A Short Course in Machine Learning

(Part III : March 4–5, 2021)

true

Contents

Introduction

This document includes all codes used in the slides for part III of the workshop. It is best if the slidset and this document are used in tendem.

The codes in this document have the following dependencies:

- The dataset `HouseData` which has been shared with you.
- The `R` packages
  - `MASS`
  - `splines`
  - `glmnet`
  - `lars`
  - `FNN`

Reminder: an `R` package can be installed in R using the command `install.packages("TheNameOfThePackage")` and it can be called with the command `library(TheNameOfThePackage)`.

1) Taylor Approximation and Polynomial Regression (Slides 9, 10)

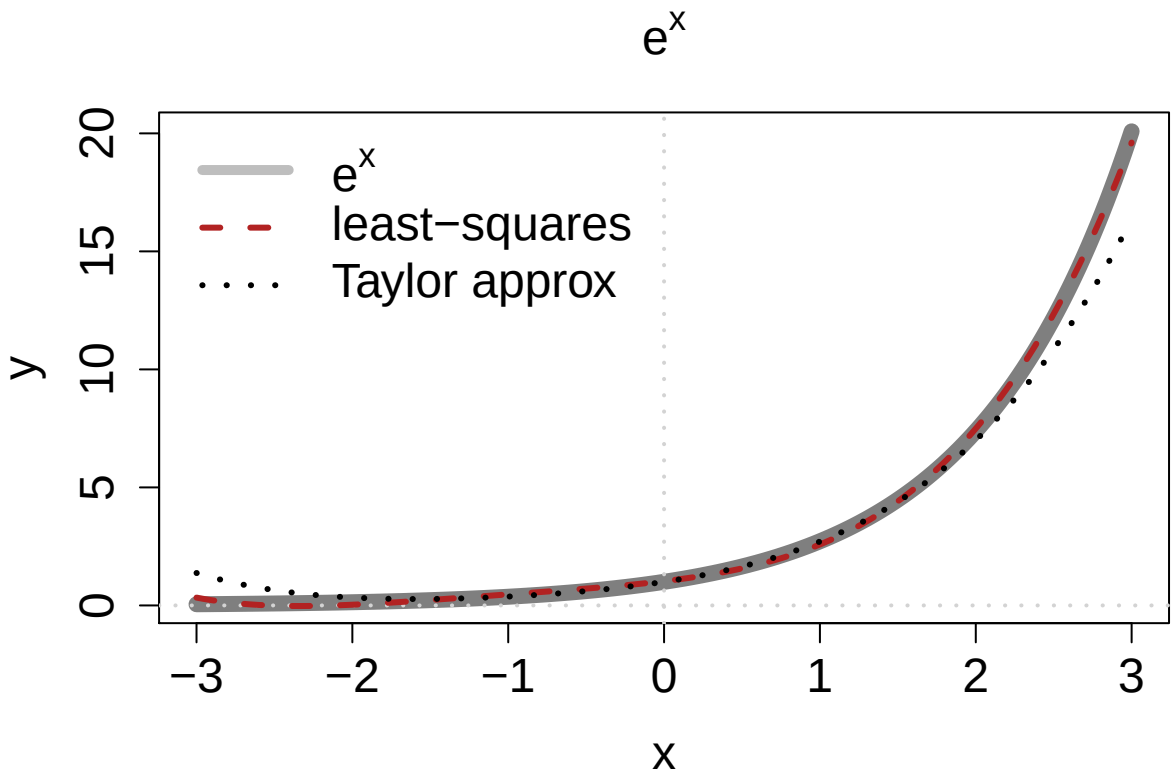
```
x <- seq(-3, 3, length.out = 1000)
y <- exp(x)
# We fit our "model" as before
#
fit <- lm(y~x + I(x^2) + I(x^3) + I(x^4))
#
# We can now compare coefficients to the Taylor series values
# First the fit
round(fit$coefficients,3)
```

```
## (Intercept)           x      I(x^2)      I(x^3)      I(x^4)
##       1.029       0.788       0.435       0.269       0.062
```

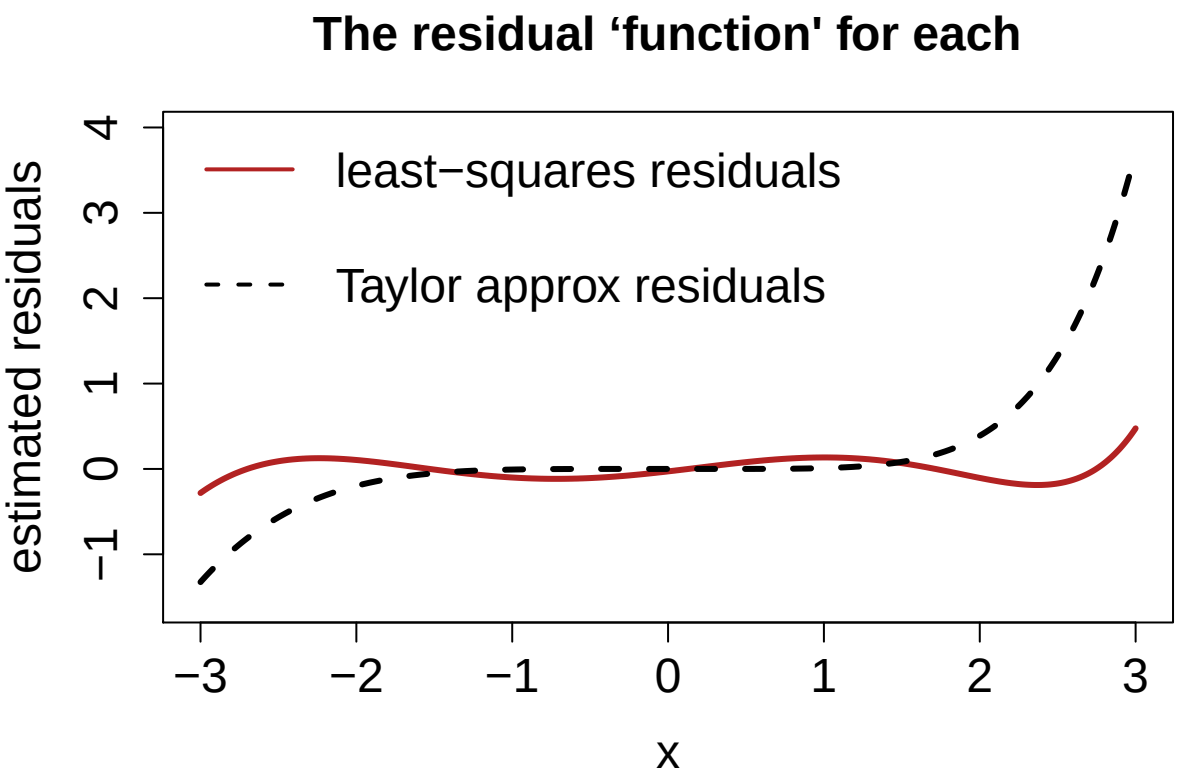
```
# and now the Taylor coefficients
round(sapply(0:4, function(x) 1/factorial(x)), 3)
```

```
## [1] 1.000 1.000 0.500 0.167 0.042
```

```
# We can compare them visually
plot(x, y, col=adjustcolor("black", 0.5), type="l",
     main = expression(e^x), xlab="x", ylab="y" , lwd=8,
     cex.lab=1.5 , cex.axis=1.5 , cex.main=1.5)
abline(h=0, col="lightgrey", lty=3 , lwd=2)
abline(v=0, col="lightgrey", lty=3, lwd=2)

# Note that the x's are in order 1:N so order(x) was unnecessary here
lines(x, fit$fitted.values, col="firebrick", lty=2, lwd=3)

# The Taylor approx
ytaylor = 1 + x + x^2/2  + x^3/6 + x^4 /24
lines(x, ytaylor, col="black", lty=3, lwd=3)

# Add a legend
legend("topleft",  bty='n', cex=1.5,
       legend = c(expression(e^x), "least-squares", "Taylor approx"),
       col = c("grey", "firebrick", "black"),
       lty=c(1, 2,3 ), lwd=c(5, 3, 3),
       text.width = 1.5
)
```




```r
# We can loook at the residuals
taylor_resids <- y - ytaylor
residlim <- extendrange(c(taylor_resids, fit$residuals))

# The residuals from ls
plot(x, fit$residuals, type="l" , lwd=3, col="firebrick",  ylim=residlim,
     main = "The residual `function' for each",
     xlab="x", ylab="estimated residuals" ,
     cex.axis=1.5 , cex.lab=1.5, cex.main=1.5)
# Taylor residuals
lines(x, taylor_resids, lty=2 , lwd=3, col="black")

# Add a legend
legend("topleft", bty="n", cex=1.5,
       legend = c("least-squares residuals", "", "Taylor approx residuals"),
       col = c("firebrick","","black"),
       lty = c(1,0,2), lwd=2,
       text.width = 2
)
```

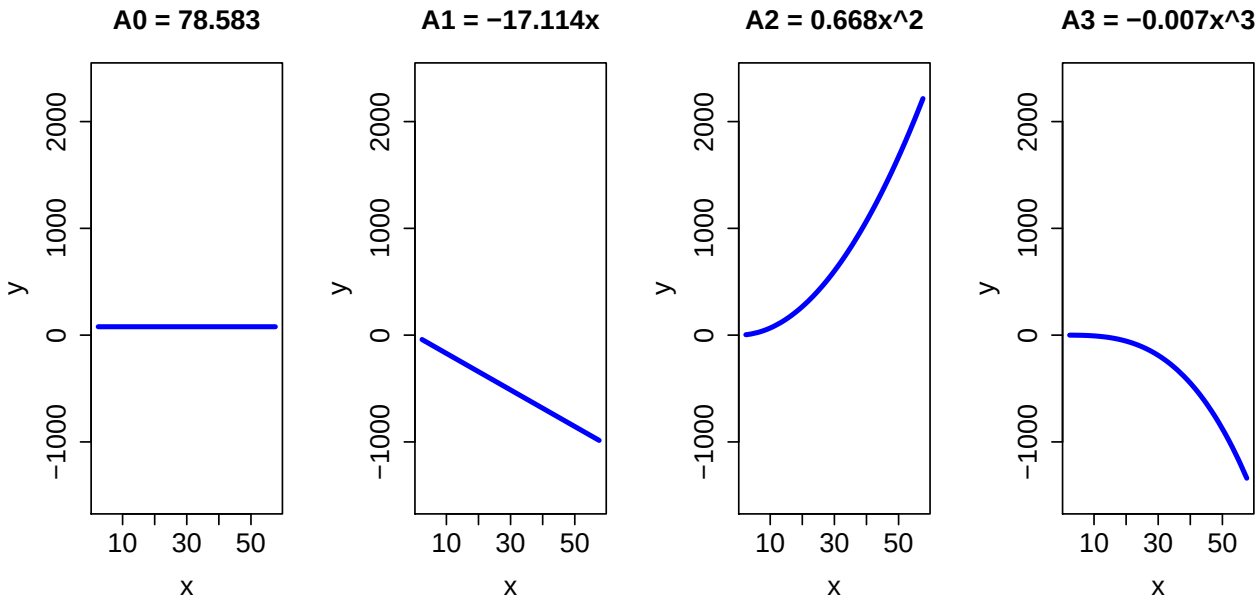

The residual ‘function' for each
least−squares residuals
Taylor approx residuals
estimated residuals
−1
0
1
2
3
4
−3
−2
−1
0
1
2
3
x

2) Generators and Polynomial Regression (Slide 13)

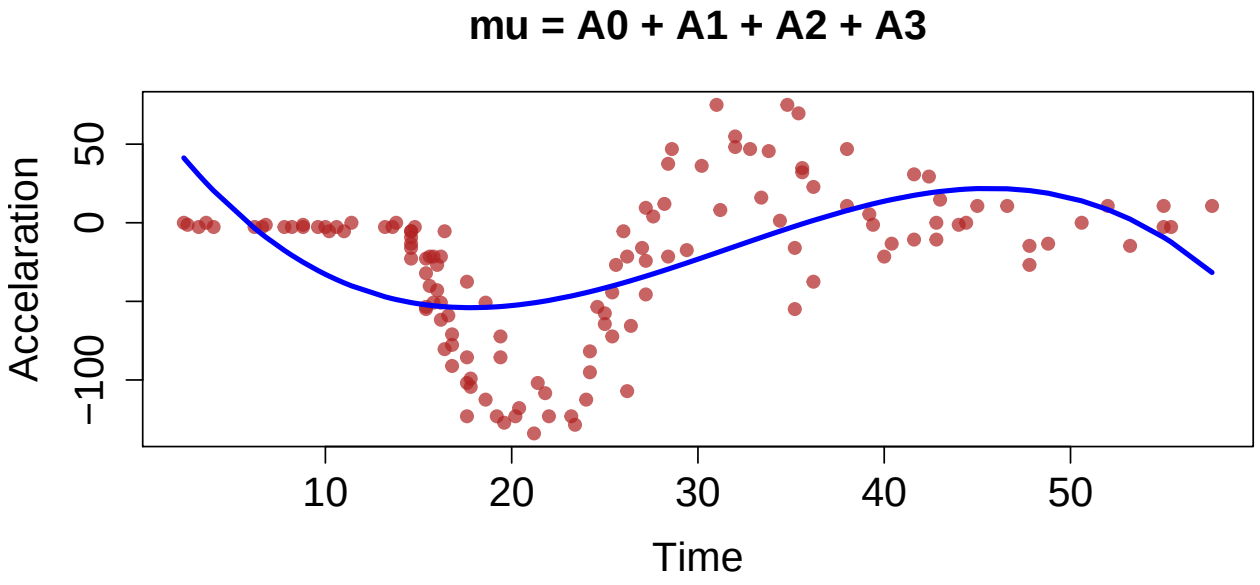
```
library(MASS)
data(mcycle)
fit <- lm(accel ~ times+I(times^2)+I(times^3) , data=mcycle)
round(coef(fit),3)
```

```
## (Intercept)       times  I(times^2)  I(times^3)
##      78.583     -17.114       0.668      -0.007
```

```
Xorder <- order(mcycle$times)
n <- length(Xorder)
g0 <- rep(fit$coefficients[1], n)    # constant term  x^0
g1 <- mcycle$times * fit$coefficients[2]     # linear term x
g2 <- mcycle$times^2 * fit$coefficients[3]   # quadratic term x^2
g3 <- mcycle$times^3 * fit$coefficients[4]   # cubic term x^3

savePar <- par(mfrow=c(1,4))
glim <- extendrange(c(g0, g1, g2, g3))
plot(mcycle$times[Xorder], g0, col="blue", type="l",
     ylim=glim, lwd=3,
     main=paste0("A0 = ",
                 round(fit$coefficients[1],3)),
     xlab="x", ylab="y",
     cex.main=1.5 , cex.axis=1.5 , cex.lab=1.5)
plot(mcycle$times[Xorder], g1[Xorder], col="blue", type="l",
     ylim=glim, lwd=3,
     main=paste0("A1 = ",
                 round(fit$coefficients[2],3),
                 "x"),
     xlab="x", ylab="y",
     cex.main=1.5 , cex.axis=1.5 , cex.lab=1.5)
plot(mcycle$times[Xorder], g2[Xorder], col="blue", type="l",
     ylim=glim, lwd=3,
     main=paste0("A2 = ",
                 round(fit$coefficients[3],3),
                 "x^2"),
     xlab="x", ylab="y",
     cex.main=1.5 , cex.axis=1.5 , cex.lab=1.5)
plot(mcycle$times[Xorder], g3[Xorder], col="blue", type="l",
     ylim=glim, lwd=3,
     main=paste0("A3 = ",
                 round(fit$coefficients[4],3),
                 "x^3"),
     xlab="x", ylab="y",
     cex.main=1.5 , cex.axis=1.5 , cex.lab=1.5)
```

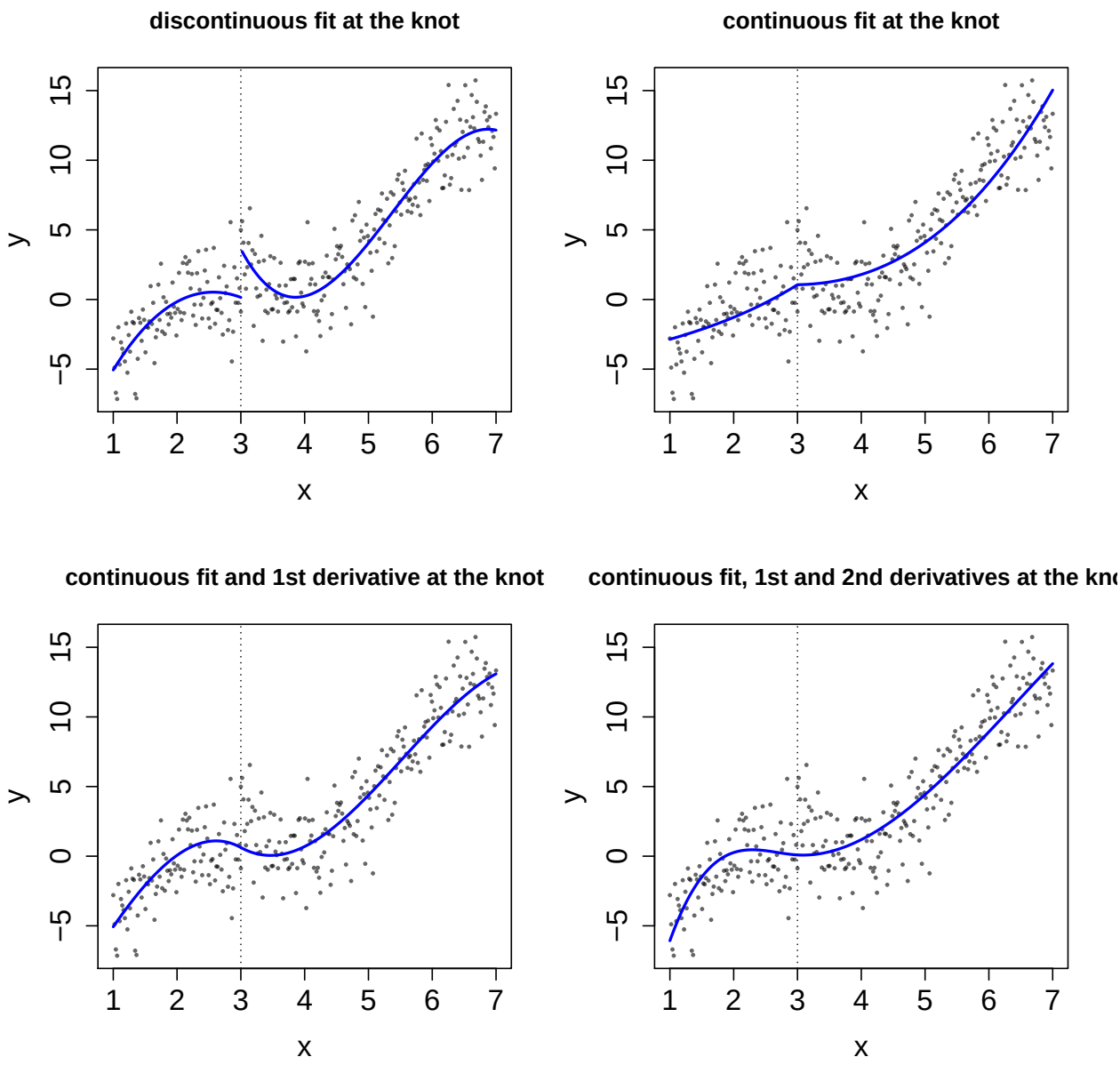



```
par(mfrow=c(1,1))
plot(mcycle$times, mcycle$accel,
     main = "mu = A0 + A1 + A2 + A3",
     xlab = "Time",
     ylab = "Accelaration",
     pch=19,
     col=adjustcolor("firebrick", 0.7),
     cex.main=1.5 , cex.axis=1.5 , cex.lab=1.5)

lines(mcycle$times[Xorder], g0[Xorder] + g1[Xorder] + g2[Xorder] + g3[Xorder],
      col="blue", lwd=3)
```



