criterion:

$$\sum_{i=1}^{p} \sum_{j=i+1}^{p} \left(k \sum_{\ell=1}^{k} c_{\ell i}^2 c_{\ell j}^2 - \frac{1}{2} \sum_{\ell=1}^{k} c_{\ell i}^2 \sum_{\ell=1}^{k} c_{\ell j}^2 \right), \tag{2.23}$$

effectively retaining half of the second term in the covarimin criterion (2.21). Some viewed the biquartimin compromise as "just right," but it is a matter of taste how much of the second term to retain. To accommodate all preferences, the general oblimin criterion replaces the fraction $\frac{1}{2}$ with a number between zero and one inclusive, symbolized by γ.

$$\sum_{i=1}^{p} \sum_{j=i+1}^{p} \left(k \sum_{\ell=1}^{k} c_{\ell i}^2 c_{\ell j}^2 - \gamma \sum_{\ell=1}^{k} c_{\ell i}^2 \sum_{\ell=1}^{k} c_{\ell j}^2 \right), \tag{2.24}$$

where $0 \leq \gamma \leq 1$. Setting $\gamma = 0$ yields quartimin, while $\gamma = \frac{1}{2}$ yields biquartimin, and $\gamma = 1$ yields covarimin.

Direct oblimin The oblimin method just described seeks to simplify the factor structure (the matrix of estimated correlations between variables and rotated factors). In contrast, *direct* oblimin seeks to simplify the factor pattern, the matrix of estimated factor loadings[21]. Both versions of oblimin find a transformation matrix $\mathbf{T}$ that minimizes a criterion of the form (2.24), subject to the restriction that the column vectors of $\mathbf{T}$ have length one. In the original oblimin, the $c_{\ell j}$ are elements of $\widehat{\mathbf{A}}\mathbf{T}$ (see Expression 2.20), while for direct oblimin, the $c_{\ell j}$ are elements of $\widehat{\mathbf{A}}(\mathbf{T}^\top)^{-1}$ (Expression 2.19). It can make a difference, because there is no reason to expect the $\mathbf{T}$ that optimizes $\widehat{\mathbf{A}}(\mathbf{T}^\top)^{-1}$ will also optimize $\widehat{\mathbf{A}}\mathbf{T}$, unless $\mathbf{T}$ is close to the identity.

The name "direct" oblimin seems to be something of a historical accident. The original oblimin algorithm really was very complicated and indirect. In the paper that introduced direct oblimin [33], Jennrich and Sampson (1966) provided a much more straightforward algorithm for minimizing the factor pattern version of (2.24). With more than a half century of hindsight, it seems that there was a failure to distinguish between directness in the criterion to be minimized and directness in the algorithm used to get the job done. At any rate, everyone seems to have bought it, and the original "indirect" version of oblimin has faded away.

The direct oblimin of Jennrich and Sampson (1966) came to full fruition almost 40 years later [7] in Bernaards and Jennrich (2005). Yes, it's the same Jennrich. Bernaards and Jennrich do the optimization directly over the columns of the $\mathbf{T}$ matrix, alternating between a gradient descent step and a projection onto the set of column vectors with length one. The mathematical expressions are remarkably simple and elegant when written in matrix form.

Bernaards and Jennrich have provided the R package `GPArotation`, which implements their method for a variety of orthogonal and oblique rotations. The options naturally include direct oblimin, but they do not include indirect oblimin, as far as I can tell. R's built-in `factanal` function has a `rotation=` option, and it can use all the methods in `GPArotation`, provided that the `GPArotation` package is loaded. Otherwise, `factanal` only knows about `varimax` and `promax`. The widely used `psych` package does factor analysis with oblique rotation using functions from `GPArotation`, so oblimin rotation in `psych` is direct oblimin. This has a lot of prominence because in `psych`'s workhorse `fa` function (`fa` for factor analysis), the default is to apply an oblimin rotation unless the user specifies otherwise. The `EFAtools` package [61] uses `GPArotation` and `psych`. I have been unable to find any R packages that do the original "indirect" oblimin.

In terms of commercial software, online documentation suggests that in SAS and SPSS, oblimin means direct oblimin. The once-great BMDP package had both direct and indirect oblimin options, but it is no longer available. In practical terms, direct oblimin rotation is your only choice unless you write your own function.

There is no obvious reason why the Bernaards and Jennrich algorithm could not be applied to the factor structure matrix instead of the factor pattern matrix. The result would be a very direct version of indirect oblimin. Should you bother to write the

[21] Again, the factor loadings are constants that are like regression coefficients, linking the rotated factors to the observed variables.

code? In my judgement, the answer is no, because simplicity in the factor loadings is probably more desirable than simplicity in the correlations between variables and factors anyway. Thinking of factor analysis as a causal model (that's the structural equation model perspective), the factors are literally producing the observed variables through the factor loadings in the factor pattern matrix. On the other hand, the factor structure matrix $\widehat{\mathbf{A}}\mathbf{T}$ is estimating $corr(\mathbf{z}, \mathbf{F})$. From the formula $corr(\mathbf{z}, \mathbf{F}) = \mathbf{\Lambda\Phi}$, the correlation between an observed variable and a factor depends on the correlations between factors as well as the direct connection of the factor to the variable.

Consider the two-factor example of Figure 2.1 and Equations (2.7), except with the observed variables $d_{i,j}$ standardized. We have, for example,

$$\begin{aligned} corr(z_{i,4}, F_{i,1}) &= cov(z_{i,4}, F_{i,1}) \\ &= cov(\lambda_{41}F_{i,1} + \lambda_{42}F_{i,2} + e_{i,4},\ F_{i,1}) \\ &= \lambda_{41}\, cov(F_{i,1}, F_{i,1}) + \lambda_{42}\, cov(F_{i,1}, F_{i,2}) + cov(e_{i,4}, F_{i,1}) \\ &= \lambda_{41}\, Var(F_{i,1}) + \lambda_{42}\, corr(F_{i,1}, F_{i,2}) + 0 \\ &= \lambda_{41} + \lambda_{42}\phi_{1,2}. \end{aligned}$$

If there were p factors, the formula would be $corr(z_{i,4}, F_{i,1}) = \lambda_{41} + \sum_{j=2}^{p} \lambda_{4j}\phi_{1,j}$.

We see that the correlation between an observed variable and a factor includes the direct link between the variable and the factor, but mixed together with the links between the variable and all the other factors, in a way that depends on the correlations between factors. This means that the interpretation of such a correlation may not be straightforward at all. For example, a high correlation could come from a strong direct link between the variable and the factor, but it could also come from a weak or zero direct link, accompanied by strong indirect effects of the other factors. Conversely, evidence of a strong direct link could be suppressed by the operation of the other factors, resulting in a near zero correlation. It's very much like the correlation-causation picture in general. Though they did not suggest this argument, Jennrich and Sampson showed good taste when they decided to focus on the factor pattern matrix.

It is important to mention that the effect of the γ parameter is vastly different for the two oblimin methods. Recall that $0 \leq \gamma \leq 1$ for indirect oblimin, with $\gamma = 0$ producing the most oblique solutions (largest estimated correlations between factors), and $\gamma = 1$ producing the most orthogonal solutions. For direct oblimin, the connection is reversed, with obliqueness *increasing* as a function of γ, rather than decreasing. For direct oblimin, very large negative γ values yield near zero correlations between factors, while the estimated correlations between factors rapidly approach ± 1 for fairly small positive values of γ. Then, still for very modest positive γ values, the matrix $\mathbf{T}$ becomes numerically singular, and the algorithm fails to converge. The usual recommendation is that γ should be zero or negative for direct oblimin.

To avoid confusion, Harman [28] uses the symbol δ instead of γ for direct oblimin, reserving the symbol γ for indirect oblimin. While SPSS follows Harman's notation, R does not. In the `oblimin` function of the `GPArotation` package, the `gam=` argument controls the value of γ for direct oblimin. Similarly, `rotate=quartimin` means direct quartimin; that is, direct oblimin with $\gamma = 0$.

If you happen to know about indirect oblimin, the vocabulary in the `GPArotation` documentation can be a trap for the unwary. In `help(oblimin)`, the `gam=` argument is documented by "0=Quartimin, .5=Biquartimin, 1=Covarimin." These are all the direct oblimin versions. It's a bit strange because, while $\gamma = 0$ is reasonable and in fact is the default, $\gamma = 1/2$ does not correspond to anything interesting for direct oblimin, and the value $\gamma = 1$ frequently leads to convergence problems.

