2. Singular Value Decomposition

Lieven Clement

statOmics, Ghent University (https://statomics.github.io)

Contents

1 Introduction

1.1 Motivation

The SVD is one of the most well used and general purpose tools from linear algebra for data processing!

Methodologically

- Dimension reduction (e.g. images, gene expression data, movie preferences)
- Used as a first step in many data reduction and machine learning approaches
- Taylor a coordinate system based on the data we have
- Solve system of linear equations for non-square matrices: e.g. linear regression
- Basis for principal component analysis (PCA) and multidimensional scaling (MDS).
  - PCA is one of the most widely used methods to study high dimensional data and to understand them in terms of their dominant patterns and correlations

Applications:

- At the heart of search engines: Google
- Basis of many facial recognition methods: e.g. Facebook
- Recommender systems such as Amazon and Netflix
- A standard tool for data exploration and dimension reduction in Genomics

1.2 Disclaimer

When you want to run the script you will have to comment out the eval=FALSE statement in some R chunks. Because the SVD takes a while on the faces example we save the svd for later use. So you have to comment the eval=FALSE statement in this chunk when you run the script for the first time.

1.3 Data

- Extended Yale Face Database B
- Cropped and aligned images of 38 individuals under 64 lighting conditions.
- Each image is 192 pixels tall and 168 pixels wide.
- Each of the facial images in our library will be reshaped into a large vector with $192 \times 168 = 32\ 256$ elements.
- We will use the 64 images of 36 people to build our models

```
library(pixmap)
library(tidyverse)
library(gridExtra)
library(grid)
library(ggmap)
library(downloader)
library(imager)
```

```
## Download and unzip data
if(!dir.exists("raw-data")) dir.create("raw-data")
download(
  "https://github.com/statOmics/HDA2020/raw/data/yalefaces_cropped.zip",
  destfile = "raw-data/yalefaces_cropped.zip", mode = "wb", quiet = TRUE
)
unzip ("raw-data/yalefaces_cropped.zip", exdir = "./raw-data")

dir <- "./raw-data/CroppedYale"
```

```
people <- list.files(dir)
people2 <- sapply(people,
  function(x) list.files(
    paste0(dir,"/",x),
    full.names=TRUE
    )
  )

facesList <- lapply(people2, function(x) read.pnm(x))

 grid.arrange(
  grobs=lapply(facesList[1+(0:35)*64],
     function(x) getChannels(x) %>%
        ggimage(.,coord_equal=TRUE)
        ),
  ncol=6)
```

1.4 Method

Let $\mathbf{X}$ be an $n \times p$ matrix e.g.

- gene expression of $p = 40000$ genes for $n = 30$ subjects
- n = 100 000 000 webpages indexed with p search terms, or
- n images each stored as a $p = 32256$ vector with the intensity of each pixel

Note: the emoji characters will not be visible in the PDF output.

$$X = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_i^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n \times p}$$

The data matrix $\mathbf{X}$ can be decomposed with the SVD into 3 matrices:

$$\mathbf{X} = \mathbf{U}_{n \times n} \Delta_{n \times p} \mathbf{V}_{p \times p}^T$$

- an orthonormal matrix $\mathbf{U}_{n \times n}$ with left singular vectors: $\mathbf{u}_j^T \mathbf{u}_k = 1$ if $k = j$ and $\mathbf{u}_j^T \mathbf{u}_k = 0$ if $j \neq k$, i.e.

$$\mathbf{U}^T \mathbf{U} = \mathbf{I}$$

- a matrix $\Delta_{n\times p}$ with only singular values: the singular values δ_i are the only non-zero elements of the matrix and are on the diagonal element $[\Delta]_{ii}$. They are also organised so that $\delta_1 > \delta_2 > ... > \delta_r$.
- an orthonormal matrix $\mathbf{V}_{p\times p}$ with right singular vectors: $\mathbf{v}_j^T\mathbf{v}_k = 1$ if $k = j$ and $\mathbf{v}_j^T\mathbf{v}_k = 0$ if $j \neq k$ otherwise, i.e.

$$\mathbf{V}^T\mathbf{V} = \mathbf{I}$$

Note, that there are only r non-zero singular values, with r the rank of matrix X: $r \leq \min(n, p)$. So we have $k = 1 ... r$ non-zero singular values. Hence, we can also rewrite the approximation by restricting us to the rank of matrix $\mathbf{X}$. Indeed, the n times p matrix Δ only contains r non-zero diagonal elements!

- So

$$\mathbf{X} = \mathbf{U}_{n\times r}\Delta_{r\times r}\mathbf{V}^T_{p\times r}$$

$$\begin{bmatrix} - & \mathbf{x}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_i^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times p} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & ... & \mathbf{u}_r \\ | & & | \end{bmatrix}_{n\times r} \begin{bmatrix} \delta_1 & & \\ & \ddots & \\ & & \delta_r \end{bmatrix}_{r\times r} \begin{bmatrix} | & & | \\ \mathbf{v}_1 & ... & \mathbf{v}_r \\ | & & | \end{bmatrix}^T_{p\times r}$$

Also note that

$$\mathbf{V}^T = \begin{bmatrix} - & \mathbf{v}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{v}_r^T & - \end{bmatrix}_{r\times p}$$

- For high dimensional data $p >>> n \rightarrow \max(r) = n$ and
- equivalently for multivariate data with $n > p \rightarrow \max(r) = p$

We can also rewrite the decomposition using the properties of matrix multiplication

$$\mathbf{X} = \delta_1 \begin{bmatrix} | \\ \mathbf{u}_1 \\ | \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \end{bmatrix} + ... + \delta_r \begin{bmatrix} | \\ \mathbf{u}_r \\ | \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_r^T & - \end{bmatrix} \quad (1)$$

$$\mathbf{X} = \sum_{k=1}^{r} \delta_k \mathbf{u}_k \mathbf{v}_k^T \quad (2)$$

- Because both $\mathbf{U}$ and $\mathbf{V}$ are orthonormal all their r vectors are having unit length and they are thus reshaped by the singular values.
- Hence, the singular values determine the importance of the rank one matrices $\delta_k\mathbf{u}_k\mathbf{v}_k^T$ in the reconstruction of the matrix $\mathbf{X}$ and they are ordered so that $\delta_1 > ... > \delta_r$.

Note, that for symmetric matrices $\mathbf{X} \longrightarrow \mathbf{U} = \mathbf{V}$.

1.5 Interpretation of singular vectors: face example

1.5.1 Convert images to vectors

1. Convert images to vectors and store them as a matrix
    - We use an `sapply` loop to loop over all faces
    - We extract the grey intensities from the pictures
    - We convert the matrix in a long skinny vector (`c`)
    - We transpose the resulting matrix from sapply

```
allFacesMx <- sapply(facesList,
                     function(x)
                        getChannels(x) %>% c
                     ) %>% t
dim(allFacesMx)
```

```
## [1]  2432 32256
```

Save memory by removing facesList object

```
rm(facesList)
gc()
```

Hence we obtain a matrix for n = 2432 images with p = 32256 intensities for each pixel of an image.

Before we do the svd we typically center the data by substracting the average of the columns, i.e. the average face.

We will only work with the first 36 people: $n = 36 \times 64 = 2304$ pictures.

```
allFacesCenteredMx <- allFacesMx[1:(36*64),]
meanFace <- colMeans(allFacesCenteredMx)

allFacesMxCentered <- allFacesCenteredMx -
  matrix(1, nrow=nrow(allFacesCenteredMx), ncol=1) %*% matrix(meanFace,nrow=1)
```

1.5.2 Visualisation of mean image

```
plotFaceVector <- function(faceVector,nrow=192,ncol=168) {
  matrix(faceVector,nrow=nrow,ncol=ncol) %>%
  ggimage()
}

meanFace %>%
  plotFaceVector
```

1.5.3 SVD

1.5.3.1 Perform SVD in R

1. We adopt svd on the centered matrix
2. We cache the result because the calculation takes 10 minutes.

```
faceSvd <- svd(allFacesMxCentered)
```

```
## Run this code manually to store the SVD for later re-use
saveRDS(faceSvd, file = "faceSvd.rds")
```

```
## Run this code manually to reload the SVD result
faceSvd <- readRDS("faceSvd.rds")
```

1.5.3.2 SVD Dimensions of **U**, **V**?

```
n <- nrow(allFacesCenteredMx)
p <- ncol(allFacesCenteredMx)
dim(faceSvd$u)
```

```
## [1] 2304 2304
```

```
dim(faceSvd$v)
```

```
## [1] 32256  2304
```

Indeed, for the face example $n < p$ so $r = n$

Check orthogonality?

We do not do it for all vectors because it takes too long. First left eigen vector and second left eigenvector. Happens in $\mathbf{U}^T\mathbf{U}$

```
t(faceSvd$u[,1])%*%faceSvd$u[,1]
```

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

```
t(faceSvd$u[,1])%*%faceSvd$u[,2]
```

```
##                [,1]
## [1,] -3.794708e-19
```

```
t(faceSvd$u[,2])%*%faceSvd$u[,2]
```

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

So we see that the left eigenvectors are orthonormal.

We check if it also holds for the rows i.e. $\mathbf{U}\mathbf{U}^T$

```
t(faceSvd$u[1,])%*%faceSvd$u[1,]
```

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

```
t(faceSvd$u[2,])%*%faceSvd$u[1,]
```

```
##                [,1]
## [1,] -1.691355e-16
```

```
t(faceSvd$u[2,])%*%faceSvd$u[2,]
```

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

We also see that the rows of $\mathbf{U}$ are orthonormal.

```
t(faceSvd$v[,1])%*%faceSvd$v[,1]
```

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

```
t(faceSvd$v[,1])%*%faceSvd$v[,2]
```

```
##               [,1]
## [1,] -9.397551e-16
```

```
t(faceSvd$v[,2])%*%faceSvd$v[,2]
```

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

So we see that the right eigenvectors are orthonormal.

```
t(faceSvd$v[1,])%*%faceSvd$v[1,]
```

```
##           [,1]
## [1,] 0.6181993
```

```
t(faceSvd$v[1,])%*%faceSvd$v[2,]
```

```
##            [,1]
## [1,] 0.07690404
```

```
t(faceSvd$v[2,])%*%faceSvd$v[2,]
```

```
##           [,1]
## [1,] 0.1017995
```

This, however does not hold for the rows of $\mathbf{V}$. This is because the matrix $\mathbf{V}$ no longer is a square matrix! $r = n$ and $r < p$!

```
grid.arrange(
 grobs=apply(
   faceSvd$v[,1:36],
   2,
   plotFaceVector
   )
 )
```

1.5.3.3 Visualize right eigenvectors V

```
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
## rescaling mat to [0,1]...
```

- Hence, the right singular vectors (in $\mathbf{V}$ of $\mathbf{X} = \mathbf{U}\Delta\mathbf{V}$) are also faces and we can thus reconstruct the original faces by linear combinations of the eigen faces.
- The first eigen faces are most important to capture overall patterns in the matrix.
- Here it are mainly characteristics and shadows that are important for all faces.
- From eigen face 5 onwards we start to see specific features.
- In this case: $n < p$, so $r = n$.

$$\mathbf{X}_{n\times p} = \mathbf{U}_{n\times n}\Delta_{n\times n}\mathbf{V}^T_{p\times n}$$

$$\begin{bmatrix} - & \mathbf{x}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_i^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times p} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_n \\ | & & | \end{bmatrix}_{n\times n} \begin{bmatrix} \delta_1 & & \\ & \ddots & \\ & & \delta_n \end{bmatrix}_{n\times n} \begin{bmatrix} | & & | \\ \mathbf{v}_1 & \dots & \mathbf{v}_n \\ | & & | \end{bmatrix}^T_{p\times n}$$

- Or upon transposing the matrix $\mathbf{V}$

$$\begin{bmatrix} - & \mathbf{x}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_i^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times p} = \begin{bmatrix} | & & | \\ \mathbf{u}_1 & \dots & \mathbf{u}_n \\ | & & | \end{bmatrix}_{n\times n} \begin{bmatrix} \delta_1 & & \\ & \ddots & \\ & & \delta_r \end{bmatrix}_{n\times n} \begin{bmatrix} - & \mathbf{v}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{v}_p^T & - \end{bmatrix}_{n\times p}$$

1.5.3.4 Reconstruction of faces via linear combination of eigen faces. In left singular vectors u_{ij} we quantify the contribution of the j^{th} eigenface in the reconstruction of face i and we rescale the importance of each eigen face by its corresponding eigen value δ_j.

$$\begin{bmatrix} - & \mathbf{x}_1^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_i^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_n^T & - \end{bmatrix}_{n\times p} = \delta_1 \begin{bmatrix} | \\ \mathbf{u}_1 \\ | \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1^T & - \end{bmatrix} + \dots + \delta_r \begin{bmatrix} | \\ \mathbf{u}_r \\ | \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_r^T & - \end{bmatrix}$$

If we truncate the eigen faces say at $k < r$ we can approximate faces using a limited number of eigen faces!

```
approximateFace <- function(meanFace,faceSvd,k){
  reconstruct <- (meanFace + faceSvd$u[1,1:k] %*%
    diag(faceSvd$d[1:k]) %*%
  t(faceSvd$v[,1:k]) %>%
  c)
}

approxHlp <- sapply(
      c(25,100,500),
      approximateFace,
      meanFace=meanFace,
      faceSvd=faceSvd)

grid.arrange(
 grobs=apply(
   cbind(
     approxHlp,
     allFacesMxCentered[1,]+meanFace
   ),
   2,
   plotFaceVector
   )
 )
```

2 SVD as a Matrix Approximation Method

- We have seen that we can use the truncted SVD to approximate matrix $\mathbf{X}$ by $\tilde{\mathbf{X}}$, with $k < r$ and

$$\tilde{\mathbf{X}} = \mathbf{U}_{n\times k}\Delta_{k\times k}\mathbf{V}_{p\times k}^T$$

- It can be shown that **SVD: optimal approximation**

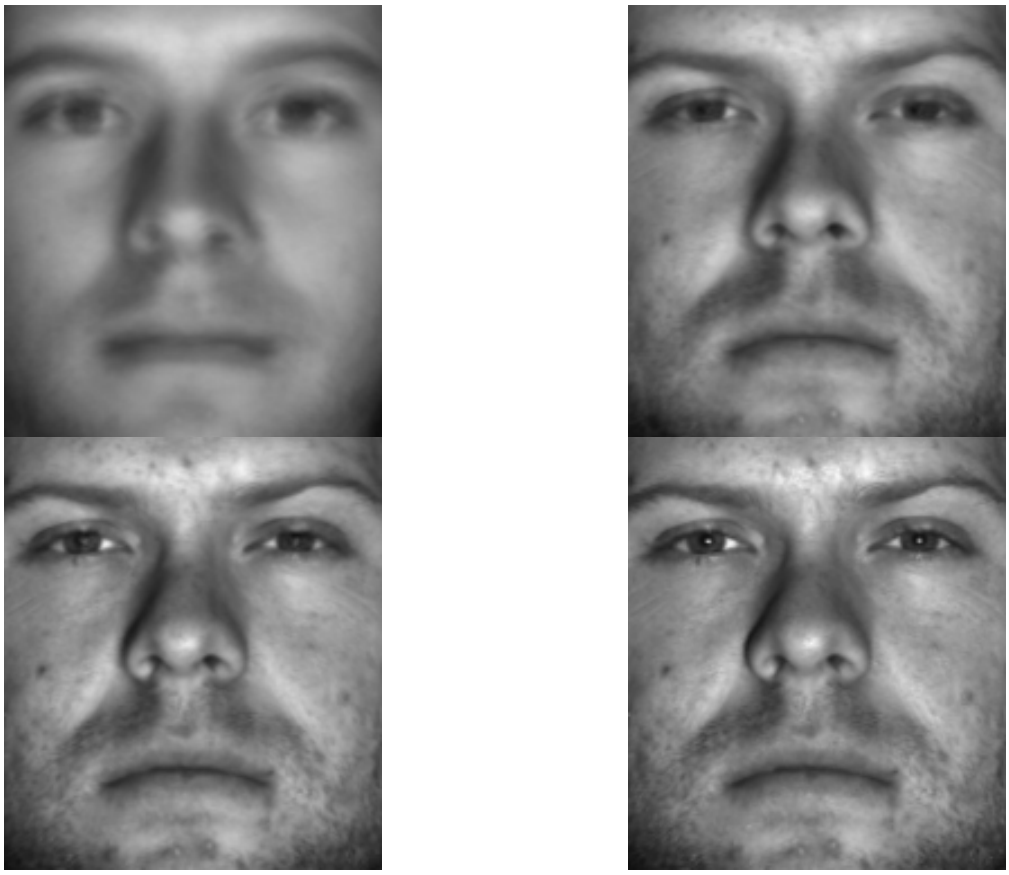


Figure 1: approximation with 25 (top left), 100 (top right) and 500 (bottom left) eigenfaces and original face (bottom right, or with all eigenfaces)

- Let $\mathbf{X}$ be an $n \times p$ matrix of rank $r \leq \min(n, p)$, and let $\mathbf{A}$ denote an $n \times p$ matrix of rank $k \leq r$, with elements denoted by a_{ij}.
- The matrix $\mathbf{A}$ of rank $k \leq r$ that minimises the Frobenius norm

$$||\mathbf{X} - \mathbf{A}||_{\text{fr}}^2 = \sum_{i=1}^{n} \sum_{j=1}^{p} (x_{ij} - a_{ij})^2$$

  is given by the truncated SVD

$$\mathbf{X}_k = \sum_{j=1}^{k} \delta_j \mathbf{u}_j \mathbf{v}_j^T.$$

- The truncated SVD has $k < r$ terms. Hence, generally $\mathbf{X}_k$ does not coincide with $\mathbf{X}$. It is considered as an approximation.
- Note, that the truncated SVD thus approximates the matrix by minimising a kind of sum of least squared errors between the elements of matrix $\mathbf{X}$ and $\mathbf{A}$ and that
- the truncated SVD $\mathbf{X}_k$ is the best rank-k approximation of $\mathbf{X}$ in terms of this Frobenius norm.
- Also, note that upon truncation

$$\mathbf{V}_{p\times k}^T \mathbf{V}_{p\times k} = \mathbf{I}_{k\times k}$$
$$\mathbf{U}_{n\times k}^T \mathbf{U}_{n\times k} = \mathbf{I}_{k\times k}$$

- But, that

$$\mathbf{V}_{p\times k} \mathbf{V}_{p\times k}^T \neq \mathbf{I}_{p\times p}!!!$$
$$\mathbf{U}_{n\times k} \mathbf{U}_{n\times k}^T \neq \mathbf{I}_{n\times n}!!!$$

Some **informal statement** about the truncated SVD

$$\mathbf{X}_k = \sum_{j=1}^{k} \delta_j \mathbf{u}_j \mathbf{v}_j^T.$$

- It can be considered as a weighted sum of matrices $\mathbf{u}_j \mathbf{v}_j^T$, with weights δ_j.
- The terms are ordered with decreasing weights $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_k > 0$.
- The matrices $\mathbf{u}_j \mathbf{v}_j^T$ are of equal "magnitude" (constructed from normalised vectors).
- Truncation at k results in k δ_j's, $k \times n$ elements in the $\mathbf{u}_j$ and $k \times p$ elements in the $\mathbf{v}_j$. Hence a total of $k + kn + kp = k(1 + n + p)$ elements (usually much smaller than np). (Note that restrictions apply to $\mathbf{u}_j$ and $\mathbf{v}_j$; hence even less independent elements).

$\longrightarrow$ **data compression**

2.1 Example 1: Image compression

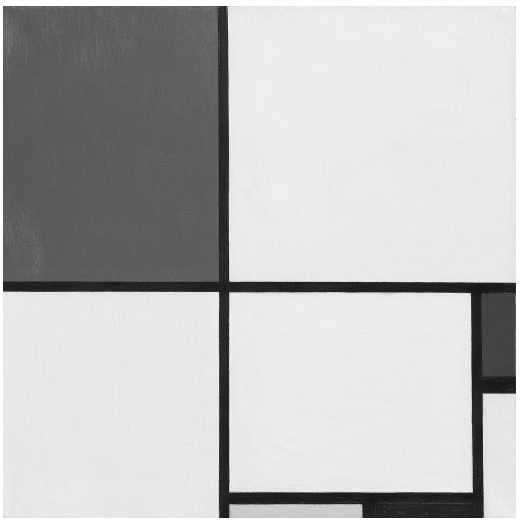
2.1.1 Painting Mondriaan: Composition_No.III with red, blue, yellow and black (1929).

- Have a look at this painting of Mondriaan (1872 – 1944), here shown in black-and-white.