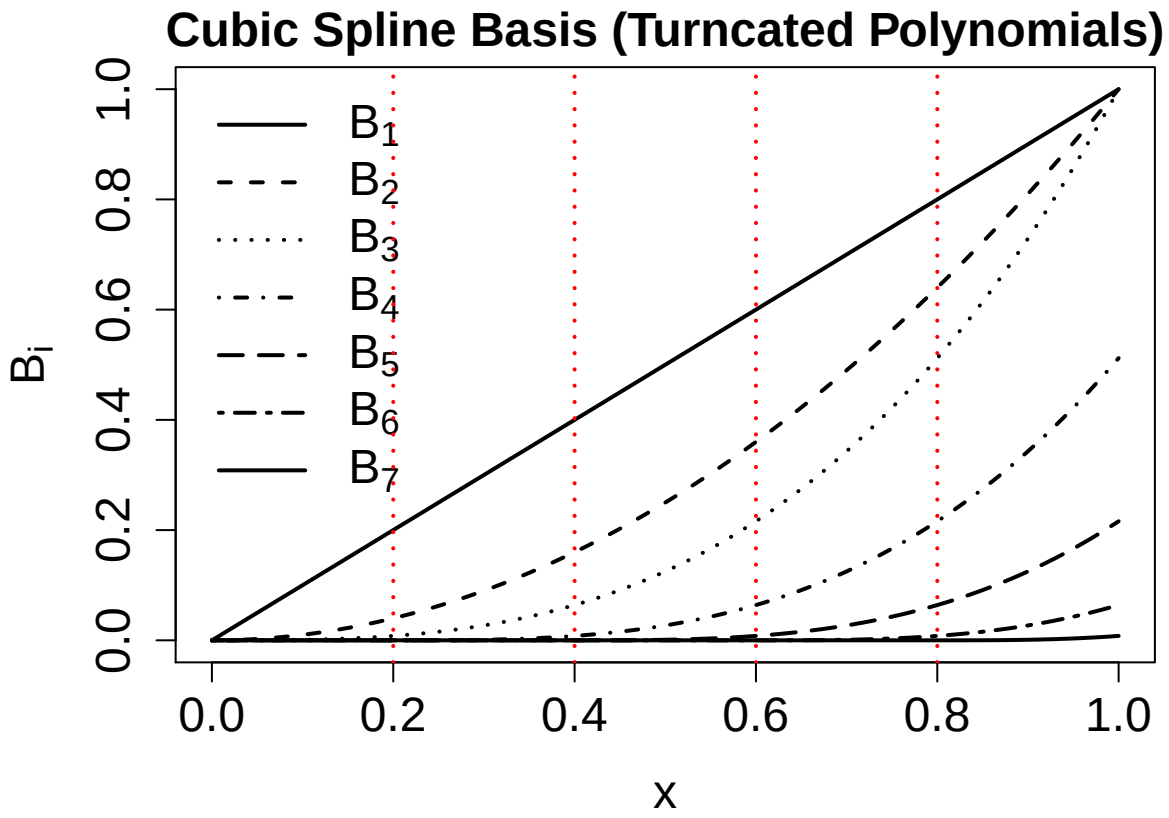
3) Piecewise Cubic Model (Slide 27)

```
set.seed(844)
x1 = seq(1,3,length=100)
y1 = (x1-4)*(x1-2)*(x1-3)+ rnorm(100,sd=2)
x2 = seq(3,7,length=200)
y2=4-(x2-3)*(x2-5)*(x2-8) + rnorm(200,sd=2)
x=c(x1,x2)
y=c(y1,y2)

SimulatedData2 = data.frame(x,y)

# piecewise cubic (discontinuous)
model1 = lm(y ~ x+I(x^2)+I(x^3) , data=subset(SimulatedData2, x<=3))
pred1 = predict(model1)
model2 = lm(y ~ x+I(x^2)+I(x^3) , data=subset(SimulatedData2, x>3))
pred2 = predict(model2)


## Looping through constraints.
title.graph = c("continuous fit at the knot",
                "continuous fit and 1st derivative at the knot",
                "continuous fit, 1st and 2nd derivatives at the knot"
                )

par(mfrow=c(2,2))
plot(y~x , cex.lab=1.5 , cex.axis=1.5 , data=SimulatedData2,
     main="discontinuous fit at the knot",
     col = adjustcolor("black",0.6) , pch=16 , cex=0.5)

abline(v=3 , lwd=1 , lty=3)
lines(pred1~SimulatedData2$x[SimulatedData2$x<=3] , lwd=2 , col="blue")
lines(pred2~SimulatedData2$x[SimulatedData2$x>3] , lwd=2, col="blue")


for(i in 1:3){
    SimulatedData2$D = ((SimulatedData2$x-3)^i)*(SimulatedData2$x>3)

    plot(y~x , cex.lab=1.5 , cex.axis=1.5 , data=SimulatedData2,
     col = adjustcolor("black",0.6) , pch=16 , cex=0.5,
     main = title.graph[i])
    abline(v=3 , lwd=1 , lty=3)

    model3 = lm(y ~ x+I(x^2)+I(x^3)+D , data=SimulatedData2)
    pred3 = predict(model3)
    lines(pred3~SimulatedData2$x , lwd=2, col="blue")

}
```

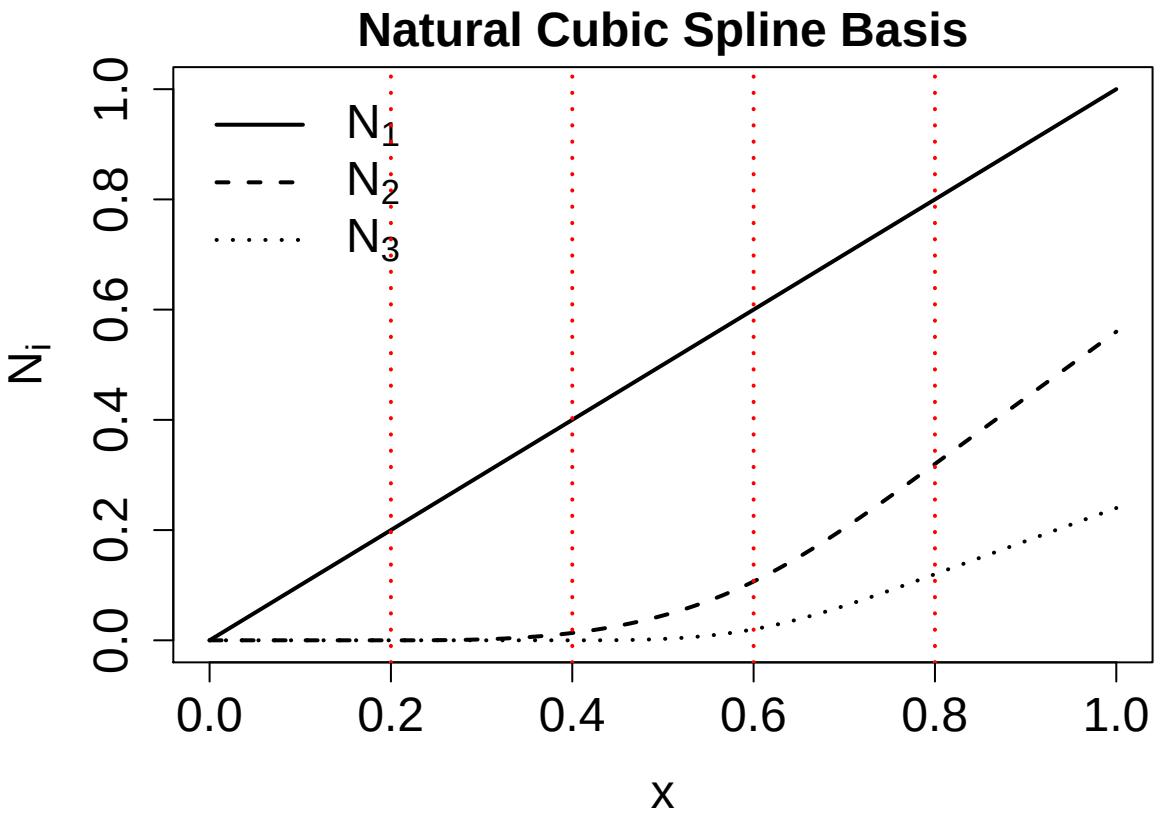

discontinuous fit at the knot
continuous fit at the knot
continuous fit and 1st derivative at the knot
continuous fit, 1st and 2nd derivatives at the kn
y
x
−5
0
5
10
15
1
2
3
4
5
6
7

4) Fitting Spline Models (Slides 32–51)

```r
x = seq(0,1,length=100)
knots = c(0.2 , 0.4 , 0.6 , 0.8)

# Defining cubic splines basis functions
u = function(x,a){
  temp = x-a
  sapply(temp , max , 0)
}

B1=x
B2=x^2
B3=x^3
B4=(u(x,0.2))^3
B5=(u(x,0.4))^3
B6=(u(x,0.6))^3
B7=(u(x,0.8))^3

# Plotting cubic splines basis functions
par(mar=c(5,5,2,2))
plot(B1~x,type="l" , ylab = expression(B[i]),
     cex.lab = 1.5 , cex.axis=1.5 , cex.main = 1.5, lty=1, col=1,lwd=2,
     main = "Cubic Spline Basis (Turncated Polynomials)")
lines(B2~x , lty=2 , col=1,lwd=2)
lines(B3~x, lty=3, col=1,lwd=2)
lines(B4~x, lty=4, col=1,lwd=2)
lines(B5~x, lty=5, col=1,lwd=2)
lines(B6~x, lty=6, col=1,lwd=2)
lines(B7~x, lty=7, col=1,lwd=2)
legend("topleft", bty="n", cex=1.5,
       legend = c(expression(B[1]), expression(B[2]),expression(B[3]),
                  expression(B[4]), expression(B[5]) , expression(B[6]),
                  expression(B[7])), lwd=2, lty = 1:7, text.width = 2)
abline(v=knots , lty=3, lwd=2 , col="red")
```

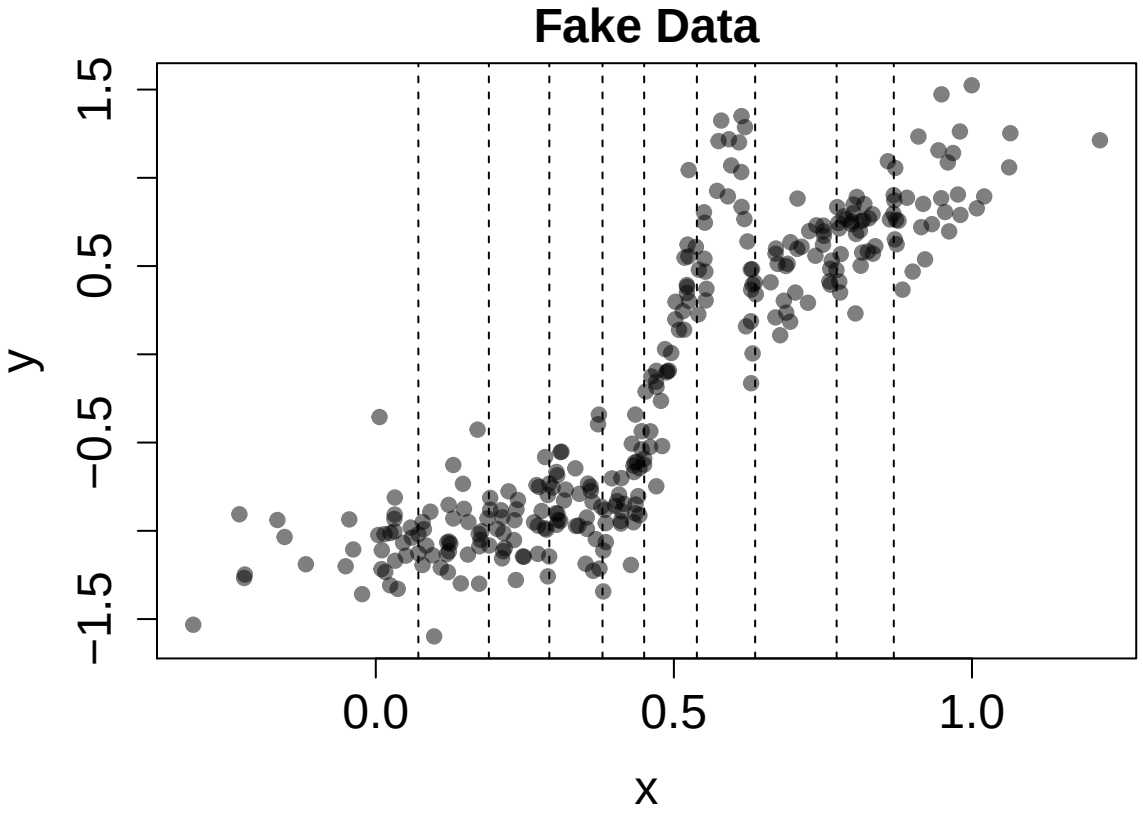



```r
# Defining natural cubic splines basis functions
N1=x
N2=(u(x,0.2)^3-u(x,0.8)^3)/(0.8-0.2)-(u(x,0.6)^3-u(x,0.8)^3)/(0.8-0.6)
N3=(u(x,0.4)^3-u(x,0.8)^3)/(0.8-0.4)-(u(x,0.6)^3-u(x,0.8)^3)/(0.8-0.6)

# Plotting natural cubic splines basis functions
par(mar=c(5,5,2,2))
plot(N1~x,type="l" , ylab = expression(N[i]),
     cex.lab = 1.5 , cex.axis=1.5 , cex.main = 1.5, lty=1, col=1,lwd=2,
     main = "Natural Cubic Spline Basis")
lines(N2~x , lty=2 , col=1,lwd=2)
lines(N3~x, lty=3, col=1,lwd=2)
legend("topleft", bty="n", cex=1.5,
       legend = c(expression(N[1]), expression(N[2]),expression(N[3])),
                  lwd=2, lty = 1:3, text.width = 2)
abline(v=knots , lty=3, lwd=2 , col="red")
```




```
## Fake Data
#
# Fake data example
#
# Get some x's
#
set.seed(12032345)
N <- 300
x <- runif(150, 0,1)
x <- c(x, rnorm(150, 0.5, 0.3))
#
#  A function for the mean of ys
#
mu <- function(x) {
  xrange <- range(x)
  vals <- vector("numeric", length=length(x))
  breaks <- quantile(x, probs=c(1/3,2/3))
  first <- x <= breaks[1]
  second <- (x > breaks[1]) & (x <= breaks[2])
  third  <- x > breaks[2]
  vals[first]  <- -(1 - x[first])^0.5 -0.1
  vals[second] <- sin(x[second] * 4 * pi) + x[second]/10
  vals[third]  <- x[third]^2
  vals
}

#
# the mean function
```

