Mandatory exercise 2
STK2100 - Spring 2020

By Ida Johanne Austad

See appendix for R-code. The code output and figures in the answers are included in the main response below.

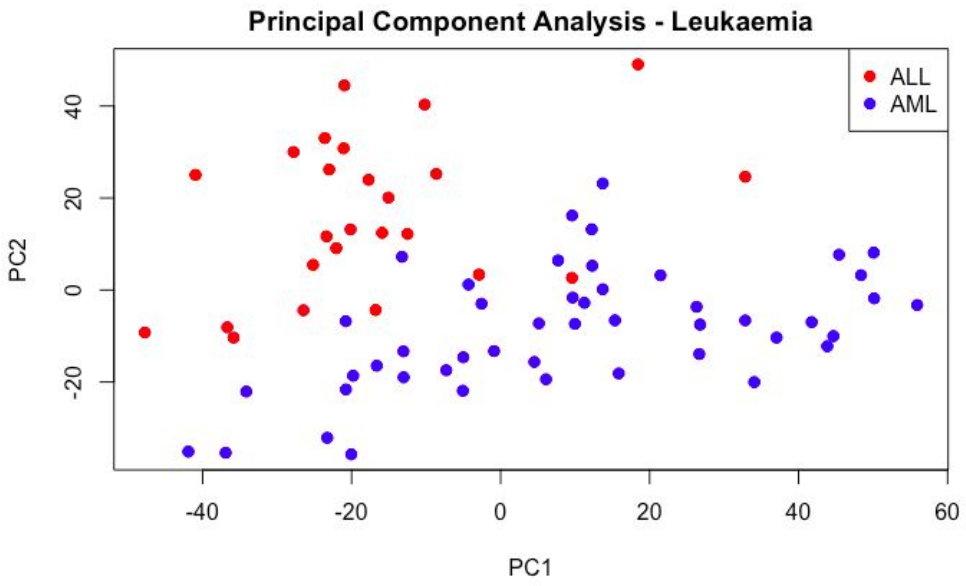
Problem 1

a)

Summary:

```
Importance of first k=2 (out of 72) components:
                            PC1      PC2
Standard deviation     26.15294 18.81732
Proportion of Variance  0.09596  0.04968
Cumulative Proportion   0.09596  0.14563
```

Plot:



b)

Output for cross validation to find penalty parameter lambda:

```
Call:  cv.glmnet(x = t(df), y = response_df[, 2], nfolds = 3, alfa = 1,
standardize = T)


Measure: Mean-Squared Error
    Lambda Measure    SE Nonzero
min  0.367   108.4 27.95      65
```

```
1se  5.455   135.2 43.75       5

Call:  cv.glmnet(x = t(df), y = response_df[, 2], nfolds = 10, alfa = 1,
standardize = T)

Measure: Mean-Squared Error
    Lambda Measure    SE Nonzero
min 0.3195   50.32 8.770      66
1se 0.9313   58.93 8.931      46

Call:  cv.glmnet(x = t(df), y = response_df[, 2], nfolds = 72, alfa = 1,
standardize = T)

Measure: Mean-Squared Error
    Lambda Measure    SE Nonzero
min 0.3195   49.33 7.314      66
1se 0.9313   56.23 8.227      46
```

What is the role of the penalty parameter in the Lasso model?
Lambda is the tuning parameter, and its "size" decides how much one should penalize for the size of the coefficients on the minimal error obtained. With different values of Lambda, different coefficients will be given. As Lambda increases the flexibility of the model decreases - resulting in decreased variance, while increasing bias. Due to the formulation of the penalty term in Lasso regression, as opposed to Ridge regression, some coefficients will be forced to be zero - hence working as a selection method for the best subset of features to include in a model.

Did you expect to find different values?
We see that cross validation with 10 and 72 folds gives the same value of lamba: 0.3195. It was expected that the estimated lambda could be different across the different cv's, as the assignment of train vs test cases is random thus and because with more subsamples, comes more opportunities to find a better (i.e. lower) value of MSE. We see that a lower MSE is obtained with 72 folds. However, the optimal lambda value is the same as for 10 folds.

c)
Selected coefficients and their estimated value using penalty term value from 3-fold cross validation:

```
> extract.coef(lambda_cv_3)
                    Value SE Coefficient
(Intercept) -60.597000178 NA (Intercept)
```

