Assignment 8

Sebastian Veuskens

Exercise 1

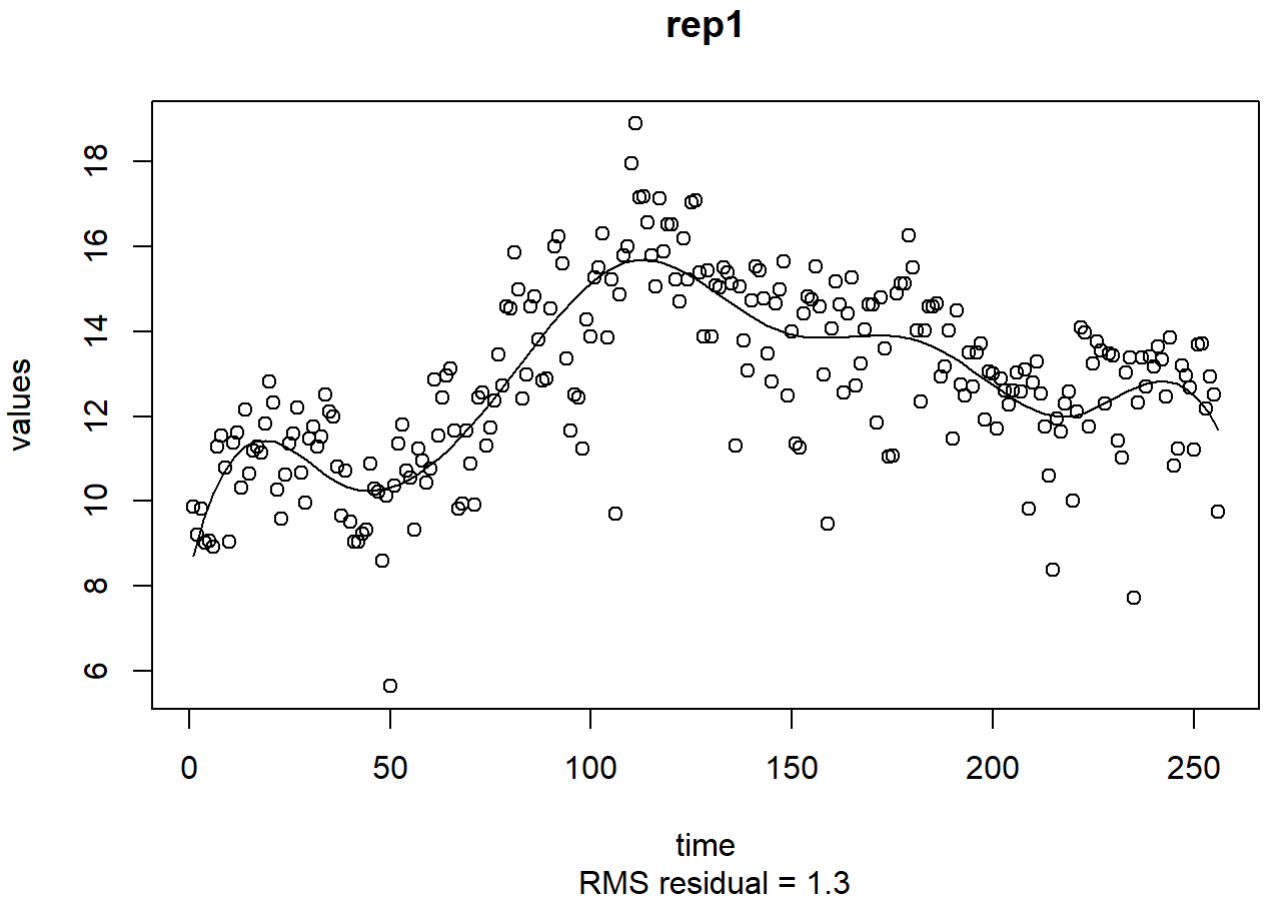
Load data

```
library(fda)
phonemes1000 <- read.table("data/phonemes1000.dat", header=TRUE)
phonemes256 <- as.matrix(phonemes1000[,1:256])
```