2.1.1.1 Load the original painting

1. fetch image from the web
2. convert into greyscale
3. plot
4. save as Matrix

```
mondriaan <- load.image("https://upload.wikimedia.org/wikipedia/commons/thumb/a/ac/Piet_Mondrian_-_Compo
mondriaan <- grayscale(mondriaan)
plot(mondriaan,axes=FALSE)
```

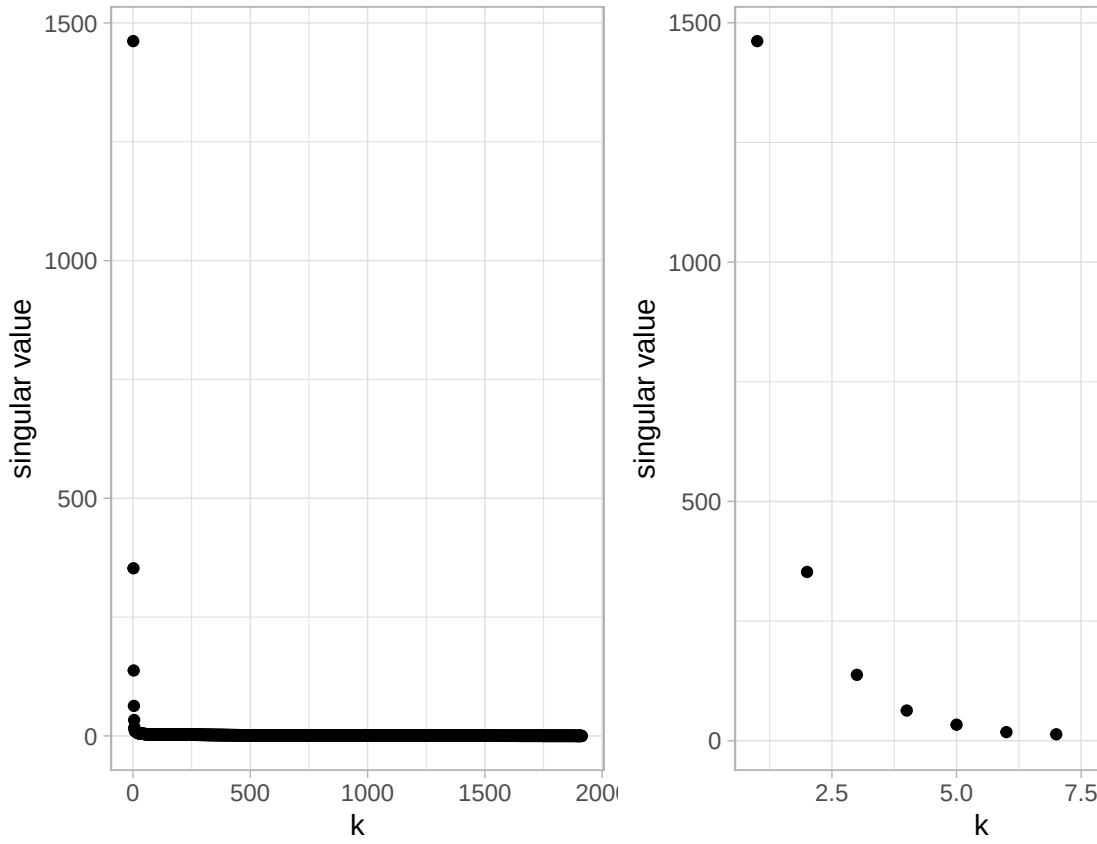


```
X <- matrix(as.data.frame(mondriaan)[,3],nrow=nrow(mondriaan),ncol=ncol(mondriaan))
```

- This picture can be represented as a 1920×1913` matrix $\mathbf{X}$ with gray scale intensities $\in [0, 1]$. ($\approx 4 \times 10^6$ data entries)
- We will here not transform the image in a vector, but will look at the performance of the SVD to compress this image. The SVD can be applied to any matrix!

```
monSvd <- svd(X)

p1 <- data.frame(x=1:length(monSvd$d),y=monSvd$d) %>%
```

```
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

p2 <- data.frame(x=1:10,y=monSvd$d[1:10]) %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

grid.arrange(p1,p2,nrow=1)
```



2.1.1.2 Singular values

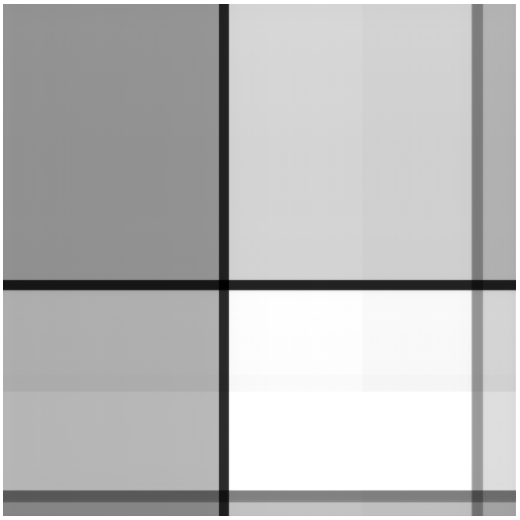
- The singular values decay very quickly!

2.1.1.3 Data compression

- We make the plot for a reconstruction with 1 singular vector. This leads to a data compression of $1 - \frac{(1+1920+1913)}{1920 \times 1913} = 99.9\%$. We only use 1 left singular vector (1920), 1 eigen value, 1 right singular vector (1913).

```
k <- 1
approxMon <- monSvd$u[,1:k] %*%
  diag(monSvd$d[1:k],ncol=k) %*%
```

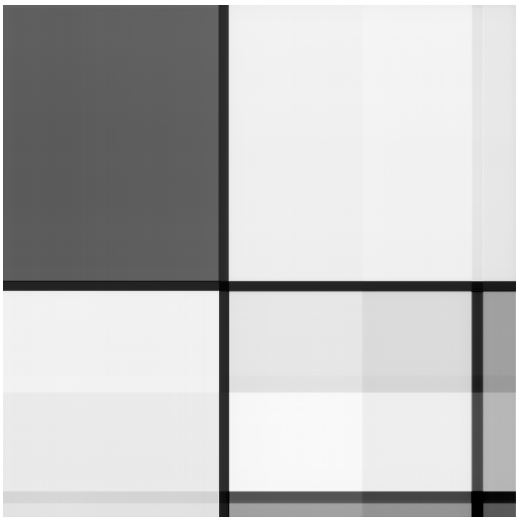
```
  t(monSvd$v[,1:k])

approxMon[approxMon < 0] <- 0
approxMon[approxMon > 1] <- 1

as.cimg(approxMon) %>%
  plot(.,main=paste0("Approximation with ",k," singular vectors"),axes=FALSE)
```



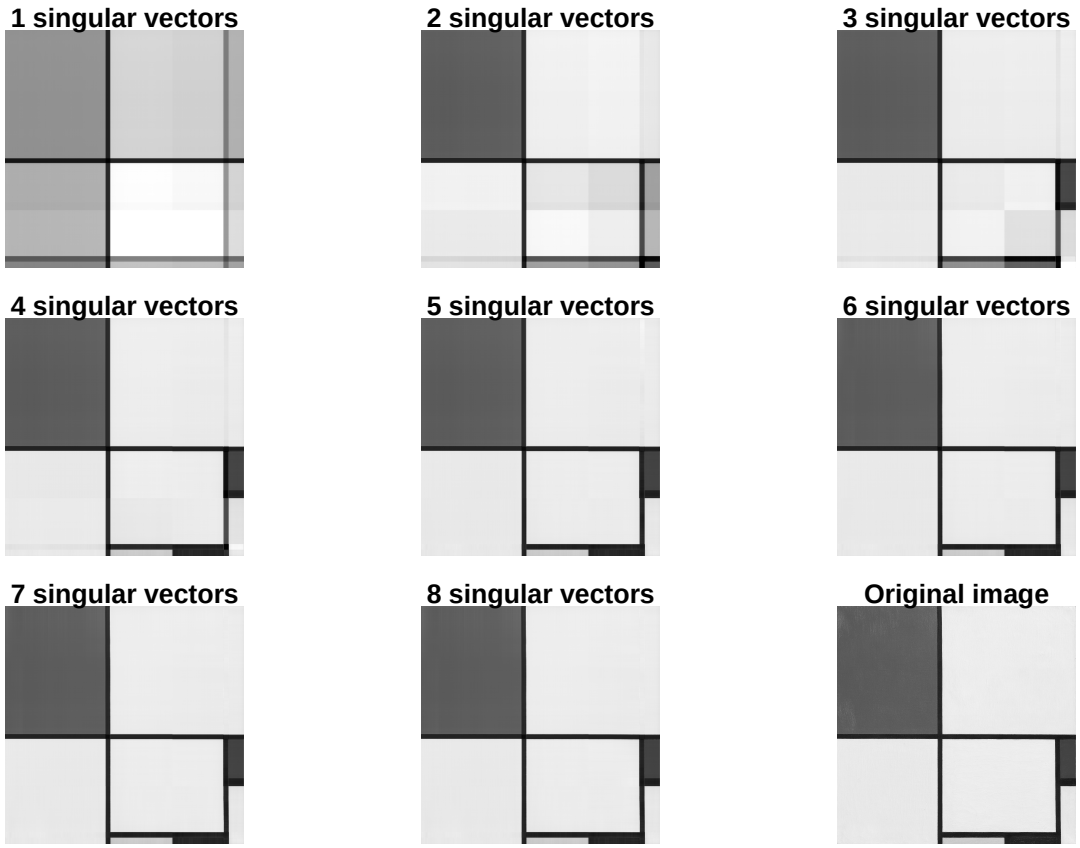
- We make the plot for a reconstruction with 2 singular vector. This leads to a data compression of $1 - \frac{2\times(1+1920+1913)}{1920\times1913} = 99.8\%$. We only use 2 left singular vectors (2×1920), 2 singular values, 2 right singular vectors (2×1913).

```
k <- 2
approxMon <- monSvd$u[,1:k] %*%
  diag(monSvd$d[1:k],ncol=k) %*%
  t(monSvd$v[,1:k])

approxMon[approxMon < 0] <- 0
approxMon[approxMon > 1] <- 1

as.cimg(approxMon) %>%
  plot(.,main=paste0("Approximation with ",k," singular vectors"),axes=FALSE)
```

Approximation with 2 singular vectors

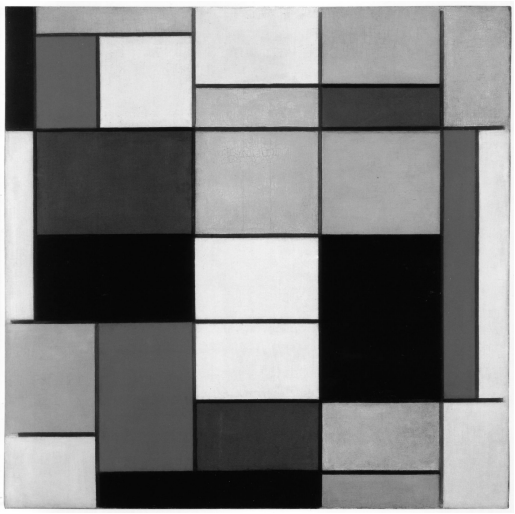


```
par (mfrow=c(3,3))
par(mar=c(1,2,1,1))
for (k in c(1:8))
{
approxMon <- monSvd$u[,1:k] %*%
  diag(monSvd$d[1:k],ncol=k) %*%
  t(monSvd$v[,1:k])


approxMon[approxMon < 0] <- 0
approxMon[approxMon > 1] <- 1


approxMon %>%
  as.cimg %>%
  plot(.,main=paste0(k," singular vectors"),axes=FALSE)
}
plot(as.cimg(X),main=paste0("Original image"),axes=FALSE)
```



2.1.2 More complex painting: Composition A, Piet Mondriaan

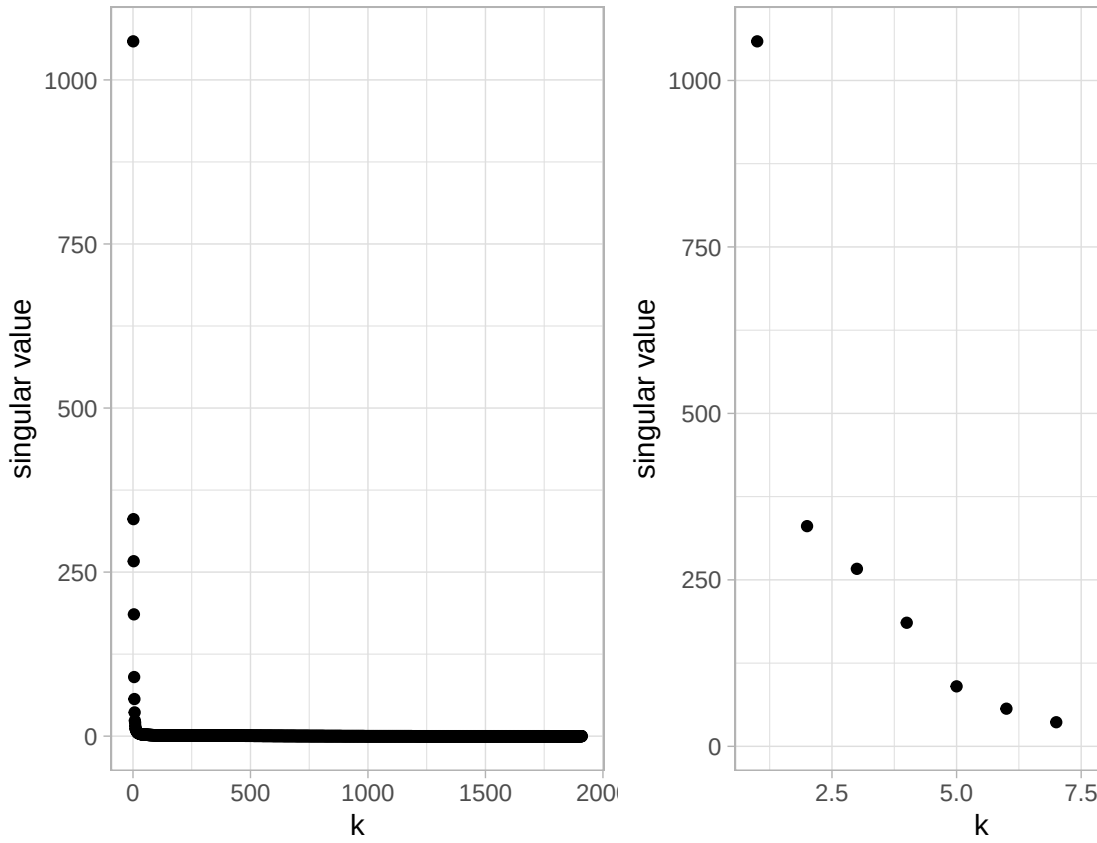
```
mondriaan <- load.image("https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Composition_A_by_Piet
mondriaan <- grayscale(mondriaan)
plot(mondriaan,axes=FALSE)
```



2.1.2.1 Load the original painting

```
X <- matrix(as.data.frame(mondriaan)[,3],nrow=dim(mondriaan)[1],ncol=dim(mondriaan)[2])
```

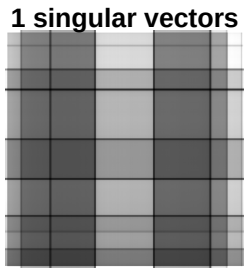
```
monSvd <- svd(X)

p1 <- data.frame(x=1:length(monSvd$d),y=monSvd$d) %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

p2 <- data.frame(x=1:10,y=monSvd$d[1:10]) %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

grid.arrange(p1,p2,nrow=1)
```

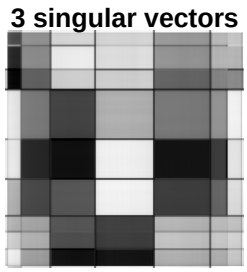


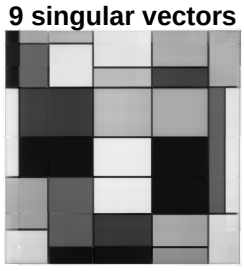
2.1.2.2 Singular values

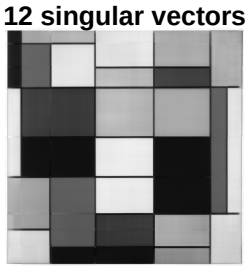
- The singular values decay a bit slower. The painting is a bit more complex. More lines and colors.

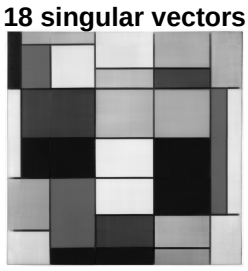
```
par (mfrow=c(3,3))
par(mar=c(1,2,1,1))
for (k in c(1,seq(3,21,3)))
{
approxMon <- monSvd$u[,1:k] %*%
  diag(monSvd$d[1:k],ncol=k) %*%
  t(monSvd$v[,1:k])


approxMon[approxMon < 0] <- 0
approxMon[approxMon > 1] <- 1


approxMon %>%
  as.cimg %>%
  plot(.,main=paste0(k," singular vectors"),axes=FALSE)
}
plot(as.cimg(X),main=paste0("Original image"),axes=FALSE)
```

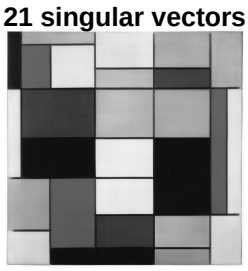








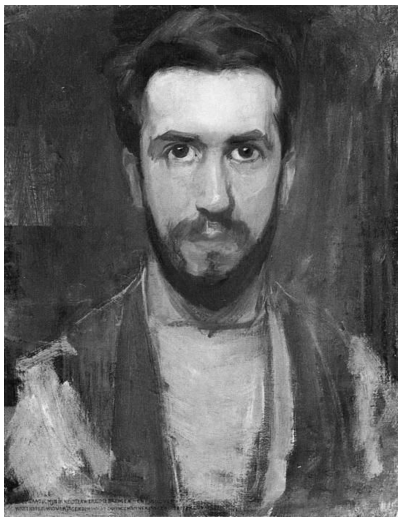




2.1.2.3 Evaluate data compression

2.1.3 Self portret Piet Mondriaan

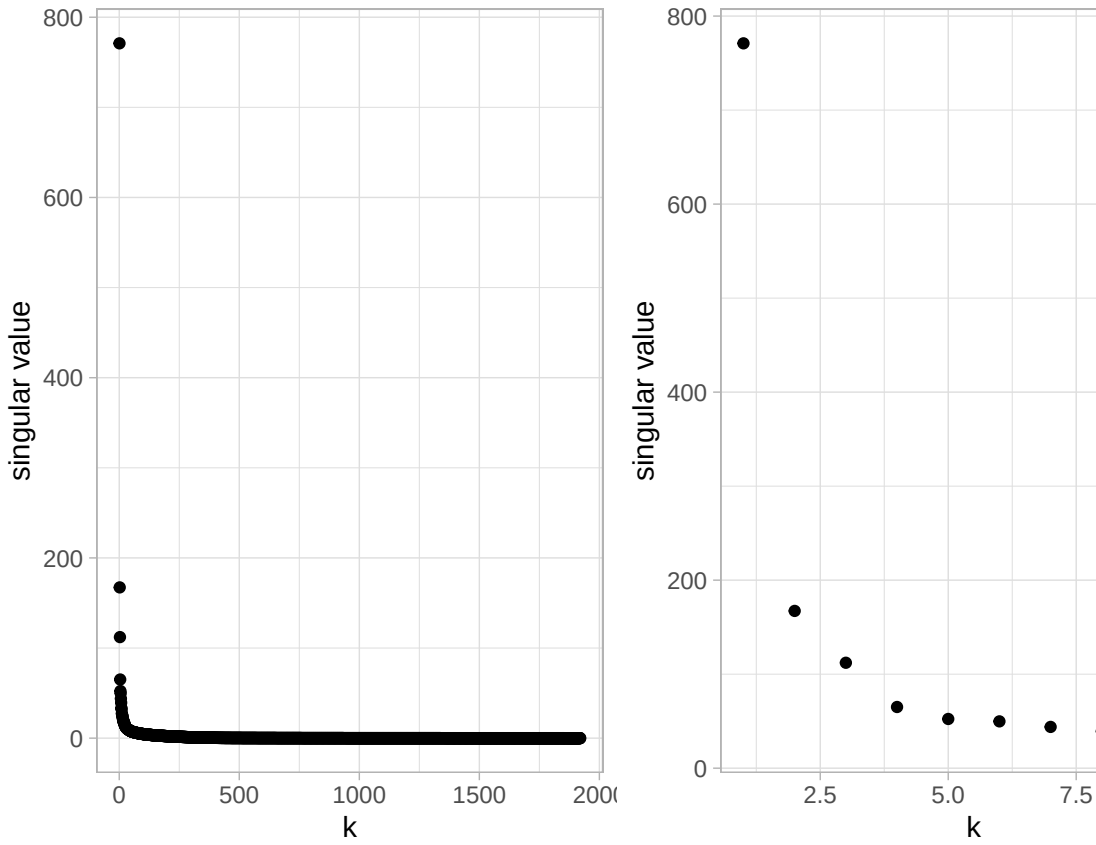
```
mondriaan <- load.image("https://upload.wikimedia.org/wikipedia/commons/thumb/6/66/Mondrian_Zelfportret
mondriaan <- grayscale(mondriaan)
plot(mondriaan,axes=FALSE)
```



2.1.3.1 Load the painting

```
X <- matrix(as.data.frame(mondriaan)[,3],nrow=dim(mondriaan)[1],ncol=dim(mondriaan)[2])
```

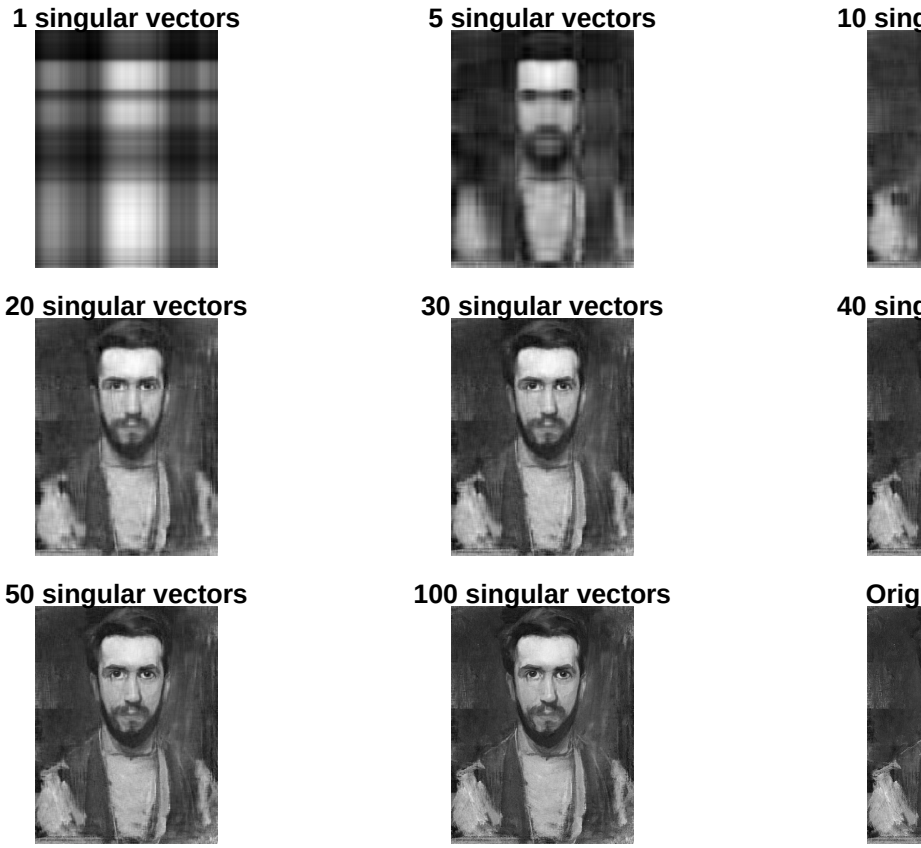
```
monSvd <- svd(X)

p1 <- data.frame(x=1:length(monSvd$d),y=monSvd$d) %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

p2 <- data.frame(x=1:10,y=monSvd$d[1:10]) %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  xlab("k") +
  ylab("singular value")

grid.arrange(p1,p2,nrow=1)
```



2.1.3.2 Singular values

- The singular values decay much slower. The painting is more complex.

```
par (mfrow=c(3,3))
par(mar=c(1,2,1,1))
for (k in c(1,5,10,20,30,40,50,100))
{
approxMon <- monSvd$u[,1:k] %*%
  diag(monSvd$d[1:k],ncol=k) %*%
  t(monSvd$v[,1:k])


approxMon[approxMon < 0] <- 0
approxMon[approxMon > 1] <- 1


approxMon %>%
  as.cimg %>%
  plot(.,main=paste0(k," singular vectors"),axes=FALSE)
}
plot(as.cimg(X),main=paste0("Original image"),axes=FALSE)
```



2.1.3.3 Evaluate compression

Here we need at least 40 singular vector. This leads to a data compression of $1 - \frac{40\times(1+1920+2494)}{1920\times2494} = 96.3\%$. We only use 40 left singular vectors (40×1920), 40 singular values, 40 right eigens vector (40×2494).