gene1	7.149852521	NA	gene1
gene11	11.802826522	NA	gene11
gene21	5.733395256	NA	gene21
gene39	2.038271215	NA	gene39
gene41	9.441158017	NA	gene41
gene51	10.587473680	NA	gene51
gene71	1.941158299	NA	gene71
gene81	1.707324771	NA	gene81
gene138	-0.332484821	NA	gene138
gene206	-0.037613544	NA	gene206
gene393	0.193016740	NA	gene393
gene652	0.045884423	NA	gene652
gene689	-0.670832020	NA	gene689
gene761	0.106595409	NA	gene761
gene887	0.897440924	NA	gene887
gene909	0.356018910	NA	gene909
gene1090	6.092120495	NA	gene1090
gene1262	-5.048611541	NA	gene1262
gene1304	0.031785329	NA	gene1304
gene1461	-0.001488615	NA	gene1461
gene1484	1.667222283	NA	gene1484
gene2013	-5.386739841	NA	gene2013
gene2154	0.881883553	NA	gene2154
gene2168	-0.489509617	NA	gene2168
gene2323	-0.249590631	NA	gene2323
gene2351	0.653585510	NA	gene2351
gene2461	0.802132439	NA	gene2461
gene2552	0.105035986	NA	gene2552
gene2966	-0.004399620	NA	gene2966
gene3069	0.228973691	NA	gene3069
gene3240	0.188096909	NA	gene3240
gene3429	1.027178870	NA	gene3429
gene3653	-0.042097565	NA	gene3653
gene3696	1.076590522	NA	gene3696
gene4016	-0.142351988	NA	gene4016

```
gene4058      0.348169406 NA    gene4058
gene4150     -3.846292665 NA    gene4150
gene4432      0.662642601 NA    gene4432
gene4471     -0.062981172 NA    gene4471
gene4606     -0.670038328 NA    gene4606
gene4629     -0.291214079 NA    gene4629
gene4635     -0.160096354 NA    gene4635
gene4671      0.854280853 NA    gene4671
gene4983     -0.124641555 NA    gene4983
gene5038      0.597648669 NA    gene5038
gene5089      1.377048903 NA    gene5089
gene5183      0.882847824 NA    gene5183
gene5197     -0.636569129 NA    gene5197
gene5223     -0.144849823 NA    gene5223
gene5388      0.002787436 NA    gene5388
gene5470      0.331870274 NA    gene5470
gene5477      1.924222757 NA    gene5477
gene5508     -0.010896942 NA    gene5508
gene5603      1.843502456 NA    gene5603
gene5781      5.097809417 NA    gene5781
gene5788      1.891885122 NA    gene5788
gene6165     -1.248215492 NA    gene6165
gene6181      0.597808014 NA    gene6181
gene6213      1.000133749 NA    gene6213
gene6490     -0.143609732 NA    gene6490
gene6518     -1.825345939 NA    gene6518
gene6697     -0.238824588 NA    gene6697
gene6771     -0.037801388 NA    gene6771
gene6790      1.680696375 NA    gene6790
gene6889     -1.114656462 NA    gene6889
```

Selected coefficients and their estimated value using penalty term value from 10-fold cross validation:

```
> extract.coef(lambda_cv_10)
                    Value SE Coefficient
```

(Intercept)	-6.051814e+01	NA	(Intercept)
gene1	7.276829e+00	NA	gene1
gene11	1.181852e+01	NA	gene11
gene21	5.866931e+00	NA	gene21
gene39	1.995023e+00	NA	gene39
gene41	9.479081e+00	NA	gene41
gene51	1.063084e+01	NA	gene51
gene71	1.937643e+00	NA	gene71
gene81	1.850043e+00	NA	gene81
gene138	-3.157486e-01	NA	gene138
gene206	-1.785301e-02	NA	gene206
gene393	4.723856e-01	NA	gene393
gene689	-6.713344e-01	NA	gene689
gene761	1.458883e-01	NA	gene761
gene887	7.917126e-01	NA	gene887
gene909	4.067878e-01	NA	gene909
gene1090	6.028821e+00	NA	gene1090
gene1262	-5.059100e+00	NA	gene1262
gene1304	5.254057e-02	NA	gene1304
gene1461	-2.592561e-02	NA	gene1461
gene1484	1.572907e+00	NA	gene1484
gene2013	-5.584451e+00	NA	gene2013
gene2154	9.215664e-01	NA	gene2154
gene2168	-4.269470e-01	NA	gene2168
gene2323	-1.887987e-01	NA	gene2323
gene2351	6.951206e-01	NA	gene2351
gene2461	8.373157e-01	NA	gene2461
gene2552	1.021810e-03	NA	gene2552
gene2966	-2.005345e-02	NA	gene2966
gene3069	2.934732e-01	NA	gene3069
gene3240	2.159219e-01	NA	gene3240
gene3429	9.816510e-01	NA	gene3429
gene3653	-4.158387e-02	NA	gene3653
gene3696	1.142370e+00	NA	gene3696
gene4016	-2.365341e-01	NA	gene4016

```
gene4058     3.549253e-01 NA    gene4058
gene4071    -1.167328e-02 NA    gene4071
gene4150    -3.859594e+00 NA    gene4150
gene4432     6.911038e-01 NA    gene4432
gene4453    -6.809118e-04 NA    gene4453
gene4471    -1.386743e-01 NA    gene4471
gene4606    -7.512143e-01 NA    gene4606
gene4629    -2.576374e-01 NA    gene4629
gene4635    -1.765622e-01 NA    gene4635
gene4671     1.017755e+00 NA    gene4671
gene4983    -1.774699e-01 NA    gene4983
gene5038     5.308109e-01 NA    gene5038
gene5089     1.214964e+00 NA    gene5089
gene5183     8.728299e-01 NA    gene5183
gene5197    -6.451333e-01 NA    gene5197
gene5223    -9.678267e-02 NA    gene5223
gene5388     4.986500e-03 NA    gene5388
gene5470     2.538081e-01 NA    gene5470
gene5477     1.793532e+00 NA    gene5477
gene5508    -5.104041e-02 NA    gene5508
gene5603     1.959205e+00 NA    gene5603
gene5781     5.057209e+00 NA    gene5781
gene5788     1.836525e+00 NA    gene5788
gene6165    -1.263856e+00 NA    gene6165
gene6181     7.243338e-01 NA    gene6181
gene6213     1.052177e+00 NA    gene6213
gene6490    -1.758501e-01 NA    gene6490
gene6518    -1.793929e+00 NA    gene6518
gene6697    -3.037262e-01 NA    gene6697
gene6771    -6.526271e-02 NA    gene6771
gene6790     1.711839e+00 NA    gene6790
gene6889    -1.150001e+00 NA    gene6889
```