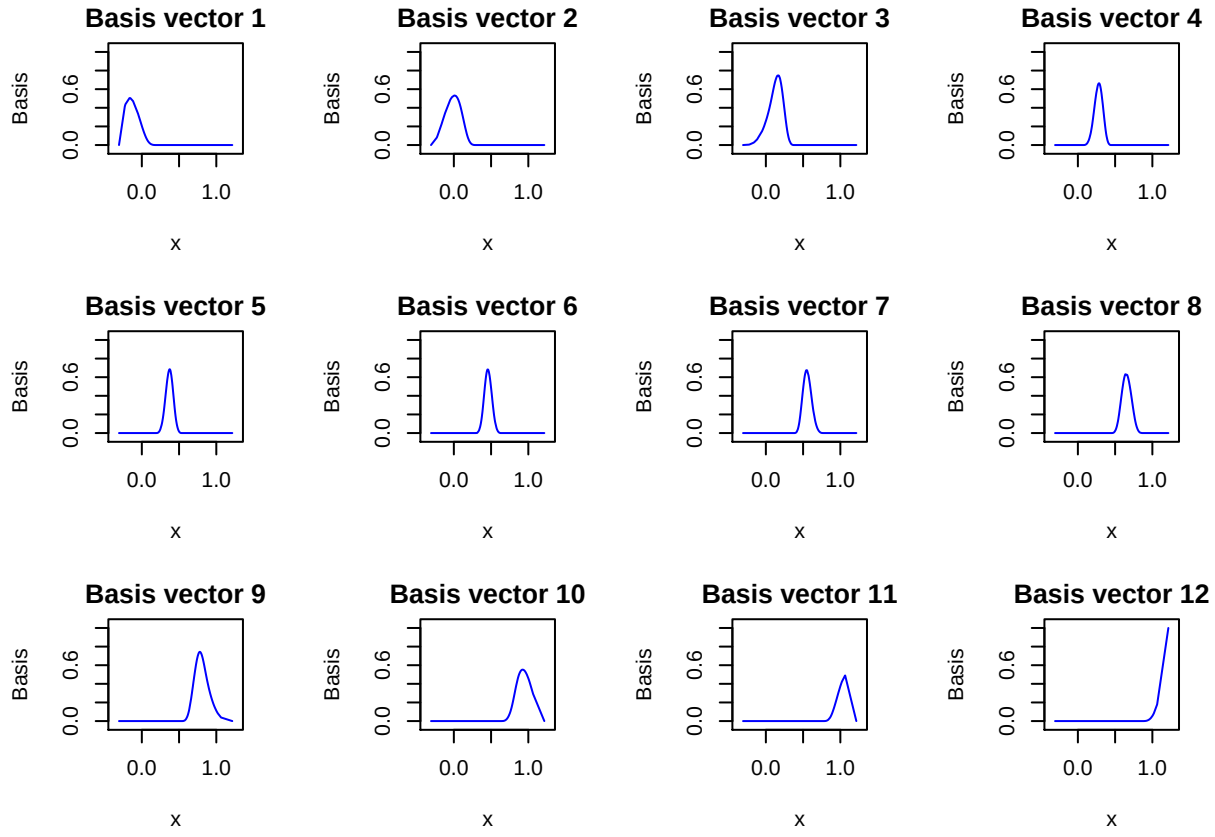
```
#
# xordered <- sort(x)
# plot(xordered, mu(xordered), type="l", lwd=2, lty=2)
#
# generate some ys
#
y <- mu(x) + rnorm(N, 0, .2)
SimulatedData = data.frame(x=x,y=y)

plot(x,y,
     pch=19, cex=1,
     main = "Fake Data",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
knots_p <- quantile(x, seq(0.1, 0.9, 0.1))
abline(v=knots_p , lty=2 )
```




```
# B-Splines for Fake Data
library(splines)
p <- 3
# Note that the knots here are the interior knots
# To match our previous fits, we might choose
Xmat <- bs(x, degree= p, knots=knots_p)
#
# This will be an N by (p + length(knots_p)) matrix
# the first few rows of which are
head(Xmat)
```

```
##      1            2            3         4           5         6          7
## [1,] 0 0.000000e+00 0.0008621395 0.2181317 0.671703715 0.1093024 0.00000000
## [2,] 0 0.000000e+00 0.0000000000 0.0000000 0.000000000 0.0000000 0.00000000
## [3,] 0 0.000000e+00 0.0000000000 0.0000000 0.000000000 0.0000000 0.13117816
## [4,] 0 0.000000e+00 0.0000000000 0.0000000 0.000000000 0.0000000 0.05366229
## [5,] 0 6.548316e-10 0.1361537090 0.6580432 0.205803081 0.0000000 0.00000000
## [6,] 0 3.487194e-02 0.6124390694 0.3499475 0.002741471 0.0000000 0.00000000
##               8         9           10           11 12
## [1,] 0.00000000 0.0000000 0.0000000000 0.000000000  0
## [2,] 0.02492778 0.6901975 0.2816912550 0.003183454  0
## [3,] 0.62430012 0.2435688 0.0009528792 0.000000000  0
## [4,] 0.53224850 0.4052864 0.0088027686 0.000000000  0
## [5,] 0.00000000 0.0000000 0.0000000000 0.000000000  0
## [6,] 0.00000000 0.0000000 0.0000000000 0.000000000  0
```

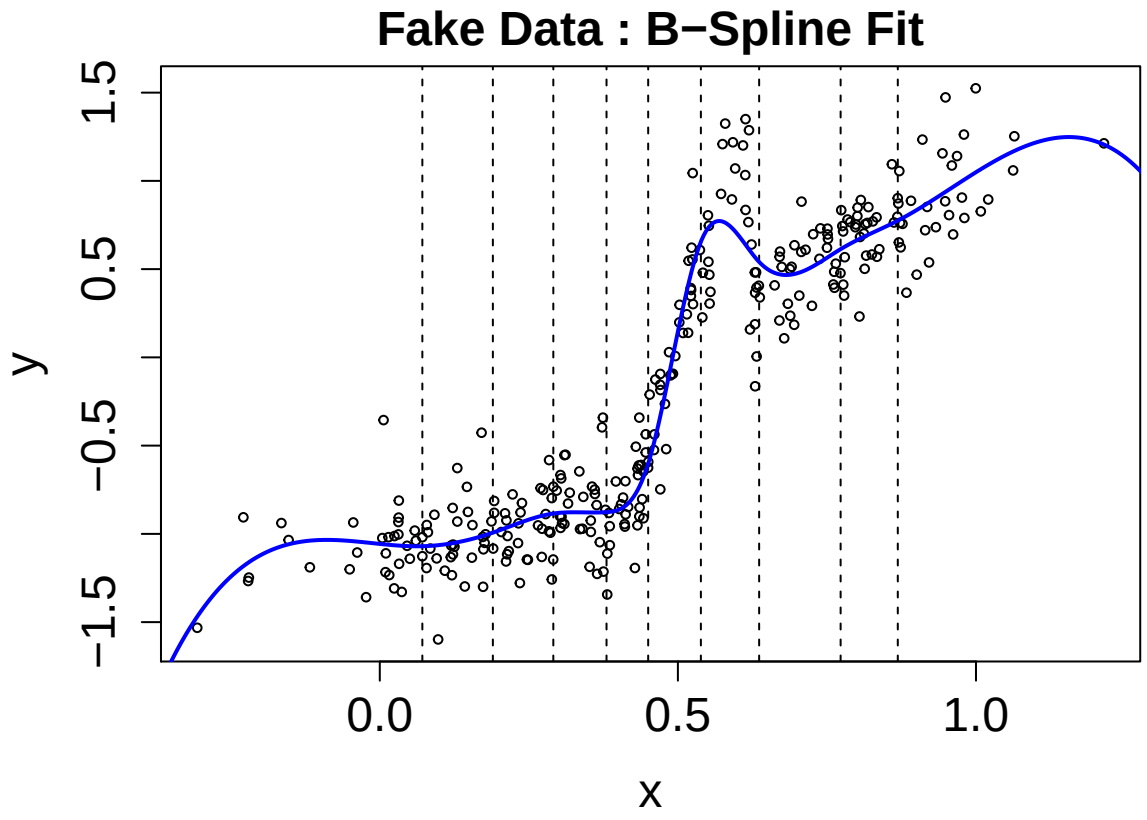
```r
Xorder <- order(x)
blim <- extendrange(Xmat)
parOptions <- par(mfrow = c(3,4))
for (j in 1:ncol(Xmat)) {
  plot(x[Xorder], Xmat[Xorder,j],
       type="l",
       ylim=blim,
       xlim = extendrange(x),
       xlab="x", ylab="Basis",
       main=paste("Basis vector", j),
       col="blue")
}
```



```r
# Fit of the B-spline model
fit.bs <- lm(y ~ bs(x, degree= p, knots=knots_p))
# Let's get predictions at a lot of x-values equi-spaced
# across the range of x so that we can see how smooth
# the fit is
xrange <- extendrange(x)
xnew <- seq(min(xrange), max(xrange), length.out=500)
ypred.bs <- predict(fit.bs, newdata= data.frame(x=xnew))
```

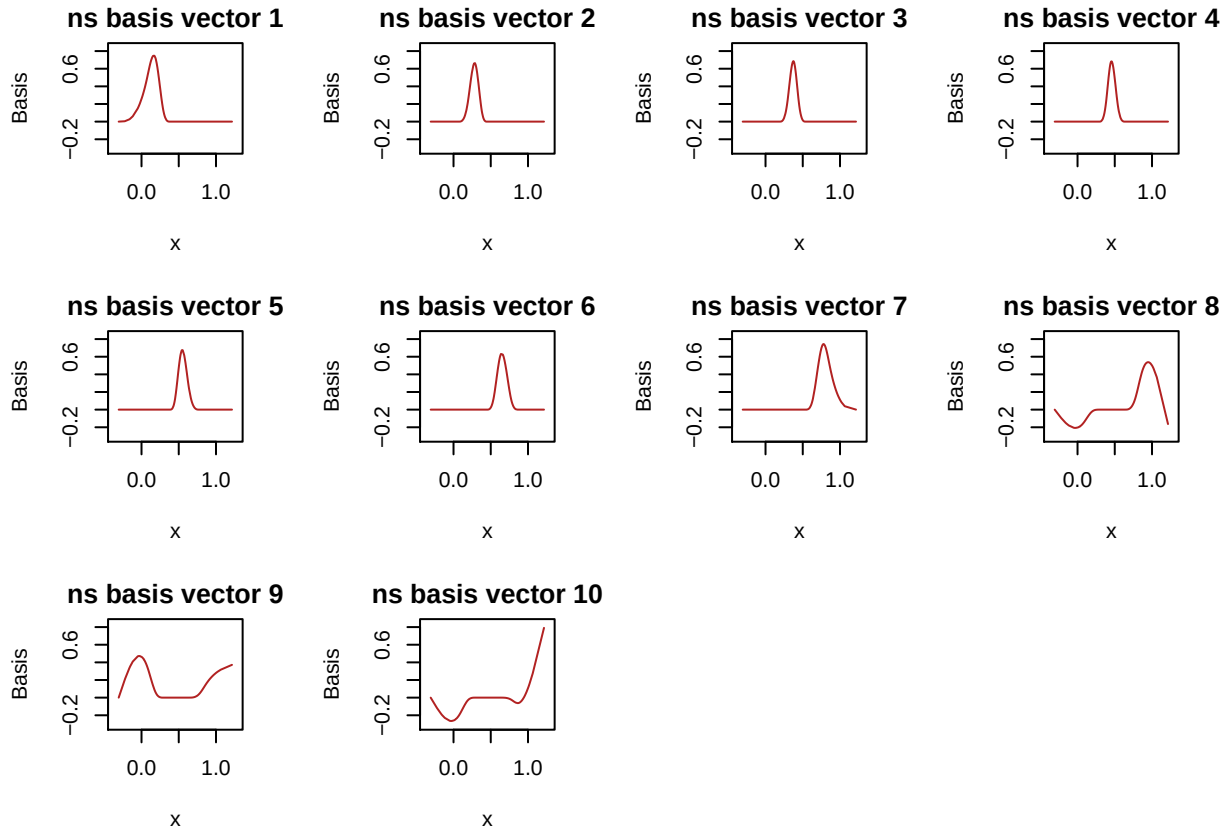
```
## Warning in bs(x, degree = 3L, knots = c(`10%` = 0.0713895502733067, `20%`
## = 0.189663972984999, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```r
par(mfrow=c(1,1))
plot(x,y,
     pch=1, cex=0.6,
     main = "Fake Data : B-Spline Fit",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
)
abline(v=knots_p , lty=2 )
lines(xnew, ypred.bs, col="blue", lwd=2)
```

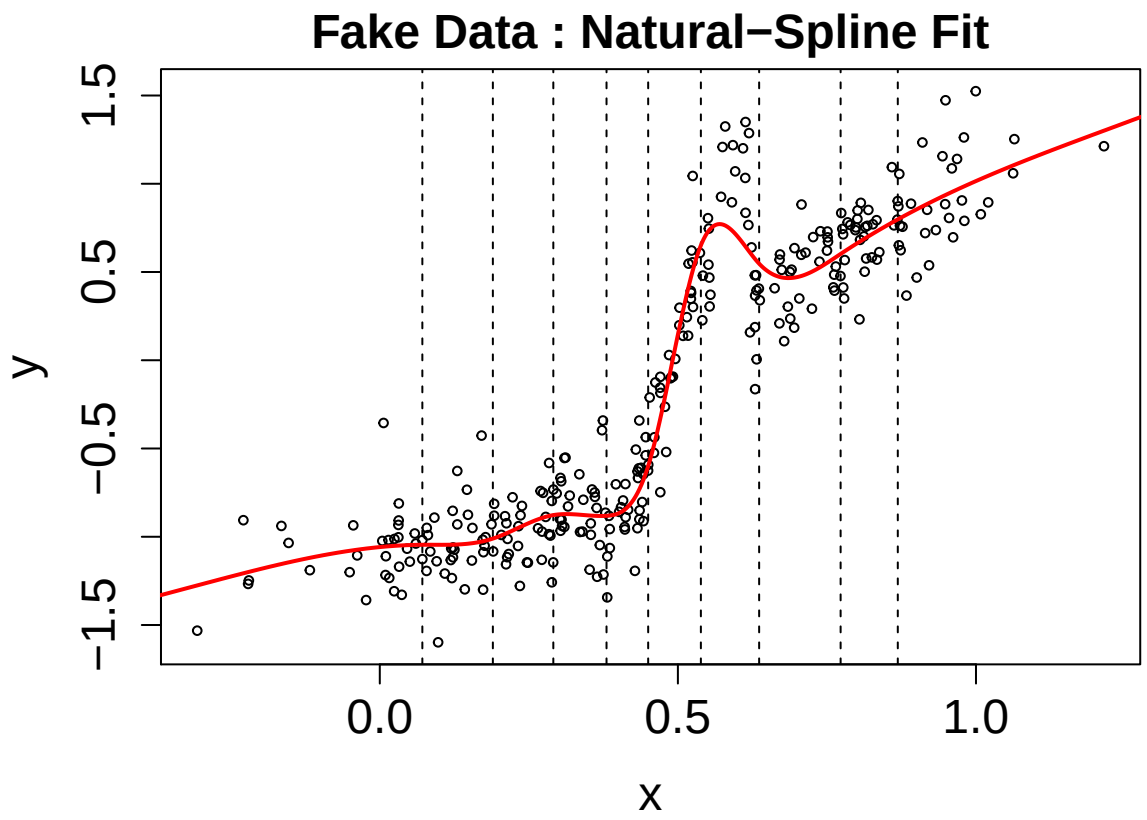



```r
# Natural Splines for Fake Data
par(mfrow = c(3,4))
Xmat.ns <- ns(x, knots=knots_p)
blim <- extendrange(Xmat.ns)
parOptions <- par(mfrow = c(3,4))
for (j in 1:ncol(Xmat.ns)) {
  plot(x[Xorder], Xmat.ns[Xorder,j],
```

```
        type="l",
        ylim=blim,
        xlim = extendrange(x),
        xlab="x", ylab="Basis",
        main=paste("ns basis vector", j),
        col="firebrick")
}

# Fit of the natural spline model
par(mfrow=c(1,1))
```




```
plot(x,y,
     pch=1, cex=0.6,
     main = "Fake Data : Natural-Spline Fit",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
)
abline(v=knots_p , lty=2 )
fit.ns <- lm(y ~ ns(x, knots=knots_p))
ypred.ns <- predict(fit.ns, newdata= data.frame(x=xnew))
lines(xnew, ypred.ns, col="red",lwd=2)
```




```
# Varying degrees of freedom
fit1 <- lm(y ~ bs(x, degree= 3, df=4))
fit2 <- lm(y ~ bs(x, degree= 3, df=5))
fit3 <- lm(y ~ bs(x, degree= 3, df=8))
ypred1 <- predict(fit1,newdata=data.frame(x=xnew))
```

```
## Warning in bs(x, degree = 3L, knots = c(`50%` = 0.450159992790922),
## Boundary.knots = c(-0.306182674194088, : some 'x' values beyond boundary knots
## may cause ill-conditioned bases
```

```
ypred2 <- predict(fit2,newdata=data.frame(x=xnew))
```

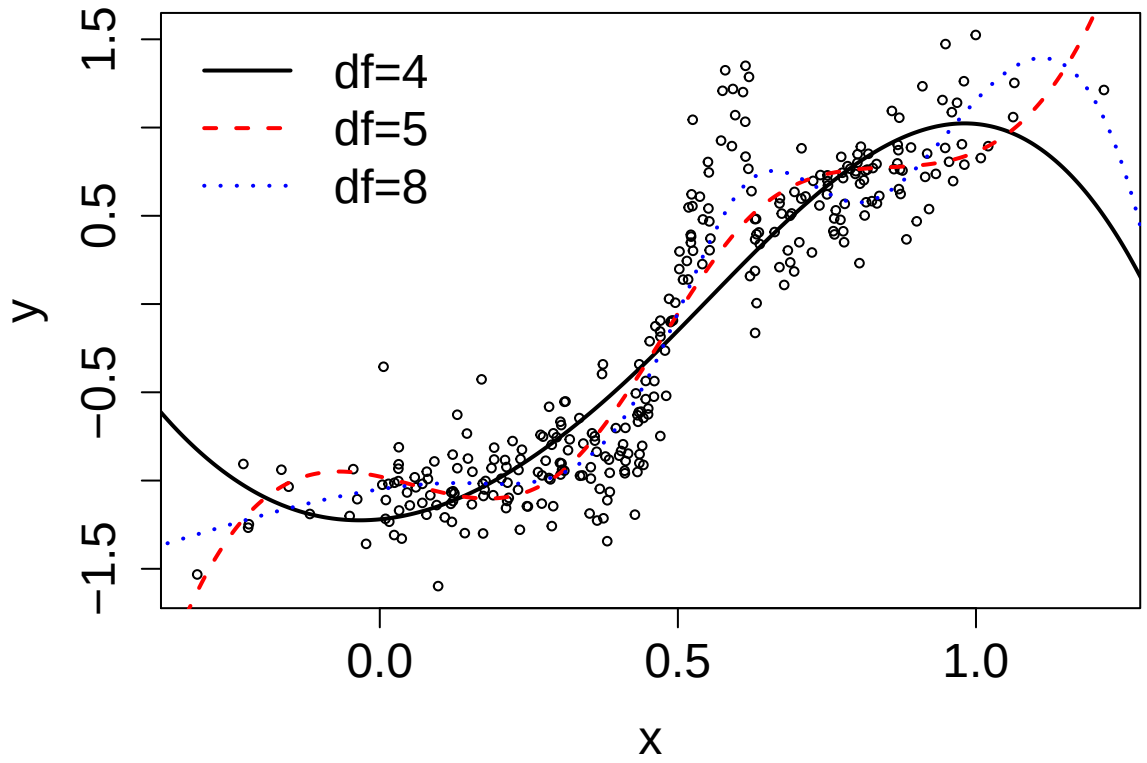
```
## Warning in bs(x, degree = 3L, knots = c(`33.33333%` = 0.310955584187616, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
ypred3 <- predict(fit3,newdata=data.frame(x=xnew))
```

```
## Warning in bs(x, degree = 3L, knots = c(`16.66667%` = 0.147590853545504, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases
```