3 Geometric interpretation

Write the **truncated SVD** as

$$\mathbf{X}_k = \mathbf{U}_k \Delta_k \mathbf{V}_k^T = \mathbf{Z}_k \mathbf{V}_k^T$$

with

$$\mathbf{Z}_k = \mathbf{U}_k \Delta_k$$

an $n \times k$ matrix.

Each of the n rows of $\mathbf{Z}_k$, say $\mathbf{z}_{k,i}^T$, represents a point in a k-dimensional space.

Because of the orthonormality of the singular vectors, we also have

$$\begin{aligned} \mathbf{X}_k \mathbf{V}_k &= \mathbf{Z}_k \mathbf{V}_k^T \mathbf{V}_k \\ \mathbf{X}_k \mathbf{V}_k &= \mathbf{Z}_k. \end{aligned}$$

Thus the matrix $\mathbf{V}_k$ is a **transformation matrix** that may be used to transform $\mathbf{X}_k$ into $\mathbf{Z}_k$, and $\mathbf{Z}_k$ into $\mathbf{X}_k$.

Note that

- The matrix $\mathbf{V}_k$ transforms the p-dimensional $\mathbf{X}_k$ into the k-dimensional $\mathbf{Z}_k$: $\mathbf{Z}_k = \mathbf{X}_k\mathbf{V}_k$. Note, however, that the matrix $\mathbf{X}_k$ must not necessarily be used for this transformation, because the SVD of the original matrix $\mathbf{X}$ also gives directly $\mathbf{Z}_k = \mathbf{U}_k\Delta_k$.
- The inverse transformation from the k-dimensional $\mathbf{Z}_k$ to the p-dimensional $\mathbf{X}_k$ is given by the transpose of $\mathbf{V}_k$: $\mathbf{Z}_k\mathbf{V}_k^T = \mathbf{X}_k$. Often inverse transformations are given by the inverse of a matrix, but thanks to the orthonormality of the columns of $\mathbf{V}_k$, we get $\mathbf{V}_k^T\mathbf{V}_k = \mathbf{I}$, and thus $\mathbf{V}_k^T$ acts as an inverse.
- The transformation from the k-dimensional $\mathbf{Z}_k$ to the p-dimensional $\mathbf{X}_k$ is transforming points from a low dimensional space (k) to a high dimensional space (p). You may not interpret this as if this transformation adds information; the transformed points in $\mathbf{X}_k$ still live in a k-dimensional subspace of the larger p-dimensional space; the matrix $\mathbf{X}_k$ is only of rank k and thus contains less information than the original data matrix $\mathbf{X}$ (if rank($\mathbf{X}$)= $r > k$).

More importantly, it can be shown that (thanks to orthonormality of $\mathbf{V}$)

$$\mathbf{X}\mathbf{V}_k = \mathbf{Z}_k.$$

This follows from (w.l.g. rank($\mathbf{X}$)=r)

$$\begin{aligned}\mathbf{X}\mathbf{V}_k &= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}_k = \mathbf{U}\mathbf{D}\begin{pmatrix}\mathbf{v}_1^T\\ \vdots\\ \mathbf{v}_r^T\end{pmatrix}(\mathbf{v}_1 \dots \mathbf{v}_k)\\ &= \mathbf{U}\mathbf{D}\mathbf{V}^T\mathbf{V}_k = \mathbf{U}\mathbf{D}\begin{pmatrix}1 & 0 & \dots & 0\\ 0 & 1 & \dots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & \dots & 1\\ 0 & 0 & \dots & 0\\ \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & \dots & 0\end{pmatrix} = \mathbf{U}_k\Delta_k = \mathbf{Z}_k\end{aligned}$$

The $p \times k$ matrix $\mathbf{V}_k$ acts as a transformation matrix: transforming n points in a p dimensional space to n points in a k dimensional space.

We take a closer look at

$$\mathbf{Z}_k = \mathbf{X}\mathbf{V}_k = \begin{pmatrix}\mathbf{x}_1^T\\ \vdots\\ \mathbf{x}_n^T\end{pmatrix}(\mathbf{v}_1 \dots \mathbf{v}_k).$$

The ith row (observation) in $\mathbf{Z}_k$ equals

$$\mathbf{z}_{k,i}^T = \mathbf{x}_i^T\mathbf{V}_k = (\mathbf{x}_i^T\mathbf{v}_1, \mathbf{x}_i^T\mathbf{v}_2, \dots, \mathbf{x}_i^T\mathbf{v}_k).$$

Hence, $\mathbf{z}_{k,i}^T = \mathbf{x}_i^T\mathbf{V}_k$ is the orthogonal projection of $\mathbf{x}_i$ onto the k-dimensional subspace spanned by the columns of $\mathbf{V}_k$.

SVD transforms data set to lower dimensional data set: The SVD thus gives a transformation of the p dimensional data to $k \leq r$ dimensional data:

$$\mathbf{Z}_k = \mathbf{X}\mathbf{V}_k.$$

This is essentially a dimension reduction.

Note that,

- The transformation from p-dimensional $\mathbf{X}$ to k-dimensional $\mathbf{Z}_k$ is important. It shows that the n points in the rows of $\mathbf{Z}_k$ are the result of projecting the n points in $\mathbf{X}$ onto the columns of $\mathbf{V}_k$ (i.e. the first k singular vectors of $\mathbf{X}$). We say that the space of $\mathbf{Z}_k$ is spanned by the column of $\mathbf{V}_k$.
- The points (rows) in $\mathbf{X}$ live in a p-dimensional space (or rank($\mathbf{X}$)= r if $r < p$) and they are thus projected onto a lower dimensional space. This is in contrast to the projection $\mathbf{X}_k\mathbf{V}_k = \mathbf{Z}_k$, because the points in $\mathbf{X}_k$ live in a k-dimensional subspace of $\mathbf{X}$.
- Note that with $k < r$ there is no unique transformation to transform $\mathbf{Z}_k$ back to $\mathbf{X}$. On the previous slide we only established the transformation $\mathbf{Z}_k\mathbf{V}_k^T = \mathbf{X}_k$. Indeed, starting from $\mathbf{X}\mathbf{V}_k = \mathbf{Z}_k$, and right-multiplying with $\mathbf{V}_k^T$ does not give the backtransformation, because $\mathbf{V}_k\mathbf{V}_k^T$ is not the identity matrix.

4 Interpretation of SVD in terms of correlation matrices

For a matrix $\mathbf{X}$ the sample variance covariance matrix estimator is $p \times p$ matrix

$$\begin{aligned} \mathbf{S} &= \frac{1}{N-1}(\mathbf{X} - \bar{\mathbf{X}})^T(\mathbf{X} - \bar{\mathbf{X}}) && (3) \\ &= \frac{1}{N-1}\left[\mathbf{X}^T\mathbf{X} - \bar{\mathbf{X}}^T\bar{\mathbf{X}}\right] && (4) \end{aligned}$$

So $\mathbf{X}^T\mathbf{X}$ defines up to a constant the variance covariance matrix of $\mathbf{X}$! When the matrix is column centered $\mathbf{S} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$.

The same holds for the rows of $\mathbf{X}$! The covariance between the subjects can be estimated as

$$\mathbf{S} = \frac{1}{p-1}\mathbf{X}\mathbf{X}^T$$

upon row centering.

Note, that

$$\begin{aligned} \mathbf{X}^T\mathbf{X} &= \mathbf{V}\Delta\mathbf{U}^T\mathbf{U}\Delta\mathbf{V}^T && (5) \\ &= \mathbf{V}\Delta^2\mathbf{V}^T && (6) \end{aligned}$$

If we rewrite the expression

$$\begin{aligned} \mathbf{X}^T\mathbf{X}\mathbf{V} &= \mathbf{V}\Delta^2\mathbf{V}^T\mathbf{V}\mathbf{X}^T\mathbf{X}\mathbf{V} && (7) \\ &= \mathbf{V}\Delta^2 && (8) \end{aligned}$$

So, if the data are centered, the SVD can be used to perform a spectral decomposition of the sample covariance matrix where the right singular vectors correspond to the eigen vectors of the covariance matrix and the eigenvalues are the squared singular values!

Similarly the left singular values can be used to estimate the covariance matrix of the rows of $\mathbf{X}$. So in our notation covariance between subjects.

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\Delta^2\mathbf{U}^T \tag{9}$$

This link is for instance very useful for recommender systems, i.e. to propose movies based on the subjects with whom you correlate. We will also exploit this when we discuss on PCA.

5 SVD and inverse of a matrix

A linear system of equations with n equations and n unknowns

$$\mathbf{A}_{n\times n}\beta = \mathbf{b}$$

can be solved by

$$\beta = \mathbf{A}^{-1}_{n\times n}\mathbf{b}$$

A unique solution exists if A is full rank.

Note, that a singular value decomposition of the square matrix $\mathbf{A} = \mathbf{V}\Delta\mathbf{V}^T$ enables the inverse to be written as

$$\mathbf{A}^{-1} = \mathbf{V}\Delta^{-1}\mathbf{V}^T$$

indeed

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{V}\Delta^{-1}\mathbf{V}^T\mathbf{V}\Delta\mathbf{V}^T = \mathbf{I}$$

Note, that the SVD generalizes this to systems of under (n<p, fat short matrices) and over determined systems (n>p tall skinny matrices):

Let

$$\mathbf{A} = \mathbf{U}\Delta\mathbf{V}^T \tag{10}$$

and we want to solve

$$\begin{aligned}
\mathbf{A}\beta &= \mathbf{b} && (11)\\
\mathbf{U}\Delta\mathbf{V}^T\beta &= \mathbf{b} && (12)\\
\mathbf{U}^T\mathbf{U}\Delta\mathbf{V}^T\beta &= \mathbf{U}^T\mathbf{b} && (13)\\
\Delta\mathbf{V}^T\beta &= \mathbf{U}^T\mathbf{b} && (14)\\
\Delta^{-1}\Delta\mathbf{V}^T\beta &= \Delta^{-1}\mathbf{U}^T\mathbf{b} && (15)\\
\mathbf{V}\mathbf{V}^T\beta &= \mathbf{V}\Delta^{-1}\mathbf{U}^T\mathbf{b} && (16)\\
\beta &= \mathbf{V}\Delta^{-1}\mathbf{U}^T\mathbf{b} && (17)
\end{aligned}$$

Note, that for an overdetermined system $n > p$ so $r \leq p$. Generally, $r = p$ and $\mathbf{V}$ is thus a square matrix so both $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}$. However, $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ but $\mathbf{U}\mathbf{U}^T \neq \mathbf{I}$ because $r < n$.

$\mathbf{A}^\dagger = \mathbf{V}\Delta^{-1}\mathbf{U}^T$ is also referred to as the pseudo inverse and it enables us to solve under and overdetermined systems of equations.

Note, that for

- underdetermined systems there typically does not exist a unique solution
- for overdetermined systems usually there does not exist an exact solution. We will focus on the latter in the next section where we explore the link between linear regression and SVD.

6 Linear regression and SVD

Suppose we have the linear regression problem with $n > p$:

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon$$

$\mathbf{X}$ is a tall skinny matrix with $n >> p$.

We know that

$$\hat{\beta} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

If we replace $\mathbf{X}$ by its SVD

$$\begin{aligned}
\hat{\beta} &= \left(\mathbf{V}\Delta^2\mathbf{V}^T\right)^{-1}\mathbf{V}\Delta\mathbf{U}^T\mathbf{Y} && (18)\\
\hat{\beta} &= \mathbf{V}\Delta^{-2}\mathbf{V}^T\mathbf{V}\Delta\mathbf{U}^T\mathbf{Y} && (19)\\
\hat{\beta} &= \mathbf{V}\Delta^{-1}\mathbf{U}^T\mathbf{Y} && (20)
\end{aligned}$$

So the SVD also solves the linear regression problem by using the pseudoinverse! If we now think about the fit:

$$\begin{aligned}
\hat{\mathbf{Y}} &= \mathbf{X}\hat{\beta} && (21)\\
&= \mathbf{U}\Delta^{-1}\mathbf{V}^T\mathbf{V}\Delta\mathbf{U}^T\mathbf{Y} && (22)\\
&= \mathbf{U}\mathbf{U}^T\mathbf{Y} && (23)
\end{aligned}$$

- For an overdetermined system $\mathbf{U}\mathbf{U}^T$ is not equal to the unity matrix $\mathbf{I}$ (for an overdetermined system only $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ because $n > r$).
- So $\hat{\mathbf{Y}} \neq \mathbf{Y}$. Hence, we typically do not have an exact solution.
- Note, that $\mathbf{U}\mathbf{U}^T$ spans the same space as the columns of $\mathbf{X}$, and will define the same p-dimensional plane in the n dimensional space $\mathcal{R}^n$, e.g. cfr $\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$. So it projects $\mathbf{Y}$ in the column space of $\mathbf{X}$ and the errors will be orthogonal onto this plane.

6.1 Example prostate dataset

6.1.1 Fit with lm

```
prostate <- read_csv(
  "https://raw.githubusercontent.com/GTPB/PSLS20/master/data/prostate.csv",
  col_types = cols()
)
lm1 <- lm(lpsa ~ lcavol + lweight + svi, prostate)
```

6.1.2 Fit with SVD

```
X <- prostate[,c(1:2,5)]
X[,3] <- as.double(X[,3]!="healthy")
X <- cbind(Intercept=1,X)

svdX <- svd(X)
betaSvd <- svdX$v %*% diag(1/svdX$d) %*% t(svdX$u) %*% prostate$lpsa

cbind(lm1$coef,betaSvd)
```

```
##                   [,1]       [,2]
## (Intercept) -0.2680724 -0.2680724
## lcavol       0.5516386  0.5516386
## lweight      0.5085359  0.5085359
## sviinvasion  0.6661583  0.6661583
```

7 SVD and Multi-Dimensional Scaling (MDS)

7.1 Example

In this section we will use a dataset on food consumption in the UK. The data originate from the UKs 'Department for Environment, Food and Rural Affairs' (DEFRA), showing the consumption in grams (per person, per week) of 17 different types of foodstuff measured and averaged in the four countries of the United Kingdom in 1997. We would like to explore the data and interpret how the food patterns of the different countries differ.

```
uk <- read_csv(
  "https://raw.githubusercontent.com/statOmics/HDA2020/data/ukFoods.csv",
  col_types = cols()
)
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
knitr::kable(uk, caption = "The full UK foods data table")
```

Table 1: The full UK foods data table

X1	England	Wales	Scotland	N.Ireland
Cheese	105	103	103	66
Carcass_meat	245	227	242	267
Other_meat	685	803	750	586
Fish	147	160	122	93
Fats_and_oils	193	235	184	209
Sugars	156	175	147	139
Fresh_potatoes	720	874	566	1033
Fresh_Veg	253	265	171	143
Other_Veg	488	570	418	355
Processed_potatoes	198	203	220	187
Processed_Veg	360	365	337	334
Fresh_fruit	1102	1137	957	674

X1	England	Wales	Scotland	N.Ireland
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft_drinks	1374	1256	1572	1506
Alcoholic_drinks	375	475	458	135
Confectionery	54	64	62	41

Note, that here the matrix is displayed with the p variables in the rows and the n experimental units (countries) in the columns. This is often done for high dimensional data where $p >> n$ because this makes it easier to look at to the raw data table. Note, that the svd calculates left and right singular vectors so it will also provide the correct solution, we just should look to the other set of singular vectors in order to get to the correct interpretation.

7.2 Motivation

The objective of Multidimensional Scaling (MDS) is to find a low-dimensional representation, say k-dimensional, of n data points such that the distances between the n points in the k-dimensional space is a good approximation of a given squared distance matrix, say $\mathbf{D}_X$.

- The squared distance matrix $\mathbf{D}_X$ may be given without knowledge of the original observations (not even the dimensionality), or it may be computed from a given set of n p-dimensional data points.
- Note that the distances between points in a k-dimensional subspace coincide with the distances between these points in the larger p-dimensional space.

Use of MDS:

- A high dimensional data matrix $\mathbf{X}$ is given, and one wants to get a visual representation (in 2 or 3 dimensions) of the observations. In this graph each point represents an observation (row of data matrix). From this graph one wants points close to one another to be similar, and observations far away from one another to be dissimilar. Thus the distances between the n points in the original p-dimensional space should be well preserved in the 2 or 3 dimensional space.
- In some applications the researcher only has knowledge of the similarity (or dissimilarity) between observations. For example, food products can be evaluated by a taste panel and a dissimilarity matrix can be completed. This dissimilarity matrix shows for each pair of food products their dissimilarity (numerical values provided by taste panel, but not objectively quantified). Given this dissimilarity matrix, a 2 or 3 dimensional graph could be helpful if the distances between the points (food products) in this graph are monotonically related to the dissimilarities provided by the taste panel. Food products close to one another in this graph, taste similarly.
- Here we discuss the "metric" MDS, which actually requires the Euclidean distances between observations. However, the method works also well if the matrix $\mathbf{D}_X$ contains dissimilarities rather than squared Euclidean distances.

In this chapter we assume that the data matrix $\mathbf{X}$ is column-centered (i.e. each column has mean zero).

Centering can be accomplished by multiplying the original data matrix $\mathbf{X}$ with the $n \times n$ **centering matrix**

$$\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^T,$$

in which $\mathbf{1}$ is an n-vector with all entries equal to 1. Hence, $\mathbf{HX}$ is the column-centered data matrix.

We will assume that $\mathbf{X}$ is already column-centered, and therefore $\mathbf{HX} = \mathbf{X}$.

(Note, that the matrix $\mathbf{1}\mathbf{1}^T$ is an $n \times n$ matrix with all entries set to 1.)

We will need the following interesting relationship between the **Gram matrix** $\mathbf{XX}^T$ and the distance matrix.

- For an $n \times p$ data matrix $\mathbf{X}$, the matrix $\mathbf{D}_X$ with squared distances has elements

$$(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j) = \|\mathbf{x}_i\|^2 - 2\mathbf{x}_i^T\mathbf{x}_j + \|\mathbf{x}_j\|.^2$$

-The $n \times n$ squared distance matrix can then be written as

$$\mathbf{D}_X = \mathbf{N} - 2\mathbf{XX}^T + \mathbf{N}^T,$$

with $\mathbf{N}$ the $n \times n$ matrix with ith row filled with $\|\mathbf{x}_i\|^2$.

- Note that the elements of $\mathbf{N}$ can also be found on the diagonal of $\mathbf{XX}^T$.
- Given the structure of the $\mathbf{H}$ and $\mathbf{N}$ matrices, it is easy to verify that

$$-\frac{1}{2}\mathbf{HD}_X\mathbf{H} = \mathbf{XX}^T.$$

  This gives an important relation between the distance matrix and the Gram matrix. We will use the notation $\mathbf{G}_X = -\frac{1}{2}\mathbf{HD}_X\mathbf{H}$.

- The $n \times n$ Gram matrix $\mathbf{XX}^T$ equals

$$\begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} (\mathbf{x}_1 \dots \mathbf{x}_n)$$

  and has thus on its (i, j)th position the inner product

$$\mathbf{x}_i^T\mathbf{x}_j = \|\mathbf{x}_i\|\|\mathbf{x}_j\| \cos < \mathbf{x}_i, \mathbf{x}_j > .$$

- On the diagonal we find $\mathbf{x}_i^T\mathbf{x}_i = \|\mathbf{x}_i\|^2$, the squared norm of the ith observation.
- The relation $-\frac{1}{2}\mathbf{HD}_X\mathbf{H} = \mathbf{XX}^T$ tells us that the distance matrix and the Gram matrix contain the same information. The distances, however, are in most situations easier to interpret.