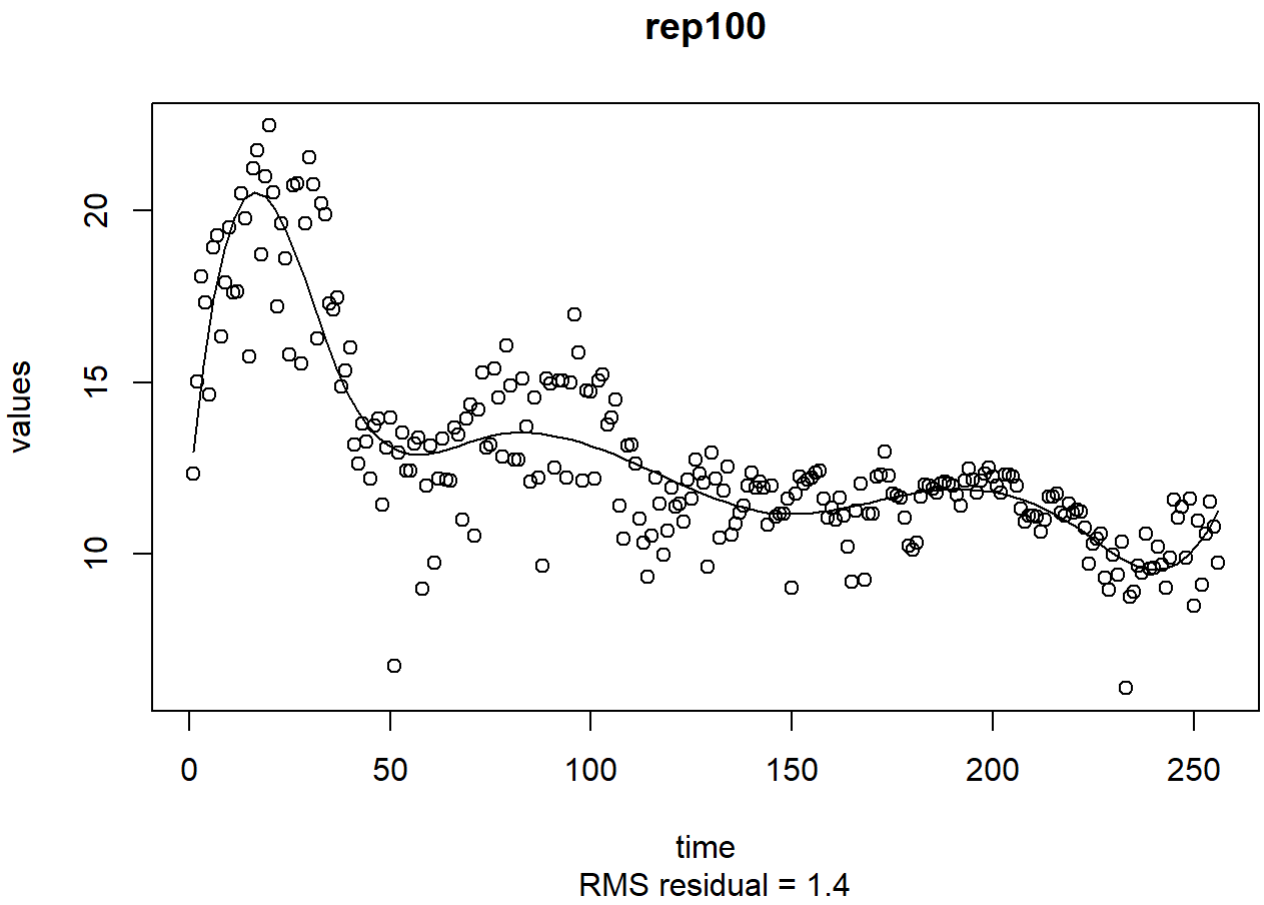
Number of b-splines

```
# Bspline Basis of 10
bbasis10 <- create.bspline.basis(c(1, 256), nbasis=10)
fd_10 <- Data2fd(1:256, y=t(phonemes256), basisobj=bbasis10)

plotfit.fd(t(phonemes256), 1:256, fd_10, index = 1, cex.pch=1)
```

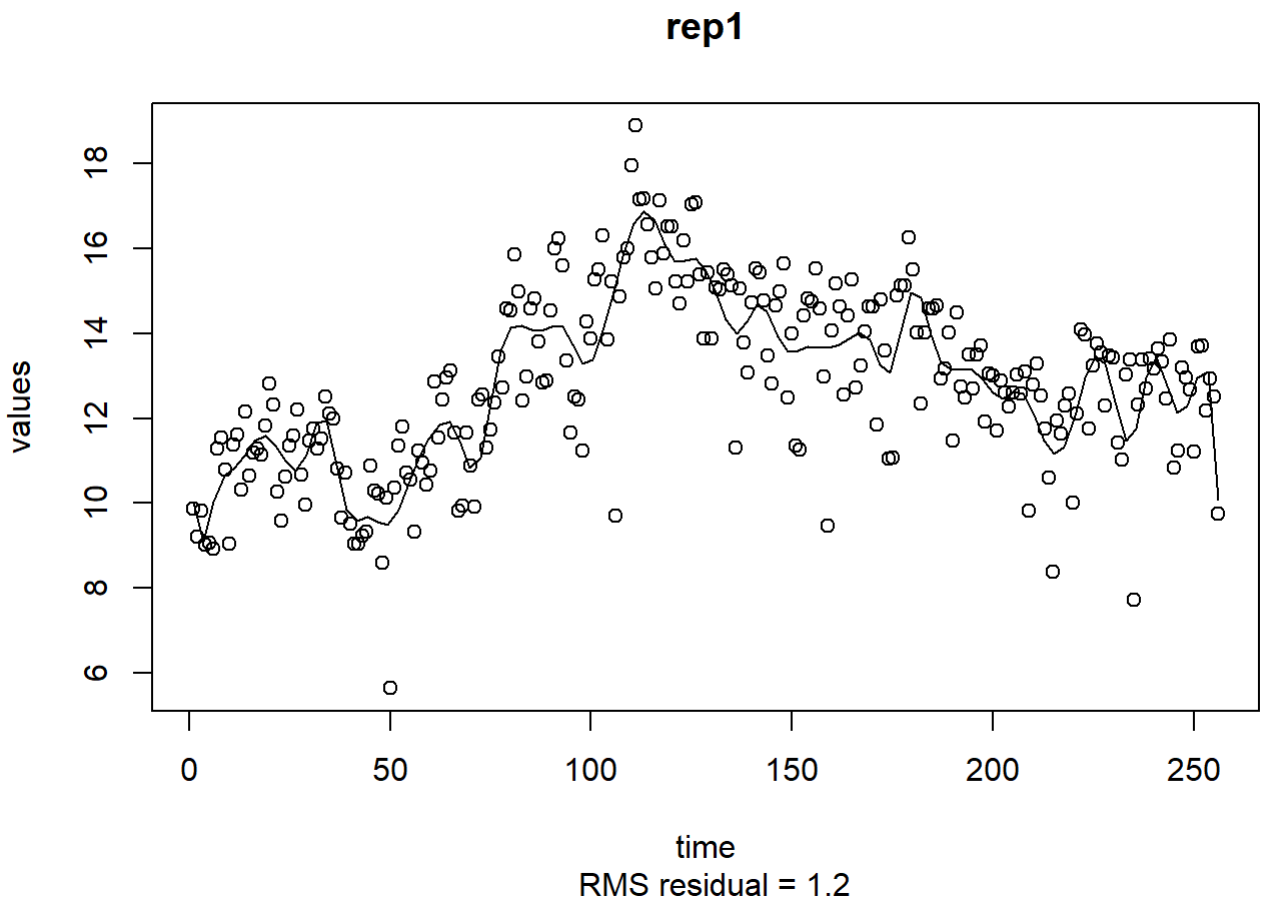



```
plotfit.fd(t(phonemes256), 1:256, fd_10, index = 100, cex.pch=1)
```