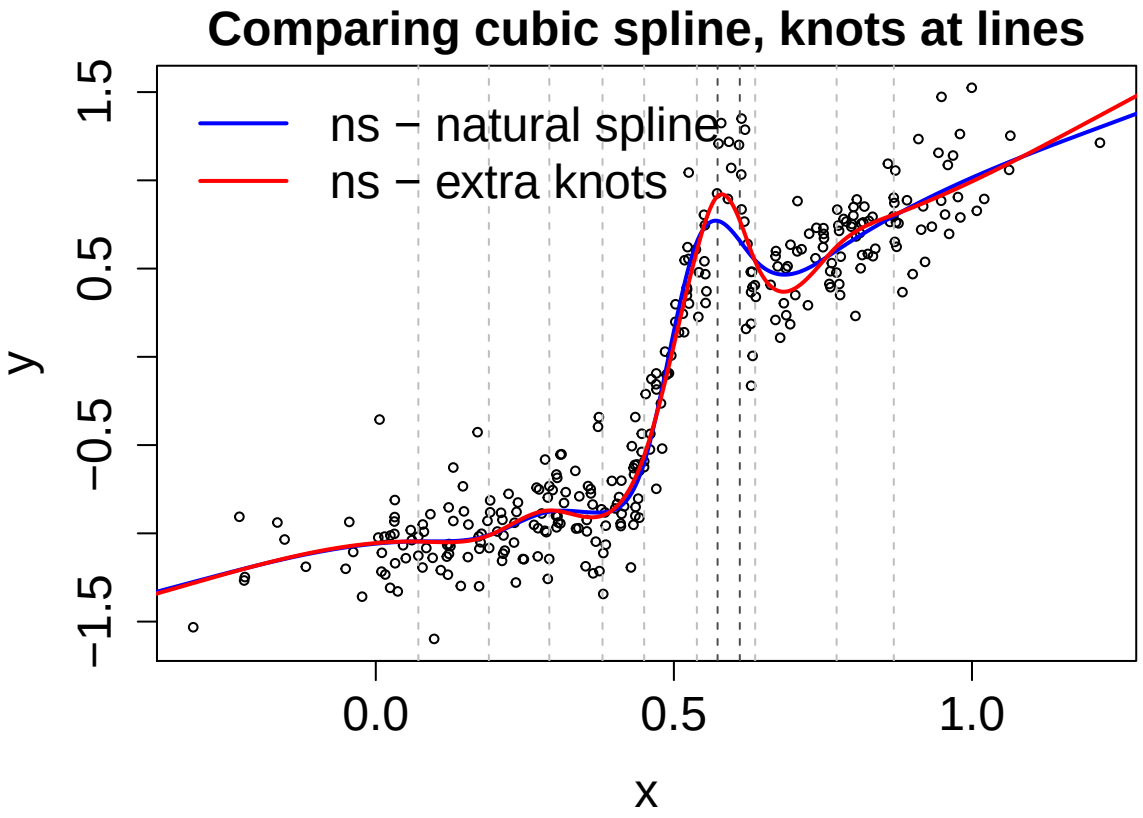
```
plot(x,y,
     pch=1, cex=0.6,
     #main = "Fake Data : B-Spline Fit",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
)
lines(xnew, ypred1, col="black", lwd=2, lty=1)
lines(xnew, ypred2, col="red", lwd=2, lty=2)
lines(xnew, ypred3, col="blue", lwd=2, lty=3)

legend("topleft", bty="n", cex=1.5,
```

```
       legend=c("df=4","df=5", "df=8"),
       lty=c(1,2,3), lwd=2,
       col=c("black", "red", "blue")
)
```




```
# More knots where there is variability
knots_p2 <- quantile(x,
                     c(seq(0.1,0.6,0.1),
                       0.63, 0.65,
                       seq(0.7,0.9,0.1)
                     )
)

fit.ns2 <- lm(y ~ ns(x, knots=knots_p2))

# And now the predicted values for plotting
ypred.ns2 <- predict(fit.ns2, newdata= data.frame(x=xnew))


plot(x,y,
     pch=1, cex=0.6,
     main = "Comparing cubic spline, knots at lines",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
)
abline(v=knots_p, col ="grey", lty=2)
abline(v=knots_p2[c(7,8)], col ="grey30", lty=2)
lines(xnew, ypred.ns, col="blue", lwd=2, lty=1)
lines(xnew, ypred.ns2, col="red", lwd=2, lty=1)
legend("topleft", bty="n", cex=1.5,
       legend=c("ns - natural spline","ns - extra knots"),
```

```
       lty=c(1,1), lwd=2,
       col=c("blue", "red")
)
```



5) Regularization Methods : Ridge, LASSO, and Elastic Net (Slides 62–67)

```
house.price <- read.table(file="HouseData.txt", header=T)
head(house.price)
```

```
##      X1 X2    X3    X4 X5 X6 X7 X8 X9    Y
## 1 4.918  1 3.472 0.998  1  7  4 42  0 25.9
## 2 5.021  1 3.531 1.500  2  7  4 62  0 29.5
## 3 4.543  1 2.275 1.175  1  6  3 40  0 27.9
## 4 4.557  1 4.050 1.232  1  6  3 54  0 25.9
## 5 5.060  1 4.455 1.121  1  6  3 42  0 29.9
## 6 3.891  1 4.455 0.988  1  6  3 56  0 29.9
```

```
dim(house.price)
```

```
## [1] 24 10
```

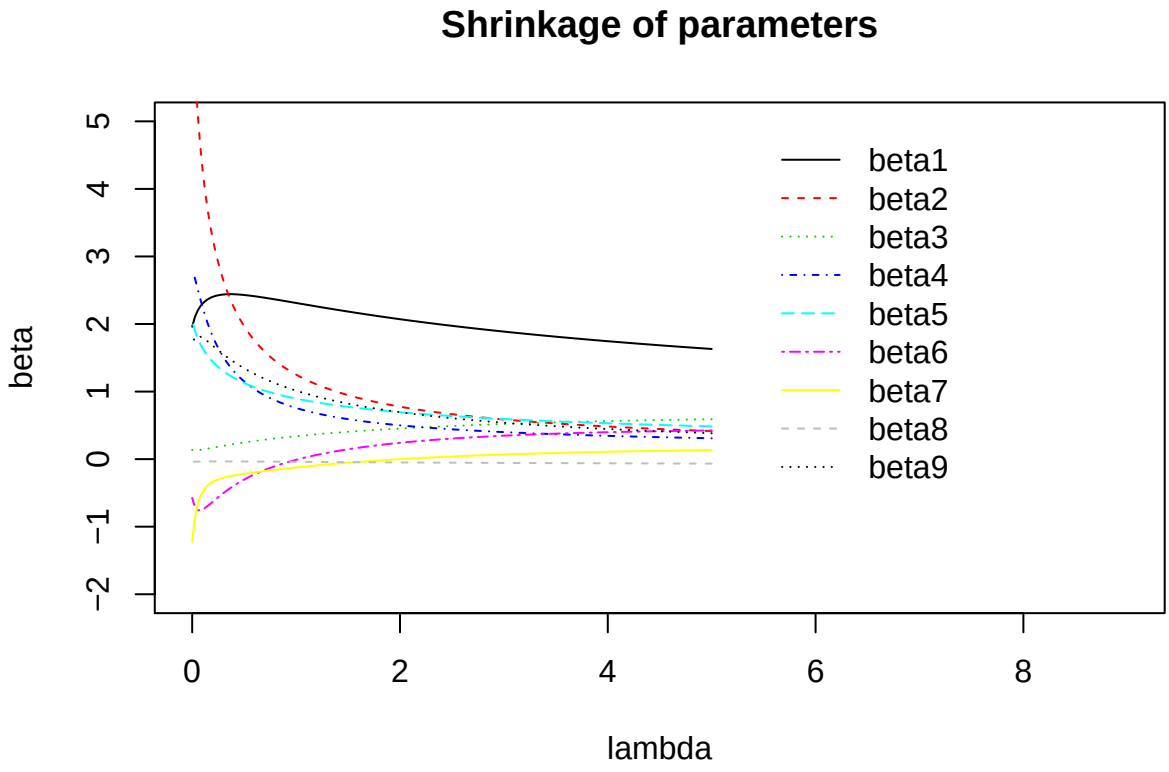
Ridge Regression

```
attach(house.price)
Y = scale(house.price$Y , center = TRUE, scale = FALSE) #centralizing the response
X=cbind(X1 , X2 , X3 , X4 , X5 , X6 , X7 , X8 , X9)
X=apply(X,2,scale,TRUE,FALSE) #centralizing the columns of X
library(glmnet)
```

```
## Loading required package: Matrix
```

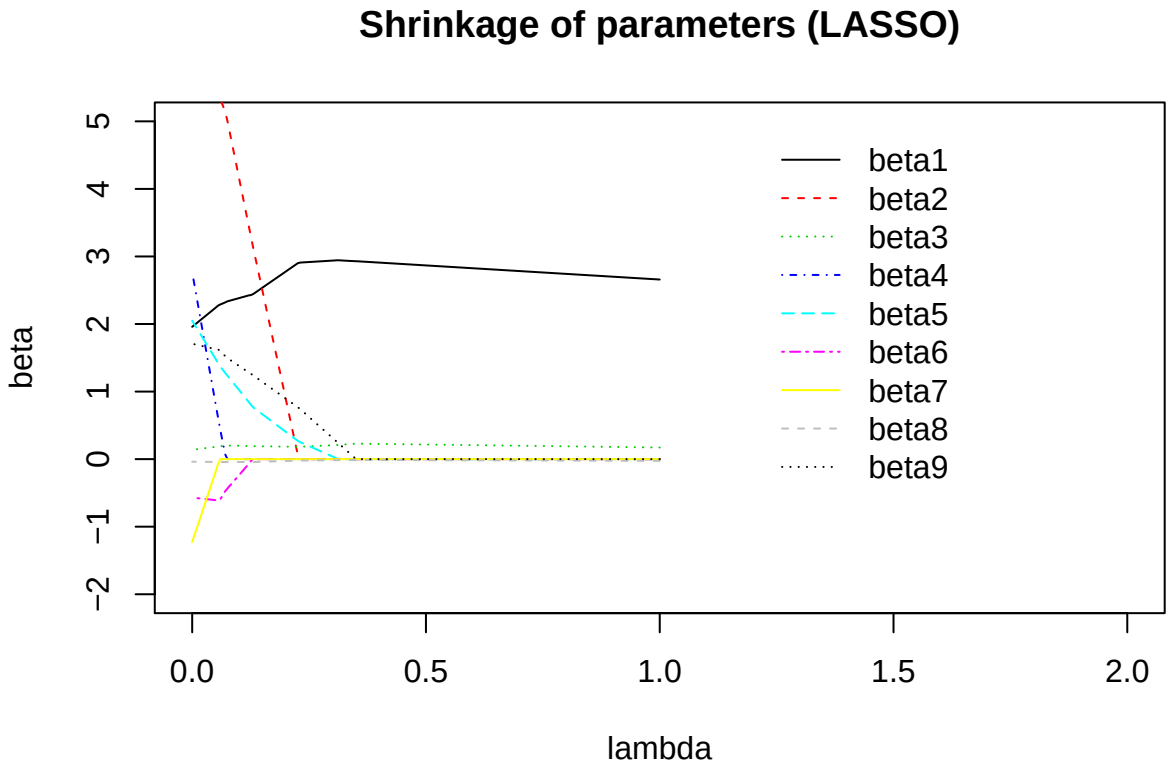
```
## Loaded glmnet 3.0-2
```

```
# fitting Ridge regression
fit.ridge=glmnet(X ,Y , alpha=0 , lambda=seq(0,5, length =200),
                 standardize=FALSE, intercept = TRUE ,  family = "gaussian")

plot(1:15, main="Shrinkage of parameters",xlab="lambda", ylab="beta", xlim=c(0,9),
     ylim=c(-2,5), pch=" ")

for(i in 2:10){
  lines(fit.ridge$lambda,coef(fit.ridge)[i,], lty=(i-1), col=(i-1))
}

legend(5.4,5, legend=list("beta1","beta2","beta3","beta4","beta5",
                          "beta6","beta7","beta8","beta9"),  lty=1:9, col=1:9,
       box.col = "white")
```



LASSO

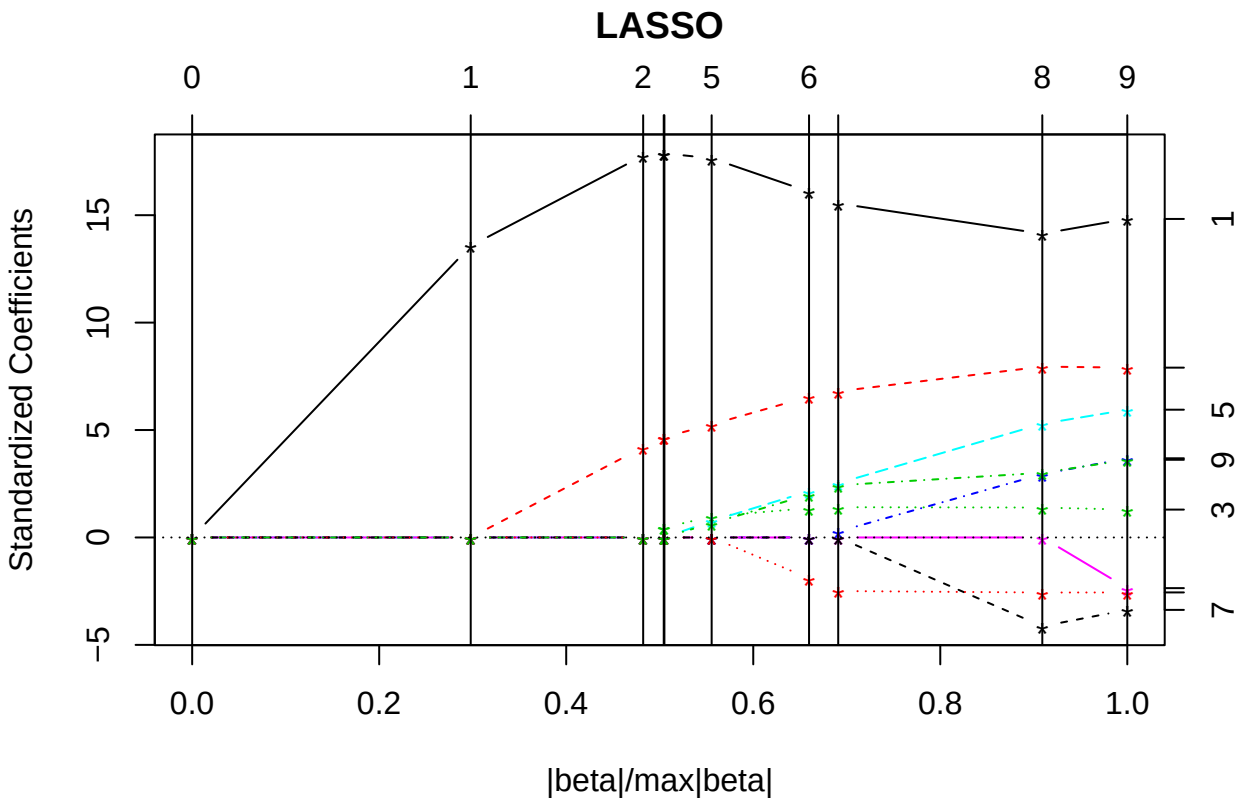
```
# Fitting the Model

fit.Lasso=glmnet(X ,  Y , alpha=1, lambda=seq(0,1, length =200),
                 standardize=FALSE, intercept = TRUE ,  family = "gaussian")

plot(1:15, main="Shrinkage of parameters (LASSO)" , xlab="lambda" ,  ylab="beta" ,
     xlim=c(0,2) , ylim=c(-2,5), pch=" ")

for(i in 2:10){
  lines(fit.Lasso$lambda,coef(fit.Lasso)[i,], lty=(i-1), col=(i-1))
}

legend(1.2,5, legend=list("beta1" , "beta2" , "beta3" , "beta4" ,
                          "beta5" , "beta6" , "beta7" , "beta8" , "beta9"),
       lty=1:9, col=1:9, box.col = "white")
```



