

# PCA, eigen decomposition and SVD

*Shane Mueller*

*2019-04-09*

## Suggested Readings:

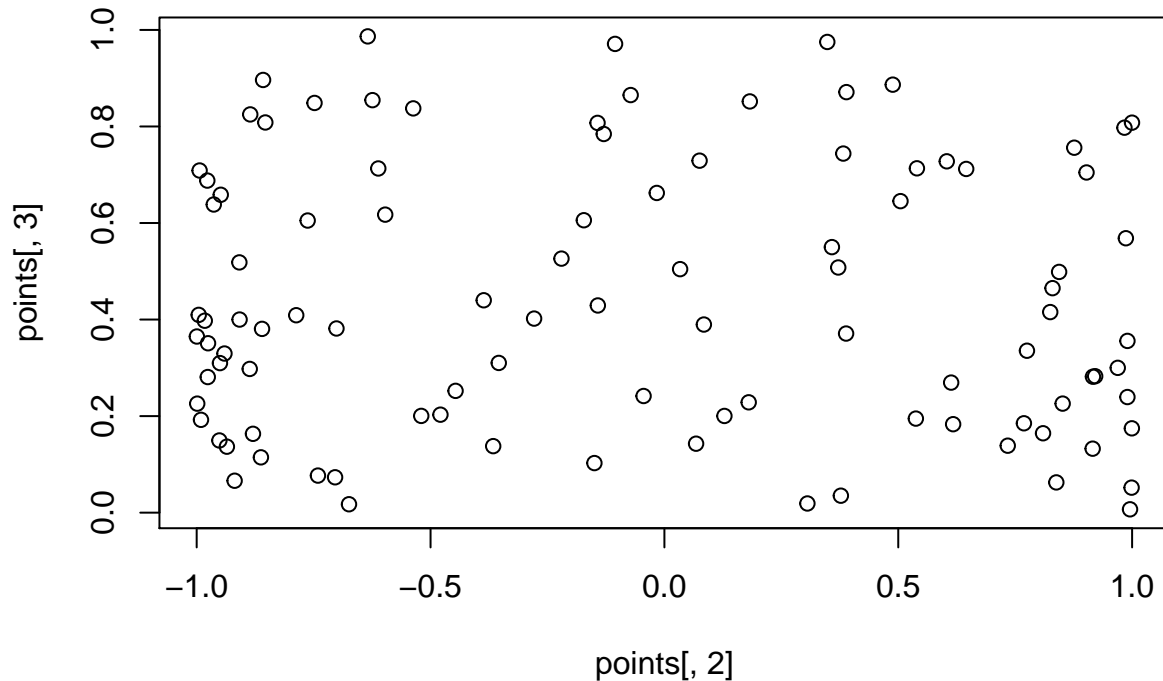
- R In action, Chapter 14
- MASS, p 302.

## Eigen Decomposition as Principal Components Analysis

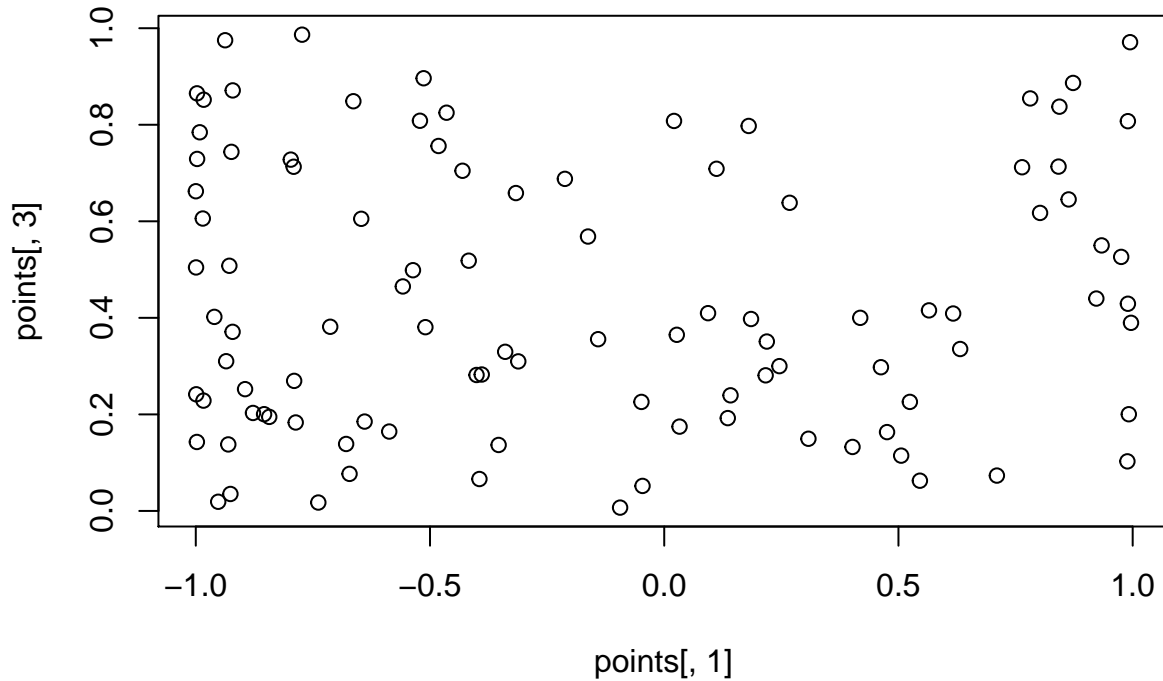
Factor analysis refers to a class of methods that, much like MDS, attempt to project high dimensional data onto a lower set of dimensions. Let's first consider this main goal.

Suppose you have a set of points in 3-dimensional space that describe some type of object, such as a cup. This might be randomly chosen points on the cup, or points on edges of the cup.

```
xy <- runif(100)
points <- cbind(cos(xy * 2 * pi), sin(xy * 2 * pi), runif(100))
plot(points[, 2], points[, 3])
```



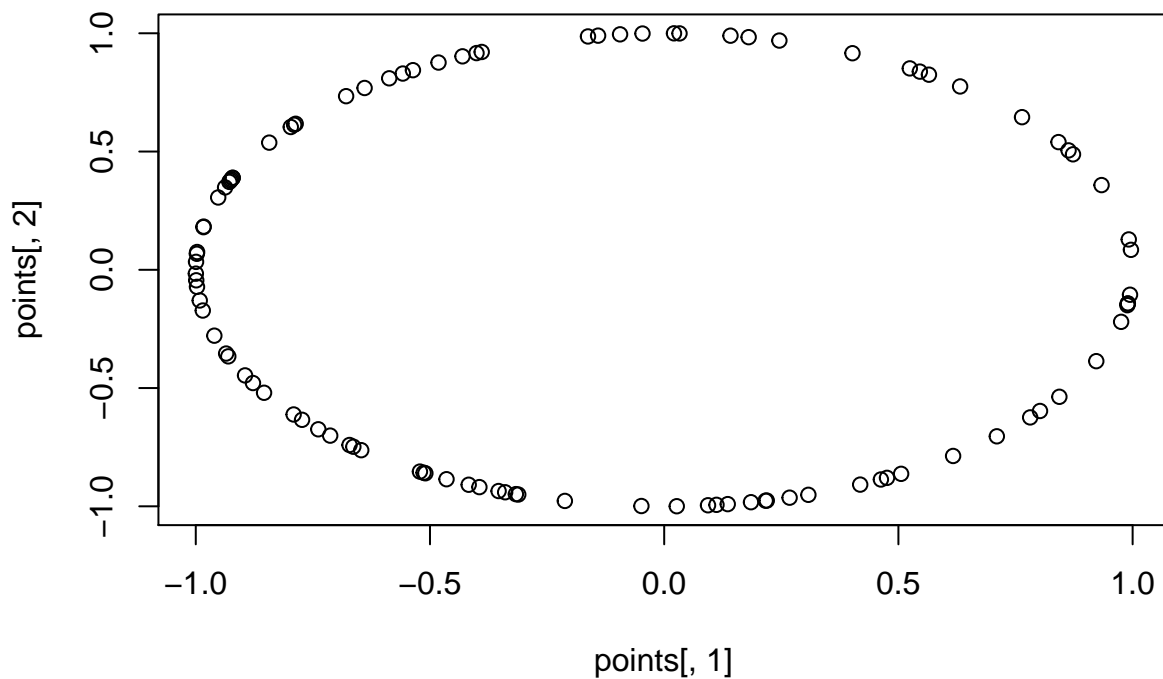
```
plot(points[, 1], points[, 3])
```



```
library(rgl)
plot3d(points)
```

If you just happen to look at the wrong pairs of points, there is no clear structure. But if you look at it the right way, you can see the cylinder emerge. The cylindrical structure means that some of the dimensions are redundant, and if we described them in different coordinates, we might learn something new. Furthermore, if those new coordinates are fewer than we started, this is called a *projection* onto a lower dimension. For example, consider the following projection:

```
plot(points[, 1], points[, 2])
```



This reveals a specific relationship between variables we didn't see before, and may indicate that P1 and P2 are related in some particular way.

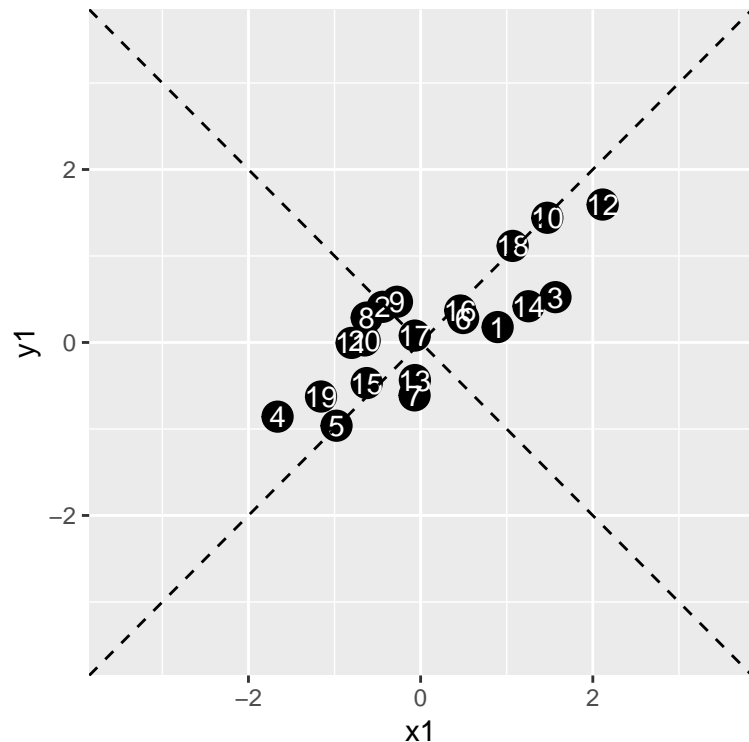
## Example: two dimensions

Suppose we had two variables we observed.

```
library(ggplot2)
base1 <- rnorm(20)
x1 <- base1 + rnorm(20) * 0.4
y1 <- base1 + rnorm(20) * 0.4

dat <- data.frame(id = 1:20, x1, y1)

ggplot(dat, aes(x = x1, y = y1)) + geom_point(size = 5) + geom_text(aes(label = id),
  col = "white") + geom_abline(slope = 1, intercept = 0, linetype = 2) + geom_abline(slope = -1,
  intercept = 0, linetype = 2) +
  xlim(-3.5, 3.5) + ylim(-3.5, 3.5)
```



The data appear to be correlated, and falling along the line  $x=y$ . This is the vector of highest variance, and might be considered the first principal component. Since we have only two dimensions, the only remaining orthogonal dimension is the dimension along the negative diagonal—the second principal component. So, suppose we wanted to rotate the axes and redescribe the data based on these two dimensions. Any point would be re-described by its closest point on the first or second new dimension.

