Richard Gonzalez
Psych 614
Version 3.2 (March 2023)

LECTURE NOTES #11: Unfolding Analysis, Principal Components & Factor Analysis

Reading Assignment

1. Davison's chapter on unfolding

2. either T&F chapters on PCA & FA or J&W chapters on PCA & FA

3. review chapter on matrix algebra in either T&F or J&W

1. Unfolding Analysis[1]

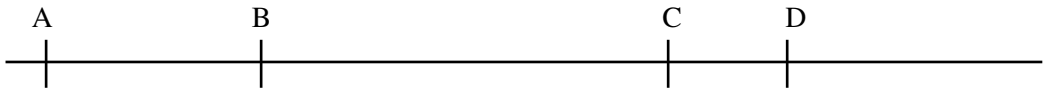
   This is a technique that allows MDS-type analyses on ranking or rating data, such as when a researcher observes rank orders. An example would be a search committee where each committee member ranks (or rates) each of the candidates. Similarity data are not needed for unfolding, so it is easier to implement than traditional MDS. The intuition underlying the idea is to find a space in which to place both stimuli and subjects in such a way that the observed rank order of the stimuli/items/candidates can be modeled. There is the usual dimensional space of stimuli as in MDS, but in addition subjects are plotted in the same space as the stimuli (unlike INDSCAL where subjects are plotted in a different "weight" space). One interpretation of this common plot is that subjects are "vectors" (arrows in the space) that fold the space to produce the rank orders or ratings that you observe.

   A motivating example is the number of teaspoons of sugar in a cup of tea. There is likely agreement on the same dimension of sweetness (the more sugar added, the sweeter the tea tastes) but people may differ in their preferences for how much sugar they want in their tea. Some people don't like any sugar, some like one teaspoon, some like two, etc. This model is different from a model that posits "more is better". Instead, unfolding posits "just the right amount is best." I like to refer to this as the Goldilocks model. The model attempts to find the points that represent the "ideal point" for each subject and the rank order is based on distances from the ideal point to each entity being ranked.

[1] Historical note: unfolding analysis was developed at UM in the 50's and 60's.

(a) Pedagogical example of the "String Model" with four stimuli that permits 7 possible ideal points on one dimension.

I call this a string model because imagine a string stretched out on a table top and along the string are taped four pieces of paper with the letters A, B, C, and D arranged as follows:



If we pick up the string at different points and let it dangle like a mobile, we will see different orders. In fact, there are a limited number of orders that are possible (out of the 24=4! total orderings) under this string representation. These orderings are:

A	B	C	D
B	A	C	D
B	C	A	D
B	C	D	A
C	B	D	A
C	D	B	A
D	C	B	A

The place where the string is picked up is called the "ideal" point, and the interpretation is that rank orders are determined with respect to the ideal points. The string model permits 7 of the 24 possible orderings. Some orderings are impossible, such as DABC because it is impossible to pick up the string in such a way where D is most preferred (closest to the ideal point) and A is second most preferred. The "string model" is illustrated below:

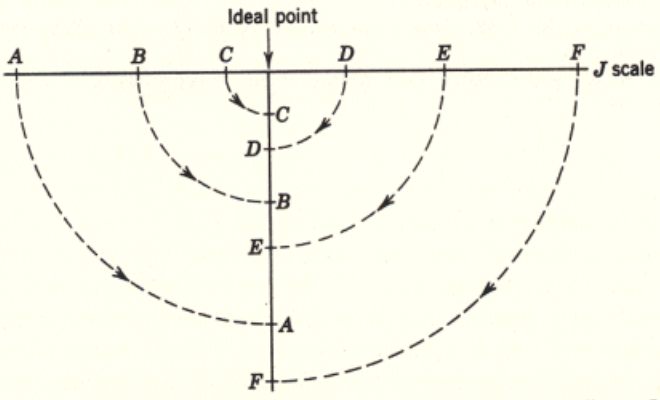


Fig. 5.1 An *I* scale preference ordering *C D B E A F*, obtained by folding a *J* scale.

Let's assume the idealized case with seven subjects and each subject provides one of the rank orders above. The "data matrix" is traditional because subjects go along the rows and each column refers to data on a particular variable (in this case a stimulus); contrast this to the symmetric distance matrices we used in the MDS/tree structure section of the course.

SPSS ALSCAL can run an unfolding analysis with this syntax. The shape of the matrix is rectangular, and we need to specify "/cond = row" to let SPSS know that data across rows are not comparable (e.g., a 2 from one subject may not mean the same thing as a 2 from another subject). It is possible to input "/levels = ordinal" but with such few data points the program has trouble so I'll use "/levels=interval" in this small example. Note that the data we input has subjects along the rows. Each subject ranked the four items on the basis of most favorite, e.g., subject 2 ranked item a as 2nd, item b as 1st, etc.

```
data list free / a b c d.
begin data
1 2 3 4
2 1 3 4
3 1 2 4
4 1 2 3
4 2 1 3
4 3 1 2
4 3 2 1
end data.

 alscal variables a b c d
 /levels = interval
 /shape = rect
 /criteria=dimens(1)
 /cond = row
 /plot = all.
```

The solution is listed below. The stimuli and the subjects (listed as "row" in the output)

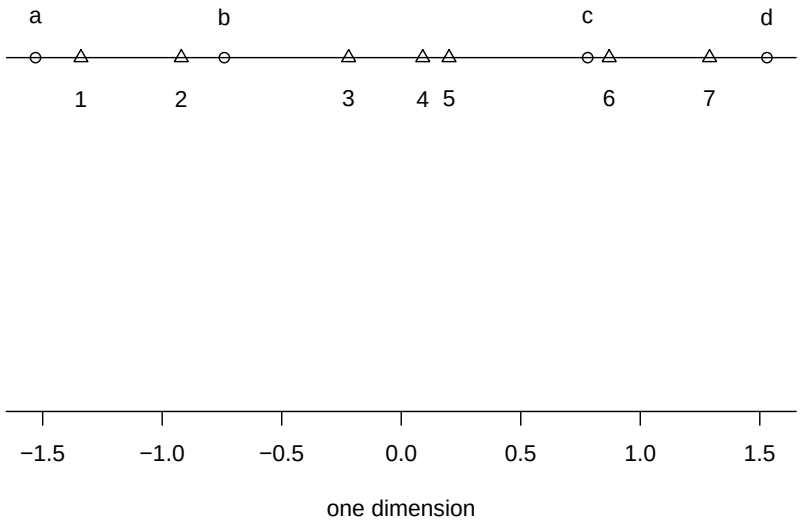
are both placed along the same dimension. This is called a biplot. The program also prints out the usual stress and r^2 values. The SPSS plot is a little difficult to read so I constructed my own plot from the coordinate space. A graph of this one dimensional solution is presented in Figure 11-1. The output separates stimuli from subjects, but both are estimated as being along the same dimension. For example, subject 1 has a scaled value (an ideal point) of -1.3396, which is between the scaled values for stimuli A and B.

```
Column
   1      A         -1.5347
   2      B          -.7370
   3      C           .7770
   4      D          1.5311
 Row
   1                -1.3396
   2                 -.9218
   3                 -.2202
   4                  .0935
   5                  .1953
   6                  .8678
   7                 1.2887
```

In R we can use the unfolding function in the smacof package. Here I illustrate with the simple 1D example; the solution is similar to the one in SPSS even though the numerical algorithm is slightly different.

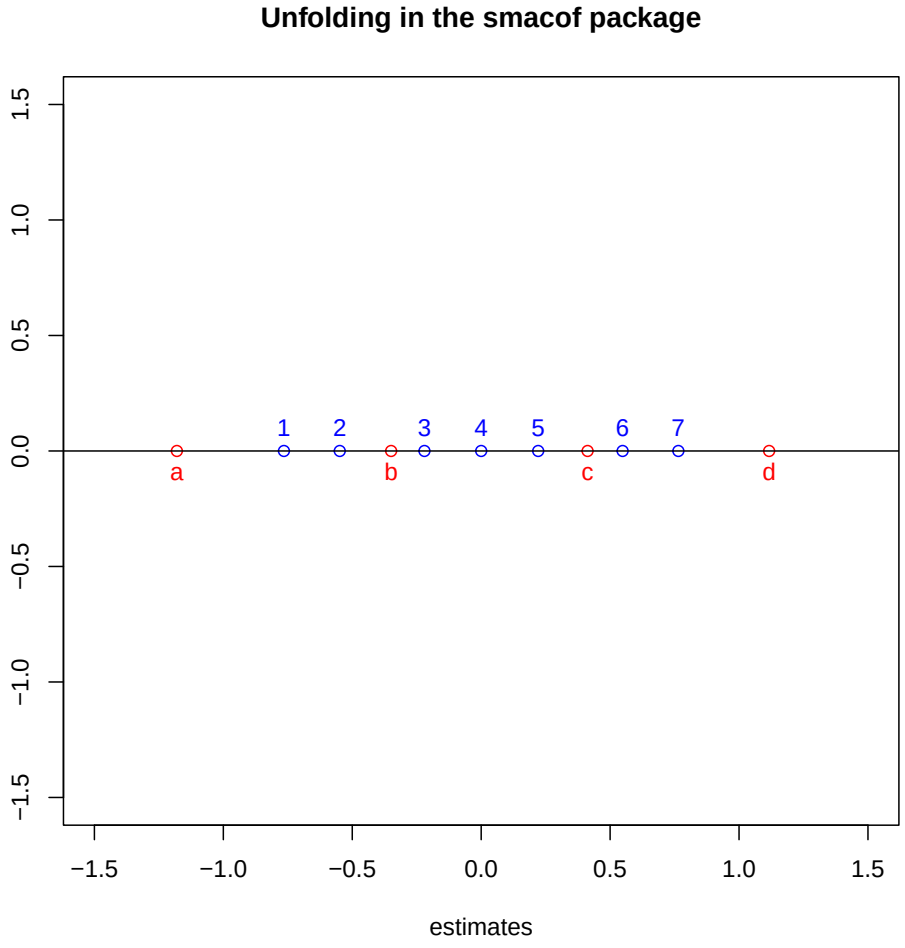
```
#read in data
data <- rbind(c(1,2,3,4),c(2,1,3,4),
              c(3,1,2,4),
              c(4,1,2,3),c(4,2,1,3),
              c(4,3,1,2),
              c(4,3,2,1))
data


##       [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    2    1    3    4
## [3,]    3    1    2    4
## [4,]    4    1    2    3
## [5,]    4    2    1    3
## [6,]    4    3    1    2
## [7,]    4    3    2    1


library(smacof)
out.unfold <- unfolding(data,ndim=1,
```

Figure 11-1: Simple example with 4 stimuli (circles) & 7 subjects (triangles represent ideal points). If you "pick up the string" at a subject's ideal point, you will reproduce the rank order given by that subject.



```
                          type="ordinal", cond="row")
summary(out.unfold)


##
## Subject configuration (rows):
##           D1
## [1,] -0.7650
## [2,] -0.5485
## [3,] -0.2201
## [4,]  0.0000
## [5,]  0.2211
## [6,]  0.5485
## [7,]  0.7647
##
## Object configuration (columns):
##           D1
## [1,] -1.1801
## [2,] -0.3492
## [3,]  0.4127
## [4,]  1.1167
##
##
## Stress per point rows:
##          SPP  SPP(%)
## [1,]  0.0000  0.0000
## [2,]  0.0000  0.0000
## [3,]  0.0000  0.0000
## [4,]  1.0060  1.0060
## [5,] 13.6821 13.6821
## [6,] 16.9014 16.9014
## [7,] 68.4105 68.4105
##
## Stress per point columns:
##          SPP  SPP(%)
## [1,]  0.0000  0.0000
## [2,]  0.0000  0.0000
## [3,]  0.0000  0.0000
## [4,]  1.0060  1.0060
## [5,] 13.6821 13.6821
## [6,] 16.9014 16.9014
## [7,] 68.4105 68.4105


#plot solution
```

```
dim <- range(pretty(range(rbind(out.unfold$conf.row,
                                out.unfold$conf.col))))
estimates <- c(out.unfold$conf.row, out.unfold$conf.col)
plot(estimates,rep(0,11),xlim=dim,ylim=dim,
     col=c(rep("blue",7),rep("red",4)),
  main="Unfolding in the smacof package",ylab="")
subs <- as.character(1:7)
stimuli <- letters[1:4]
abline(h=0)
text(estimates[1:7],rep(0,7),subs,pos=3,
     col="blue")
text(estimates[8:11],rep(0,4),stimuli,pos=1,
     col="red")
```



(b) Applications of One Dimensional Unfolding

Some interesting applications of this technique have been used in political science such as the analysis of Supreme Court decisions. For each case the file shows whether a judge sided on the liberal side or the conservative side (1 or 0, respectively, so the data are binary). I present two examples for all non-unanimous court cases in 2000. Figure 11-2 is a simple one dimensional solution broken up into different types of cases; you can see, for example, justices Kennedy and Thomas are swing judges appearing sometimes on the liberal side but mostly on the conservative side. Special programs need to be run on binary data such as this example (ALSCAL in SPSS can handle binary data but unfolding in the smacof package in R cannot).

Here is what the data file looks like

```
library(MCMCpack)
data(SupremeCourt)
head(SupremeCourt)
```

```
##   Rehnquist Stevens O'Connor Scalia Kennedy Souter Thomas
## 1         0       1        1      0       1      1      0
## 2         0       1        0      0       0      1      0
## 3         0       1        0      0       0      1      0
## 4         0       0        0      0       0      0      0
## 5         1       1        0      0       1      0      0
## 6         0       1        0      0       0      0      0
##   Ginsburg Breyer
## 1        1      1
## 2        1      1
## 3        1      1
## 4        0      1
## 5        0      0
## 6        0      0
```

Figure 11-3 is a more modern type of unfolding analysis that uses a Bayesian framework to put distributions on the ideal points (so ideal distributions instead of points). Sometimes the distributions overlap so we would expect those judges to be more likely to switch relative positions than judges where the distributions do not overlap. This example was analyzed using the MCMCpack package in R, which provides Bayesian approaches to several common analyses.

(c) Multidimensional Unfolding

Figure 11-2: 2000 Supreme Court Cases (Islam et al).

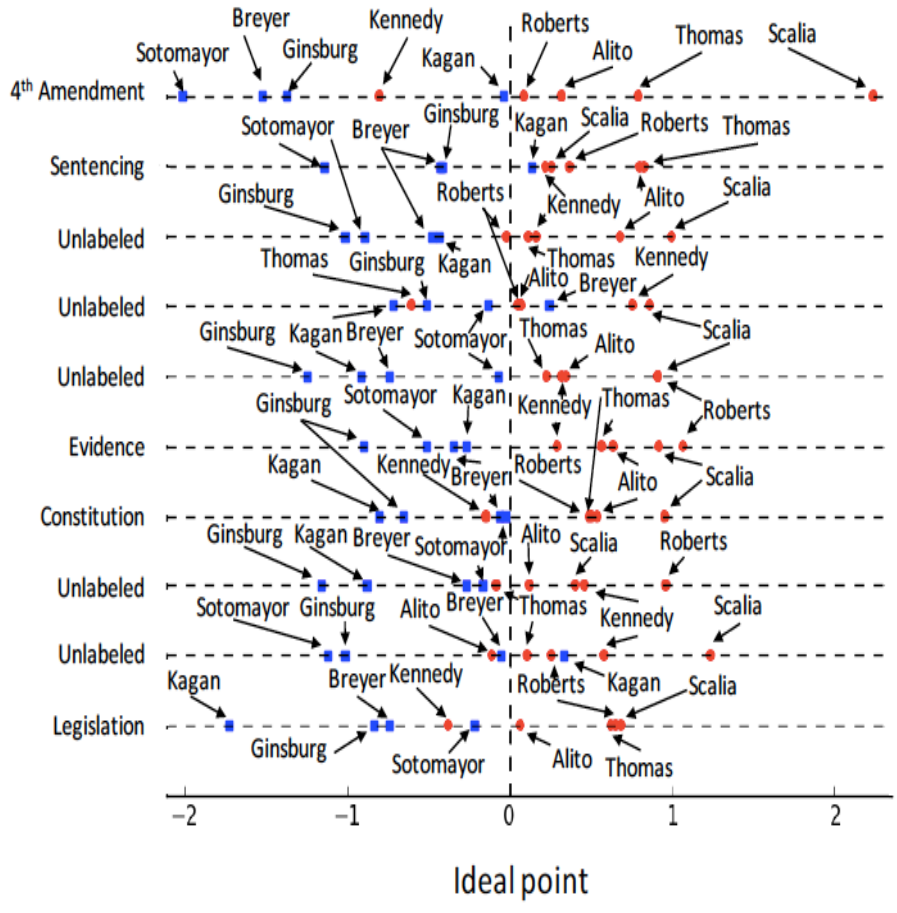
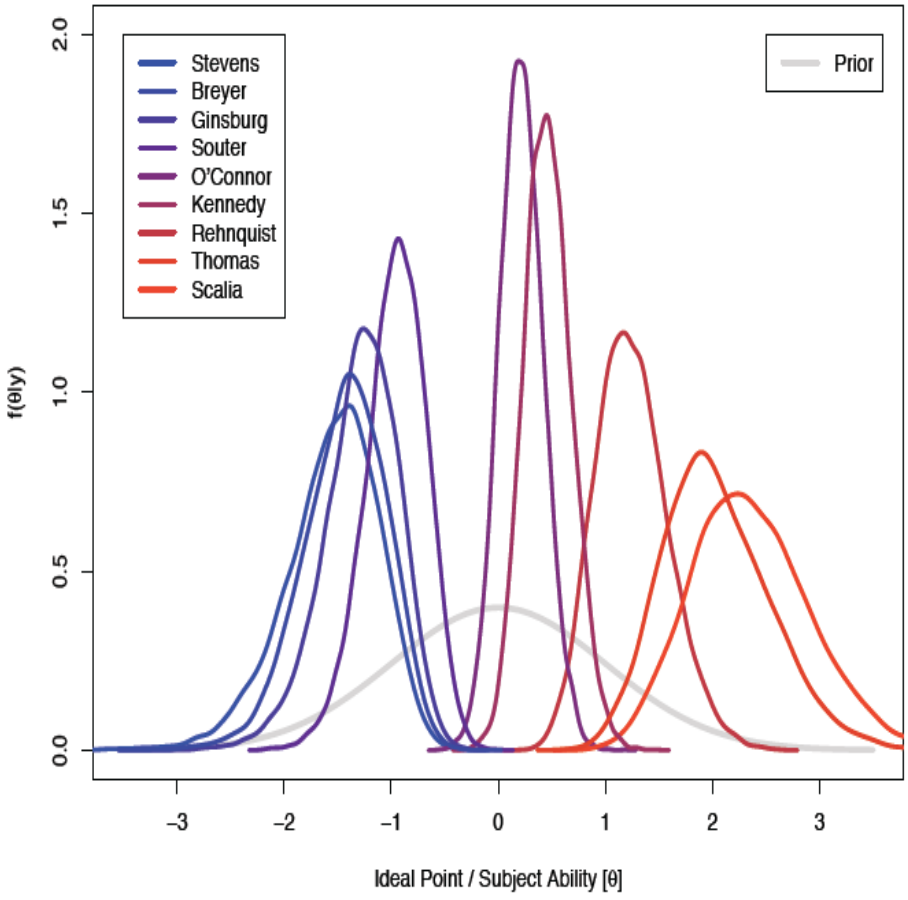


Figure 11-3: 2000 Supreme Court Cases, Bayesian Analysis with Ideal Distributions (MCMCpack tutorial).



This simple "string model" can be extended to more complicated, multidimensional situations. Intuitively, there are regions in the stimulus space that correspond to the same rank order (much like in the one dimensional case where intervals on the string represent the same observed rank order). This is analogous to INDSCAL where there is both a stimulus configuration space and a subject space, but in unfolding the two are presented together in one plot. Sometimes it helps to draw arrows from the origin to subject ideal points to help distinguish them from the points representing the items[2]

Here is a small example that you can try. Six subjects rank order their favorite colors (6 different colors are presented). Note that here subjects are not rating similarity, but are ranking the objects according to preference. 1 is most preferred; 4 is least preferred. The solution gives both a spatial configuration and also information on each subject's preferences; both the spatial configuration and the ideal points are plotted on the same plot. The final configuration is shown in Figure 11-4.

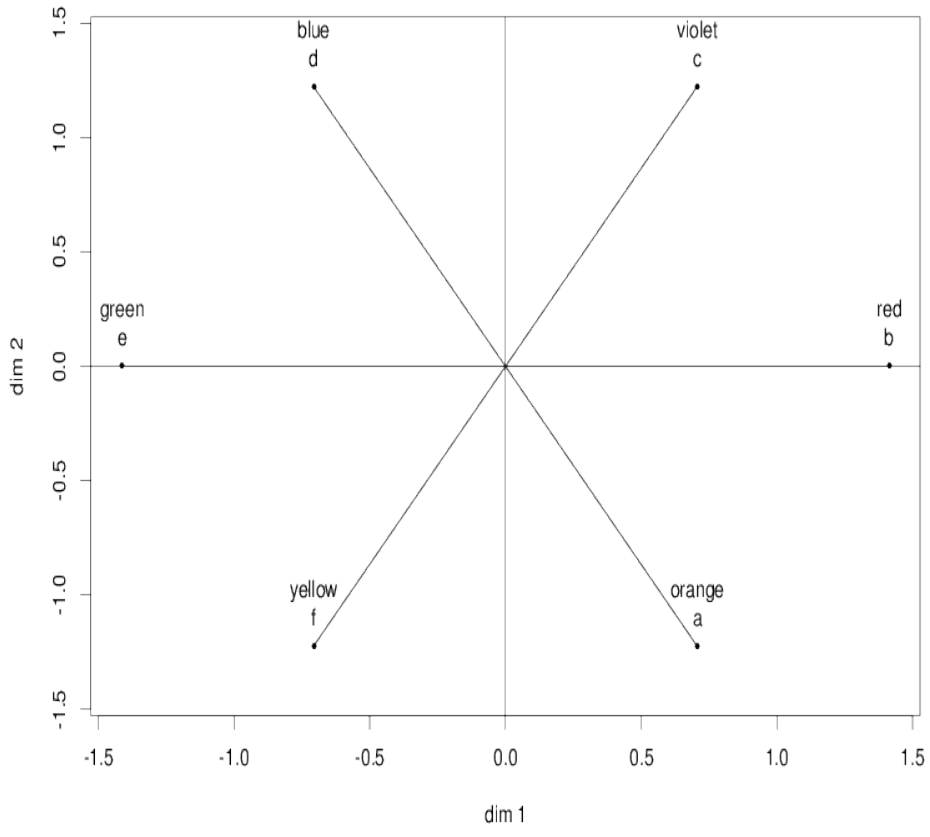
Subject	orange	red	violet	blue	green	yellow
A	1	2	3	4	3	2
B	2	1	2	3	4	3
C	3	2	1	2	3	4
D	4	3	2	1	2	3
E	3	4	3	2	1	2
F	2	3	4	3	2	1

```
data list free / a b c d e f.

begin data
1 2 3 4 3 2
2 1 2 3 4 3
3 2 1 2 3 4
4 3 2 1 2 3
3 4 3 2 1 2
2 3 4 3 2 1
end data.

alscal variables a b c d e f
  /shape = rectangular
  /level=ordinal
  /condition=row
  /criteria=dimens(2)
  /plot all.
```

Another example of unfolding is one where 42 participants ordered 15 breakfast items according to their preference. The 15 items include toast, jelly doughnut, blueberry

[2]Some versions of unfolding represent "ideal points" as vectors and the original rank order of the stimuli are reproduced from the projections onto the subject ideal vectors. In this type of unfolding, vectors that have similar angles correspond to items that have similar rank orders. There are other versions where rank orders are modeled as the distance from each stimuli to the subject's ideal point.

Figure 11-4: Color example example using ALSCAL with levels=ordinal.



muffin, cinnamon bun, etc.

See Figure 11-5 for the two dimensional solution (given as an example). The majority of subjects have ideal points roughly at the point (0,-5), the large blob of points near danish pastry (no surprise). It isn't clear how to interpret the two dimensional solution that emerged in this example, but the horizontal may be a hard vs soft dimension. There are some natural clusters of breakfast items that emerged like jelly doughnut and doughnut as well as coffee cake and cinnamon bun.

(d) The Model

The unfolding models works with a rectangular data matrix as input. It estimates a dimensional coordinate representation to place stimuli (e.g., the objects ranked) and ideal points (i.e., information about the units producing the rankings) together in the same representation. This allows the computation of distance between an object's representation and the ideal point. Unfolding analysis places both in the same plot because the individual data are modeled as ideal points embedded in the "stimulus configuration matrix." Similar to INDSCAL, cases could be drawn in terms of vectors from the origin (I'll provide motivation for this suggestion below in the "more is better" representation), however in INDSCAL there are two separate spaces, one for stimuli and one for subjects rather than placing both in the same representation.

Distance in this model is defined as

$$d_{is} = \sqrt{\sum_{k}(x_{ik} - x_{sk})^2} \tag{11-1}$$

where i denotes a stimulus, s denotes the ideal point, and k is a counter for dimensions. Distance is computed with respect to an item and an ideal point. So, all stimuli are modeled with respect to their distance from an "unknown" ideal point s, the technique estimates these unknown ideal points using the preference/rank/rating data that is supplied.