Selected coefficients and their estimated value using penalty term value from leave one out cross validation:

```
> extract.coef(lambda_loocv)
                    Value SE Coefficient
(Intercept) -6.051814e+01 NA (Intercept)
gene1        7.276829e+00 NA       gene1
gene11       1.181852e+01 NA      gene11
gene21       5.866931e+00 NA      gene21
gene39       1.995023e+00 NA      gene39
gene41       9.479081e+00 NA      gene41
gene51       1.063084e+01 NA      gene51
gene71       1.937643e+00 NA      gene71
gene81       1.850043e+00 NA      gene81
gene138     -3.157486e-01 NA     gene138
gene206     -1.785301e-02 NA     gene206
gene393      4.723856e-01 NA     gene393
gene689     -6.713344e-01 NA     gene689
gene761      1.458883e-01 NA     gene761
gene887      7.917126e-01 NA     gene887
gene909      4.067878e-01 NA     gene909
gene1090     6.028821e+00 NA    gene1090
gene1262    -5.059100e+00 NA    gene1262
gene1304     5.254057e-02 NA    gene1304
gene1461    -2.592561e-02 NA    gene1461
gene1484     1.572907e+00 NA    gene1484
gene2013    -5.584451e+00 NA    gene2013
gene2154     9.215664e-01 NA    gene2154
gene2168    -4.269470e-01 NA    gene2168
gene2323    -1.887987e-01 NA    gene2323
gene2351     6.951206e-01 NA    gene2351
gene2461     8.373157e-01 NA    gene2461
gene2552     1.021810e-03 NA    gene2552
gene2966    -2.005345e-02 NA    gene2966
gene3069     2.934732e-01 NA    gene3069
gene3240     2.159219e-01 NA    gene3240
gene3429     9.816510e-01 NA    gene3429
gene3653    -4.158387e-02 NA    gene3653
gene3696     1.142370e+00 NA    gene3696
gene4016    -2.365341e-01 NA    gene4016
gene4058     3.549253e-01 NA    gene4058
gene4071    -1.167328e-02 NA    gene4071
gene4150    -3.859594e+00 NA    gene4150
gene4432     6.911038e-01 NA    gene4432
gene4453    -6.809118e-04 NA    gene4453
gene4471    -1.386743e-01 NA    gene4471
gene4606    -7.512143e-01 NA    gene4606
gene4629    -2.576374e-01 NA    gene4629
```

```
gene4635    -1.765622e-01 NA    gene4635
gene4671     1.017755e+00 NA    gene4671
gene4983    -1.774699e-01 NA    gene4983
gene5038     5.308109e-01 NA    gene5038
gene5089     1.214964e+00 NA    gene5089
gene5183     8.728299e-01 NA    gene5183
gene5197    -6.451333e-01 NA    gene5197
gene5223    -9.678267e-02 NA    gene5223
gene5388     4.986500e-03 NA    gene5388
gene5470     2.538081e-01 NA    gene5470
gene5477     1.793532e+00 NA    gene5477
gene5508    -5.104041e-02 NA    gene5508
gene5603     1.959205e+00 NA    gene5603
gene5781     5.057209e+00 NA    gene5781
gene5788     1.836525e+00 NA    gene5788
gene6165    -1.263856e+00 NA    gene6165
gene6181     7.243338e-01 NA    gene6181
gene6213     1.052177e+00 NA    gene6213
gene6490    -1.758501e-01 NA    gene6490
gene6518    -1.793929e+00 NA    gene6518
gene6697    -3.037262e-01 NA    gene6697
gene6771    -6.526271e-02 NA    gene6771
gene6790     1.711839e+00 NA    gene6790
gene6889    -1.150001e+00 NA    gene6889
```

d)

Choose 10-fold cross validation to find the optimal value of lambda, as this is often said to be sufficient (as it was in the Lasso case as well):

```
Call:  cv.glmnet(x = t_df, y = response_df[, 2], nfolds = 10, alfa = 0,
standardize = T)

Measure: Mean-Squared Error

    Lambda Measure     SE Nonzero
min 0.3195   57.70  8.711      66
1se 0.9757   66.38 11.253      46
```