If we look at the variability along this new first component, it looks like it will have a standard deviation of around 2; and the second dimension will have a standard deviation much smaller—maybe less than 1. If we do an eigen decomposition on the covariance matrix, we lose the original data (everything is filtered through the covariance matrix), but we have redescribed the data in terms of how they map onto the new vectors. The eigenvalues describe the variance

```
e <- eigen(cov(dat[, -1]))
print(e)
```

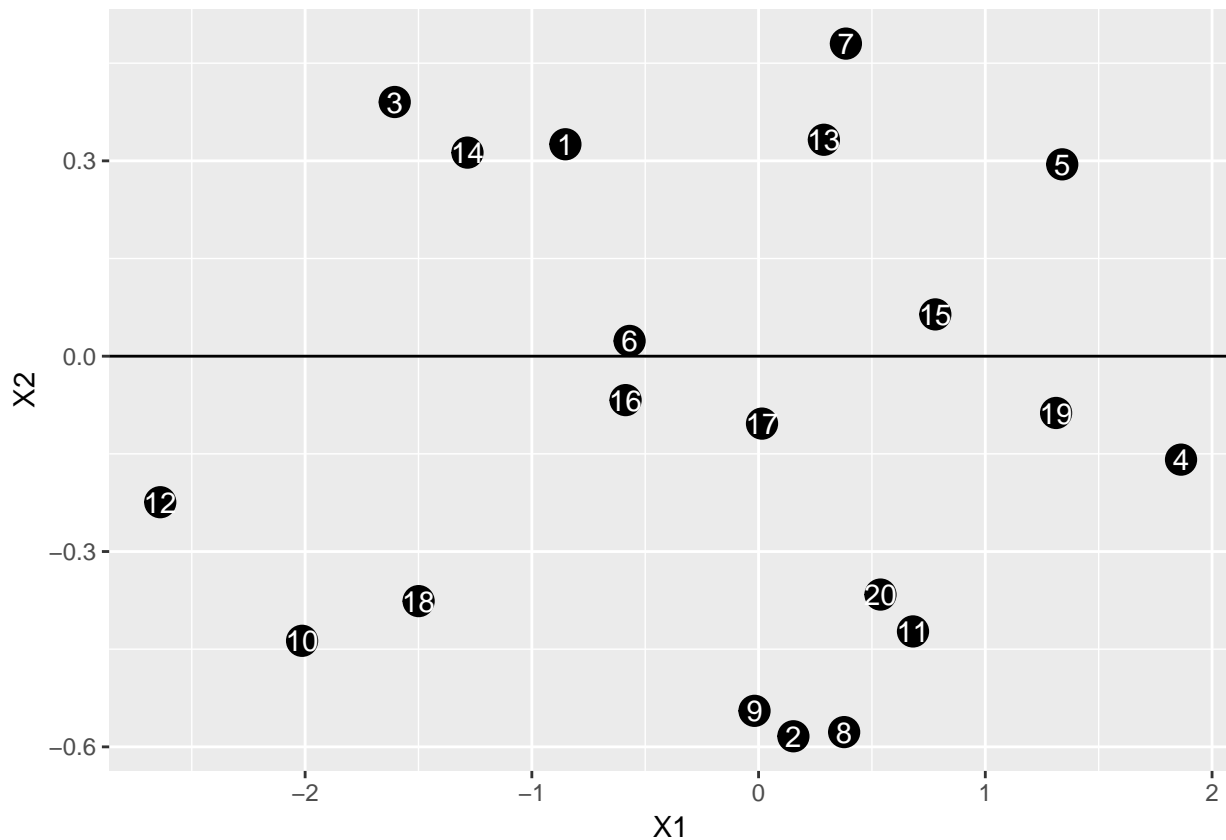
```
eigen() decomposition
$values
[1] 1.4249251 0.1238983
```

```
$vectors
      [,1]      [,2]
[1,] -0.8465004  0.5323881
[2,] -0.5323881 -0.8465004
```

We can use the eigenvector matrix to rotate the original data into this new set of axes. Notice how the arrangement is the same with an angular rotation; all the points are in the same configuration. Also, the variance in each dimension is the same as the eigenvalues.

```
rotated <- data.frame(id = dat$id, as.matrix(dat[, 2:3]) %*% e$vectors)
```

```
ggplot(rotated, aes(x = X1, y = X2)) + geom_point(size = 5) + geom_text(aes(label = id),
  col = "white") + geom_abline(intercept = 0, slope = 0)
```



```
print(var(rotated$X1))
```

```
[1] 1.424925
```

```
print(var(rotated$X2))
```

```
[1] 0.1238983
```

So, what eigen decomposition is doing is rotating the entire space to a new set of orthogonal dimensions; so

that the first axis is the one with the greatest variance, the next axis is the next-largest, and so on. In this case, because two variables were correlated, it picked the axis of greatest correlation as the first vector. If we have a lot more variables, the first most important dimension might strongly combine only some of the original values, whereas a second dimension might pick out a second set of common variance for a different set of variables.

## Example of Eigen Decomposition: Two latent factors

Here is an example where there are two ‘true’ underlying factors that the data are generated from—one set of three items based on one factor, one set based on a second factor, and a final set based on both. There is noise added to each dimension, so we wouldn’t be able to compress the data into just two dimensions, but it might have most of the variance on two.

```
library(ggplot2)
library(GGally)

base1 <- rnorm(100)
base2 <- rnorm(100)

x1 <- base1 + rnorm(100) * 0.6
x2 <- base1 + rnorm(100) * 0.6
x3 <- base1 + rnorm(100) * 0.6

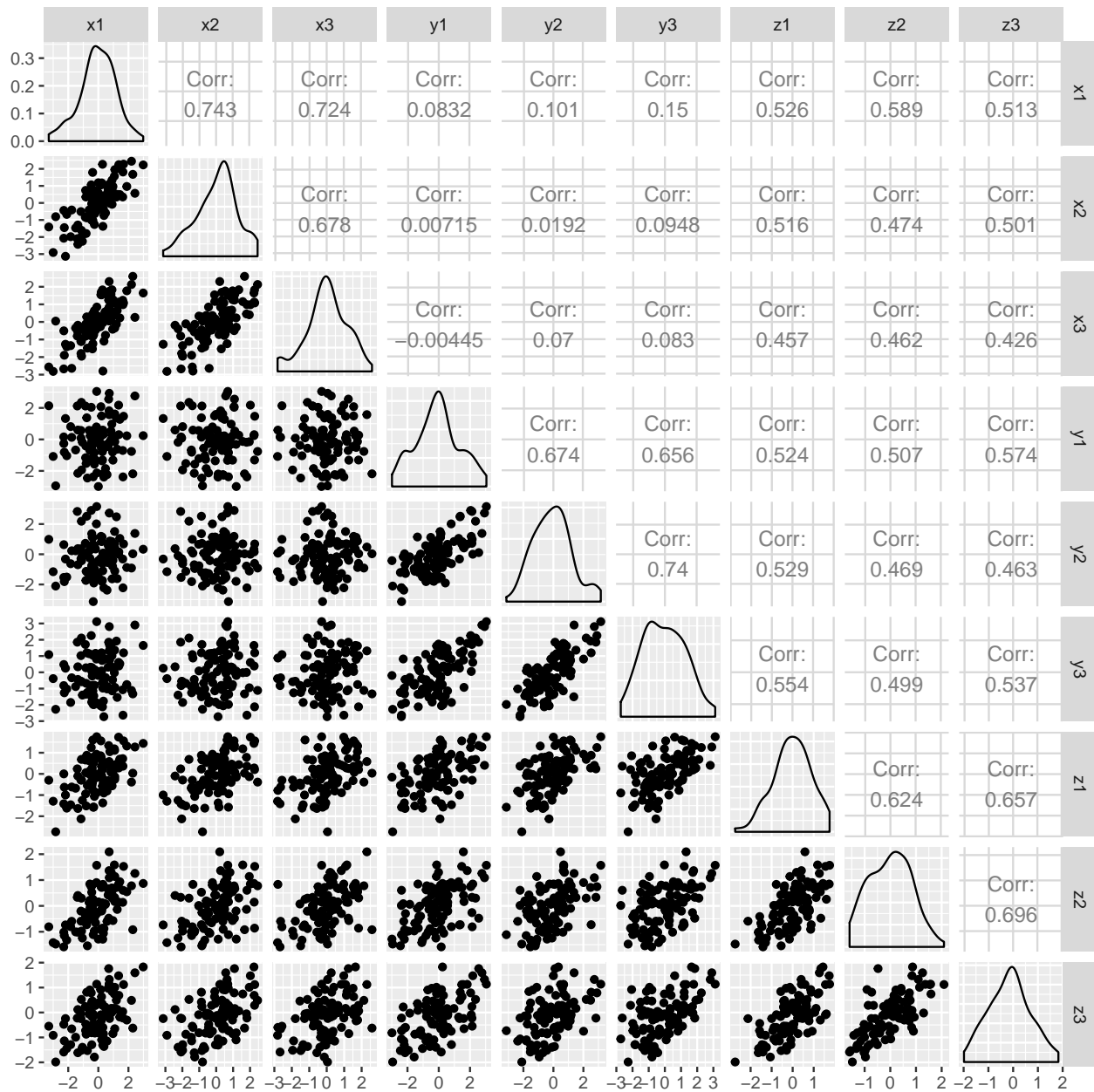
y1 <- base2 - rnorm(100) * 0.8
y2 <- base2 + rnorm(100) * 0.7
y3 <- base2 - rnorm(100) * 0.7

z1 <- (base1 + base2)/2 + rnorm(100) * 0.5
z2 <- (base1 + base2)/2 + rnorm(100) * 0.5
z3 <- (base1 + base2)/2 + rnorm(100) * 0.5

## z1 <- rnorm(100)*.5 z2 <- rnorm(100)*.5 z3 <- rnorm(100)*.5

dat <- data.frame(x1, x2, x3, y1, y2, y3, z1, z2, z3)

ggpairs(dat)
```



Within each set of x/y, there is moderately strong correlations; and z variables are likewise correlated with everything. Rather than treating each variable separately, what if we could make a linear combination of terms—just like regression, that best combine to account for a common pool of variance. But in this case, we have two independent sources of variance. It seems like an impossible task—how can we create a regression predicting a dependent variable we don't know? And how can we do it when the values we know are some complex unlabeled mixture of these values?

It seems remarkable, but this is what eigen decomposition does. Eigen is a German term meaning its 'own', and it is also sometimes referred to as the 'characteristic vector'. Eigen decomposition looks at the entire multivariate set of data and finds a new set of

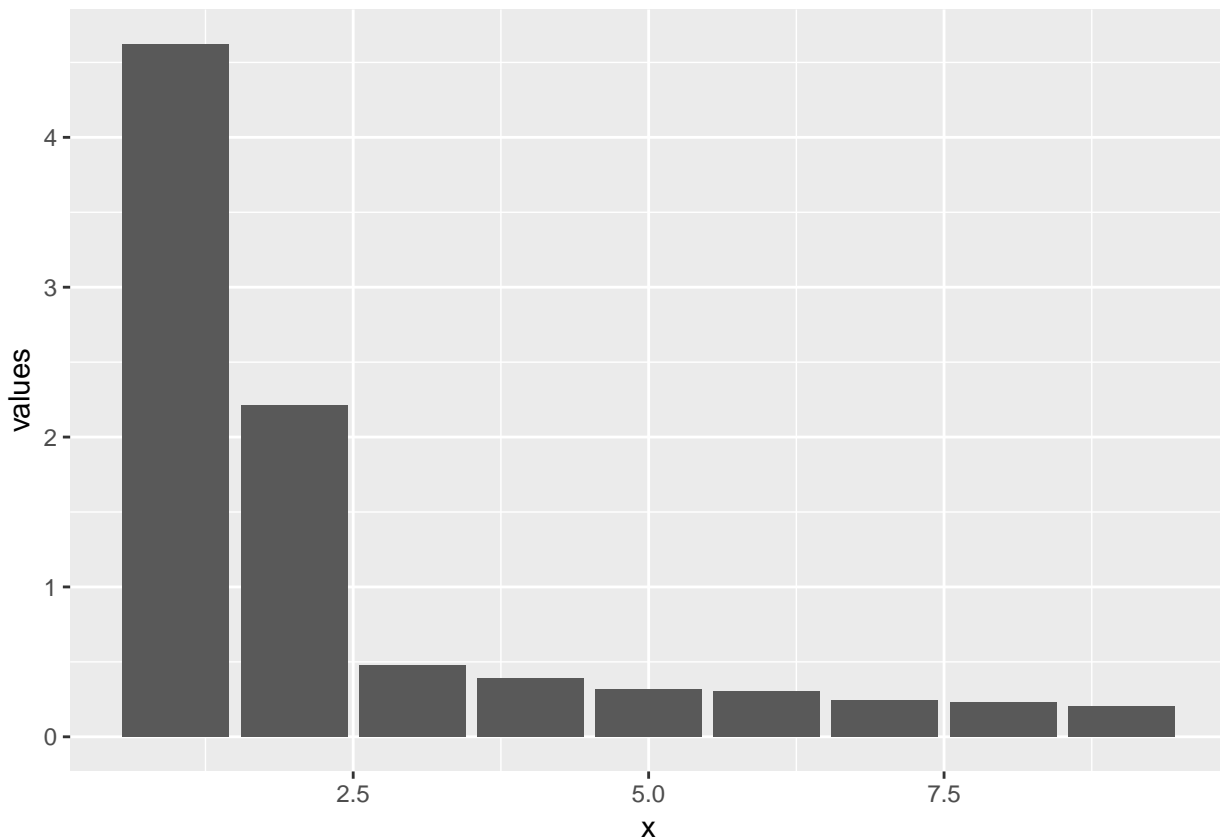
```
e <- eigen(cor(dat))
print(round(e$values, 4))
```

```
[1] 4.6221 2.2099 0.4775 0.3886 0.3198 0.3015 0.2417 0.2328 0.2061
```

```
round(e$eigenvectors, 4)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] -0.3198 -0.3978 -0.0257 -0.2113 -0.0889 -0.2351  0.3118 -0.5178
[2,] -0.2900 -0.4307 -0.0208  0.4124 -0.3290  0.0045  0.4278  0.1663
[3,] -0.2771 -0.4222 -0.3562 -0.3007  0.4679  0.4571 -0.2306 -0.0057
[4,] -0.2937  0.4141  0.3550  0.0685  0.4770  0.0403  0.3213 -0.4148
[5,] -0.2935  0.4066 -0.4704 -0.2059  0.0614 -0.0277  0.4786  0.4614
[6,] -0.3135  0.3743 -0.3418 -0.0013 -0.5538  0.2271 -0.2959 -0.4179
[7,] -0.3953  0.0181 -0.2020  0.5544  0.2965 -0.4552 -0.4142  0.0722
[8,] -0.3918 -0.0147  0.3606 -0.5555 -0.1651 -0.4188 -0.2571  0.2602
      [,9]
[1,] -0.5155
[2,]  0.4908
[3,]  0.2158
[4,]  0.3272
[5,] -0.1957
[6,]  0.1560
[7,] -0.1526
[8,]  0.2667
[ reached getOption("max.print") -- omitted 1 row ]
```

```
ggplot(data.frame(x = 1:9, values = e$values), aes(x = x, y = values)) + geom_bar(stat = "identity")
```