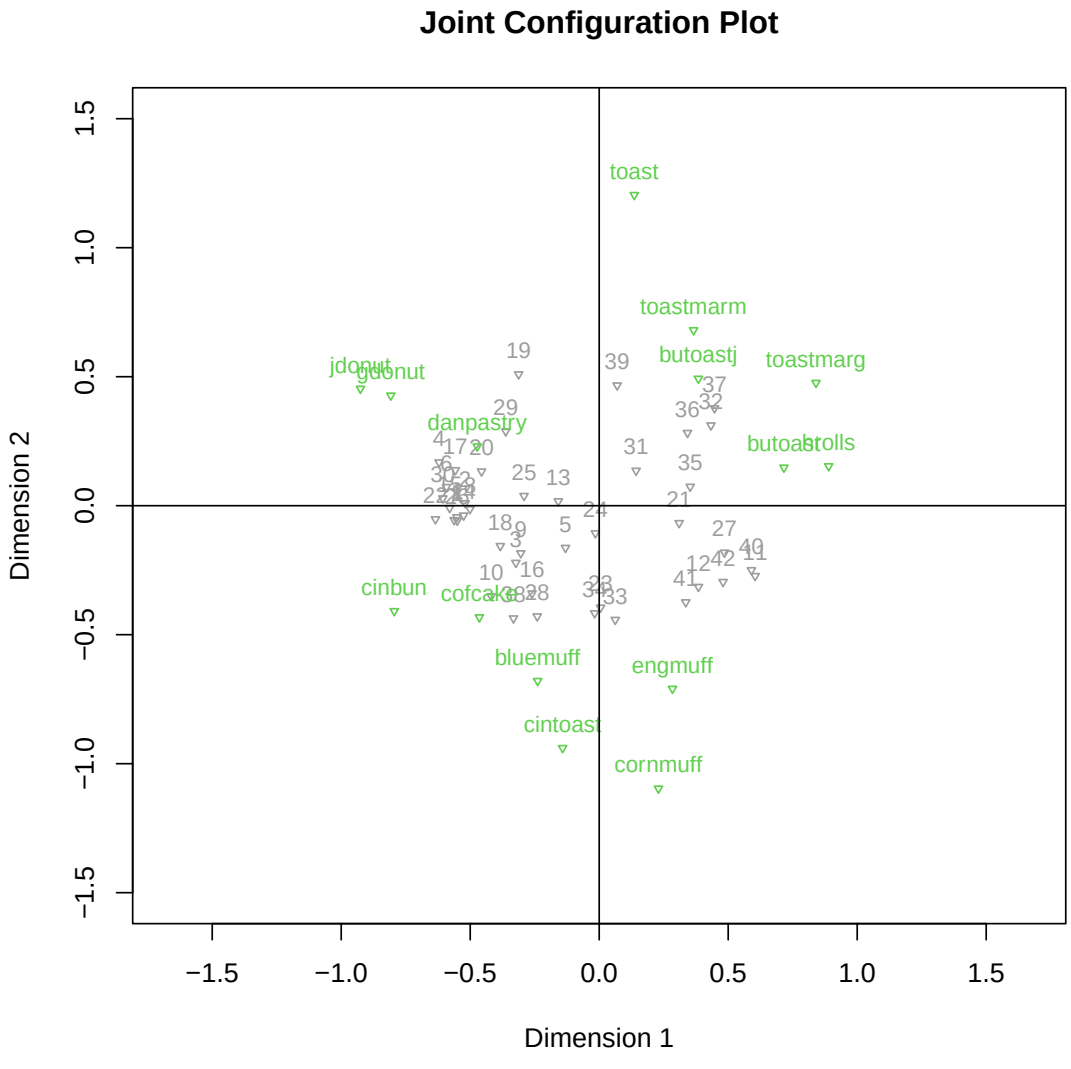
There is a metric called stress that minimizes the discrepancy between the observed data (the ranks) and the model-implied distances in the sense that items that are close to the ideal point should be ranked high and items not close to the ideal point should be ranked low.

The applications of this model are broad. Any data set that involves rank orders could be analyzed with in this framework.

(e) Extensions of Unfolding Analysis

Figure 11-5: Two dimensional unfolding example with breakfast data set
Some R code:

```
library(smacof)
res <- unfolding(breakfast)
dim <- range(pretty(range(rbind(res$conf.row,res$conf.col))))
plot(res, type = "p", pch = 25, col.columns = 3, ylim=dim,
     xlim=dim,
label.conf.columns = list(label = TRUE, pos = 3, col = 3),
col.rows = 8, label.conf.rows =
  list(label = TRUE, pos = 3, col = 8))
abline(h=0,v=0)
```



There is also the possibility of merging an MDS (or INDSCAL) with an unfolding analysis. The kind of unfolding analysis I have presented so far finds both a spatial representation and a preference structure for the same data. This is called "internal unfolding". There is a sense in which that is asking "too much" from a data set. The type of data used in unfolding doesn't need to include all pairwise distances; instead the technique models the preference order in terms of distance from a single point. So, before we believe a particular solution too much it should be replicated or independently validated.

There is a more sophisticated merger of MDS with unfolding. One can collect both similarity data and preference data. The program takes both pieces of information and tries to merge them (ALSCAL can do this). This is called "external unfolding" because the spatial representation is found from different data than the preference structure, and both similarity and preference are merged into a single representation. See Carroll (1972) and also Davison's book for more details.

Preference modeling in the context of an unfolding analysis can go in two distinct ways. So far I've talked about the ideal point model from the perspective of a model that assumes a decreasing preference function away from the ideal point in any direction—the Goldilocks idea where your favorite is closest to the ideal point, etc. This representation contrasts with other ways to model preference that use a "more is better" framework. This notion has an increasing preference function. So, for example, more dollars are always better than less (there probably isn't an ideal amount of money for you so that if I offered you higher salary in excess of your ideal point you would reject the salary offer—interesting thought experiment, what would Goldilocks do in a salary negotiation?.

The ideal point version of the model treats ideal points as points on the plot and distances from the ideal are computed, much like the unidimenstinal string example presented at the beginning of this section. The further away from the ideal point, the less desirable the stimuli. The "more is better" model, however, treats "ideal points" as vectors from the origin and computes projections of each stimuli to the ideal vector—in this vector-based model one can imagine creating new stimuli that go further and further out along the "ideal vector" and those new stimuli would be more preferred. While this vector-based representation has been studied in theoretical papers there are not many publically available programs to run this (check out the vmu function in the smacof package, the GGUM package in R and connections of unfolding approaches to Item Response Theory—IRT).

(f) Deep thoughts ...

There are many differences between the kind of modeling we did in ANOVA and regression and the kind of modeling we see in MDS/ADDTREE/UNFOLDING sections.

Aside from the data structure being different, the availability of statistical tests, and all the other obvious differences, I see a fundamental distinction in what is being modeled. In ANOVA/Regression we are applying a general purpose tool to model data. But are we really modeling underlying mechanism with ANOVA/regression? Would we say that the person's behavior follows $\mu + \alpha + \epsilon$? Would we say that a psychological construct is determined by $\beta_0 + \beta_1 X + \epsilon$? Probably not as this is not a very convincing model of the underlying process or mechanism. Maybe it is fine for modeling a data process with noise (i.e., the "plus ϵ" part) or for simple analytic goals such as are there differences between group means or checking an association between two variables while holding constant a third (such as a part correlation in the context of regression analysis).

The type of modeling we see in MDS/ADDTREE/UNFOLDING is different. We are taking a stab at an underlying psychological process, trying to represent that process mathematically, and then testing the fit of data to the mathematical model. Different psychological processes (dimensional versus featural comparisons, increasing preference versus preference functions peaked at the ideal point), may require different mathematical representations. A benefit of this type of modeling is that when we reject a model we reject a psychological theory or mechanism. For example, if we reject the triangle inequality we know that the underlying process or mechanism doesn't follow a distance representation, whereas when an R^2 in a regression is low we can't really conclude much other than saying additivity of predictors and the associated statistical assumptions of independence, equal variance and normality may be off. Later in these lecture notes I will show a more formal connection between regression and analyses on distance matrices. If you would like to learn more about this type of modeling (a modeling that is more directly tied to psychological theory and underlying mechanisms), check out the area of mathematical psychology [3].

2. Principal Components Analysis (PCA)[4]

I will present PCA as a special case of MDS, and then I will "re-present" PCA in the way it is usually taught so that you will be able to communicate with others who learned it in a different way. People are not accustomed to thinking of PCA as a special case of MDS, but thinking in terms of MDS helps make this complicated procedure simple and easy to understand. You will see this theme of recasting multivariate statistical problems in terms of MDS throughout the rest of the term. That is why we spent relatively more time on MDS so that these techniques become easier to understand.

[3]The UM Psychology department sometimes offer a graduate-level course on mathematical psychology.

[4]For the neural network/connectionist modelers in the class: PCA is equivalent to a neural network with one hidden layer such that there are as many nodes in the hidden layer as factors, every input has a connection to every node in the hidden layer, and the output layer is linear (rather than more typical output layers such as softmax). Neural networks can be extended to include nonlinear outputs, multiple layers (deep learning), etc., so are more general than PCA and provide new extensions of traditional PCA models. See Qiu et al (2012, Neural network implementations for PCA and its extensions, *ISRN Artificial Intelligence*) for more details.

One goal of PCA is to reduce many variables (actually, to reduce the correlation matrix of many variables) down to a more manageable set of dimensions. This is analogous to the goal of MDS where many distances are reduced to a relatively small dimensional space.

(a) PCA as a simple extension of metric MDS (i.e,. when "/levels = interval" is specified in ALSCAL or type = "interval" in smacof).

You compute a correlation matrix (or a covariance matrix, which won't necessarily lead to the same solution) between the variables, and treat that matrix as the "similarity" matrix that you analyze with MDS using the "interval" subcommand. That's all there is to PCA—it is identical to MDS, we interpret dimensions in the same way (though in PCA they are called "components"). If you use the "ordinal" option in MDS you have a nonmetric form of PCA because the key information used is the ordering of the similarity matrix entries.

The data analyst manually creates the proximity matrix, in this case the correlation matrix, from the variables as opposed to the typical MDS data set where the proximity matrix is observed directly. All other interpretations of MDS (problems and tricks such as rotating, using external variables to validate the interpretation of dimensions, etc.) extend to PCA. The key difference between the data analyst manually creating the similarity matrix as opposed to the subject generating the similarity matrix is that in the former the dimensions are imposed by the analyst. Recall that one nice feature of MDS is that it permits one to "recover" the dimensions subjects are using. The approach of asking subjects to rate items on a set of variables and then manually creating a proximity matrix from those responses has the limitation that the dimensions are imposed on the subjects. This is not an argument against PCA (which is identical to metric MDS applied to a correlation matrix) but an argument against computing a proximity matrix from other variables rather than observing a proximity matrix directly.

As I show in the linear algebra appendix, a correlation r can be converted into a distance by the transformation $\sqrt{2(1-r)}$. Some people ignore the constant and the square root; they treat the correlation r directly as a measure of similarity.

That was relatively painless. We just learned a new statistical procedure, which fell out naturally from what we already know about MDS. Next, I will introduce the same procedure, PCA, the way it is usually taught.

(b) The usual pedagogical introduction to PCA

The typical intuition surrounding PCA is to create new variables as linear combinations of existing variables. You could do this in an ad hoc manner by summing up variables

you think are related (giving each variable a weight of 1, just like the unit contrast, and computing the weighted sum) and treating the sum as a new variable. However, there are better ways. For example, rather than weighting each variable by one you could try to find optimal weights. How do you find these optimal weights?

First, let's consider the case of linear regression, I know that was a long time ago in this class. Suppose you had three predictors of some dependent variable. If you didn't know about regression you might just take the sum of the three predictors (somehow deal with scaling if the predictors are on different scales) and check how closely the sum maps onto the dependent variable. But, with knowledge of regression you can make better predictions under a particular definition (i.e., minimizing squared error). That is, regression finds βs to weight each variable in some optimal way. You can think of the βs as a set of contrast weights that weight variables rather than cell means. PCA uses this weighting idea to find weights to create new dependent variables. As we will see, PCA finds the optimal weights without using a dependent variable (so in other words, it runs a regression-like analysis to find βs but doesn't have a dependent variable). PCA does magic in computing a regression with no Y.

The underlying logic of PCA is that a set of variables X_1, X_2, ..., X_p (where p is the number of variables) can be reduced into a smaller set of variables Y_1, Y_2, ..., Y_m (where m is the desired number of dimensions). The problem is that the Y's are not observed so we can't do regression in the usual direct manner. But, hey, the lack of a dependent variable doesn't have to stop us from moving forward. We seek a set of weights $A_1 = (a_{11}, a_{12}, \ldots, a_{1p})$ that when applied to all X's yield Y_1. That is,

$$Y_1 = a_{11}X_1 + a_{12}X_2 + \ldots + a_{1p}X_p \quad (11\text{-}2)$$

Similarly, we seek another set of weights A_2 that when applied to the same X's yield Y_2, as in

$$Y_2 = a_{21}X_1 + a_{22}X_2 + \ldots + a_{2p}X_p \quad (11\text{-}3)$$

We can continue this process to fit Y_3 with weights A_3, Y_4 with weights A_4, etc. The As are defined such that they produce Ys. that are orthogonal to each other.

When someone says they ran a PCA to reduce the dimensionality of their data that means they reduced the observed X's (the variables) to a smaller set of Y's, the composite scores. Once the new Y's are in hand, then subsequent analyses (such as ANOVA and regression) can use the fewer number of Y's rather than the more numerous X's. One use of this I mentioned in the context of multicolinearity in regression: if there are several predictors that are highly intercorrelated, one solution to the multicolinearity problem is to conduct a PCA on those predictors to represent them as a smaller set of new variables (and the PCA technique has ways of making those new variables have little or no correlation, if that is a desired goal).

We also want A_1 and A_2 to be orthogonal (actually, we want all pairs of weights A to be orthogonal). You can think of A_1 and A_2 as contrasts defined over the predictors X but instead of the data analyst choosing the weights a computer algorithm estimate the weights.

Below I discuss how we find these sets of weights $A_1, A_2, \ldots, A_p$. Each of the A vectors are defined so $\Sigma_j a_j^2 = 1$ such that the variance of the Y_i is maximized.

One way to depict these unobserved PCs is to compare with linear regression. Take a state-level data set (so 50 points) of Murder rate and Rape rate per 100,000 residents. The left panel of Figure 11-6 shows the usual regression with Murder as a predictor and Rape as a dependent variable. The vertical residuals are also shown; think back to the rubber band and stick example from the introduction of regression in Lecture Notes 6. The right panel of Figure 11-6 shows the plot of the first PC with projections (right angles) from the point to the PC. Both panels of Figure 11-6 show the same data points, but the left shows the regression with vertical residuals and the right shows the principal component with residuals defined as projections (90 degrees to component). In the right panel, neither variable plays the role of predictor or dependent variable, both are projected on the same line, which represents the first PC, and both variables are in some sense put on equal footing. The left panel, however, gives priority to the variable presented on the vertical axis by defining residuals vertically as we discussed in the regression portion of the course.

(c) A tangent: linear algebra. See Appendix 4.

(d) Eigenvalues and Eigenvectors

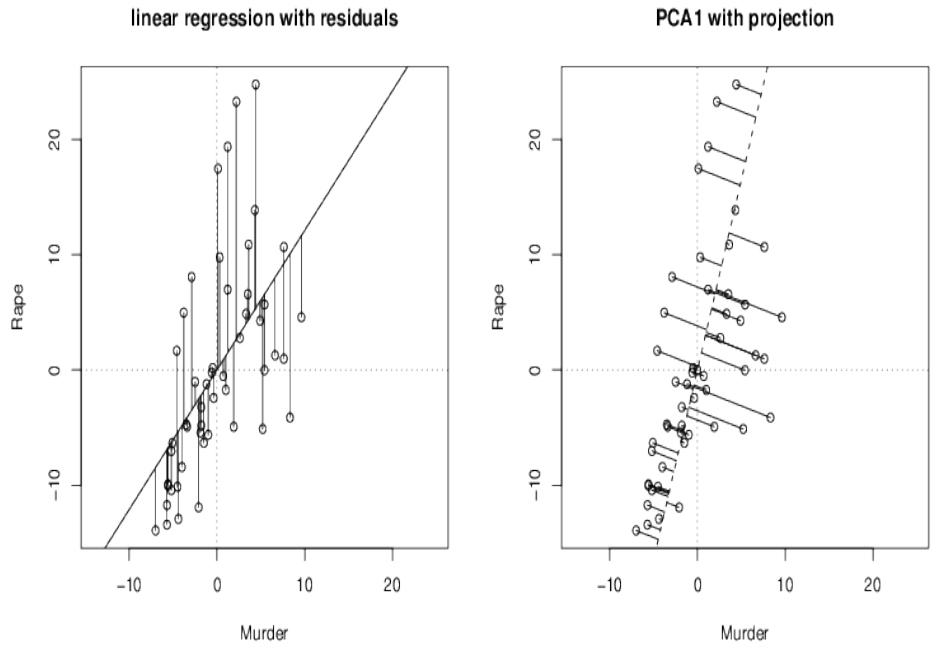
We now turn attention to a key concept in PCA: eigenvalues and eigenvectors. We consider two equivalent definitions:

i. $Mx = \lambda x$

The intuition of this definition is that the effect of matrix M on x is identical to the effect of the single number λ on x. Hence, the single number λ contains all the relevant information about the matrix M (relevant in the sense of its effect on vector x). Recall that matrix multiplication of a vector can be thought of geometrically—the matrix M "moves" vector x elsewhere in the space (a matrix has the effect of shrinking, expanding, translating, rotating, or any combination of these, a vector).

The λ's are called eigenvalues; the x's are called eigenvectors. Note that the λ's and the x's come in pairs in the sense that for every λ there is an x that fits the equation

Figure 11-6: Show regression versus PCA.



above.

ii. M = VLV' (spectral decomposition)

The intuition here is that the matrix M can be decomposed into "elementary" parts: V is the matrix of eigenvectors and L is the diagonal matrix of eigenvalues.

Here is an example correlation matrix. I will find eigenvalues and eigenvectors, and then compare that to the SPSS output.

```
1.00 0.55 0.43 0.32 0.28 0.36
0.55 1.00 0.50 0.25 0.31 0.32
0.43 0.50 1.00 0.39 0.25 0.33
0.32 0.25 0.39 1.00 0.43 0.49
0.28 0.31 0.25 0.43 1.00 0.44
0.36 0.32 0.33 0.49 0.44 1.00
```

The eigenvalues are:

```
2.89 1.02 0.65 0.56 0.49 0.39
```

The eigenvectors are in each column, in the same order as the eigenvalues.

```
0.42 -0.39 -0.23  0.46  0.50 -0.39
0.42 -0.48 -0.29 -0.14 -0.19  0.67
0.41 -0.32  0.56 -0.42 -0.26 -0.41
0.41  0.42  0.48  0.01  0.51  0.40
0.38  0.45 -0.56 -0.52  0.07 -0.25
0.42  0.37  0.00  0.56 -0.61 -0.05
```

The ambitious reader may want to check this computation and verify (by applying the spectral decomposition formula) that the eigenvalues and eigenvectors can reproduce the original correlation matrix within round-off error.

It is useful to compare this simple matrix with SPSS output. Here is the syntax I used. Notice that I did a trick of entering the correlation matrix directly (I'm not using the begin data/end data syntax to enter raw data but instead the correlation matrix); the FACTOR procedure can accept either raw data or matrices as input. The number 1

must be in the diagonal because that is the correlation of a variable with itself (from MDS you know we automatically have a similarity matrix if the maximum value is in the diagonal and the distance/similarity axioms hold). I'm asking for all six factors (there can be as many factors as there are variables but not more and usually the goal is to reduce the complexity so we select the number of factors to be much smaller than the number of variables) so we can see where the eigenvalues and eigenvectors appear in computer output such as SPSS.

```
matrix data variables = a b c d e f
 /format full
 /contents = corr
 /n = 100.

begin data
 1 0.55 0.43 0.32 0.28 0.36
 0.55 1 0.50 0.25 0.31 0.32
 0.43 0.50 1 0.39 0.25 0.33
 0.32 0.25 0.39 1 0.43 0.49
 0.28 0.31 0.25 0.43 1 0.44
 0.36 0.32 0.33 0.49 0.44 1
end data.

execute.

factor
 /matrix in(cor=*)
 /print fscore initial extraction rotation correlation repr
 /plot eigen rotation
 /criteria = factors(6).
```

The SPSS output follows:

```
Variable     Communality  *  Factor   Eigenvalue   Pct of Var   Cum Pct
                          *
A               1.00000   *     1       2.88634       48.1         48.1
B               1.00000   *     2       1.01781       17.0         65.1
C               1.00000   *     3        .65258       10.9         75.9
D               1.00000   *     4        .56233        9.4         85.3
E               1.00000   *     5        .48946        8.2         93.5
F               1.00000   *     6        .39149        6.5        100.0

Factor Matrix:

              Factor  1     Factor  2     Factor  3     Factor  4     Factor  5     Factor  6

A               .71339       -.38879        .18611       -.34836        .35252        .24425
```

B	.70999	-.48354	.23339	.10714	-.13494	-.42184
C	.70147	-.32391	-.45115	.31663	-.18338	.25611
D	.68914	.42512	-.39147	-.00700	.35676	-.25256
E	.63755	.45445	.45461	.39312	.04742	.15345
F	.70703	.37506	-.00293	-.41791	-.42875	.03087

Reproduced Correlation Matrix:

	A	B	C	D	E	F
A	1.00000*	.00000	.00000	.00000	.00000	.00000
B	.55000	1.00000*	.00000	.00000	.00000	.00000
C	.43000	.50000	1.00000*	.00000	.00000	.00000
D	.32000	.25000	.39000	1.00000*	.00000	.00000
E	.28000	.31000	.25000	.43000	1.00000*	.00000
F	.36000	.32000	.33000	.49000	.44000	1.00000*

The lower left triangle contains the reproduced correlation matrix; the diagonal, reproduced communalities; and the upper right triangle residuals between the observed correlations and the reproduced correlations.

There are 0 (.0%) residuals (above diagonal) with absolute values > 0.05.

VARIMAX converged in 6 iterations.

Rotated Factor Matrix:

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6
A	.10234	.17772	.92723	.12046	.14182	.25211
B	.12864	.22977	.25778	.07124	.11691	.91949
C	.08293	.93316	.17529	.16691	.12123	.22150
D	.19406	.16692	.11834	.93179	.21757	.07000
E	.94916	.07972	.09691	.18451	.18737	.11914
F	.19833	.12226	.14098	.21900	.92987	.11500

Factor Transformation Matrix:

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6
Factor 1	.38166	.41464	.41938	.40474	.41439	.41352
Factor 2	.45134	-.32365	-.38779	.41930	.36890	-.47881
Factor 3	.56178	-.55740	.23135	-.48696	-.00570	.28812
Factor 4	.52155	.42192	-.46510	-.01244	-.56010	.14069
Factor 5	.06685	-.26266	.50322	.50980	-.61330	-.19307
Factor 6	.24197	.40696	.38816	-.40397	.04822	-.67798

Hi-Res Chart # 2:Factor plot of factors 1, 2, 3

Factor Score Coefficient Matrix:

	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5	Factor 6
A	-.04183	-.12951	1.21050	-.07618	-.10477	-.28193
B	-.09941	-.22931	-.27548	.01857	-.06496	1.24282
C	-.01234	1.19134	-.13109	-.16069	-.06681	-.23828

D	-.17123	-.16105	-.07847	1.19625	-.21472	.02082
E	1.14311	-.01221	-.04359	-.18026	-.17981	-.10782
F	-.16979	-.06660	-.10576	-.21333	1.19475	-.06533

Let me walk you through this output. The first table lists the eigenvalues, and you can see that the sum of the eigenvalues equals the number of variables, in this case 6. This is a general result. The sum of the eigenvalues will be equal to the sum of the diagonal of the original matrix. Since a correlation matrix has ones in the diagonal, the sum will equal the number of variables. We call the sum of the diagonal of a matrix the trace; the trace is also equal to the sum of the eigenvalues. We'll talk about the communalities later.

The factor matrix table has the eigenvectors in each column multiplied by the square root of its associated eigenvalue. For instance, the first column is (.71339 .70999 .70147 .68914 .63755 .707) and if you divide each number by $\sqrt{2.886}$, you'll get the first eigenvector I listed in my hand computation. So the factor matrix printed by SPSS is a normalized matrix of eigenvectors. As you can see, PCA is exactly the same as finding eigenvalues and their associated eigenvectors.

The next matrix is the reproduced correlation matrix. That is, using all the eigenvalues and eigenvectors we perfectly reproduce the original correlation matrix. Usually you don't ask for all possible factors but a small number such as 2 or 3. The reproduced correlation matrix can be examined much like a residual matrix to see where the simplified model is messing up relative to the observed correlation matrix. Indeed, it is possible to compare the observed correlation matrix to the reproduced correlation matrix, in the spirit of a residual, to construct statistical tests (we'll do this later).

The next two matrices are the rotation matrix (what SPSS computed as the best rotation under the extraction method you requested) and the resulting rotated factor matrix. You interpret the rotated factor matrix. The underlying space is in terms of variables; in MDS terminology you are trying to interpret the dimensions of a space that has variables as points (not subjects as points). You may find it helpful to focus on the extreme factor loadings when trying to interpret the dimensions. That is, circle the largest numbers within each dimension and look for variables that have high numbers on one dimension but low numbers on the other dimensions.

The final matrix is the factor score matrix. This matrix gives the weights that you would use to create the factor scores manually. For instance, if you wanted to assign to each subject a score on (rotated) factor 1 you multiply, for each subject, the score on the variable with the numbers in the first column of the factor score coefficient matrix. For example, (-.04183)*variableA + (-.09941)*variableB + . . . + (-.16979)*variableF, where for each variable you substitute the subject's score. This allows you to create a score for

each subject on each factor. In order to do this, you obviously need the raw data to plug in each subject's scores; it cannot be done from the correlation matrix). The factor score coefficient matrix that SPSS prints out is based on the rotated solution.[5] In SPSS if you multiply the factor score matrix by $\sqrt{\lambda}$ (the eigenvalue), you get back the eigenvector.

I personally prefer to interpret the factors using the factor score coefficient matrix instead of the factor matrix because the former is expressed directly in terms of the estimated weights used to create the factors so they are analogous to βs in a regression. I am in a very small minority, however, as most people prefer to interpret the elements in the factor matrix. Usually the two matrices are very similar, but in some cases they may differ; when they differ, it is the factor score matrix that is interpretable. The reason has to do with features of linear combinations that are masked by correlations (see Harris, *Prime of Multivariate Statistics* for examples). The two will differ when rotations are performed (or when numbers other than 1 appear in the diagonal of the correlation matrix, as some types of factor analysis require).

rotated factor matrix

The rotated factor matrix is identical to the Pearson correlation of every observed variable with every rotated factor score. In other words, if you compute the factor score for each subject on factor 1 using the factor score coefficient matrix and correlated these factor scores with observed variable A, the correlation is identical to the first entry in the factor matrix (i.e., the loading of variable A on Factor 1).