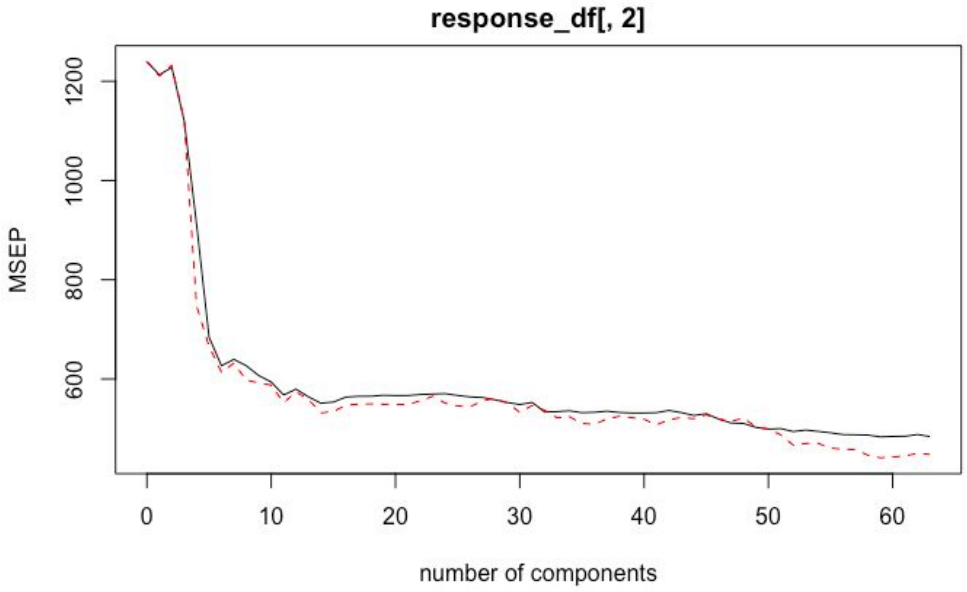
Find that the optimal lambda is 0.3195 with Ridge as well. Resulting first 11 coefficients:

```
                  Value SE Coefficient
(Intercept) -60.51814216 NA (Intercept)
gene1         7.27682887 NA       gene1
gene11       11.81852398 NA      gene11
gene21        5.86693116 NA      gene21
```

```
gene39        1.99502264 NA      gene39
gene41        9.47908082 NA      gene41
gene51       10.63083759 NA      gene51
gene71        1.93764327 NA      gene71
gene81        1.85004314 NA      gene81
gene138      -0.31574863 NA     gene138
gene206      -0.01785301 NA     gene206
gene393       0.47238564 NA     gene393
```

e)

Validation plot resulting from cross validation with increasing number of components:



Based on the plot, 4 components were selected as we see the largest improvement in terms of drop in MSEP (Mean Squared Error of Prediction) up until the 4th principal component.

Summary of model fitted with 4 principal components:

```
Data:       X dimension: 72 7128
      Y dimension: 72 1
Fit method: svdpc
Number of components considered: 4
TRAINING: % variance explained
                  1 comps  2 comps  3 comps  4 comps
X                   9.596   14.563    19.05    23.00
response_df[, 2]    3.659    3.721    15.98    57.22
```

f)

Below are the test errors for the 5 models:

```
           test_error
lasso_3     581.7487
lasso_10    599.8073
```

```
lasso_loocv 599.8073
ridge_10    599.8073
PCA_4       2496.307
```

g)

Results from doing c) and d) with optional penalty parameter (1se):

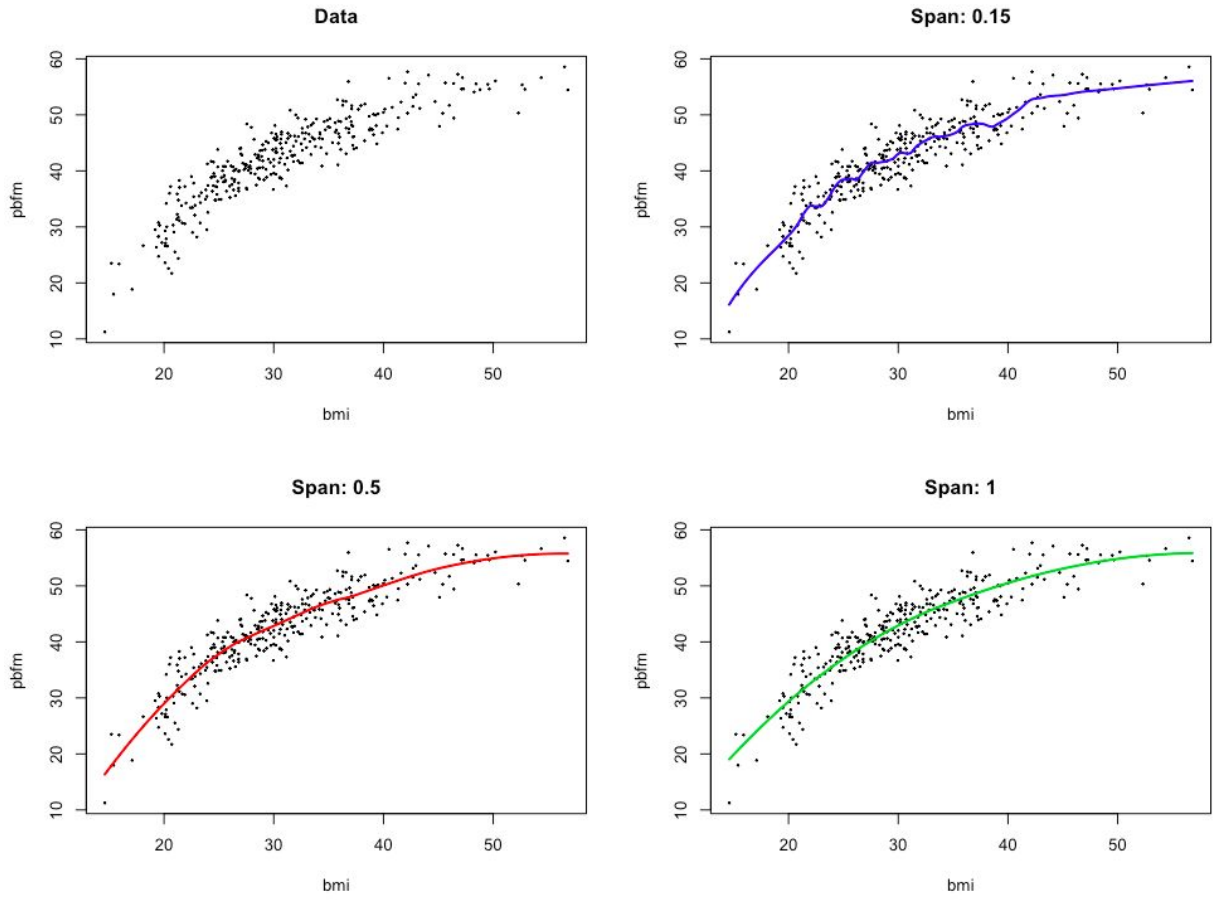
```
           test_error_1se
lasso_3     119.4597
lasso_10    474.1709
lasso_loocv 474.1709
ridge_10    464.0514
```