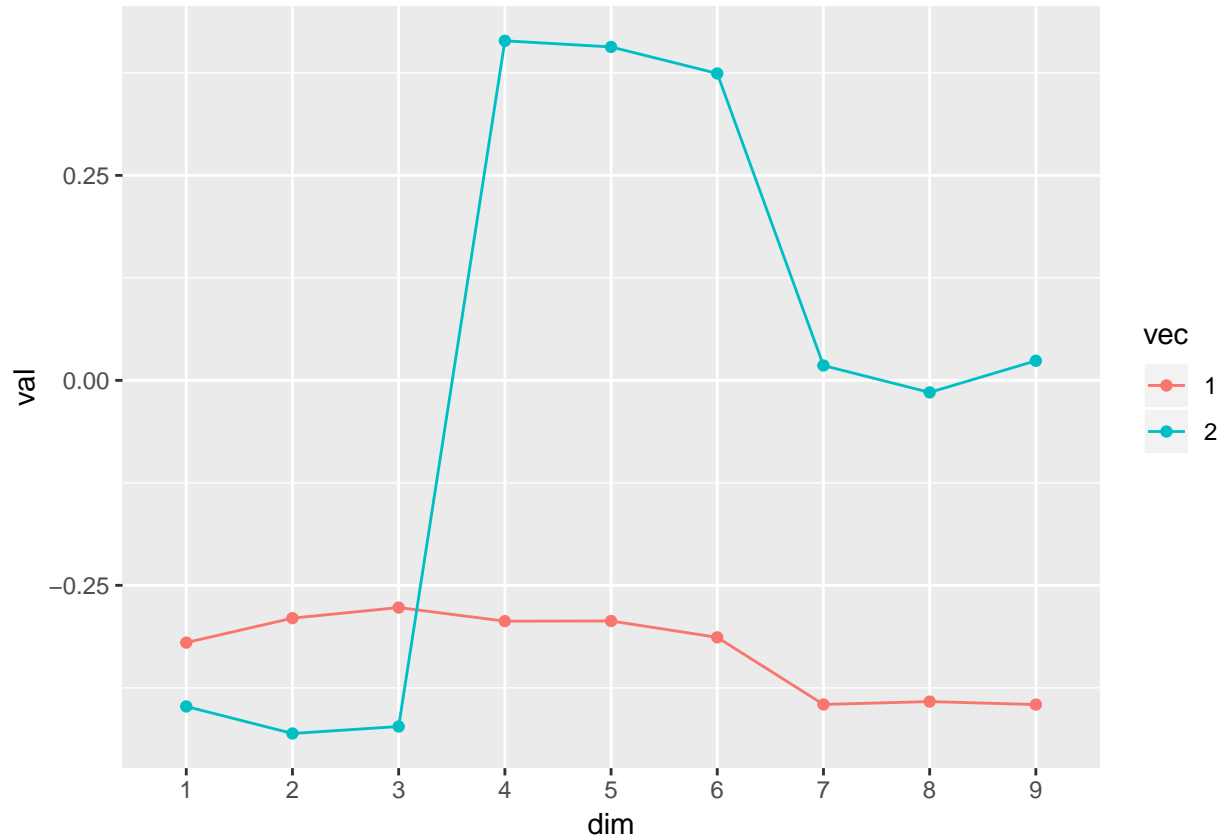
Now, the total variance in the first two dimensions is fairly large, and it trails off after that. What about the new dimensions: let's look at just the first two dimensions:

```

dat <- data.frame(dim = as.factor(rep(1:9, 9)), vec = as.factor(rep(1:9, each = 9)),
  val = as.vector(e$variables))[1:18, ]

ggplot(dat, aes(x = dim, y = val, group = vec, col = vec)) + geom_point() +
  geom_line()

```



Here, the first vector captures the shared variance across all questions—the last three questions are related to both groups. Any value close to 0 indicates that the question/item is near 0 on that component. Then, the second component gets coded as a binary factor that either explains items 1-3 or items 4-6. This sort of automatically creates a contrast coding between the first two dimensions.

## Example of Eigen Decomposition: network flows

Suppose you had a network with different nodes and transition probabilities between nodes. For example, you might have three economic classes: poverty, middle-class, and wealthy. Any person from one generation has a certain probability of ending up in each of the other classes, maybe with a structure like this:

```

transition <- rbind(c(0.5, 0.5, 0), c(0.25, 0.5, 0.25), c(0, 0.5, 0.5))

```

For any generation of people, there is a distribution of class membership, and we can derive the subsequent distribution by cross-multiplying

```

dsn <- runif(3)

dsn <- dsn/sum(dsn)
overtime <- matrix(0, nrow = 100, ncol = 3)
overtime[1, ] <- dsn
for (i in 2:100) {

```

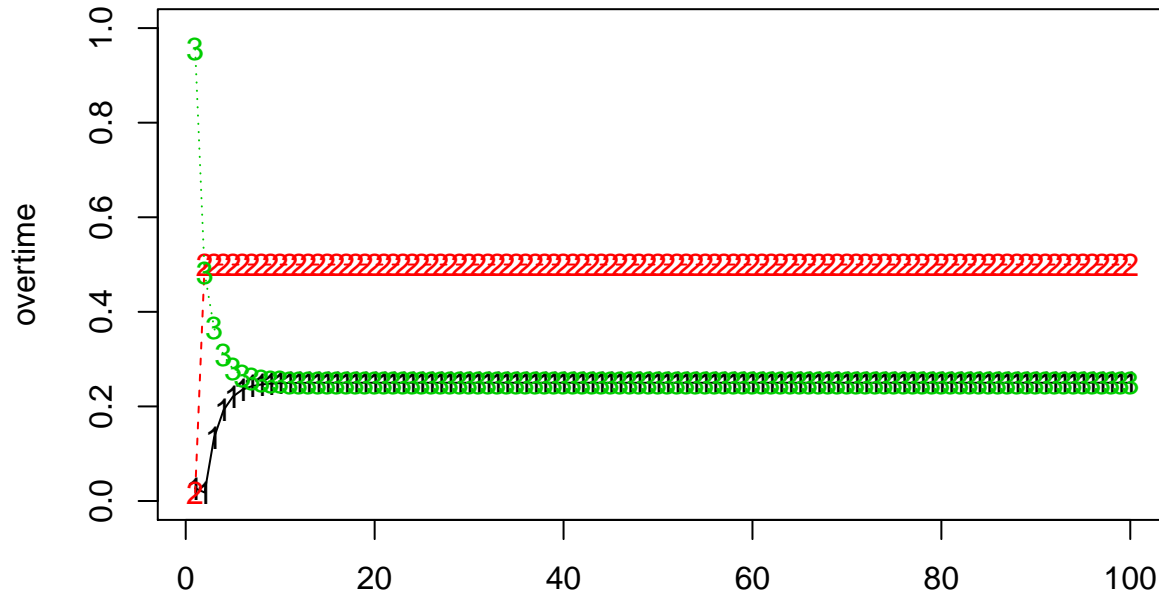


```

dsn <- dsn %*% transition
# dsn <- dsn/sum(dsn)
overtime[i, ] <- dsn
}

matplot(overtime, ylim = c(0, 1), type = "o")

```



```

e <- eigen(t(transition))

print(e)

eigen() decomposition
$values
[1] 1.000000e+00 5.000000e-01 -7.019029e-17

$vectors
      [,1]      [,2]      [,3]
[1,] 0.4082483 -7.071068e-01 0.4082483
[2,] 0.8164966 8.112147e-16 -0.8164966
[3,] 0.4082483 7.071068e-01 0.4082483

print(dsn)

```

```

      [,1] [,2] [,3]
[1,] 0.25 0.5 0.25

```

Notice that the eigenvectors have one with all positive values, and it is (.4,.8,.4). If we normalize this, we get (.25, .5, .25), which is the stationary distribution of this stochastic matrix. Sometimes, there are multiple stable distributions depending on the starting conditions, and these would each map onto a distinct eigenvector. In this case, the second eigenvector is essentially -1,0,1; this vector cannot happen for a transition matrix (you can't have negative probability), so in this case there are no other stable distributions.

Here is another case where there is complete ability to transition between groups. Now, the first eigenvector is all negative, which also predicts the stationary distribution, at least ordinally.

```

transition <- rbind(c(0.5, 0.4, 0.1), c(0.4, 0.5, 0.1), c(0.1, 0.4, 0.5))
e <- eigen(t(transition))
print("Eigen decomposition:")

```

```
[1] "Eigen decomposition:"
```

```
print(e)
```

```
eigen() decomposition
```

```
$values
```

```
[1] 1.0 0.4 0.1
```

```
$vectors
```

```

      [,1]      [,2]      [,3]
[1,] -0.6337502 -7.071068e-01 -7.071068e-01
[2,] -0.7242860  2.266233e-16  7.071068e-01
[3,] -0.2716072  7.071068e-01  3.462330e-17

```

```
e$vectors[, 1]/sum(e$vectors[, 1])
```

```
[1] 0.3888889 0.4444444 0.1666667
```

```
dsn <- runif(3)
```

```
dsn <- dsn/sum(dsn)
```

```
overtime <- matrix(0, nrow = 100, ncol = 3)
```

```
overtime[1, ] <- dsn
```

```
for (i in 2:100) {
```

```
  dsn <- dsn %*% transition
```

```
  # dsn <- dsn/sum(dsn)
```

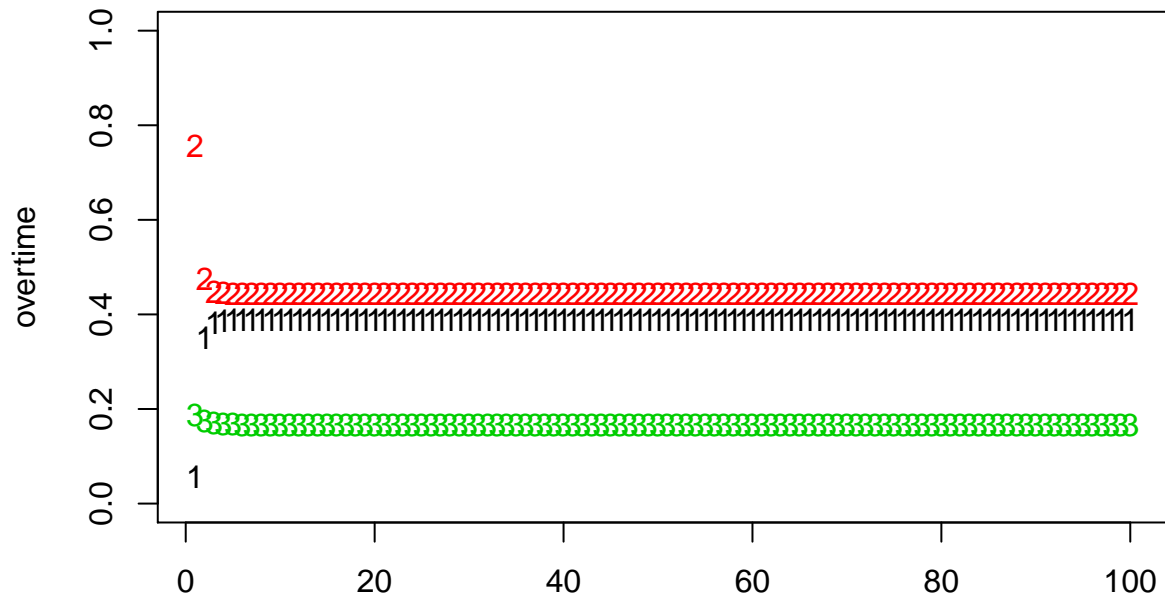
```
  overtime[i, ] <- dsn
```

```
}
```

```
print("-----")
```

```
[1] "-----"
```

```
matplot(overtime, ylim = c(0, 1))
```



```
print("Stationary distribution:")
```

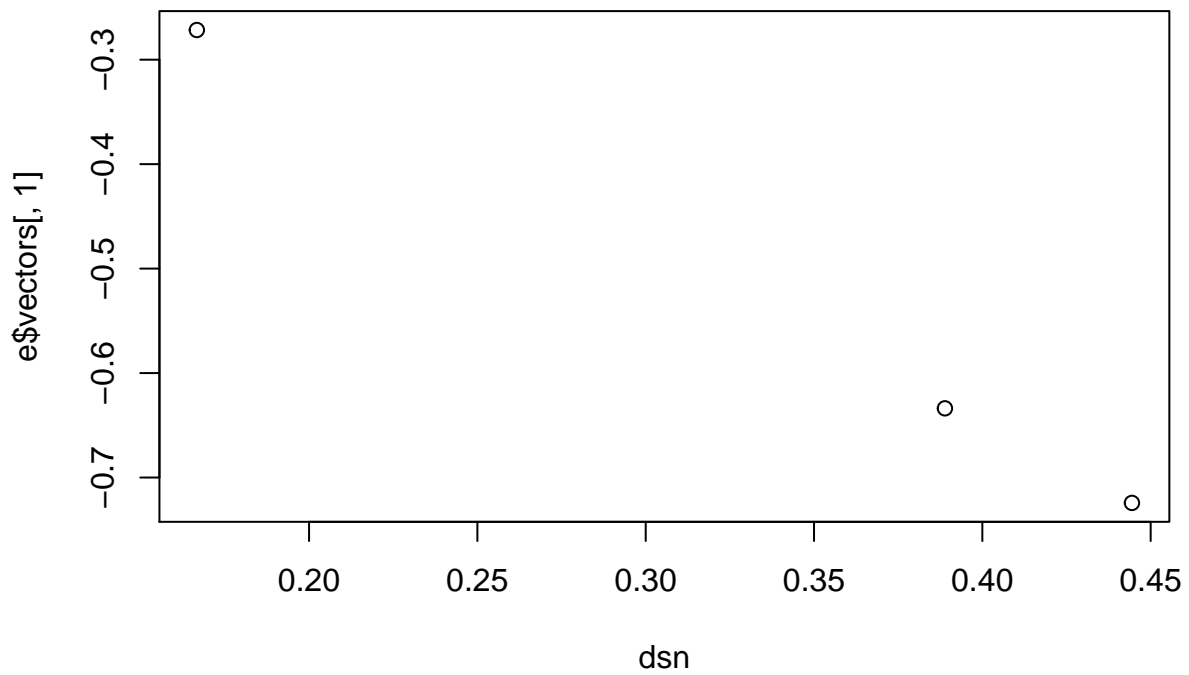
```
[1] "Stationary distribution:"
```

```
print(dsn)
```

```
      [,1]      [,2]      [,3]
[1,] 0.3888889 0.4444444 0.1666667
```

```
plot(dsn, e$eigenvectors[, 1], main = "Comparing stable distribution to first eigenvector")
```

### Comparing stable distribution to first eigenvector



## Factors and Dimensions

Oftentimes, the number of things we can measure about a person are greater than the number of theoretically-interesting constructs we think these measureables explain. For example, on a personality questionnaire, one asks a bunch of related questions that we believe all involve the same thing, with the hope that we will get robust and reliable measures. A person might by chance agree with one of the statements, but it is unlikely that they will agree with them all just by chance.