The output also mentions communalities. These are the sum of squares of the rows of the loading matrix. Each variable has its own row in the factor matrix, hence its own communality. When all possible factors are included all communalities will be one (for the principal components form of factor extraction). As you remove factors, the communalities will decrease.

As with MDS we can use a scree plot to help decide on the number of factors (see Figure 11-7). The SPSS FACTOR procedure has a built in scree plot.

Usually, we aren't interested in extracting all possible factors. Instead, we want a smaller number of factors, just like in MDS when we only wanted a few dimensions. I will rerun this example, this time requesting only two factors (the SPSS default is to take all factors having eigenvalues greater than 1). I used the same syntax as above except for omitting the subcommand /criteria = factor(6). The resulting output follows, with corresponding scree plot (Figure 11-7) and rotated factor space (Figure 11-8):

[5] Different programs/textbooks use different terms. What SPSS refers to as factor score coefficient matrix is sometimes called the "factor pattern matrix". What SPSS calls the factor matrix is sometimes called the "factor structure" matrix. Other terms for the various matrices are sometimes used too. Different versions of SPSS also use different terms such as "component" score coefficient matrix. One can go nuts in this literature.

Variable	Communality	*	Factor	Eigenvalue	Pct of Var	Cum Pct
		*				
A	1.00000	*	1	2.88634	48.1	48.1
B	1.00000	*	2	1.01781	17.0	65.1
C	1.00000	*	3	.65258	10.9	75.9
D	1.00000	*	4	.56233	9.4	85.3
E	1.00000	*	5	.48946	8.2	93.5
F	1.00000	*	6	.39149	6.5	100.0

Factor Matrix:

	Factor 1	Factor 2
A	.71339	-.38879
B	.70999	-.48354
C	.70147	-.32391
D	.68914	.42512
E	.63755	.45445
F	.70703	.37506

Final Statistics:

Variable	Communality	*	Factor	Eigenvalue	Pct of Var	Cum Pct
		*				
A	.66008	*	1	2.88634	48.1	48.1
B	.73789	*	2	1.01781	17.0	65.1
C	.59699	*				
D	.65564	*				
E	.61299	*				
F	.64056	*				

Reproduced Correlation Matrix:

	A	B	C	D	E	F
A	.66008*	-.14449	-.19636	-.00634	.00186	.00143
B	.69449	.73789*	-.15466	-.03372	.07709	-.00062
C	.62636	.65466	.59699*	.04429	-.05002	-.04448
D	.32634	.28372	.34571	.65564*	-.20255	-.15669
E	.27814	.23291	.30002	.63255	.61299*	-.18121
F	.35857	.32062	.37448	.64669	.62121	.64056*

The lower left triangle contains the reproduced correlation matrix; the diagonal, reproduced communalities; and the upper right triangle residuals between the observed correlations and the reproduced correlations.

There are 8 (53.0%) residuals (above diagonal) with absolute values > 0.05.

VARIMAX converged in 3 iterations.

Rotated Factor Matrix:

	Factor 1	Factor 2
A	.78386	.21364
B	.84703	.14293
C	.73034	.25218
D	.20267	.78394
E	.14514	.76937

```
F               .25024        .76023

Factor Transformation Matrix:

                Factor  1     Factor  2

Factor  1        .72133        .69259
Factor  2       -.69259        .72133


Factor Score Coefficient Matrix:


               Factor  1     Factor  2

A               .44284       -.10436
B               .50647       -.17233
C               .39572       -.06124
D              -.11706        .46665
E              -.14991        .47506
F              -.07852        .43547
```

You can see that this reproduced correlation matrix isn't a perfect reproduction of the original correlation matrix, but it isn't too far off from the matrix of observed correlations. Thus, the first two factors (aka eigenvectors) provide a reasonable representation of the six variables without too much loss of information.

Eigenvalues and eigenvectors are useful in many respects. Google, the internet search company, uses eigenvectors as the guts of their search engine. A distance matrix is set up indicating which web pages call which web pages (a "link" matrix) and the eigenvectors of that matrix are used in Google's PageRank algorithm to decide how to rank the importance of web pages that emerge from a search (aside from any dollars a company has paid Google to make their page show up first...).

(e) Eigenvalues and eigenvectors in R

Here I show how to get the eigenvalues and the eigenvectors from a covariance or a correlation matrix as input. This mimics the earlier description of a simple spectral decomposition of the 6x6 correlation matrix on page 11-21.

```
cors <- c(1, 0.55, 0.43, 0.32, 0.28, 0.36, 0.55, 1, 0.5, 0.25,
    0.31, 0.32, 0.43, 0.5, 1, 0.39, 0.25, 0.33, 0.32, 0.25, 0.39,
    1, 0.43, 0.49, 0.28, 0.31, 0.25, 0.43, 1, 0.44, 0.36, 0.32,
    0.33, 0.49, 0.44, 1)
cormat <- matrix(cors, 6, 6, byrow = T)
cormat
```

Figure 11-7: Scree plot for six variables.

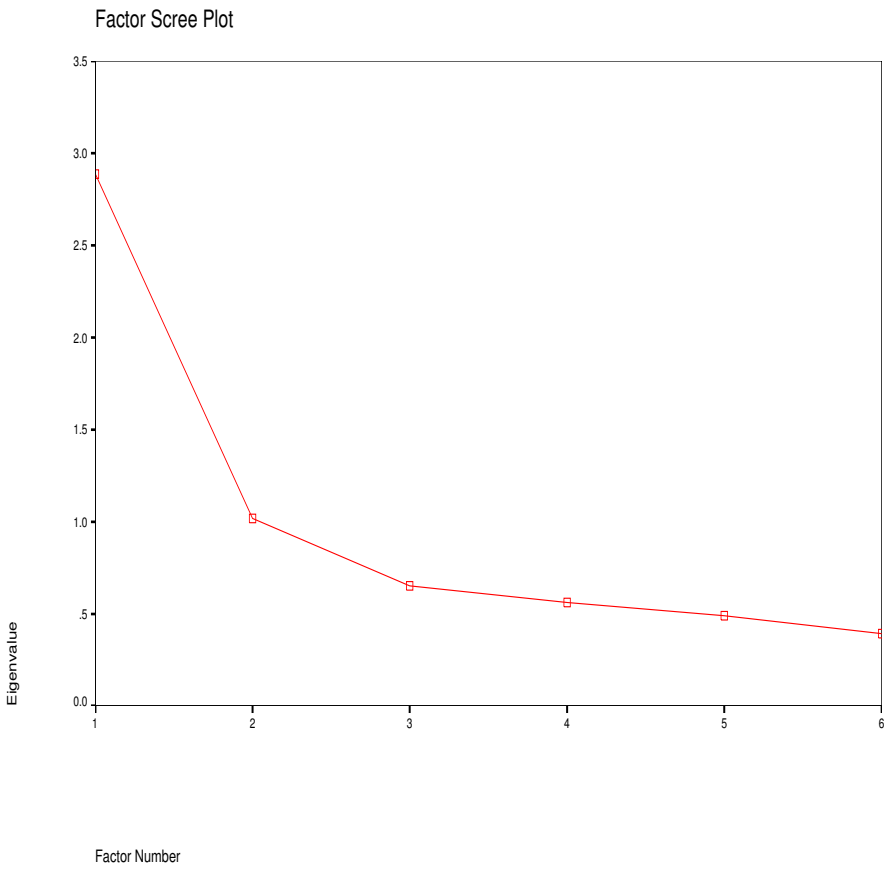
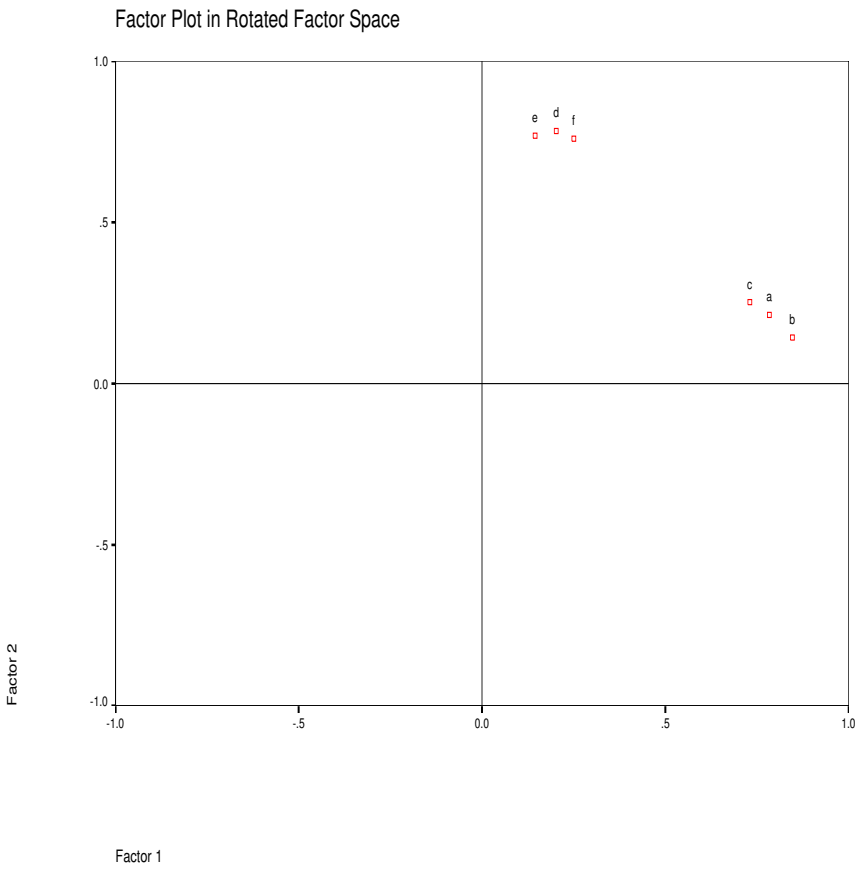


Figure 11-8: Rotated factor space for six variables.



```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 1.00 0.55 0.43 0.32 0.28 0.36
## [2,] 0.55 1.00 0.50 0.25 0.31 0.32
## [3,] 0.43 0.50 1.00 0.39 0.25 0.33
## [4,] 0.32 0.25 0.39 1.00 0.43 0.49
## [5,] 0.28 0.31 0.25 0.43 1.00 0.44
## [6,] 0.36 0.32 0.33 0.49 0.44 1.00
```

```
pca <- svd(cormat)
pca
```

```
## $d
## [1] 2.8863413 1.0178078 0.6525765 0.5623258 0.4894629
## [6] 0.3914857
##
## $u
##            [,1]       [,2]         [,3]         [,4]
## [1,] -0.4199064  0.3853701 -0.230390275  0.464546537
## [2,] -0.4179033  0.4792903 -0.288917090 -0.142878565
## [3,] -0.4128937  0.3210685  0.558477833 -0.422244477
## [4,] -0.4056314 -0.4213843  0.484598681  0.009333956
## [5,] -0.3752666 -0.4504563 -0.562761718 -0.524235768
## [6,] -0.4161638 -0.3717662  0.003621186  0.557298457
##             [,5]        [,6]
## [1,] -0.50388288 -0.39036827
## [2,]  0.19287651  0.67420239
## [3,]  0.26211040 -0.40932147
## [4,] -0.50993611  0.40365898
## [5,] -0.06778699 -0.24524509
## [6,]  0.61283678 -0.04933518
##
## $v
##            [,1]       [,2]         [,3]         [,4]
## [1,] -0.4199064  0.3853701 -0.230390275  0.464546537
## [2,] -0.4179033  0.4792903 -0.288917090 -0.142878565
## [3,] -0.4128937  0.3210685  0.558477833 -0.422244477
## [4,] -0.4056314 -0.4213843  0.484598681  0.009333956
## [5,] -0.3752666 -0.4504563 -0.562761718 -0.524235768
## [6,] -0.4161638 -0.3717662  0.003621186  0.557298457
##             [,5]        [,6]
## [1,] -0.50388288 -0.39036827
## [2,]  0.19287651  0.67420239
## [3,]  0.26211040 -0.40932147
```

```
## [4,] -0.50993611  0.40365898
## [5,] -0.06778699 -0.24524509
## [6,]  0.61283678 -0.04933518
```

Here I show how to recreate the original matrix with the eigenvalue/eigenvector pairs (for 1 eigenvector; for 1 and 2; for 1, 2, and 3; etc).

```
# little function in R to recreate the input matrix from
# the eigenvalue/eigenvector pairs
decomp <- function(pca, dim) {
    beta <- diag(pca$d)[1:dim, 1:dim]
    if (dim == 1)
        beta <- as.matrix(beta)
    u <- pca$u[, 1:dim]
    v <- t(pca$v)[1:dim, ]
    return(round(t(u %*% beta %*% v), 2))
}

decomp(pca, 1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.51 0.51 0.50 0.49 0.45 0.50
## [2,] 0.51 0.50 0.50 0.49 0.45 0.50
## [3,] 0.50 0.50 0.49 0.48 0.45 0.50
## [4,] 0.49 0.49 0.48 0.47 0.44 0.49
## [5,] 0.45 0.45 0.45 0.44 0.41 0.45
## [6,] 0.50 0.50 0.50 0.49 0.45 0.50
```

```
decomp(pca, 2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.66 0.69 0.63 0.33 0.28 0.36
## [2,] 0.69 0.74 0.65 0.28 0.23 0.32
## [3,] 0.63 0.65 0.60 0.35 0.30 0.37
## [4,] 0.33 0.28 0.35 0.66 0.63 0.65
## [5,] 0.28 0.23 0.30 0.63 0.61 0.62
## [6,] 0.36 0.32 0.37 0.65 0.62 0.64
```

```
decomp(pca, 3)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.69 0.74 0.54 0.25 0.36 0.36
## [2,] 0.74 0.79 0.55 0.19 0.34 0.32
## [3,] 0.54 0.55 0.80 0.52 0.09 0.38
## [4,] 0.25 0.19 0.52 0.81 0.45 0.65
## [5,] 0.36 0.34 0.09 0.45 0.82 0.62
## [6,] 0.36 0.32 0.38 0.65 0.62 0.64
```

```
decomp(pca, 4)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.82 0.70 0.43 0.26 0.23 0.50
## [2,] 0.70 0.80 0.58 0.19 0.38 0.28
## [3,] 0.43 0.58 0.90 0.52 0.22 0.24
## [4,] 0.26 0.19 0.52 0.81 0.45 0.65
## [5,] 0.23 0.38 0.22 0.45 0.97 0.46
## [6,] 0.50 0.28 0.24 0.65 0.46 0.82
```

```
decomp(pca, 5)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.94 0.65 0.37 0.38 0.24 0.35
## [2,] 0.65 0.82 0.61 0.14 0.37 0.33
## [3,] 0.37 0.61 0.93 0.45 0.21 0.32
## [4,] 0.38 0.14 0.45 0.94 0.47 0.50
## [5,] 0.24 0.37 0.21 0.47 0.98 0.44
## [6,] 0.35 0.33 0.32 0.50 0.44 1.00
```

```
# using all 6 recreates the original matrix perfectly
decomp(pca, 6)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 1.00 0.55 0.43 0.32 0.28 0.36
## [2,] 0.55 1.00 0.50 0.25 0.31 0.32
## [3,] 0.43 0.50 1.00 0.39 0.25 0.33
## [4,] 0.32 0.25 0.39 1.00 0.43 0.49
## [5,] 0.28 0.31 0.25 0.43 1.00 0.44
## [6,] 0.36 0.32 0.33 0.49 0.44 1.00
```

The R appendix mentions several packages for doing PCA. These packages typically take the raw data as input, internally compute the correlation or covariance matrix (user specifies which one they want), and produce a variety of output, some of the packages also produce a variety of useful plots as shown later in these lecture notes as I work through some examples.

(f) More intuition on PCA

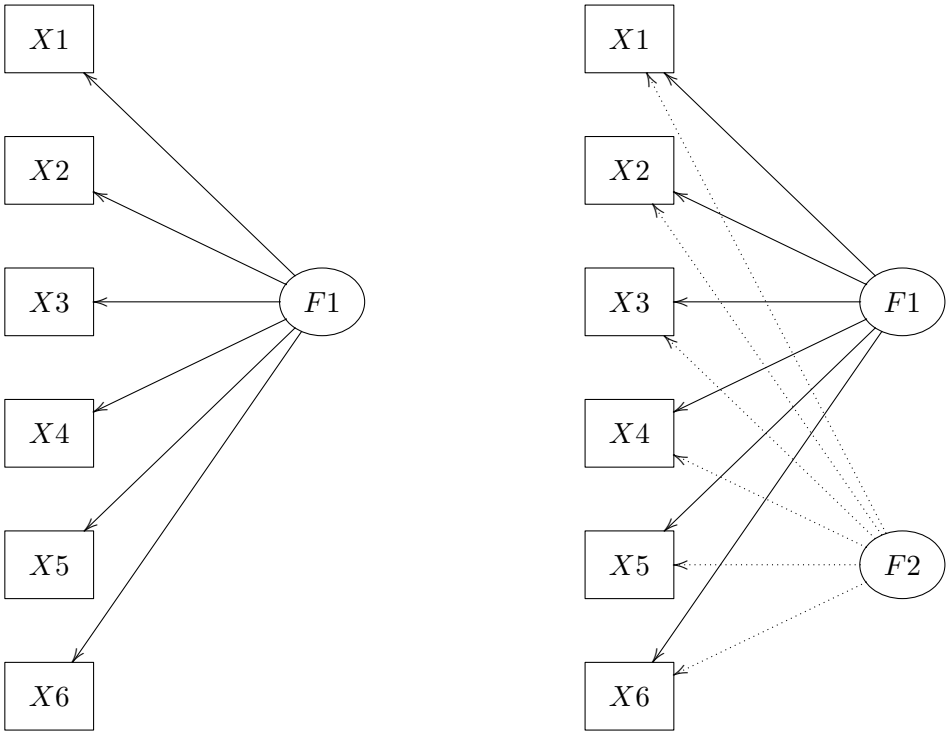
One way to gain intuition about PCA is by using the visual representation of a graph with a few conventions: denote observed variables as squares unobserved variables as circles. A component (aka factor, dimension) is an unobserved variable so would be drawn as a circle. We draw arrows from the circles to the square to indicate that, according the model, the unobserved factor "creates" (aka gives rise to) the observed variable. The factor score corresponds to an individual's value on the unobserved variable. An element in the eigenvector is the numerical value associated with each arrow (e.g. the 3rd element of the eigenvector corresponds to the weight, or arrow, that one multiplies the factor score to yield the observed score). See Figure 11-9. A second factor would be represented with a second circle, and the associated eigenvector would be represented by new arrows emerging from the new circle to each of the observed variables. In PCA the two factors (F1 and F2) are independent from each other (recall that eigenvectors are orthogonal), but there are extensions that allow the factors to correlate. The two factors correspond to two dimensions in MDS.

We will come back to these types of representations in Lecture #13 when discussing structural equation modeling (SEM). SEM allows one to set some arrows (elements in the eigenvector) to zero in order to allow for statistical tests of research hypotheses. If you predict that a particular observed variable is unrelated to the underlying factor, then you will want to test whether the value of the eigenvector corresponding to that variable is zero. SEM can do much more such as allowing correlations between factors, testing regression models in the context of simultaneously creating factors (such as one factor used to predict another factor), and more complicated latent variable models such as latent growth curves (each subjects has his/her own slope and intercept).

(g) Published example using PCA: depression and rumination

This example comes from a short paper I did with Wendy Treynor and Susan Nolen-Hoeksema (Treynor, Gonzalez, and Nolen-Hoeksema, 2003, *Cognitive Therapy and Research, 27,* 247-59). We studied a standard self-report scale of rumination that has 22 items. Using PCA we noted that 10 items within the scale formed two dimensions: one dimension we interpreted as pondering (cognitive aspect of rumination) and the other dimension we interpreted as brooding (affective aspect of rumination). There were over 1300 participants from a community based sample. Figure 11-10 shows the key aspects

Figure 11-9: LEFT: Representation of a single factor (F1) and its corresponding observed variables (X1 to X6). RIGHT: Representation of two factors (F1 and F2) and corresponding observed variables (X1 to X6).



from the PCA output: the eigenvalues and their percentage accounted for measures, the scree plot, the components matrix, and the components plot. We'll discuss the details in class.

(h) Creating factors from raw data

It is instructive to examine how to compute the factors from the information in the SPSS and R output. This will be useful when you want to plot subjects in the computed space rather than variables and will also make it clearer how PCA works.

I will present two examples—one that is simple and can show the hand computations, another that is with actual data.

The first example is taken from Harris, *A Primer of Multivariate Statistics*. The following table shows five subjects measured on two variables (here data are entered in the way one would for ANOVA and regression with each row being a different subject).

Subject	X_1	X_2
1	2	0
2	4	1
3	1	3
4	-7	-3
5	0	-1

The variance-covariance matrix and the correlation matrix for these data are:

Covariance Matrix

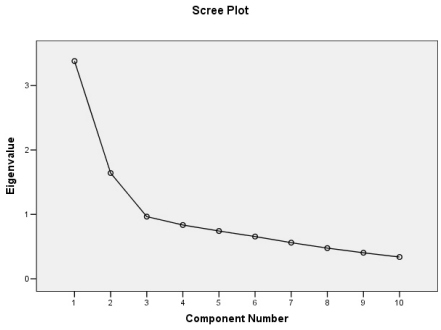
variable	X_1	X_2
X_1	17.5	7
X_2	7	5

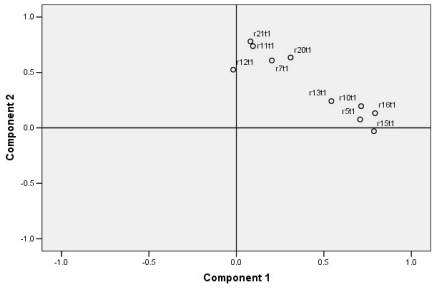
Correlation Matrix

variable	X_1	X_2
X_1	1	.748
X_2	.748	1

First, I'll do a PC on the covariance matrix. The eigenvalues of the covariance matrix are 20.635 and 1.865. The sum of these two eigenvalues equals the sum of the diagonal of the covariance matrix (17.5 + 5 = 22.5). The two eigenvectors are (.913, .409) and

Figure 11-10: PCA from rumination example. The components plot shows two clusters of five items.





Rotated Component Matrix(a)

	Component	
	1	2
r5t1	.707	.076
r7t1	.203	.608
r10t1	.713	.196
r11t1	.095	.737
r12t1	-.018	.525
r13t1	.543	.241
r15t1	.786	-.030
r16t1	.793	.132
r20t1	.310	.635
r21t1	.081	.780

Extraction Method: Principal Component Analysis.
Rotation Method: Varimax with Kaiser Normalization.
a Rotation converged in 3 iterations.

Total Variance Explained

	Initial Eigenvalues			Extraction Sums of Squared Loadings			Rotation Sums of Squared Loadings		
Component	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %	Total	% of Variance	Cumulative %
1	3.381	33.810	33.810	3.381	33.810	33.810	2.703	27.028	27.028
2	1.642	16.419	50.230	1.642	16.419	50.230	2.320	23.202	50.230
3	.965	9.654	59.883						
4	.835	8.351	68.234						
5	.742	7.420	75.654						
6	.656	6.558	82.212						
7	.561	5.607	87.819						
8	.476	4.762	92.581						
9	.404	4.039	96.620						
10	.338	3.380	100.000						

Extraction Method: Principal Component Analysis.

(-.409, .913). The first factor is created by taking the first eigenvector and multiplying that with the data matrix (i.e., .913X_1 + .409X_2); similarly for the second factor. The subject values for each factor are listed below:

Subject	X_1	X_2	Factor 1	Factor 2
1	2	0	1.826	-.818
2	4	1	4.061	-.783
3	1	3	2.140	2.33
4	-7	-3	-7.618	.184
5	0	-1	-.409	-.913

In this way, factors are created directly from data and the eigenvectors. One can go backwards from the factors back to the data by taking the transpose of the eigenvector matrix (described later), e.g., raw data X_1 can be reproduced by taking .913factor1 - .409factor2, and raw data X_2 can be reproduced by taking .409factor1 + .913factor2. For example, we can recreate the observed score for the first subject by taking that subject's two factor scores, multiplying each by the values of the eigenvector, resulting in .913*1.826 + (-.409)*(-.818) = 2.[6]

If you convert the raw data to Z-scores and repeat everything I did above but with Z-scores the covariance matrix will be identical to the correlation matrix. In that case, the eigenvalues are 1.748 and .252 (again, their sum is identical to the sum of the diagonal, which in this case is 2), and the two eigenvectors are (-.707, -.707) and (.707, -.707). Thus, the first eigenvector is like the unit contrast (1, 1) and the second eigenvector is like the (1, -1) contrast. This suggests that the first factor is assessing something about how the two variables sum whereas the second factor is assessing something about the difference between the two variables. The table of Z-scores and corresponding factor scores (following the same computation as above but with Z-scores instead of raw data) is below:

Subject	ZX_1	ZX_2	Factor 1	Factor 2
1	.478	0	-.388	.338
2	.956	.447	-.992	.3599
3	.239	1.34	-1.12	-.780
4	-1.673	-1.341	2.132	-.235
5	0	-.447	.316	.316

Again, it is possible to go backwards from the factor scores to the original Z-scores.

[6]For those following the matrix algebra. If the eigenvector matrix has the eigenvectors as the columns, the PC factors are created using the columns of the eigenvector matrix whereas the raw scores are created from the PC factors using the rows of the eigenvector matrix.