We see that the test error results are better using the latter value for the penalty parameter - as suggested by some authors. We see that it is the Lasso model where 3 folds is used in the cross validation which yields the best improvement and result. Also, we see that the Ridge regression model surpasses the other Lasso models using this penalty value.

Problem 2

a)

Figure showing 3 different local regression models with varying values of span, affecting the flexibility of the non-linear fit.



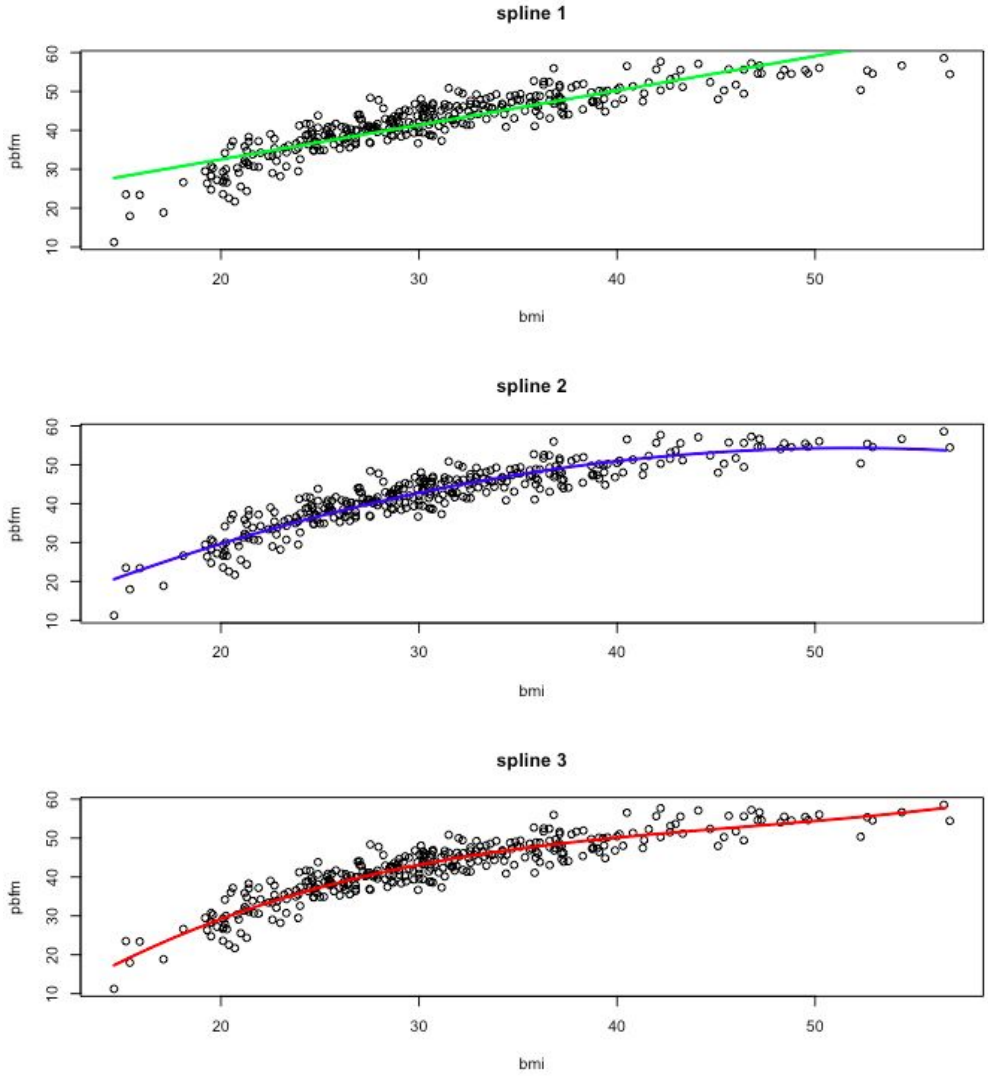
Different values of span were tested and visually investigated through plots, with the aim to find the fitted model which best generalized the overall distribution of the data. We can see from the plot that with a span of 0.15 the blue line is too wiggly, fitting to very local movements in the data. With a span of 1 the other hand, the green line, one could argue that the starting and middle part slightly fails at capturing the distribution of data points. For instance in the middle section, around a BMI of 40, the fitted model does not capture the fact that more of the data points at at a lower pbfm. This is however captured by the fitted model using a span of 0.5, were the slope of the curve decreases a bit at approximately BMI 25-30

b)

Found the optimal span to be 0.6148132 using loess.as.

c)