Measurements and questions that hang together like this are conceived of as a *factor*. This is similar to the previous notions of clusters, but there can be differences. A factor is a set of questions that covary together; not one in which the same answers are given by everybody. If you know all but one answer someone gives to the answers that are in a factor, you should have a pretty good idea of what the remaining answer is, regardless of whether their answer is low or high on the factor.

Thus, if you have a 10 measurements (answers) that are all a part of a common factor, and if you can project them down onto a single combined dimension without losing much information, you have succeeded. For example, if you were to measure left arm length, right arm length, left thigh length, right thigh length, left forearm length, right forearm length, left calf length, and right calf length, chances are these will all vary fairly closely together. A factor describing these might be a weighted average of these 8 measures (maybe even weighing them all equally). These would project down from 8 dimensions to just 1 or 2. If you started adding other measurements (measurements of shoe size, weight, waist size, etc.), these may not be a closely related, and may in fact form other factors.

A number of related methods have been developed to do this sort of analysis, and they go generally under the name ‘factor analysis’, but some are also called Principal Components Analysis, exploratory factor analysis (EFA), Confirmatory Factor Analysis (CFA), and as you make more complex hypotheses about the relationship between variable, there are other related methods (Structural Equation Modeling, LISREL, Path Analysis, etc.). The naming of these is often confusing and historically has been tied to specific software packages rather than generic analysis routines, so it is important to understand clearly what methods you are using, and what methods other are using, when you do this sort of analysis.

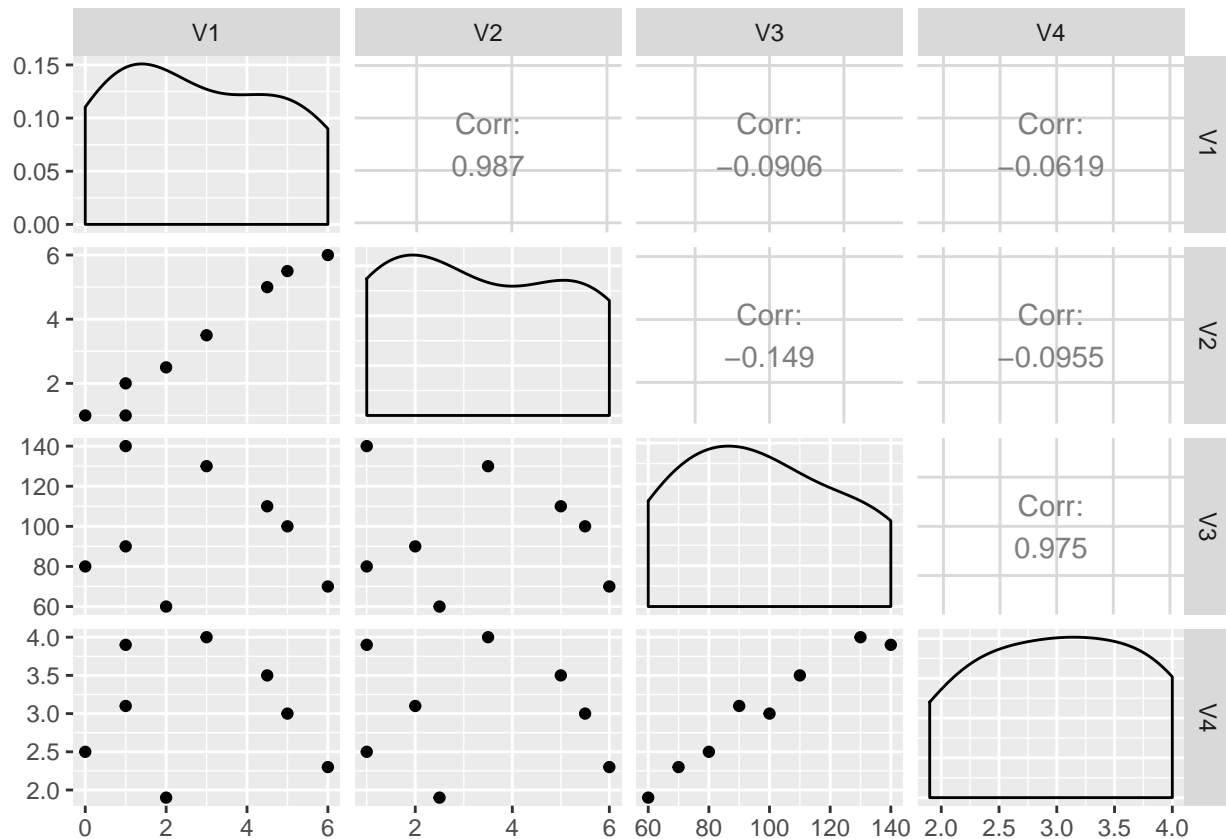
Typically, factor analysis focuses on analyzing the correlational structure of a set of questions or measures. When thinking about correlations, a factor might be a set of questions that are highly positively correlated *within* the set, but are uncorrelated with questions outside the set. Typically, negatively-correlated items are incorporated into the factor the same way positively-correlated items would be. This is somewhat like looking into the distance measures we examined with previous methods. With  $N$  questions, there are  $N^2$  correlations in a matrix, which is sort of like placing each question in an  $N$ -dimensional space. Factor analysis attempts to find a number of dimensions smaller than  $N$  that explains most of the same data.

### Example: four questions with two factors

Let’s suppose we have a set of 4 questions related to two different factors: happiness and intelligence. for example:

```
data <- rbind(c(5, 5.5, 100, 3), c(4.5, 5, 110, 3.5), c(3, 3.5, 130, 4), c(1,
  1, 140, 3.9), c(0, 1, 80, 2.5), c(1, 2, 90, 3.1), c(6, 6, 70, 2.3), c(2,
  2.5, 60, 1.9))

# pairs(data) cor(data)
library(GGally)
ggpairs(as.data.frame(data))
```



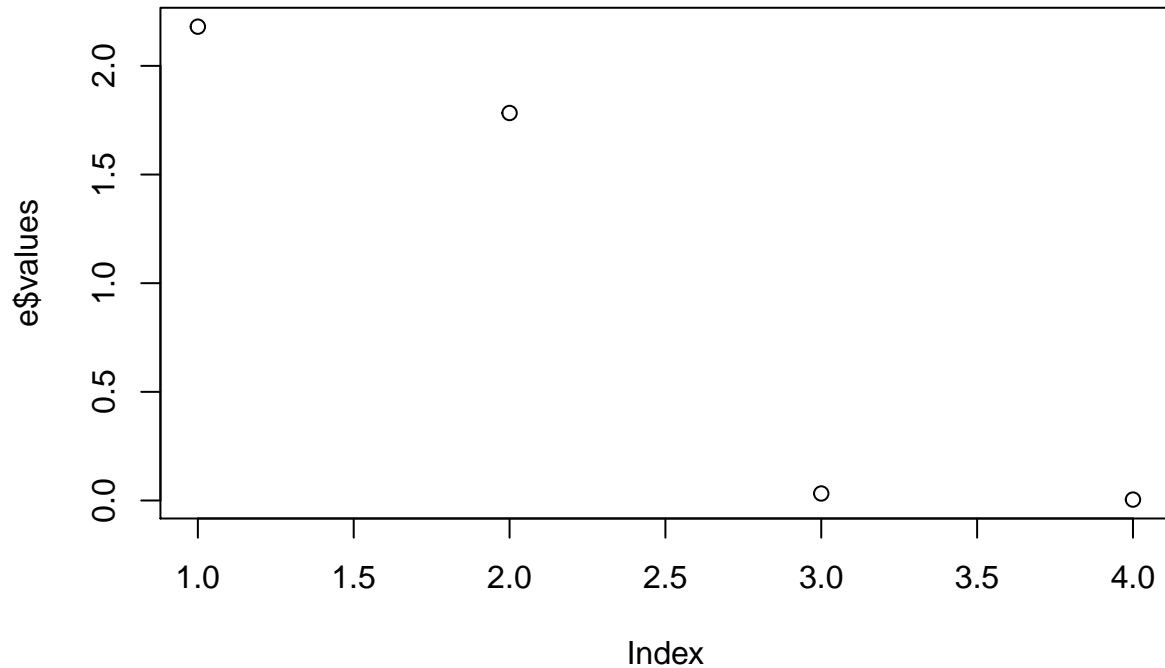
Notice that Q1/Q2 are highly correlated, as are Q3/Q4, but there is low correlations between these clusters. A factor analysis should be able to figure it out.

## Principal Components Analysis/eigen decomposition as factor analysis

Principal Components Analysis (PCA) is the simplest version of factor analysis. It is typically based on eigen decomposition, which produces an outcome somewhat like an ANOVA model does. That is, for an ANOVA model, you might try to predict responses based on a weighted average of predictor variables. The outcome of an eigen decomposition of the correlation matrix finds a weighted average of predictor variables that can reproduce the correlation matrix... without having the predictor variables to start with.

What PCA does is transforms the data onto a new set of axes that best account for common data.

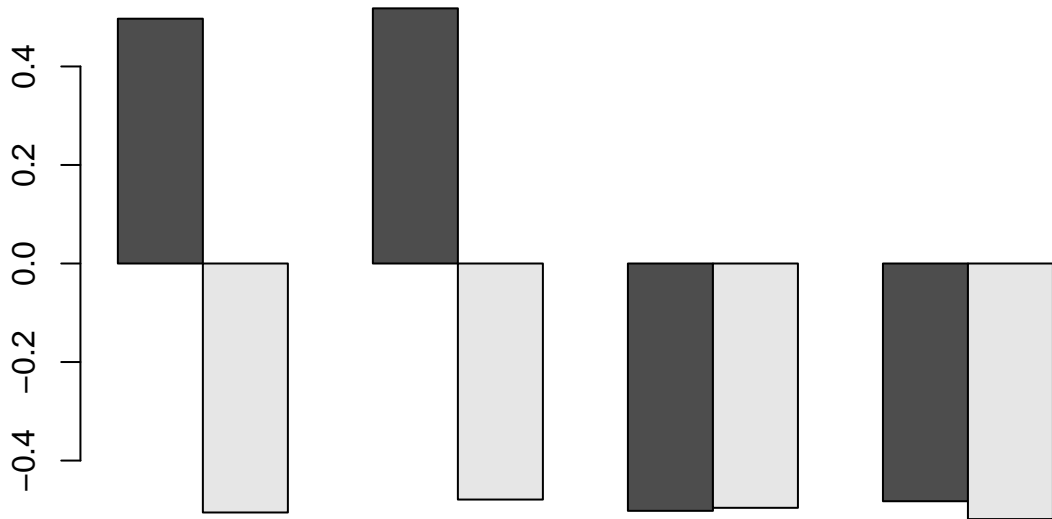
```
e <- eigen(cor(data))
plot(e$values)
```



```
e$eigenvectors
```

	[,1]	[,2]	[,3]	[,4]
[1,]	0.4968782	-0.5053846	-0.3846191	0.5914107
[2,]	0.5179202	-0.4791360	0.3608988	-0.6098684
[3,]	-0.5020219	-0.4959033	-0.5930808	-0.3876973
[4,]	-0.4825399	-0.5187437	0.6083382	0.3577496

```
barplot(t(e$eigenvectors[, 1:2]), beside = T)
```