So, the distinction between doing a principal components on a covariance matrix versus a correlation matrix boils down to whether you want to analyze raw data or Z-scores. When variables are measured on different scales (such as height, weight, a 7pt scale, reaction time) the results from a covariance matrix may be difficult to interpret because you have to keep scale in mind; a correlation matrix renormalizes everything so all variables are on the same scale. Also, if some variables have much larger variances than other variables, a PCA on the covariance matrix may be influenced by the variables with greater variance; this points to one advantage for doing PCA on the correlation matrix. The correlation matrix forces all variables to have the same variance and thus to be on the same scale.

In this second example I will use the raw data from the track dataset (record times for each country across different track events for men). Here I will show some additional features of PCA and make connections to SPSS output. The syntax is given by:

```
data list free
 /country (A20) m100 m200 m400 m800 m1500 k5 k10 marathon.

begin data
 Argentin 10.39 20.81 46.84  1.81   3.7 14.04 29.36  137.72
 Australi 10.31 20.06 44.84  1.74  3.57 13.28 27.66   128.3
 Austria  10.44 20.81 46.82  1.79   3.6 13.26 27.72   135.9
 Belgium  10.34 20.68 45.04  1.73   3.6 13.22 27.45  129.95
 Bermuda  10.28 20.58 45.91   1.8  3.75 14.68 30.55  146.62
 Brazil   10.22 20.43 45.21  1.73  3.66 13.62 28.62  133.13
 Burma    10.64 21.52  48.3   1.8  3.85 14.45 30.28  139.95
 Canada   10.17 20.22 45.68  1.76  3.63 13.55 28.09  130.15
 Chile    10.34  20.8  46.2  1.79  3.71 13.61  29.3  134.03
 China    10.51 21.04  47.3  1.81  3.73  13.9 29.13  133.53
 Columbia 10.43 21.05  46.1  1.82  3.74 13.49 27.88  131.35
 CookIs   12.18  23.2 52.94  2.02  4.24  16.7 35.38   164.7
 CostaR   10.94  21.9 48.66  1.87  3.84 14.03 28.81  136.58
 Czechosl 10.35 20.65 45.64  1.76  3.58 13.42 28.19  134.32
 Denmark  10.56 20.52 45.89  1.78  3.61  13.5 28.11  130.78
 Dominica 10.14 20.65  46.8  1.82  3.82 14.91 31.45  154.12
 Finland  10.43 20.69 45.49  1.74  3.61 13.27 27.52  130.87
 France   10.11 20.38 45.28  1.73  3.57 13.34 27.97   132.3
 EastG    10.12 20.33 44.87  1.73  3.56 13.17 27.42  129.92
 WestG    10.16 20.37  44.5  1.73  3.53 13.21 27.61  132.23
 UK       10.11 20.21 44.93   1.7  3.51 13.01 27.51  129.13
 Greece   10.22 20.71 46.56  1.78  3.64 14.59 28.45   134.6
 Guatemal 10.98 21.82  48.4  1.89   3.8 14.16 30.11  139.33
 Hungary  10.26 20.62 46.02  1.77  3.62 13.49 28.44  132.58
 India     10.6 21.42 45.73  1.76  3.73 13.77 28.81  131.98
 Indonesi 10.59 21.49  47.8  1.84  3.92 14.73 30.79  148.83
 Ireland  10.61 20.96  46.3  1.79  3.56 13.32 27.81  132.35
 Israel   10.71    21  47.8  1.77  3.72 13.66 28.93  137.55
 Italy    10.01 19.72 45.26  1.73   3.6 13.23 27.52  131.08
 Japan    10.34 20.81 45.86  1.79  3.64 13.41 27.72  128.63
 Kenya    10.46 20.66 44.92  1.73  3.55  13.1 27.38  129.75
 SouthK   10.34 20.89  46.9  1.79  3.77 13.96 29.23  136.25
 NorthK   10.91 21.94  47.3  1.85  3.77 14.13 29.67  130.87
 Luxembou 10.35 20.77  47.4  1.82  3.67 13.64 29.08  141.27
 Malaysia  10.4 20.92  46.3  1.82   3.8 14.64 31.01   154.1
 Mauritiu 11.19 22.45  47.7  1.88  3.83 15.06 31.77  152.23
 Mexico   10.42  21.3  46.1   1.8  3.65 13.46 27.95   129.2
```

```
 Netherla 10.52 20.95  45.1  1.74  3.62 13.36 27.61  129.02
 NewZea   10.51 20.88  46.1  1.74  3.54 13.21  27.7  128.98
 Norway   10.55 21.16 46.71  1.76  3.62 13.34 27.69  131.48
 Papua    10.96 21.78  47.9   1.9  4.01 14.72 31.36  148.22
 Philippi 10.78 21.64 46.24  1.81  3.83 14.74 30.64  145.27
 Poland   10.16 20.24 45.36  1.76   3.6 13.29 27.89  131.58
 Portugal 10.53 21.17  46.7  1.79  3.62 13.13 27.38  128.65
 Rumania  10.41 20.98 45.87  1.76  3.64 13.25 27.67   132.5
 Singapor 10.38 21.28  47.4  1.88  3.89 15.11 31.32  157.77
 Spain    10.42 20.77 45.98  1.76  3.55 13.31 27.73  131.57
 Sweden   10.25 20.61 45.63  1.77  3.61 13.29 27.94  130.63
 Switzerl 10.37 20.46 45.78  1.78  3.55 13.22 27.91   131.2
 Taiwan   10.59 21.29  46.8  1.79  3.77 14.07 30.07  139.27
 Thailand 10.39 21.09 47.91  1.83  3.84 15.23 32.56   149.9
 Turkey   10.71 21.43  47.6  1.79  3.67 13.56 28.58   131.5
 USA       9.93 19.75 43.86  1.73  3.53  13.2 27.43  128.22
 USSR     10.07    20  44.6  1.75  3.59  13.2 27.53  130.55
 WSamoa   10.82 21.86    49  2.02  4.24 16.28 34.71  161.83
end data.

factor variables = m100 m200 m400 m800 m1500 k5 k10 marathon
 /plot eigen rotation
 /criteria=factors (2)
 /print all.
```

Note that some of these times are in seconds and other times are in minutes. This is not a problem for the FACTOR procedure in SPSS because it is based on the correlation matrix, which is independent of scale[7]. The resulting output is (abbreviated):

```
Variable     Communality  *  Factor   Eigenvalue   Pct of Var   Cum Pct
                          *
M100             1.00000  *      1       6.62215       82.8         82.8
M200             1.00000  *      2        .87762       11.0         93.7
M400             1.00000  *      3        .15932        2.0         95.7
M800             1.00000  *      4        .12405        1.6         97.3
M1500            1.00000  *      5        .07988        1.0         98.3
K5               1.00000  *      6        .06797         .8         99.1
K10              1.00000  *      7        .04642         .6         99.7
MARATHON         1.00000  *      8        .02260         .3        100.0

Factor Matrix:
              Factor  1     Factor  2

M100            .81718        .53106
M200            .86717        .43246
M400            .91520        .23259
M800            .94875        .01164
M1500           .95937       -.13096
K5              .93766       -.29231
K10             .94384       -.28747
MARATHON        .87990       -.41123

Final Statistics:
Variable     Communality  *  Factor   Eigenvalue   Pct of Var   Cum Pct
                          *
```

[7]However, in more general situations some programs may define principal components on the covariance matrix, and then scale will matter. In that case, the investigator should put the variables on the same scale otherwise some events will receive less "weight" than they deserve. Watch out—not all versions of SPSS allow the FACTOR procedure to analyze a covariance matrix.

M100	.94981	*	1	6.62215	82.8	82.8
M200	.93900	*	2	.87762	11.0	93.7
M400	.89169	*				
M800	.90027	*				
M1500	.93754	*				
K5	.96466	*				
K10	.97346	*				
MARATHON	.94332	*				

Reproduced Correlation Matrix:

	M100	M200	M400	M800	M1500	K5	K10	MARATHON
M100	.94981*	-.01566	-.03026	-.02546	-.01420	.00845	.01391	.01930
M200	.93830	.93900*	-.04349	-.02114	-.00035	.00868	.00240	.01100
M400	.87141	.89422	.89169*	-.00084	-.01229	-.01155	-.00973	-.00465
M800	.78149	.82776	.87101	.90027*	.00936	-.02261	-.02307	-.02354
M1500	.71443	.77530	.84756	.90868	.93754*	-.00974	-.00844	-.03245
K5	.61101	.68670	.79016	.88621	.93785	.96466*	.00560	-.01307
K10	.61862	.69414	.79694	.89212	.94314	.96903	.97346*	-.00552
MARATHON	.50065	.58518	.70964	.83002	.89800	.94525	.94869	.94332*

The lower left triangle contains the reproduced correlation matrix; the diagonal, reproduced communalities; and the upper right triangle residuals between the observed correlations and the reproduced correlations.

There are 0 (.0%) residuals (above diagonal) with absolute values > 0.05.
VARIMAX converged in 3 iterations.

Rotated Factor Matrix:

	Factor 1	Factor 2
M100	.27430	.93519
M200	.37644	.89291
M400	.54306	.77252
M800	.71240	.62670
M1500	.81333	.52539
K5	.90194	.38881
K10	.90346	.39651
MARATHON	.93554	.26095

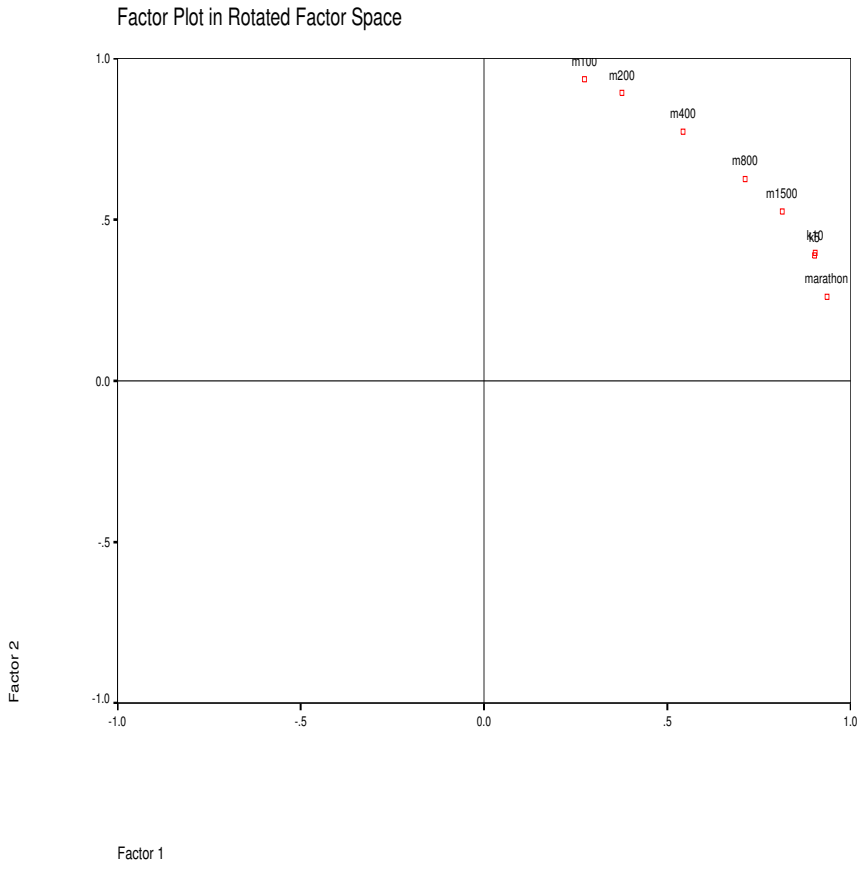
Factor Transformation Matrix:

	Factor 1	Factor 2
Factor 1	.75888	.65124
Factor 2	-.65124	.75888

Factor Score Coefficient Matrix:

	Factor 1	Factor 2
M100	-.30042	.53957
M200	-.22153	.45922
M400	-.06771	.29112
M800	.10008	.10337

Figure 11-11: Rotated factor space for six variables.



M1500	.20712	-.01890
K5	.32436	-.16055
K10	.32148	-.15576
MARATHON	.40598	-.26906

The factor score coefficient matrix can be used to create the factors. First, convert all raw data into Z-scores. In SPSS you can create Z-scores through the DESCRIPTIVES command using the "/SAVE" subcommand. This will append to your data file Z-scores of all the variables you listed. The variable names will be the same except that the letter Z is added to the beginning of each variable name[8]. Once all variables have been converted to Z-scores, then you can use the factor score coefficient matrix for weights. For

[8]You could equivalently compute Z-scores manually using the compute command. Take each variable, subtract the mean, and divide the difference by the standard deviation.

example, factor 1 = (-.300*ZM100) + (-.2215*ZM200) + . . . + (.40598*ZMARATH). Make sure you understand where I got these weights from the output. You could compute factor scores either for the rotated solution or for the unrotated solution just by using the proper factor score coefficient matrix (i.e., to create the unrotated factors rerun the FACTOR procedure with the subcommand /rotation=norotate and you'll get the needed factor score coefficients weights for the unrotated solution). It is safest to compute factor scores on whichever factor structure you are going to interpret (if you interpret the rotated factor space, then compute the factor scores for the rotated factors). Fortunately, the FACTOR procedure has a way of computing the factor scores and saving them into your data matrix. Just add the line "/SAVE reg(all)" to the FACTOR command (see Appendix 2).

It is possible to plot the factor scores for each country based on the factor structure. This factor space with individual cases plotted as points is usually useful in research, but it is rarely used. This is in the spirit of the individual space in INDSCAL and the other plot with variables is analogous to the stimulus configuration space.

Here is a version of this plot in R on the unrotated solution, so the first component is essentially the sum of all events and the second is essentially the difference between short and long events.

```
library(FactoMineR)
library(factoextra)
data <-
  read.table("track.data")
data <- data.frame(data[,-1],row.names=data[,1])
names(data) <- c("M100", "M200", "M400", "M800",
   "M1500", "K5", "K10", "MARATHON")
out.pca.track <- PCA(data,scale.unit=T,ncp=2,graph=FALSE)
#could instead run the standard R pca command
#out.pca <- prcomp(data, scale=T,retx=T)

#print eigenvalues and loadings; same as SPSS unrotated
#for standardized
out.pca.track$eig
```

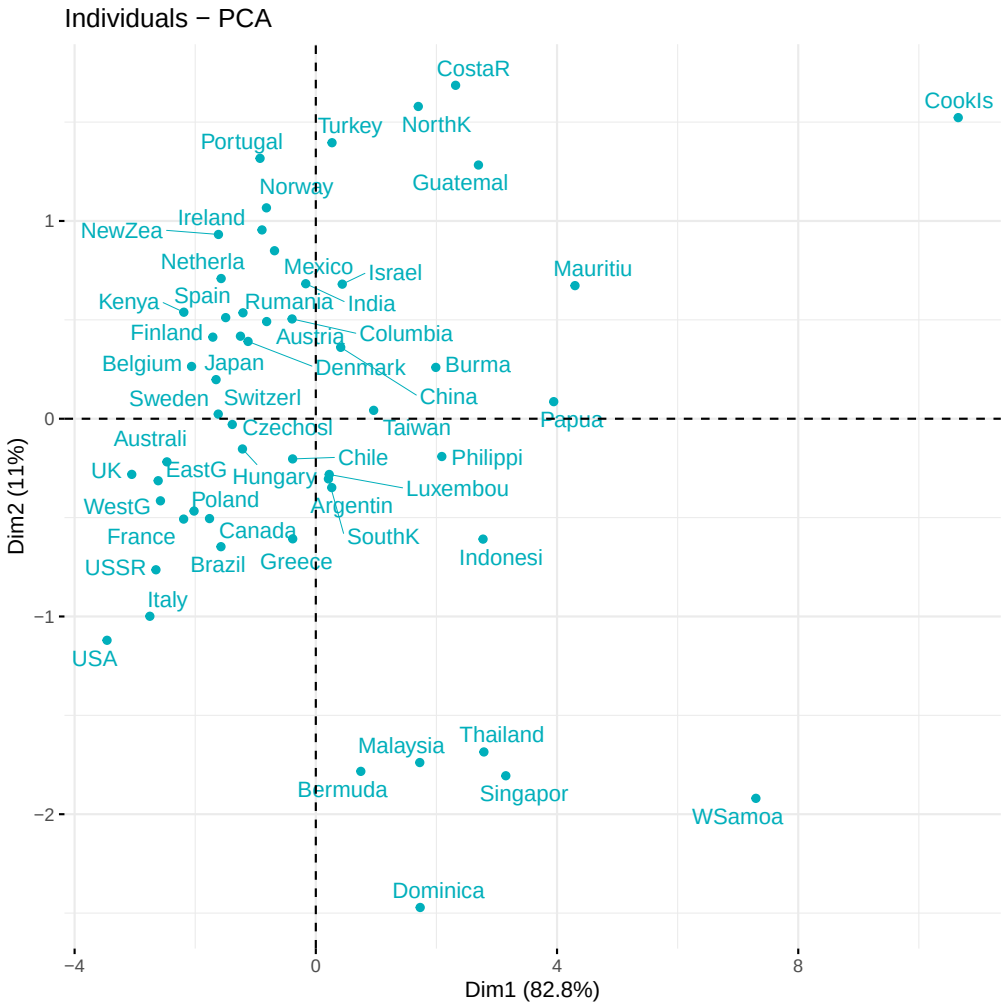
```
##        eigenvalue percentage of variance
## comp 1 6.62214613              82.7768266
## comp 2 0.87761829              10.9702287
## comp 3 0.15932114               1.9915143
## comp 4 0.12404939               1.5506173
## comp 5 0.07988027               0.9985034
## comp 6 0.06796515               0.8495644
```

```
## comp 7 0.04641953              0.5802441
## comp 8 0.02260010              0.2825012
##        cumulative percentage of variance
## comp 1                           82.77683
## comp 2                           93.74706
## comp 3                           95.73857
## comp 4                           97.28919
## comp 5                           98.28769
## comp 6                           99.13725
## comp 7                           99.71750
## comp 8                          100.00000
```

```
out.pca.track$var$cor
```

```
##               Dim.1       Dim.2
## M100      0.8171850  0.53105812
## M200      0.8671665  0.43245706
## M400      0.9152012  0.23258563
## M800      0.9487545  0.01164452
## M1500     0.9593714 -0.13096330
## K5        0.9376633 -0.29231413
## K10       0.9438353 -0.28747025
## MARATHON 0.8798965 -0.41122586
```

```
#produce plat
#some points not plotted due to overlap
fviz_pca_ind(out.pca.track, col.ind = "#00AFBB",
   repel = TRUE)
```




```
#rotated solution through GPArotate package
library(GPArotation)
#slightly different algorithm than SPSS
Varimax(out.pca.track$var$cor)


## Orthogonal rotation method varimax converged.
## Loadings:
##          Dim.1 Dim.2
## M100     0.281 0.933
## M200     0.383 0.890
## M400     0.549 0.768
## M800     0.717 0.621
## M1500    0.817 0.519
## K5       0.905 0.382
## K10      0.906 0.390
```

```
## MARATHON 0.938 0.254
##
## Rotating matrix:
##        [,1]  [,2]
## [1,]  0.764 0.645
## [2,] -0.645 0.764


#when you try different programs you get slightly
#different results
#there are too many different ways of doing things,
#especially in FA
varimax(out.pca.track$var$cor)


## $loadings
##
## Loadings:
##          Dim.1 Dim.2
## M100     0.277 0.934
## M200     0.379 0.892
## M400     0.545 0.771
## M800     0.714 0.625
## M1500    0.815 0.523
## K5       0.903 0.386
## K10      0.905 0.394
## MARATHON 0.936 0.258
##
##                Dim.1 Dim.2
## SS loadings    4.202 3.298
## Proportion Var 0.525 0.412
## Cumulative Var 0.525 0.937
##
## $rotmat
##            [,1]      [,2]
## [1,]  0.7607065 0.6490960
## [2,] -0.6490960 0.7607065


#and another way
library(psych)
out.pca.psych <- principal(data,nfactors=2,
                           rotate="varimax",covar=F)
out.pca.psych
```

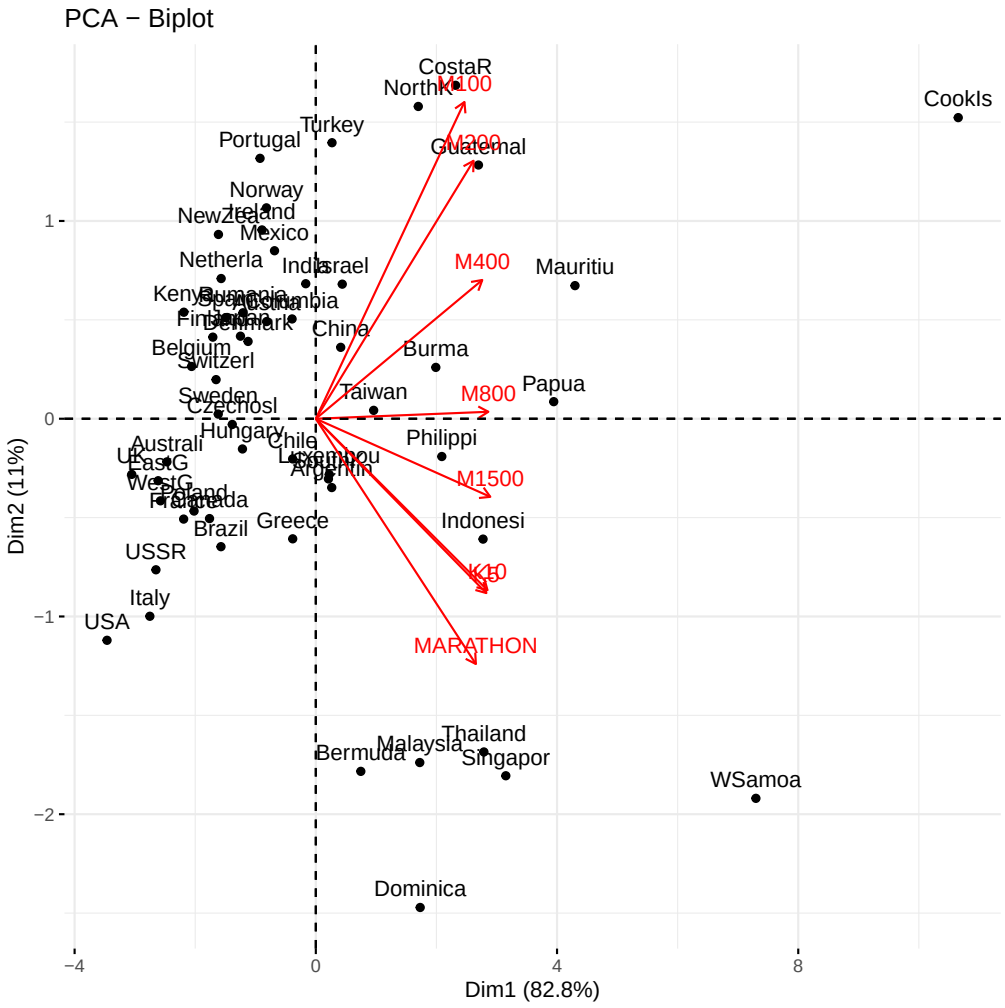
```
## Principal Components Analysis
## Call: principal(r = data, nfactors = 2, rotate = "varimax", covar = F
## Standardized loadings (pattern matrix) based upon correlation matrix
##           RC1  RC2   h2    u2 com
## M100     0.28 0.93 0.95 0.050 1.2
## M200     0.38 0.89 0.94 0.061 1.3
## M400     0.55 0.77 0.89 0.108 1.8
## M800     0.71 0.62 0.90 0.100 2.0
## M1500    0.81 0.52 0.94 0.062 1.7
## K5       0.90 0.39 0.96 0.035 1.4
## K10      0.90 0.39 0.97 0.027 1.4
## MARATHON 0.94 0.26 0.94 0.057 1.2
##
##                        RC1  RC2
## SS loadings           4.20 3.30
## Proportion Var        0.53 0.41
## Cumulative Var        0.53 0.94
## Proportion Explained  0.56 0.44
## Cumulative Proportion 0.56 1.00
##
## Mean item complexity =  1.5
## Test of the hypothesis that 2 components are sufficient.
##
## The root mean square of the residuals (RMSR) is  0.02
##  with the empirical chi square  0.96  with prob <  1
##
## Fit based upon off diagonal values = 1
```

Why is Cook Islands an outlier?

biplot and R

The biplot puts both the variable space and the subject space on the same plot (two spaces in the same plot, in the spirit of unfolding). Look at R's biplot() command which runs with both princomp and prcomp. The package psych also has a biplot command to interface with its fa() and principal() commands. Here I use the biplot command in the package factoextra to plot the PCA I ran in the previous command.

```
fviz_pca_biplot(out.pca.track, label = "all", col.var = "red",
    ggtheme = theme_minimal())
```



I've just put you through a lot of difficult material. The basic idea is that most people stop at the plot of the variable space (such as Figure 11-11 on page 11-41). But I think that the figure with the individual points plotted is just as, if not more, useful.

As we saw before there is a relation between the variables and the factor scores. The correlation between each variable and a factor is identical to an entry in the factor matrix. For example, I took the track data, converted all scores to Z-scores, used the factor score coefficient matrix from the rotated solution to create the rotated factors, and then correlated each factor with each raw variable. Here are the correlations between manually created factor 1 and each of the raw variables for the track data:

```
          M100   M200   M400   M800  M1500     K5    K10 MARATHON
factor 1 0.274 0.3763 0.5431 0.7126 0.8137 0.9018 0.9034   0.9352
```

Do you recognize these correlations? Look at the first column of the rotated factor matrix for the track data and you'll see that these correlations are identical to the entries in the first column. Thus, the factor matrix contains the correlations between each raw variable and each factor. Similarly, if you correlate the second factor with each observed variable, you would recreate the second column of the rotated factor matrix.