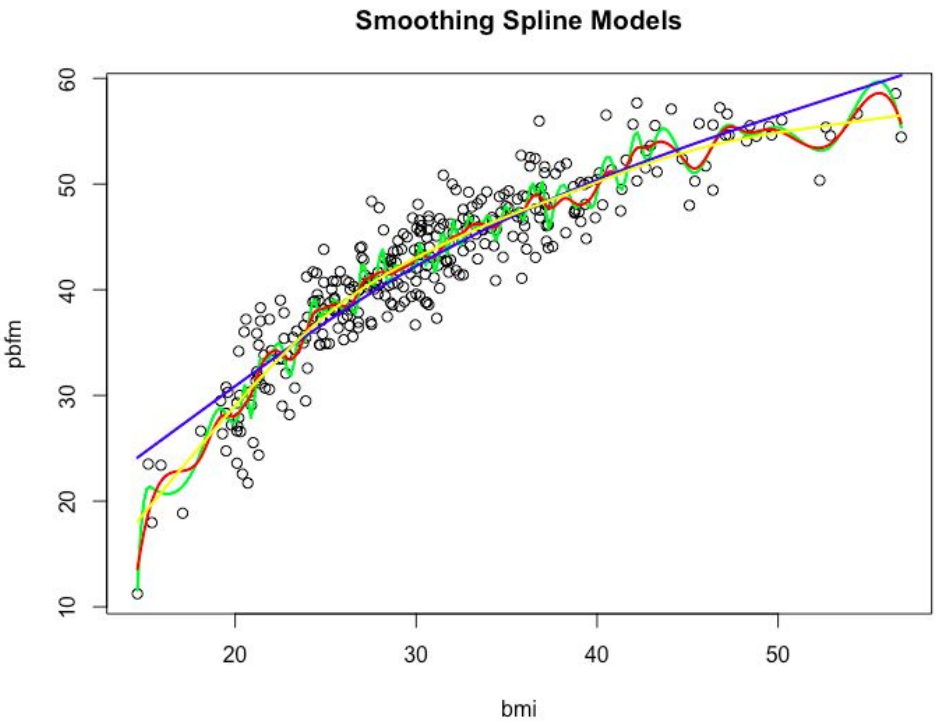
Below is the result of the tree fitted regression spline models of order 1, 2 and 3, using 4 degrees of freedom (see explanation below).



The placement of the knots were chosen in accordance with what is suggested by James et al. (2013). They suggest that one could choose to manually place more knots where we believe/observe that the function we are trying to model seems to change rapidly, introducing more flexibility in these locations. Still, they say that it is common practice to place the knots uniformly, by specifying the desired degrees of freedom and let the software place the same number of knots (4 in our case) at uniform quantiles of the data.

d)

The resulting plot shows three fitted smoothing spline models with 3 different smoothing parameters chosen manually, and one chosen through LOOCV (yellow line).



When we fit a smoothing spline, we do not have to select the number of knots - as there will be one at each data point. Instead we choose the tuning parameter λ. The tuning parameter in smoothing splines controls the “roughness” of the smoothing, and as such the effective number of degrees of freedom. Here, three values were selected manually (green: 0.25, red: 0.5, blue: 1.25) and one selected through LOOCV as suggested by James et al. (2013) (in yellow). Using LOOCV finds the optimal value of the tuning parameter to be 1.023437. The plot above shows how the fitted model using the tuning parameter selected through LOOCV seems to generalize best to the data, not adhering to local variations.

e)

In this section, the following models were fitted:

- One linear model.
- 3 polynomial models of order 3, 5 and 8.
- One local regression model using the optimal span from 2b).
- 3 spline regression models of degree 1, 2 and 3.

The data was split into a train and test set. ⅔ for training, and ⅓ for testing. The MSE was calculated for all of the models for comparison.

The resulting MSEs were:

```
> cbind(mse)
         mse
linear   115.8149
poly_3   120.5885
poly_5   120.6858
```

```
poly_8   120.3267
localr   120.332
spline_1 120.6496
spline_2 120.7263
```

The results in terms of MSE indicate that the linear model fits best.

References:

1) Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *An Introduction to Statistical Learning : with Applications in R.* New York :Springer, 2013.