Here is an illustration of factor analysis with oblique rotation for the Mind-body data. To make the example complete, we begin by reading the data. Then, loading the `GPArotation` package makes `rotate=oblimin` available in `factanal`.

```
> rm(list=ls())
> bodymind = read.table('http://www.utstat.toronto.edu/~brunner/openSEM/data/bodymind.data.txt')
> dat = as.matrix(bodymind[,2:10]) # Omit sex. dat is now a numeric matrix.
> # install.packages("GPArotation", dependencies=TRUE) # Only need to do this once
> library(GPArotation)
> ob2 = factanal(dat, factors=2, rotation='oblimin'); print(ob2, cutoff=0)

Call:
factanal(x = dat, factors = 2, rotation = "oblimin")

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.616   0.274   0.264   0.324   0.618   0.016   0.473   0.577   0.633

Loadings:
        Factor1 Factor2
progmat  0.081   0.584
reason  -0.027   0.862
verbal   0.012   0.853
headlng  0.829  -0.019
headbrd  0.655  -0.125
headcir  0.982   0.025
bizyg    0.688   0.088
weight   0.656  -0.014
height   0.600   0.014

               Factor1 Factor2
SS loadings      3.353   1.836
Proportion Var   0.373   0.204
Cumulative Var   0.373   0.577

Factor Correlations:
        Factor1 Factor2
Factor1   1.000   0.384
Factor2   0.384   1.000

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 87.55 on 19 degrees of freedom.
The p-value is 8.97e-11
```

Note the matrix $\widehat{\mathbf{\Phi}}$ under `Factor Correlations`, with an estimated correlation between factors of 0.348. For comparison, here is a repeat of the analysis with a varimax rotation.

```
> print( factanal(dat, factors=2, rotation='varimax'), cutoff=0) # For comparison

Call:
factanal(x = dat, factors = 2, rotation = "varimax")

Uniquenesses:
progmat  reason  verbal headlng headbrd headcir   bizyg  weight  height
  0.616   0.274   0.264   0.324   0.618   0.016   0.473   0.577   0.633

Loadings:
        Factor1 Factor2
progmat 0.181   0.592
reason  0.124   0.843
verbal  0.160   0.843
headlng 0.806   0.161
headbrd 0.618   0.019
headcir 0.963   0.238
bizyg   0.687   0.236
weight  0.638   0.129
height  0.588   0.144

               Factor1 Factor2
SS loadings      3.257   1.948
Proportion Var   0.362   0.216
Cumulative Var   0.362   0.578

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 87.55 on 19 degrees of freedom.
The p-value is 8.97e-11
```

The estimated uniquenesses are the same, as they should be. The factor loadings are quite similar; they are perhaps a bit sharper for the oblique rotation, so the oblique rotation allowed a closer approach to simple structure. The little sub-table under the factor loadings, starting with `SS loadings`, is also similar for the varimax and oblimin rotations. However, this is deceiving. That subtable, generated as part of the print method for an object of class loadings, is appropriate only when factors are orthogonal. In that case, squared factor loadings are separate components of variance, and `SS loadings` makes sense. With an oblique rotation there is no such interpretation, and the next example will make that table look as nonsensical as it really is.

First, just note that the test for number of factors (the chi-squared test for goodness of fit) is identical for the varimax and oblimin rotations. That is because displays for all rotations, orthogonal or oblique, simply report the test for the initial solution, which is orthogonal.

Now let us return to the `SS loadings` table under the factor loadings for oblimin rotation. With higher values of γ in (2.24), estimated correlations between factors become larger, and the factor pattern matrix becomes more dissimilar to the factor structure matrix. Now, R's built-in `factanal` function will not accept a γ argument (at least not in a natural way), but the `fa` function in the `psych` package will.

```
> # Try fa with gam: Check SS loadings
> # install.packages("psych", dependencies=TRUE) # Only need to do this once
> library(psych); library(psychTools)
> # fa(dat,nfactors=2, fm='ml', rotate = 'oblimin', gam=0) # Same results as ob2
> psych0 = fa(dat,nfactors=2, fm='ml', rotate = 'oblimin', gam=1)
> psych0$loadings

Loadings:
        ML1    ML2
progmat -0.503 -1.064
reason  -1.030 -1.733
verbal  -0.943 -1.672
headlng  1.640  0.952
headbrd  1.420  0.969
headcir  1.888  1.033
bizyg    1.243  0.585
weight   1.295  0.750
height   1.155  0.634

                  ML1    ML2
SS loadings    15.031 11.149
Proportion Var  1.670  1.239
Cumulative Var  1.670  2.909
```

I rest my case. The matrix of factor loadings is definitely a factor pattern and not a factor structure matrix, because its elements are not correlations. The `SS loadings` table still squares them, adds them up and divides by nine (the number of variables) in order to get proportions of explained variance. This is nonsense, because the resulting proportions are greater than one.

One does not need to use the `psych` package to be able to specify the γ parameter. It's better to use the `oblimin` function in the `GPArotation` package[22]. To do this, first fit an initial, orthogonal model. Then use the `oblimin` function on the unrotated factor loadings $\widehat{\mathbf{A}}$.

```
> fit2a = factanal(dat,factors=2,rotation='none')
> Ahat = fit2a$loadings
> O2b = oblimin(Ahat); O2b # Matches ob2 (gamma=0)
Oblique rotation method Oblimin Quartimin converged.
Loadings:
        Factor1 Factor2
progmat  0.0810  0.5837
reason  -0.0271  0.8619
verbal   0.0121  0.8533
headlng  0.8293 -0.0194
headbrd  0.6554 -0.1249
headcir  0.9822  0.0251
bizyg    0.6880  0.0876
weight   0.6558 -0.0140
```

[22] Of course the people who wrote the `psych` package might not agree. `psych` can do a lot of things, and if you need or want to do them, you should use the `psych` package.

```
height   0.6001  0.0141

Rotating matrix:
       [,1]   [,2]
[1,]  0.975 0.0608
[2,] -0.471 1.0813

Phi:
      [,1]  [,2]
[1,] 1.000 0.384
[2,] 0.384 1.000
```

The factor loadings match `ob2`, with a default value of $\gamma = 0$. Note that the rotation is described as `Oblimin Quartimin`, which is accurate as long as it's understood to be direct oblimin. The so-called `Rotating matrix` is $\left(\widehat{\mathbf{T}}^{\top}\right)^{-1}$. Looking at the list of items produced by the `oblimin` function,

```
> ls(O2b)
[1] "convergence" "Gq"          "loadings"    "method"      "orthogonal"  "Phi"         "Table"
[8] "Th"
```

The `Th` item is $\widehat{\mathbf{T}}$, so the following matches the "`Rotating matrix`."

```
> solve(t(O2b$Th))
           [,1]       [,2]
[1,]  0.9750642 0.06083524
[2,] -0.4713192 1.08129141
```

The `loadings` item is the rotated factor pattern matrix. Fortunately, it is not an object of class "loadings," so it does not use the misleading print method.

```
> O2b$loadings
            Factor1     Factor2
progmat  0.08096306  0.58366083
reason  -0.02710434  0.86193259
verbal   0.01213684  0.85326892
headlng  0.82927554 -0.01942290
headbrd  0.65541656 -0.12490579
headcir  0.98223567  0.02510846
bizyg    0.68800789  0.08760370
weight   0.65576088 -0.01399468
height   0.60011168  0.01406469
```

It is instructive to look at the results for $\gamma = 1/2$, described as (direct) `Oblimin Biquartimin`.

```
> O2c = oblimin(Ahat, gam = 0.5); O2c
Oblique rotation method Oblimin Biquartimin converged.
Loadings:
        Factor1 Factor2
progmat -0.0399  0.6440
reason  -0.2201  0.9756
verbal  -0.1751  0.9593
```

```
headlng  0.9153 -0.1604
headbrd  0.7476 -0.2502
headcir  1.0734 -0.1358
bizyg    0.7364 -0.0162
weight   0.7234 -0.1253
height   0.6561 -0.0844

Rotating matrix:
       [,1]    [,2]
[1,]  1.058 -0.0944
[2,] -0.756  1.2969

Phi:
      [,1]  [,2]
[1,] 1.000 0.639
[2,] 0.639 1.000
```

The estimated correlation between factors is larger, and so are the estimated factor loadings. It still tells the same general story, with the first factor representing physical size, the the second factor reflecting performance on the mental tests.

To give an idea of how the correlation between factors varies as a function of γ, I fit a series of models with different γ values, covering a wide range.

```
> # Correlation between factors as a function of gamma
> options(scipen=999) # To suppress scientific notation
> gammaval = c(-500, -100, -50, -10, 0, 0.25, 0.50, 0.75, 1)
> ngamma = length(gammaval); phi12 = numeric(ngamma)
> for(j in 1:ngamma) phi12[j] = oblimin(Ahat, gam = gammaval[j])$Phi[1,2]
> round(rbind(gammaval,phi12),3)
            [,1]     [,2]    [,3]    [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
gammaval -500.000 -100.000 -50.000 -10.000 0.000 0.250 0.500 0.750 1.000
phi12       0.002    0.011   0.022   0.105 0.384 0.481 0.639 0.802 0.935
```

Observe how the correlation between factors approaches zero very slowly as $\gamma \to -\infty$, and approaches one rapidly for positive values of γ ; it could also approach -1 for increasing gamma, depending on the data and the starting values for the oblimin minimization.

One might well ask, what's the right γ value? The answer is that there is no right answer. γ is not an unknown parameter of the statistical model, and it is not something that can be estimated. It's a setting that determines the criterion to be minimized in order to seek a simple structure in the estimated factor loadings. Typically, users try different γ values, and settle on one that produces results that seem reasonable for the data. Or, they just use the software default of $\gamma = 0$.