Recall that the factors themselves are not directly observed; they are created from the factor coefficient score matrix. This same connection between the correlations and the factor matrix holds for any factor score coefficient matrix (e.g., regardless of whether one uses the rotated or unrotated factor coefficient matrix). To verify this, let me try the same thing with the unrotated factors. When you correlate the unrotated factors with each observed variable, you get back the correlations printed in the unrotated factor matrix. Here are the correlations for the first unrotated factor; double check that these are the same correlations within roundoff error as those in the unrotated factor matrix.

```
              M100    M200    M400    M800   M1500      K5     K10 MARATHON
factor 1   0.81765 0.86761 0.91594 0.94866 0.95903 0.93734 0.94347 0.87942
```

communalities

Communalities correspond to the percent of variance in each observed variable accounted for by the factors. PCA requires that the factors be orthogonal, thus we can use the old formula from regression that the total R^2 is equal to the sum of the individual Pearson correlations squared. So, we have for the first variable M100 an $R^2 = .2743^2 + .93519^2 = .9498$; for the second variable M200 an $R^2 = .3764^2 + .89291^2 = .9390$, etc. Again, these are the percent of variance in each observed variable accounted for by the linear combination of principal components that have been extracted (in this example, two). If all principal components are extracted, then the observed variables will be reproduced perfectly and all R^2s are equal to 1. The communalities will remain invariant to rotation. You should check this (same communalities regardless of whether you use the factor matrix or the rotated factor matrix).

The computed factor scores for each subject are useful because they can be analyzed like any other variable—either as a predictor or a dependent variable in a subsequent analysis. For instance, you may want to test whether the factor scores for the men differ from the factor scores for the women. You could simply perform a two sample t-test comparing the mean factor score for men to the mean factor score for women. Later I will present techniques that merge principal components with ANOVA-like techniques that have the ability to test hypotheses between groups.

Short cut by actual users of PCA

In practice most people don't use the actual factor scores. Instead, people take a sensible short cut. If we interpret the first factor in the track example to be the four long distance variables, then it seems silly to include the four short distance events in the computation of the first factor. Similarly, it seems silly to include the four long events in

the computation of the second factor (short distance events). So most people use some kind of cutoff of the factor loadings (or factor scores). In effect to define factor 1 people use the weights (0, 0, 0, 0,.21, .32, .32, .41) so that the four short events get weights of zero and to define factor 2 they use the weights (.54, .46, .28, .1, 0, 0, 0, 0). Some people even go further and don't bother giving such specific weights to the items that get nonzero weights. Why not just use the weights (0,0,0,0,1,1,1,1) and (1,1,1,1,0,0,0,0) for the two factors respectively? Hey, those look mighty close to two contrasts (yes, they are related to contrasts as we will see in Lecture Notes 12). This short cut makes sense to me because we are defining factors based on overall pattern. It is more likely that the general weights such as (0,0,0,0,1,1,1,1) will replicated over different studies; it is less likely that the specific weights of, say, (-.3, -.2, -.07, .1, .21, .32, .32, .41) that a program may use to create factor 1 will replicate across different studies. So both for ease in interpreting the factors (variables are either "in or out", for you Project Runway TV show fans) and for robustness, it makes sense to take this short cut and use simple 1s and 0s when creating factor scores. SPSS and R, as far as I can tell, do not allow this short cut to occur in how it saves factor scores to the data file. You'll need to do this manually through the COMPUTE command. Oh, one detail you should be mindful about is that the factor score weights do have some information about scale, so you may want to deviate from just using "1s" as weights for the variables that define the factor. For example, it wouldn't make much sense to add a variable measured in seconds to a variable measured in minutes (or add a variable that is measured in seconds to one that is measured as a 7 point Likert scale).

Machine learning and sending terms to 0

Machine learning implements this idea of sending some variables to 0 in the form of a systematic algorithm. For example, there is a machine learning algorithm for doing PCA. It adds a penalty for having too many nonzero terms; mathematically if a term is close to 0 then the penalty sends it to be zero. This is pretty much what we do in PCA when we send loadings below a threshold to 0. There are many algorithms in the machine learning literature that do this, for example, there is one based on absolute values, which they call an L1 norm but amounts to being a city block distance metric, and another based on euclidean values, which they call an L2 norm but amounts to being euclidean distance. There is even one that combines both L1 and L2 norms, called elastic net. There are versions of these penalty functions not only for PCA but for other statistical techniques including regression. In the application to regression, the logic is that βs close to 0 get sent to 0, so that all that remains are the key predictors (i.e., those βs not close to zero). This allows one to select predictors from an arbitrary large list of predictors. This technique of sending some terms to 0 (whether the terms are βs or factor weights or other statistical estimates) is also known as the Lasso technique and is based on sum of absolute values (L1); the version based on sum of squared values (L2) shrinks parameters rather than specifically sending some to 0. The package glmnet in R performs this type of sparsity for regression models.

A key limitation of current machine learning algorithms is that the estimates are biased and we don't understand their standard errors. We know they are biased because un-

biased estimates emerge when optimal solutions are computed, such as computing all the βs in a regression model. When some terms such as βs are sent to 0, then we no longer have unbiased estimates and it is difficult to estimate the standard error of biased estimates. We saw something similar to deal with multicolinearity (ridge regression is essentially this idea using an L2 norm) where we sacrificed unbiased parameters to get smaller standard errors; there is a connection between ridge regression and these more general approaches. One solution is to use permutation or resampling techniques like bootstrapping to estimate standard errors. Another solution is not to worry at all about confidence intervals and hypothesis testing and instead adopt a different criterion such as cross validation. The idea of cross validation, as we have seen before, is that one partitions a sample into "training" and "testing" parts, say 80% training and 20% testing. One estimates the model on the training subset and then uses the testing subset to see how well the model works (such as computing MSE, computing misclassification rate, etc). So cross validation evaluates models based on how well the model performs in out of sample prediction (where sample means the subset of data used to estimate the model).

regularization

Here is an example of sparsity using the sparsepca package in R. This process of shrinking small coefficients to 0 is also known as regularization or penalization depending on the literature you read. Another R package that you may want to look at is the more general regsem package.

```
library(sparsepca)
#example from help file
#create a data set with intentionally redundant variables
m <- 10000
V1 <- rnorm(m, 0, 290)
V2 <- rnorm(m, 0, 300)
V3 <- -0.1*V1 + 0.1*V2 + rnorm(m,0,100)
X <- cbind(V1=V1,V2=V1,V3=V1,V4=V1, V5=V2,V6=V2,
           V7=V2,V8=V2, V9= V3,V10=V3)
X <- X + matrix(rnorm(length(X),0,1), ncol = ncol(X),
                nrow = nrow(X))
# Compute SPCA
out <- spca(X, k=3, alpha=1e-3, beta=1e-3, center = TRUE,
            scale = FALSE, verbose=0)
print(out)


## Standard deviations:
## [1] 609.257 577.050 135.019
##
## Eigenvalues:
## [1] 371194.0 332986.8  18230.2
##
```

```
## Sparse loadings:
##          [,1]   [,2]   [,3]
##  [1,]  0.000 -0.498  0.000
##  [2,]  0.000 -0.498  0.000
##  [3,]  0.000 -0.498  0.000
##  [4,]  0.000 -0.498  0.000
##  [5,] -0.499  0.000  0.000
##  [6,] -0.499  0.000  0.000
##  [7,] -0.499  0.000  0.000
##  [8,] -0.499  0.000  0.000
##  [9,]  0.000  0.000 -0.675
## [10,]  0.000  0.000 -0.675
```

```
summary(out)
```

```
##                                PC1         PC2        PC3
## Explained variance      371193.991 332986.802 18230.197
## Standard deviations        609.257    577.050   135.019
## Proportion of variance       0.512      0.459     0.025
## Cumulative proportion        0.512      0.971     0.996
```

The output is relatively easy to interpret because many of the small terms were sent to 0. For comparison here is the same data analysed with a traditional PCA, centering the data as with the spca example and only printing out the first 3 components rounded to 3 decimal places. You can see the clutter of small coefficients got dropped but the values kept are slightly different (that's because the sparse version introduced bias).

```
out.pca <- prcomp(X, center = T, scale = F)
round(out.pca$sdev[1:3], 3)
```

```
## [1] 610.217 578.000 138.117
```

```
round(out.pca$rotation[, 1:3], 3)
```

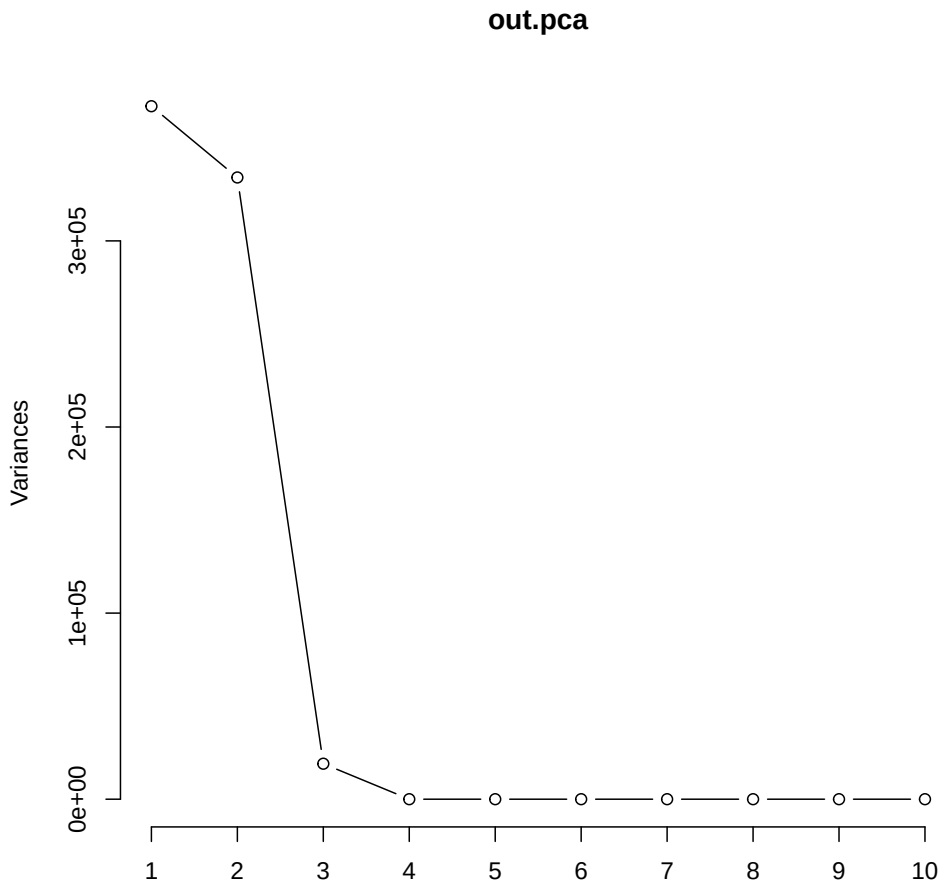
```
##        PC1    PC2    PC3
## V1   0.091 -0.490 -0.038
## V2   0.091 -0.490 -0.038
## V3   0.091 -0.490 -0.038
## V4   0.091 -0.490 -0.038
```

```
## V5  -0.490 -0.094  0.037
## V6  -0.490 -0.094  0.037
## V7  -0.490 -0.094  0.037
## V8  -0.490 -0.094  0.037
## V9  -0.061  0.043 -0.703
## V10 -0.061  0.043 -0.703
```

There are many different approaches for doing these types of analyses. The better ones also perform cross-validation to aid the decision of the tuning parameters, like the glmnet package for regression.

As a bonus, there is the screeplot for the full blown PCA, with a clear elbow at 3 components.

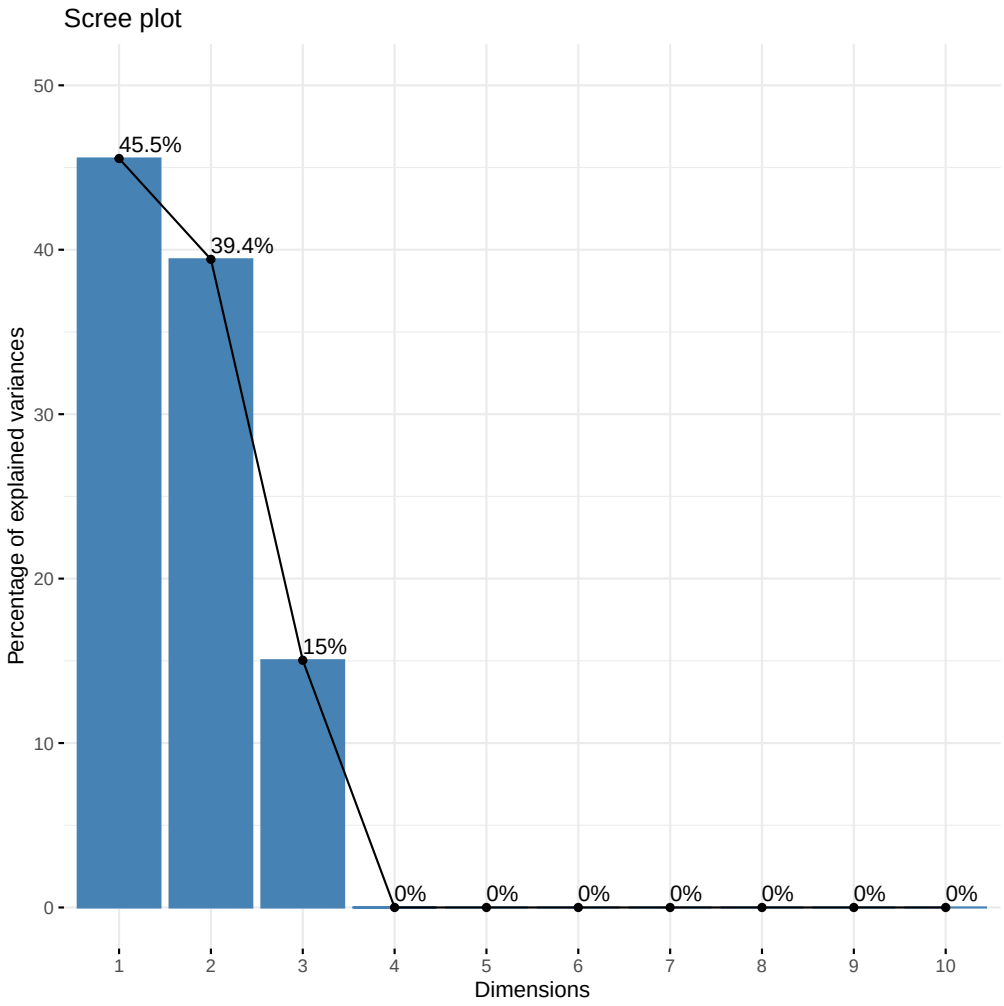
```
screeplot(out.pca, type = "lines")
```



The corrplot package provides some additional visualizations that works well with FactoMineR and factoextra (for more packages see the R appendix at the end of these notes).

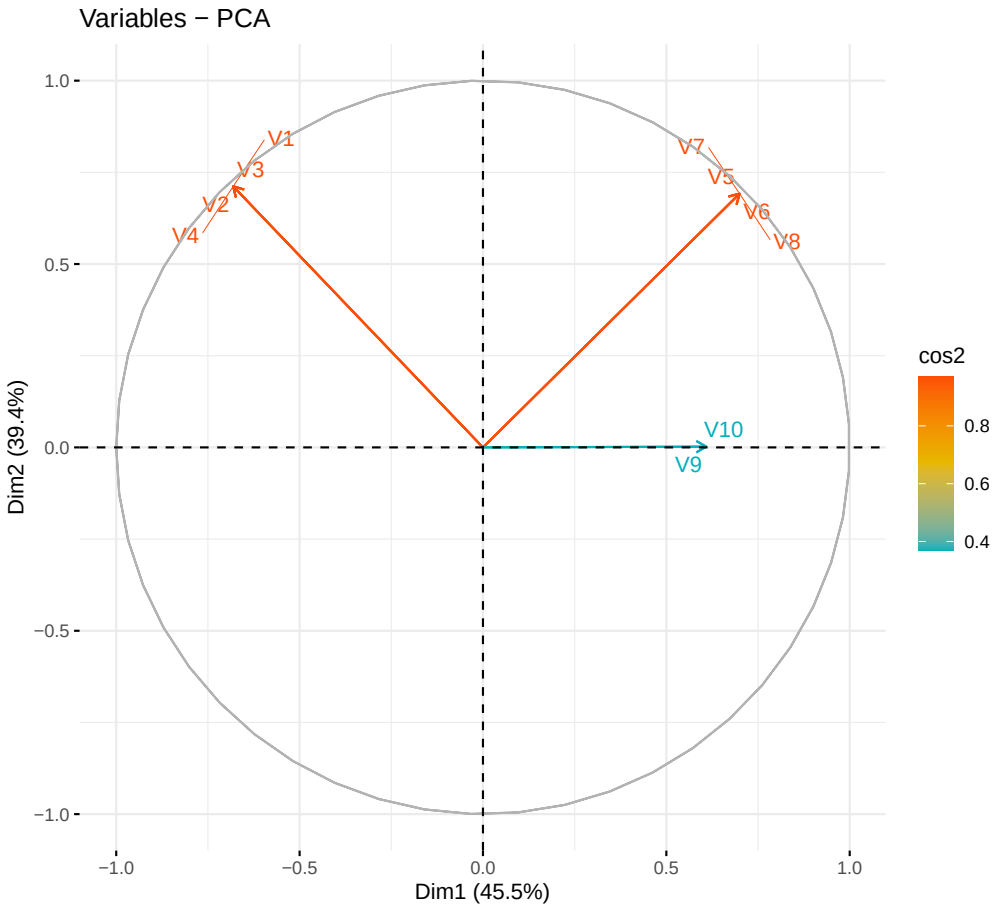
```
library(FactoMineR)
library(factoextra)
library(corrplot)
# run pca with 3 dimensions
out.PCA <- PCA(X, ncp = 3, graph = FALSE, scale.unit = T)
get_eigenvalue(out.PCA)


##          eigenvalue variance.percent
## Dim.1  4.554793e+00     4.554793e+01
## Dim.2  3.941711e+00     3.941711e+01
## Dim.3  1.503337e+00     1.503337e+01
```

```
## Dim.4  9.020575e-05     9.020575e-04
## Dim.5  1.211510e-05     1.211510e-04
## Dim.6  1.204231e-05     1.204231e-04
## Dim.7  1.180431e-05     1.180431e-04
## Dim.8  1.102894e-05     1.102894e-04
## Dim.9  1.093654e-05     1.093654e-04
## Dim.10 1.055812e-05     1.055812e-04
##        cumulative.variance.percent
## Dim.1                     45.54793
## Dim.2                     84.96504
## Dim.3                     99.99841
## Dim.4                     99.99932
## Dim.5                     99.99944
## Dim.6                     99.99956
## Dim.7                     99.99967
## Dim.8                     99.99979
## Dim.9                     99.99989
## Dim.10                   100.00000


# scree plot
fviz_eig(out.PCA, addlabels = TRUE, ylim = c(0, 50))
```



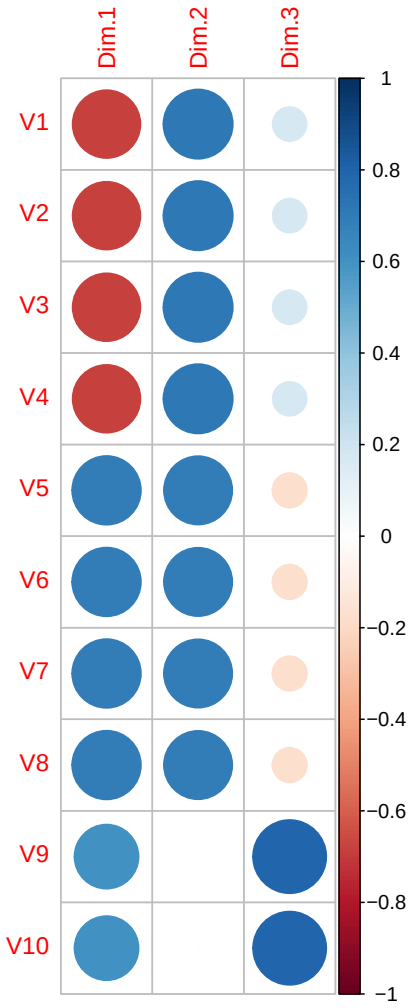
Creative plot of variables on two dimensional space based on the cos measure (discussed later). Note the clustering of 4 variables, 4 variables and 2 variables, which is exactly how this hypothetical data set was constructed (used for the sparsity example).

```
fviz_pca_var(out.PCA, col.var = "cos2", gradient.cols = c("#00AFBB",
    "#E7B800", "#FC4E07"), repel = TRUE)
```



And great looking plot of the contribution of each variable for each factor.

```
extra.PCA <- get_pca_var(out.PCA)
# plot of most contributing variables
corrplot(extra.PCA$cor, is.corr = T, cl.ratio = 0.4)
```



(i) PCA "assumptions"

- linearity must hold! Because PCA is based on correlations, your scatterplots should be adequate representations of your data, otherwise the factors as linear combinations of data wouldn't make much sense. Nonlinearities and outliers will distort the correlations and covariances, hence will distort PCA. If there nonlinearities, consider a nonparametric (e.g., ordinal) version of PCA.
- the correlation matrix is not "degenerate" (e.g., the observed matrix is not the identity matrix meaning all correlations are 0, or there aren't linearly dependent variables meaning some correlations are 1)
- linear combinations of the variables (i.e., the factors, aka components) must make sense (one shouldn't have combinations between "apples" and "oranges" that produce meaningless weighted sums)

(j) Test of significance in PCA

If you performed a PCA on the covariance matrix, you can create tests on the eigenvalues and build confidence intervals around them. Standard theory shows that under the usual assumptions (e.g., multivariate normal distribution) a sample eigenvalue $\hat{\lambda}$ has a normal distribution with mean equal to population λ and variance equal to $\frac{2\lambda^2}{N}$. This leads to the usual "estimate" and "standard error of the estimate" we saw last semester in Psychology 613. Thus, a confidence interval can be constructed by

$$\hat{\lambda} \quad \pm \quad 1.96\sqrt{\frac{2\lambda^2}{N}} \tag{11-4}$$

You can also construct a test comparing the reproduced matrix to the observed matrix. The test is distributed as χ^2 and you need to compare your computed test statistic to the value in the χ^2 table. Here is the test statistic with definition of terms coming right after:

$$\chi^2 \quad \sim \quad -(N - \frac{2P}{3} - \frac{M}{3} - \frac{11}{6}) \ln \frac{|\hat{R}|}{|R|} \tag{11-5}$$

where N is the number of subjects, P is the number of principal components requested, M is the number of variables in the covariance matrix, $\hat{R}$ is the reproduced covariance matrix, R is the observed covariance matrix, and in this equation the vertical bars refer to the determinant of the matrix embedded between the vertical bars. I won't explain the determinant here except to point out that the determinant is identical to the product of all the eigenvalues (e.g., if there the matrix is 10×10, then the determinant is identical to the product of all 10 eigenvalues)[9]. The degrees of freedom for this χ^2 test are ((P - M)^2 - M - P)/2. I don't think many programs print out tests in the context of principal components analysis, so if you want these tests you'll have to do them by hand (or write you own code in R).

Analogous tests for PCA on a correlation matrix exist but the formulas are much longer to write out because they have to take into account the standardization (i.e., change of scale).

(k) Rotations

A rotation can be defined as a matrix operation (see the linear algebra appendix for details). Denoting the rotation matrix T and the initial factor matrix L, we define a new

[9] In multivariate statistics programs you may sometimes encounter an error that says "matrix not positive definite". Essentially, that means that the determinant is zero or negative, which means that it is not possible to invert the matrix and that at least one eigenvalue is zero or negative. Causes of this could be that some of the variables are linearly dependent, you used the "pairwise method" for dealing with missing data, or perhaps a variable is a constant. For a deeper discussion of this and related problems see Wothke's (1993) chapter in Bollen & Long, *Testing Structural Equations Models.*

rotated factor matrix by the matrix multiplication LT. Note that the communalities are unaffected by the choice of transformation. That is, the communalities are the diagonal elements of the matrix LL' and are identical to the diagonal elements of $L^*L^{*\prime}$, where the prime denotes transpose and $*$ denotes a rotated space. Communalities can be interpreted as the length of a vector, and length is unaffected by rotation.

Thurstone suggested several criteria (or desiderata) for rotations. One was that variables should load onto some factors and not others (i.e., leading to entries of 0 in each column of the factor matrix). One trick to accomplish this goal is to compute the variance of the communalities, and then perform an ANOVA-like decomposition on this variance into something akin to between and within sums of squares. The *quartimax* rotation maximizes the sum of squares of all elements in the factor matrix (i.e., rotate to get numbers as far apart as possible, thus increasing the sum of squares). Quartimax tends to find a single "all purpose" first factor on which all variables load (something analogous to a unit vector). Some people don't like this because it goes against one of Thurstone's desiderata that variables should load on some factors and not others. All variables will likely load highly on the single common factor.

Kaiser suggested that it may be better to work with each column of the factor matrix; that is, define a rotation that maximizes the sum of squares loading for each column of the factor matrix. Further, he suggested that before maximizing each column should be normalized by dividing each row of the factor matrix by the square root of the communality for that variable. This is the technique known today as *varimax* (which can be either raw or normalized depending on whether or not the factor matrix was first normalized). This is the default rotation in SPSS.

(l) PCA is useful in dealing with multicolinearity in regression. Recall that predictors that are highly correlated create problems for the standard errors of regression slopes. If your analysis problem has predictors that are highly correlated, you might be able to reduce the predictors down to a smaller set of uncorrelated factors. Simply run a PCA prior to the regression to create factors, and then use the factors as predictors instead of the larger set of correlated predictors. Later, we'll discuss a more general technique that allows you to create factors and run regressions simultaneously (rather than two steps).

(m) Consensus Analysis

One interesting use of PCA is what is called "consensus analysis." Take the track and country example I used earlier in this notes. I showed how to analyze it by correlating the 8 events with each other, which gave us an understanding of how records for the events relate to each other. Another analysis would be to correlate each country with each other, so take the 8 records for country i and correlate with the 8 records for country j. Do that for all countries i and j. This generates a correlation matrix measuring how

similar each country is to every other country. Now do a PCA on that matrix. If the first component accounts for a large percentage of the variance, then you can use the loadings of each country on the first component as a type of "aggregate score." Countries that have fast times across all 8 events are faster "across the board" than countries that have lower loadings.