7.3 Link with the SVD

Note, that we have shown that we can rewrite $\mathbf{XX}^T$ using the SVD:

$$\mathbf{XX}^T = \mathbf{U}\Delta^2\mathbf{U}^T$$

Because the truncated SVD of $\mathbf{X}$ minimises the Frobenius norm $\|\mathbf{X} - \mathbf{X}_k\|_F^2$, it this has an important consequence:

- Let $\mathbf{D}_X$ denote the $n \times n$ matrix with the squared Euclidian distances between the n data points $\mathbf{x}_i$ in the original p-dimensional space.
- Let $\mathbf{D}_{Zk}$ denote the $n \times n$ matrix with the squared Euclidian distances between the n transformed data points $\mathbf{z}_{k,i}$ in the reduced k-dimensional space.

Then, it can be shown that the truncated SVD also minimises

$$\|\mathbf{D}_X - \mathbf{D}_{Zk}\|_F^2.$$

Or in other words: The n points $\mathbf{z}_{k,i}$ in the k-dimensional subspace spanned by the columns of $\mathbf{V}_k$ are the best approximation of the n original points $\mathbf{x}_i$ in the original p-dimensional space in the sense that the Euclidean distances between the n original points are best approximated by the Euclidean distances between the n transformed points, among all possible k-dimensional linear subspaces.

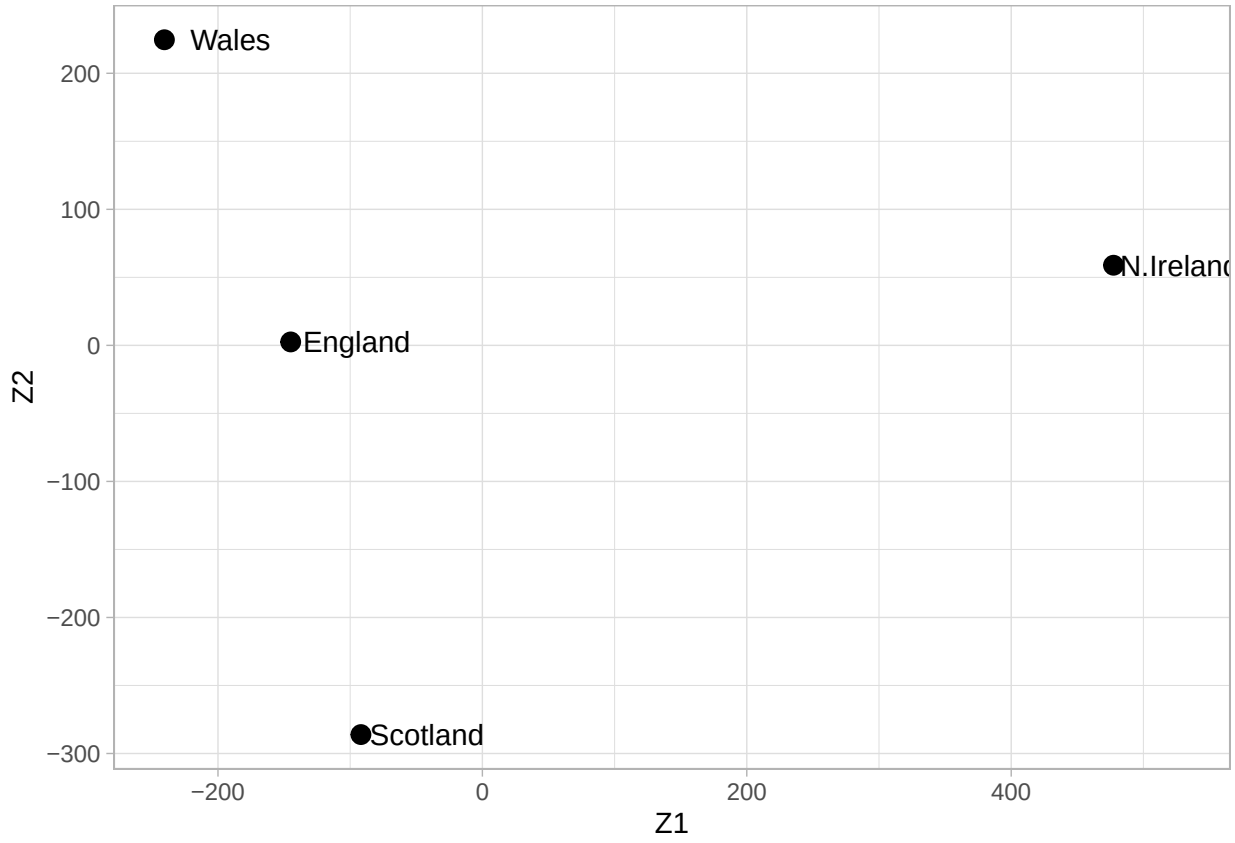
- If $\mathbf{X}$ is known, we can readily obtain the k-dimensional projection by the SVD of $\mathbf{X}$
- If we only know the distance matrix $\mathbf{D}$
  1. We can readily obtain the Gram matrix
  $$-\frac{1}{2}\mathbf{H}\mathbf{D}_X\mathbf{H} = \mathbf{X}\mathbf{X}^T$$
  2. The truncated SVD of the squared and symmetric Gram matrix
  $$\mathbf{U}_k\Delta_k'\mathbf{U}_k^T$$
  3. from which also can obtain
  $$\mathbf{Z}_k = \mathbf{U}_k\Delta_k,$$
  with $\Delta_k = {}'^{\frac{1}{2}}_k$

7.4 Example

```
X <- as.matrix(t(uk[,-1]))
n <- nrow(X)
H <- diag(n) - matrix(1/n,nrow=n,ncol=n)
X <- H%*%X
svdUk <- svd(X)

k <- 2
Uk <- svdUk$u[,1:k]
Dk <- diag(svdUk$d[1:k])
Zk <- Uk%*%Dk
rownames(Zk) <- colnames(uk)[-1]
colnames(Zk) <- paste0("Z",1:k)
Zk
```

```
##                   Z1          Z2
## England   -144.99315    2.532999
## Wales     -240.52915  224.646925
## Scotland   -91.86934 -286.081786
## N.Ireland  477.39164   58.901862
```

```
Zk %>%
  as.data.frame %>%
  ggplot(aes(x=Z1,y=Z2,label=rownames(Zk))) +
  geom_point(size = 3) +
  geom_text(nudge_x = 50)
```



- The graph suggests that Wales and England are quite similar in terms of food consumption, and North Ireland seem to have a very different food consumption pattern.
- Also note that England is close to the origin, meaning that the food consumption pattern is close to the average pattern in the UK.
- A few questions remain:
  - How well can the 17-dimensional data be represented in a 2-dimensional subspace?
  - How can we interpret the distances (differences) between the data points in terms of the original 17 variables?

7.5 The biplot

The biplot is a single 2-dimensional graph which displays the information in both $\mathbf{Z}_2 = \mathbf{U}_2\Delta_2$ and $\mathbf{V}_2$.

The plot of $\mathbf{Z}_2$ has been discussed previously (e.g. best approximation of distances).

The name "bi"plot refers to the plotting of two parts of the SVD ($\mathbf{Z}$ and $\mathbf{V}$) in a single graph.

From the geometrical interpretation of the SVD we know that $\mathbf{Z}_2$ is the orthogonal projection of $\mathbf{X}$ on $\mathbf{V}_2$ the basis spanned by the first two singular values. Thus for the i^{th} individual we get

$$\mathbf{z}_{2,i} = \mathbf{x}_i \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}$$

We also know that we can approximate $\mathbf{X}$ by $\mathbf{X}_2$ which equals:

$$\mathbf{X}_2 = \mathbf{Z_2}\mathbf{V}_2^T \tag{24}$$

$$= \mathbf{Z_2} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \end{bmatrix} = \mathbf{Z_2} \begin{bmatrix} \tilde{\mathbf{v}}_{2,1} \dots \tilde{\mathbf{v}}_{2,p} \end{bmatrix} \tag{25}$$

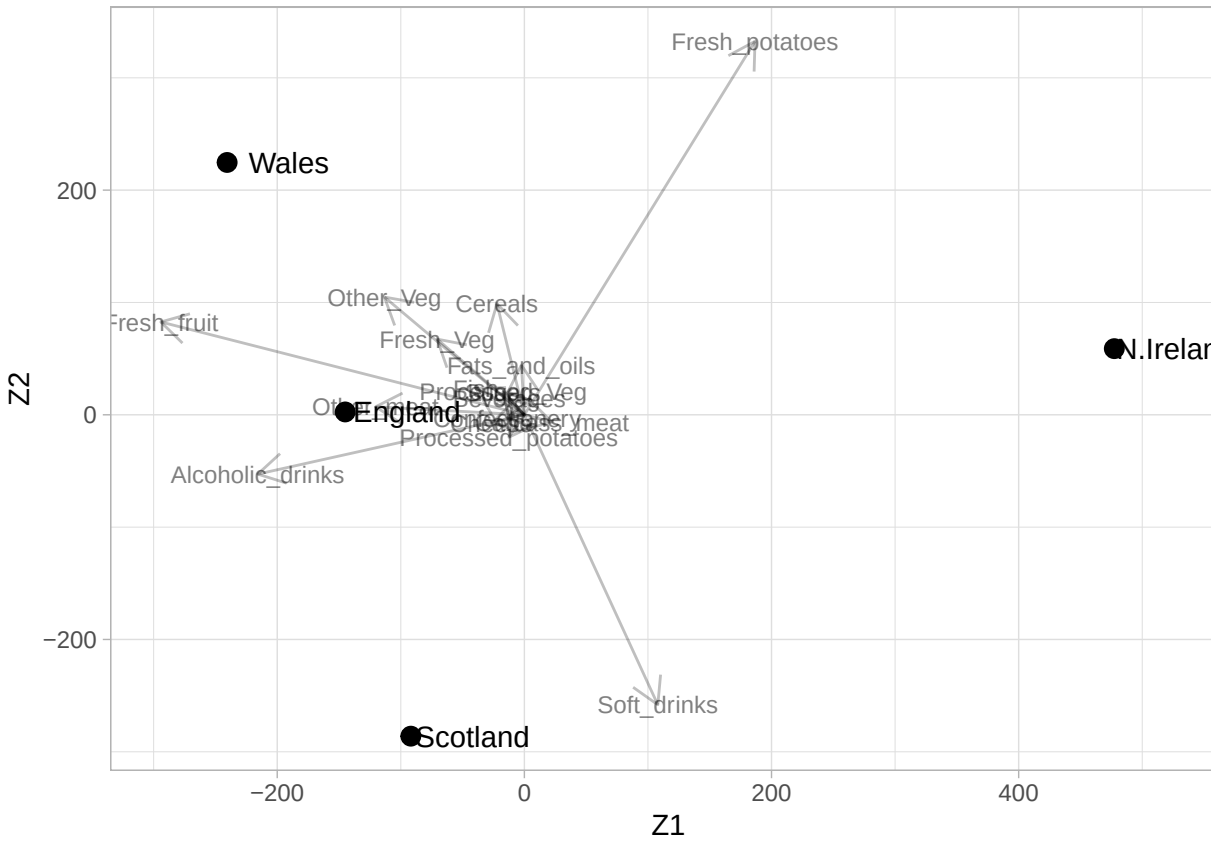
with $\mathbf{v}_1$ and $\mathbf{v}_2$ the first two right singular vector and $\tilde{\mathbf{v}}_j$ the j$^{\text{th}}$ column of the matrix $\mathbf{V}_2^T$.

- This basically shows us that the orthogonal projection of $\mathbf{z}_{2,i}$ for subject/experimental unit i on $\tilde{\mathbf{v}}_{2,j}$ gives us the approximation of the value for j$^{\text{th}}$ variable that was observed for the i^{th} experimental unit, i.e. $x_{2,ij}$.

$$x_{2,ij} = \mathbf{z}_{2,i}^T \tilde{\mathbf{v}}_{2,j}$$

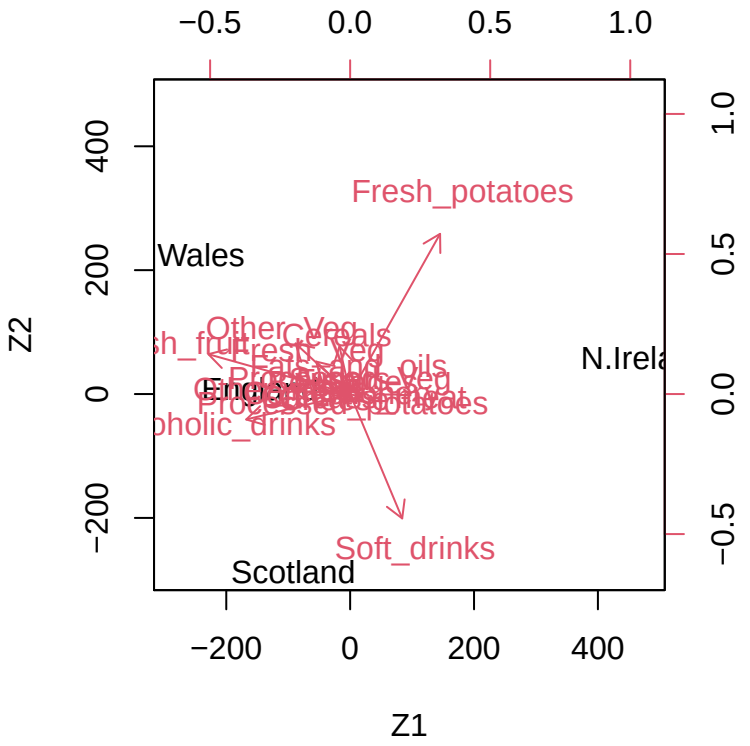
7.5.1 UK example

```
Vk<-svdUk$v[,1:k]
rownames(Vk) <- uk %>% pull(X1)
colnames(Vk) <- colnames(Zk)
scaleFactor <- mean(svdUk$d[1:k])

diyBiplot <- ggplot() +
  geom_point(
    data=Zk %>% as.data.frame,
    aes(x=Z1,y=Z2),
    size = 3) +
  geom_text(
    data=Zk %>% as.data.frame,
    aes(x=Z1,y=Z2,label=rownames(Zk)),
    nudge_x = 50) +
  geom_segment(
    data=Vk %>% as.data.frame,
    aes(x=0, y=0, xend=Z1*scaleFactor, yend=Z2*scaleFactor),
    arrow=arrow(length=unit(0.4,"cm")),
    alpha=0.25) +
  geom_text(
    data=Vk %>% as.data.frame,
    aes(x=Z1*scaleFactor, y=Z2*scaleFactor, label=rownames(Vk)),
    alpha=0.5,
    size=3)

diyBiplot
```



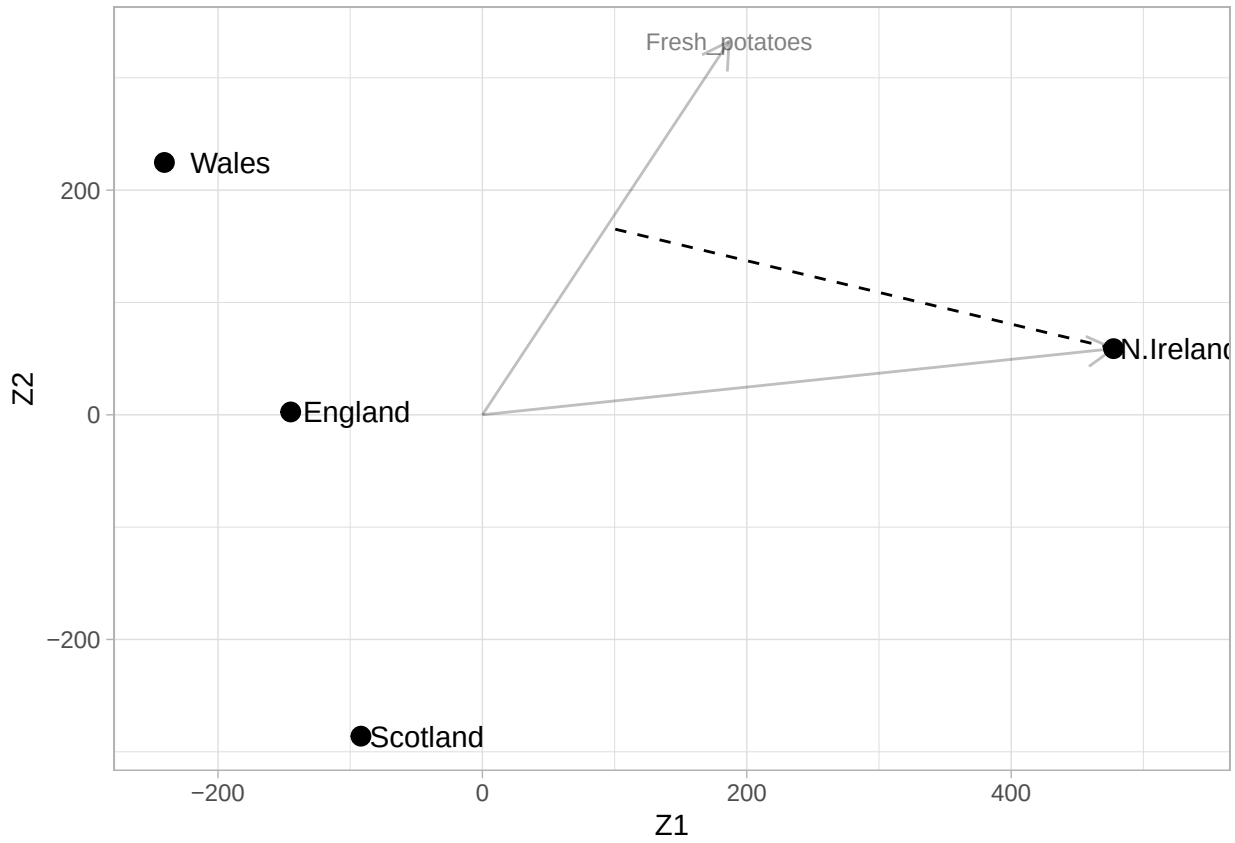
7.5.1.1 Biplot

```
biplot(Zk,Vk)
```



The R code shows two ways of constructing a biplot: the first method starts from the SVD of $\mathbf{X}$ and plots the p vectors $\tilde{\mathbf{v}}_{2,j}$ as arrows. Note that we used a scaling factor to give the arrows a convenient length in the graph (not too small, not too large); it does not affect the interpretation. The second way of obtaining the biplot is simply by using the R built-in function biplot.

7.5.1.2 Illustration of projection We project $\mathbf{z}_{2,i}$ for N.Ireland on Fresh_potatoes.




```
rownames(X) <- rownames(Zk)
colnames(X) <- rownames(Vk)
X[,"Fresh_potatoes"]
```

```
##   England     Wales  Scotland N.Ireland
##    -78.25     75.75   -232.25    234.75
```

```
Zk["N.Ireland",]%*%Vk["Fresh_potatoes",]
```

```
##          [,1]
## [1,] 233.7418
```

We observe that N. Ireland has a consumption in Fresh_potatoes that is 234.75 above the average and this is well approximated by the projection 233.74.

Also note that

- The origin corresponds to the sample average of the 17-dimensional observations in the data matrix
- England is close to the origin and thus one could say that the food consumption in England is as the average in the UK
- Projecting the $\mathbf{Z}_2$ coordinates for North Ireland orthogonally onto the vector $\tilde{\mathbf{v}}_{2,j}$ of fresh potatoes, we get a large and positive $x_{2,ij}$. Hence, people in N. Ireland tend to eat more fresh potatoes than on average in the UK.
- We can do the same for the other vectors.

7.5.2 Further interpretation of the Biplot

From $\mathbf{x}_{k,i}^T = \mathbf{z}_{k,i}^T \mathbf{V}_k^T$ we learnt that the original data can be (approximately) reconstructed from the $\mathbf{z}_{k,i}$.