As a final example, consider a three-factor model for the Mind-body data. Before doing this, I will disclose that I expect one mental factor and two physical factors, and that the physical factors will be more correlated with one another than either of them is with the mental factor.

```
> fit3a = factanal(dat,factors=3,rotation='none')
> A3hat = fit3a$loadings
```

```
> oblimin(A3hat)
Oblique rotation method Oblimin Quartimin converged.
Loadings:
         Factor1 Factor2 Factor3
progmat  0.18819 -0.0948  0.5686
reason  -0.07027 -0.0170  0.9104
verbal  -0.00706  0.0312  0.8253
headlng  1.02610 -0.0549 -0.0135
headbrd -0.10556  0.9136 -0.0746
headcir  0.59540  0.4652  0.1061
bizyg    0.11682  0.7509  0.1408
weight   0.31027  0.4429  0.0422
height   0.38873  0.3609  0.0477

Rotating matrix:
        [,1]    [,2]    [,3]
[1,]  1.0070 -0.0171 0.00256
[2,] -0.5830  0.8516 0.60471
[3,]  0.0544 -0.7553 0.87791

Phi:
      [,1]  [,2]  [,3]
[1,] 1.000 0.465 0.327
[2,] 0.465 1.000 0.254
[3,] 0.327 0.254 1.000
```

The third factor is definitely mental, and could be called "academic ability" without raising much controversy. The first factor is dominated by head length and to a lesser extent by head circumference; it could be called "head size." The second factor has its highest loadings on head breadth and bizygomatic breadth. It could be called "face width." The picture is quite similar to what appeared with an orthogonal (varimax) rotation. The correlation between the two physical factors is higher than the others in the $\widehat{\mathbf{\Phi}}$ matrix, but not notably so.

2.5 Factor Scores

My man Harman [28] suggests that there are two potential reasons for doing factor analysis. One is to understand how certain unobservable factors give rise to a set of observable data. The other reason is data reduction. You have a lot of variables, and you'd like to work with a smaller set that contains essentially the same information. So you do a factor analysis, and then somehow "estimate" the values of the factors for all the members of your sample. The estimates are called *factor scores*. They may be more interpretable than the original data, in the sense that they might represent the underlying quantities that the data were intended to measure. Certainly, there will be fewer of them. If only for this reason, they may be easier to think about and to incorporate into subsequent data analyses.

Principal components Frequently,

2.6 A Dose of Reality

Let us take a step back from all these interesting details, and consider what we have. In Sections 2.2 and 2.3, it was shown that the parameters of the exploratory factor analysis model are not identifiable, even if they are constrained by making the factors uncorrelated. Infinitely many sets of parameter values are consistent with any data set, so that using the data alone to distinguish between them is hopeless. The solution in exploratory factor analysis is rotation. After locating a family of parameter sets that are all equally reasonable given the data (and arguably better than other values outside the family), one rotates the factors in such a way that the factor loadings achieve a simple structure, one that is scientifically meaningful.

The problem is that in statistics, there *is* such a thing as a true parameter value[23]. If the truth resembles simple structure, rotation will take you closer to the truth. If the truth does not resemble simple structure, rotation will take you farther away. The factor analysts have a deep philosophical answer to this, but before dealing with that I will give a few examples using simulated data. The advantage of simulated data is that we know exactly what the true parameter values are.

In the first example, the truth corresponds to simple structure. There are two uncorrelated factors and eight observed variables. The first four variables load only on factor one, and the last four load only on factor two. This is an extreme case of simple structure, and looks very much like varimax. All the distributions are normal, so that the model underlying maximum likelihood estimation is exactly correct. All the variables are centered, and the true variances of both factors and observed variables are exactly equal to one. In the code, the factor loadings (the only parameters) are denoted by $\mathtt{L}_{ij}$. The sample size is huge, so that sampling error does not make the pattern of results harder to see.

```
> rm(list=ls())
> n = 50000 # Huge sample size
> # True factor loadings have a simple structure like varimax (All communalities = 0.49)
> # Factor loadings
> L11 = 0.7; L12 = 0.0
> L21 = 0.7; L22 = 0.0
> L31 = 0.7; L32 = 0.0
> L41 = 0.7; L42 = 0.0
> L51 = 0.0; L52 = 0.7
> L61 = 0.0; L62 = 0.7
> L71 = 0.0; L72 = 0.7
> L81 = 0.0; L82 = 0.7
> # Error Variances
> v1 = 1 - L11**2 - L12**2
> v2 = 1 - L21**2 - L22**2
> v3 = 1 - L31**2 - L32**2
> v4 = 1 - L41**2 - L42**2
> v5 = 1 - L51**2 - L52**2
> v6 = 1 - L61**2 - L62**2
```

[23] Except maybe in the mind of the most radical subjective Bayesian.

```
> v7 = 1 - L71**2 - L72**2
> v8 = 1 - L81**2 - L82**2
> # Generate data
> set.seed(9999)
> F1 = rnorm(n,0,1); F2 = rnorm(n,0,1)
> d1 = L11*F1 + L12*F2 + rnorm(n,0,sqrt(v1))
> d2 = L21*F1 + L22*F2 + rnorm(n,0,sqrt(v2))
> d3 = L31*F1 + L32*F2 + rnorm(n,0,sqrt(v3))
> d4 = L41*F1 + L42*F2 + rnorm(n,0,sqrt(v4))
> d5 = L51*F1 + L52*F2 + rnorm(n,0,sqrt(v5))
> d6 = L61*F1 + L62*F2 + rnorm(n,0,sqrt(v6))
> d7 = L71*F1 + L72*F2 + rnorm(n,0,sqrt(v7))
> d8 = L81*F1 + L82*F2 + rnorm(n,0,sqrt(v8))
> dmat = cbind(d1,d2,d3,d4,d5,d6,d7,d8)
```

We fit a two-factor model by maximum likelihood, with a varimax rotation.

```
> factanal(dmat,factors=2,rotation='varimax')

Call:
factanal(x = dmat, factors = 2, rotation = "varimax")

Uniquenesses:
   d1    d2    d3    d4    d5    d6    d7    d8
0.506 0.510 0.519 0.511 0.507 0.505 0.508 0.510

Loadings:
   Factor1 Factor2
d1          0.698
d2          0.694
d3          0.688
d4          0.695
d5  0.697
d6  0.699
d7  0.696
d8  0.695

                Factor1 Factor2
SS loadings       1.971   1.953
Proportion Var    0.246   0.244
Cumulative Var    0.246   0.491

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 10.22 on 13 degrees of freedom.
The p-value is 0.676
```

It is arbitrary which factor is called Factor 1 and which is called Factor 2. Other than that, all the estimates are right on the money. The model is correct, and it fits. Everything is perfect.

In the second example, the true pattern of factor loadings is not at all like varimax. Everything else is very similar to the first example.

```
> # Truth is not like varimax (All communalities = 0.50)
> # Factor loadings
> L11 = 0.5; L12 = -0.5
> L21 = 0.5; L22 = -0.5
> L31 = 0.5; L32 = -0.5
> L41 = 0.5; L42 = -0.5
> L51 = 0.5; L52 =  0.5
> L61 = 0.5; L62 =  0.5
> L71 = 0.5; L72 =  0.5
> L81 = 0.5; L82 =  0.5
> # Error Variances
> v1 = 1 - L11**2 - L12**2
> v2 = 1 - L21**2 - L22**2
> v3 = 1 - L31**2 - L32**2
> v4 = 1 - L41**2 - L42**2
> v5 = 1 - L51**2 - L52**2
> v6 = 1 - L61**2 - L62**2
> v7 = 1 - L71**2 - L72**2
> v8 = 1 - L81**2 - L82**2
> # Generate data
> set.seed(8888)
> F1 = rnorm(n,0,1); F2 = rnorm(n,0,1)
> d1 = L11*F1 + L12*F2 + rnorm(n,0,sqrt(v1))
> d2 = L21*F1 + L22*F2 + rnorm(n,0,sqrt(v2))
> d3 = L31*F1 + L32*F2 + rnorm(n,0,sqrt(v3))
> d4 = L41*F1 + L42*F2 + rnorm(n,0,sqrt(v4))
> d5 = L51*F1 + L52*F2 + rnorm(n,0,sqrt(v5))
> d6 = L61*F1 + L62*F2 + rnorm(n,0,sqrt(v6))
> d7 = L71*F1 + L72*F2 + rnorm(n,0,sqrt(v7))
> d8 = L81*F1 + L82*F2 + rnorm(n,0,sqrt(v8))
> dmat = cbind(d1,d2,d3,d4,d5,d6,d7,d8)
```

Again we fit a two-factor model with a varimax rotation.

```
> notsimple = factanal(dmat,factors=2,rotation='varimax'); notsimple

Call:
factanal(x = dmat, factors = 2, rotation = "varimax")

Uniquenesses:
   d1    d2    d3    d4    d5    d6    d7    d8
0.496 0.496 0.504 0.504 0.497 0.495 0.503 0.499

Loadings:
   Factor1 Factor2
d1          0.708
d2          0.708
d3          0.702
d4          0.702
d5  0.708
d6  0.709
d7  0.703
d8  0.706
```

```
                Factor1 Factor2
SS loadings       2.007   2.000
Proportion Var    0.251   0.250
Cumulative Var    0.251   0.501

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 9.58 on 13 degrees of freedom.
The p-value is 0.728
```

This time, only the estimates of communality (which are identifiable) and the goodness of fit test perform well. Everything else is awful. In particular, the estimates of the loadings are very similar to the estimates in the first example, and very far from the truth.

While the factor analysis for this second example clearly failed to yield a good estimate, it did yield a set of numbers that are only an orthogonal rotation away from an estimate that is very good indeed. If you think of the likelihood function as a high-dimensional mountain range, the maximum elevation is attained on a sort of ridge, with all points on the ridge at the same altitude. As the sample size increases, the ridge gets higher and higher, and its location changes a little bit, but less and less with increasing n. Meanwhile, the rest of the landscape melts into a featureless plain. The initial constrained maximum likelihood estimation lands you at one point on the ridge, and then an orthogonal rotation walks you along the ridge (say there is a path along the ridge)[24]. In these simulated data, the path actually passes very close to the true parameter value — very close indeed, since the sample size in this simulated data set is so large.