Anthropologists use this type of analysis when assessing cultural competence of informants. Ask several informants the same set of questions. Correlate across informants to create an informant by informant correlation matrix (just like the country by country correlation matrix above for the track example) and do a PCA on that matrix. Informants that have loadings on the first component "know," they are experts about their own culture and can be weighted more than informants with lower loadings. I'm just giving a basic outline of the technique, which for our purposes is simply a PCA on a correlation matrix.

(n) Using PCA in Natural Language Processing

PCA is also used in Natural Language Processing (NLP) approaches. To provide an illustration, I will use the first nine lecture notes in this class as the corpus (the set of text documents) that I will analyze. I won't be able to cover all the details involved in NLP. Typically, one has many documents or artificially breaks up a document into smaller chunks such as paragraphs. I'll simply use the first 9 lecture notes as the 9 documents to analyze. I won't do much cleaning beyond some basic work such as removing punctuation and removing stopwords; I won't perform additional cleaning for words that don't provide much information such as "lecture" and "notes" (the phrase "lecture notes" appears often in those nine documents).

```
#several packages I found useful for nlp
library(ggplot2)
library(dplyr)
library(tm)
library(pdftools)
library(lsa)
library(LSAfun)
library(topicmodels)
library(tidytext)
library(tidyr)
library(qdap)
#install.packages("BiocManager")
#BiocManager::install("Rgraphviz")
library(Rgraphviz)
library(igraph)
library(ggraph)
```

```
#read the lecture notes
files <- list.files(path="./filecopy/",pattern = "pdf$",
                    full.names=T)

#just use ln1-9 (ln 10-13 appear in 2nd-5th position)
files <- files[-1*c(2,3,4,5)]
files


## [1] "./filecopy//file1.pdf" "./filecopy//file2.pdf"
## [3] "./filecopy//file3.pdf" "./filecopy//file4.pdf"
## [5] "./filecopy//file5.pdf" "./filecopy//file6.pdf"
## [7] "./filecopy//file7.pdf" "./filecopy//file8.pdf"
## [9] "./filecopy//file9.pdf"


#define a corpus with all 9 documents
corp <- Corpus(URISource(files),
    readerControl = list(reader = readPDF))

#some initial cleaning such as
#remove punctuation and stopwords, stem words, etc
corp <- tm_map(corp, removePunctuation, ucp = TRUE)
rich.tdm <- TermDocumentMatrix(corp,
           control =
           list(stopwords = TRUE,
           tolower = TRUE,
           stemming = TRUE,
           removeNumbers = TRUE,
           bounds = list(global = c(3, Inf)))
           )
rich.tdm.m <- as.matrix(rich.tdm)
head(rich.tdm.m)


##             Docs
## Terms        file1.pdf file2.pdf file3.pdf file4.pdf
##   |tobserv           2         0         1         0
##   abil               2         1         0         3
##   abl                1         1         2         2
##   absolut            4         0         4         2
##   accept             0         1         1         0
##   accomplish         2         0         0         3
##             Docs
## Terms        file5.pdf file6.pdf file7.pdf file8.pdf
```

```
##   |tobserv           1         3         0         0
##   abil               1         1         1         1
##   abl                3         3         2         4
##   absolut            0         3         0         0
##   accept             0         0         0         0
##   accomplish         6         1         0         1
##              Docs
## Terms         file9.pdf
##   |tobserv           0
##   abil               6
##   abl                2
##   absolut            1
##   accept             2
##   accomplish         6
```

Now that the term by document matrix is set up in rich.tdm (number of times each word appears in each document) we can begin to perform some computations. The latent semantic analysis is simply a singular value decomposition on this rectangular matrix as I show below.

```
notes_lsa = lsa(rich.tdm.m, d = 2)
head(as.textmatrix(notes_lsa))
```

```
##             file1.pdf file2.pdf file3.pdf file4.pdf
## |tobserv   0.5131933 0.4466072 0.6188010 0.6804870
## abil       1.2073140 1.0868097 0.7133234 1.1635235
## abl        1.6378048 1.4555774 1.3529335 1.8053506
## absolut    0.9322933 0.7555225 2.2705188 1.9115162
## accept     0.2765614 0.2393307 0.3611498 0.3830199
## accomplish 1.7519037 1.5111125 2.3894095 2.4861690
##             file5.pdf file6.pdf file7.pdf file8.pdf
## |tobserv    1.175874 0.6383225 0.4577307 0.9359033
## abil        2.411989 1.8795895 1.4035206 2.7957819
## abl         3.455889 2.3536944 1.7344513 3.4844261
## absolut     2.683240 0.5761036 0.3271140 0.7830038
## accept      0.646890 0.3299068 0.2344946 0.4822177
## accomplish  4.146306 2.0380731 1.4406895 2.9733042
##             file9.pdf
## |tobserv   0.7427374
## abil       2.0809279
## abl        2.6498132
## absolut    0.8341957
## accept     0.3878278
```

```
## accomplish 2.4110459
```

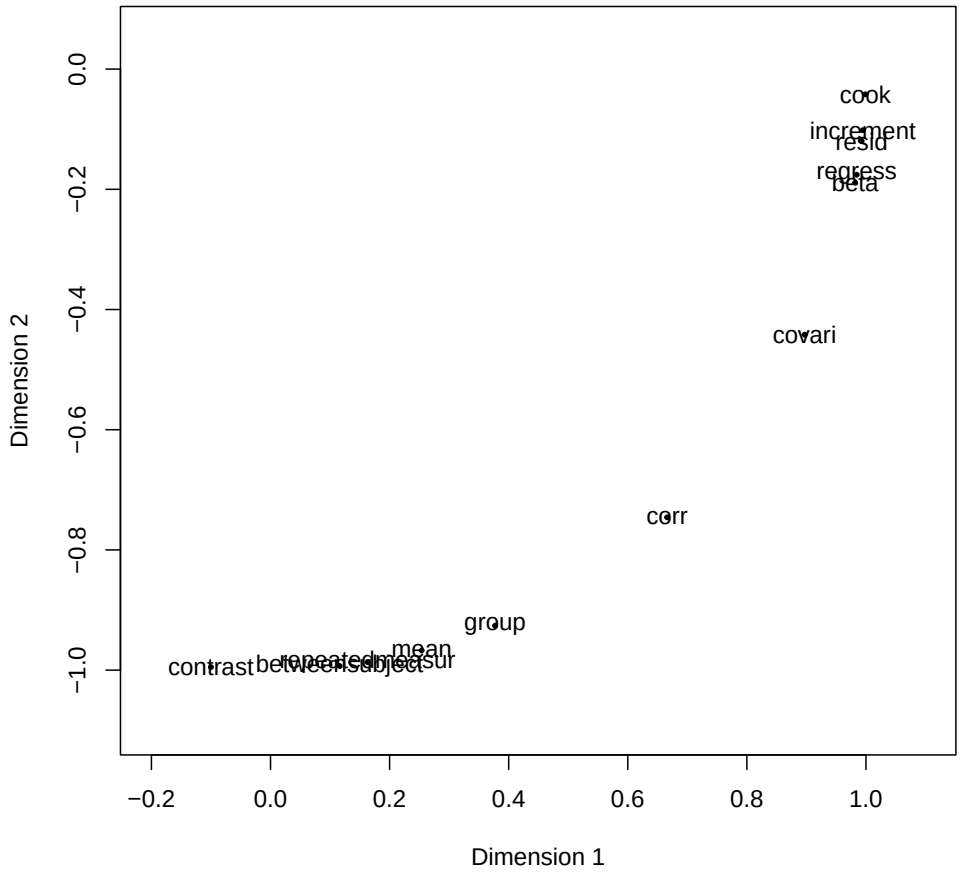
This is the predicted term by document matrix using 2 dimensions derived from the eigenvalues and eigenvectors. To see this look at the eigenvalues and eigenvectors from the LSA output below and compare them to the ones that are manually computed with the svd() function—they are identical.

```
# eigen vals are 3rd element of the list
notes_lsa[[3]]


## [1] 1675.8738  604.0422


# compare to computing eigen vals directly
svd.out <- svd(rich.tdm.m)

# first two eigenvals
svd.out$d[1:2]


## [1] 1675.8738  604.0422


# first two eigenvecs
notes_lsa[[2]]


##                 [,1]        [,2]
## file1.pdf -0.2282154  0.05947430
## file2.pdf -0.1909853  0.07945768
## file3.pdf -0.4316960 -0.49729401
## file4.pdf -0.3948119 -0.25633027
## file5.pdf -0.5976025 -0.13527596
## file6.pdf -0.2041924  0.36360131
## file7.pdf -0.1346812  0.30341932
## file8.pdf -0.2909646  0.56372177
## file9.pdf -0.2599651  0.34174806


# compare to manual computation
svd.out$v[, 1:2]


##             [,1]        [,2]
```

```
##  [1,] -0.2282154  0.05947430
##  [2,] -0.1909853  0.07945768
##  [3,] -0.4316960 -0.49729401
##  [4,] -0.3948119 -0.25633027
##  [5,] -0.5976025 -0.13527596
##  [6,] -0.2041924  0.36360131
##  [7,] -0.1346812  0.30341932
##  [8,] -0.2909646  0.56372177
##  [9,] -0.2599651  0.34174806
```

One can play with this output and examine specific sets of words such as in this example where I pick out words that are likely to appear in ANOVA or regression notes and then plot those words on the two dimensions resulting from the LSA.

```
#pick some key words that reflect ANOVA or regression topics
wordlist = c("regress", "betweensubject", "group", "mean",
             "corr","resid","increment","cook","contrast",
             "repeatedmeasur","beta","covari")

#plot all the words selected
plot_wordlist(wordlist,
    method = "PCA",
    dims = 2,
    tvectors = as.textmatrix(notes_lsa))
```




```
##                           x          y
## regress           0.9844507 -0.1756611
## betweensubject    0.1162397 -0.9932212
## group             0.3772983 -0.9260918
## mean              0.2538942 -0.9672320
## corr              0.6661364 -0.7458299
## resid             0.9928202 -0.1196159
## increment         0.9948205 -0.1016473
## cook              0.9991200 -0.0419425
## contrast         -0.1000421 -0.9949832
## repeatedmeasur    0.1630334 -0.9866205
## beta              0.9819671 -0.1890517
## covari            0.8970655 -0.4418976
```

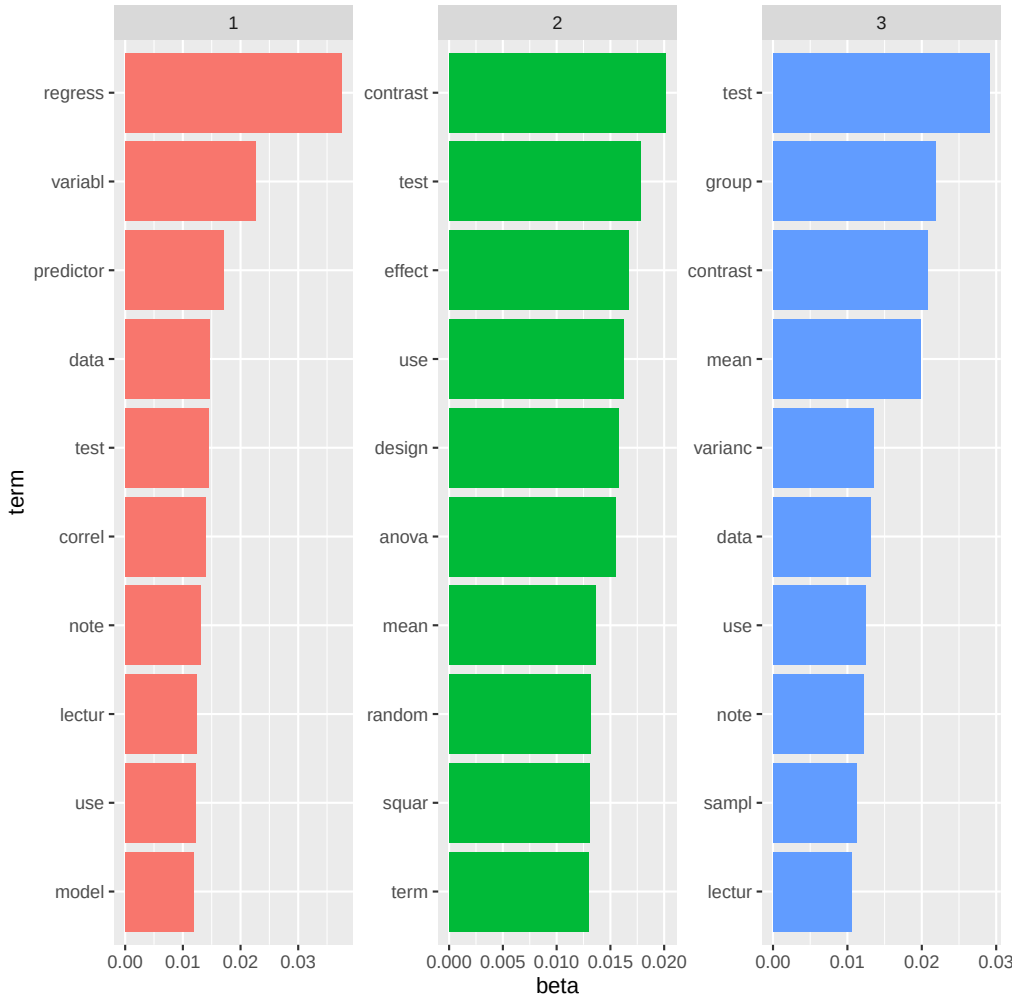
This plot suggests that the two dimensions may be overfitting the data because ANOVA

vs regression appears as a circular pattern (recall LN10 that suggested if there is one dimension and the analyst requests two dimensions, then a circular pattern may emerge). There could be additional processing on this output, such as considering rotations of the LSA solution to clarify the interpretation. So, PCA can be used to analyze text.

An alternative approach uses something closer to the clustering and trees approach introduced in Lecture Notes 10, and is called Latent Dirichlet Allocation (LDA). This approach models a word as a mixture of different latent topics. To illustrate, I'll perform an LDA on three topics.

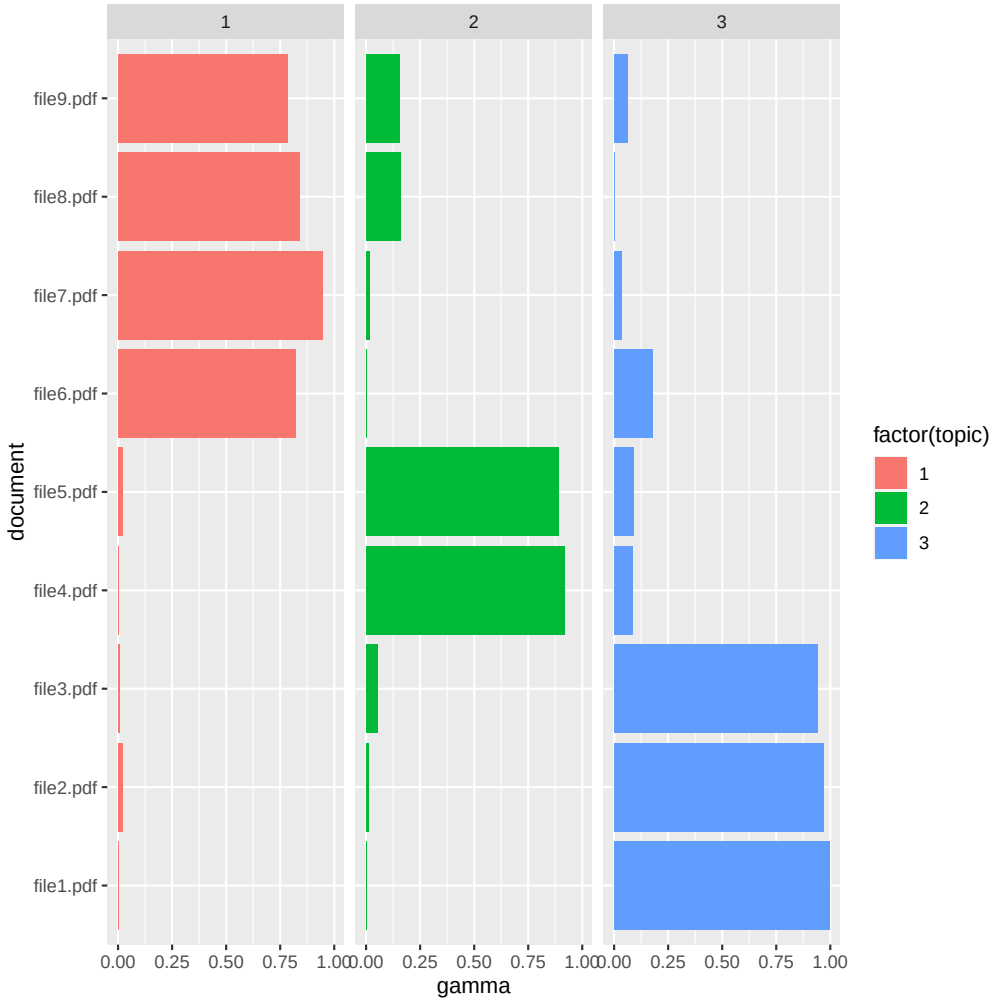
```
rg_lda <- LDA(as.dtm(rich.tdm), k = 3,
              control = list(seed = 1234))
rg_lda


## A LDA_VEM topic model with 3 topics.


ap_topics <- tidy(rg_lda, matrix = "beta")
ap_topics


## # A tibble: 4,287 x 3
##    topic term            beta
##    <int> <chr>          <dbl>
##  1     1 |tobserv 0.000115
##  2     2 |tobserv 0.000000291
##  3     3 |tobserv 0.000178
##  4     1 abil     0.000398
##  5     2 abil     0.000170
##  6     3 abil     0.000132
##  7     1 abl      0.000500
##  8     2 abl      0.000202
##  9     3 abl      0.000173
## 10     1 absolut  0.000115
## # i 4,277 more rows


ap_top_terms <- ap_topics %>%
  group_by(topic) %>%
  slice_max(beta, n = 10) %>%
  ungroup() %>%
  arrange(topic, -beta)

ap_top_terms %>%
```

```
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered()
```




```
beta_wide <- ap_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  pivot_wider(names_from = topic, values_from = beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(logratio = log2(topic2 / topic1))


ap_documents <- tidy(rg_lda, matrix = "gamma")
ap_documents
```

```
## # A tibble: 27 x 3
##    document  topic     gamma
##    <chr>     <int>     <dbl>
##  1 file1.pdf     1 0.00334
##  2 file2.pdf     1 0.0189
##  3 file3.pdf     1 0.00544
##  4 file4.pdf     1 0.0000204
##  5 file5.pdf     1 0.0216
##  6 file6.pdf     1 0.821
##  7 file7.pdf     1 0.947
##  8 file8.pdf     1 0.842
##  9 file9.pdf     1 0.784
## 10 file1.pdf     2 0.0000291
## # i 17 more rows


ap_documents %>%
  ggplot(aes(x=document,y=gamma,fill=factor(topic))) +
  geom_bar(stat="identity") +
  coord_flip() + facet_wrap(~factor(topic))
```



As with any of the methods covered in these two sets of lecture notes, we have to interpret the dimensions or components or clusters. One can use similar techniques with additional data to aid in the interpretation, or even test the interpretation.

For completeness, here is a way to compute a distance matrix manually on the (potentially sparse) matrix of terms by document (i.e., rows are the terms used, columns are the documents, so a cell refers to the number of times that term appeared in that document). I first convert the tdm matrix into a similarity matrix, then transform it to a distance matrix, ensuring the distance matrix has 0s in the diagonal and perform hierarchical clustering on that distance matrix. Interesting that the solution places LN7 as an outlier. The first two LN form their own pair, then the three remaining ANOVA lecture notes (3, 4, and 5). LN8 and LN9 have some ANOVA material so they are closer to LN3-5, than LN6 that presents regression as a new topic in the course.

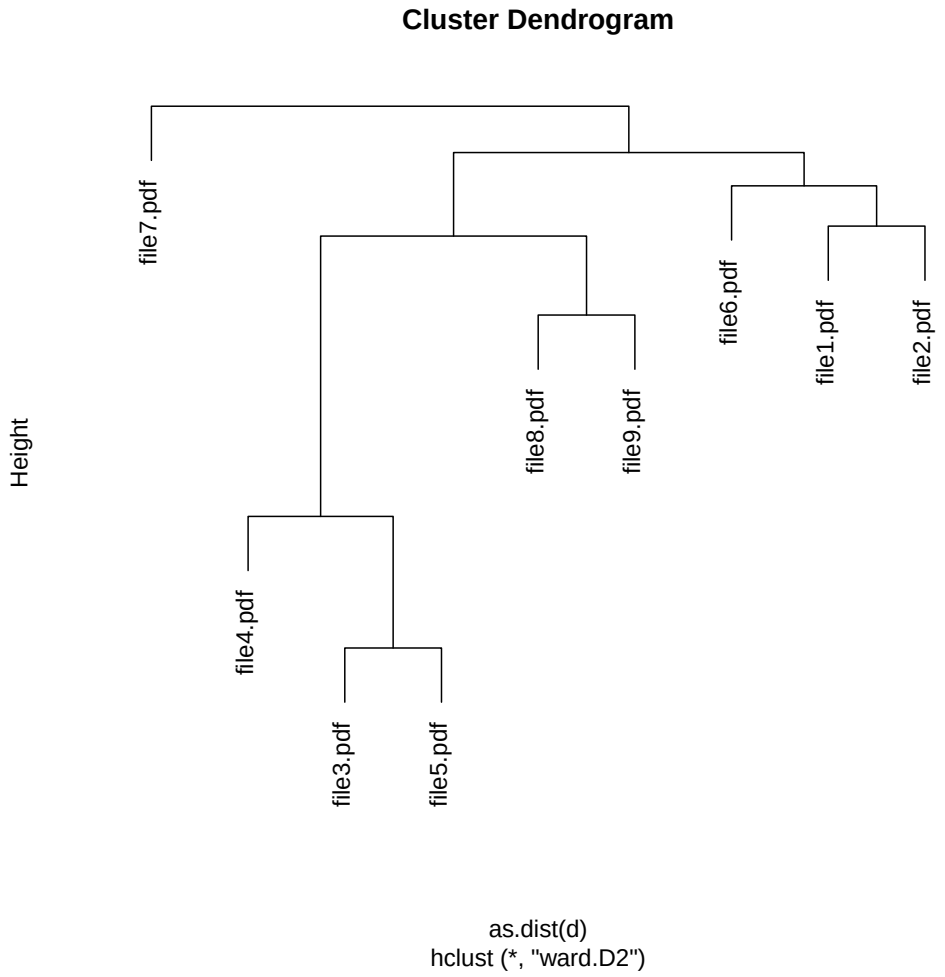
```
rich.tdm.matrix <- as.matrix(rich.tdm)
head(rich.tdm.matrix)
```

```
##             Docs
## Terms        file1.pdf file2.pdf file3.pdf file4.pdf
##   |tobserv           2         0         1         0
##   abil               2         1         0         3
##   abl                1         1         2         2
##   absolut            4         0         4         2
##   accept             0         1         1         0
##   accomplish         2         0         0         3
##             Docs
## Terms        file5.pdf file6.pdf file7.pdf file8.pdf
##   |tobserv           1         3         0         0
##   abil               1         1         1         1
##   abl                3         3         2         4
##   absolut            0         3         0         0
##   accept             0         0         0         0
##   accomplish         6         1         0         1
##             Docs
## Terms        file9.pdf
##   |tobserv           0
##   abil               6
##   abl                2
##   absolut            1
##   accept             2
##   accomplish         6
```

```
#might be good to remove sparse terms with something like
#out <- removeSparseTerms(rich.tdm, sparse = 0.975)

sim <- t(rich.tdm.matrix) %*% rich.tdm.matrix
#d <- 1/sim * 1000000

d <- max(sim) - sim
diag(d) <- 0
round(d,2)
```

```
##            Docs
## Docs        file1.pdf file2.pdf file3.pdf file4.pdf
##   file1.pdf         0    986259    865533    952313
##   file2.pdf    986259         0    942334    959028
```

```
##   file3.pdf    865533    942334         0    718790
##   file4.pdf    952313    959028    718790         0
##   file5.pdf    817891    870054    510214    514109
##   file6.pdf   1000231   1040419    948430    976158
##   file7.pdf   1067887   1074484   1024959   1035125
##   file8.pdf    997609   1011300    881294    874275
##   file9.pdf    977924   1013050    885815    916539
##            Docs
## Docs        file5.pdf file6.pdf file7.pdf file8.pdf
##   file1.pdf    817891   1000231   1067887    997609
##   file2.pdf    870054   1040419   1074484   1011300
##   file3.pdf    510214    948430   1024959    881294
##   file4.pdf    514109    976158   1035125    874275
##   file5.pdf         0    860161    972890    717241
##   file6.pdf    860161         0   1037941    940145
##   file7.pdf    972890   1037941         0    982441
##   file8.pdf    717241    940145    982441         0
##   file9.pdf    753561    971609   1029244    885897
##            Docs
## Docs        file9.pdf
##   file1.pdf    977924
##   file2.pdf   1013050
##   file3.pdf    885815
##   file4.pdf    916539
##   file5.pdf    753561
##   file6.pdf    971609
##   file7.pdf   1029244
##   file8.pdf    885897
##   file9.pdf         0


hc <- hclust( as.dist(d), method ="ward.D2")
plot( hc, yaxt ='n')
```



Of course, there are many more details to address and analytics that can be performed, but this presentation is meant mostly to show the connection to the material in LN10 and 11 (distances, PCA, clustering). If you plan to do NLP in your research, I recommend working through a dedicated textbook or taking a semester long class.