We now show how the dimensions of $\mathbf{Z}_k$ (columns or variables) can be interpreted. Remember,

$$\mathbf{z}_{k,i}^T = \mathbf{x}_i^T \mathbf{V}_k = \left(\mathbf{x}_i^T \mathbf{v}_1, \mathbf{x}_i^T \mathbf{v}_2, \ldots, \mathbf{x}_i^T \mathbf{v}_k\right).$$

The jth column of $\mathbf{Z}_k$ can be written as

$$\mathbf{z}_{k,j} = \mathbf{X}\mathbf{v}_j,$$

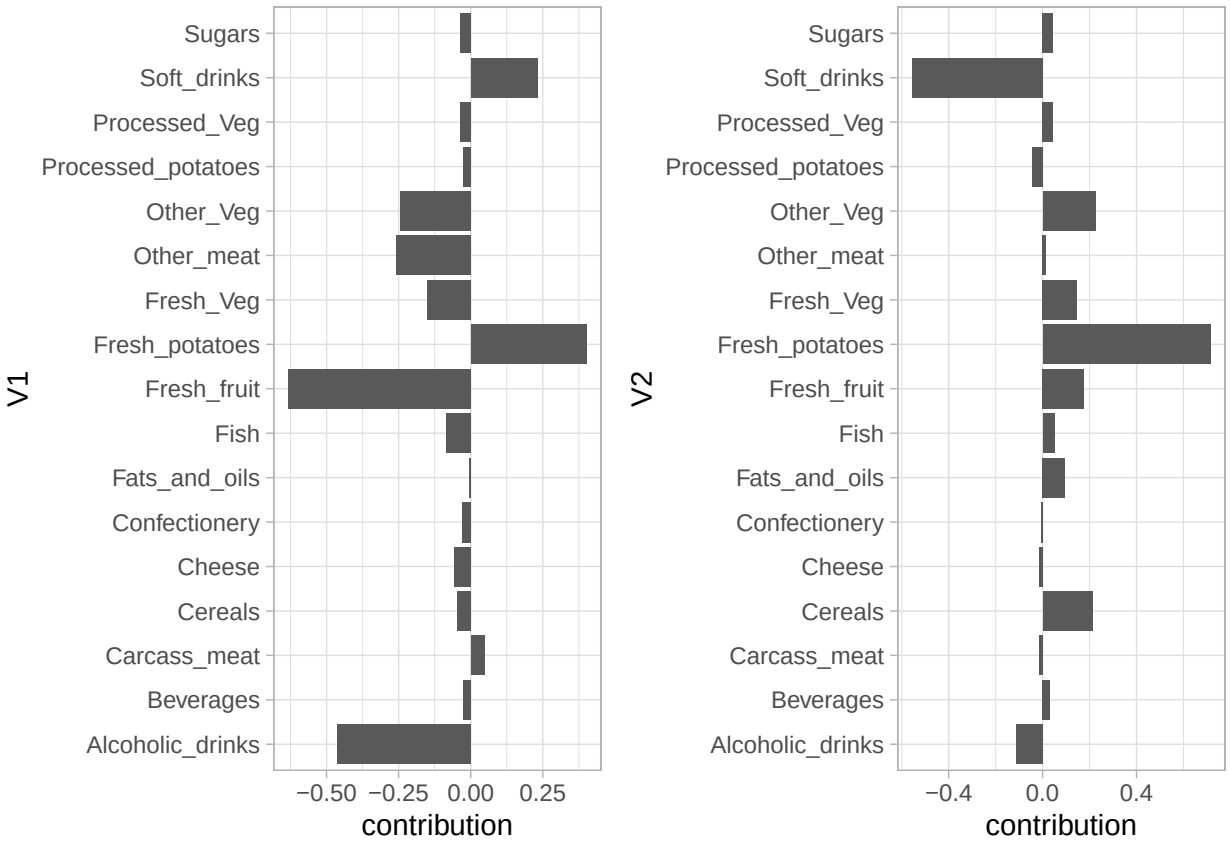
An individual element of $\mathbf{Z}_k$ can be written as

$$z_{k,ij} = \mathbf{x}_i^T \mathbf{v}_j,$$

which is a linear combination of the observations on the p variables in $\mathbf{x}_i$, with coefficients given by the p elements in $\mathbf{v}_j$. Understanding the elements in $\mathbf{v}_j$ will give us insight into the interpretation of the jth dimension of $\mathbf{Z}_k$.

7.6 Illustration Uk food

```
p1 <- ggplot() +
  geom_bar(
    aes( x=rownames(Vk), y=Vk[,1]),
    stat="identity") +
  xlab("V1") +
  ylab("contribution") +
  coord_flip()

p2 <-
ggplot() +
  geom_bar(
    aes( x=rownames(Vk), y=Vk[,2]),
    stat="identity") +
  xlab("V2") +
  ylab("contribution") +
  coord_flip()

grid.arrange(p1,p2,ncol=2)
```



In giving an interpretation to the elements in $\mathbf{v}_1$ and $\mathbf{v}_2$, we ignore the elements close to zero (this is a subjective decision; later we see more objective methods).

For $\mathbf{v}_1$ (1st dimension of $\mathbf{Z}_2$):

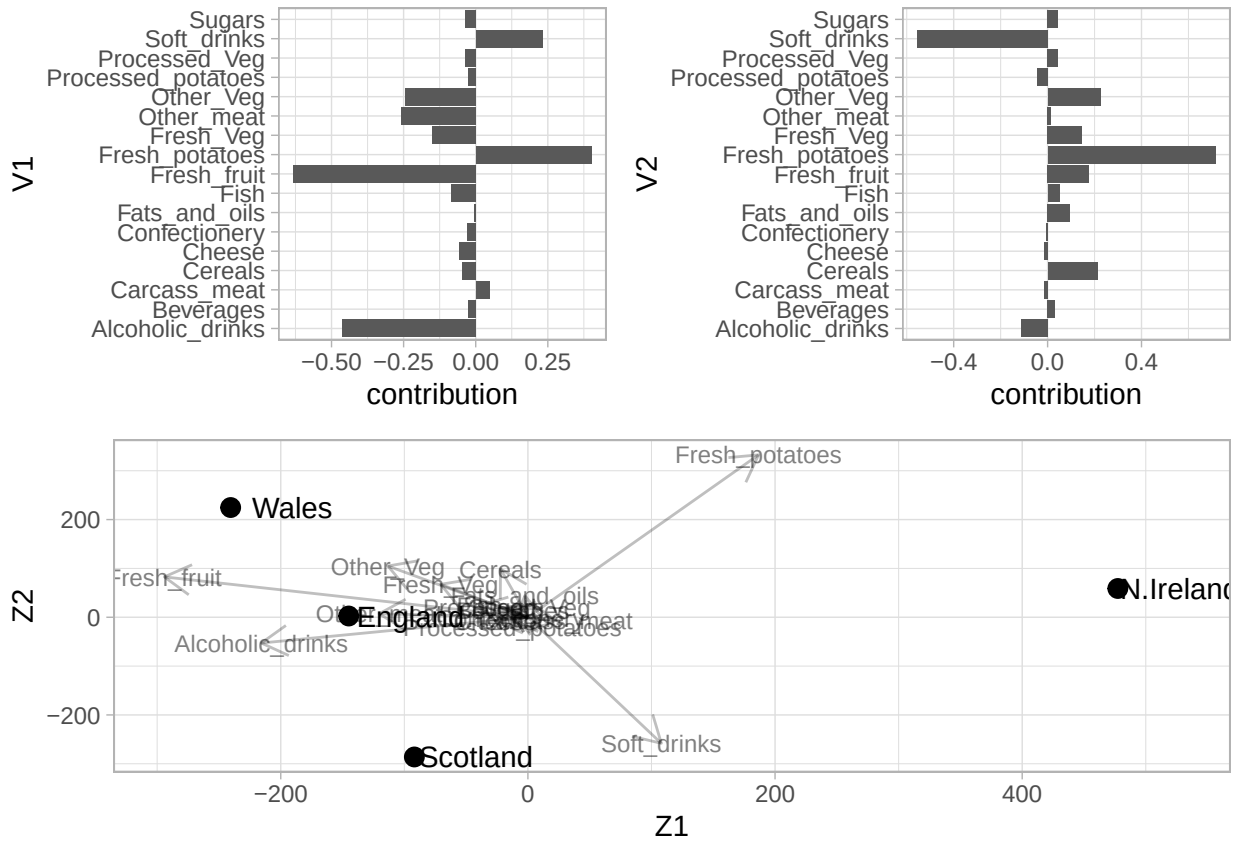
- contrast of soft drinks and fresh potatoes versus fresh fruit and alcoholic drinks
- a large value of z_{i1} can result from eating many fresh potatoes and drinking a lot of soft drinks, but eating only few fresh fruit and drinking not much alcoholic drinks
- a small value of z_{i1} can result from eating few fresh potatoes and drinking only few soft drinks, but eating a lot of fresh fruit and drinking much alcoholic drinks

For $\mathbf{v}_2$ (2nd dimension of $\mathbf{Z}_2$):

- contrast of soft drinks versus fresh potatoes
- a large value of z_{i2} can result from eating many fresh potatoes, but drinking not much soft drinks
- a small value of z_{i2} can result from eating few fresh potatoes, but drinking much soft drinks .

The elements in v1 and v2 are also shown in the biplot.

```
grid.arrange(p1, p2, diyBiplot, ncol=2, layout_matrix = rbind(c(1,2),c(3,3)))
```



From the graph we see e.g. that N. Ireland has a high score for the first dimension. Now that we can give an interpretation to the dimension, we conclude that in Northern Ireland people eat relatively much fresh potatoes and drink many soft drinks, but they do not drink much alcoholic drinks and eat not much fresh fruit. We had already come to these findings by projecting N. Ireland on the vectors of these four food products. This comes to no surprise: both interpretations arise from the same data set and its SVD, and so no contradictory results should arise. Other conclusions can be found in a similar fashion.

We have derived the interpretation of the first dimension from the barplot of the elements of $\mathbf{v}_1$. However, there is actually no need to make a separate plot to read the elements of $\mathbf{v}_1$; they can also be read from projecting the vectors in the biplot onto the first dimension (i.e. basis vector of the first dimension). For example, fresh potatoes is in the 7th column of $\mathbf{X}$ and thus in the biplot its vector is $\tilde{\mathbf{v}}_{2,7}$; it is a 2-dimensional vector (2-dimensional biplot is shown). In the space of the biplot (i.e. the column space of $\mathbf{Z}_2$), the first basis vector is given by $(1, 0)$. Projecting $\tilde{\mathbf{v}}_{2,7}^T$ orthogonally onto $(1, 0)$ gives

$$\tilde{\mathbf{v}}_{2,7}^T \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (v_{71}, v_{72}) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = v_{71}$$

which is the seventh element of the first right-singular vector of $\mathbf{X}$, which is thus the bar of fresh potatoes in the barplot of the first dimension.

8 SVD and principal component analysis (PCA)

- PCA is basically a SVD

- PCA adds another layer of interpretation
- PCA comes with its own terminology
- One of the most widely used algorithms for dimension reduction and data exploration of multivariate and high dimensional data.
- It is motivated from the decomposition of the Variance covariance matrix of the data

8.1 Variance covariance matrix

- For a given centered data matrix $\mathbf{X}$ the $p \times p$ covariance matrix can be estimated by

$$\Sigma_X = \frac{1}{n-1}\mathbf{X}^T\mathbf{X} = \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T,$$

  i.e. the (i,j)th element is given by (column means are zero)

$$\frac{1}{n-1}\sum_{m=1}^{n}x_{mi}x_{mj} = \frac{1}{n-1}\sum_{m=1}^{n}(x_{mi}-\bar{x}_i)(x_{mj}-\bar{x}_j).$$

Note, that when we forget to write the factor $1/(n-1)$ all the derivations still hold. It is only a proportionality factor and it does not affect the interpretation.

8.2 Conventional derivation of PCA

PCA is usually introduced as follows.

Let

$$y_i = \mathbf{x}_i^T\mathbf{a}$$

with $\mathbf{a}$ a p-vector of constants. Hence y_i is a linear combination (or transformation) of $\mathbf{x}_i$.

PCA aims at finding $\mathbf{a}$ such that the sample variance among the n y_i's is maximal, with

$$\begin{aligned}\frac{1}{n-1}\sum_{i=1}^{n}(y_i-\bar{y})^2 = \frac{1}{n-1}\sum_{i=1}^{n}y_i^2 & \\ &= \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i^T\mathbf{a})^2 = \frac{1}{n-1}\mathbf{a}^T\left(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T\right)\mathbf{a} \\ &= \frac{1}{n-1}\mathbf{a}^T\mathbf{X}^T\mathbf{X}\mathbf{a} = \mathbf{a}^T\Sigma_X\mathbf{a}\end{aligned}$$

in which $\Sigma_X = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$ is the sample covariance matrix of $\mathbf{X}$.

8.2.1 Problem for optimisation

Finding $\mathbf{a}$ that maximises

$$\text{var}[y] = \mathbf{a}^T\Sigma_X\mathbf{a}$$

has a trivial solution (set all elements of $\mathbf{a}$ equal to ∞). To avoid the trivial solution, the solution must satisfy a restriction, e.g.

$$\|\mathbf{a}\|^2 = \mathbf{a}^T\mathbf{a} = 1.$$

The solution of the constrained maximisation problem may be formulated as

$$\mathbf{a} = \text{ArgMax}_{b:\|b\|^2=1} \mathbf{b}^T \Sigma_X \mathbf{b}.$$

By introducing a Lagrange multiplier λ, we get an unconstrained maximisation problem,

$$\mathbf{a} = \text{ArgMax}_b \left(\mathbf{b}^T \Sigma_X \mathbf{b} - \lambda(\mathbf{b}^T \mathbf{b} - 1) \right).$$

Note that,

- If the constraint is linear, then the constrained optimisation problem (here: maximisation) may be replaced by an unconstrained optimisation problem of the same criterion but with an "penalty term" added. This method is due to Lagrange. The penalty term vanishes when the constraint is satisfied.
- If the constraint is satisfied, $\mathbf{b}^T \mathbf{b} = \|\mathbf{b}\|^2 = 1$ and thus $\mathbf{b}^T \mathbf{b} - 1 = 0$ and the "penalty" term in the unconstrained criterion vanishes.
- The Lagrange multiplier λ is to be considered as an extra parameter that may have to be estimated from the data.

The solution of

$$\mathbf{a} = \text{ArgMax}_b \left(\mathbf{b}^T \Sigma_X \mathbf{b} - \lambda(\mathbf{b}^T \mathbf{b} - 1) \right)$$

is obtained by differentiating $\mathbf{b}^T \Sigma_X \mathbf{b} - \lambda(\mathbf{b}^T \mathbf{b} - 1)$ w.r.t. $\mathbf{b}$, equating it to zero and solving for $\mathbf{b}$.

$$\begin{aligned} \frac{\partial}{\partial \mathbf{b}} \left(\mathbf{b}^T \Sigma_X \mathbf{b} - \lambda(\mathbf{b}^T \mathbf{b} - 1) \right) &= 0 \\ 2\Sigma_X \mathbf{b} - 2\lambda \mathbf{b} &= 0. \end{aligned}$$

Hence, we need the solution of

$$\Sigma_X \mathbf{b} = \lambda \mathbf{b}.$$

This equation has r solutions:

- $\mathbf{b} = \mathbf{e}_j$: the jth eigenvector of Σ_X
- $\lambda = \lambda_j$: the jth eigenvalue of Σ_X.

The eigenvectors are orthonormal, i.e. $\mathbf{e}_i^T \mathbf{e}_j = 1$ if $i = j$ and $\mathbf{e}_i^T \mathbf{e}_j = 0$ otherwise.

Consider the following calculations, with $\mathbf{b} = \mathbf{e}_j$, the $j$the eigenvector of Σ_X.

$$\begin{aligned} \text{var}[y] &= \mathbf{e}_j^T \Sigma_X \mathbf{e}_j \\ &= \mathbf{e}_j^T (\Sigma_X \mathbf{e}_j) \\ &= \mathbf{e}_j^T \left(\lambda_j \mathbf{e}_j \right) \\ &= \lambda_j \mathbf{e}_j^T \mathbf{e}_j \\ &= \lambda_j. \end{aligned}$$

By convention the eigenvectors/eigenvalues are ordered so that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r.$$

Hence, the first eigenvector $\mathbf{e}_1$ gives the largest variance λ_1. Hence, this is the solution we were looking for.

We now switch notation and from now onwards we denote y by z.

The observations on the first principal component (PC) are then given by

$$z_{i1} = \mathbf{x}_i^T \mathbf{e}_1.$$

In PCA terminology they are referred to as the **scores** of the first PC. The elements in the eigenvector $\mathbf{e}_i$ that make up the transformation are known as the **loadings** of the first PC.

The first PC is thus a new variable (construct) that has the largest variance among all linear transformations of the original variables.

Variability between observations is considered as informative to understand differences between the observations.

The equation

$$\Sigma_X \mathbf{b} = \lambda \mathbf{b}$$

has $r = \text{rank}(\Sigma_X)$ solutions. The eigenvectors are orthonormal, i.e.

$$\forall i \neq j : \mathbf{e}_i^T \mathbf{e}_j = 0 \text{ and } i = j : \mathbf{e}_i^T \mathbf{e}_j = 1$$

Hence (for $i \neq j$)

$$\text{cov}\left[\mathbf{x}^T \mathbf{e}_i, \mathbf{x}^T \mathbf{e}_j\right] = \mathbf{e}_i^T \text{var}[\mathbf{x}] \mathbf{e}_j = \mathbf{e}_i^T \Sigma_X \mathbf{e}_j = \mathbf{e}_i^T \left(\lambda_j \mathbf{e}_j\right) = \lambda_j \mathbf{e}_i^T \mathbf{e}_j = 0$$

If $z_j = \mathbf{x}_i^T \mathbf{e}_j$ denotes the jth PC, then

$$\text{cov}\left[z_i, z_j\right] = \text{cov}\left[\mathbf{x}^T \mathbf{e}_i, \mathbf{x}^T \mathbf{e}_j\right] = 0 \text{ if } i \neq j$$

$$\text{var}\left[z_j\right] = \lambda_j.$$

We say that the jth PC maximises the variance among all linear transformations such that it is uncorrelated with the previous PCs.

8.2.2 Interpretation of PCA

A PCA is a transformation of the original p variables to r PCs such that

- the first PC has largest variance, equal to first eigenvalue of Σ_X
- the next PCs have decreasing variances (decreasing information content)
- all PCs are mutually uncorrelated (no information-overlap).

8.3 Link with SVD

If we write the eigen value decomposition in matrix form:

$$\Sigma_X \mathbf{B} = \mathbf{B}\Lambda.$$

with Λ is a diagonal matrix with diagonal elements $[\lambda_i]_{ii}$.

We recognise an expression that we have seen when discussing the link between the SVD and the sample covariance matrix

$$\mathbf{X}^T\mathbf{X}\mathbf{V} \quad = \quad \mathbf{V}\Delta^2 \tag{26}$$

- So, the SVD can be used to solve the spectral decomposition (eigen value eigen vector) problem and the eigen vectors coincide with the right singular vectors and the eigen values are the singular values squared!

8.4 Conservation of variance

- The total variance in a data set $\mathbf{X}$ is given by the sum of the variances of the variables,

$$\sigma^2_{\text{tot}} = \sum_{j=1}^{p} \text{var}\left[X_j\right] = \text{trace}(\Sigma_X)$$

- For a symmetric matrix it holds that

$$\text{trace}(\Sigma_X) = \sum_{j=1}^{r} \lambda_r.$$

- Since $\lambda_r = \text{var}\left[Z_j\right]$, we find

$$\sigma^2_{\text{tot}} = \sum_{j=1}^{p} \text{var}\left[X_j\right] = \sum_{j=1}^{r} \text{var}\left[Z_j\right].$$

$\longrightarrow$ Thus no information is lost by the PCA transformation from the original p-dimensional space to the r-dimensional PCA space.

8.5 Choosing the number of dimensions

For PCA the reasoning is usually based on the relative variance of a PC,

$$\frac{\text{var}\left[Z_j\right]}{\sigma^2_{\text{tot}}} = \frac{\lambda_j}{\sum_{i=1}^{r}\lambda_i} = \frac{\delta_j^2}{\sum_{i=1}^{r}\delta_i^2}.$$

If the first k PCs are selected for further use, then they represent

$$100 \times \frac{\sum_{j=1}^{k}\lambda_j}{\sum_{i=1}^{r}\lambda_i}\% = 100 \times \frac{\sum_{j=1}^{k}\delta_j^2}{\sum_{i=1}^{r}\delta_i^2}\%$$

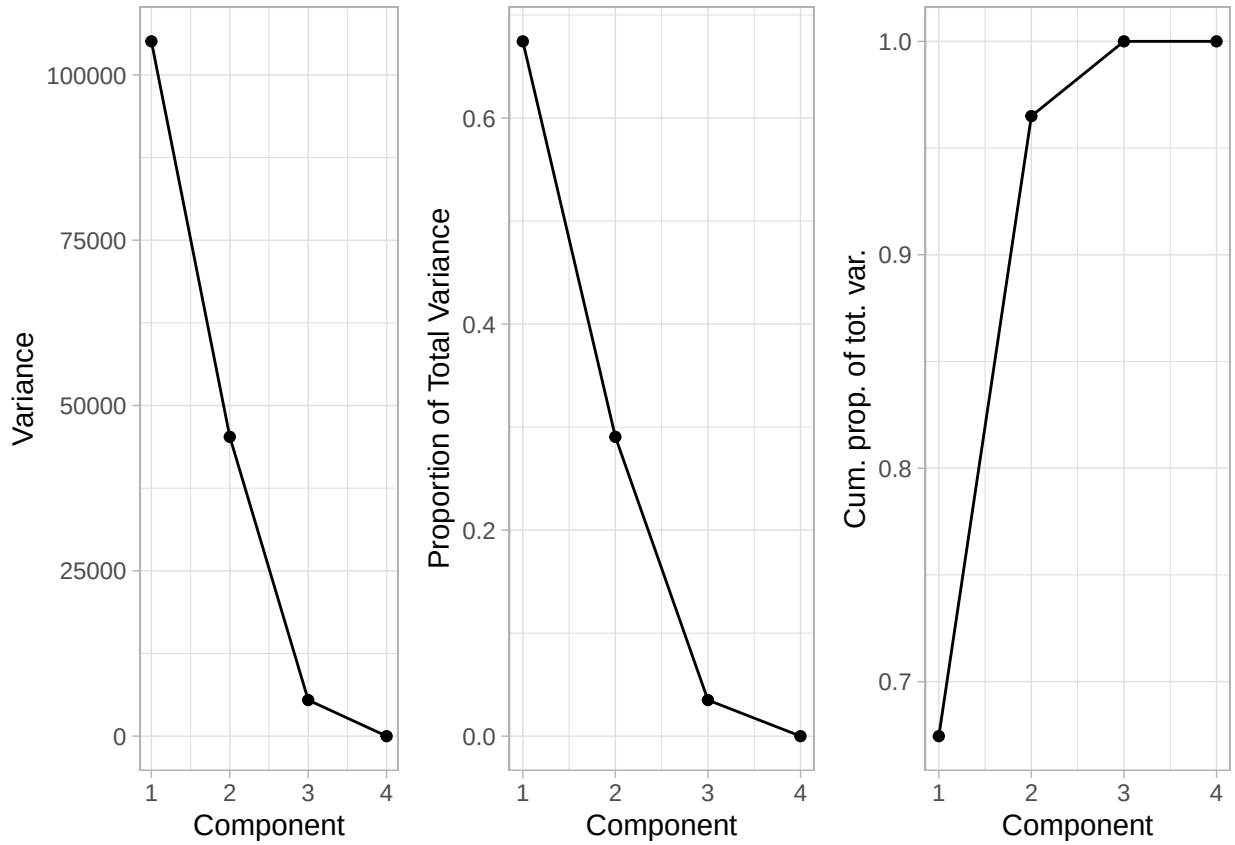
of the total variance (or information) of the original data set $\mathbf{X}$.