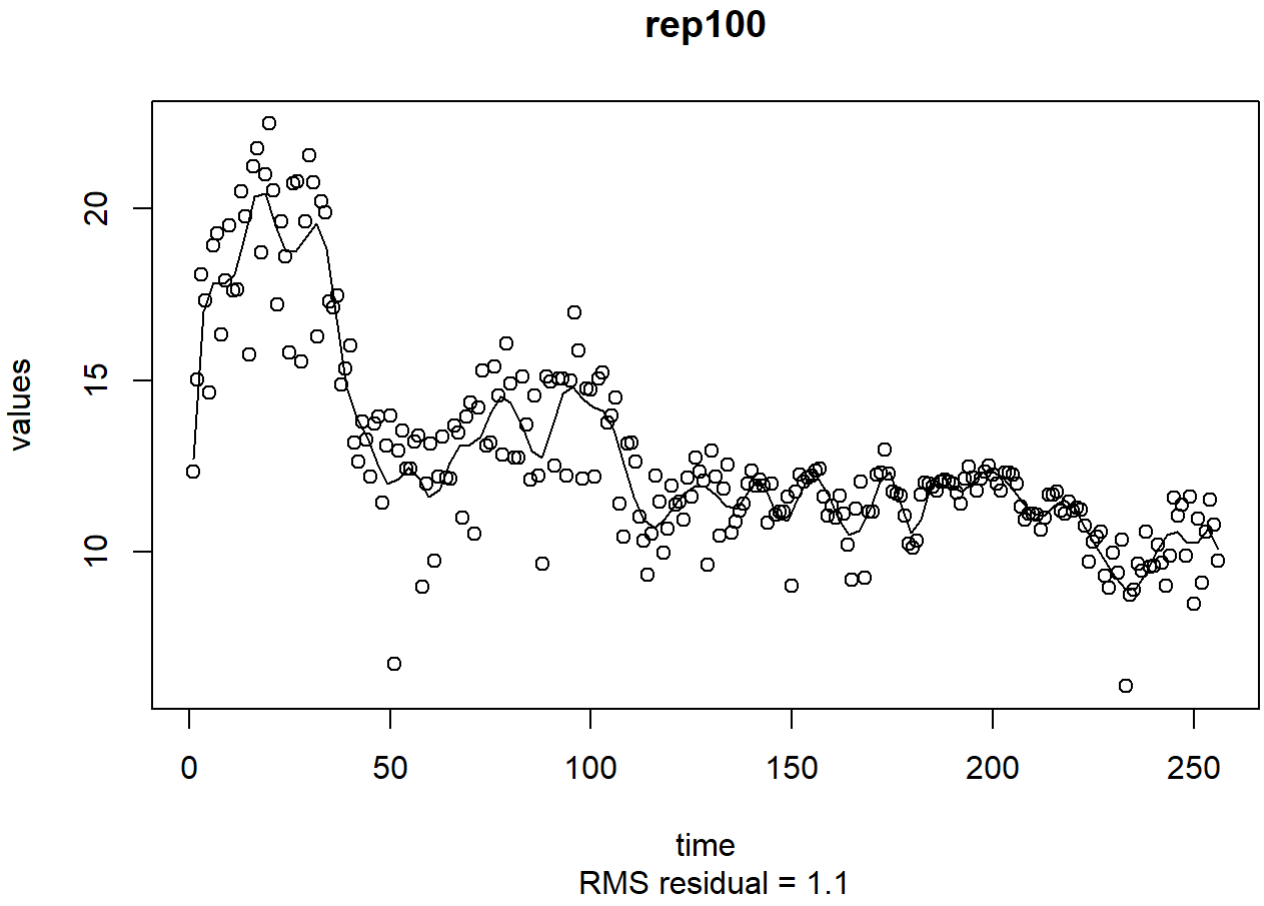



```
# Bspline Basis of 50
bbasis50 <- create.bspline.basis(c(1, 256), nbasis=50)
fd_50 <- Data2fd(1:256, y=t(phonemes256), basisobj=bbasis50)

plotfit.fd(t(phonemes256), 1:256, fd_50, index = 1, cex.pch=1)
```




```
plotfit.fd(t(phonemes256), 1:256, fd_50, index = 100, cex.pch=1)
```



```
# Bspline Basis of 200
bbasis200 <- create.bspline.basis(c(1, 256), nbasis=200)
fd_200 <- Data2fd(1:256, y=t(phonemes256), basisobj=bbasis200)
```

50 seems like suitable number of b-splines, since it reduces the dimensionality of the data, while still maintaining the dynamics of the observations up to a fairly got degree of detail.