To find the point on the path that is closest to where the treasure is hidden, we will rotate the factor solution in the second example using a criterion that has not been mentioned before now. We will carry out a *Procrustes* rotation[25]. In Procustes rotation, the rotation matrix is chosen to minimize the difference between the matrix of estimated loadings and a target matrix, using least squares. There are orthogonal and oblique versions of Procrustes rotation. For our present purposes we want an orthogonal version. The `MCMCpack` package has a good one.

```
> # Procrustes rotation
> # install.packages("MCMCpack", dependencies=TRUE) # Only need to do this once
> library(MCMCpack)
Loading required package: coda
Loading required package: MASS
##
## Markov Chain Monte Carlo Package (MCMCpack)
```

[24] In this picture of the likelihood function, what is simple structure? Parameter values are literally coordinates, like latitude and longitude. This means that choosing a simple structure is like choosing a "good" location on the path, based on pleasing numerical values for the co-ordinates. For example, both latitude and longitude are integers, or divisible by eight. It's a lucky spot; let's stop here.

[25] Procrustes is a character in classic Greek mythology. He was a very bad man who would invite travellers to a free dinner and bed at his castle. Everybody fit the bed in the guest room, one way or the other. If travellers were too short, Procrustes would hammer them and stretch them with ropes until they fit. If they were too tall, he would cut off their feet. The survival rate for his guests was essentially zero. Then one day Theseus came along and gave Procrustes a taste of his own medicine.

```
## Copyright (C) 2003-2021 Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park
##
## Support provided by the U.S. National Science Foundation
## (Grants SES-0350646 and SES-0350613)
##
> # help(procrustes)
> L = notsimple$loadings; print(L,cutoff=0) # Factor loadings for the second example
Loadings:
   Factor1 Factor2
d1  0.047   0.708
d2  0.056   0.708
d3  0.054   0.702
d4  0.052   0.702
d5  0.708  -0.050
d6  0.709  -0.054
d7  0.703  -0.052
d8  0.706  -0.052

               Factor1 Factor2
SS loadings      2.007   2.000
Proportion Var   0.251   0.250
Cumulative Var   0.251   0.501
```

The target matrix will be the matrix of true factor loadings. Of course we can only do this because it's a simulation, and we know what the true parameter values are.

```
> Lambda = rbind(c(L11,L12),               # True factor loadings
+                c(L21,L22),
+                c(L31,L32),
+                c(L41,L42),
+                c(L51,L52),
+                c(L61,L62),
+                c(L71,L72),
+                c(L81,L82)  )
> Lambda # True Lambda -- How close can we get to this?
     [,1] [,2]
[1,]  0.5 -0.5
[2,]  0.5 -0.5
[3,]  0.5 -0.5
[4,]  0.5 -0.5
[5,]  0.5  0.5
[6,]  0.5  0.5
[7,]  0.5  0.5
[8,]  0.5  0.5
```

Now carry out the Procrustes rotation.

```
> pro = procrustes(X = L, Xstar = Lambda) # Rotate X to approximate Xstar.
> pro$X.new
        [,1]       [,2]
d1 0.4981332 -0.5056613
d2 0.5049341 -0.4994469
d3 0.4994946 -0.4962512
```

```
d4 0.4978127 -0.4979441
d5 0.5033950  0.5000182
d6 0.5014220  0.5038790
d7 0.4986956  0.4981095
d8 0.5003778  0.5008127
```

That's *very* close to the target. To see how close, look at it rounded and compare the result to `Lambda` above.

```
> round(pro$X.new,2)
   [,1]  [,2]
d1  0.5 -0.51
d2  0.5 -0.50
d3  0.5 -0.50
d4  0.5 -0.50
d5  0.5  0.50
d6  0.5  0.50
d7  0.5  0.50
d8  0.5  0.50
```

To really see how impressive this is, note that a Procruste rotation cannot fit an arbitrary target very well. In the final part of this example, the matrix `M` contains factor loadings that produce a covariance matrix very different from the one produced by `Lambda`. Can we rotate to fit this one?

```
> M  = rbind(c(0.30,0.64),
+            c(0.30,0.64),
+            c(0.30,0.64),
+            c(0.30,0.64),
+            c(0.30,0.64),
+            c(0.30,0.64),
+            c(0.30,0.64)  ); M
     [,1] [,2]
[1,]  0.3 0.64
[2,]  0.3 0.64
[3,]  0.3 0.64
[4,]  0.3 0.64
[5,]  0.3 0.64
[6,]  0.3 0.64
[7,]  0.3 0.64
[8,]  0.3 0.64
> procrustes(X = L, Xstar = M)$X.new
         [,1]      [,2]
d1 -0.2470153 0.6654424
d2 -0.2385050 0.6689701
d3 -0.2379146 0.6626890
d4 -0.2401607 0.6618827
d5  0.6661897 0.2441638
d6  0.6688511 0.2407410
d7  0.6624699 0.2407156
d8  0.6656312 0.2410942
```

So the closest one can get to this particular target with an orthogonal rotation is ridiculously far away.

The main point here that for the second simulated data example, the one where varimax rotation failed, an excellent estimate is actually somewhere in the collection of factor matrices that can be reached by an orthogonal rotation. The problem is that we don't know which one. This is true for real data sets, too. I cannot think of any exceptions.

Oblique rotations If anything, the problem is a bit worse with oblique rotations, because they can miss the truth and find an inferior solution, even if the true factor loadings typify simple structure. For the next example, there will be three factors. The first factor is independent of the others, but factors two and three are highly correlated. There are nine observed variables. The first three variables load only on factor one, the second three load only on factor two, and the last three load only on factor three. That's a clear example of simple structure.

```
> Phi = rbind(c(1.0, 0.0, 0.0),
+             c(0.0, 1.0, 0.9),
+             c(0.0, 0.9, 1.0))
>
> Lambda = rbind(c(0.9, 0.0, 0.0),
+                c(0.9, 0.0, 0.0),
+                c(0.9, 0.0, 0.0),
+                c(0.0, 0.9, 0.0),
+                c(0.0, 0.9, 0.0),
+                c(0.0, 0.9, 0.0),
+                c(0.0, 0.0, 0.9),
+                c(0.0, 0.0, 0.9),
+                c(0.0, 0.0, 0.9)  )
```

The standardized model will hold exactly in the population. For this, it is necessary to calculate the matrix $\mathbf{\Omega}$ in $cov(\mathbf{z}) = cov(\mathbf{\Lambda F} + \mathbf{e}) = \mathbf{\Lambda \Phi \Lambda}^\top + \mathbf{\Omega}$.

```
> # Calculate Omega, the 9 x 9 covariance matrix of the error terms.
> Lambda %*% Phi %*% t(Lambda)
      [,1] [,2] [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
 [1,] 0.81 0.81 0.81 0.000 0.000 0.000 0.000 0.000 0.000
 [2,] 0.81 0.81 0.81 0.000 0.000 0.000 0.000 0.000 0.000
 [3,] 0.81 0.81 0.81 0.000 0.000 0.000 0.000 0.000 0.000
 [4,] 0.00 0.00 0.00 0.810 0.810 0.810 0.729 0.729 0.729
 [5,] 0.00 0.00 0.00 0.810 0.810 0.810 0.729 0.729 0.729
 [6,] 0.00 0.00 0.00 0.810 0.810 0.810 0.729 0.729 0.729
 [7,] 0.00 0.00 0.00 0.729 0.729 0.729 0.810 0.810 0.810
 [8,] 0.00 0.00 0.00 0.729 0.729 0.729 0.810 0.810 0.810
 [9,] 0.00 0.00 0.00 0.729 0.729 0.729 0.810 0.810 0.810
> diag(Lambda %*% Phi %*% t(Lambda))
[1] 0.81 0.81 0.81 0.81 0.81 0.81 0.81 0.81 0.81
> Omega = diag(1-0.81,nrow=9,ncol=9); Omega
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
```

```
 [1,] 0.19 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [2,] 0.00 0.19 0.00 0.00 0.00 0.00 0.00 0.00 0.00
 [3,] 0.00 0.00 0.19 0.00 0.00 0.00 0.00 0.00 0.00
 [4,] 0.00 0.00 0.00 0.19 0.00 0.00 0.00 0.00 0.00
 [5,] 0.00 0.00 0.00 0.00 0.19 0.00 0.00 0.00 0.00
 [6,] 0.00 0.00 0.00 0.00 0.00 0.19 0.00 0.00 0.00
 [7,] 0.00 0.00 0.00 0.00 0.00 0.00 0.19 0.00 0.00
 [8,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.19 0.00
 [9,] 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.19
```

Now we will generate the random data set. We need a function for simulating multivariate normal data. Such a function is available in several packages, but I prefer one that I wrote; it is available for download and free for public use under the usual GNU conditions. As you can see from the code, it uses spectral decomposition to transform a set of standard normals into a multivariate normal.

```
> rm(list=ls())
> # Need function for simulating multivariate normal data.
> source("http://www.utstat.toronto.edu/~brunner/Rfunctions/rmvn.txt")
> rmvn # Type the function name to see the code.
function(nn,mu,sigma)
# Returns an nn by kk matrix, rows are independent MVN(mu,sigma)

    kk <- length(mu)
    dsig <- dim(sigma)
    if(dsig[1] != dsig[2]) stop("Sigma must be square.")
    if(dsig[1] != kk) stop("Sizes of sigma and mu are inconsistent.")
    ev <- eigen(sigma)
    sqrl <- diag(sqrt(ev$values))
    PP <- ev$vectors
    ZZ <- rnorm(nn*kk) ; dim(ZZ) <- c(kk,nn)
    rmvn <- t(PP%*%sqrl%*%ZZ+mu)
    rmvn
```