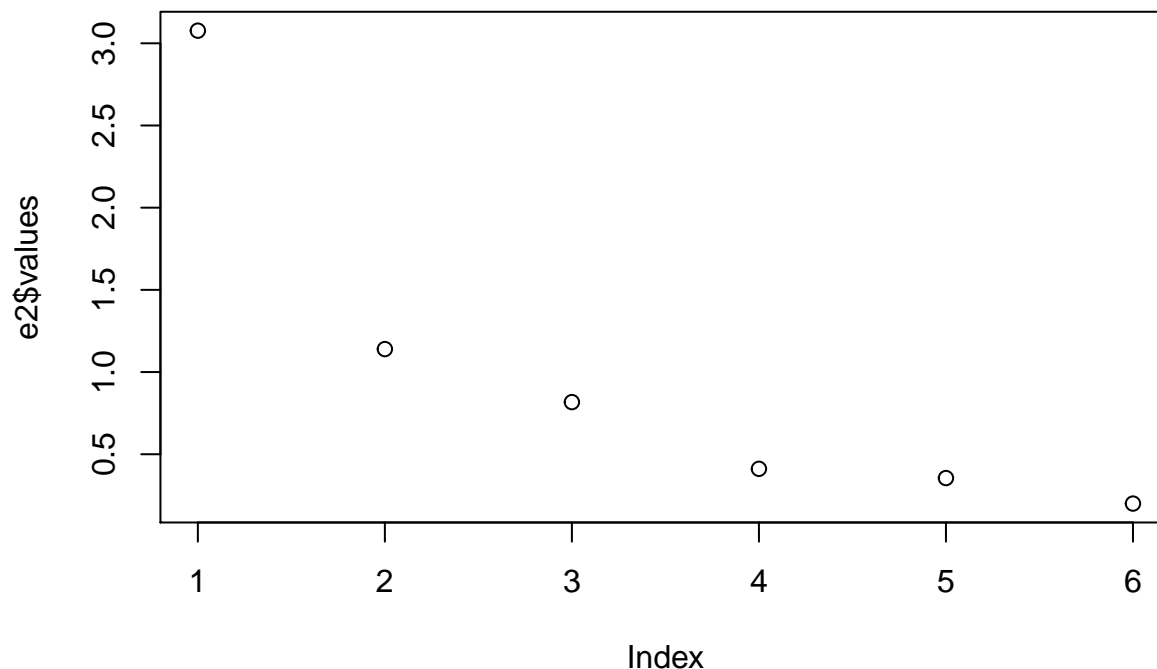


```
#
```

If we look at `e$values`, this gives a measure of the importance of each dimension. `e$eigenvectors` gives us the actual dimensions. Each column of `e$eigenvectors` is considered a factor—a way of weighting answers from each question to compute a composite. Here, after 2 dimensions, the eigen values go below 1.0, and fall off considerably, indicating most of the variability is accounted for. Let's look at another slightly more complex data set: one on human abilities. We will use `ability.cov`—the covariance of these abilities.

## Example: Human abilities

```
library(GPArotation)
data(ability.cov)
e2 <- eigen(cov2cor(ability.cov$cov))
plot(e2$values)
```



```
vec <- e2$vectors
rownames(vec) <- rownames(ability.cov$cov)
round(vec, 3)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
general	-0.471	0.002	0.072	0.863	0.037	-0.164
picture	-0.358	0.408	0.593	-0.268	0.531	0.002
blocks	-0.434	0.404	0.064	-0.201	-0.775	0.051
maze	-0.288	0.404	-0.794	-0.097	0.334	0.052
reading	-0.440	-0.507	-0.015	-0.100	0.056	0.733
vocab	-0.430	-0.501	-0.090	-0.352	0.021	-0.657

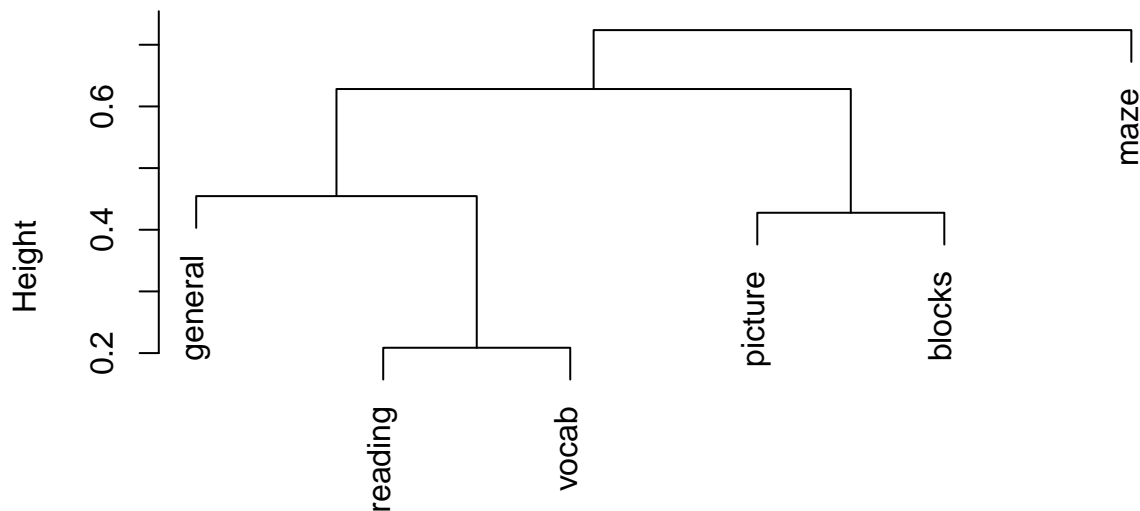
Here, there is a large drop-off between 1 and 2 eigenvalues, but it really flattens out after 3 dimensions. Looking at the eigenvectors, vector values that are different from 0 indicate a high positive or negative involvement of a particular question in a particular factor. Here, the first factor is generally equally involved in each question—maybe general intelligence. This makes sense if you look at the covariance, because there is a lot of positive correlation among the whole set. The second factor is Q234 - Q56; maybe visual versus lexical intelligence. This is one way of carving up the fact that there might be a general intelligence that impacts everyone, and a second factor that isn't general but might be visual or vocabulary. This is a bit misleading though—if someone is good at both or bad at both, it gets explained as general intelligence; if they are good or bad at one or the other, it gets explained as a specific type of intelligence.

The third vector is essentially picture versus maze, which load together on factor 2. Maybe there are aspects of these two visual tasks that differ. Looking at this, a hierarchical structure appears, which might indicate a hierarchical clustering would be better. I'll create a distance measure from the covariance converting to a correlation and subtracting from 1.0.

```
library(cluster)

cor <- ability.cov$cov/sqrt(diag(ability.cov$cov) %*% t(diag(ability.cov$cov)))
dist <- as.dist(1 - cor)
a <- agnes(dist)
plot(a, which = 2)
```

### Dendrogram of agnes(x = dist)



dist  
Agglomerative Coefficient = 0.44

This sort of maps onto our intuition; maze here looks like a bit of an outlier.

The princomp function does the same thing as eigen(), with a few additional bells and whistles. Notice that the vectors are identical. Another option in the psych::principal function.

```
p <- princomp(covmat = cov2cor(ability.cov$cov))
plot(p)
summary(p)
```

Importance of components:

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
Standard deviation	1.7540877	1.0675615	0.9039839	0.64133665	0.59588119
Proportion of Variance	0.5128039	0.1899479	0.1361978	0.06855212	0.05917907
Cumulative Proportion	0.5128039	0.7027518	0.8389497	0.90750178	0.96668085
	Comp.6				
Standard deviation	0.44711845				
Proportion of Variance	0.03331915				
Cumulative Proportion	1.00000000				

```
loadings(p)
```

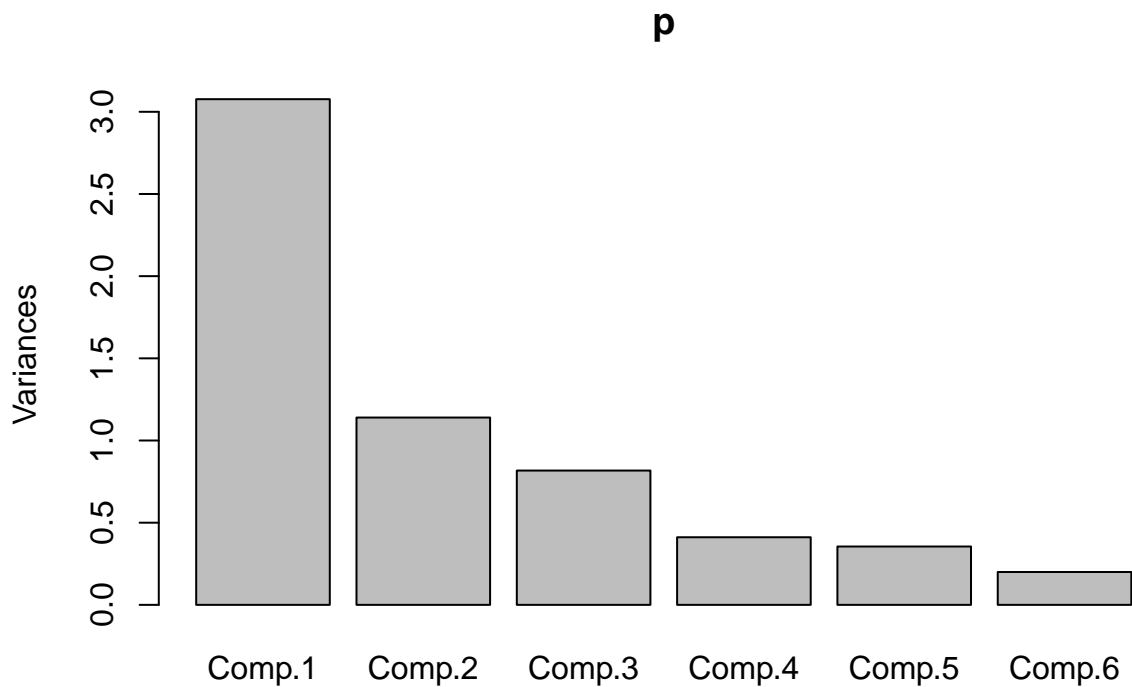
Loadings:



	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6
general	0.471			0.863		0.164
picture	0.358	0.408	0.593	-0.268	0.531	
blocks	0.434	0.404		-0.201	-0.775	
maze	0.288	0.404	-0.794		0.334	
reading	0.440	-0.507		-0.100		-0.733
vocab	0.430	-0.501		-0.352		0.657

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6
SS loadings	1.000	1.000	1.000	1.000	1.000	1.000
Proportion Var	0.167	0.167	0.167	0.167	0.167	0.167
Cumulative Var	0.167	0.333	0.500	0.667	0.833	1.000

`screeplot(p)`



The advantage of `princomp` is that it lets you use the raw data, and will compute scores based on those base data. There is a related function called `prcomp`, which will do a similar analysis, but handles situations where you have fewer participants than questions.

### Exercise: Big five data

Do a PCA of the big five data:

```
data <- read.csv("bigfive.csv")

dat.vals <- data[, -1] ##remove subject code
qtype <- c("E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "E", "A", "C",
          "N", "O", "E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "E", "A", "C",
          "N", "O", "E", "A", "C", "N", "O", "E", "A", "C", "N", "O", "O", "A", "C",
          "O")
valence <- c(1, -1, 1, 1, 1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1, 1, -1, 1,
            1, -1, 1, -1, -1, 1, 1, -1, 1, 1, 1, -1, 1, 1, -1, -1, 1, -1, 1, 1, 1, -1,
            1, -1, 1)
```

```

add <- c(6, 0, 0)[valence + 2]

tmp <- dat.vals
tmp[is.na(tmp)] <- 3.5
reversed <- t(t(tmp) * valence + add)
## reverse code questions:

bytype <- reversed[, order(qtype)]
key <- sort(qtype)
colnames(bytype) <- paste(key, 1:44, sep = "")

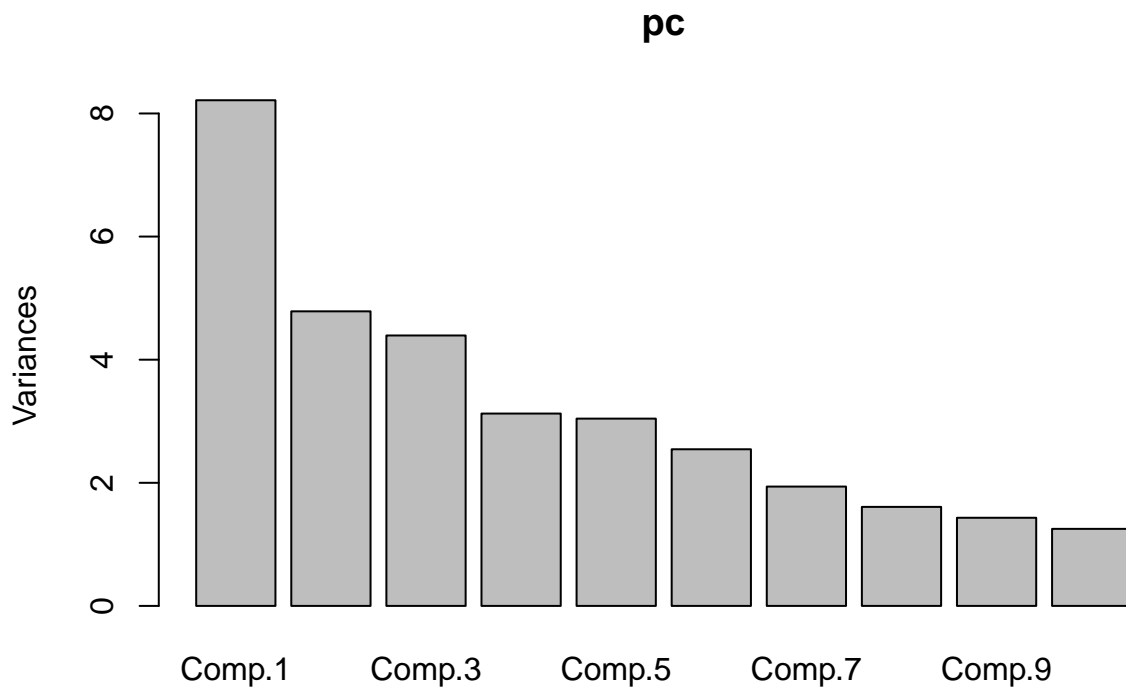
```

Here is one way to visualize what PCA does. We can make an image plot of the correlation matrix of these questions. Because they are ordered, we see bands in the image:

```

pc <- princomp(bytype)
plot(pc)

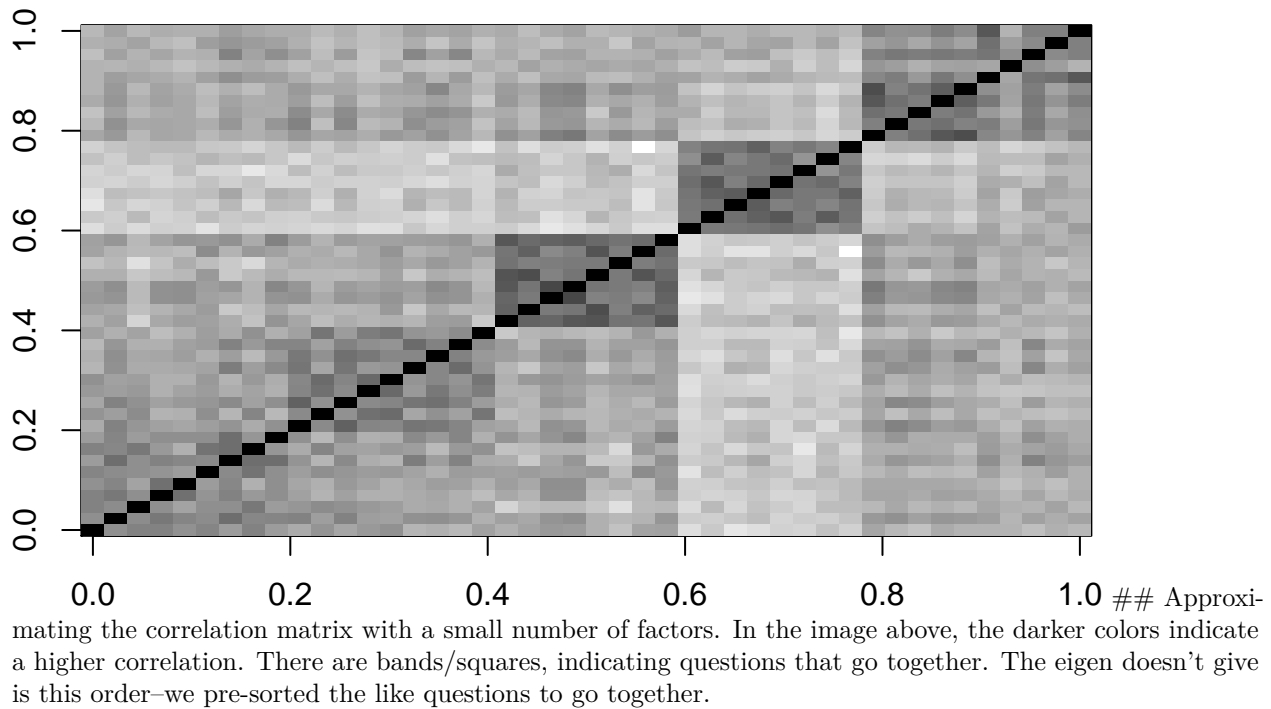
```



```

cbt <- cor(bytype)
image(cbt, col = grey(100:0/100))

```



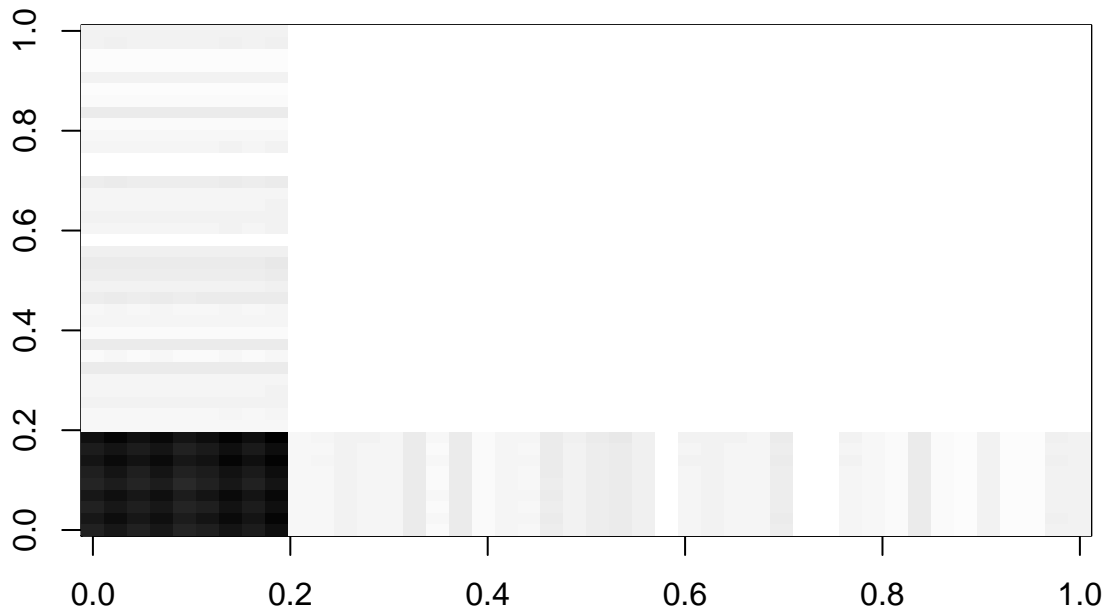
When we use eigen decomposition, we are breaking the correlation into independent elements. We can recompute the main correlational structure from each element by taking the cross product of each factor with itself. Then, by adding these together (in the proportion specified by the eigenvalues), we can re-represent the original correlation matrix. If we have 45 variables, we can then look at how much of the original structure we can reproduce from a small number of factors.

Suppose we had a vector that picked out the each personality factor, one at a time.

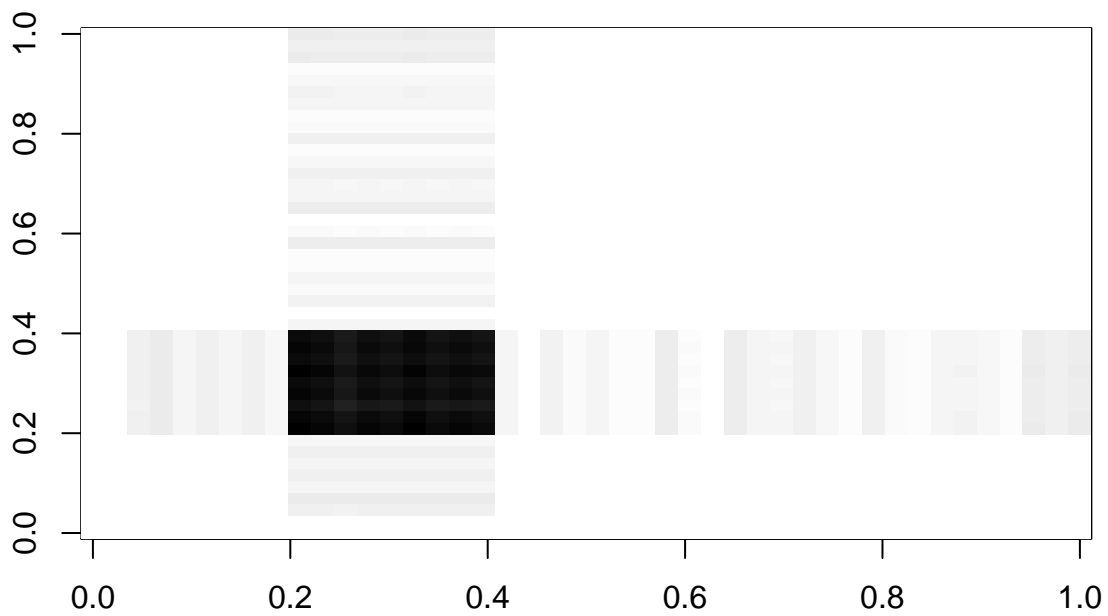
```
length <- length(key)
noise <- 0.1
f.a <- (key == "A") + runif(length(key)) * noise
f.c <- (key == "C") + runif(length(key)) * noise
f.e <- (key == "E") + runif(length(key)) * noise
f.n <- (key == "N") + runif(length(key)) * noise
f.o <- (key == "O") + runif(length(key)) * noise
```

If we compute expected correlation if everything within a group was the same, we can do that by using a matrix product

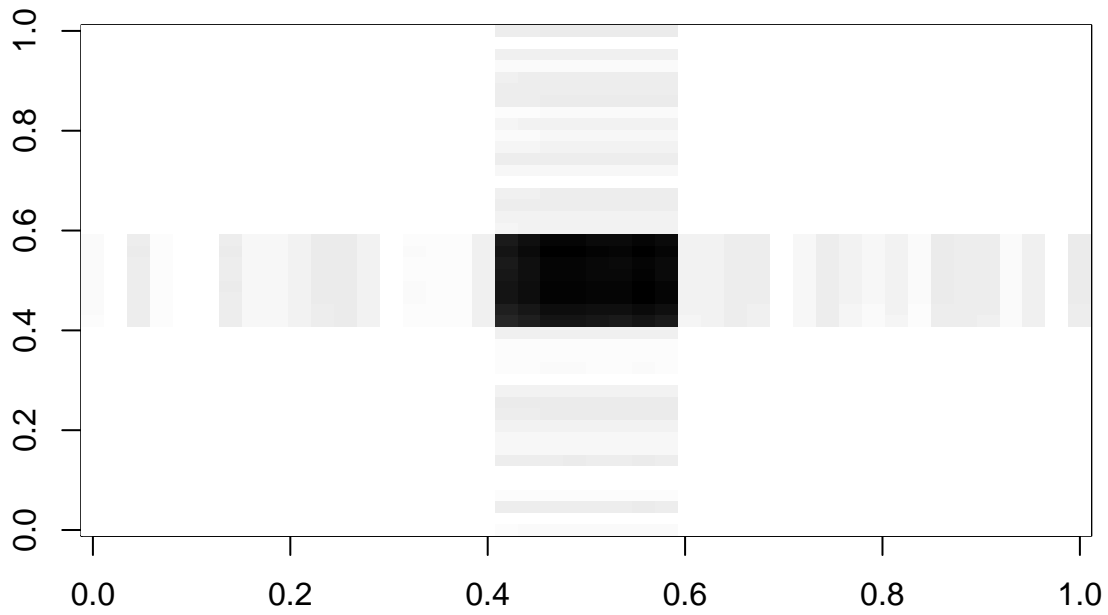
```
cor.a <- f.a %>% t(f.a)
cor.c <- f.c %>% t(f.c)
cor.e <- f.e %>% t(f.e)
cor.n <- f.n %>% t(f.n)
cor.o <- f.o %>% t(f.o)
image(cor.a, col = grey(100:0/100))
```



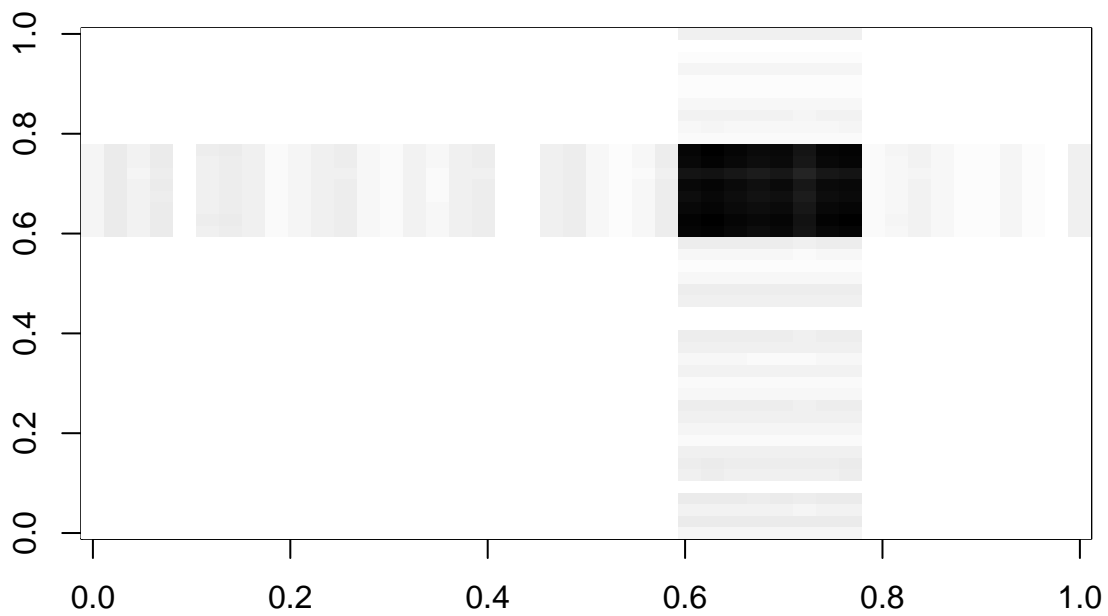
```
image(cor.c, col = grey(100:0/100))
```



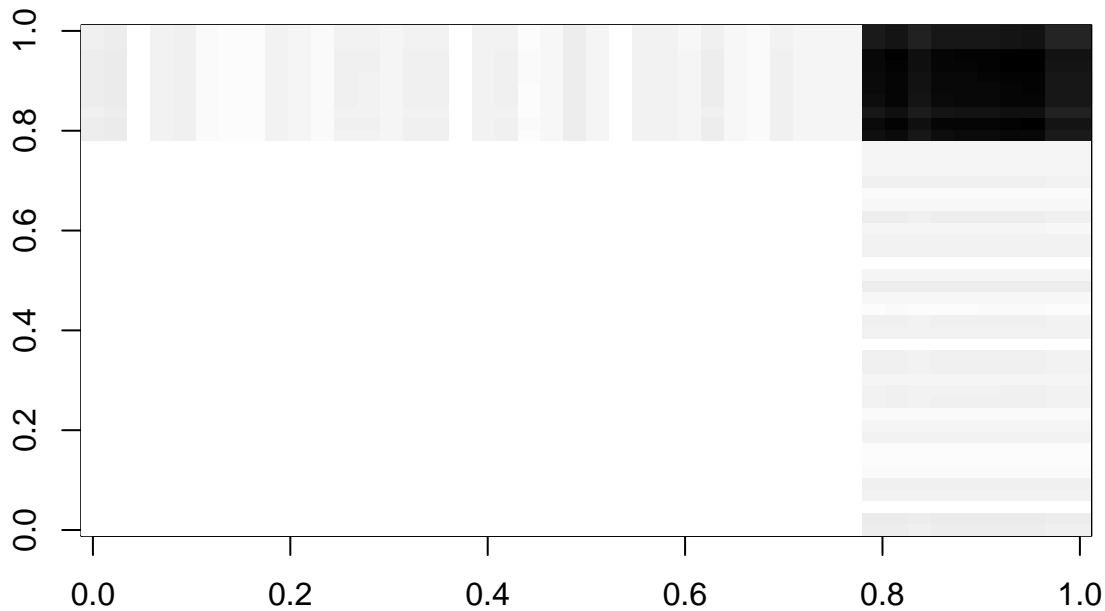
```
image(cor.e, col = grey(100:0/100))
```