Principal components

```
# Try different number of components
phoneme_pca3 <- pca.fd(fd_50, nharm=3)
phoneme_pca5 <- pca.fd(fd_50, nharm=5)
phoneme_pca10 <- pca.fd(fd_50, nharm=10)

phoneme_pca10$varprop
```

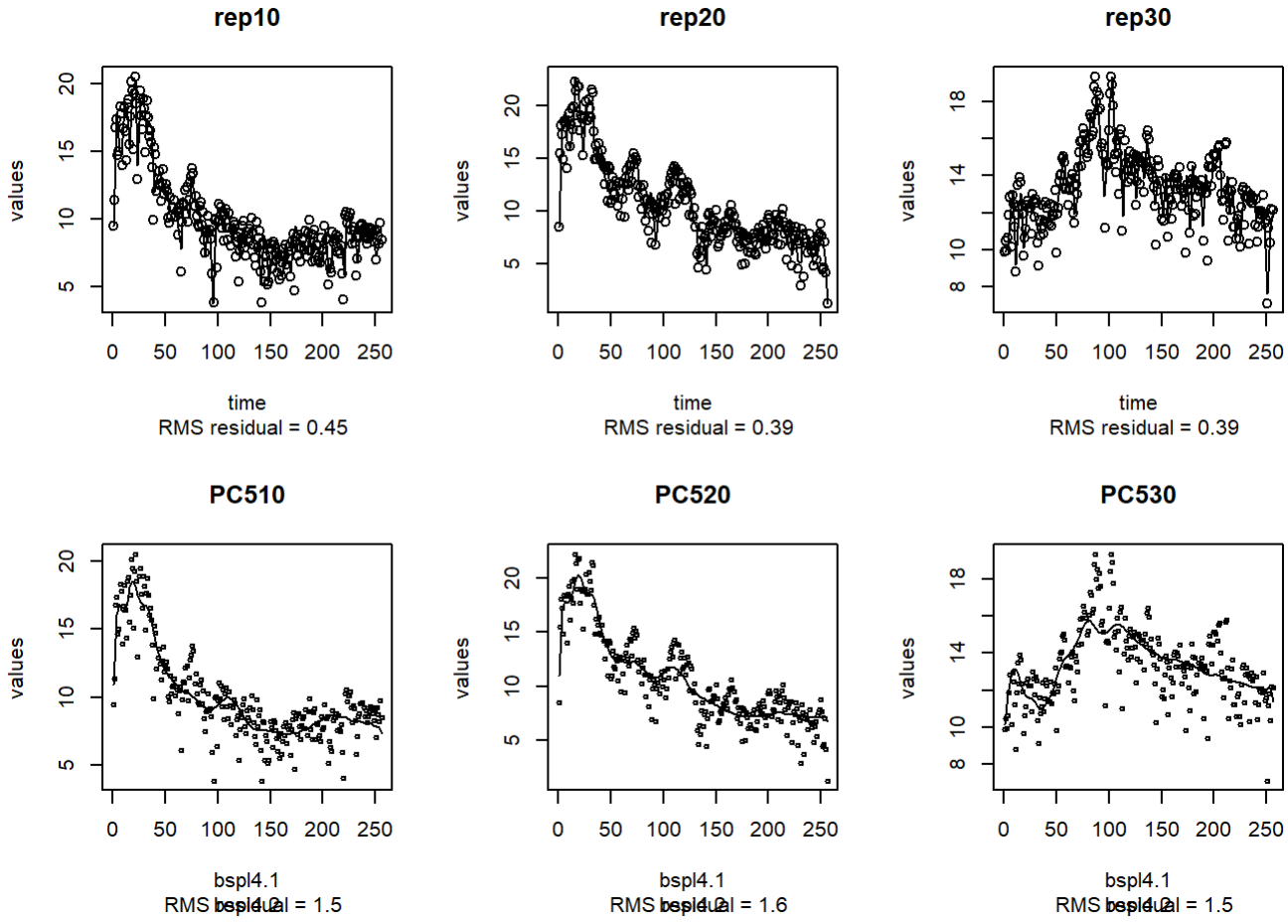
```
##  [1] 0.595197976 0.210219774 0.060664326 0.022037806 0.014810603 0.012467164
##  [7] 0.009708805 0.008689495 0.006746595 0.005246932
```

```
cumsum(phoneme_pca10$varprop)
```