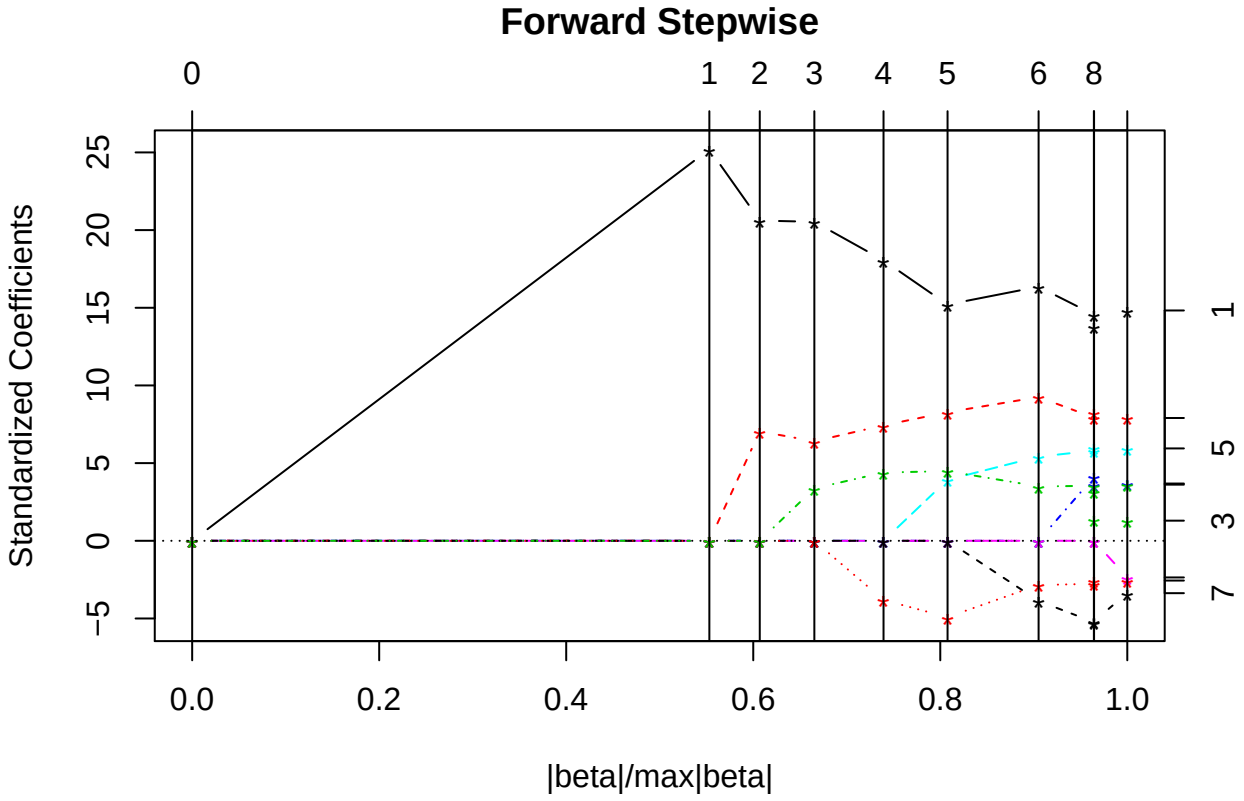

```
# LASSO : variable selection)

library(lars)
```

```
## Loaded lars 1.2
```

```
house.lasso = lars(X ,  Y, type="lasso")

plot(house.lasso)
```




```
house.step = lars(X ,  Y , type="step")

plot(house.step)
```



Cross-Validation

```
set.seed(123)
indx = sample(1:24,20)
X.modeling = X[indx,] #setting 20 datapoints for modelling
Y.modeling = Y[indx]
X.testing = X[-indx,] #setting the remaining 4 datapoints for testing
Y.testing = Y[-indx]
cv.lasso <- cv.glmnet(X.modeling , Y.modeling , alpha=1, nfolds = 5) # 5-fold cross validation
cv.lasso # cross-validation results
```

```
##
## Call:  cv.glmnet(x = X.modeling, y = Y.modeling, nfolds = 5, alpha = 1)
##
## Measure: Mean-Squared Error
##
##     Lambda Measure    SE Nonzero
## min 0.8972   13.54 2.973       4
## 1se 1.8885   16.35 3.154       3
```

```
cv.lasso$lambda.min # lambda with the minimum cross-validated result
```

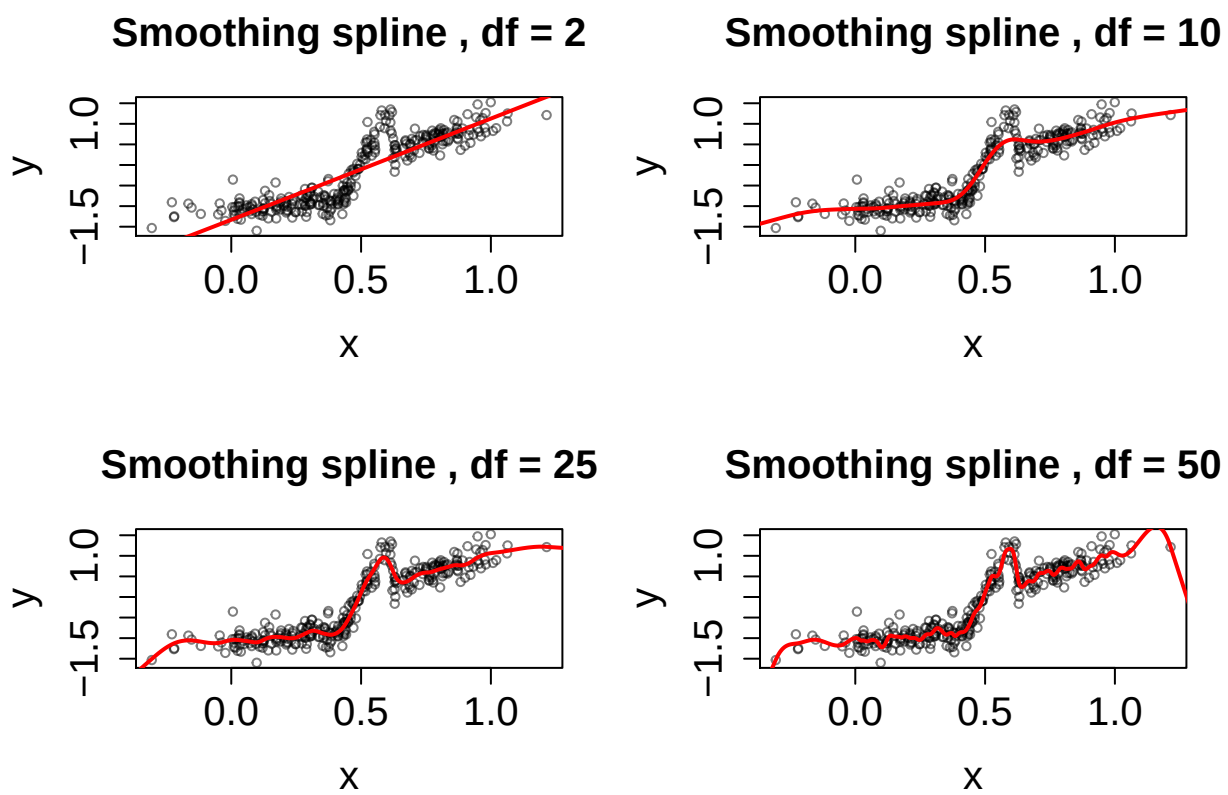
```
## [1] 0.897168
```

```
fit.lasso.CrossValidated=glmnet(X.modeling , Y.modeling , alpha=1 ,
    lambda=cv.lasso$lambda.min) # fitting the optimum model
lasso.pred.CrossValidated=predict(fit.lasso.CrossValidated,
    newx=X.testing) #predicting the test data

#calculating the prediction error
lasso.pred.CrossValidated.SSE = sum((Y.testing-lasso.pred.CrossValidated)^2)
lasso.pred.CrossValidated.SSE
```

```
## [1] 100.1323
```

6) Smoothing Splines (Slides 83–150)

```r
# Fake data example (see the corresponding x and y from earlier examples/codes)

## Fitting smoothing splines with given effective degress of freedom (here 10)
df <- c(2,10,25,50)
par(mfrow=c(2,2))
for(i in 1:4){
  sm <- smooth.spline(x, y, df = df[i])

  # plotting the resutls
  xrange <- extendrange(x)
  xnew <- seq(min(xrange), max(xrange), length.out=500)
  ypred.sm <- predict(sm, x=xnew)$y # fitted values
  plot(x,y,
       pch=1, cex=0.7,
       main = paste("Smoothing spline , df =", df[i]),
       cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
       col=adjustcolor("black" , alpha=0.5)
  )
  lines(xnew, ypred.sm, col="red", lwd=2)

}
```




```r
## Leave-obe-out cross-validation
sm.LOOCV <- smooth.spline(x, y, cv = TRUE)
sm.LOOCV
```

```
## Call:
## smooth.spline(x = x, y = y, cv = TRUE)
##
## Smoothing Parameter  spar= 0.6402688  lambda= 7.07313e-06 (14 iterations)
## Equivalent Degrees of Freedom (Df): 25.82318
## Penalized Criterion (RSS): 12.62976
## PRESS(l.o.o. CV): 54.50422
```

```
sm.LOOCV$lambda
```

```
## [1] 7.07313e-06
```

```
sm.LOOCV$df
```

```
## [1] 25.82318
```

```
## Generalized cross-validation
sm.GCV <- smooth.spline(x, y, cv = FALSE)
sm.GCV
```

```
## Call:
## smooth.spline(x = x, y = y, cv = FALSE)
##
## Smoothing Parameter  spar= 0.5603612  lambda= 1.87198e-06 (11 iterations)
## Equivalent Degrees of Freedom (Df): 33.9024
## Penalized Criterion (RSS): 11.7393
## GCV: 1.65062
```

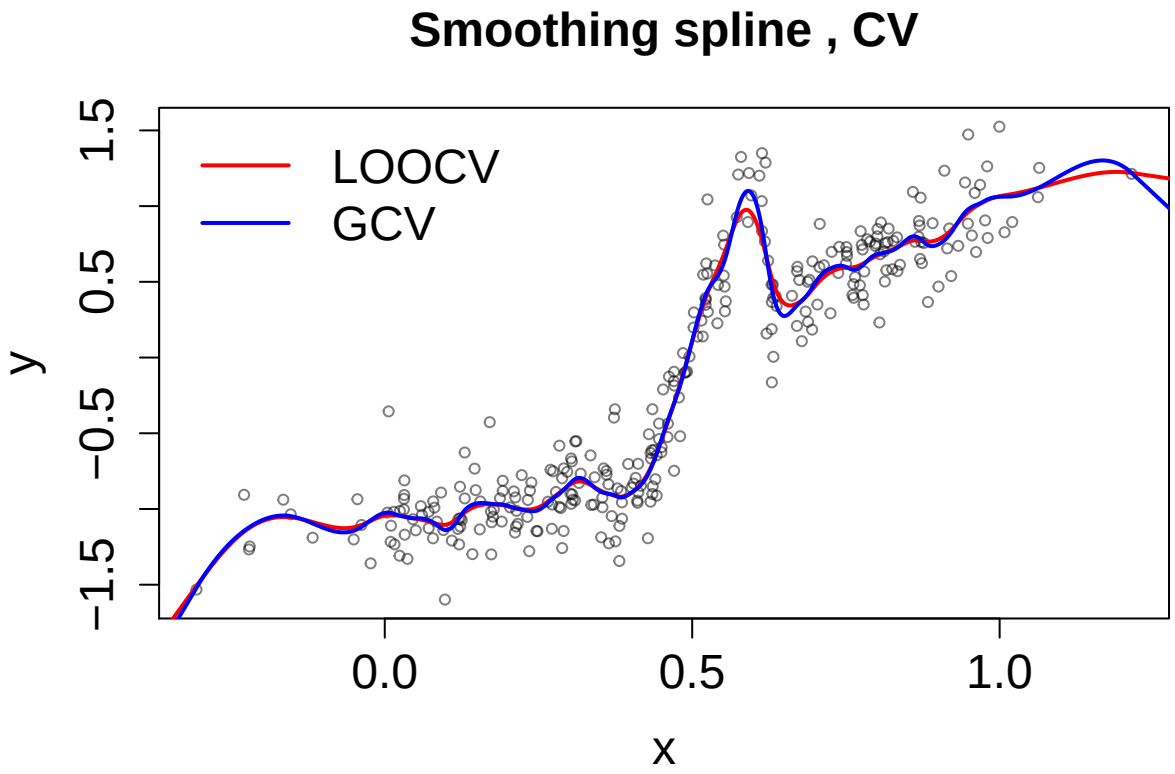
```
sm.GCV$lambda
```

```
## [1] 1.87198e-06
```

```
sm.GCV$df
```

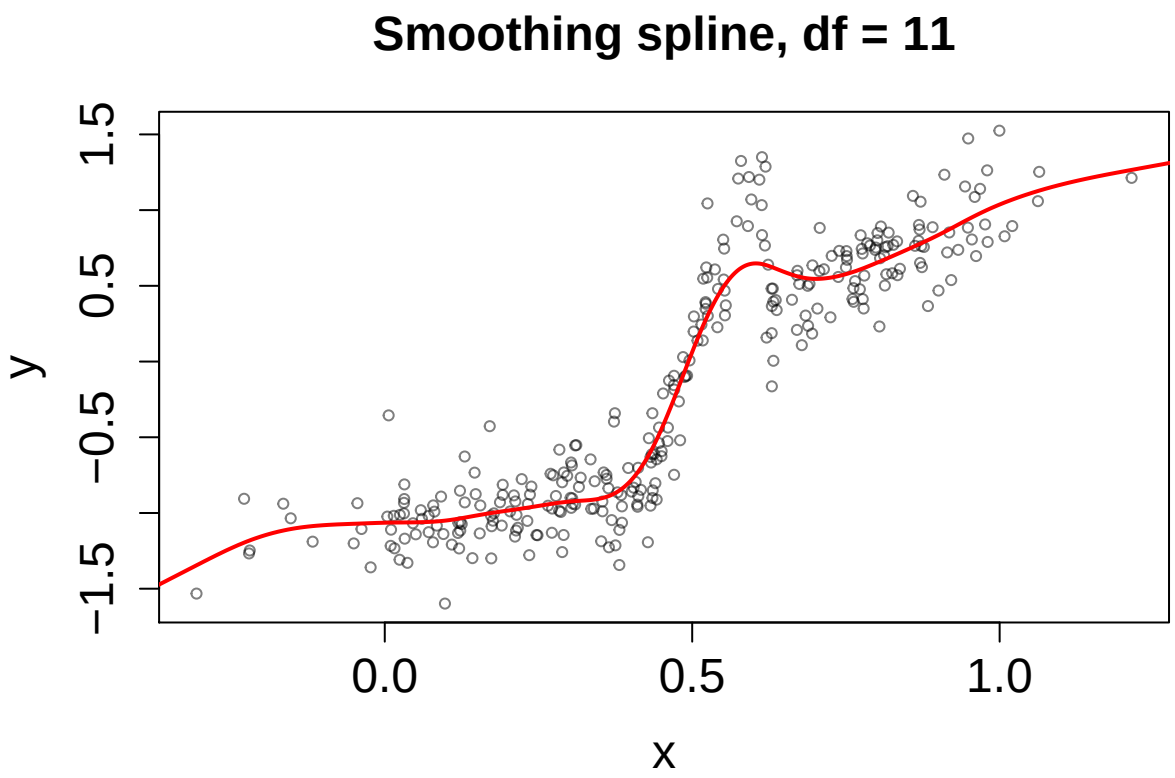
```
## [1] 33.9024
```

```
# ploting LOOCV and GCV in one plot
par(mfrow=c(1,1))
plot(x,y,
     pch=1, cex=0.7,
     main = "Smoothing spline , CV",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
ypred.sm.LOOCV <- predict(sm.LOOCV, x=xnew)$y
lines(xnew, ypred.sm.LOOCV, col="red", lwd=2)
ypred.sm.GCV <- predict(sm.GCV, x=xnew)$y
lines(xnew, ypred.sm.GCV, col="blue", lwd=2)

legend("topleft",  bty='n', cex=1.5,
       legend = c("LOOCV", "GCV"),
       col = c("red", "blue"),
       lty=c(1, 1), lwd=c(2,2),
       text.width = 1.5
)
```



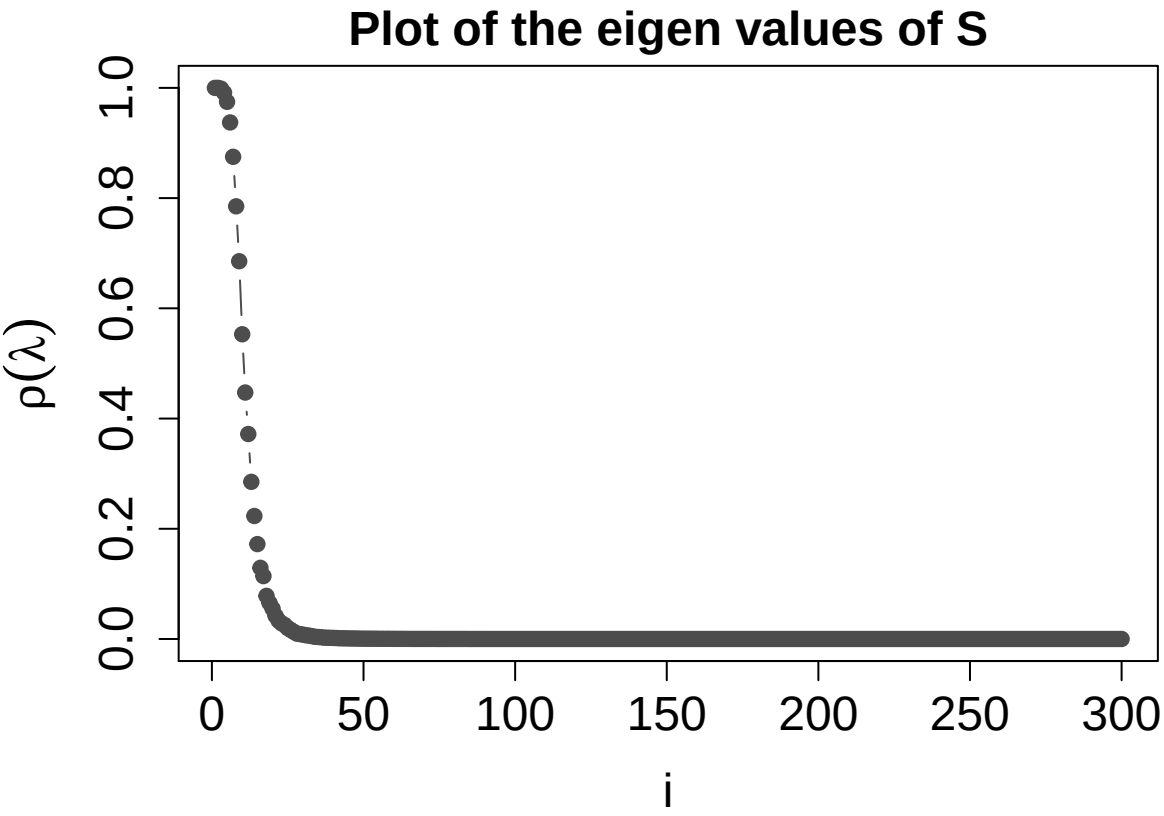
Next, the eigendecomposition of smoothing splines is discssued.

```
df <- 11
sm <- smooth.spline(x, y, df = df)

# plotting the resutls
xrange <- extendrange(x)
xnew <- seq(min(xrange), max(xrange), length.out=500)
ypred.sm <- predict(sm, x=xnew)$y
plot(x,y,
     pch=1, cex=0.7,
     main = paste("Smoothing spline, df =", df),
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
lines(xnew, ypred.sm, col="red", lwd=2)
```