8.5.1 UK example

A scree plot is often used to look at the eigenvalues. Here it is shown for the UK consumption data.

```
n <- nrow(X)
r <- ncol(svdUk$v)
totVar <- sum(svdUk$d^2)/(n-1)
vars <- data.frame(comp=1:4,var=svdUk$d^2/(n-1)) %>%
  mutate(propVar=var/totVar,cumVar=cumsum(var/totVar))

pVar1 <- vars %>%
  ggplot(aes(x=comp:r,y=var)) +
  geom_point() +
  geom_line() +
  xlab("Component") +
  ylab("Variance")

pVar2 <- vars %>%
  ggplot(aes(x=comp:r,y=propVar)) +
  geom_point() +
  geom_line() +
  xlab("Component") +
  ylab("Proportion of Total Variance")

pVar3 <- vars %>%
  ggplot(aes(x=comp:r,y=cumVar)) +
  geom_point() +
  geom_line() +
  xlab("Component") +
  ylab("Cum. prop. of tot. var.")

grid.arrange(pVar1, pVar2, pVar3, nrow=1)
```

```
## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used

## Warning in comp:r: numerical expression has 4 elements: only the first used
```



From these graphs we may conclude that with using k = 2 dimensions, more than 95% of the total variance is retained.

8.5.2 Choosing the number of dimensions SVD

More generally, scree plots are also informative for the SVD.

The motivation comes from

$$\text{Min}_{A:\text{rank}(A)=k}\|\mathbf{X}-\mathbf{A}\|_F^2 = \|\mathbf{X}-\mathbf{X}_k\|_F^2 = \sum_{j=k+1}^{r}\delta_j^2 = \sum_{j=k+1}^{r}\lambda_j.$$

This is known as the **approximation error** of the matrix $\mathbf{X}_k$.

Hence,

$$\frac{\sum_{j=1}^{k}\delta_j^2}{\sum_{j=1}^{r}\delta_j^2} = \frac{\sum_{j=1}^{k}\lambda_j}{\sum_{j=1}^{r}\lambda_j}$$

still makes sense as a relative quality measure for the SVD in general (including matrix approximation and MDS).

8.5.3 Rules of thumb to select number of dimensions

Select k such that

- at least 80% of the total variance is retained in the PC space, or, equivalently, at most 20% relative approximation error
- adding more dimensions does not add more information (in a relative sense); this can often be visually detected as the "knee" or "elbow" in the scree plot
- none of the dimensions has a variance smaller than the variance of one of the p original variables (this rule is typically only used when the variables are first standardised - see later).

8.6 Covariance vs Correlation Matrix

8.6.1 Direction of largest variability

Now that we know how the PCs are constructed, we can give an additional interpretation to the first eigenvector / right-singular vector of Σ_X.

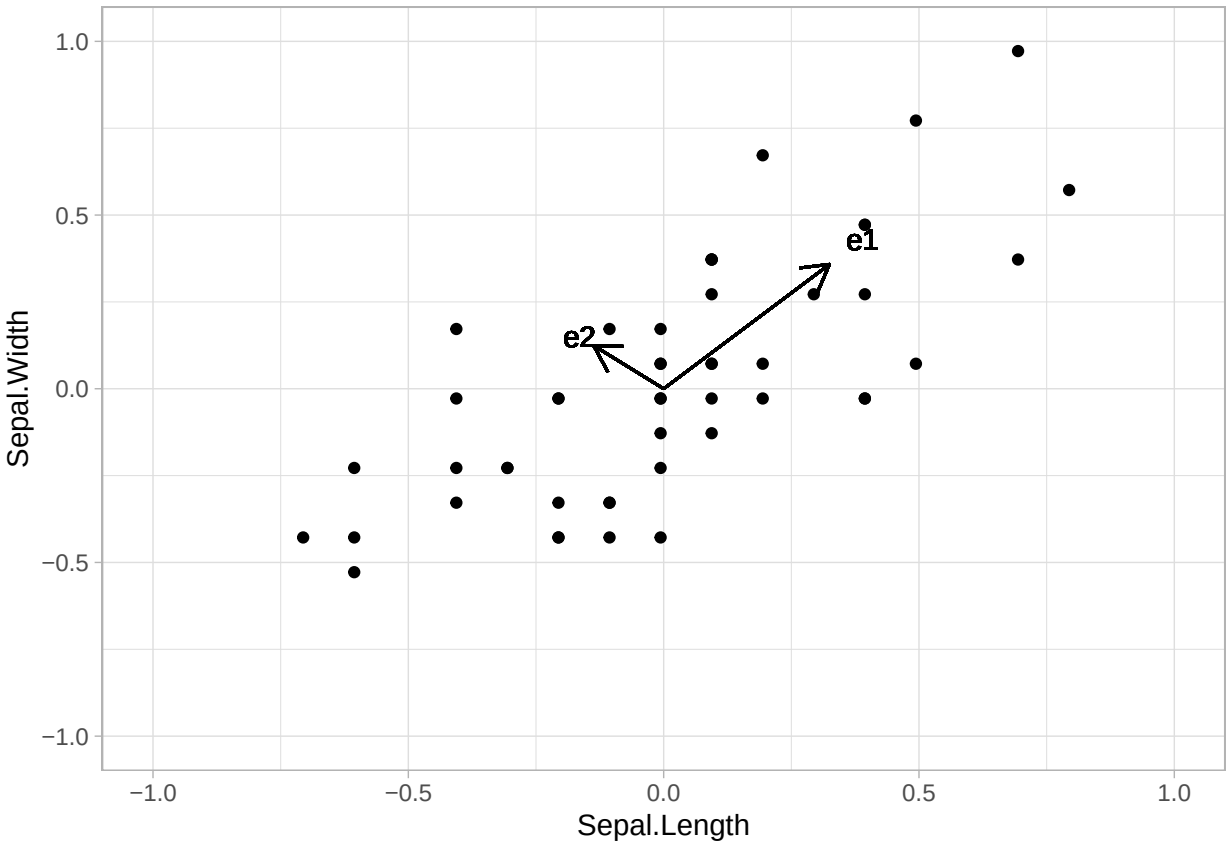
- The first eigenvector of Σ_X is the direction in the original p-dimensional space of the largest variability.
- The second eigenvector of Σ_X is the direction in the original p-dimensional space of the second largest variability, among all directions orthogonal to the first eigenvector

8.6.1.1 Iris example The iris dataset in R contains information on leaves of iris flowers from different species. Here, we will focus on the setosa species and on the sepal length and sepal width.

```
irisSetosa <- iris %>%
  filter(Species == "setosa") %>%
  dplyr::select("Sepal.Length","Sepal.Width")

nIris <- nrow(irisSetosa)
hIris <- diag(nIris) - matrix(1/nIris, nIris, nIris)
irisX <- irisSetosa %>%
  as.matrix
irisX <- hIris %*% irisX

irisSvd <- svd(irisX)

pIris <- irisX %>%
  as.data.frame %>%
  ggplot(aes(x=Sepal.Length,y=Sepal.Width)) +
  geom_point()

pIris <- pIris +
  geom_segment(
    aes(
      x = 0,
      y = 0,
      xend = -irisSvd$v[1,1]*irisSvd$d[1]/sqrt(nIris-1),
      yend = -irisSvd$v[2,1]*irisSvd$d[1]/sqrt(nIris-1)
      ),
      arrow = arrow(length=unit(0.4,"cm"))
    ) +
  geom_segment(
    aes(
```

```
      x = 0,
      y = 0,
      xend = irisSvd$v[1,2]*irisSvd$d[2]/sqrt(nIris-1),
      yend = irisSvd$v[2,2]*irisSvd$d[2]/sqrt(nIris-1)
      ),
    arrow = arrow(length=unit(0.4,"cm"))
    ) +
  geom_text(
    aes(
      x = -irisSvd$v[1,1]*irisSvd$d[1]/sqrt(nIris-1)*1.2,
      y = -irisSvd$v[2,1]*irisSvd$d[1]/sqrt(nIris-1)*1.2,
      label="e1"
      )
    ) +
  geom_text(
    aes(
      x = irisSvd$v[1,2]*irisSvd$d[2]/sqrt(nIris-1)*1.2,
      y = irisSvd$v[2,2]*irisSvd$d[2]/sqrt(nIris-1)*1.2,
      label="e2"
      )
    ) +
  xlim(-1, 1) +
  ylim(-1,1)

pIris
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



8.6.2 Covariance versus correlation matrix

So far we have worked with

- the column-centered data matrix $\mathbf{X}$
- the SVD of $\mathbf{X}$ and $\Sigma_X \propto \mathbf{X}^T\mathbf{X}$ (the covariance matrix).

In some situations it is better to start from the standardised variables:

- substract from each element in $\mathbf{X}$ the column-mean
- divide each centered element in $\mathbf{X}$ by the column-specific standard deviation

Thus each element x_{ij} is replaced with

$$\frac{x_{ij} - \bar{x}_j}{s_j},$$

with $\bar{x}_j$ and s_j the column mean and column standard deviation.

- In matrix notation, the standardisation, starting from the centered matrix $\mathbf{X}$ is computed as

  $$\mathbf{X}\mathbf{S}'^{-1}$$

  where $\mathbf{S}'$ is a diagonal matrix with the column-specific standard deviations.

```
var(irisX)
```

8.6.2.1 Iris Example

```
##              Sepal.Length Sepal.Width
## Sepal.Length   0.12424898  0.09921633
## Sepal.Width    0.09921633  0.14368980
```

```
Sprime <- diag(sqrt(diag(var(irisX))))
Xs <- irisX%*%solve(Sprime)
var(Xs)
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.7425467
## [2,] 0.7425467 1.0000000
```

Much faster to use the scale function. By default center=TRUE and scale=TRUE.

```
Xs <- irisSetosa %>% scale
var(Xs)
```

```
##              Sepal.Length Sepal.Width
## Sepal.Length    1.0000000   0.7425467
## Sepal.Width     0.7425467   1.0000000
```

Show problem of different units:

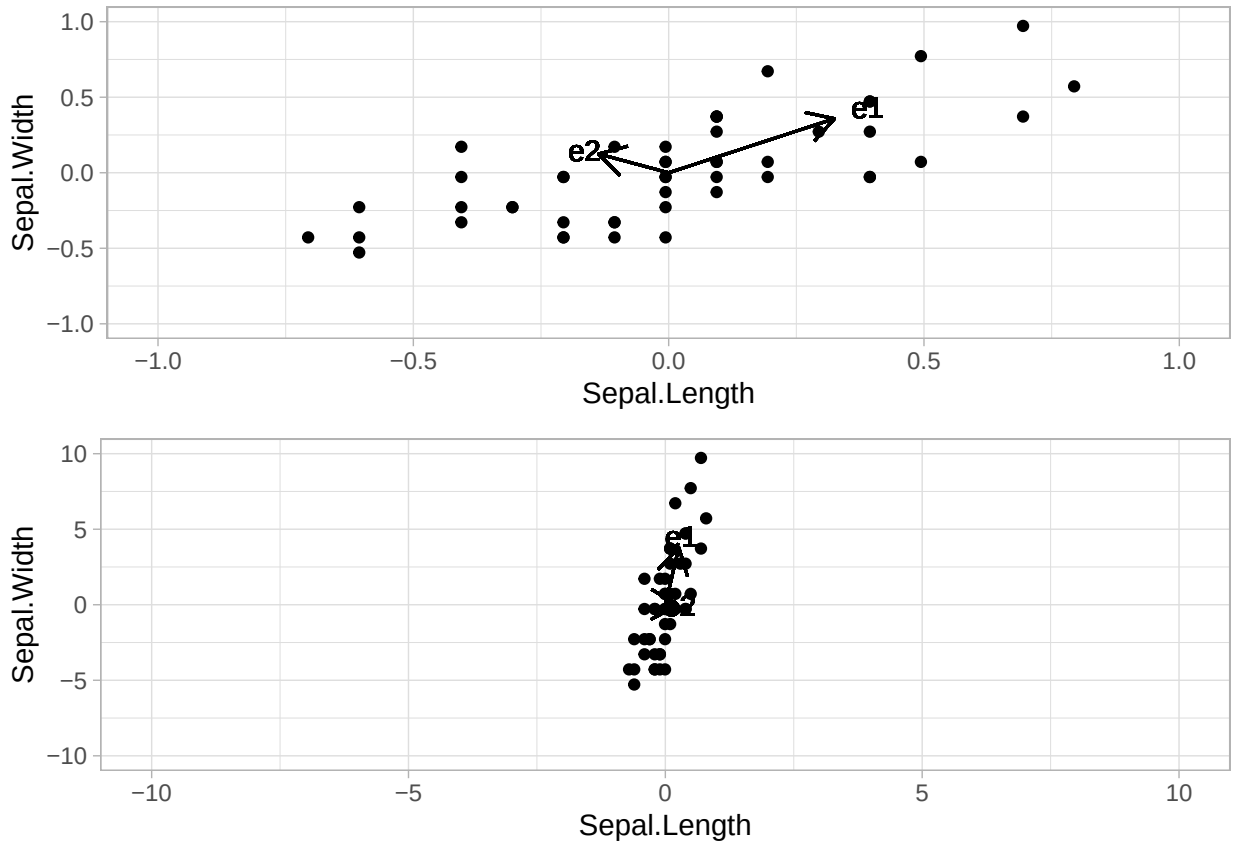
- Sepal length in cm
- Sepal width in mm

```
irisX2 <- irisX
irisX2[,2] <- irisX2[,2]*10

pIris2 <- irisX2 %>%
  as.data.frame %>%
  ggplot(aes(x=Sepal.Length,y=Sepal.Width)) +
  geom_point()

irisSvd2 <- svd(irisX2)

pIris2 <- pIris2 +
  geom_segment(
    aes(
      x = 0,
      y = 0,
      xend = irisSvd2$v[1,1]*irisSvd2$d[1]/sqrt(nIris-1),
      yend = irisSvd2$v[2,1]*irisSvd2$d[1]/sqrt(nIris-1)
      ),
      arrow = arrow(length=unit(0.4,"cm"))
```

```
    ) +
  geom_segment(
    aes(
      x = 0,
      y = 0,
      xend = irisSvd2$v[1,2]*irisSvd2$d[2]/sqrt(nIris-1),
      yend = irisSvd2$v[2,2]*irisSvd2$d[2]/sqrt(nIris-1)
      ),
    arrow = arrow(length=unit(0.4,"cm"))
    ) +
  geom_text(
    aes(
      x = irisSvd2$v[1,1]*irisSvd2$d[1]/sqrt(nIris-1)*1.2,
      y = irisSvd2$v[2,1]*irisSvd2$d[1]/sqrt(nIris-1)*1.2,
      label="e1"
      )
    ) +
  geom_text(
    aes(
      x = irisSvd2$v[1,2]*irisSvd2$d[2]/sqrt(nIris-1)*1.2,
      y = irisSvd2$v[2,2]*irisSvd2$d[2]/sqrt(nIris-1)*1.2,
      label="e2"
      )
    ) +
  xlim(-10, 10) +
  ylim(-10,10)

grid.arrange(pIris, pIris2)
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```



Top: original matrix – Bottom: second column of $\mathbf{X}$ multiplied by 10 (e.g. moving from cm to mm). Directions of maximal variability are affected by the units of the variables.

8.6.3 Recommendations

When to use correlation and when covariance?

- use correlation when columns of $\mathbf{X}$ are expressed in different units
- use covariance when columns of $\mathbf{X}$ are expressed in the same units.

There may be exceptions.

8.7 PCA and the Multivariate Normal Distribution

The density function of a multivariate normal distribution (MVN) is given by

$$f(\mathbf{x}) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right),$$

where

- μ is the multivariate mean vector (p-dimensional). The jth element is $\mu_j = \mathrm{E}\left[X_j\right]$
- Σ is the $p \times p$ covariance matrix. The (i, j)th element is $\sigma_{ij} = \mathrm{cov}\left[X_i, X_j\right]$.

To get a better understanding of the MVN we focus on the exponential which has the factor

$$(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)$$

This factor

- is the only factor in the density function that depends on $\mathbf{x}$
- is a **quadratic** form
- is constant in points $\mathbf{x}$ with constant density $f(\mathbf{x})$.

Consider $p = 2$ (bivariate normal). Then, all $\mathbf{x} \in \mathbb{R}^2$ for which

$$(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu) = \text{constant } c^2$$

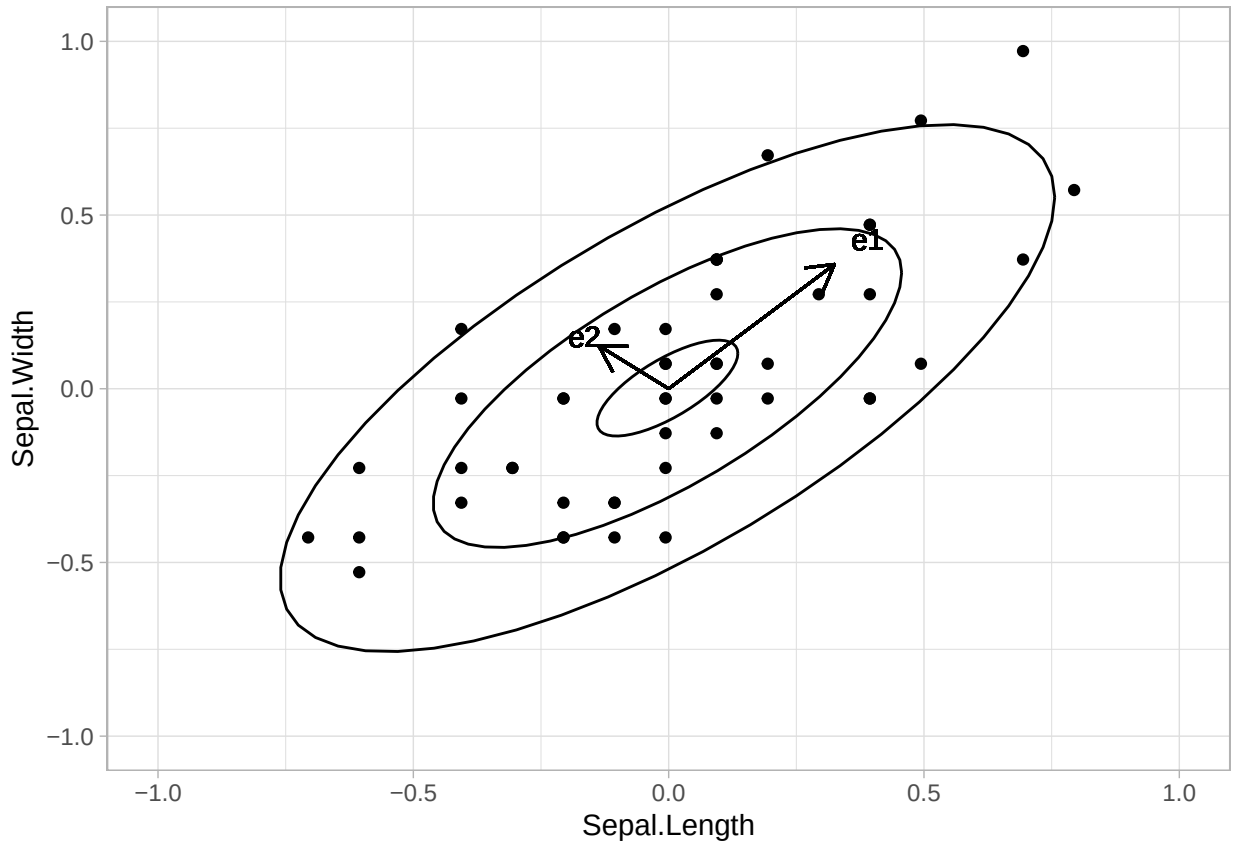
lie on an ellipse with center μ.

These ellipses are known as **constant density ellipses**.