In the simulation below, the large sample size of $n = 10,000$ means that the results will not be blurred much by sampling error.

```
> # Generate data
> set.seed(9999)
> n = 10000
> Fac = rmvn(n,mu=c(0,0,0),sigma=Phi)     # n x 3 matrix of factor values
> err = rmvn(n,mu=numeric(9),sigma=Omega) # n x 9 matrix of error terms
>
> #     n x 3       3 x 9     n x 9
> dat =  Fac  %*% t(Lambda) +  err
```

Compare the sample correlation matrix to the true correlation matrix. They are close, as one would expect with this sample size. Of course this is a way to check for mistakes in calculation or programming.

```
> round(cor(dat),3)
       [,1]   [,2]   [,3]   [,4]   [,5]  [,6]   [,7]   [,8]   [,9]
 [1,]  1.000  0.808  0.806 -0.004 -0.001 0.008  0.004  0.003 -0.002
 [2,]  0.808  1.000  0.809 -0.001 -0.006 0.000 -0.002 -0.003 -0.006
 [3,]  0.806  0.809  1.000 -0.002 -0.007 0.003 -0.002 -0.001 -0.004
 [4,] -0.004 -0.001 -0.002  1.000  0.811 0.809  0.731  0.732  0.730
 [5,] -0.001 -0.006 -0.007  0.811  1.000 0.812  0.726  0.728  0.725
 [6,]  0.008  0.000  0.003  0.809  0.812 1.000  0.729  0.730  0.726
 [7,]  0.004 -0.002 -0.002  0.731  0.726 0.729  1.000  0.810  0.809
 [8,]  0.003 -0.003 -0.001  0.732  0.728 0.730  0.810  1.000  0.808
 [9,] -0.002 -0.006 -0.004  0.730  0.725 0.726  0.809  0.808  1.000
> Lambda %*% Phi %*% t(Lambda) + Omega # Compare
      [,1] [,2] [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
 [1,] 1.00 0.81 0.81 0.000 0.000 0.000 0.000 0.000 0.000
 [2,] 0.81 1.00 0.81 0.000 0.000 0.000 0.000 0.000 0.000
 [3,] 0.81 0.81 1.00 0.000 0.000 0.000 0.000 0.000 0.000
 [4,] 0.00 0.00 0.00 1.000 0.810 0.810 0.729 0.729 0.729
 [5,] 0.00 0.00 0.00 0.810 1.000 0.810 0.729 0.729 0.729
 [6,] 0.00 0.00 0.00 0.810 0.810 1.000 0.729 0.729 0.729
 [7,] 0.00 0.00 0.00 0.729 0.729 0.729 1.000 0.810 0.810
 [8,] 0.00 0.00 0.00 0.729 0.729 0.729 0.810 1.000 0.810
 [9,] 0.00 0.00 0.00 0.729 0.729 0.729 0.810 0.810 1.000
```

When I decided on this example, I thought that common methods for determining the number of factors might fail, because it could be hard to tell the two highly correlated factors from a single factor. Indeed, there are two eigenvalues greater than one, and the others are not even close; this points to two factors. Other common tests for number of factors give varying results. However, testing for goodness of fit performed really well. Fitting a model with just two factors,

```
> factanal(dat,factors=2)

Call:
factanal(x = dat, factors = 2)

Uniquenesses:
[1] 0.195 0.189 0.192 0.235 0.239 0.238 0.240 0.239 0.243

Loadings:
      Factor1 Factor2
 [1,]          0.897
 [2,]          0.900
 [3,]          0.899
 [4,]  0.875
 [5,]  0.872
 [6,]  0.873
 [7,]  0.872
 [8,]  0.873
 [9,]  0.870

                Factor1 Factor2
SS loadings       4.566   2.424
```

```
Proportion Var   0.507   0.269
Cumulative Var   0.507   0.777

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 3179.9 on 19 degrees of freedom.
The p-value is 0
```

Varimax was fooled into thinking that the last six variables all came from the same factor, but the two-factor model did not come close to fitting. Trying a three-factor model,

```
> factanal(dat,factors=3)

Call:
factanal(x = dat, factors = 3)

Uniquenesses:
[1] 0.195 0.189 0.192 0.193 0.185 0.190 0.188 0.192 0.193

Loadings:
      Factor1 Factor2 Factor3
 [1,]          0.897
 [2,]          0.900
 [3,]          0.899
 [4,]  0.878          -0.191
 [5,]  0.878          -0.212
 [6,]  0.877          -0.201
 [7,]  0.877           0.205
 [8,]  0.877           0.197
 [9,]  0.875           0.204

               Factor1 Factor2 Factor3
SS loadings      4.615   2.424   0.244
Proportion Var   0.513   0.269   0.027
Cumulative Var   0.513   0.782   0.809

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 12.73 on 12 degrees of freedom.
The p-value is 0.389
```

This model fits nicely; the goodness of fit test located the true number of factors. This has been my experience with uncorrelated factors, too. The chi-squared test for goodness of fit is an excellent tool for determining the number of factors, and the larger the sample size, the better it gets — with simulated data. Of course that's the problem. We must bear in mind the suggestion that for any real data set, there could easily be hundreds of common factors. When this is true, no model will fit if the sample size is large enough.

In any case, suppose we know that there are three factors. The true matrix of factor loadings is an extreme example of simple structure. Can oblimin find it? If the factors were uncorrelated, one coud trust varimax to locate this easy truth. The first attempt will use the default setting of $\gamma = 0$.

```
> # install.packages("GPArotation", dependencies=TRUE) # Only need to do this once
> library(GPArotation)
>
> threefac = factanal(dat,factors=3,rotation='none'); Ahat = threefac$loadings
> options(scipen=999) # Suppress scientific notation for now
> oblimin(Ahat)
Oblique rotation method Oblimin Quartimin converged.
Loadings:
        Factor1  Factor2    Factor3
 [1,]  0.002820  0.89720  0.0024216
 [2,] -0.001803  0.90038 -0.0022990
 [3,] -0.000956  0.89891  0.0000235
 [4,]  0.878590 -0.00141 -0.1905515
 [5,]  0.878124 -0.00382 -0.2111222
 [6,]  0.877928  0.00538 -0.2005067
 [7,]  0.876644  0.00142  0.2058478
 [8,]  0.876572  0.00120  0.1972533
 [9,]  0.874316 -0.00314  0.2046256

Rotating matrix:
         [,1]      [,2]       [,3]
[1,]  1.00000 -0.001673 -0.0003306
[2,]  0.00307  1.000000 -0.0000706
[3,] -0.00201  0.000551  1.0000028

Phi:
         [,1]      [,2]      [,3]
[1,]  1.00000 -0.001395  0.002345
[2,] -0.00140  1.000000 -0.000484
[3,]  0.00234 -0.000484  1.000000
```

Both the $\widehat{\mathbf{\Lambda}}$ and $\widehat{\mathbf{\Phi}}$ matrices are way off. $\widehat{\mathbf{\Phi}}$ is nearly the identity, and $\widehat{\mathbf{\Lambda}}$ is essentially the varimax solution. Increasing the value of γ to encourage more highly correlated factors,

```
> oblimin(Ahat, gam = 0.5) # For more highly correlated factors (truth).
Oblique rotation method Oblimin Biquartimin converged.
Loadings:
      Factor1 Factor2 Factor3
 [1,]   0.117  1.0222   0.220
 [2,]   0.114  1.0234   0.215
 [3,]   0.114  1.0227   0.217
 [4,]   0.978  0.0304  -0.291
 [5,]   0.983  0.0197  -0.317
 [6,]   0.981  0.0342  -0.301
 [7,]   0.864  0.1860   0.195
 [8,]   0.866  0.1825   0.184
 [9,]   0.861  0.1801   0.193

Rotating matrix:
       [,1]  [,2]    [,3]
[1,]  1.052 0.118 -0.0661
[2,]  0.131 1.138  0.2414
```

```
[3,] -0.286 0.385  1.2240

Phi:
       [,1]   [,2]   [,3]
[1,]  1.000 -0.313  0.396
[2,] -0.313  1.000 -0.551
[3,]  0.396 -0.551  1.000
```

Once again, the results are nowhere near the true parameter values. Increasing the value of γ once again,

```
> oblimin(Ahat, gam = 0.75)
Oblique rotation method Oblimin g=0.75 NOT converged.
Loadings:
      Factor1 Factor2 Factor3
 [1,]    4.31   9.678    4.72
 [2,]    4.28   9.650    4.72
 [3,]    4.28   9.662    4.73
 [4,]    7.81  -0.429   -8.08
 [5,]    7.76  -0.685   -8.30
 [6,]    7.82  -0.468   -8.14
 [7,]    8.43   4.017   -4.01
 [8,]    8.42   3.919   -4.10
 [9,]    8.39   3.950   -4.03

Rotating matrix:
     [,1]  [,2]  [,3]
[1,] 9.23  1.93 -6.99
[2,] 4.80 10.76  5.24
[3,] 1.57 11.15 10.21

Phi:
       [,1]   [,2]   [,3]
[1,]  1.000 -0.995  0.994
[2,] -0.995  1.000 -0.997
[3,]  0.994 -0.997  1.000
Warning message:
In GPFoblq(L, Tmat = Tmat, normalize = normalize, eps = eps, maxit = maxit,  :
  convergence not obtained in GPFoblq. 1000 iterations used.
```

This time, the algorithm did not converge (this is common with "large" positive values of γ), and the estimates are to be ignored. They are just the current values when the job ran out of iterations. The value of the oblimin criterion was marching off to $-\infty$.