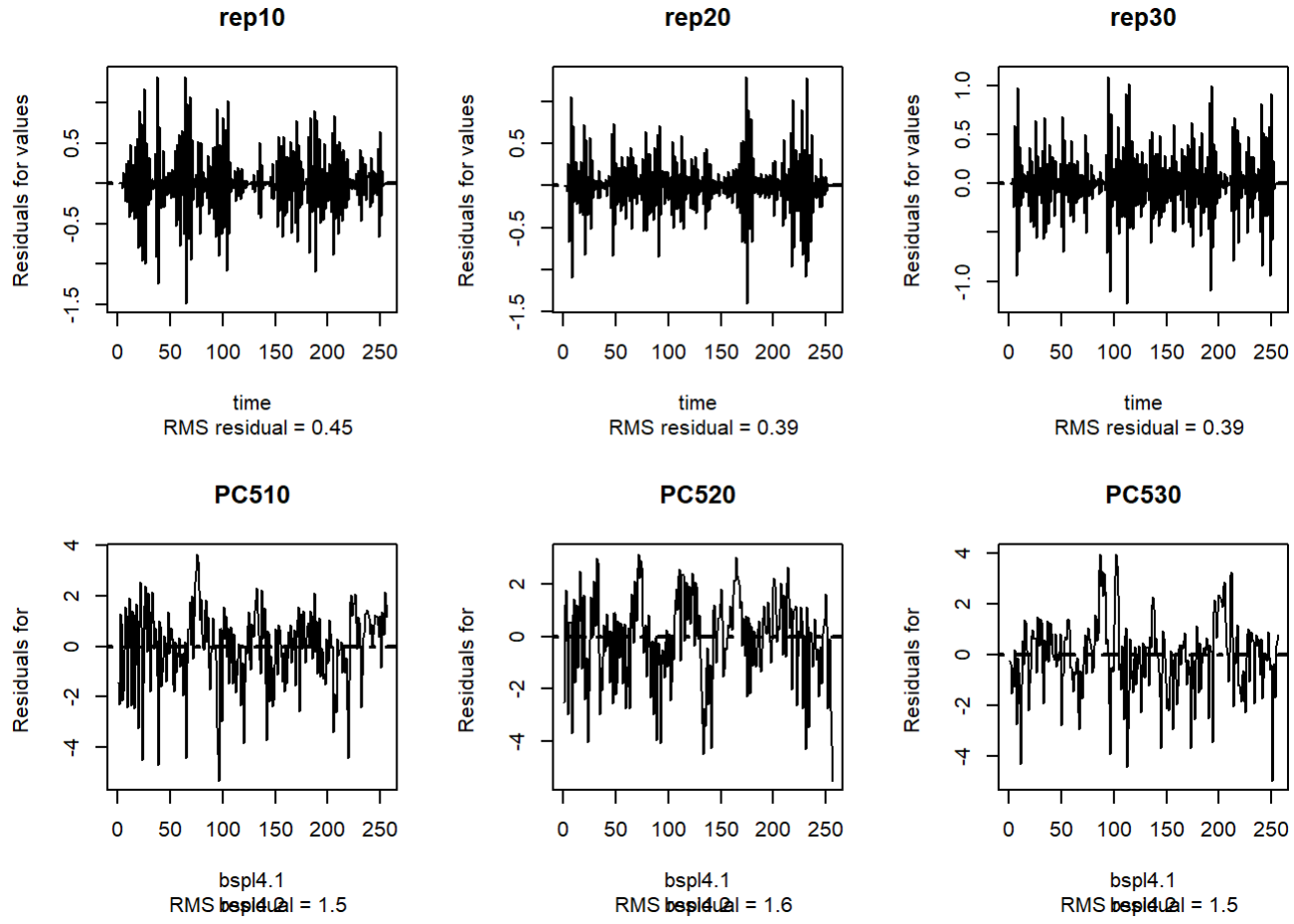
```
##  [1] 0.5951980 0.8054178 0.8660821 0.8881199 0.9029305 0.9153976 0.9251065
##  [8] 0.9337959 0.9405425 0.9457895
```

```
# 5 components model already 90% of the variance
# More components only model marginally more variance

phoneme_approx <- phoneme_pca5$harmonics
mean_phoneme <- mean.fd(fd_50)
pca_coefi <- phoneme_pca5$harmonics$coefs %*% phoneme_pca5$scores[1,]+mean_phoneme$coefs
phoneme_approx$coefs <- pca_coefi
for (i in 2:nrow(phoneme_pca5$scores)) {
    pca_coefi <- phoneme_pca5$harmonics$coefs %*% phoneme_pca5$scores[i,]+mean_phoneme$coefs
    phoneme_approx$coefs <- cbind(phoneme_approx$coefs, pca_coefi)
}
par(mfrow=c(2,3))
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 10, cex.pch=1)
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 20, cex.pch=1)
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 30, cex.pch=1)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=10, cex.pch=0.5)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=20, cex.pch=0.5)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=30, cex.pch=0.5)
```




```
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 10, cex.pch=1, residual=T)
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 20, cex.pch=1, residual=T)
plotfit.fd(t(phonemes256), 1:256, fd_200, index = 30, cex.pch=1, residual=T)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=10, cex.pch=0.5, residual=T)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=20, cex.pch=0.5, residual=T)
plotfit.fd(t(phonemes256), 1:256, phoneme_approx, index=30, cex.pch=0.5, residual=T)
```




```
par(mfrow=c(1,1))
```