8.7.1 iris example

```
pIris +
  stat_ellipse() +
  stat_ellipse(level=.68) +
  stat_ellipse(level=.1)
```

```
## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing missing values (geom_point).
```



Now plug in the SVD of $\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T$,

$$\begin{aligned}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu) &= c^2\\ (\mathbf{x}-\mu)^T\mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T(\mathbf{x}-\mu) &= c^2\end{aligned}$$

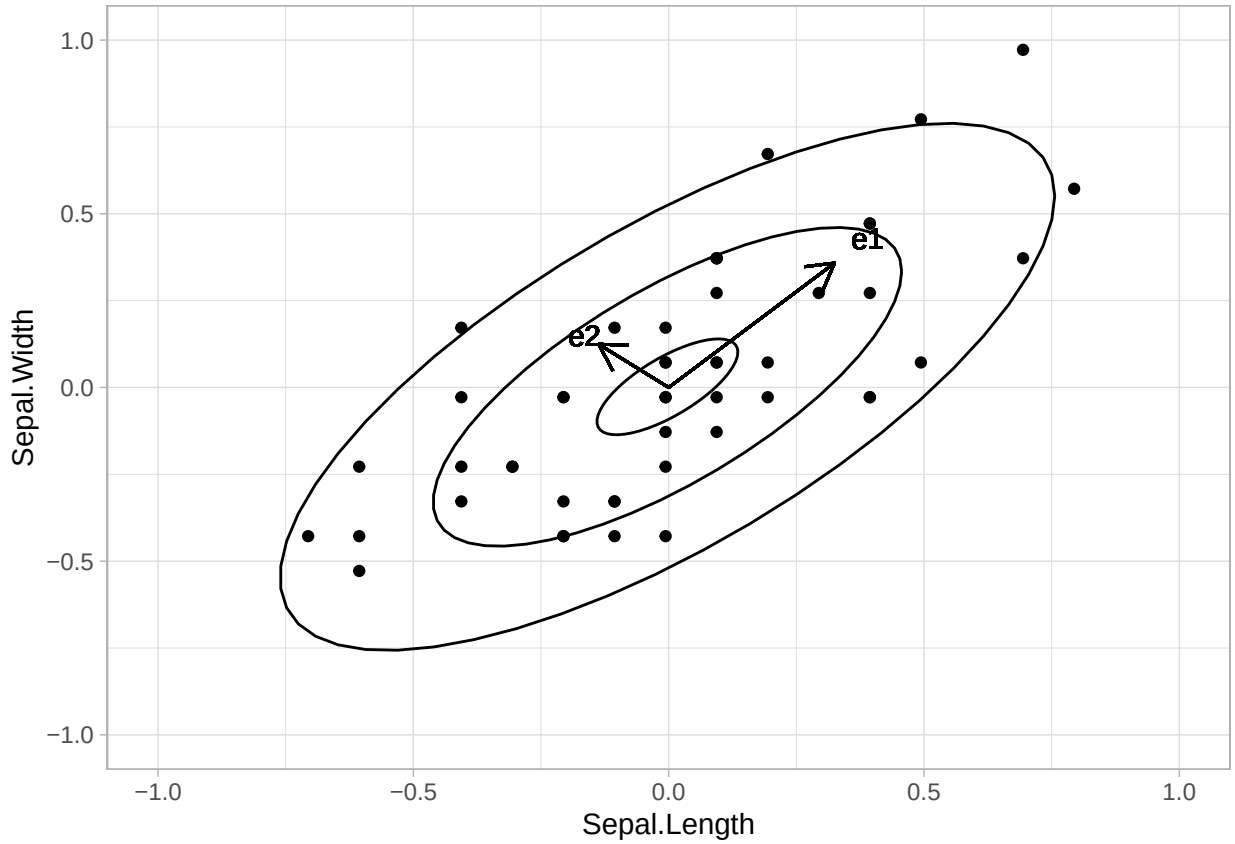
Note that $\mathbf{x}-\mu$ is the centered $\mathbf{x}$. Without loss of generality, take $\mu = \mathbf{0}$. Hence,

$$\begin{aligned}\mathbf{x}^T\mathbf{V}\mathbf{D}^{-1}\mathbf{V}^T\mathbf{x} &= c^2\\ \left(\mathbf{x}^T\mathbf{V}\right)\mathbf{D}^{-1}\left(\mathbf{x}^T\mathbf{V}\right)^T &= c^2\\ \sum_{j=1}^{p}(\mathbf{x}^T\mathbf{v}_j)^2/\delta_j &= c^2\\ \sum_{j=1}^{p}(z_j)^2/(c^2\delta_j) &= 1.\end{aligned}$$

The last equation is the equation of an ellipse with axes parallel to the basis of $(z_1, \ldots, z_p)$ and with half axis lengths $c\sqrt{\lambda_j} = c\delta_j$ with λ_j the jth eigenvalue of Σ.

```
pIris +
  stat_ellipse() +
  stat_ellipse(level=.68) +
  stat_ellipse(level=.1)
```

```
## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing non-finite values (stat_ellipse).

## Warning: Removed 1 rows containing missing values (geom_point).
```



This graphs shows $n = 50$ data points on $p = 2$ dimensions. The two ellipses are constant density ellipses for three different values of c (i.e. three different constant densities). The inner ellipse corresponds to the largest constant density and the outer to the smallest constant density. The arrows show the two eigenvectors / singular vectors and they are scaled according to the sqrt of the eigen values and they form the axes of the constant density ellipses. It is clear that the first axis is pointing to the larger variance.

8.8 Biplot

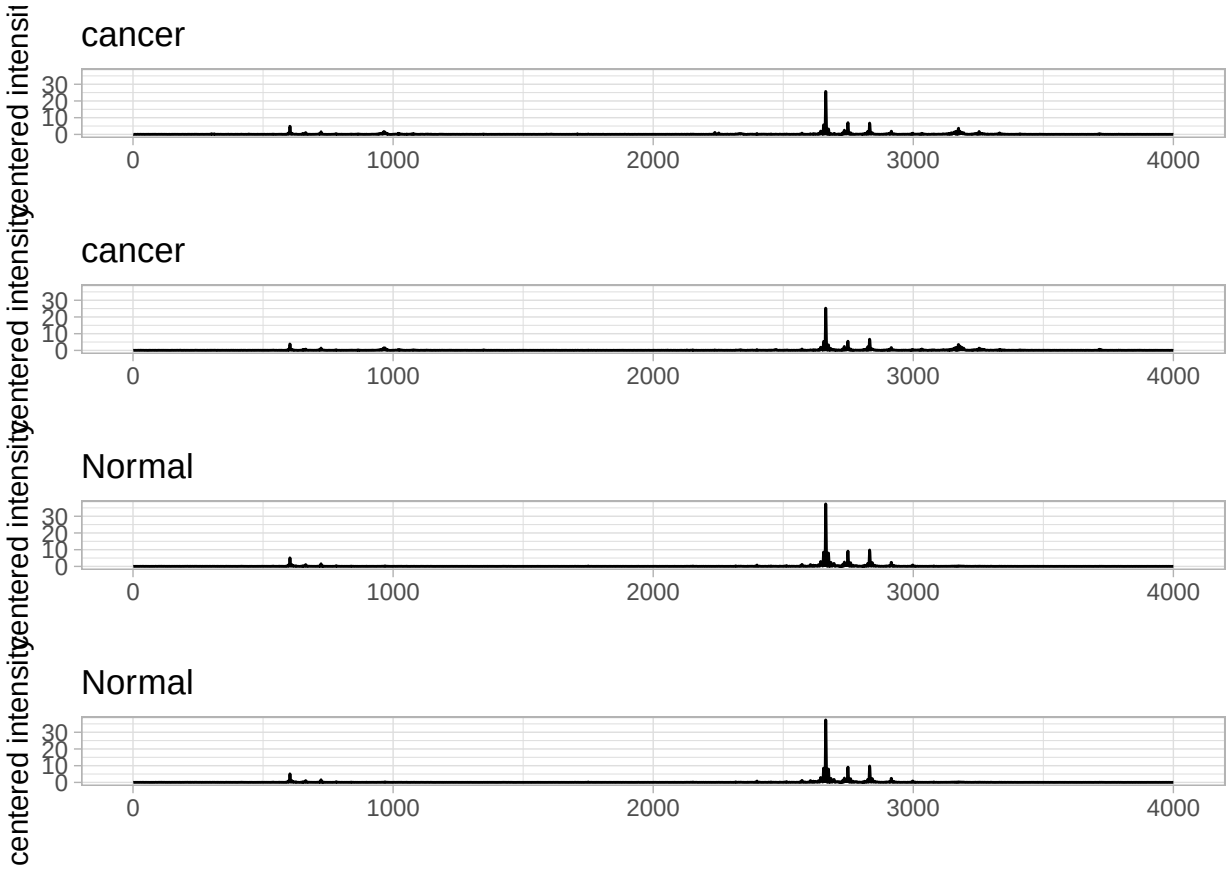
- Because of the close connection between PCA and the SVD, the biplot as discussed before is still meaningful, with the first axis pointing into the direction of largest variance.

8.9 Ovarian Cancer Example

The ovarian cancer data set consists of proteomics data for 216 patients, 121 of whom have ovarian cancer, and 95 of whom do not. For each subject, the expression of 4000 spectral features is assessed. The first 121 rows consist of data for the cancer patients.

8.9.1 Importing the data

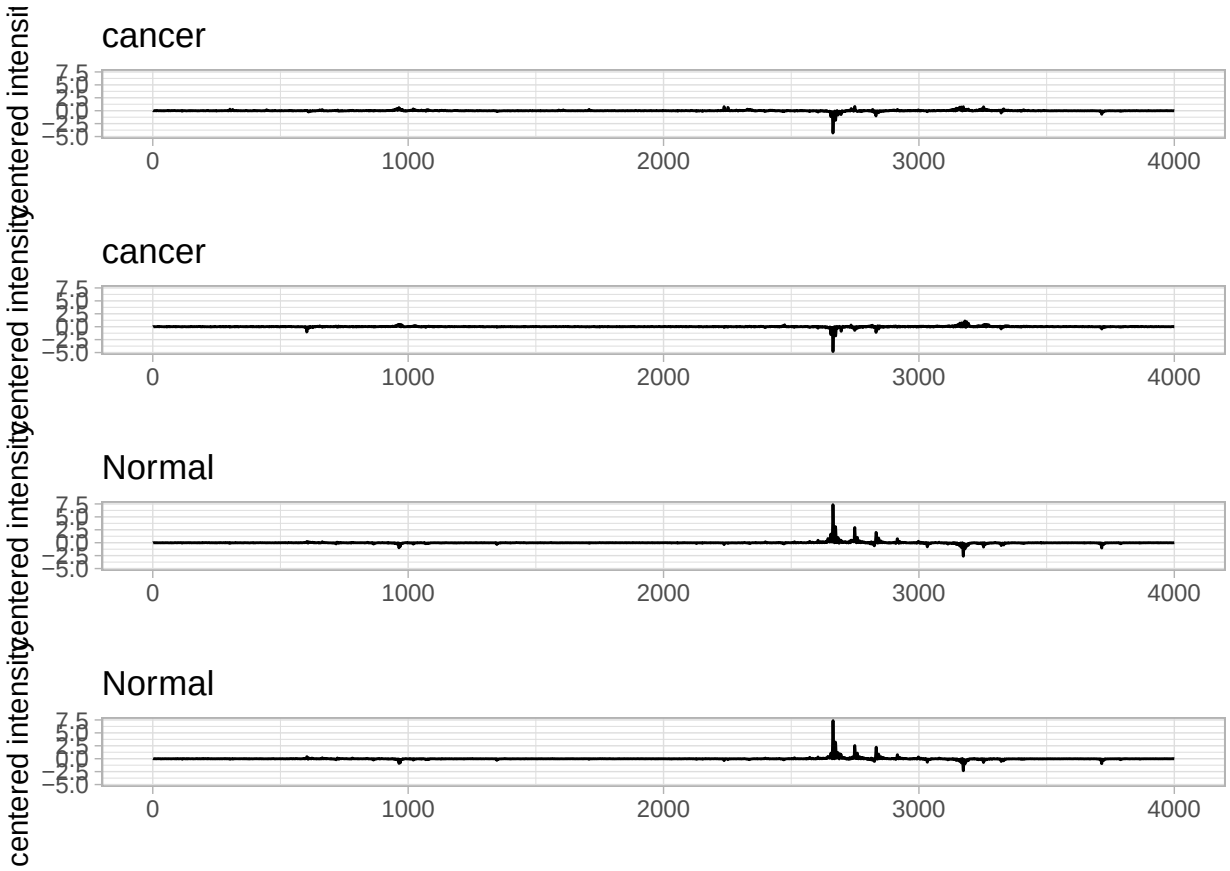
```
ovarian <- read_csv(
  "https://raw.githubusercontent.com/statOmics/HDA2020/data/ovarian.csv",
  col_names = FALSE,
  col_types = cols()
)
grid.arrange(
  qplot(1:4000,
    ovarian[1,] %>% unlist,
    geom="line",
    ylab="centered intensity",
    xlab="",main="cancer",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[2,] %>% unlist,
    geom="line",
    ylab="centered intensity",
    xlab="",main="cancer",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[200,] %>% unlist,
    geom="line",
    ylab="centered intensity",
    xlab="",main="Normal",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[200,] %>% unlist,
    geom="line",
    ylab="centered intensity",
    xlab="",main="Normal",
    ylim=range(ovarian[c(1,2,200,201),])),
  ncol=1
  )
```



Centering

```
ovarian <-  scale(ovarian, scale=FALSE)
grid.arrange(
  qplot(1:4000,
    ovarian[1,],
    geom="line",
    ylab="centered intensity",
    xlab="",main="cancer",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[2,],
    geom="line",
    ylab="centered intensity",
    xlab="",main="cancer",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[200,],
    geom="line",
    ylab="centered intensity",
    xlab="",main="Normal",
    ylim=range(ovarian[c(1,2,200,201),])),
  qplot(1:4000,
    ovarian[201,],
    geom="line",
    ylab="centered intensity",
```

```
    xlab="",main="Normal",
    ylim=range(ovarian[c(1,2,200,201),])),
  ncol=1
  )
```

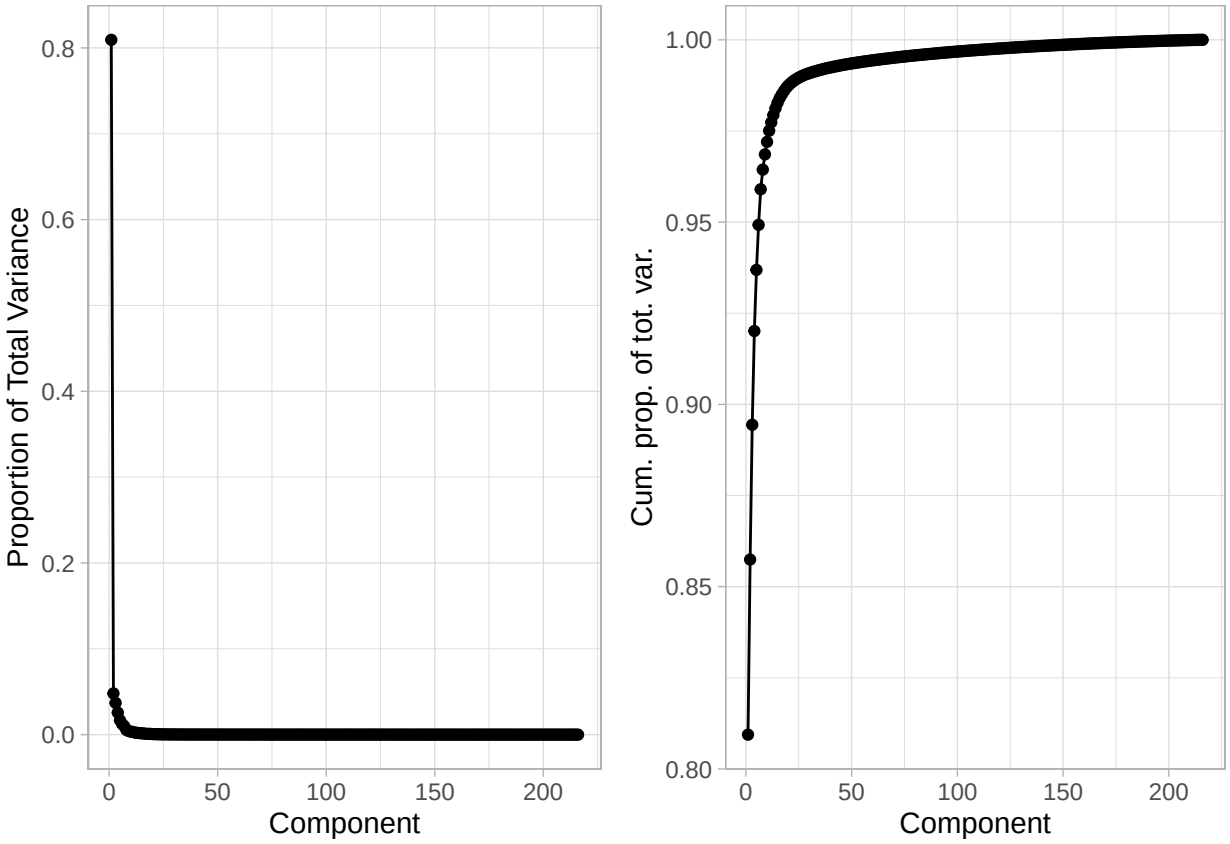


8.9.2 SVD Analysis

```
svdOvarian <- svd(ovarian)

nOvarian <- nrow(ovarian)
r <- ncol(svdOvarian$v)

totVar <- sum(svdOvarian$d^2)/(nOvarian-1)
vars <- data.frame(comp=1:r,var=svdOvarian$d^2/(nOvarian-1)) %>%
  mutate(propVar=var/totVar,cumVar=cumsum(var/totVar))

pVar2 <- vars %>%
  ggplot(aes(x=comp:r,y=propVar)) +
  geom_point() +
  geom_line() +
  xlab("Component") +
  ylab("Proportion of Total Variance")
```

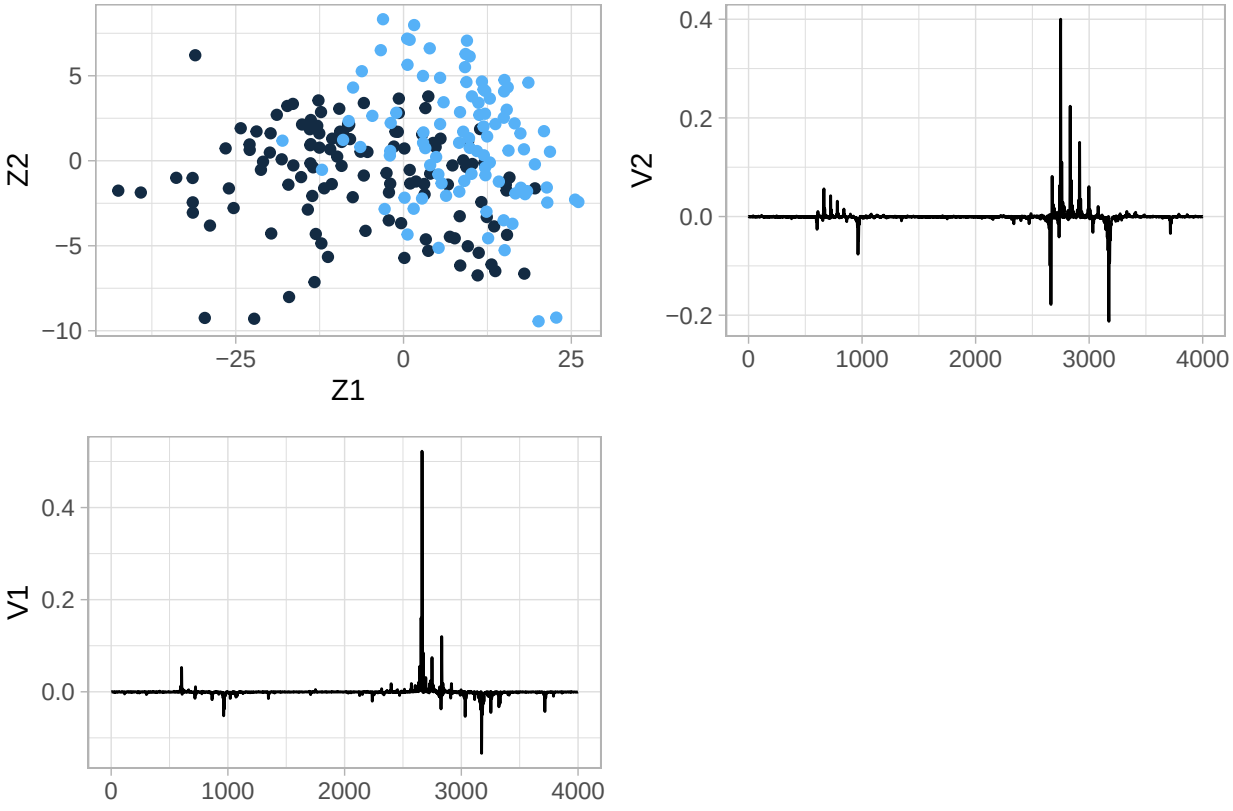
```
pVar3 <- vars %>%
  ggplot(aes(x=comp:r,y=cumVar)) +
  geom_point() +
  geom_line() +
  xlab("Component") +
  ylab("Cum. prop. of tot. var.")

grid.arrange(pVar2, pVar3, nrow=1)
```

```
## Warning in comp:r: numerical expression has 216 elements: only the first used

## Warning in comp:r: numerical expression has 216 elements: only the first used

## Warning in comp:r: numerical expression has 216 elements: only the first used

## Warning in comp:r: numerical expression has 216 elements: only the first used

## Warning in comp:r: numerical expression has 216 elements: only the first used

## Warning in comp:r: numerical expression has 216 elements: only the first used
```