Lowering the value of γ a bit,

```
> oblimin(Ahat, gam = 0.6)
Oblique rotation method Oblimin g=0.6 converged.
Loadings:
      Factor1 Factor2   Factor3
 [1,]   0.306  1.3650  0.427736
 [2,]   0.301  1.3641  0.423094
 [3,]   0.301  1.3642  0.425963
```

```
 [4,]   1.213  0.1018 -0.651437
 [5,]   1.216  0.0792 -0.686801
 [6,]   1.217  0.1028 -0.664617
 [7,]   1.138  0.4683  0.013499
 [8,]   1.140  0.4600 -0.000951
 [9,]   1.134  0.4595  0.010177

Rotating matrix:
       [,1]  [,2]   [,3]
[1,]  1.341 0.314 -0.379
[2,]  0.341 1.519  0.472
[3,] -0.188 0.915  1.673

Phi:
       [,1]   [,2]   [,3]
[1,]  1.000 -0.669  0.659
[2,] -0.669  1.000 -0.812
[3,]  0.659 -0.812  1.000
```

This time, the maximum absolute correlation between factors is in the right vicinity, but the values of the estimated correlations are way off, and the estimated factor loadings are nowhere near the truth.

It is clear that adjusting the value of γ does not help at all. Possibly the numerical search is getting caught in a local minimum. By default, the search starts with the transformation matrix $\mathbf{T}$ equal to the identity. Using a combination of calculation and guesswork (the details are not important), I came up with a promising $\mathbf{T}$ matrix, denoted by `T_try`.

```
> T_try
                [,1]         [,2]        [,3]
Factor1 -0.001603061  0.975610583 0.973941825
Factor2  0.999998098  0.002998459 0.002938147
Factor3  0.001110803 -0.219488039 0.226778941
```

This transformation matrix reproduces the true correlations between factors and the true factor loadings quite well. Checking $\mathbf{T}^\top\mathbf{T} = \mathbf{\Phi}$,

```
# Test T_try
> M = t(T_try) %*% T_try; round(M,2)
     [,1] [,2] [,3]
[1,]    1  0.0  0.0
[2,]    0  1.0  0.9
[3,]    0  0.9  1.0
> Phi
     [,1] [,2] [,3]
[1,]    1  0.0  0.0
[2,]    0  1.0  0.9
[3,]    0  0.9  1.0
```

The match is perfect, to two decimal places. Now try $\mathbf{\Lambda} = \mathbf{A}\left(\mathbf{T}^\top\right)^{-1}$.

```
> round(Ahat %*% solve(t(T_try)), 2) # Compare Lambda
      [,1] [,2]  [,3]
 [1,]  0.9 0.00  0.00
 [2,]  0.9 0.01 -0.01
 [3,]  0.9 0.00  0.00
 [4,]  0.0 0.88  0.02
 [5,]  0.0 0.93 -0.03
 [6,]  0.0 0.91 -0.01
 [7,]  0.0 0.00  0.90
 [8,]  0.0 0.01  0.89
 [9,]  0.0 0.00  0.90
> Lambda
      [,1] [,2] [,3]
 [1,]  0.9  0.0  0.0
 [2,]  0.9  0.0  0.0
 [3,]  0.9  0.0  0.0
 [4,]  0.0  0.9  0.0
 [5,]  0.0  0.9  0.0
 [6,]  0.0  0.9  0.0
 [7,]  0.0  0.0  0.9
 [8,]  0.0  0.0  0.9
 [9,]  0.0  0.0  0.9
```

The reason it's possible to approximate $\mathbf{\Phi}$ and $\mathbf{\Lambda}$ so well is the large sample size. Of course, as in the orthogonal case, there are infinitely many other $\mathbf{T}$ matrices that fit the data equally well. When `T_try` is used as a starting value, the simple structure in $\widehat{\mathbf{\Lambda}}$ ensures that the numerical search stays very close to where it started.

```
> # Use T_try as a starting value
> oblimin(Ahat, Tmat=T_try, gam=0)
Oblique rotation method Oblimin Quartimin converged.
Loadings:
       Factor1   Factor2  Factor3
 [1,]  0.89721 -0.003859  0.00671
 [2,]  0.90038  0.004258 -0.00617
 [3,]  0.89891 -0.000471 -0.00056
 [4,] -0.00139  0.876491  0.02450
 [5,] -0.00381  0.922002 -0.02156
 [6,]  0.00540  0.898292  0.00198
 [7,]  0.00159 -0.006151  0.90648
 [8,]  0.00136  0.012927  0.88730
 [9,] -0.00297 -0.004632  0.90257

Rotating matrix:
             [,1]     [,2]    [,3]
Factor1 -0.001572  0.51597 0.51025
Factor2  1.000000  0.00182 0.00127
Factor3  0.000933 -2.22516 2.22648

Phi:
         [,1]     [,2]     [,3]
[1,]  1.00000 -0.00122 -0.00159
```

```
[2,] -0.00122  1.00000  0.89908
[3,] -0.00159  0.89908  1.00000
```

One could not ask for nicer results. Notice how a large value of γ is not necessary to get a high estimated correlation between factors. Furthermore, the oblimin criterion is actually *lower* for this solution than for the one with the default starting value, so the earlier search found a local minimum that was higher than the global minimum. Here's how to tell.

An `oblimin` object is a list, and one of the items in the list is a table showing the iteration history. For some reason, the table is called `Table`. The second column of the table gives the value of the oblimin criterion. There is one row in the table for each iteration, so tables can be quite long. We will use R's `tail` function to look at just the last four lines of the tables. First comes the one with the default starting value for $\mathbf{T}$ (the identity), and then the one starting with `T_try`.

```
> tail(oblimin(Ahat, gam=0)$Table)
      [,1]       [,2]      [,3] [,4]
[91,]   90 0.09396019 -4.936630 0.50
[92,]   91 0.09396019 -4.947460 0.50
[93,]   92 0.09396019 -4.958262 0.50
[94,]   93 0.09396019 -4.969037 0.50
[95,]   94 0.09396019 -4.745909 1.00
[96,]   95 0.09396019 -5.054387 0.25
> tail(oblimin(Ahat, Tmat=T_try, gam=0)$Table)
      [,1]         [,2]      [,3]   [,4]
[38,]   37 0.0005909207 -4.741182 0.1250
[39,]   38 0.0005909207 -4.949552 0.0625
[40,]   39 0.0005909207 -4.985353 0.1250
[41,]   40 0.0005909207 -4.991386 0.1250
[42,]   41 0.0005909207 -4.950799 0.1250
[43,]   42 0.0005909207 -5.158412 0.0625
```

Starting with `T_try` was possible only because I knew the true $\mathbf{\Lambda}$ matrix. The key to finding such a hidden solution with real data (if one exists) is to try different starting values for $\mathbf{T}$. The `GPArotation` package has a useful function called `Random.Start`, which generates a random orthogonal matrix. The single argument of the function `Random.Start` is the number of rows and columns. While the transformation matrix $\mathbf{T}$ is not constrained to be orthogonal, the non-zero off-diagonal elements mix things up enough so that it works quite well. What I did was to execute the following code repeatedly until something interesting happened.

```
> oblimin(Ahat, Tmat=Random.Start(3))
```

After just three tries, I got the following.

```
Oblique rotation method Oblimin Quartimin converged.
Loadings:
       Factor1   Factor2   Factor3
 [1,] -0.89721  0.006711  0.003860
```

```
 [2,] -0.90038 -0.006167 -0.004258
 [3,] -0.89891 -0.000561  0.000472
 [4,]  0.00139  0.024497 -0.876491
 [5,]  0.00381 -0.021562 -0.922002
 [6,] -0.00540  0.001983 -0.898292
 [7,] -0.00159  0.906484  0.006151
 [8,] -0.00136  0.887303 -0.012927
 [9,]  0.00297  0.902572  0.004632

Rotating matrix:
          [,1]    [,2]     [,3]
[1,]  0.001572 0.51025 -0.51597
[2,] -1.000000 0.00127 -0.00182
[3,] -0.000933 2.22648  2.22516

Phi:
         [,1]     [,2]     [,3]
[1,]  1.00000  0.00159 -0.00122
[2,]  0.00159  1.00000 -0.89908
[3,] -0.00122 -0.89908  1.00000
```

This is the same solution obtained using `T_try` as a starting value, except that the the signs of all the factor loadings for factors one and three are reversed, and the correlation between factors two and three is negative instead of positive. This is perfectly good. Since the oblimin criterion is a function of the *squared* factor loadings, switching the signs of the loadings in any column produces the same value of the function being minimized, and the function has at least 2^p local minima. As in orthogonal factor analysis, one may reflect factors at will, and the only consequence is the word one uses to describe the factor. One may call it "anti-racism" instead of "racism," or "mental health" instead of "mental illness." It is entirely a matter of convenience. Naturally, when one does this one must also switch the signs of the correlations between the factor in question and all the other factors. That is what has happened here.

Continuing to execute the code, on the eleventh try I got a version of the correct solution with only factor three reflected, and on the thirteenth try I got a version with only factor one reflected.