```
image(cor.n, col = grey(100:0/100))
```

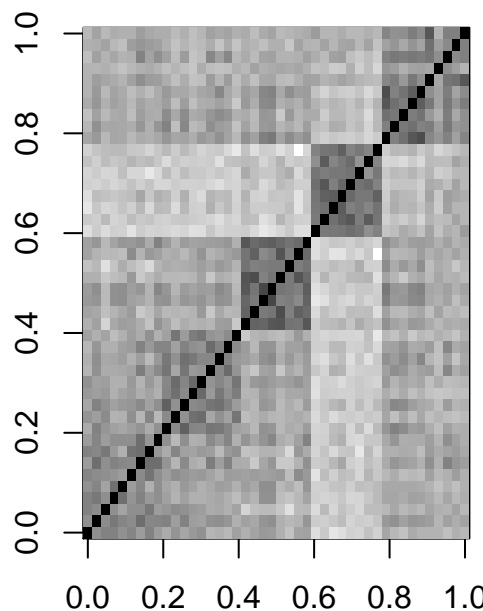
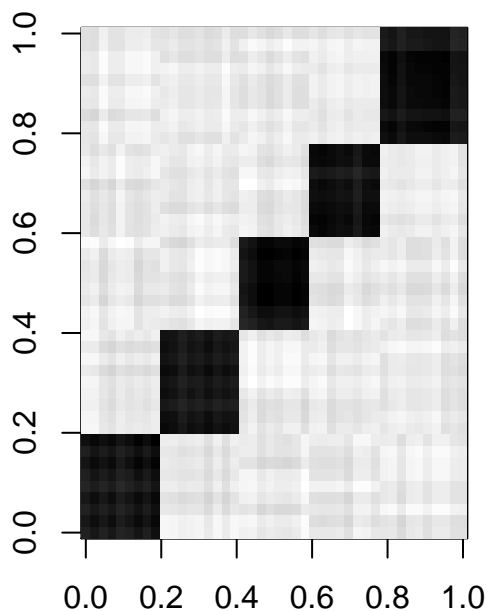


```
image(cor.o, col = grey(100:0/100))
```



If we add each of these up, we approximate the original correlation matrix

```
par(mfrow = c(1, 2))
image(cor.a + cor.c + cor.e + cor.n + cor.o, col = grey(100:0/100))
image(cbt, col = grey(100:0/100))
```



So, on the left is our faked factor with a little noise, which maps exactly onto our big-five theory. On the right is the truth. These resemble one another, although not perfectly. We can do this with the real data and real factors obtained via PCA. That is the basic idea of PCA, and factor analysis in general—to find factors like f.n, f.o, etc that will recombine to model the correlation matrix. PCA finds these factors for you, and the really amazing thing about PCA is that the top few factors will usually reconstruct the matrix fairly well, with the noise being captured by the less important eigenvectors.

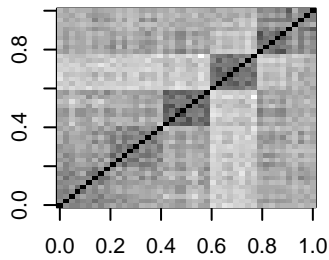
```
e <- eigen(cbt)
par(mfrow = c(2, 3))
image(cbt, col = grey(100:0/100), main = "Complete correlation")
image((e$vector[, 1] * t(e$value[1])) %*% t(e$vector[, 1]), col = grey(100:0/100),
```

```

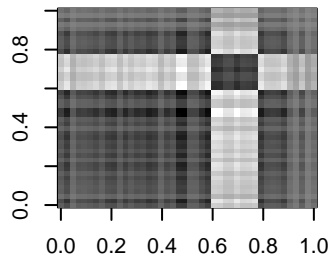
main = "One eigenvector")
image((e$eigenvectors[, 1:2] * (e$values[1:2])) %*% t(e$eigenvectors[, 1:2]), col = grey(100:0/100),
main = "Two eigenvectors")
image((e$eigenvectors[, 1:3] * (e$values[1:3])) %*% t(e$eigenvectors[, 1:3]), col = grey(100:0/100),
main = "Three eigenvectors")
image((e$eigenvectors[, 1:4] * (e$values[1:4])) %*% t(e$eigenvectors[, 1:4]), col = grey(100:0/100),
main = "Four eigenvectors")
image((e$eigenvectors[, 1:5] * (e$values[1:5])) %*% t(e$eigenvectors[, 1:5]), col = grey(100:0/100),
main = "Five eigenvectors")

```

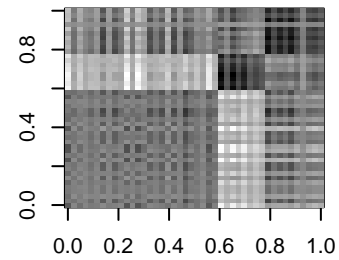
**Complete correlation**



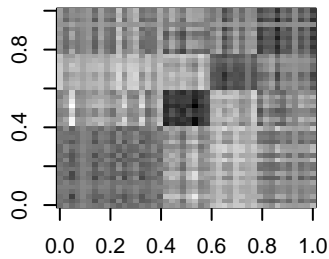
**One eigenvector**



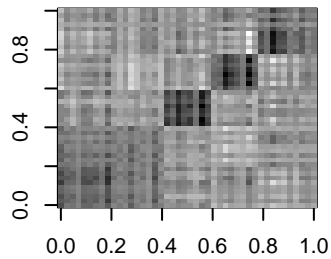
**Two eigenvectors**



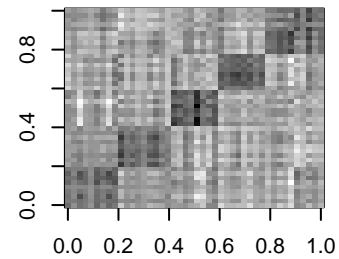
**Three eigenvectors**



**Four eigenvectors**



**Five eigenvectors**

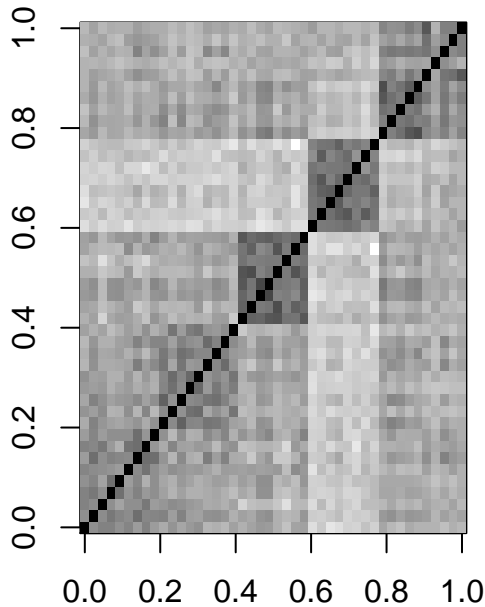


```

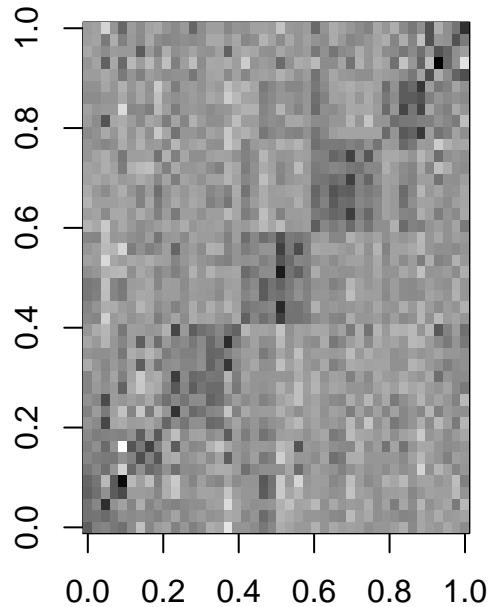
par(mfrow = c(1, 2))
image(cbt, col = grey(100:0/100), main = "Complete correlation")
image((e$eigenvectors[, 1:10] * e$values[1:10]) %*% t(e$eigenvectors[, 1:10]), col = grey(100:0/100),
main = "Ten eigenvectors")

```

### Complete correlation



### Ten eigenvectors



By the time we use five eigenvectors, the matrix gets to be fairly similar to the original. This is 5 out of 44 possible vectors, so we are compressing the data down by a factor of nearly 90%, finding just the important things, and throwing out everything else—treating it as noise. Also shown above is what we get if we go up to 10 factors—not much better than 5.

This might resemble image or audio compression if you are familiar with how these work. In fact, they are quite similar! Projecting to a subspace could alternately be referred to as compressing data.

#### #Eigen decomposition versus SVD

To do the PCA, we essentially did two steps: first computing correlation or covariance, and then eigen decomposition of that covariance matrix. The original data was “bytype”, and the correlation matrix was “cbt”. There is a parallel process that computes a similar decomposition using the raw feature data, called Singular Value Decomposition (SVD). Some PCA or Factor Analysis methods use SVD, and it can be used effectively for very large data sets, where it is intractable to compute a complete correlation matrix.

Let’s compare the eigen decomposition to the SVD on the raw data; using 5-features:

```
factors1 <- eigen(cov(bytype))  
  
## a 15-factor solution using SVD  
factors2 <- svd(bytype, nu = 15)  
summary(factors1)
```

```
      Length Class  Mode  
values    44  -none- numeric  
vectors 1936  -none- numeric
```