funFEM clustering

```
library(funFEM)

set.seed(12345)
phen_clus1 <- funFEM(fd_50, K=2:10)
str(phen_clus1)
```

```
## List of 15
##  $ model        : chr "AkjBk"
##  $ K            : int 8
##  $ cls          : int [1:1000] 4 2 7 7 6 1 6 2 6 6 ...
##  $ P            : num [1:1000, 1:8] 5.46e-53 9.39e-07 1.99e-86 2.86e-88 3.28e-17 ...
##  $ prms         :List of 7
##   ..$ K    : int 8
##   ..$ p    : int 50
##   ..$ mean : num [1:8, 1:7] -26.2 -33.4 -38.8 -46.7 -53.4 ...
##   ..$ my   : num [1:8, 1:50] 9.32 10.44 11.66 10.22 10.68 ...
##   ..$ prop : num [1:8] 0.0999 0.1694 0.0713 0.089 0.1098 ...
##   ..$ D    : num [1:8, 1:50, 1:50] 5.98 8.92 8.6 9.8 6.45 ...
##   ..$ model: chr "AkjBk"
##  $ U            : num [1:50, 1:7] -0.3196 -0.0409 0.2537 0.4234 -0.1988 ...
##  $ aic          : num -110724
##  $ bic          : num -111826
##  $ icl          : num -111793
##  $ loglik       : num [1:40] -113002 -111348 -111139 -110387 -111644 ...
##  $ ll           : num -110275
##  $ nbprm        : num 449
##  $ crit         : chr "bic"
##  $ allCriterions:'data.frame':   9 obs. of  7 variables:
##   ..$ K    : int [1:9] 2 3 4 5 6 7 8 9 10
##   ..$ model: chr [1:9] "AkjBk" "AkjBk" "AkjBk" "AkjBk" ...
##   ..$ bic  : num [1:9] -125121 -116653 -114383 -113096 -112998 ...
##   ..$ aic  : num [1:9] -124983 -116373 -113953 -112509 -112248 ...
##   ..$ icl  : num [1:9] -125120 -116650 -114351 -113080 -112977 ...
##   ..$ nbprm: num [1:9] 56 114 175 239 306 376 449 525 604
##   ..$ ll   : num [1:9] -124927 -116259 -113778 -112270 -111942 ...
##  $ call         : language funFEM(fd = fd_50, K = 2:10)
##  - attr(*, "class")= chr "fem"
```

```
# Procedure to get
femmodels <- c("DkBk", "DkB", "DBk", "DB", "AkjBk", "AkjB", "AkB", "AkBk", "AjBk", "AjB", "AB
k", "AB")
nmodels <- length(femmodels)
femresults <- list() # Saves output for all models
bestk <- numeric(0) # Vector of best number of clusters for each model
bestbic <- numeric(0) # Vector of bic for best clustering for each model
K = 2:10 # Number of clusters to try out
fembic <- matrix(NA,nrow=nmodels,ncol=max(K)) # BIC for all clusterings and models

functional_data_obj <- fd_50
for (i in 1:nmodels){ # This takes a long time!
    print(femmodels[i])
    femresults[[i]] <- funFEM(functional_data_obj,model=femmodels[i],K=K)
    fembic[i,K] <- femresults[[i]]$allCriterions$bic
    bestk[i] <- which(fembic[i,]==max(fembic[i,K],na.rm=TRUE))
    bestbic[i] <- max(fembic[i,K],na.rm=TRUE)
}
```

```
## [1] "DkBk"
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## [1] "DkB"
## [1] "DBk"
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## [1] "DB"
## [1] "AkjBk"
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## [1] "AkjB"
## [1] "AkB"
## [1] "AkBk"
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## [1] "AjBk"
## [1] "AjB"
## [1] "ABk"
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## Error in .fstep(fd, T, lambda) : One cluster is almost empty!
## [1] "AB"
```

```
besti <- which(bestbic==max(bestbic,na.rm=TRUE))
besti
```

```
## [1] 5
```

```
bestk[besti]
```

```
## [1] 7
```

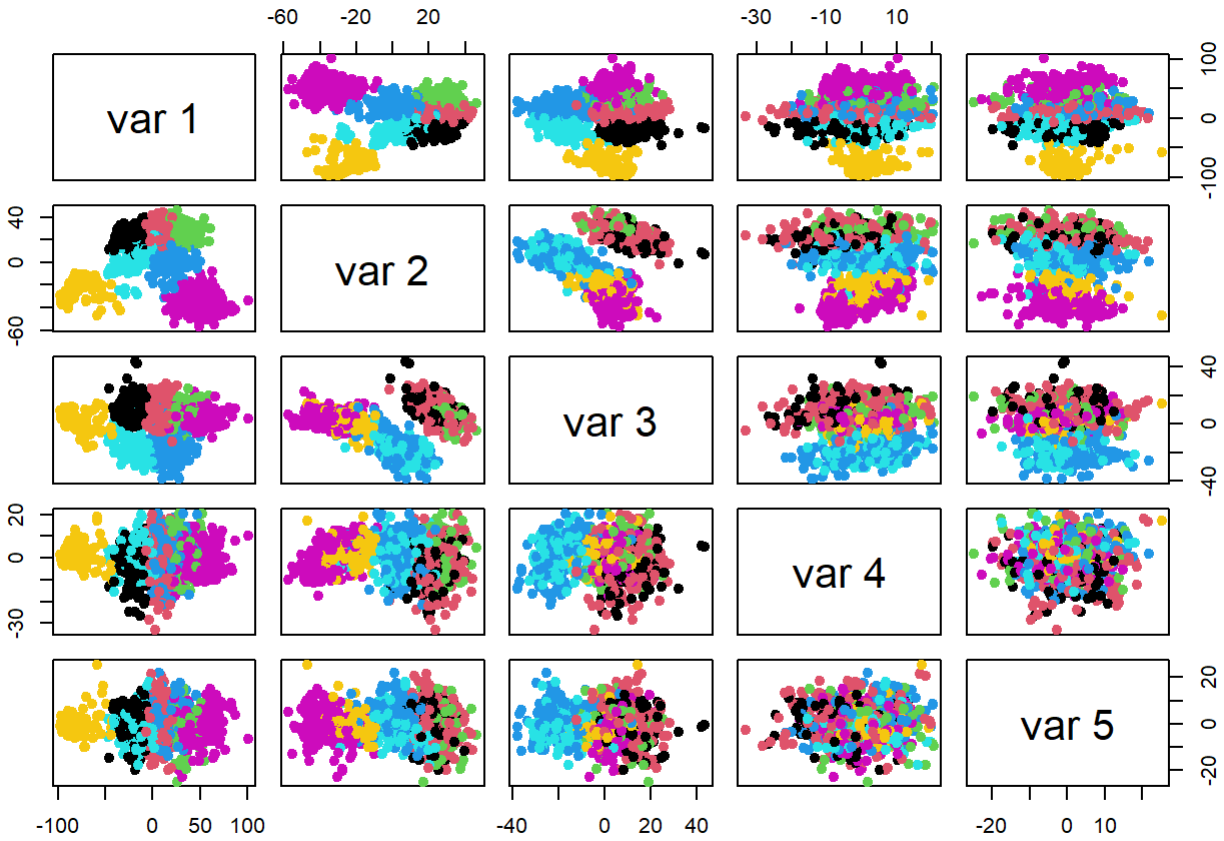
```
best_model <- femresults[[besti]]
str(best_model$prms$my)
```

```
##  num [1:7, 1:50] 10.59 11.41 11.72 10.37 9.45 ...
```