The conclusion is that if the true factor pattern has a simple structure, oblimin rotation may miss it unless one tries numerous starting values[26]. In fact, even if the truth has a fairly simple structure, there may be another solution that fits the data just as well and which has a structure that is even simpler. In this case, multiple starting values will lead you to the answer that is prettier, but wrong. Of course, if the truth does not happen to be simple, there is no hope at all.

Frequently, simulation studies involve thousands, or even millions of random data sets. Here, you really only need two simulated data sets to see how unsuccessful exploratory

[26] I tried `Random.Start` a large number of times with the Mind-body data, and got the same results each time apart from reflections.

factor analysis can be. It all depends on how closely the true pattern of factor loadings approximates simple structure. If the truth looks like the result of a varimax rotation, then a varimax rotation will probably find it – or it will find something equivalent, with one or more factors reflected. If the truth does not resemble a varimax rotation, then a varimax rotation will settle on a simple structure that may be quite different from the truth. Should we expect the truth in any particular field to resemble simple structure? I really can't see why.

The factor analysts have an answer, and it goes back to the early days, from the time when the indeterminacy of factor solutions was first recognized. The argument is that a factor solution is essentially a scientific theory of the data. In the philosophy of science, it is widely accepted that there can be many different theories that fit a set of data equally well. In this situation, a principle known as Occam's razor says that all other things being equal, a simpler explanation is better. Thus, the simple structure located by a varimax or some other good rotation method is the preferred estimate.

My response is that while the factor analysis model itself is like a scientific theory, the unknown constants in the model are numerical quantities that are subject to estimation, like the speed of light in relativity theory. In the case of unconstrained exploratory factor analysis, lack of parameter identifiability means that there are infinitely many potential estimates that are equally reasonable given the data. Choosing a set of numerical values that tells a pleasing story is one option, but the truth may be much closer to a completely different set of values – one that is equally compatible with the data. Viewed as a method of statistical estimation, exploratory factor analysis is a failure, period. It is something a statistician should never do, except perhaps for money.

2.7 Rotating Principal Components

Something can be salvaged from all this. Rotation is what makes factor analysis results understandable. In R, a nice thing about the stand-alone `varimax` function is that it can also be used to rotate principal components. The result is a set of uncorrelated linear combinations of the variables that explain exactly the same amount of variance as the original components, but are easier to interpret. This section is a bit of a digression, but the end product is a useful data analysis trick.

From Section 2.1, we have the $k \times 1$ standardized data vector $\mathbf{z}$, the correlation matrix $cov(\mathbf{z}) = \boldsymbol{\Sigma}$, the spectral decomposition $\boldsymbol{\Sigma} = \mathbf{CDC}^\top$, and the vector of principal components $\mathbf{y} = \mathbf{C}^\top\mathbf{z}$. The ordered eigenvalues in the diagonal matrix $\mathbf{D}$ are both the variances of the principal components and the amounts of variance in $\mathbf{z}$ that they explain. It is

helpful to calculate the matrix of correlations

$$\begin{aligned} corr(\mathbf{z},\mathbf{y}) &= cov(\mathbf{z},\mathbf{D}^{-1/2}\mathbf{y}) \\ &= cov(\mathbf{z},\mathbf{D}^{-1/2}\mathbf{C}^\top\mathbf{z}) \\ &= cov(\mathbf{z})\left(\mathbf{D}^{-1/2}\mathbf{C}^\top\right)^\top \\ &= \mathbf{\Sigma}\mathbf{C}\mathbf{D}^{-1/2} \\ &= \mathbf{C}\mathbf{D}\underbrace{\mathbf{C}^\top\mathbf{C}}_{\mathbf{I}}\mathbf{D}^{-1/2} \\ &= \mathbf{C}\mathbf{D}\mathbf{D}^{-1/2} \\ &= \mathbf{C}\mathbf{D}^{1/2}, \end{aligned} \tag{2.25}$$

a formula equivalent to the scalar version (2.3).

We don't retain all the principal components. Instead, we summarize the variables with a smaller set of p principal components that explain a good part of the total variance. Typically, components associated with eigenvalues greater than one are retained. This may be accomplished with a $p\times k$ *selection matrix* that will be denoted by $\mathbf{S}$ (for selection), and is not to be mistaken for a sample covariance matrix. Each row of $\mathbf{S}$ has a one in the position of a component to be retained, and the rest zeros. For example, if there were five principal components, the first two may be selected as follows.

$$\mathbf{S}\mathbf{y} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

If $\mathbf{A}$ is any $k \times k$ matrix, then $\mathbf{S}\mathbf{A}\mathbf{S}^\top$ is the $p \times p$ sub-matrix with rows and columns indicated by $\mathbf{S}$. A sub-matrix of the identity is another (smaller) identity matrix, so $\mathbf{S}\mathbf{S}^\top = \mathbf{I}_p$. Selection matrices are quite flexible and can even be used to re-order variables, but here they will just be used to select the first p principal components.

Simply rotating a set of selected principal components is not a good choice, because the resulting linear combinations are correlated.

$$\begin{aligned} cov(\mathbf{R}\mathbf{S}\mathbf{y}) &= \mathbf{R}\mathbf{S}cov(\mathbf{y})\left(\mathbf{R}\mathbf{S}\right)^\top \\ &= \mathbf{R}\mathbf{S}\mathbf{D}\mathbf{S}^\top\mathbf{R}^\top, \end{aligned}$$

a matrix that in general will not be diagonal unless all the eigenvalues equal one. Because the eigenvalues are the variances of the principal components, this suggests standardizing the principal components before rotating them. It is more convenient (mathematically, not computationally) to standardize first, and then select. The result is

$$\begin{aligned} \mathbf{f} &= \mathbf{S}\mathbf{D}^{-1/2}\mathbf{y} \\ &= \mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\mathbf{z}. \end{aligned}$$

The notation $\mathbf{f}$ is meant to suggest that the standardized principal components are analogous to factors, even though they are not really factors.

Applying a rotation to $\mathbf{f}$, we have $\mathbf{f}' = \mathbf{R}\mathbf{f}$, with covariance matrix

$$\begin{aligned} cov(\mathbf{f}') &= cov(\mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\mathbf{z}) \\ &= \mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top cov(\mathbf{z})\left(\mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\right)^\top \\ &= \mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\boldsymbol{\Sigma}\mathbf{C}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\mathbf{C}\mathbf{D}\mathbf{C}^\top\mathbf{C}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\underbrace{\mathbf{C}^\top\mathbf{C}}_{\mathbf{I}}\mathbf{D}\underbrace{\mathbf{C}^\top\mathbf{C}}_{\mathbf{I}}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{R}\mathbf{S}\underbrace{\mathbf{D}^{-1/2}\mathbf{D}\mathbf{D}^{-1/2}}_{\mathbf{I}}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{R}\underbrace{\mathbf{S}\mathbf{S}^\top}_{\mathbf{I}}\mathbf{R}^\top \\ &= \mathbf{R}\mathbf{R}^\top \\ &= \mathbf{I}. \end{aligned} \tag{2.26}$$

Thus, by scaling[27] the selected principal components and *then* rotating, we obtain linear combinations that are uncorrelated. Since their expected values are zero and their variances are one, they are still standardized after rotation.

The $k \times p$ matrix of correlations between the original variables and the rotated components is

$$\begin{aligned} corr(\mathbf{z}, \mathbf{f}') = cov(\mathbf{z}, \mathbf{f}') &= cov(\mathbf{z}, \mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\mathbf{z}) \\ &= cov(\mathbf{z})\left(\mathbf{R}\mathbf{S}\mathbf{D}^{-1/2}\mathbf{C}^\top\right)^\top \\ &= \boldsymbol{\Sigma}\,\mathbf{C}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{C}\mathbf{D}\underbrace{\mathbf{C}^\top\mathbf{C}}_{\mathbf{I}}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{C}\mathbf{D}\mathbf{D}^{-1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= \mathbf{C}\mathbf{D}^{1/2}\mathbf{S}^\top\mathbf{R}^\top \\ &= corr(\mathbf{z}, \mathbf{y})\mathbf{S}^\top\mathbf{R}^\top \end{aligned}$$

from (2.25).

Since $corr(\mathbf{z}, \mathbf{y})\mathbf{S}^\top$ is just the first p columns of the $k \times k$ matrix $corr(\mathbf{z}, \mathbf{y})$, we can select principal components first and then compute the correlations, yielding

$$corr(\mathbf{z}, \mathbf{f}') = corr(\mathbf{z}, \mathbf{S}\mathbf{y})\mathbf{R}^\top. \tag{2.27}$$

Furthermore, scaling and rotation does not affect the amount of variance explained by the first p components. By (2.2) and (2.3), the variance in z_j explained by the first

[27] Scaling the components to have variance one is the same as standardizing, because they already have expected value zero.

p components is the sum of the squared correlations between z_j and those components. There are k such quantities, one for each observed variable. They are the diagonal elements of the matrix $corr(\mathbf{z}, \mathbf{Sy})corr(\mathbf{z}, \mathbf{Sy})^\top$.

By (2.27), the corresponding sums of squared correlations between the variables and the scaled and rotated components are on the main diagonal of

$$\begin{aligned} corr(\mathbf{z}, \mathbf{f}')corr(\mathbf{z}, \mathbf{f}')^\top &= corr(\mathbf{z}, \mathbf{Sy})\mathbf{R}^\top \left(corr(\mathbf{z}, \mathbf{Sy})\mathbf{R}^\top\right)^\top \\ &= corr(\mathbf{z}, \mathbf{Sy}) \underbrace{\mathbf{R}^\top \mathbf{R}}_{\mathbf{I}} corr(\mathbf{z}, \mathbf{Sy})^\top \\ &= corr(\mathbf{z}, \mathbf{Sy})corr(\mathbf{z}, \mathbf{Sy})^\top. \end{aligned}$$

That is, for each variable, the sum of squared correlations with the first p original components is the same as the sum of squared correlations with the scaled and rotated components $\mathbf{f}'$.