3. Independent Components Analysis (ICA)

ICA is an extension of PCA that is gaining popularity in signal processing and multivariate time series analyses. The basic idea of the model is that it assumes the observed signals X are composed of unknown sources S that are "mixed" or summed together. To get the intuition of ICA an analogy is helpful. Imagine you are at a wedding reception. There are sounds coming from many sources: the band through the speakers, the sounds on the dance floor, a heated family discussion at one of the tables, etc. If you put microphones at different locations of the room, each microphone will pick up the sounds originating from close sources as well as

background originating from sources relatively further away. So, each microphone is picking up a mixture of sounds. ICA solves the problem of "unmixing" the signals to isolate primarily the nearby sounds (e.g., the signal coming from the microphone near the family discussion would be purged of the sounds from the dance floor, the band and other sounds; the signal coming from the microphone near the dance floor would be purged of the sounds from the family discussion; etc.).

To sketch out the model in symbols. We observe matrix X (such as a time by microphone matrix since each microphone produces a temporal waveform) and model estimates a decomposition

$$X = WS$$

where S is the matrix that contains the original uncontaminated sources and W is the mixing matrix. So what we observe, X, is a result of mixing the original sources S. The problem the algorithm solves is finding both unknown W and unknown S. If we can figure out W, then we can apply the inverse of W to the observed X to recover the unknown S.

To make this work, ICA makes the assumption that the components are independent and not normally distributed. The reason we don't want to assume normality is that normal distributions do not have tall spikes or long tails, and we, essentially, need that information in order to pull off this "unmixing" wonder. Normal distributions don't have kurtosis, and ICA makes use of kurtosis. The framework of ICA extends PCA because it uses the singular value decomposition to estimate the "mixing" matrix W, thus providing a way to compute the source matrix S with the originating waveforms..

I'll present an interesting visual example where I take two images (two source matrices S), create new "mixed up" images (analogous to the microphones that pick up primarily one signal but also some extra background sound) by applying matrix W to the S matrices. First, let's get two images loaded up into R. These will play the role of the true sources.

```
library(fastICA)
# adapted from http://www.di.fc.ul.pt/~jpn/r/ica/index.html
# install bioconductor and EBImage; uncomment next lines if
# (!requireNamespace('BiocManager', quietly = TRUE))
# install.packages('BiocManager') BiocManager::install()
# BiocManager::install(c('EBImage'))
library("EBImage")

# some vacation pictures from Austria the house from the
# Sound of Music
S1 <- readImage("austria1.jpg")
S2 <- readImage("austria2.jpg")
# make image size smaller for speed
```

```
S1 <- resize(S1, w = dim(S1)[1]/5, filter = "bilinear")
S2 <- resize(S2, w = dim(S2)[1]/5, filter = "bilinear")
# convert to grayscale
S1 <- channel(S1, "gray")
S2 <- channel(S2, "gray")
# add back the 3rd dim in array
dim(S1) <- c(dim(S1), 1)
dim(S2) <- c(dim(S2), 1)

par(mfrow = c(1, 2))
display(normalize(S1), method = "raster", frame = 1)
display(normalize(S2), method = "raster", frame = 1)
mtext("Two Pictures: Originals", outer = T, line = -5, side = 3)
```

Two Pictures: Originals

Now I'll mix those two pictures to create two new ones that are composites of both pictures. I will treat these two new mixed images as the signals we observe (i.e., the analogue of the microphone that picks up sound from multiple sources). We will pretend that we don't know the original photos (nor the mixing matrix) and will test whether we can recover the original images from these composite images. In the mixed images, note how the clouds from one of the original pictures have been added to the other photo, the tree branches in the foreground of one of the original pictures has been added to the other, etc. Essentially, both pictures are distorted combinations of the two originals.

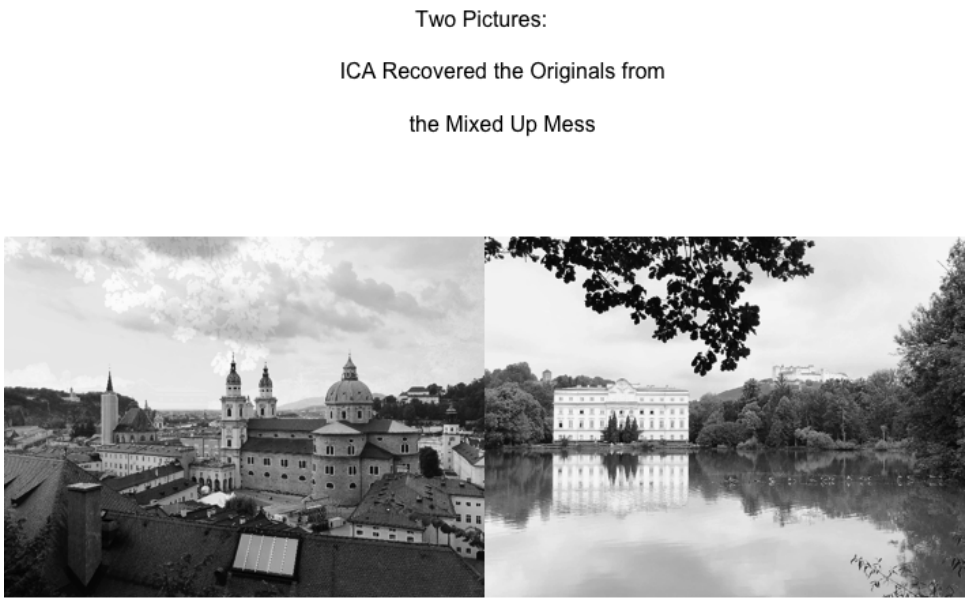
```
# mix both
W <- matrix(c(0.8, 0.2, 0.5, 0.67), ncol = 2, byrow = TRUE)

X1 <- W[1, 1] * S1 + W[1, 2] * S2
X2 <- W[2, 1] * S1 + W[2, 2] * S2
par(mfrow = c(1, 2))
display(normalize(X1), method = "raster", frame = 1)
display(normalize(X2), method = "raster", frame = 1)
mtext("Two Pictures: Mixtures of Originals", outer = T, line = -5,
    side = 3)
```

Two Pictures: Mixtures of Originals

Finally, the ICA analysis. We can "unmix" these two images using ICA. I'll create a new data matrix with both pictures in the same matrix, and run an ICA with two components. I need to change the dimensions of the output of the ICA so that the output can display correctly. I do not make use of original S nor the mixing matrix W in what I do below; I am acting as the only two pieces of information I have are the two pictures that have been "mixed up."

```
data <- cbind(X1, X2)
model <- fastICA(data, n.comp = 2, method = "C")

# fix dimensions of ICA output separately for each
# component
S1_hat <- model$S[, 1]
dim(S1_hat) <- dim(X1)
```

```
S2_hat <- model$S[, 2]
dim(S2_hat) <- dim(X2)

# sometimes the negative comes out of ICA so need to
# rescale e.g., S2_hat <- max(S2_hat) - S2_hat
par(mfrow = c(1, 2))
display(normalize(S1_hat), method = "raster", frame = 1)
display(normalize(S2_hat), method = "raster", frame = 1)
mtext("Two Pictures: \n
      ICA Recovered the Originals from\n
      the Mixed Up Mess",
    outer = T, line = -5, side = 3)
```



You can find more examples on the web, including demonstrations with sounds (the microphones example). I think we'll see more applications of ICA in the next few years as people learn how to use it. Various applications of ICA include removing the noise due to eye blinks from EEG data.

A key extension of ICA and other methods presented so far in these lecture notes and in

lecture notes 10 is that we need to have a model of error (that is, the "plus epsilon" part of the model we saw in ANOVA and regression). We now transition into methods that begin to add a model of error to the ideas developed in LN 10 and LN-11.

We can use the intuition of ICA as unmixing a set of observed data into underlying unobserved signals to get a handle of what needs to be done in handling the "plus ϵ" idea, In the analogous way that we partition sum of squares into the systematic part (SS between or SS regression) and the error part (sum of squares error or sum of squares residual, for ANOVA and regression respectively), we can imagine using PCA-like procedures to "unmix" (analogous to what I demonstrated ICA is doing) a covariance matrix into the systematic part (the underlying signals we care about) and the error part. This will parallel the structural model ideas we developed last semester, and will serve as a way to connect the structural model to the MDS/PCA material. We turn to factor analysis as one approach to adding a "plus epsilon" to this framework, thereby decomposing an observed proximity matrix into a systematic part and an error (epsilon) part.

4. Factor Analysis

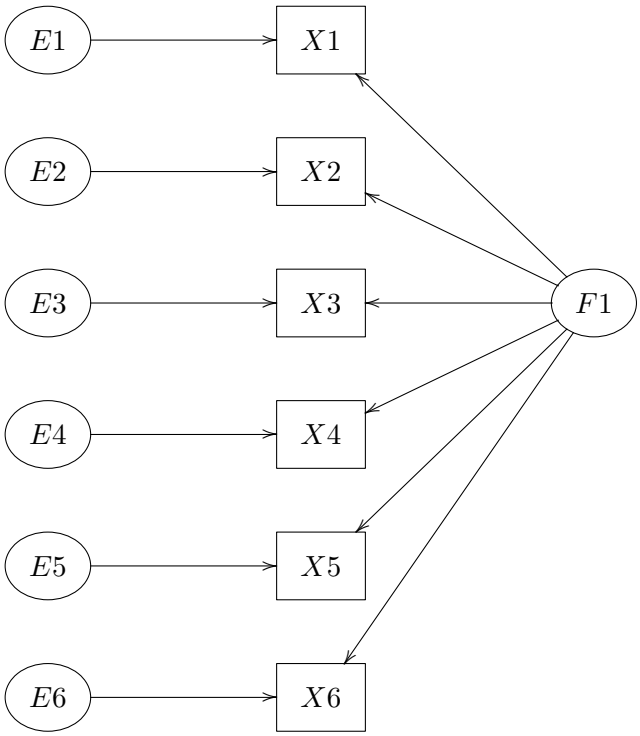
A limitation of principal components analysis is that it doesn't handle error (i.e., there was no ϵ term in our PCA discussions). We need a way to separate out variance that is due to a common factor from variance that is unique to the specific variable from error variance. We can represent this graphically by adding error terms to the model we presented earlier for PCA (see Figure 11-12)

One trick is to put *communality estimates* in the diagonals of the correlation matrix rather than 1. The trick forms the basis of factor analysis (FA), which attempts to deal with error in measurement. This particular form of FA is now outdated because structural equation modelling (SEM) can do a better job of dealing with error. In any case, factor analysis has historical interest because it recognized that the limitation of PCA was that it did not account for error. Also, rotations are not possible in SEM but one can perform rotations in FA.

A major note: researchers in this literature get fussy about using the word "factor" to apply only to Factor Analysis and not other methods such as PCA. I use the word factor rather loosely and synonymously with other terms like components. For me, it is the model that carries the key information like whether there are error terms as depicted in Figure 11-12. Since many of these concepts are related to each other and in some situations are special cases of more general concepts, I am not as fussy as many of my colleagues. Also, that the same program can be used to estimate most of these things (PCA, exploratory FA, confirmatory FA, etc) just adds to the confusion among users that these may not be as distinct as some methodologists would prefer them to be.

(a) The Big Debate in FA: What To Put In The Diagonal? Some suggest putting the

Figure 11-12: Representation of a single factor (F1), its corresponding observed variables (X1 to X6), and the errors.



reliability estimates of each variable in the diagonal of the correlation matrix (rather than the 1), some suggest putting the R^2 of the regression with that variable as the DV and all other variables as the IV. It is possible to show that these two correspond to lower and upper bounds, respectively, on the true communality (see Harris, *Primer to Multivariate Statistics*).

Regardless of what is placed in the diagonal, the sum of the diagonals will be less than the number of variables, as in PCA because numbers less than one are being placed in the diagonal. Further, the goal of FA is to estimate the diagonals in a manner that minimizes the number of nonzero eigenvalues. In PCA no eigenvalues can be zero (unless the variables were linearly independent).

In the FA framework, communality is the percentage of variance in a variable explained by the common factor. Denoting communality for variable i as h_i^2 we have

$$1 = h_i^2 + u^2 \tag{11-6}$$

where u^2 is the variance of everything else. The term u^2 is usually decomposed further into two parts:

$$u^2 = s^2 + c^2 \tag{11-7}$$

where s^2 is specific variance estimated as (reliability - communality), and error variance c^2 is estimated as 1 - reliability. Note that in this framework, reliability is a sum of variance explained by both common and specific factors. Reliability may be measured in several ways (e.g., directly in the form of test-retest, in the form of internal consistency). More on reliability when I discuss structural equations modeling later in the term.

The structural model for FA is that each observed variable i is a weighted sum of common factor scores plus a unique factor score associated with that variable:

$$X_i = \beta_1 F_1 + \beta_2 F_2 + \ldots + \beta_k F_k + \epsilon_i \tag{11-8}$$

where F_k are the common factors, ϵ_i is the usual error term, and there is no intercept because the variables are assumed centered.

Thus, there is key conceptual difference between PCA and FA. The latter tries to separate factor variance from specific and error variance. PCA lumps specific and error variance into factor variance. The number of factors may be part of the estimation, depending on the method of extraction.

One needs to be careful when interpreting the eigenvectors when communalities other than one are used. The reason is that the trace of the correlation matrix (sum of the diagonals) will be less than the number of variables. See Harris' *Primer of Multivariate*

Statistics for a more complete discussion of the complexities of factor analysis. Further, the creation of factor scores is tricky in FA because the presence of unique and common factors creates an indeterminacy problem, hence the various methods of computing factor scores in SPSS (see Grice, 2001, *Psychological Methods*, for a review).

5. Misc issues in PCA & FA

(a) Factor extraction

There are many different types of factor extraction methods. Some are based on maximum likelihood methods and some people have argued that those techniques are on stronger statistical footing than the traditional PCA I presented here. There are other procedures that have "funny" properties, such as principal axis factoring, which permits communalities to be greater than 1. But all of this is beyond the scope of this course and I don't want to dwell on FA given that there are other more modern techniques available. In Lecture Notes 13, we'll move on to a related technique, structural equations modelling (SEM), that is better behaved than both FA and PCA. SEM many techniques as special cases including FA, PCA, regression, path analysis, and ANOVA. SEM also serves as a foundation for some interesting generalizations, such as merging (a) clustering of variables and/or subjects simultaneous to (b) fitting of factors, testing correlation matrices across different groups, estimation of latent growth curves, and many more techniques.

(b) Number of factors

Much research has now shown that looking at screeplots or using the criterion of eigenvalues greater than 1 (when using correlation matrices) is not optimal. One technique that has emerged is Parallel Analysis.

Parallel analysis is also known as Humphrey-Ilgen parallel analysis. Parallel analysis is now often recommended as the best method to assess the number of factors (Velicer, Eaton, and Fava, 2000; Lance, Butts, and Michels, 2006). Parallel analysis selects the factors that have fit beyond that expected by chance (so there is an underlying statistical model). The actual data are factor analyzed, and separately one does a factor analysis of a matrix of random numbers representing the same number of cases and variables. For both actual and random solutions, the number of factors on the x axis and cumulative eigenvalues on the y axis is plotted. Where the two lines intersect determines the number of factors to be extracted. Though not available directly in SPSS or SAS, O'Connor (2000) presents programs to implement parallel analysis in SPSS, SAS, and MATLAB. These programs are located at https://people.ok.ubc.ca/brioconn/nfactors/nfactors.html. The paran (available through CRAN) and paramap (available at the website in the pre-

vious sentence) libraries in R compute parallel analysis.

For a recent review of several methods to identify the number of factors see Auerswald and Moshagen, 2019, Psychological Methods, 24, 468-491.

Appendix 1: ALSCAL SYNTAX: UNFOLDING ANALYSES

The syntax for an unfolding analysis is similar to that for an MDS or and INDSCAL. Here is an example for a one dimensional solution.

```
alscal variables a b c d
 /levels = ordinal
 /shape = rect
 /criteria=dimens(1)
 /cond = row
 /plot all.
```

The /LEVELS subcommand can take ORDINAL, INTERVAL or RATIO as arguments (usually ORDINAL will be most appropriate). The /SHAPE subcommand is rectangular because one enters a subject by variable matrix, and there should be more subjects than variables. The /COND=ROW subcommand means that comparisons are meaningful only within a row. For example, within a row if a subject assigns a 3 to the first stimulus and a 5 to the second stimulus, we can infer that the subject intended to assign a greater number to the second stimulus. In this case, 3 and 5 occur on the same row because they come from the same subject so they can be meaningfully compared. On the other hand, if another subject assigns a 2 to the first stimulus we *cannot* compare that the first subject assigned a 3 to the first stimulus and the second subject assigned a 2; these two numbers appear on different rows and thus are not comparable[10]. MacCallum and colleagues have an article on the uses and abuses of "CONDITION" (1977, *Psychometrika, 42*, 297-305).

[10]The default for /CONDITION is MATRIX, meaning that all numbers within the same matrix are comparable but not across matrices. This is exactly the behavior one wants when running an MDS or an INDSCAL but not when running an unfolding analysis.

Appendix 2: FACTOR procedure in SPSS

The procedure FACTOR in SPSS performs principal components, among other things. The basic syntax is as follows:

```
FACTOR
 /variables LIST-OF-VARIABLES
 /print initial correlation extraction rotation fscore
 /plot eigen rotation
 /criteria factors(2)
 /extraction pc
 /rotation varimax
 /save reg(all) .
```

List all the variables you want to analyze in the VARIABLES subcommand. For purposes of keeping track of missing data, you can list a larger set of variables under the VARIABLES subcommand and then do repeated factor analyses on subsets of the variables using the ANALYSIS subcommand (analogous to the regression command).

The PRINT subcommand is useful to ask for the initial solution so you can see the unrotated solution, the correlation matrix, the rotated solution and the factor matrix (the later matrix contains the eigenvectors of the correlation matrix).

It is useful to request plots of the rotated solution as well as the eigenvalues. The latter plot is in the form of a scree plot, like we performed in MDS, and helps us decide on the number of factors.

Just as in ALSCAL, we specify the number of factors. The SPSS default is to print all factors that have an eigenvalue greater than 1, but I prefer the explicit syntax where you specify number of factors.

The final three subcommands specify the method of extraction (in this class we have only discussed "PC", or principal components), which rotation (e.g., varimax, quartimax, norotate), and to save the factor scores for all computed factors using the regression method. Saving the factor scores is useful because you can plot subjects in the factor space rather than merely the variables (see the track data in these lecture notes for an example). To have the factor scores for the unrotated solution print out and save, you need to specify /rotation=norotate.

Appendix 3: Input/output of matrices in SPSS

It is sometimes convenient to run a PCA or factor analysis directly from the correlation matrix. For instance, if you have a published paper that presents the correlation matrix but you don't have access to the raw data, and you want to run your own PCA. Reasons for wanting to do this may include 1) the investigator didn't present a PCA, 2) you don't like the rotation that was used and you want to try your own, and 3) you want to look at more factors than were presented in the original paper.

Instead of a data list free command you use the syntax below. Note that you need to tell the program how many subjects the correlation is based on; there is no way the program could automatically know the sample size from merely looking at the correlation matrix.

```
matrix data variables = gest gaze verb
 /format free
 /contents=corr
 /n=24.

begin data
[correlation matrix goes here; it can be lower half and the diagonals
must be all 1's]
end data.

factor
 /matrix in(corr=*)
 [etc with the usual factor commands].
```

The meaning of the asterisk in "(corr=*)" is to use whatever data is active in the current data editor as opposed to reading in a file. It is possible to read in a file that already has a correlation matrix saved (say from a previous SPSS command) by replacing the asterisk with a file name (and full path). The CORRELATION command can compute correlations and save them for later use. Simply add this line to the rest of the junk you would add to the CORRELATION command:

```
/matrix = out(corr=*)
```

Again, the asterisk means to use the current available data set (i.e., what is currently "living" in the data editor of SPSS). The key phrase is "in" or "out", which instructs SPSS programs to apply the following instructions to either A or B. In some versions of SPSS one uses /matrix out(filename) and /matrix in(cor=filename) instead of the * syntax, but I'm not clear on which version requires which syntax.

There is an analogous procedure called PROXIMITIES that could be used to input matrices into ALSCAL. For example,

```
proximities variables LIST
 /view = variables
 /measure = corr
 /matrix = out(temp)
```

will save the correlation matrix into a file temp, which can then be read into ALSCAL.

Appendix 4: Relevant Functions and packages in R

1. Unfolding analyses can be done in either the package munfold or smacof. The munfold algorithm uses a simple extension of single value decomposition (see Linear Algebra appendix). There are also tools in the MCMCpack package using Bayesian approaches to unfolding models, which I used to plot the Supreme Court data and the estimated distribution of each justice earlier in these lecture notes. As I mentioned in an earlier lecture notes, the Bayesian approach places uncertainty in a different place (the parameter rather than the data) and so each parameter (in this case each justice's ideal point) has its own distribution that can be plotted.

2. PCA and factor analyses can be done with the princomp function and factanal functions, respectively. Also, take a look at the prcomp function. There are supporting functions, like screeplot and loadings, that are also helpful in extracting relevant pieces from the PCA and FA output. Package psych also has additional PCA and FA functions including biplots. Package GPArotation has additional rotation features that can be used in several pca and fa commands in R.

3. factoextra provides many additional functions for plotting PCA and FA.

4. The package Correlplot provides some interesting plots such as correlograms and biplots that can be used for understanding the information in correlation matrices. It incorporates ideas both from PCA and principal FA. Take a look at the package corrplot too.

5. The paran package computes parallel analysis, which is becoming a standard way to decide on the number of factors or components needed in an FA or PCA. See also the paramap package, which can be downloaded at
https://people.ok.ubc.ca/brioconn/nfactors/nfactors.html.

Appendix 5: Brief review of linear algebra and matrices

1. Linear algebra is important in multivariate statistical techniques (such as factor analysis) so you will probably encounter it in subsequent statistics courses. But also fields where complicated statistics are not used often rely on linear algebra. In the field of color vision, linear algebra was used to solve the color constancy problem. Connectionist models rely heavily on linear algebra (at least the simple models do).

2. In this class I will only review the linear algebra that is relevant to the statistical material I want to present. You should be able to work comfortably with matrices and vectors (i.e., add, subtract, multiply, and "divide"). If you want to understand many of the concepts used in statistics such as degrees of freedom and orthogonality I recommend you take a course in linear algebra.

3. Vectors

   A vector is a column (or row) of numbers that have a particular order. The total number of entries in the vector is its dimension. The ordered pair ($\overline{X}$, $\overline{Y}$) is a vector of dimension 2. The contrast (1,-2,1,0) is a vector of dimension 4. A column of N data points in SPSS is a vector of dimension N.

   A vector can be expressed either as a row or as a column. For example, here is a row vector

   $$\begin{pmatrix} 0 & 1 & -1 \end{pmatrix}$$

   and here is a column vector

   $$\begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

   Keeping track of whether a vector is a column or a row will become important later.

   Geometrically, vectors define a point in an N-dimensional space, where N is the number of elements in the vector. For example, in high school you dealt with ordered pairs such as (2,4) and plotted points on a two dimensional Cartesian grid. Such an ordered pair is a vector of dimension 2.

   Vectors can be given a more detailed geometric interpretation. Simply draw a line between the origin and the point (place an arrowhead at the point to indicate direction). With such line segments it is possible to talk about angles between points as well as distances.

   We can use this geometric intuition to conceptualize the correlation. First, think of an N-dimensional space (where N refers to the number of subjects). The column X contains N

elements (one for each subject) and defines one point in that N-dimensional space. That is, each subject gets his own dimension and the point X in that space represents all the observations for variable X. The column of Y numbers similarly defines a point in the N-dimensional space. Now you have two vectors (X and Y) in an N-dimensional space. There will be an angle between these two vectors. The cosine of that angle is equal to the Pearson correlation.

Addition and multiplication by a scalar (a real number) is done element by element. For example, to add 5 to a vector we must make the 5 into a vector and then add the two vectors:

$$\begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} + \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} = \begin{pmatrix} 6 \\ 3 \\ 6 \end{pmatrix}$$

but multiplication by a scalar is simply:

$$\begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} 2 = \begin{pmatrix} 2 \\ -4 \\ 2 \end{pmatrix}.$$

Thus, while multiplication can be in the form of a scalar, addition must be between two vectors of the same dimension (i.e., the scalar 5 was written out as a column vector).

We need to define the operation called "dot product," which is the multiplication of two vectors each having the same dimension. A dot product is an element by element multiplication followed by a sum of the products. For example, consider the two vectors A = (2 4 6) and B = (5 3 1). The dot product AB is a single number: 2*5 + 4*3 + 6*1 = 28.

Note that the dot product is related to sums of squares in the sense that a product of a vector with itself leads to sum of squares of the elements. Thus, the sum of squared deviations from the mean, which we dealt with in ANOVA, is simply the dot product of the deviation vector with itself.

Another example of the dot product involves computing contrasts over means. Denote λ as the contrast vector and X as the vector of means, the contrast value is found by the dot product between λ and X (denoted λX), which is much simpler to write than the form using the sum notation $\sum_{i=1}^{T} \lambda_i X_i$.

4. Matrices

A matrix is simply a table of numbers with I rows and J columns. The dimension of a matrix is defined by the number of rows and the number of columns (i.e., IxJ). The reason for developing a mathematical language for matrices is that they occur often so it is convenient to know how to work with them (e.g., operations analogous to adding, multiplying, dividing,

and transforming that we all know how to do with numbers). Note that a single number can be treated as a matrix having one row and one column. So, what you know about number systems (such as the rationals or the reals) is just a special case of a more general system for matrices. There are even systems more general than matrices but we won't get into that here (these general theories fall in the domain of abstract algebra).

A matrix can also be viewed as a collection of vectors. For example, an 10x20 matrix can be viewed as 10 vectors of dimension 20 (or as 20 vectors of dimension 10). The term "dimension" is singular; e.g., we say a matrix has dimension 10x20.

Examples of matrices include a data file in SPSS where the rows are subjects and the columns are grouping codes/dependent variables. Consider the small data file with four subjects; the first column codes subject number, the second column codes condition, and the third column represents the observed data.

$$\begin{pmatrix} 1 & 1 & 4 \\ 2 & 1 & 6 \\ 3 & 2 & 1 \\ 4 & 2 & 2 \end{pmatrix}$$

Two matrices are equal if and only if both are of the same dimension and all corresponding entries are equal.