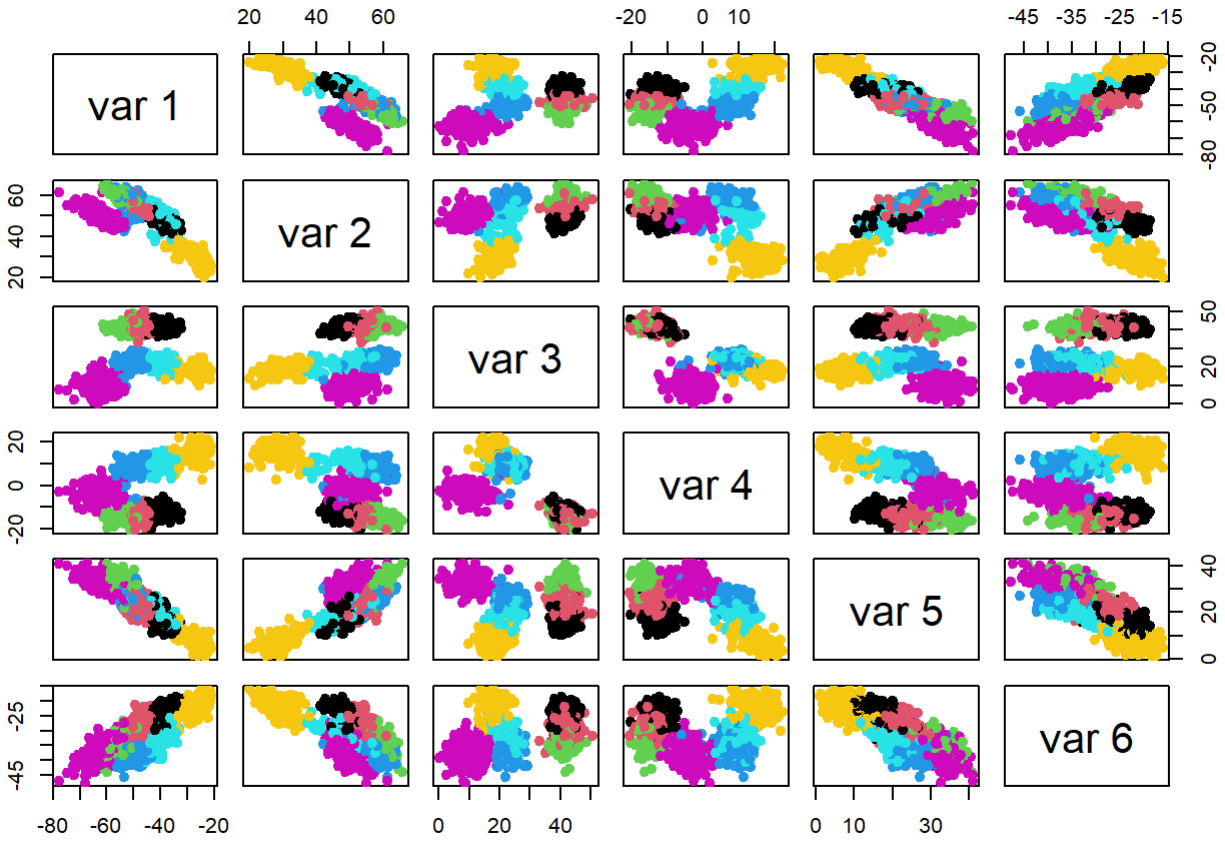
The AkjBk model performs the best, according to the BIC, with 7 clusters.

Visualize funFEM results

```
pairs(phoneme_pca5$scores, col=best_model$cls, pch=19)
```




```
fd_proj <- t(fd_50$coefs) %*% best_model$U
pairs(fd_proj, col=best_model$cls, pch=19) # -> 7 clusters, thus (7 - 1 =) 6 dimensional subs
pace
```



My clustering

```
library(smacof)
library(rgl)
library(prabclus)
library(cluster)
data(tetragonula)
km_clus <- kmeans(phonemes256, centers=7)
adjustedRandIndex(km_clus$cluster, best_model$cls)
```

```
## [1] 0.7143535
```

The ARI is very similar, meaning that the clusters have very similar features. This does not necessarily indicate a correct cluster, but indicates that these two different algorithms discovered the same underlying pattern of the data.

Exercise 2

We know that it should hold:

B(s_min) = B(s_max) = 0

In this case, s_min = 1 and s_max = 4. The B-spline: B(x) = ax**2 + bx + c, for x in (1,4) Thus, the B-spline should fulfill the properties of continuous derivatives for the knots, as mentioned in the slides.