Appendix: R code to produce the output presented in main section

```
# Mandatory exercise 2, Spring 2020, by Ida Austad

# PROBLEM 1

install.packages("glmnet", repos = "https://cran.us.r-project.org")
library(pls)
library(glmnet)
library(coefplot)

#import data
df <- read.csv("http://web.stanford.edu/~hastie/CASI_files/DATA/leukemia_big.csv",
               header = T,
               sep = ",")

num_patients = 72
num_gene_expressions = 7128

AML_patients = grepl("AML", names(df))
ALL_patients = grepl("ALL", names(df))

#a)

#perform principal component analysis of df, with 2 principal components.
PC_analysis = prcomp(t(df), center = T, scale = T, rank. = 2)
summary(PC_analysis)

#plot the two principal components in different colors.
plot(PC_analysis$x, col=c("blue", "red"), main="Principal Component Analysis - Leukaemia",
xlab="PC1", ylab = "PC2", pch=1)
X = PC_analysis$x
points(X[ALL_patients,1], X[ALL_patients, 2], col = "4", pch = 19)
points(X[AML_patients,1], X[AML_patients, 2], col = "2", pch = 19)
legend("topright", legend = c("ALL", "AML"), col = c(2, 4), pch = c(19, 19))

#b)
response_df <-
read.csv("https://www.uio.no/studier/emner/matnat/math/STK2100/v20/eksamen/response_train.csv",
               header = T,
               sep = ",")

#transpose and rename columns
```

```
t_df=t(df)
colnames(t_df) <- paste0("gene", 1:ncol(t_df))

set.seed(222)

# Lasso regularization with cross validation: 3 folds
lambda_cv_3 = cv.glmnet(x =t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds = 3)
lambda_cv_3

# Lasso regularization with cross validation: 10 folds
lambda_cv_10 = cv.glmnet(x =t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds = 10)
lambda_cv_10

# Lasso regularization with cross validation: 72 folds (number of patients)
lambda_loocv = cv.glmnet(x =t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds = 72)
lambda_loocv

# c)
# fit lasso models with the optimal (i.e. minimal) lambda value obtained in
# the various k-fold cvs
lasso_model_3 = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds = 3,
lambda = lambda_cv_3$lambda.min)

lasso_model_10 = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds = 10,
lambda = lambda_cv_10$lambda.min)

lasso_model_loocv = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds =
72, lambda = lambda_loocv$lambda.min)

# get the features which are not set to zero in the various models
# (using the best version given by the
# CV-models directly for nice output)
extract.coef(lambda_cv_3)
extract.coef(lambda_cv_10)
extract.coef(lambda_loocv)

# d)

N = 11

# set alfa to 0 for Ridge, and use 10-fold cross validation to find optimal penalty term
ridge_cv_10 = cv.glmnet(x=t_df, y = response_df[,2], alfa = 0, standardize = T, nfolds = 10,
lambda = )
ridge_cv_10

#fit model using optimal penalty term
ridge_model_10  = glmnet(x = t_df, y = response_df[,2], alfa = 0, standardize = T, nfolds = 10,
lambda = ridge_cv_10$lambda.min)
```

```r
#get 11 first coefficients (excluding intercept)
ridge_coefs = extract.coef(ridge_cv_10)
head(ridge_coefs,N+1)


# e)

# perform PCA using CV for selecting number of principal components
PCA_model_CV = pcr(response_df[,2]~., data = data.frame(t_df), scale = T, validation = "CV")

# plot to select number of components
validationplot(PCA_model_CV, val.type="MSEP")

# select 4 components
model_PCA_4 = pcr(response_df[,2]~t_df, scale = T, ncomp = 4)
validationplot(model_PCA_4, val.type="MSEP" )

summary(model_PCA_4)

# f)

# import test data
test_df <-
read.csv("https://www.uio.no/studier/emner/matnat/math/STK2100/v20/eksamen/test_set.csv",
               header = T,
               sep = ",")

# calculate test error results
test_error <- list()
test_error$lasso_3 <- mean((test_df$y - predict(lasso_model_3, newx =
as.matrix(test_df[,0:-1])))^2)
test_error$lasso_10 <- mean((test_df$y - predict(lasso_model_10, newx =
as.matrix(test_df[,0:-1])))^2)
test_error$lasso_loocv <- mean((test_df$y - predict(lasso_model_loocv, newx =
as.matrix(test_df[,0:-1])))^2)
test_error$ridge_10 <- mean((test_df$y - predict(ridge_model_10, newx =
as.matrix(test_df[,0:-1])))^2)
test_error$PCA_4 <- mean((test_df$y - predict(PCA_model_CV, newdata =
as.matrix(test_df[,0:-1]), ncomp=4))^2)

cbind(test_error)

# g)

# fit lasso models with the "optional"" lambda value (within 1 se of the error)
lasso_model_3_1se = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds =
3, lambda = lambda_cv_3$lambda.1se)
```

```
lasso_model_10_1se = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds =
10, lambda = lambda_cv_10$lambda.1se)

lasso_model_loocv_1se = glmnet(x = t_df, y = response_df[,2], alfa = 1, standardize = T, nfolds
= 72, lambda = lambda_loocv$lambda.1se)

#fit rigde regression model using optional lambda
ridge_model_10_1se = glmnet(x = t_df, y = response_df[,2], alfa = 0, standardize = T, nfolds =
10, lambda = ridge_model_10$lambda.1se)

# calculate test error with optional penalty term
test_error_1se <- list()
test_error_1se$lasso_3 <- mean((test_df$y - predict(lasso_model_3_1se, newx =
as.matrix(test_df[,0:-1])))^2)
test_error_1se$lasso_10 <- mean((test_df$y - predict(lasso_model_10_1se, newx =