It remains to show that the sum of squared correlations of the variables with $\mathbf{f}'$ is the variance explained by $\mathbf{f}'$. This is true because

1. Following the calculations leading to (2.3), we have this general result. Let the random variable $w = a_1x_1 + \cdots + a_kx_k$, where $a_, \ldots, a_k$ are non-zero constants, $Var(x_j) = \sigma_j^2$, and $Cov(x_i, x_j) = 0$ for $i \neq j$. Then the variance in w that is explained by a subset of x variables is the sum of their squared correlations with w.

2. For $i = 1, \ldots, k$, $z_i = a_{i,1}f'_1 + \cdots + a_{i,p}f'_p + c_{i,p+1}y_{p+1} + \cdots + c_{i,k}y_k$, where $\mathbf{f}' = [f'_j]$. It is a homework problem to write a matrix expression for the $a_{i,j}$.

3. The matrix of covariances between $\mathbf{f}'$ and the principal components $y_{p+1}, \ldots, y_k$ is zero.

The conclusion is that for each variable, the variance explained by the rotated linear combinations $\mathbf{f}'$ is equal to the variance explained by the first p original components.

To summarize, one can select the first p out of k principal components, and then scale them to have variance one. This yields $\mathbf{f}$. Applying a rotation (or reflection) yields $\mathbf{f}' = \mathbf{Rf}$. The random variables in $\mathbf{f}'$ have these properties:

- They are uncorrelated.
- They explain the same amount of variance as the first p principal components.
- Their correlations with the observed variables are equal to the correlations of the first p principal components with the observed variables, but post-multiplied by the transpose of the rotation matrix. This is equation (2.27).

All this holds for any $p \times p$ rotation matrix — that is, for any orthogonal matrix $\mathbf{R}$.

Now, it is not at all mandatory to scale and rotate the principal components, but it can be useful, because the original components, though unique, are often difficult to understand in terms of the input variables. Rotation to something approaching simple

structure can result in linear combinations of the variables that are uncorrelated, collectively just as good as the principal components in terms of explaining variance, and also easy to understand. The only thing that is lost is the property that the first one explains the most possible variance, and so on.

The mechanics of rotation can be directly borrowed from factor analysis. Recalling factor analysis with rotation,

$$\begin{aligned} \mathbf{z} &= \mathbf{\Lambda F} + \mathbf{e} \\ &= (\mathbf{\Lambda R}^\top)(\mathbf{RF}) + \mathbf{e} \\ &= (\mathbf{\Lambda R}^\top)\mathbf{F}' + \mathbf{e}, \end{aligned}$$

where $\mathbf{F}'$ denotes the rotated factors. Based on an initial solution $\widehat{\mathbf{\Lambda}}$, the rotation matrix $\mathbf{R}$ is chosen so that $\widehat{\mathbf{\Lambda}}\mathbf{R}^\top$ has a simple structure.

Comparing Equation (2.27) to the corresponding results for factor analysis,

$$corr(\mathbf{z}, \mathbf{f}') = corr(\mathbf{z}, \mathbf{Sy})\mathbf{R}^\top \quad corr(\mathbf{z}, \mathbf{F}') = corr(\mathbf{z}, \mathbf{F})\mathbf{R}^\top.$$

So, one can simply take the matrix of sample correlations between the variables and the first p principal components, and hand it to a rotation algorithm like varimax. The result will be simplified matrix of correlations between the variables and and a set of rotated components $\mathbf{f}'$ – as well as the rotation matrix that gets the job done.

Illustrating with the Mind-body data, we begin with `pc2`, the earlier `prcomp` object that retained just the two principal components, the ones with eigenvalues greater than one.

```
> pc2 = prcomp(dat, scale = T, rank=2)
> ls(pc2)
[1] "center"   "rotation" "scale"    "sdev"     "x"
```

The list element `pc2$x` is an $n \times 2$ matrix of the two principal components that are retained. Looking at the correlations of these principal components with the variables,

```
> cor(dat,pc2$x)
               PC1        PC2
progmat -0.4709330 -0.6299014
reason  -0.4981509 -0.7277446
verbal  -0.5519561 -0.6910097
headlng -0.7500678  0.1156757
headbrd -0.6073970  0.3689507
headcir -0.9063741  0.1686041
bizyg   -0.8298157  0.2293757
weight  -0.7274347  0.2792455
height  -0.7364050  0.2490749
```

Correlations of raw (unrotated) principal components with variables are always hard to understand, but the minus signs make it worse. We can just flip the signs and everything still correct, because correlations between variables and principal components have the same signs as eigenvector elements. The definition of an eigenvector and corresponding eigenvalue is $\mathbf{Ax} = \lambda\mathbf{x}$. Thus, if $\mathbf{x}$ is an eigenvector corresponding to λ, so is $-\mathbf{x}$. The choice of sign is arbitrary.

```
> y = - pc2$x # Principal components (reflected, still unrotated)
> M1 = cor(dat,y); M1 # Correlations between variables and components
              PC1        PC2
progmat 0.4709330  0.6299014
reason  0.4981509  0.7277446
verbal  0.5519561  0.6910097
headlng 0.7500678 -0.1156757
headbrd 0.6073970 -0.3689507
headcir 0.9063741 -0.1686041
bizyg   0.8298157 -0.2293757
weight  0.7274347 -0.2792455
height  0.7364050 -0.2490749
```

Applying a rotation to these correlations is very easy.

```
> vmax1 = varimax(M1); print(vmax1, cutoff=0)
$loadings

Loadings:
        PC1    PC2
progmat  0.122  0.777
reason   0.100  0.876
verbal   0.165  0.869
headlng  0.717  0.248
headbrd  0.709 -0.042
headcir  0.880  0.274
bizyg    0.841  0.185
weight   0.774  0.093
height   0.767  0.124

                 PC1   PC2
SS loadings    3.739 2.323
Proportion Var 0.415 0.258
Cumulative Var 0.415 0.674

$rotmat
           [,1]      [,2]
[1,]  0.8841526 0.4671982
[2,] -0.4671982 0.8841526
```

The pattern of correlations is clear. After rotation, the first component represents physical size, and the second component represents performance on the mental tests. The 67.4% of variance explained is the same as the percentage of variance explained before rotation:

```
> sum(M1^2)/9
[1] 0.6735697
```

As a quick cross-check, we calculate the scaled principal components $\mathbf{f}$, apply the rotation from `vmax1` to obtain $\mathbf{f}'$, and verify that $corr(\mathbf{z}, \mathbf{f}')$ corresponds to the "loadings" produced by the `varimax` function. In `fprime = f %*% t(R)`, note the post-multiplication by $\mathbf{R}^\top$, rather than pre-multiplication by $\mathbf{R}$. This is because the n random $\mathbf{f}$ vectors are in the rows of a matrix, and thus are transposed.

```
> f = scale(y)
> # Note that pc2$rotmat is the transpose of the rotation matrix that is applied to the factors
> R = t(vmax1$rotmat) # Transpose it for notation consistent with the text.
> fprime = f %*% t(R)
> round(cor(dat,fprime),3)
         [,1]   [,2]
progmat 0.122  0.777
reason  0.100  0.876
verbal  0.165  0.869
headlng 0.717  0.248
headbrd 0.709 -0.042
headcir 0.880  0.274
bizyg   0.841  0.185
weight  0.774  0.093
height  0.767  0.124
> print(vmax1$loadings,cutoff=0) # For comparison

Loadings:
        PC1    PC2
progmat  0.122  0.777
reason   0.100  0.876
verbal   0.165  0.869
headlng  0.717  0.248
headbrd  0.709 -0.042
headcir  0.880  0.274
bizyg    0.841  0.185
weight   0.774  0.093
height   0.767  0.124

                 PC1   PC2
SS loadings    3.739 2.323
Proportion Var 0.415 0.258
Cumulative Var 0.415 0.674
```

This works so well that I really can't see why anyone would want to do principal components *without* rotation. In fact, rotating principal components is a fairly common practice. Social scientists do it all the time. Many are led down this path by the default "factor analysis" method in SPSS and SAS being principal components (!) and the default rotation method being varimax.

It's interesting what these users do when they obtain a new data set with the same variables, or when they use a set of variables that have previously been "factor analyzed" by another author. Rather than using the weights (eigenvectors) from the first study, they tend to form "scales" by simply adding up the variables that correlate primarily with the same component, or possibly adding up z values if the variables are on really different scales (as the physical variables are in our example). Thus, they would get a "size" variable and a "smart" variable from the Mind-body data. The reasoning is usually not explicit, but I believe they may be thinking that the particular weights may be quite specific to the sub-population from which they obtained the data, and the weights may also be subject to sampling error. They want something more portable and generalizable, so they go with a cruder linear combination. In my view, this may be pretty good practice.

Generally speaking, the more sophisticated the user, the less likely he or she is to apply a rotation to principal components. After all, rotation is a central tool in exploratory factor analysis, and principal components analysis definitely is not factor analysis. So why do it? This little section provides the answer. I hope it establishes that scaling and then rotating a set principal components makes them easier to interpret, without sacrificing anything important.