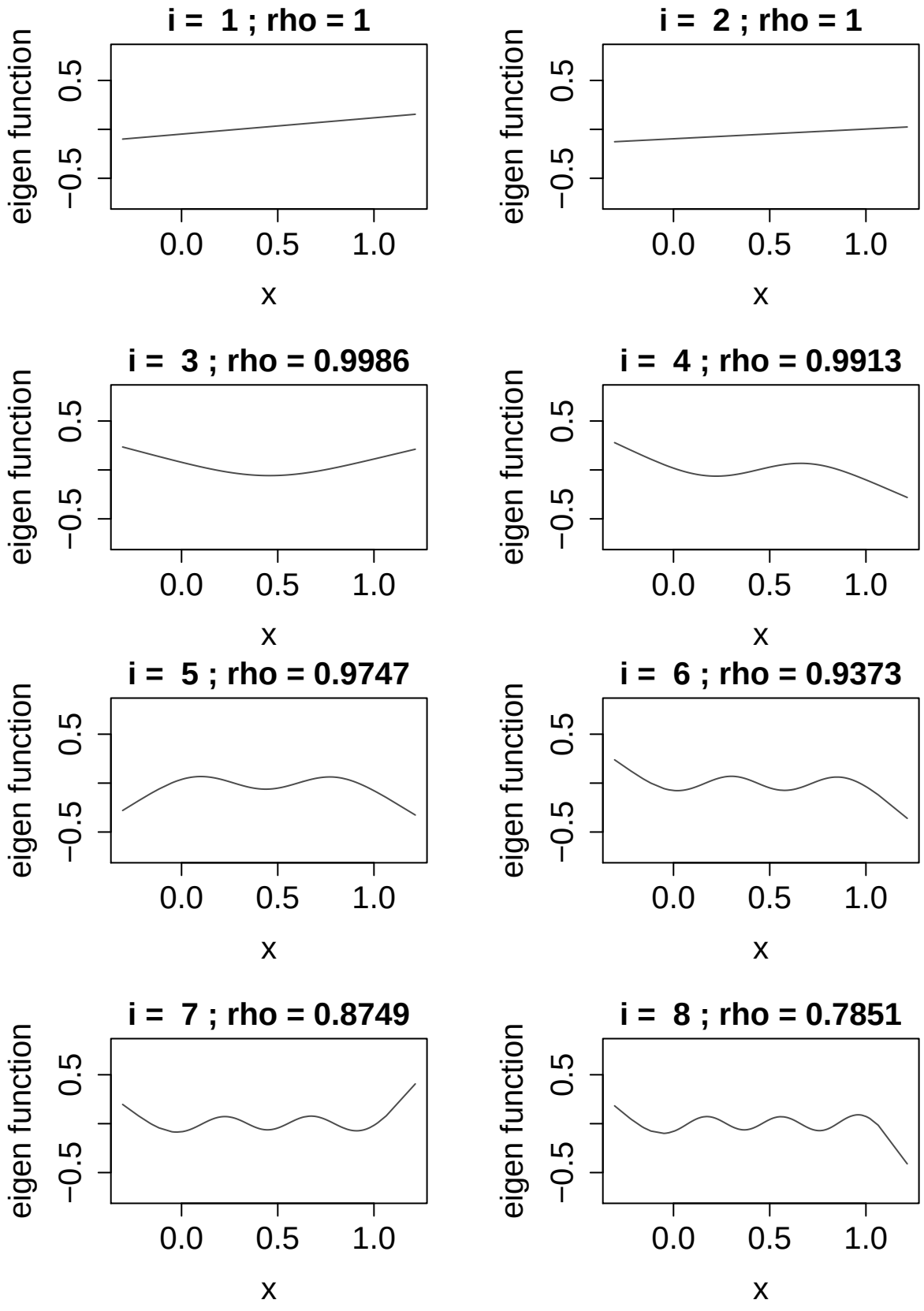

```
## Smoother matrix is not directly available in R.
## We can however access it a little less directly by recognizing
## that we can pick off its ith column by choosing the y vector to
## be fitted to be the vector ei=(0,...,0,1,0,...,0)T which is zero everywhere
## but in its ith coordinate where it is 1.
smootherMatrix <- function(x,df) {
  n <- length(x)
  S <- matrix(0, n, n)
  for (i in 1:n) {
    ei = rep(0, n)
    ei[i] <- 1
    # insert the fit into the i'th column of S
    S[,i]  <- predict(smooth.spline(x, ei, df=df), x)$y
  }
  # To make sure the result is (numerically) symmetric
  S <- (S + t(S))/2
  # and return the symmetric matrix
  S
}

# Now we can get S.lambda and its eigen decompositiotn
# First get S
S <- smootherMatrix(x, 11)

# and then its eigen decomposition
eigS <- eigen(S)

# We can look at its eigen values in a plot
par(mar=c(5,5,2,2))
```

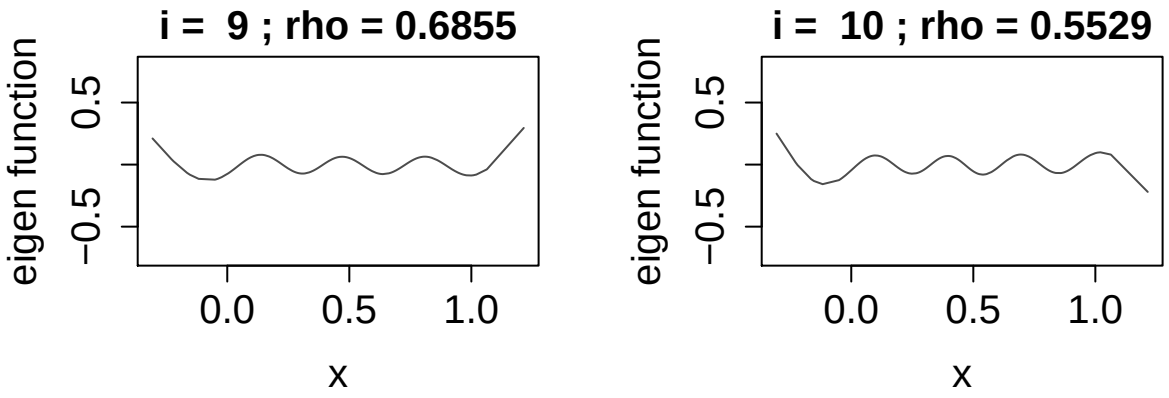
```
plot(eigS$values, type="b",
     main=paste("Plot of the eigen values of S"),
     ylab = expression(rho(lambda)),
     xlab="i",
     pch=19, col="grey30",
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5)
```

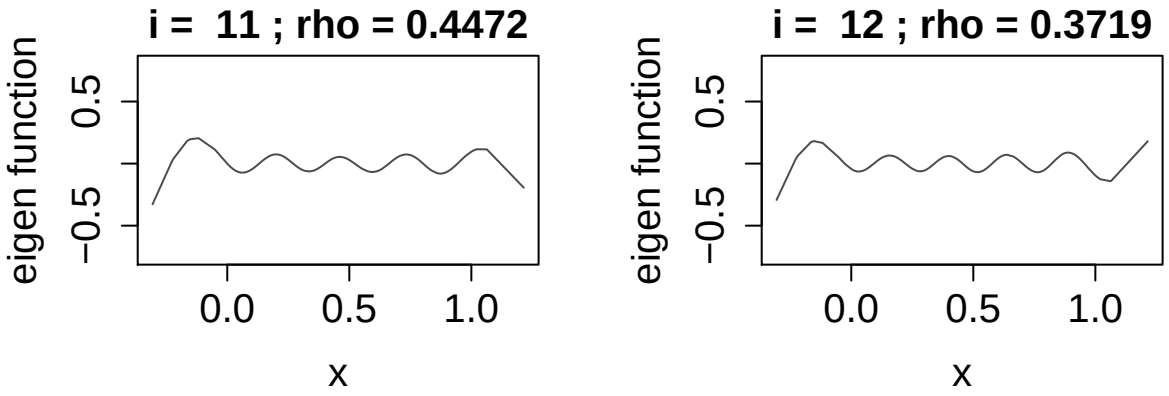



```
# Plotting the eigen-vectors
plotEigenBases <- function(x,
                           eigenDecomp,
                           indices){

  Xorder <- order(x)
  orderedX <- x[Xorder]
  vectors <- eigenDecomp$vectors
  ylim <- range(vectors)
  values <- eigenDecomp$values
  for (i in indices){
    plot(orderedX, vectors[Xorder, i],
         type="l", col=adjustcolor("black", 0.7),
         xlab="x", ylab="eigen function",
         ylim = ylim,
         main=paste("i = ", i, "; rho =", round(values[i],4)),
         cex.axis=1.5 , cex.main=1.5, cex.lab=1.5)
  }
}

parOptions <- par(mfrow=c(2,2))
```

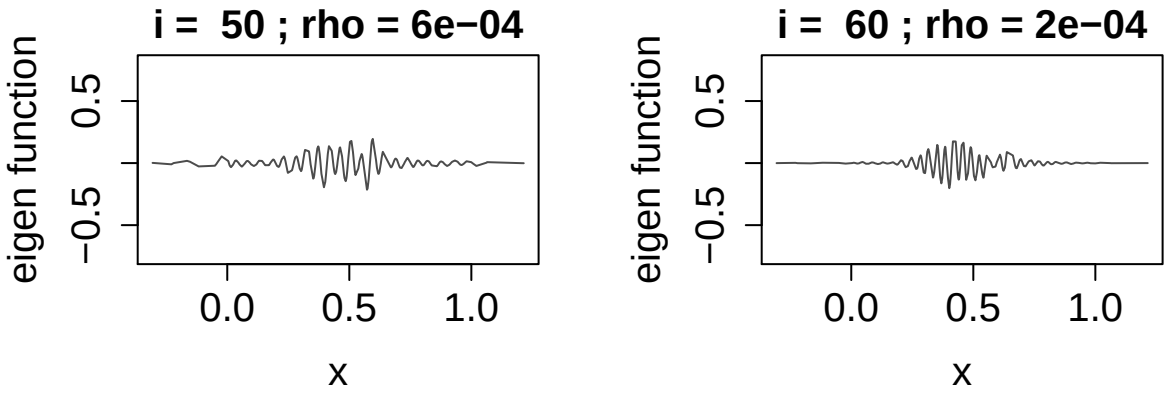
```
plotEigenBases(x, eigS, 1:12)
```

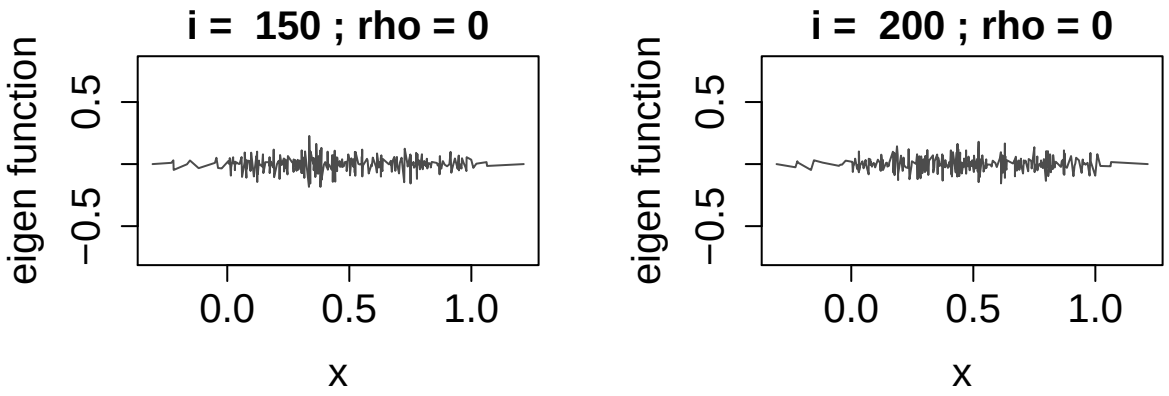







```
plotEigenBases(x, eigS, c(50,60,150,200))
```





7) KNN and Local Weighting (Slides 105–150)

KNN

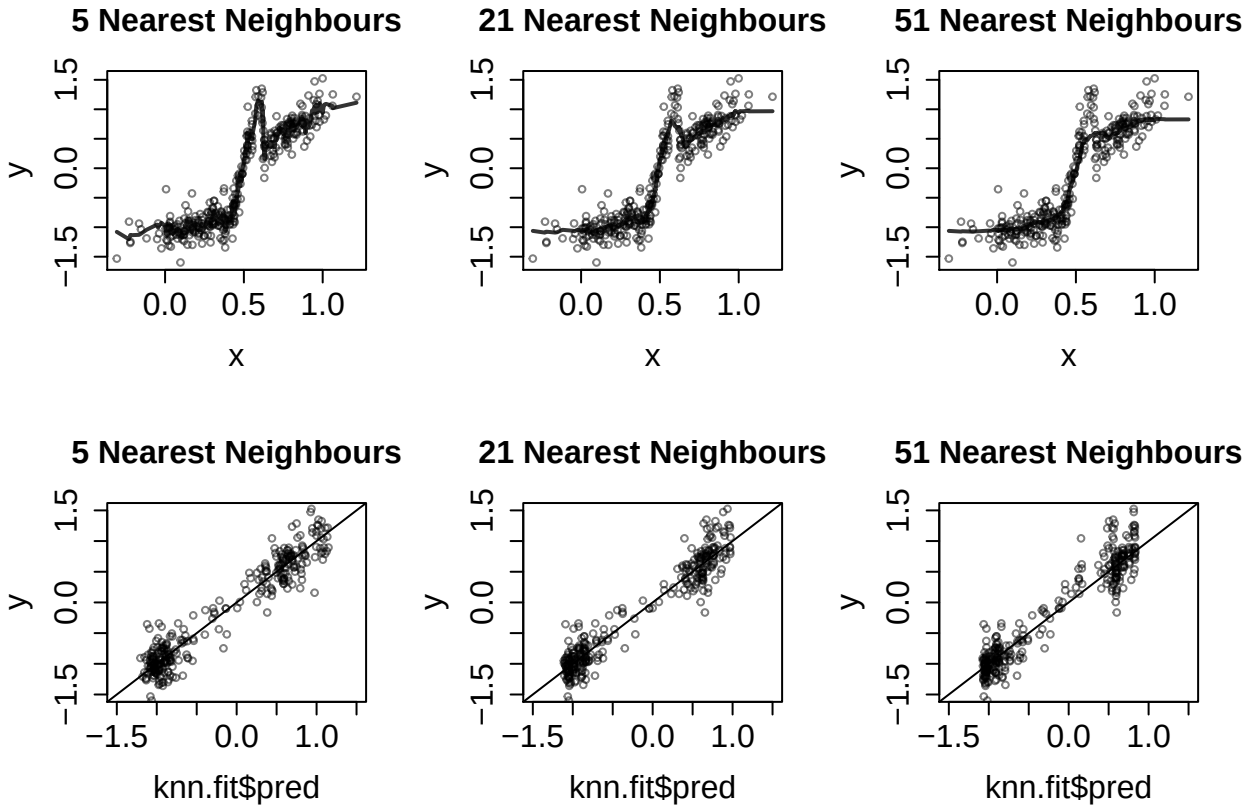
```
layout.matrix <- matrix(c(1, 2, 3, 4 , 5 , 6), nrow = 2, ncol = 3)
layout(mat = layout.matrix,
       heights = c(1, 1 , 1), # Heights of the two rows
       widths = c(1, 1 , 1)) # Widths of the two columns


#### Local model with averaging (degree 0 polynomial)

library(FNN)

# Let's try a few values for k

k = c(5,21,51)
TITLES = c("5 Nearest Neighbours",
                "21 Nearest Neighbours",
                "51 Nearest Neighbours")

for(i in 1:3){
  knn.fit <- knn.reg(x, y=y, k=k[i])
  title.graph = TITLES[i]
  print(TITLES[i])
  print(knn.fit)

  # Actual plotting
  Xorder <- order(x)

  # plotting the data
  xrange <- extendrange(x)
  xnew <- seq(min(xrange), max(xrange), length.out=500)
  plot(x,y,
       pch=1, cex=0.7,
       main = title.graph,
       cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
       col=adjustcolor("black" , alpha=0.5)
  )

  # Adding the fitted model
  lines(x[Xorder], knn.fit$pred[Xorder],
      col=adjustcolor("black", 0.8),
      lwd=2)

  # plotting true y vs. y.hat to check the fit
  plot(knn.fit$pred , y ,
     pch=1, cex=0.7,
     main = title.graph,
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
```

```
     col=adjustcolor("black" , alpha=0.5),
     xlim=c(-1.5,1.5),
     ylim=c(-1.5,1.5))
  abline(a=0,b=1)
}
```

```
## [1] "5 Nearest Neighbours"
## PRESS =  16.2288
## R2-Predict =  0.9213303
```

```
## [1] "21 Nearest Neighbours"
## PRESS =  15.77232
## R2-Predict =  0.9235431
```