```
summary(factors2)
```

```
      Length Class  Mode  
d      44  -none- numeric  
u 15255  -none- numeric  
v  1936  -none- numeric
```

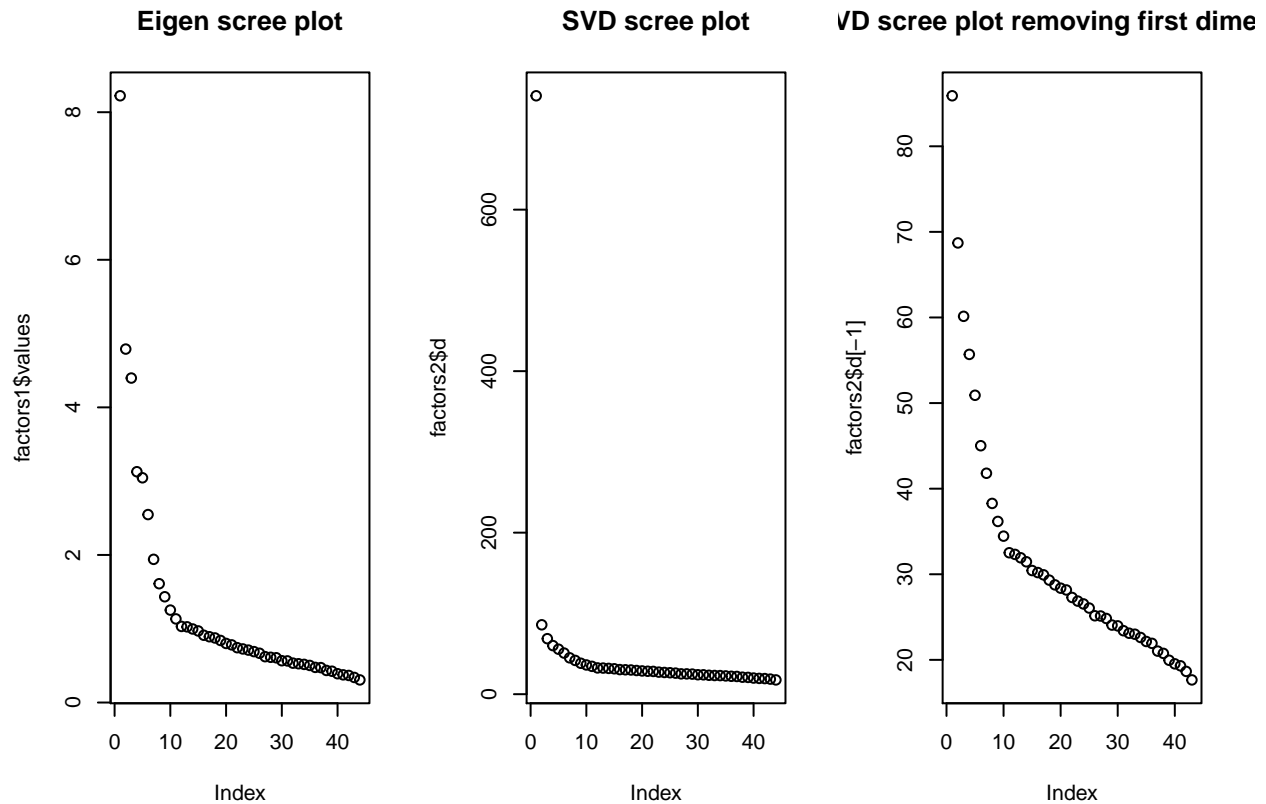
The eigen decomposes the square matrix into a vector 44 long and a square matrix. The svd decomposes into



a vector and two rectangular matrices.

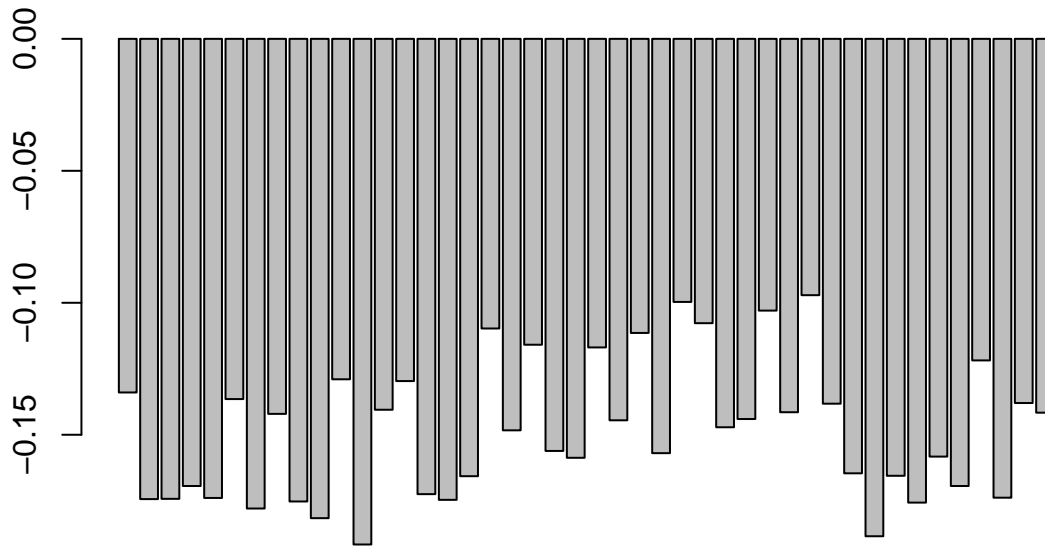
Here is the ‘scree’ plot for each–eigen values (variance) and the svd d matrix. Notice that the first factor of the SVD is much larger; this is in part because we have already normalized everything in PCA by using correlation. So the first dimension is essentially an ‘intercept’ term, which tries to get the overall level right. Removing this, the two screeplots look fairly similar.

```
par(mfrow = c(1, 3))
plot(factors1$values, main = "Eigen scree plot")
plot(factors2$d, main = "SVD scree plot")
plot(factors2$d[-1], main = "SVD scree plot removing first dimension")
```



The first dimension of SVD is not very interesting here–just an intercept and because all terms were relatively similar, they do not differ by question:

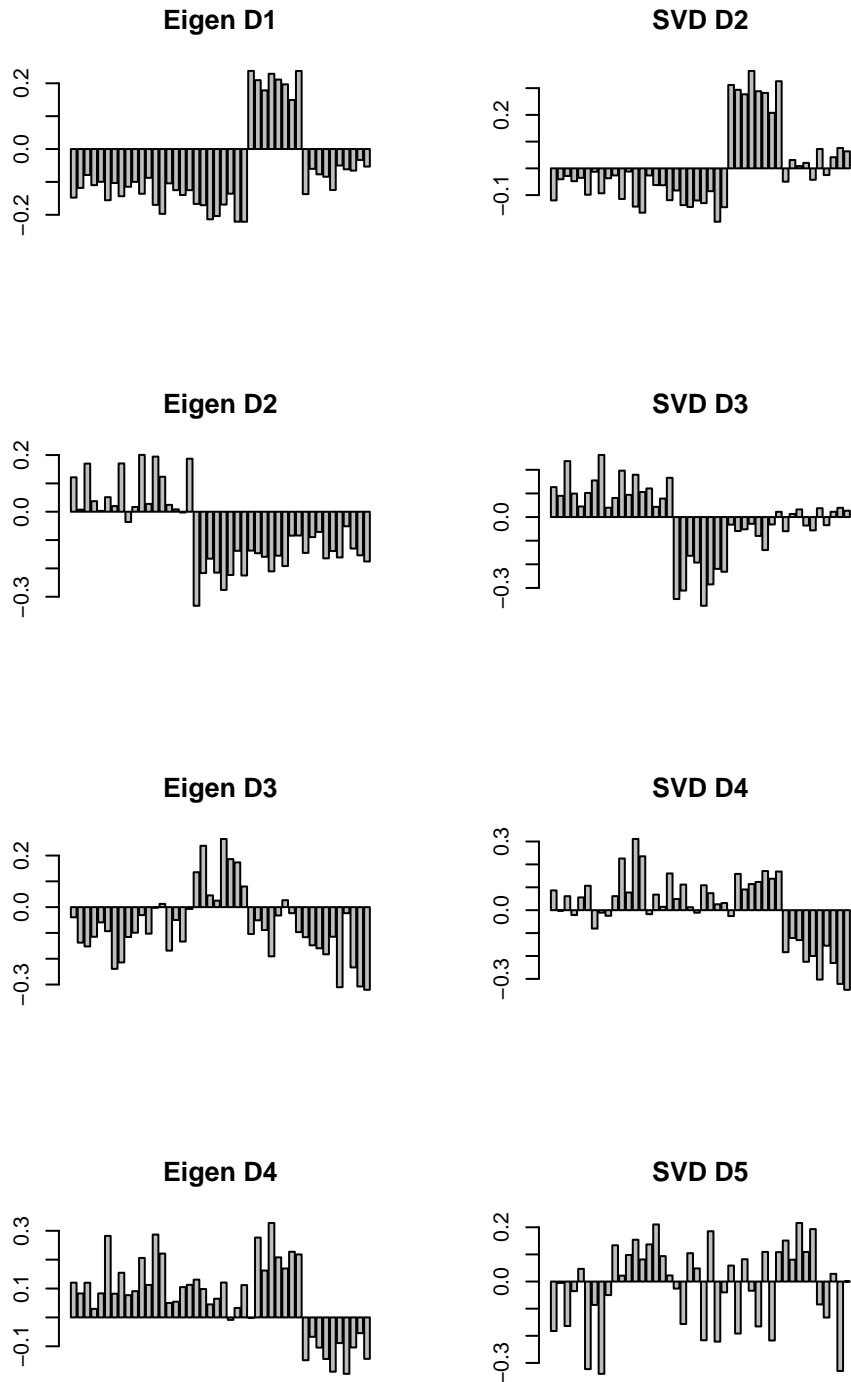
```
barplot(factors2$v[, 1])
```



Following this, the remaining terms of SVD map onto the terms of eigen decomposition fairly well.

```
par(mfcol = c(4, 2))
barplot(factors1$ectors[, 1], main = "Eigen D1")
barplot(factors1$ectors[, 2], main = "Eigen D2")
barplot(factors1$ectors[, 3], main = "Eigen D3")
barplot(factors1$ectors[, 4], main = "Eigen D4")
```

```
barplot(factors2$v[, 2], main = "SVD D2")
barplot(factors2$v[, 3], main = "SVD D3")
barplot(factors2$v[, 4], main = "SVD D4")
barplot(factors2$v[, 5], main = "SVD D5")
```



The results are not identical—computing the correlation of the data does normalization that removes some information that SVD uses, but the results are similar up to the first few dimensions.

We can calculate the correlation between the first ten factors of each.

```
round(cor(factors1$vector, factors2$v), 3)[1:10, 1:10]
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0.339	0.985	0.018	0.120	-0.033	-0.050	-0.050	-0.044	-0.034
[2,]	-0.116	-0.312	0.875	0.330	-0.026	-0.061	-0.087	-0.083	-0.044
[3,]	0.441	-0.345	-0.685	0.593	-0.021	-0.074	-0.124	-0.122	-0.055

```

[4,]  0.341  0.188  0.117  0.865 -0.266  0.124  0.209  0.192  0.070
[5,] -0.408  0.054  0.027  0.203  0.973  0.030  0.059  0.056  0.016
[6,]  0.511  0.012  0.013  0.027  0.015 -0.998  0.041  0.031  0.013
[7,]  0.147  0.033  0.019  0.047  0.016 -0.004 -0.950  0.286  0.042
      [,10]
[1,]  0.032
[2,]  0.052
[3,]  0.072
[4,] -0.099
[5,] -0.027
[6,] -0.014
[7,] -0.063
[ reached getOption("max.print") -- omitted 3 rows ]

```

Notice that when we correlate the two factor solutions, there are a number of high correspondences: E1 versus S2; E2 vs S4, E3 vs -S3, etc. However, the correspondence is not exact, for a number of reasons. First, SVD solution will solve just the top n dimensions; which has the effect of ‘compressing’ into fewer dimension. Also, eigen decomposition throws out some data by using the correlation (and even the covariance) matrix.

By doing appropriate scaling, we can get much more similar (and essentially identical) values from the two methods.

## Choosing between SVD and Eigen decomposition

In one sense, you never have to choose between these methods; eigen decomposition requires a square matrix, and SVD a rectangular matrix. If you have a square matrix (a distance or correlation matrix), then you use eigen decomposition; otherwise you might try SVD.

But if you have the rectangular matrix (data x features), you might choose to create the correlation matrix and then do eigen decomposition on that. This is a fairly typical approach, and it has the advantage that some factor analytic approaches will require this correlation/covariance structure as input, and permit others to easily understand what you are doing.

However, there are times when computing the complete correlation matrix is not feasible, and you would prefer to go directly from an itemxfeature matrix to a small set of implied features. One of the greatest successes of this approach is with a method called ‘Latent Semantic Analysis’. LSA does SVD on large termxdocument data matrices. For example, imagine a text corpus with 1 million documents (rows) and 200,000 words or terms (columns). Computing the correlation of words by documents may be impossible—it is a matrix with 40 billion entries. And computing eigen decomposition on a 200,000 x 200,000 matrix may never finish. But reasonable implementation of SVD (ones available to academics in the 1980s and 1990s) could solve the SVD with 100-300 dimensions. LSA turns out to create a nice semantic space—words that tend to appear together in documents tend to have similar meaning, and it turns the co-occurrences into a feature space.

## Summary

Within R, there are a number of related function that do PCA, and a number of decisions about how to do it.

- Using eigen on either covariance or correlation matrix
- using svd on raw data (possibly normalizing/scaling variables)
- princomp, which uses eigen, with either covariance or correlation matrix
- prcomp, which uses svd, with or without scaling variables

- principal within the psych:: package does eigen decomposition and provides more readable output and goodness of fit statistics, with loadings rescaled by eigenvalues. This also permits using alternate rotations, which we will discuss in factor analysis sections.
- factoextra functions (mostly useful for special-purpose graphing using ggplot2 and special analytics)
- Routines in other libraries such as ade4, factomineR, and epPCA in ExPosition, which focuses on SVD.

In general, the different methods can probably all be made to give identical results with the right scaling, centering, and other assumptions. However, choosing between them depends on several factors:

- Ease and value of interpretation, output, and statistics
- Computational feasibility of computing eigen decomposition and correlation/covariance matrix on large data sets
- If there are too few observations with respect to the number of questions, the correlation matrix across questions will be rank-deficient: it will necessarily contain fewer principal components than observations. In these cases, the prcomp svd will be able to produce a solution (although that solution might not be worth much because of the low number of observations).

Finally, principal components analysis useful for data processing, feature reduction, and the like; but for a more statistical analysis, scientists usually prefer a more complex set of related routines known colloquially as exploratory factor analysis.