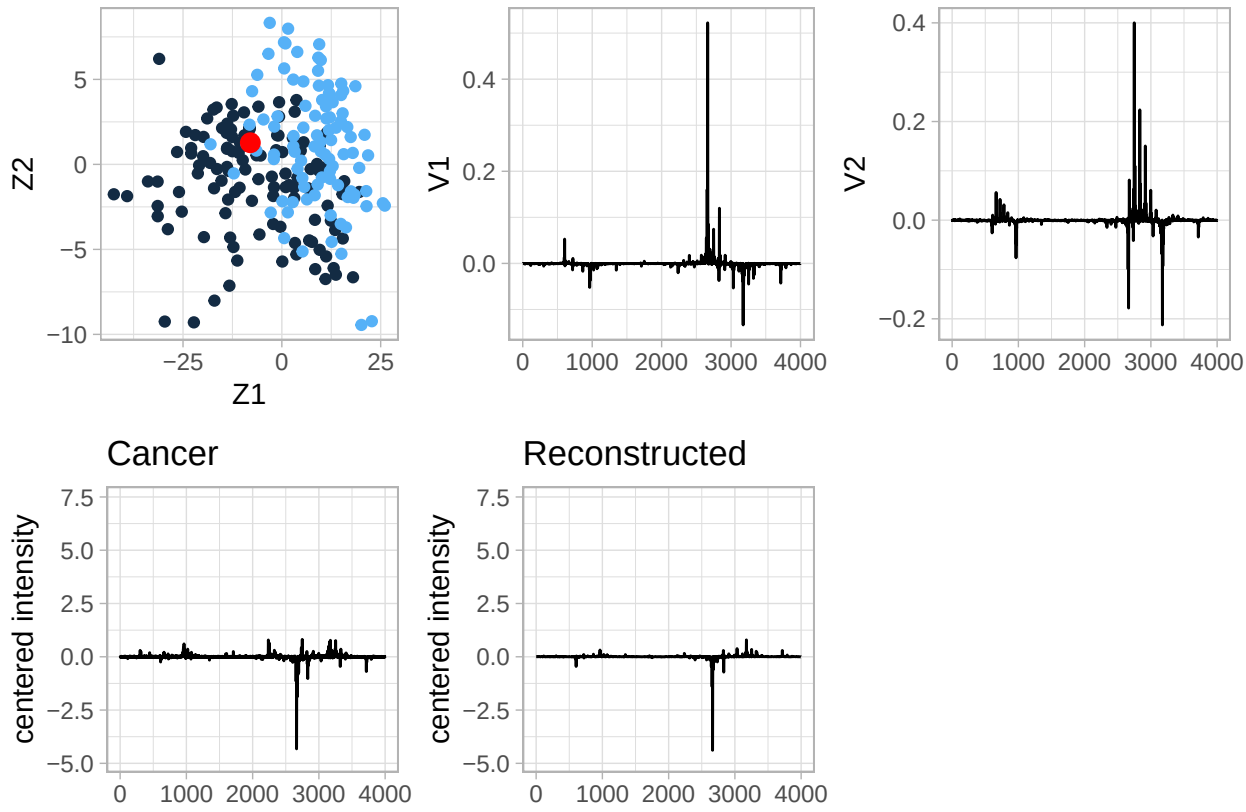
We see that we can explain a lot of the variability using a few PC's!

```
Zk <- svdOvarian$u[,1:6]%*%diag(svdOvarian$d[1:6])
colnames(Zk) <- paste0("Z",1:6)
Vk <- svdOvarian$v[,1:6]
colnames(Vk) <- paste0("V",1:6)
reduced <- data.frame(Zk,cancer=c(rep(1,121),rep(2,95)))

pOv1 <- reduced %>%
  ggplot(aes(x=Z1,y=Z2,col=cancer)) +
  geom_point()  +
  theme(legend.position = "none")

pOv2 <- Vk %>%
  as.data.frame %>%
  ggplot(aes(x=1:4000,y=V1)) +
  geom_line() +
  xlab("")

pOv3 <- Vk %>%
  as.data.frame %>%
  ggplot(aes(x=1:4000,y=V2)) +
  geom_line() +
  xlab("")

grid.arrange(pOv1,pOv2,pOv3,layout_matrix = rbind(c(1,3),c(2,NA)))
```

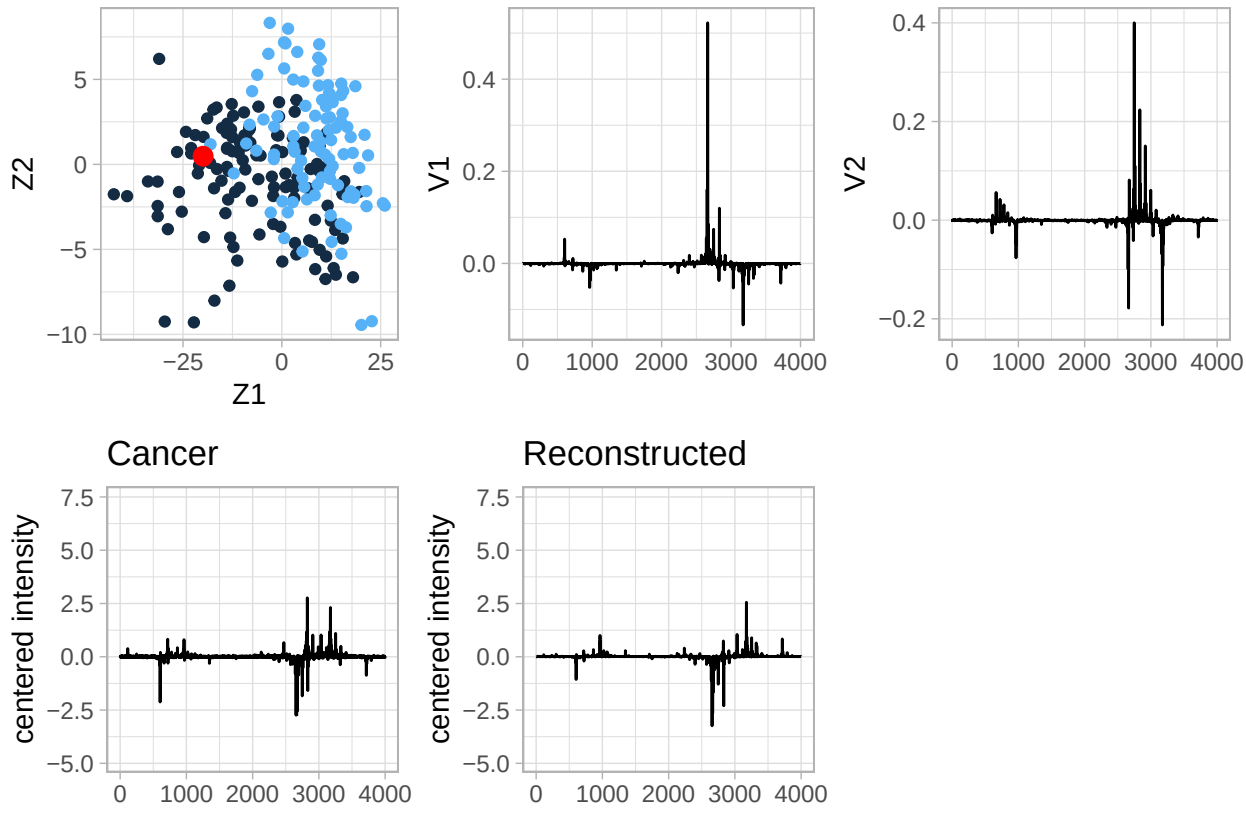


Reconstruction of profiles:

```
i <- 1
pOv4 <- qplot(1:4000,
  ovarian[i,],
  geom="line",
  ylab="centered intensity",
  xlab="",main="Cancer",
  ylim=range(ovarian[c(1,2,200,201),]))

pOv5 <- qplot(1:4000,
  Vk[,1:2] %*%Zk[i,1:2],
  geom="line",
  ylab="centered intensity",
  xlab="",main="Reconstructed",
  ylim=range(ovarian[c(1,2,200,201),]))



grid.arrange(
  pOv1 +
    annotate("point", x = reduced[i,1], y = reduced[i,2], colour = "red",cex=3),
  pOv2,
  pOv3,
  pOv4,
  pOv5,
  layout_matrix = rbind(c(1,2,3),c(4,5,NA)))
```



```
i <- 3
pOv4 <- qplot(1:4000,
  ovarian[i,],
  geom="line",
  ylab="centered intensity",
  xlab="",main="Cancer",
  ylim=range(ovarian[c(1,2,200,201),]))

pOv5 <- qplot(1:4000,
  Vk[,1:2] %*%Zk[i,1:2],
  geom="line",
  ylab="centered intensity",
  xlab="",main="Reconstructed",
  ylim=range(ovarian[c(1,2,200,201),]))



grid.arrange(
  pOv1 +
    annotate("point", x = reduced[i,1], y = reduced[i,2], colour = "red",cex=3),
  pOv2,
  pOv3,
  pOv4,
  pOv5,
  layout_matrix = rbind(c(1,2,3),c(4,5,NA)))
```



Acknowledgement

- Olivier Thas for sharing his materials of Analysis of High Dimensional Data 2019-2020, which I used as the starting point for this chapter.

Session info

Session info

```
## [1] "2020-12-10 18:34:15 UTC"

## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.0.3 (2020-10-10)
##  os       macOS Catalina 10.15.7
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       UTC
##  date     2020-12-10
##
## - Packages -------------------------------------------------------------------
##  package               * version    date       lib
##  AIMS                  * 1.22.0     2020-10-27 [1]
##  amap                    0.8-18     2019-12-12 [1]
##  AnnotationDbi           1.52.0     2020-10-27 [1]
##  AnnotationHub         * 2.22.0     2020-10-27 [1]
##  askpass                 1.1        2019-01-13 [1]
##  assertthat              0.2.1      2019-03-21 [1]
##  backports               1.2.1      2020-12-09 [1]
##  beachmat                2.6.2      2020-11-24 [1]
##  beeswarm                0.2.3      2016-04-25 [1]
##  Biobase               * 2.50.0     2020-10-27 [1]
##  BiocFileCache         * 1.14.0     2020-10-27 [1]
##  BiocGenerics          * 0.36.0     2020-10-27 [1]
##  BiocManager             1.30.10    2019-11-16 [1]
##  BiocNeighbors           1.8.2      2020-12-07 [1]
##  BiocParallel            1.24.1     2020-11-06 [1]
##  BiocSingular            1.6.0      2020-10-27 [1]
##  BiocVersion             3.12.0     2020-05-14 [1]
##  biomaRt               * 2.46.0     2020-10-27 [1]
##  bit                     4.0.4      2020-08-04 [1]
##  bit64                   4.0.5      2020-08-30 [1]
##  bitops                  1.0-6      2013-08-17 [1]
##  blob                    1.2.1      2020-01-20 [1]
##  bmp                     0.3        2017-09-11 [1]
##  boot                  * 1.3-25     2020-04-26 [1]
##  bootstrap               2019.6     2019-06-17 [1]
##  breastCancerMAINZ     * 1.28.0     2020-10-29 [1]
##  broom                   0.7.2      2020-10-20 [1]
```

```
##  callr                   3.5.1      2020-10-13 [1]
##  CCA                   * 1.2        2012-10-29 [1]
##  cellranger              1.1.0      2016-07-27 [1]
##  class                   7.3-17     2020-04-26 [2]
##  cli                     2.2.0      2020-11-20 [1]
##  cluster               * 2.1.0      2019-06-19 [1]
##  codetools               0.2-16     2018-12-24 [2]
##  colorspace              2.0-0      2020-11-11 [1]
##  crayon                  1.3.4      2017-09-16 [1]
##  curl                    4.3        2019-12-02 [1]
##  DAAG                  * 1.24       2020-03-10 [1]
##  DBI                     1.1.0      2019-12-15 [1]
##  dbplyr                * 2.0.0      2020-11-03 [1]
##  DelayedArray            0.16.0     2020-10-27 [1]
##  DelayedMatrixStats      1.12.1     2020-11-24 [1]
##  desc                    1.2.0      2018-05-01 [1]
##  devtools                2.3.2      2020-09-18 [1]
##  digest                  0.6.27     2020-10-24 [1]
##  dotCall64             * 1.0-0      2018-07-30 [1]
##  downloader            * 0.4        2015-07-09 [1]
##  dplyr                 * 1.0.2      2020-08-18 [1]
##  e1071                 * 1.7-4      2020-10-14 [1]
##  elasticnet              1.3        2020-05-15 [1]
##  ellipsis                0.3.1      2020-05-15 [1]
##  emo                     0.0.0.9000 2020-12-10 [1]
##  evaluate                0.14       2019-05-28 [1]
##  ExperimentHub         * 1.16.0     2020-10-27 [1]
##  fansi                   0.4.1      2020-01-08 [1]
##  farver                  2.0.3      2020-01-16 [1]
##  fastmap                 1.0.1      2019-10-08 [1]
##  fda                   * 5.1.7      2020-11-28 [1]
##  fds                   * 1.8        2018-10-31 [1]
##  fields                * 11.6       2020-10-09 [1]
##  forcats               * 0.5.0      2020-03-01 [1]
##  foreach                 1.5.1      2020-10-15 [1]
##  fs                      1.5.0      2020-07-31 [1]
##  genefu                * 2.22.0     2020-10-27 [1]
##  generics                0.1.0      2020-10-31 [1]
##  GenomeInfoDb          * 1.26.2     2020-12-08 [1]
##  GenomeInfoDbData        1.2.4      2020-12-10 [1]
##  GenomicRanges         * 1.42.0     2020-10-27 [1]
##  gganimate             * 1.0.7      2020-10-15 [1]
##  ggbeeswarm              0.6.0      2017-08-07 [1]
##  ggbiplot              * 0.55       2020-12-10 [1]
##  ggforce               * 0.3.2      2020-06-23 [1]
##  ggmap                 * 3.0.0      2019-02-05 [1]
##  ggplot2               * 3.3.2      2020-06-19 [1]
##  GIGrvg                  0.5        2017-06-10 [1]
##  git2r                   0.27.1     2020-05-03 [1]
##  glmnet                * 4.0-2      2020-06-16 [1]
##  glue                    1.4.2      2020-08-27 [1]
##  gridExtra             * 2.3        2017-09-09 [1]
##  gtable                  0.3.0      2019-03-25 [1]
##  haven                   2.3.1      2020-06-01 [1]
```

```
##  hdrcde                    3.3        2018-12-21 [1]
##  highr                     0.8        2019-03-20 [1]
##  hms                       0.5.3      2020-01-08 [1]
##  htmltools                 0.5.0      2020-06-16 [1]
##  httpuv                    1.5.4      2020-06-06 [1]
##  httr                      1.4.2      2020-07-20 [1]
##  HyperbolicDist            0.6-2      2009-09-23 [1]
##  iC10                    * 1.5        2019-02-08 [1]
##  iC10TrainingData        * 1.3.1      2018-08-24 [1]
##  igraph                    1.2.6      2020-10-06 [1]
##  imager                  * 0.42.3     2020-05-11 [1]
##  impute                  * 1.64.0     2020-10-27 [1]
##  interactiveDisplayBase    1.28.0     2020-10-27 [1]
##  IRanges                 * 2.24.0     2020-10-27 [1]
##  irlba                     2.3.3      2019-02-05 [1]
##  iterators                 1.0.13     2020-10-15 [1]
##  jpeg                      0.1-8.1    2019-10-24 [1]
##  jsonlite                  1.7.2      2020-12-09 [1]
##  KernSmooth                2.23-17    2020-04-26 [2]
##  knitr                     1.30       2020-09-22 [1]
##  ks                        1.11.7     2020-02-11 [1]
##  labeling                  0.4.2      2020-10-20 [1]
##  lars                      1.2        2013-04-24 [1]
##  later                     1.1.0.1    2020-06-05 [1]
##  latex2exp               * 0.4.0      2015-11-30 [1]
##  lattice                 * 0.20-41    2020-04-02 [2]
##  latticeExtra              0.6-29     2019-12-19 [1]
##  lava                      1.6.8.1    2020-11-04 [1]
##  lifecycle                 0.2.0      2020-03-06 [1]
##  limma                   * 3.46.0     2020-10-27 [1]
##  locfdr                  * 1.1-8      2015-07-15 [1]
##  lubridate                 1.7.9.2    2020-11-13 [1]
##  magick                    2.5.2      2020-11-10 [1]
##  magrittr                * 2.0.1      2020-11-17 [1]
##  maps                      3.3.0      2018-04-03 [1]
##  MASS                    * 7.3-53     2020-09-09 [2]
##  Matrix                  * 1.2-18     2019-11-27 [2]
##  MatrixGenerics          * 1.2.0      2020-10-27 [1]
##  matrixStats             * 0.57.0     2020-09-25 [1]
##  mclust                  * 5.4.7      2020-11-20 [1]
##  mda                       0.5-2      2020-06-29 [1]
##  memoise                   1.1.0      2017-04-21 [1]
##  mgcv                    * 1.8-33     2020-08-27 [1]
##  mime                      0.9        2020-02-04 [1]
##  misc3d                    0.9-0      2020-09-06 [1]
##  modelr                    0.1.8      2020-05-19 [1]
##  munsell                   0.5.0      2018-06-12 [1]
##  muscData                * 1.4.0      2020-10-29 [1]
##  mvtnorm                   1.1-1      2020-06-09 [1]
##  nlme                    * 3.1-149    2020-08-23 [2]
##  NormalBetaPrime         * 2.2        2019-01-19 [1]
##  openssl                   1.4.3      2020-09-18 [1]
##  pamr                    * 1.56.1     2019-04-22 [1]
##  pcaPP                   * 1.9-73     2018-01-14 [1]
```

```
##  pillar                  1.4.7      2020-11-20 [1]
##  pixmap                * 0.4-11     2011-07-19 [1]
##  pkgbuild                1.1.0      2020-07-13 [1]
##  pkgconfig               2.0.3      2019-09-22 [1]
##  pkgload                 1.1.0      2020-05-29 [1]
##  plot3D                * 1.3        2019-12-18 [1]
##  plotROC               * 2.2.1      2018-06-23 [1]
##  pls                   * 2.7-3      2020-08-07 [1]
##  plyr                  * 1.8.6      2020-03-03 [1]
##  PMA                   * 1.2.1      2020-02-03 [1]
##  png                     0.1-7      2013-12-03 [1]
##  polyclip                1.10-0     2019-03-14 [1]
##  pracma                  2.2.9      2019-12-15 [1]
##  prettyunits             1.1.1      2020-01-24 [1]
##  processx                3.4.5      2020-11-30 [1]
##  prodlim               * 2019.11.13 2019-11-17 [1]
##  progress                1.2.2      2019-05-16 [1]
##  promises                1.1.1      2020-06-09 [1]
##  ps                      1.5.0      2020-12-05 [1]
##  pscl                    1.5.5      2020-03-07 [1]
##  purrr                 * 0.3.4      2020-04-17 [1]
##  R6                      2.5.0      2020-10-28 [1]
##  rainbow               * 3.6        2019-01-29 [1]
##  rappdirs                0.3.1      2016-03-28 [1]
##  RColorBrewer            1.1-2      2014-12-07 [1]
##  Rcpp                    1.0.5      2020-07-06 [1]
##  RCurl                 * 1.98-1.2   2020-04-18 [1]
##  readbitmap              0.1.5      2018-06-27 [1]
##  readr                 * 1.4.0      2020-10-05 [1]
##  readxl                  1.3.1      2019-03-13 [1]
##  remotes                 2.2.0      2020-07-21 [1]
##  reprex                  0.3.0      2019-05-16 [1]
##  RgoogleMaps             1.4.5.3    2020-02-12 [1]
##  rjson                   0.2.20     2018-06-08 [1]
##  rlang                   0.4.9      2020-11-26 [1]
##  rmarkdown               2.5        2020-10-21 [1]
##  rmeta                   3.0        2018-03-20 [1]
##  rprojroot               2.0.2      2020-11-15 [1]
##  RSQLite                 2.2.1      2020-09-30 [1]
##  rstudioapi              0.13       2020-11-12 [1]
##  rsvd                    1.0.3      2020-02-17 [1]
##  rvest                   0.3.6      2020-07-25 [1]
##  S4Vectors             * 0.28.0     2020-10-27 [1]
##  scales                * 1.1.1      2020-05-11 [1]
##  scater                * 1.18.3     2020-11-08 [1]
##  scuttle                 1.0.3      2020-11-23 [1]
##  SemiPar               * 1.0-4.2    2018-04-16 [1]
##  sessioninfo             1.1.1      2018-11-05 [1]
##  shape                   1.4.5      2020-09-13 [1]
##  shiny                   1.5.0      2020-06-23 [1]
##  SingleCellExperiment  * 1.12.0     2020-10-27 [1]
##  sp                      1.4-4      2020-10-07 [1]
##  spam                  * 2.5-1      2019-12-12 [1]
##  sparseLDA             * 0.1-9      2016-09-22 [1]
```

```
##  sparseMatrixStats        1.2.0      2020-10-27 [1]
##  stringi                  1.5.3      2020-09-09 [1]
##  stringr                * 1.4.0      2019-02-10 [1]
##  SummarizedExperiment   * 1.20.0     2020-10-27 [1]
##  SuppDists                1.1-9.5    2020-01-18 [1]
##  survcomp               * 1.40.0     2020-10-27 [1]
##  survival               * 3.2-7      2020-09-28 [2]
##  survivalROC              1.0.3      2013-01-13 [1]
##  testthat                 3.0.0      2020-10-31 [1]
##  tibble                 * 3.0.4      2020-10-12 [1]
##  tidyr                  * 1.1.2      2020-08-27 [1]
##  tidyselect               1.1.0      2020-05-11 [1]
##  tidyverse              * 1.3.0      2019-11-21 [1]
##  tiff                     0.1-6      2020-11-17 [1]
##  tinytex                  0.27       2020-11-01 [1]
##  truncnorm                1.0-8      2018-02-27 [1]
##  tweenr                   1.0.1      2018-12-14 [1]
##  usethis                  2.0.0      2020-12-10 [1]
##  utf8                     1.1.4      2018-05-24 [1]
##  vctrs                    0.3.5      2020-11-17 [1]
##  vipor                    0.4.5      2017-03-22 [1]
##  viridis                  0.5.1      2018-03-29 [1]
##  viridisLite              0.3.0      2018-02-01 [1]
##  withr                    2.3.0      2020-09-22 [1]
##  xfun                     0.19       2020-10-30 [1]
##  XML                      3.99-0.5   2020-07-23 [1]
##  xml2                     1.3.2      2020-04-23 [1]
##  xtable                   1.8-4      2019-04-21 [1]
##  XVector                  0.30.0     2020-10-28 [1]
##  yaml                     2.2.1      2020-02-01 [1]
##  zlibbioc                 1.36.0     2020-10-28 [1]
##  source
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
```

```
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.1)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Github (hadley/emo@3f03b11)
##  CRAN (R 4.0.1)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Github (vqv/ggbiplot@7325e88)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
```

```
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.3)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
```

```
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
```

```
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.3)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.1)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  CRAN (R 4.0.2)
##  Bioconductor
##  CRAN (R 4.0.2)
##  Bioconductor
##
## [1] /Users/runner/work/_temp/Library
## [2] /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```

Home