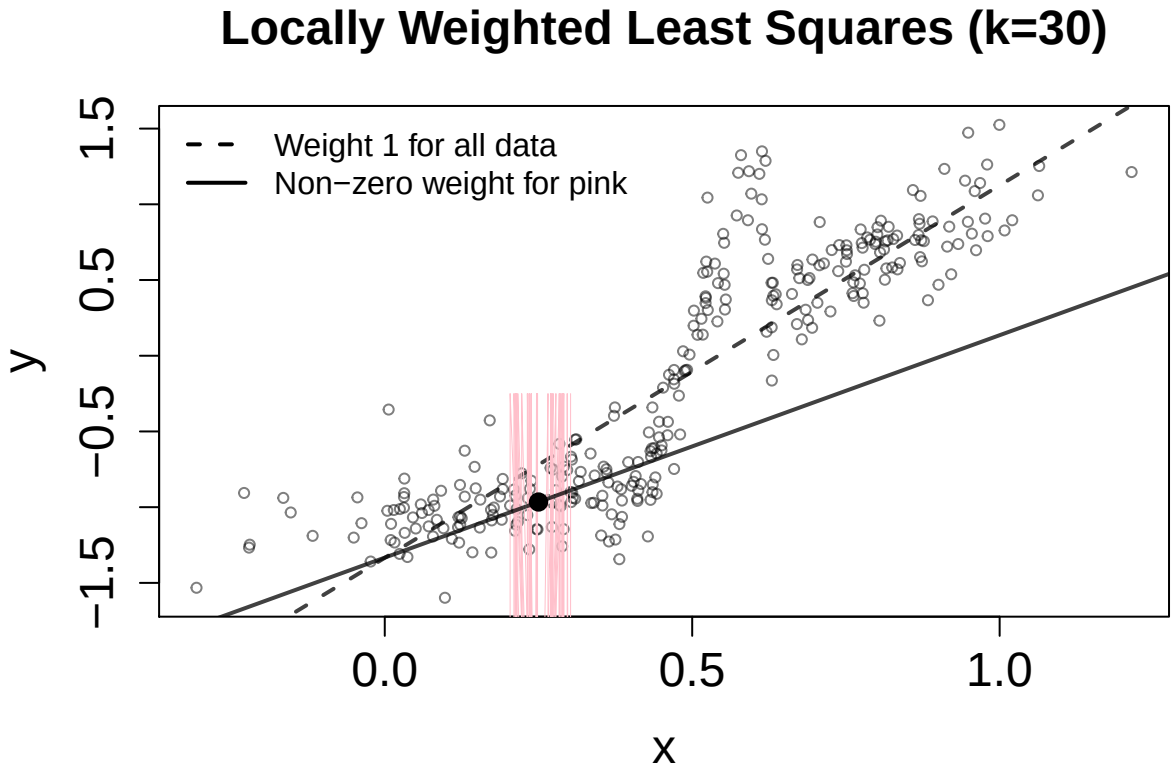
```
## [1] "51 Nearest Neighbours"
## PRESS =  19.45984
## R2-Predict =  0.9056677
```



Local Weighting

```
#  The least-squares fit to all of the data
global.fit <- lm(y~x, data=SimulatedData)
alpha_hat_global <- global.fit$coefficients[1]
beta_hat_global <- global.fit$coefficients[2]

plot(x,y,
     pch=1, cex=0.7,
     main = "Locally Weighted Least Squares (k=30)",
```

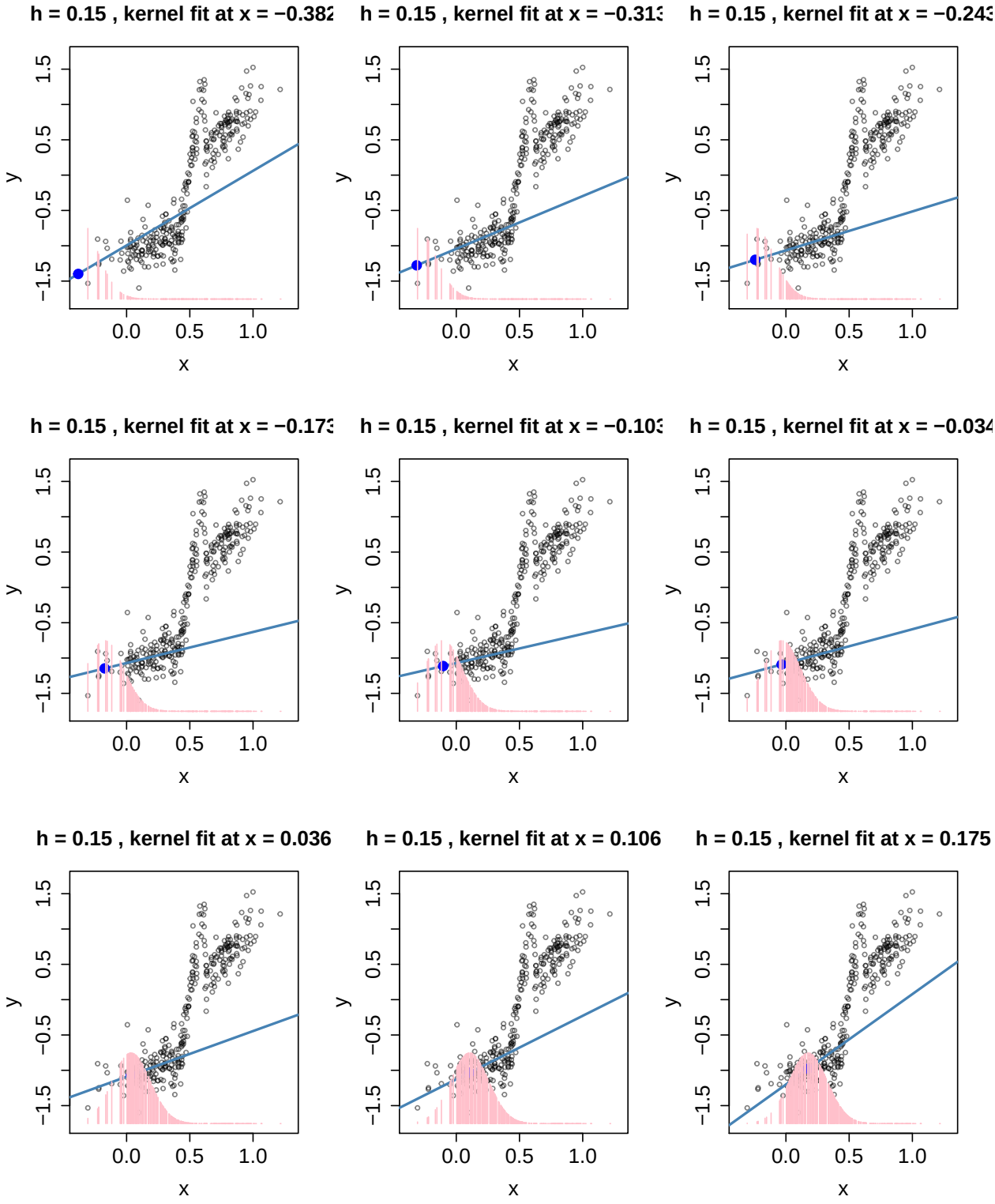
```
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)

abline(a=alpha_hat_global, b=beta_hat_global,
       col=adjustcolor("black", 0.75),
       lty=2, lwd=2)

xloc = 0.25
neighbours <- get.knnx(x, query=xloc, k=30)$nn.index
lines(rep(x[neighbours],each=2),rep(c(-10,-0.25),length(neighbours)),
      col="pink" , lwd=0.5)

local.fit = lm(y~x , data = SimulatedData , subset = neighbours)
alpha_hat_local <- local.fit$coefficients[1]
beta_hat_local <- local.fit$coefficients[2]
points(xloc,alpha_hat_local+beta_hat_local*xloc , pch=19 , cex=1.2)

abline(a=alpha_hat_local, b=beta_hat_local,
       col=adjustcolor("black", 0.75),
       lty=1, lwd=2)


legend("topleft", bty = "n",
       legend=c("Weight 1 for all data","Non-zero weight for pink"),
       lty=c(2,1), lwd=2
)
```

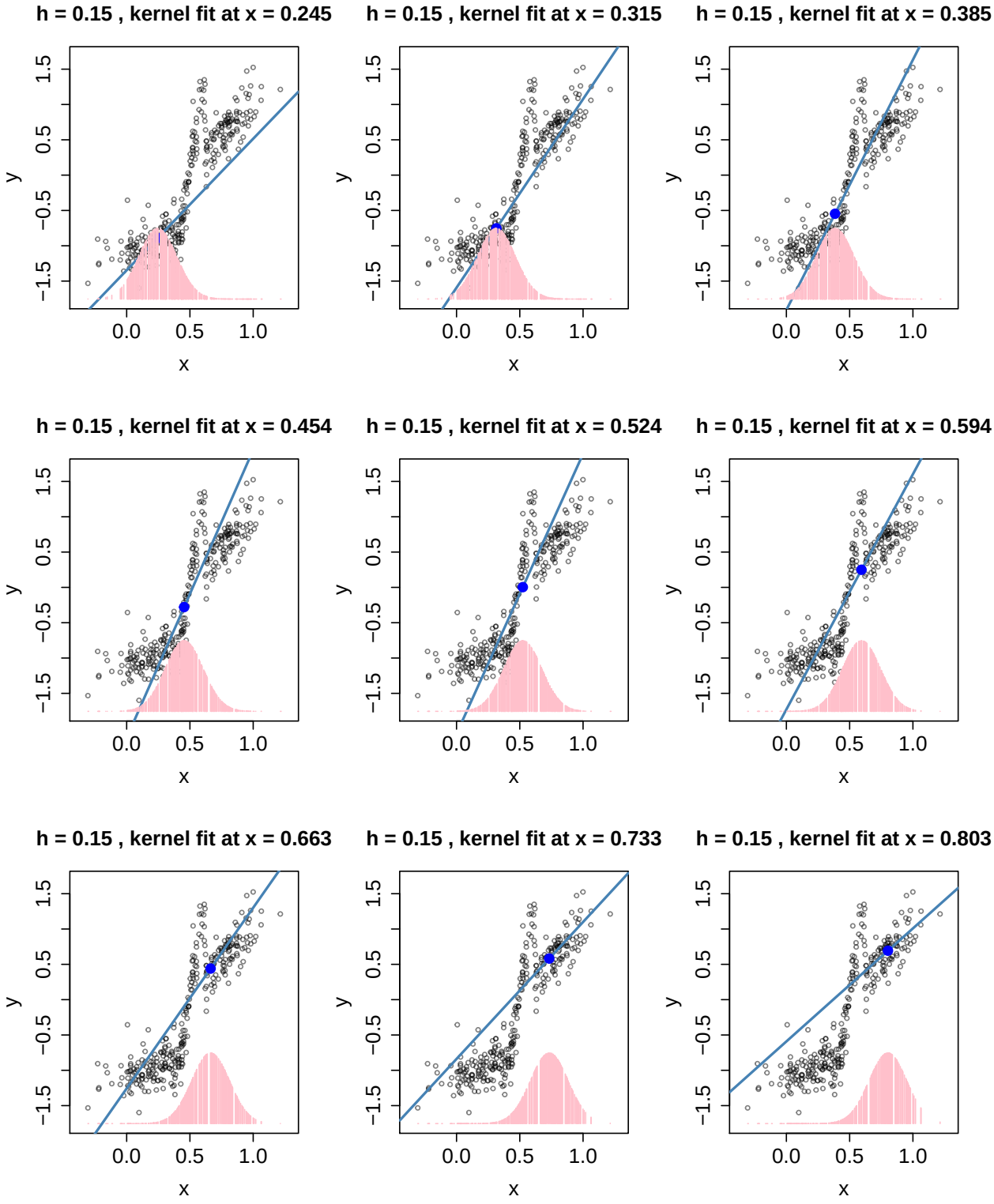


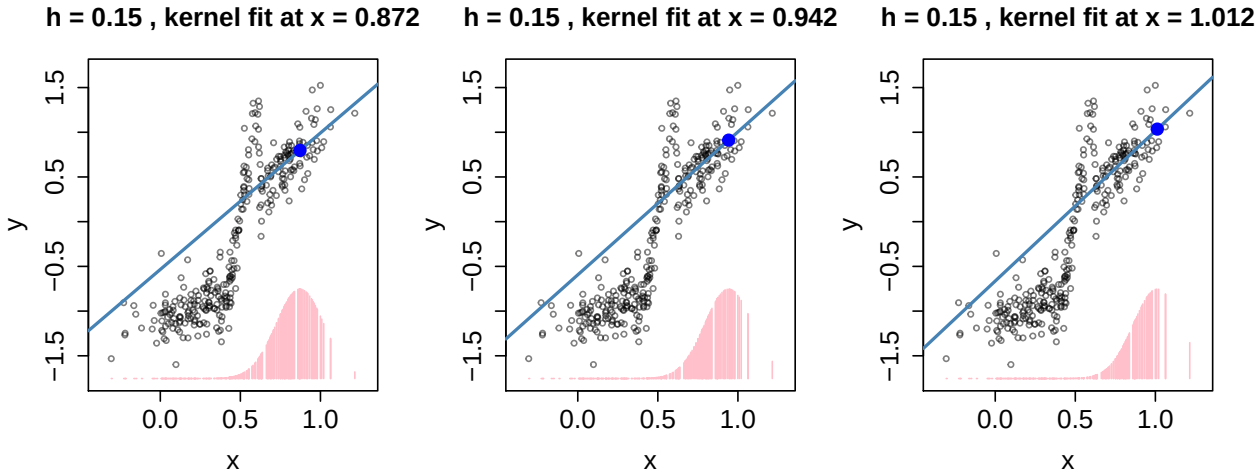
Next, we use non-trivial weights, i.e. a weight function with values more general than 0 and 1.

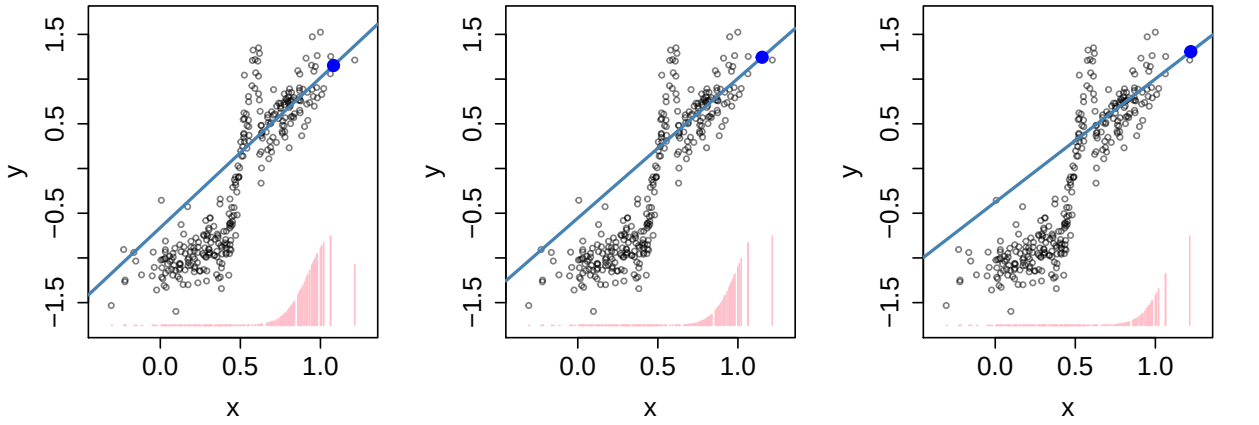
```r
# Construct the Gaussian kernel. Note that bandwidth h to be chosen objectively later
GaussWeight <- function(xloc, x, h=1) {
  # Normal density
  dnorm(x, mean=xloc, sd=h)
}

xrange <- extendrange(x)
locs <- seq(xrange[1], xrange[2], length.out = 25)

# Generating the local weighted fits at x values in locs vector.

# The bandwidth corresponds to the standard
# deviation to be used in
# the Gaussian density function.
# For our data (whose x values lie between 0 and 1)
# a bandwidth of 0.25 will give non-zero weights to all of the
# (xi, yi) pairs.
h0=0.15 #this is the h in the kernel function. The smaller this is, the more local the fits are

pred=c()
par(mfrow=c(3,3))
#This loop creates the images used in the animations
#I showed during the lecture.
for(loc.counter in 1:length(locs)){

  #location at which we are estimating
  xloc <- locs[loc.counter]

  #Calculating the weights
  wts <- GaussWeight(xloc, x, h=h0) / sum(GaussWeight(xloc, x, h=h0))

  # And use the weights to fit a line
  fitw <- lm(y~x, data=SimulatedData, weights=wts)

  # plotting the data
  yrange <- extendrange(y)
  plot(x,y, xlim=xrange,ylim=yrange,
       pch=1, cex=0.7,
       main = paste("h =", as.character(h0),", kernel fit at x =",as.character(round(xloc,3))),
       cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
       col=adjustcolor("black" , alpha=0.5)
  )

  #  With the new fit added
  #
  abline(fitw$coefficients, col="steelblue", lwd=2)

  # The line was fitted with weights centred around x = xloc
  # We can get the fitted value there as
  pred[loc.counter] <- predict(fitw, newdata=data.frame(x=xloc))

  # And plot it for emphasis
  points(xloc, pred[loc.counter], col="blue", pch=19, cex=1.5)
```