An additional constrain is given by max(B(x)) = 1 max(B(x)) = B(2.5) = 6.25 * a + 2.5 * b + c = 1

These 9 constraints can be solved through linear algebra:

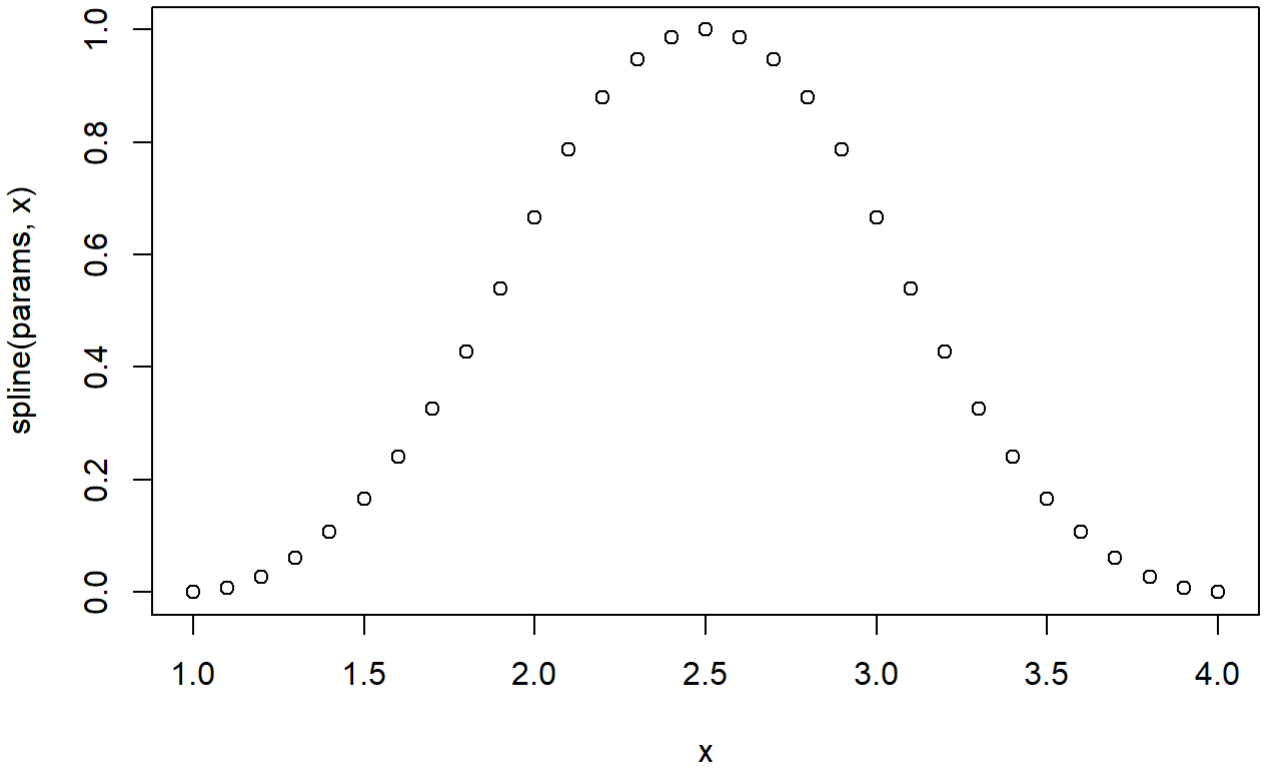
```
equations <- rbind(c(1, 1, 1, 0, 0, 0, 0, 0, 0),
                   c(2, 1, 0, 0, 0, 0, 0, 0, 0),
                   c(4, 2, 1, -4, -2, -1, 0, 0, 0),
                   c(4, 1, 0, -4, -1, 0, 0, 0, 0),
                   c(0, 0, 0, 9, 3, 1, -9, -3, -1),
                   c(0, 0, 0, 6, 1, 0, -6, -1, 0),
                   c(0, 0, 0, 0, 0, 0, 16, 4, 1),
                   c(0, 0, 0, 0, 0, 0, 8, 1, 0),
                   c(0, 0, 0, 6.25, 2.5, 1, 0, 0, 0))
values <- c(0, 0, 0, 0, 0, 0, 0, 0, 1)

params <- solve(equations, values)
x <- seq(1, 4, by=0.1)

spline <- function(params, x) {
    res <- numeric(length(x))
    i <- 1
    for (x_i in x) {
        if (x_i < 2) {
            res[i] <-params[1] * (x_i ** 2) + params[2] * x_i + params[3]
        } else if (x_i < 3) {
            res[i] <- params[4] * (x_i ** 2) + params[5] * x_i + params[6]
        } else {
            res[i] <- params[7] * (x_i ** 2) + params[8] * x_i + params[9]
        }
        i <- i + 1
    }
    return(res)
}
spline(params, x)
```

```
##  [1] -8.881784e-16  6.666667e-03  2.666667e-02  6.000000e-02  1.066667e-01
##  [6]  1.666667e-01  2.400000e-01  3.266667e-01  4.266667e-01  5.400000e-01
## [11]  6.666667e-01  7.866667e-01  8.800000e-01  9.466667e-01  9.866667e-01
## [16]  1.000000e+00  9.866667e-01  9.466667e-01  8.800000e-01  7.866667e-01
## [21]  6.666667e-01  5.400000e-01  4.266667e-01  3.266667e-01  2.400000e-01
## [26]  1.666667e-01  1.066667e-01  6.000000e-02  2.666667e-02  6.666667e-03
## [31]  0.000000e+00
```

```
plot(x, spline(params, x))
```



The spline looks like its configured correctly. The parameters for the polynomials can be found in the params object.

Exercise 3

```
library(fpc)
```

a)

Discriminant coordinates are a method or tool to provide a way to parameterize points on some different coordinate axes, somehow similar to the Principal Component Analysis procedure. They are the coordinates that capture a special point and help to distinguish different components. Asymmetric weighted discriminant coordinates are a type of asymmetrization of DCs. Instead of W, the cavariance matrix of H-class is used. The main difference is the minimum value the denominator can take now: It does not become too small, even when the projected variance is reduced. Additionally, the between-groups matrix takes also into account now the squared differences between the points of different classes.