Addition and subtraction of matrices occurs element by element. Thus, two matrices must be of the same size to perform addition and subtraction.

Consider matrix A = $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ and matrix B = $\begin{pmatrix} 4 & 4 \\ 5 & 5 \end{pmatrix}$. Addition is:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 4 & 4 \\ 5 & 5 \end{pmatrix} = \begin{pmatrix} 1+4 & 2+4 \\ 3+5 & 4+5 \end{pmatrix}$$
$$= \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}$$

Multiplication of matrices involves a bunch of dot products. Consider the multiplication of matrix M (3x2) with matrix N (2x3). This yields a 3x3 matrix MN that contains all possible dot products of the rows of M with the columns of N. That is, the MN matrix has in the 1,1 entry the dot product of row 1 from M with column 1 from N, the MN matrix has in the 2,1

entry the dot product of row 2 from M with column 1 from N, the MN matrix has in the 3,2 entry the dot product of row 3 from M with column 2 from N, etc.

For example, consider the product of matrix A and matrix B above. We know that the product of a 2×2 matrix with a 2×2 matrix will lead to a 2×2 matrix.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 4 & 4 \\ 5 & 5 \end{pmatrix} = \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix}$$

First, compute the dot product of the first row of A (large, bold font) with the first column of B (large, bold font). The result is printed in the first row/first column of the righthand matrix (in large, bold font).

$$\begin{pmatrix} \mathbf{1} & \mathbf{2} \\ 3 & 4 \end{pmatrix}\begin{pmatrix} \mathbf{4} & 4 \\ \mathbf{5} & 5 \end{pmatrix} = \begin{pmatrix} \mathbf{14} & ? \\ ? & ? \end{pmatrix}$$

Second, compute the dot product of the first row of A with the second column of B

$$\begin{pmatrix} \mathbf{1} & \mathbf{2} \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 4 & \mathbf{4} \\ 5 & \mathbf{5} \end{pmatrix} = \begin{pmatrix} 14 & \mathbf{14} \\ ? & ? \end{pmatrix}$$

Third, compute the dot product of the second row of A with the first column of B

$$\begin{pmatrix} 1 & 2 \\ \mathbf{3} & \mathbf{4} \end{pmatrix}\begin{pmatrix} \mathbf{4} & 4 \\ \mathbf{5} & 5 \end{pmatrix} = \begin{pmatrix} 14 & 14 \\ \mathbf{32} & ? \end{pmatrix}$$

Continue this process until you exhaust all cells in the product matrix. So the typical way one multiplies matrices with each other is to take dot products of all combinations of rows from the matrix on the left with all columns from the matrix on the right.

There is a type of matrix multiplication that is cell by cell, e.g., a 3 x 3 matrix times a 3 x 3 matrix yields a 3 x 3 matrix where each cell in the resulting matrix is a product of the two cells. This is called a *Hadamard product.*

The concept of division by matrices is somewhat tricky. Recall from elementary school that one way to define division is in terms of multiplication by the reciprocal. For example, it is the same operation to divide by 5 as it is to multiply by $\frac{1}{5}$. Obviously, a number multiplied with its own reciprocal will equal 1. The intuition underlying division by matrices is to find a way to define the notion of a reciprocal. We know that this general notion needs to satisfy the property that matrix A multiplied by "the reciprocal" should give the matrix analog of 1 (the identity matrix that has 1's in the diagonal and 0's everywhere else).

The inverse of a matrix is similarly defined: the inverse of matrix A, denoted A^{-1}, is defined so that $A^{-1}A = I$, where I is the identity matrix, which has 1's in the diagonal and zeros everywhere else. This is in a form analogous to $5^{-1}5 = 1$ in the case of numbers. The computation of an inverse is not a simple matter and is beyond the scope of this course.

An example will help. Consider the matrix A and its inverse A^{-1}. Verify for yourself that the product on the left hand side yields a matrix with ones in the diagonal and zero everywhere else. The identity matrix I is the matrix analog of the number 1.

$$\begin{array}{rcl} AA^{-1} & = & I \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} -2 & 1 \\ 1.5 & -.5 \end{pmatrix} & = & \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

5. Transpose

Transposing a matrix involves switching the rows and columns. Essentially, row i becomes column i, and column j becomes row j, for all rows and columns. Here is an example. The transpose of this matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

is this second matrix

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

The transpose is usually denoted by a superscript prime as in matrix A'. Some people use the superscript "t" instead of a prime.

6. Here I'll give some applications of vectors and matrices with simple things like means, variances, and contrasts.

A variance can be written as follows. Let X be a column vector of data, 1 be a row vector of 1's (same number of elements as X), and N be the sample size. The variance can now be expressed as

$$\frac{X^t X - \frac{(X^t 1)(X^t 1)}{N}}{N-1}$$

Many of the things we've learned can be re-formulated in terms of matrix algebra. For example, the structural model for regression, which we wrote as

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i \tag{11-9}$$

for subjects i = 1,..., N. With matrix algebra we can write the regression structural model in a general way

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{11-10}$$

where Y, β, and ϵ are vectors and X is a matrix, as shown below:

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_N \end{pmatrix} = \begin{pmatrix} 1 & X_1 \\ \vdots & \\ 1 & X_N \end{pmatrix} \begin{pmatrix} \beta_0 \\ \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_N \end{pmatrix} \tag{11-11}$$

You should check your understanding that Equation 11-9 is identical to Equation 11-10 for the case of one predictor. Equation 11-10 is more compact because it is the same regardless of the number of predictors That is, when you add more predictors to the regression structural model, all that changes is the number of columns of matrix $\mathbf{X}$ and the number of rows of vector $\boldsymbol{\beta}$—you still write Equation 11-10 the same way.

The correlation coefficient can also be reformulated in terms of matrix algebra. First, I need to define the length (in Euclidean distance) of a vector. Length is simply the square root of the dot product of a vector with itself (think back to the Minkowski distance metric when we did MDS), and is usually denoted with two double horizontal lines as in the length of vector X is $\|X\|_2$. In symbols, the length of vector X is the

$$\|X\|_2 = \sqrt{X^t X} \quad (11\text{-}12)$$
$$= \sqrt{\Sigma X_i^2} \quad (11\text{-}13)$$

with the subscript 2 referring to the Euclidean length (also known as Euclidean norm or L2 as we saw in Lecture Notes 10). Again, think back to the Minkowski distance where there are other possible length measures such as the absolute value (denoted with a subscript 1), which is equivalent to the "city block" distance metric.

The Pearson correlation between data vectors a and b is defined as

$$r = \frac{a^t b}{\|a\|_2 \, \|b\|_2} \quad (11\text{-}14)$$

The correlation is identical to the cosine of the angle between vector a and vector b (which are both in N dimensional space). This angle is usually denoted θ. These vectors a and b should be "mean centered" (i.e., the mean subtracted from data point) to establish the origin.[11]

The correlation can be given a distance interpretation when all data are converted to Z-scores. This forces data to have a variance of 1, which is identical to forcing all vectors to have the same length. To show this, I first note that the distance between two vectors X & Y (i.e., the distance between the two tips of the arrows) is denoted d(X,Y) and is equal to $\|X\text{-}Y\|_2$ where the vertical bars denote length as defined above. The term $d(X,Y)^2$ can be decomposed by the law of cosines (recall that law from high school):

$$d(X,Y)^2 = \|X\|_2^2 + \|Y\|_2^2 - 2\|X\|_2 \, \|Y\|_2 \cos\theta \quad (11\text{-}15)$$

Because the two vectors have been normalized to have unit length (the Z-score transformation did this) and noting that the cosine of θ is identical to the correlation, Equation 11-15 simplifies to

$$d(X,Y) = \sqrt{1 + 1 - 2r} \quad (11\text{-}16)$$
$$= \sqrt{2 - 2r} \quad (11\text{-}17)$$

Thus, the correlation is related to distance in the special case when the vectors on which the distance is being defined are converted to Z-scores. One simply takes the square root of a linear transformation of r, where the slope and intercept of the linear transformation are -2 and 2, respectively.

7. Develop the geometry associated with the weighted average of basis vectors.

In class I will show the special vectors (1 0 0), (0 1 0), and (0 0 1) in 3d space and define the notion of vectors spanning a (sub)space.

[11] I can't resist throwing a challenge for those students who have had some linear algebra. Show that the t value from the two-sample t-test is equal to the cotangent of the angle between the data vector and the grouping vector (both of those vectors are in N-dimensional space) times $\sqrt{N-2}$.

8. Projection and regression

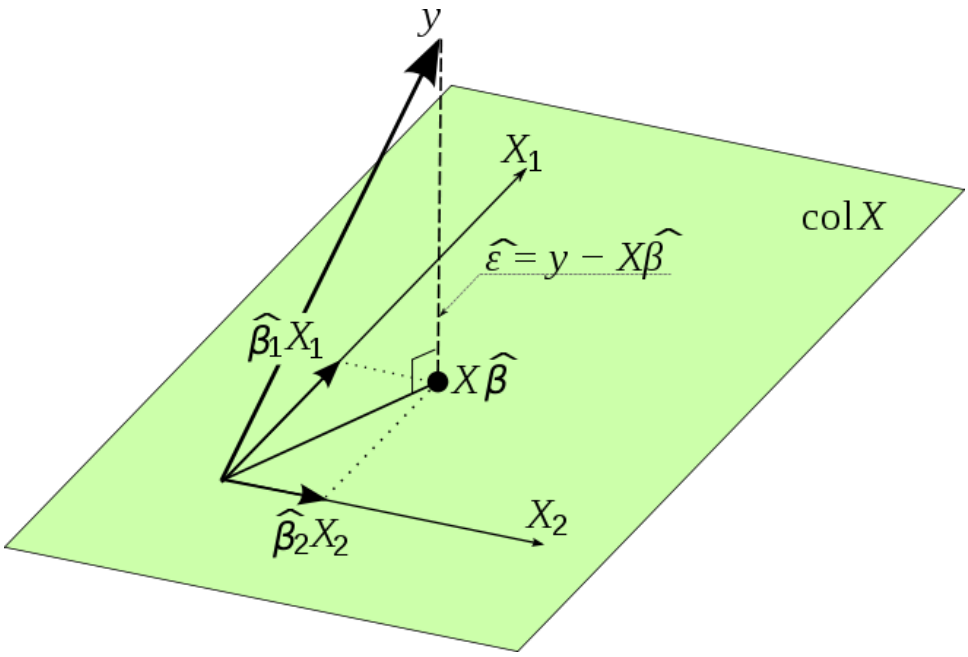
The predictor variables span a space, which most likely does not include the observed vector Y. Linear regression simply "projects" the observed vector Y onto the space spanned by the predictor variables. That projection defines a new vector $\hat{\mathrm{Y}}$, which contains the predicted values, i.e., the fits of the model, of the regression line. The usual least-squares regression just performs a projection. The parameters of the structural model (e.g., the intercept and the slope(s)) are simply another—equivalent—way to characterize the projection vector.

Projection of vector b on vector a is defined as $\frac{\mathrm{a^t b}}{\mathrm{a^t a}} a$. An example is the mean, which can be thought of as a projection onto the unit vector. For example, for a vector of data b = (1 2 3) one can project vector b onto the unit vector a = (1 1 1). Here is the computation:

$$\begin{aligned} \text{projection} &= \frac{\begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}}{\begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \\ &= 2 \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \end{aligned}$$

which has the form mean times the unit vector.

In regression, we are projecting the data vector Y onto the column space of the predictor matrix X to find $\hat{\mathrm{Y}}$, which is in the space spanned by the predictors. The residuals represent the distance between Y and $\hat{\mathrm{Y}}$. There is a nice conceptual representation of this in the figure below. The observed dependent variable Y is in a N dimensional space. The model is the subspace spanned by linear combinations of the predictors, which has dimensionality of number of parameters. This partitions the degrees of freedom by number of parameters in the model and N minus number of parameters in the error.



9. More regression in linear algebra

The betas in a regression are computed using the following linear algebra form:

$$\beta = (X^t X)^{-1} X^t Y \tag{11-18}$$

The way to derive this is to start with the usual regression ignoring the error term. The term β is our unknown and both Y and X are known (i.e., we know the predictors and the dependent variable but we don't know the weights β).

$$Y = X\beta \tag{11-19}$$

Just like in high school algebra we want to solve for the unknown β so need to "divide both sides by X" in order to isolate β, but this requires the inverse because we are dealing with matrices. However, inverses are difficult to deal with for rectangular matrices, so we first convert X into a symmetric square matrix by multiplying both sides by X^t because the product $X^t X$ is a square symmetric matrix (it is like an uncentered sum of squares matrix between predictor variables). Once we do that we can take the inverse of $X^t X$ and multiply

both sides by the inverse. Recalling that the inverse has the property that multiplying a matrix by its inverse leads to the identity matrix we have the following

$$
\begin{aligned}
Y &= X\beta && \text{(11-20)}\\
X^tY &= X^tX\beta && \text{(11-21)}\\
(X^tX)^{-1}X^tY &= (X^tX)^{-1}X^tX\beta && \text{(11-22)}\\
(X^tX)^{-1}X^tY &= 1\beta && \text{(11-23)}\\
(X^tX)^{-1}X^tY &= \beta && \text{(11-24)}
\end{aligned}
$$

10. Special matrices

A diagonal matrix is one whose off diagonals are all zero, as in

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 16 \end{pmatrix}$$

The identity matrix is a diagonal matrix such that all entries in the diagonal equal 1, as in

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The identity has the special property that a matrix times the identity equals the original matrix (much like multiplying a number by 1). The identity matrix is usually denoted by a capital I. Thus, AI = IA =A. The inverse of a matrix has the property that when it is multiplied with its original matrix, the product is one (much like when 1/5 is multiplied by 5, the result is one). This can be denoted by $AA^{-1} = A^{-1}A = I$.

A scalar matrix is a diagonal matrix where all entries in the diagonal are equal to each other. The identity matrix is a special case of a scalar matrix where all elements in the diagonal equal 1. The scalar matrix comes into play in, say, ANOVA where we assume equality of variances. MSE plays the role of the scalar (i.e., in the between-subjects ANOVA the MSE appears in each element of the diagonal representing the "variance" of that group and the off diagonals are all 0 because the groups are independent).

A symmetric matrix has the property that $A = A^t$. That is, the element a_{ij} equals a_{ji} for all i and j.

Another special matrix is the orthonormal matrix. This is a set of orthogonal vectors put into columns of a matrix and the columns of the matrix are normalized by the length of the respective column. For example, consider the two contrasts (1,-1,0) and (1,1,-2) that we may use in a 3-group between-subjects ANOVA. Put those contrasts into a matrix along with the unit vector to represent the grand mean.

```
x <- cbind(c(1, 1, 1), c(1, -1, 0), c(1, 1, -2))
# display matrix
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1   -1    1
## [3,]    1    0   -2
```

```
library(far)
x.orthonorm <- orthonormalization(x)
x.orthonorm
```

```
##           [,1]       [,2]       [,3]
## [1,] 0.5773503  0.7071068  0.4082483
## [2,] 0.5773503 -0.7071068  0.4082483
## [3,] 0.5773503  0.0000000 -0.8164966
```

Each column in X now has a length of 1 (e.g., think Euclidean distance and square each entry in a column, sum and take the sqrt). This is the length computation of Equation 11-12. We saw in ANOVA that the programs sometimes converted our contrasts into another set of contrasts that were normalized. This is the operation we saw back then.

A good orthonormalization routine can also "fill in" missing contrasts. Here is an example where I drop the last (1,1,-2) contrasts, run the smaller matrix through orthonormalization() and it "finds" the missing orthogonal contrast. It can do this because with two orthogonal contrasts (unit and 1,-1,0) there is only one more contrast that is orthogonal to both of the contrasts already in the mix. Here the unit contrast counts as one of the contrasts (unlike last semester where we didn't directly count the unit contrast, such as in our statement "there are as many orthogonal contrasts as groups minus 1"; the minus 1 is the unit contrast).

```
x.new <- x[, -3]
x.new
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1   -1
## [3,]    1    0
```

```
orthonormalization(x.new)
```

```
##           [,1]       [,2]       [,3]
## [1,] 0.5773503  0.7071068 -0.4082483
## [2,] 0.5773503 -0.7071068 -0.4082483
## [3,] 0.5773503  0.0000000  0.8164966
```

Orthonormal matrices have special properties such as the determinant is equal to 1 or -1. The determinant is defined elsewhere in these notes—one way to compute the determinant of a matrix is to multiply all eigenvalues together, so eigenval1 $*$ eigenval2 $*$ eigenval3 etc.

11. Rotation

Rotations can also be expressed in the language of matrices. A vector can be rotated by angle θ using the rotation matrix of sines and cosines. For example, the vector V = (2 1) can be rotated 90 degrees counter clockwise by multiplying with the rotation matrix as follows

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix} \tag{11-25}$$

where θ is the angle of rotation, in this case 90 degrees counter clockwise. For 90 degrees counter clockwise we have $\cos 90 = 0$ and $\sin 90 = 1$[12]. In our example,

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix} \tag{11-26}$$

Rotations are defined on pairs of axes. When you have more than two axes, you need to define rotations on all possible pairs. Rotation matrices are orthogonal so it is possible to multiply them together and order is irrelevant. So, if you have three dimensions and want to consider a rotation, you'll need to perform three pairwise rotations. Each rotation will have its own matrix. You can summarize the entire operation by a single matrix T, which is the product of the three individual pairwise rotation matrices T_1, T_2, and T_3.

[12] In R, trigometric functions are expressed in radians rather than degrees so 90 degrees corresponds to $\frac{\pi}{2}$ radians.

12. Eigenvalues and eigenvectors (aka characteristic roots and characteristic vectors)

The basic idea about eigenvalues is that they decompose and summarize a given matrix into smaller chunks of information. The definition is given by this equation where A is a matrix, λ is a single number, and X is vector:

$$AX = \lambda X \tag{11-27}$$

The meaning of this equation is very simple. Whatever matrix A does to the vector X when they are multiplied (the left hand side) is *identical* to what the single number λ does to X (the right hand side). Thus, the single number λ completely captures what matrix A does to X. We can then use the single number λ instead of the matrix A when we want to study the effects of multiplying by a large matrix. Suppose A was a 1000 $\times$ 1000 matrix. If Equation 11-27 holds for a particular vector X, then there is a sense in which the single number λ does the same thing to vector X as what matrix A does to vector X.

As you can probably guess, the λ and the X are related to each other; they come in pairs.

Here is a simple numerical example. Let A be the matrix

$$\begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}$$

Matrix A has two pairs of eigenvalues and eigenvectors. One pair is $\lambda_1 = 3$ and $X_1 = (1\ 0)$; you should check that $AX_1 = \lambda_1 X_1$. The second pair is $\lambda_2 = 2$ and $X_2 = (0\ 1)$, and you should double check that $AX_2 = \lambda_2 X_2$.

Eigenvalues and eigenvectors are important in what is known as a spectral decomposition (here, for real, symmetric matrices). The spectral decomposition takes matrix A and decomposes as follows:

$$A = L\Lambda L^t \tag{11-28}$$

where A is a real, symmetric matrix, Λ is a diagonal matrix whose elements are the eigenvalues of A, and L is the matrix that has the eigenvectors of A as columns.

Here is a numerical example. This matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

has eigenvalues 3 and 1, with eigenvectors (.707, .707) and (.707, -.707), respectively. You should check that the following matrix multiplication reproduces the original matrix A

$$\begin{pmatrix} .707 & .707 \\ .707 & -.707 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} .707 & .707 \\ .707 & -.707 \end{pmatrix}$$

One application of spectral decomposition is in reduction of information. Once matrix A is decomposed I can take fewer eigenvalue/eigenvector pairs and approximate the original matrix. For instance, only taking the first eigenvalue/eigenvector pair of matrix A, I get this approximation to matrix A:

$$\begin{aligned} \hat{A} &= \begin{pmatrix} .707 \\ .707 \end{pmatrix} \begin{pmatrix} 3 \end{pmatrix} \begin{pmatrix} .707 & .707 \end{pmatrix} \\ &= \begin{pmatrix} 1.499 & 1.499 \\ 1.499 & 1.499 \end{pmatrix} \end{aligned}$$

Thus, we approximate the original matrix A with only one eigenvalue/eigenvector pair. The savings in terms of number of elements to store increases as the matrix A gets large.

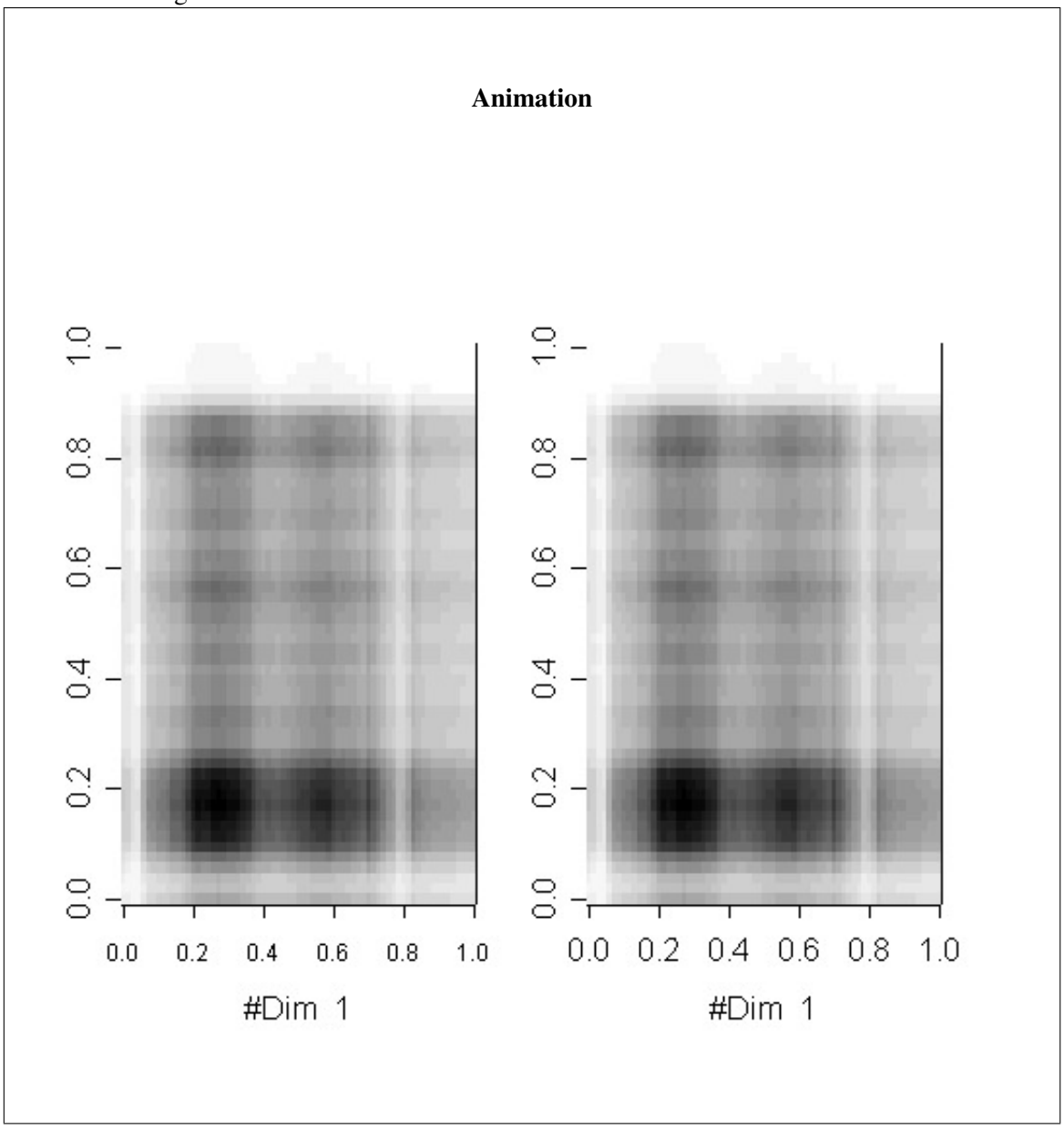
autoencoder

This idea of the eigenvalue/vector decomposition is now popular in machine learning circles in terms of an autoencoder. First we "encode" a matrix using a dimension reduction procedure like computing eigenvectors. That strips the key properties from the "noise" (e.g., the first two or three eigenvectors could be taken as the key property and the remaining eigenvectors treated as noise). Second, "decode" from the key properties to reproduce a new matrix using only those key properties. The example in the previous paragraph where I took a 2x2 matrix, saved the first eigenvalue/vector pair, and then used that eigenvalue/vector pair to reconstruct a new matrix is an example of such an autoencoder (encode then decode) procedure.

Here is an example using a picture of a human face. The picture is just a matrix of gray scale values. Each square (pixel) is white, black or a shade of gray.

I did an eigenvalue/vector decomposition of this image and created this animation. On the right side is the display of each eigenvector and on the left side is the resulting image using the cumulative eigenvectors. So the first eigenvector is merely a set of dark and light gray patches (not at all a human shape), the second eigenvector adds more detail, and slowly each progressive eigenvector adds more detail such as distinguishing the head and shoulders from the background and eventually distinguishing the shading around the nose and other facial features. The left starts to look like a human image as the additional eigenvectors add more (high frequency) information. Eventually, additional eigenvectors don't improve the resulting image. The frames switch about every 2 seconds with a new eigenvector appearing on the right and that eigenvector added to the previous eigenvectors (i.e., the $L\Lambda L^t$ computation of the resulting matrix where the new eigenvector gets added to L as another column and the

eigenvalue gets added to Λ as another entry in the matrix). The concept of an autoencoder can be merged with deep learning and nonlinear layers to produce more general and more flexible algorithms than PCA[13]



These ideas are used in image processing, compact storage of digital information, and transmission of digital information over the web.

[13]The animation works when opening the pdf file in Adobe Reader; other pdf viewers may or may not work.