as.matrix(test_df[,0:-1])))^2)
test_error_1se$lasso_loocv <- mean((test_df$y - predict(lasso_model_loocv_1se, newx =
as.matrix(test_df[,0:-1])))^2)
test_error_1se$ridge_10 <- mean((test_df$y - predict(ridge_model_10_1se, newx =
as.matrix(test_df[,0:-1])))^2)

cbind(test_error_1se)

# ------------------------------------------------------------------------

# PROBLEM 2

library(kknn)
library(splines)
library(fANCOVA)

# read in, inspect and prep data
path <- file.path(getwd() , "res_bodyfat.csv")
bodyfat <- read.csv(path)
summary(bodyfat)
head(bodyfat)
attach(bodyfat)

# a)

# do local regression with different spans (0.15,0.5 & 1)
X_0 = seq(min(bmi), max(bmi), length=250)

# plot points without model
par(mfrow=c(2,2))
plot(bmi, pbfm, cex =0.2, main= "Data")

# fit model with span 0.15
plot(bmi, pbfm, cex =0.2, main= "Span: 0.15")
```

```
lr_015 = loess(pbfm~bmi, span=0.15, data = bodyfat)
lines(x = X_0, y = predict(lr_015, newdata = X_0), col = "4", lwd=2)

# fit model with span 0.5
plot(bmi, pbfm, cex =0.2, main= "Span: 0.5")
lr_05 = loess(pbfm~bmi, span=0.5, data = bodyfat)
lines(x = X_0, y = predict(lr_05, newdata = X_0), col = "2", lwd=2)

# fit model with span 1
plot(bmi, pbfm, cex =0.2, main= "Span: 1")
lr_1 = loess(pbfm~bmi, span=1, data = bodyfat)
lines(x = X_0, y = predict(lr_1, newdata = X_0), col = "3", lwd=2)

# b)

# fit and plot local regression model (when not setting user.span, it is chosen to the optimal)
lr_best = loess.as(pbfm, bmi, criterion="gcv", plot=TRUE)

#look at summary to find optimal span
summary(lr_best)

# c)

par(mfrow=c(3,1))
bmi_range =range(bmi)
X_sequence = seq(from =bmi_range[1], to= bmi_range[2])

#fit regression spline models of different orders (1,2,3)
spline_model_1 = lm(pbfm ~ bs(bmi, df=4, degree=1), data=bodyfat )
spline_model_2 = lm(pbfm ~ bs(bmi, df=4, degree=2), data=bodyfat )
spline_model_3 = lm(pbfm ~ bs(bmi, df=4, degree=3), data=bodyfat )

# predict using the spline models
pred_smodel_1 = predict(spline_model_1, newdata = list(bmi=X_sequence), se=T)
pred_smodel_2 = predict(spline_model_2, newdata = list(bmi=X_sequence), se=T)
pred_smodel_3 = predict(spline_model_3, newdata = list(bmi=X_sequence), se=T)

# plot result of the three predictions
plot(bmi, pbfm, main = "spline 1")
lines(X_sequence, pred_smodel_1$fit, col="green", lwd=2)
plot(bmi, pbfm, main = "spline 2")
lines(X_sequence, pred_smodel_2$fit, col="blue", lwd=2)
plot(bmi, pbfm, main = "spline 3")
lines(X_sequence, pred_smodel_3$fit, col="red", lwd=2)

# d)
X_0 = seq(min(bmi), max(bmi), length = 250)

# fit smoothing spline models, with varying smoothing parameters (manual vs using loocv)
```

```r
smooth_model_025 = smooth.spline(bmi, pbfm, spar = 0.25)
smooth_model_05 = smooth.spline(bmi, pbfm, spar = 0.5)
smooth_model_125 = smooth.spline(bmi, pbfm, spar = 1.25)
smooth_model_auto = smooth.spline(bmi, pbfm, cv=TRUE)

#get smoothing paramter value chosen through loocv
smooth_model_auto$spar

# plot all fitted models
par(mfrow=c(1,1))
plot(bmi, pbfm, main = "Smoothing Spline Models")
lines(predict(smooth_model_025, x = X_0), col="green", lwd=2)
lines(predict(smooth_model_05, x = X_0), col="red", lwd=2)
lines(predict(smooth_model_125, x = X_0), col="blue", lwd=2)
lines(predict(smooth_model_auto, x = X_0), col="yellow", lwd=2)

# e)

set.seed(222)
n = length(bmi)

# create train and test sets
train <- sample(n, 2*n/3, replace = FALSE)
X.train = bmi[train]
y.train = pbfm[train]
X.test = bmi[-train]
y.test = pbfm[-train]
model <- list()
mse <- list()

# Fit simple linear regression model
model$linear <- lm(y.train~X.train, bodyfat)
summary(model$linear)
mse$linear <- mean((y.test - predict(model$linear, x = X.test))^2)

# Fit polynomial regression model
model$poly_3 <- lm(y.train~poly(X.train, degree = 3))
model$poly_5 <- lm(y.train~poly(X.train, degree = 5))
model$poly_8 <- lm(y.train~poly(X.train, degree = 8))
mse$poly_3 <- mean((y.test - predict(model$poly_3, x = X.test))^2)
mse$poly_5 <- mean((y.test - predict(model$poly_5, x = X.test))^2)
mse$poly_8 <- mean((y.test - predict(model$poly_8, x = X.test))^2)
summary(model$poly_3)
summary(model$poly_5)
summary(model$poly_8)

# Fit local regression model (using optimal span found previously)
model$localr = loess(y.train~X.train, span=0.6148132, data= data.frame(X.train))
mse$localr <- mean((y.test - predict(model$localr, x = X.test))^2)
```

```
summary(model$localr)

# Fit regression spline models
train_range =range(X.train)
X.seq = seq(from =train_range[1], to= train_range[2])

model$spline_1 = lm(y.train ~ bs(X.train, df=4, degree=1), data=bodyfat )
model$spline_2 = lm(y.train ~ bs(X.train, df=4, degree=2), data=bodyfat )
model$spline_3 = lm(y.train ~ bs(X.train, df=4, degree=3), data=bodyfat )

mse$spline_1 <- mean((y.test - predict(model$spline_1, x = X.test))^2)
mse$spline_2 <- mean((y.test - predict(model$spline_2, x = X.test))^2)
mse$spline_3 <- mean((y.test - predict(model$spline_3, x = X.test))^2)

summary(model$spline_1)
summary(model$spline_2)
summary(model$spline_3)

# compare resulting mses to find best model
cbind(mse)
```