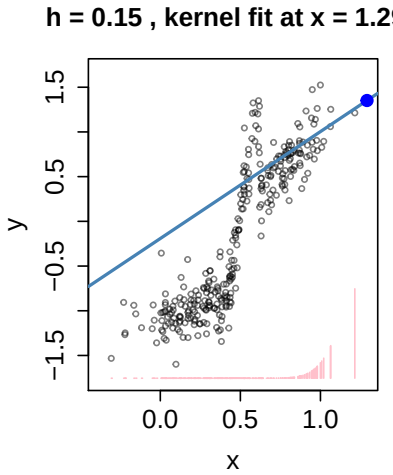
```
  # We can add the weights to the display as little
  # a vertical line at each point
  # We need to know the range of the ys
  ybottom <- min(yrange)
  #
  # Draw the weight lines
  for (i in 1:length(wts)) {
    lines(x=rep(x[i],2),
          y=c(ybottom, ybottom + wts[i]/(max(wts))),
  # the dividing by 3*max(wts) is for plotting purposes only
          col="pink",
          lty=1)
  }
}
```


h = 0.15 , kernel fit at x = −0.382
h = 0.15 , kernel fit at x = −0.313
h = 0.15 , kernel fit at x = −0.243
h = 0.15 , kernel fit at x = −0.173
h = 0.15 , kernel fit at x = −0.103
h = 0.15 , kernel fit at x = −0.034
h = 0.15 , kernel fit at x = 0.036
h = 0.15 , kernel fit at x = 0.106
h = 0.15 , kernel fit at x = 0.175
x
y
0.0
0.5
1.0
−1.5
−0.5
0.5
1.5


h = 0.15 , kernel fit at x = 0.245
h = 0.15 , kernel fit at x = 0.315
h = 0.15 , kernel fit at x = 0.385
h = 0.15 , kernel fit at x = 0.454
h = 0.15 , kernel fit at x = 0.524
h = 0.15 , kernel fit at x = 0.594
h = 0.15 , kernel fit at x = 0.663
h = 0.15 , kernel fit at x = 0.733
h = 0.15 , kernel fit at x = 0.803
y
x
−1.5
−0.5
0.5
1.5
0.0
0.5
1.0

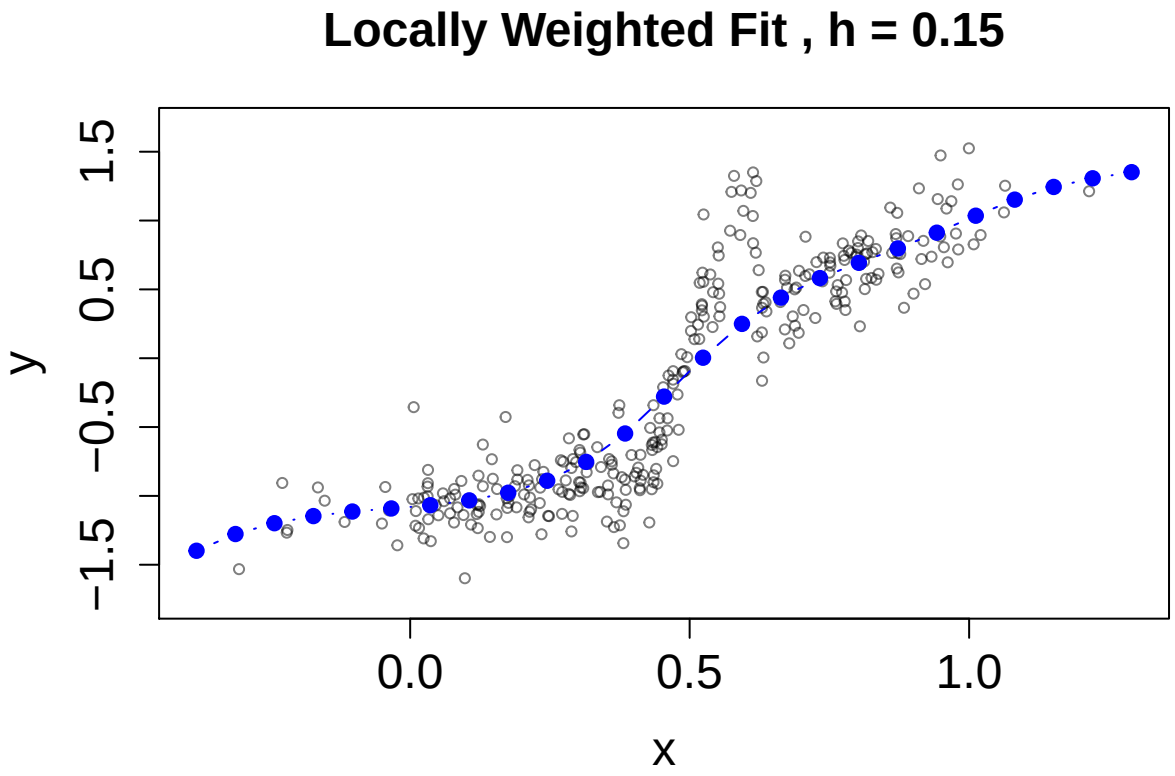






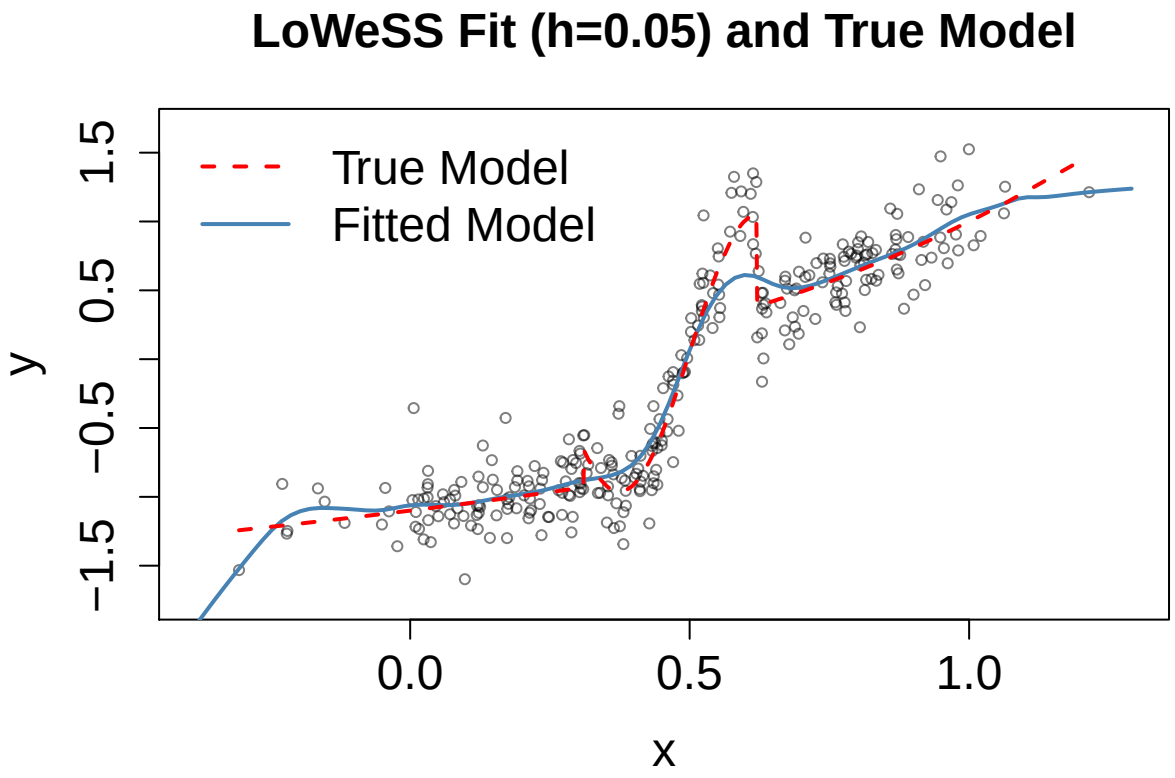

```
# plot the data and superimpose the fit
par(mfrow=c(1,1))
plot(x,y, xlim=xrange,ylim=yrange,
     pch=1, cex=0.7,
     main =  paste("Locally Weighted Fit , h =",as.character(h0)) ,
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
```

```
     col=adjustcolor("black" , alpha=0.5)
)
# Superimposing the fitted model
lines(pred~locs , col="blue", pch=19,type="b")
```

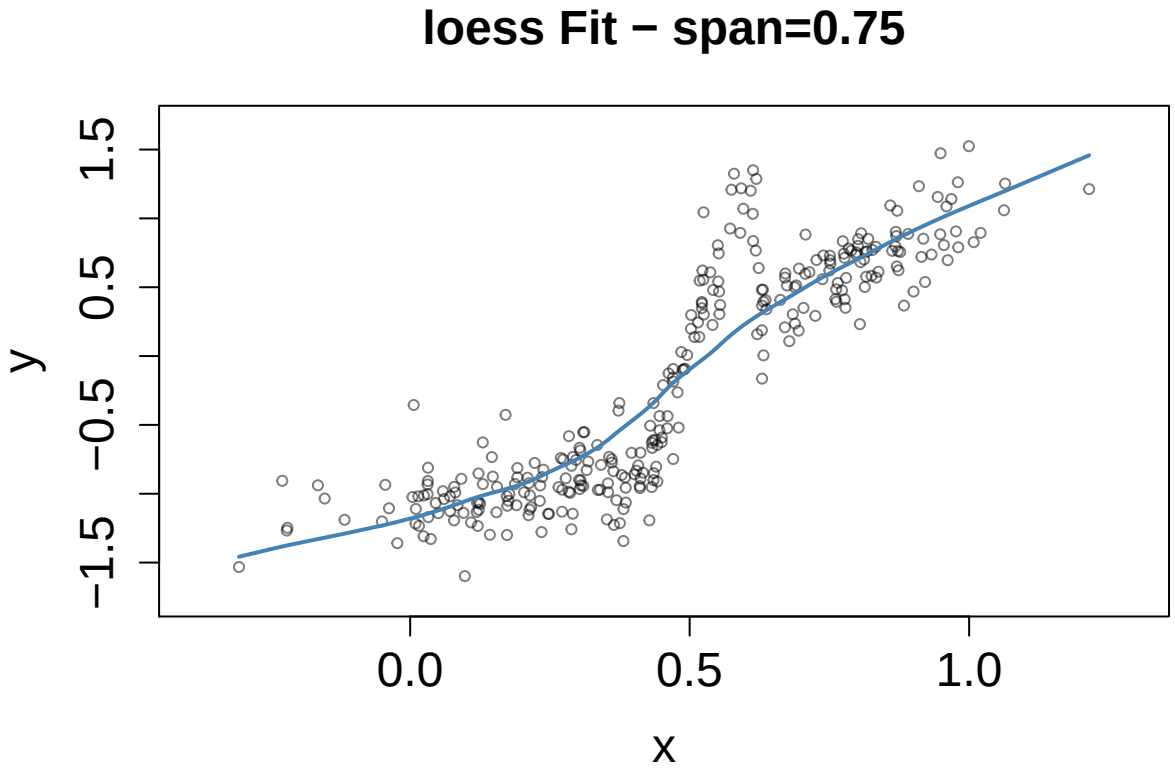



```
# Putting together the fitted model and the ture model
# Calculate the model for h=0.05 and and at 100 values of x
locs <- seq(xrange[1], xrange[2], length.out = 100)
# Generating the local weighted fits at x values in locs vector.
pred=c()
h0=0.05

for(loc.counter in 1:length(locs)){
  xloc <- locs[loc.counter]
  wts <- GaussWeight(xloc, x, h=h0) / sum(GaussWeight(xloc, x, h=h0))
  fitw <- lm(y~x, data=SimulatedData, weights=wts)
  pred[loc.counter] <- predict(fitw, newdata=data.frame(x=xloc))
}
# plot the data and superimpose the fit
plot(x,y, xlim=xrange,ylim=yrange,
     pch=1, cex=0.7,
     main =  paste0("LoWeSS Fit (h=",as.character(h0),")", " and True Model") ,
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
lines(locs, pred, col="steelblue", lwd=2)

xordered <- sort(x)
mu.x =  mu(xordered) # This is the true model (without the noise)
```

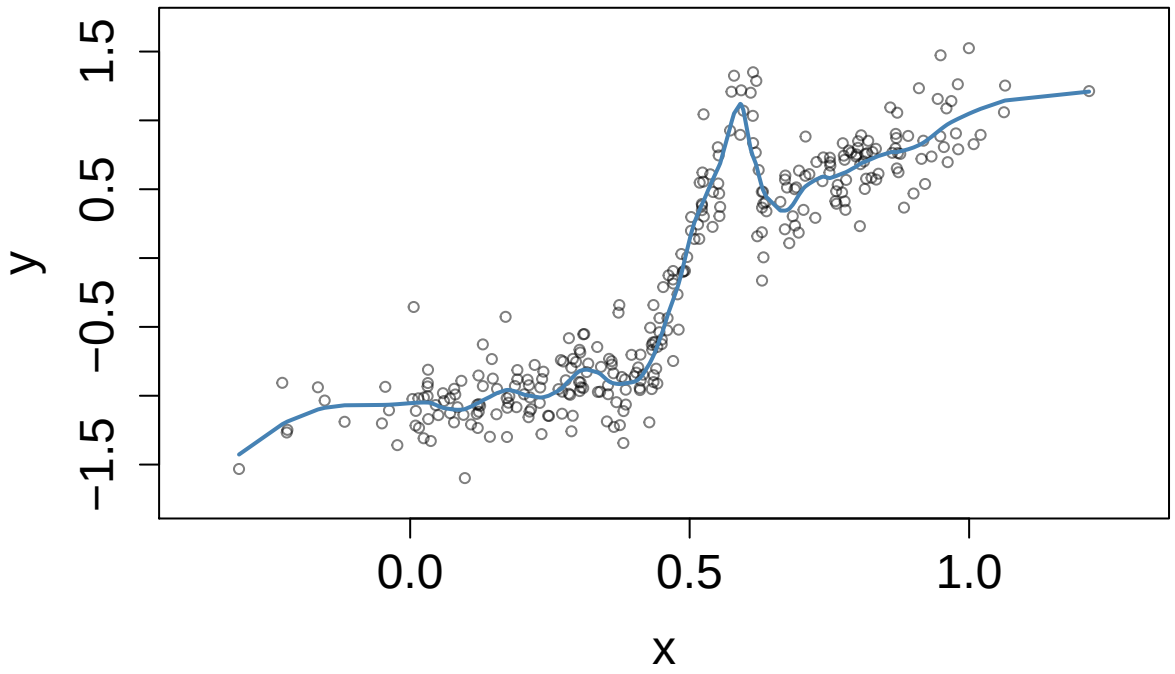
```
lines(xordered, mu.x, col="red", lwd=2 , lty=2)
legend("topleft",  bty='n', cex=1.5,
       legend = c("True Model", "Fitted Model"),
       col = c("red", "steelblue"),
       lty=c(2, 1), lwd=c(2,2),
       text.width = 1.5
)
```




```
# Using loess function
plot(x,y, xlim=xrange,ylim=yrange,
     pch=1, cex=0.7,
     main =  paste0("loess Fit - span=0.75") ,
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
fit <- loess(y~x, degree=1, data=SimulatedData)
# Get the values predicted by the loess
pred <- predict(fit)
#
# Get the order of the x
Xorder <- order(x)
#
lines(x[Xorder],pred[Xorder], lwd=2, col="steelblue")
```

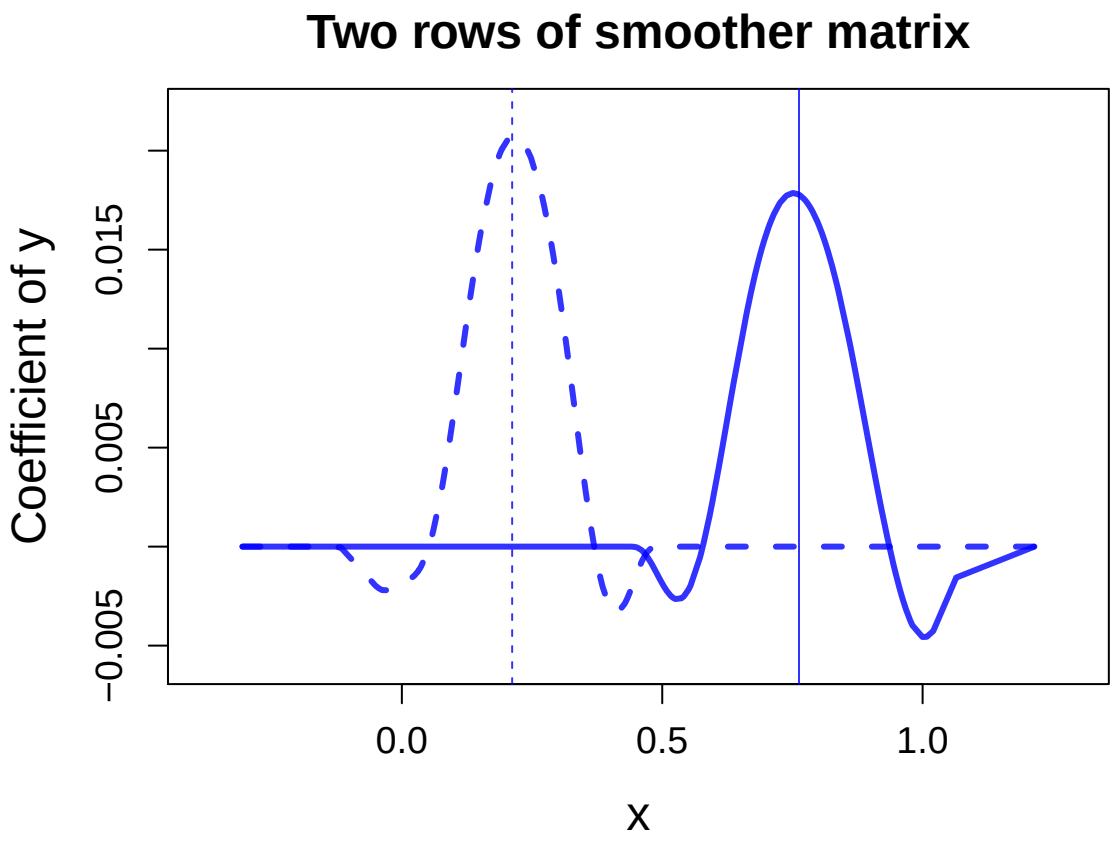



```r
# Now for span=0.15
plot(x,y, xlim=xrange,ylim=yrange,
     pch=1, cex=0.7,
     main =  paste0("loess Fit - span=0.15") ,
     cex.axis=1.5 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("black" , alpha=0.5)
)
fit2 <- loess(y~x, span=0.15, data=SimulatedData)
# Get the values predicted by the loess
pred <- predict(fit2)
#
# Get the order of the x
Xorder <- order(x)
#
lines(x[Xorder],pred[Xorder], lwd=2, col="steelblue")
```



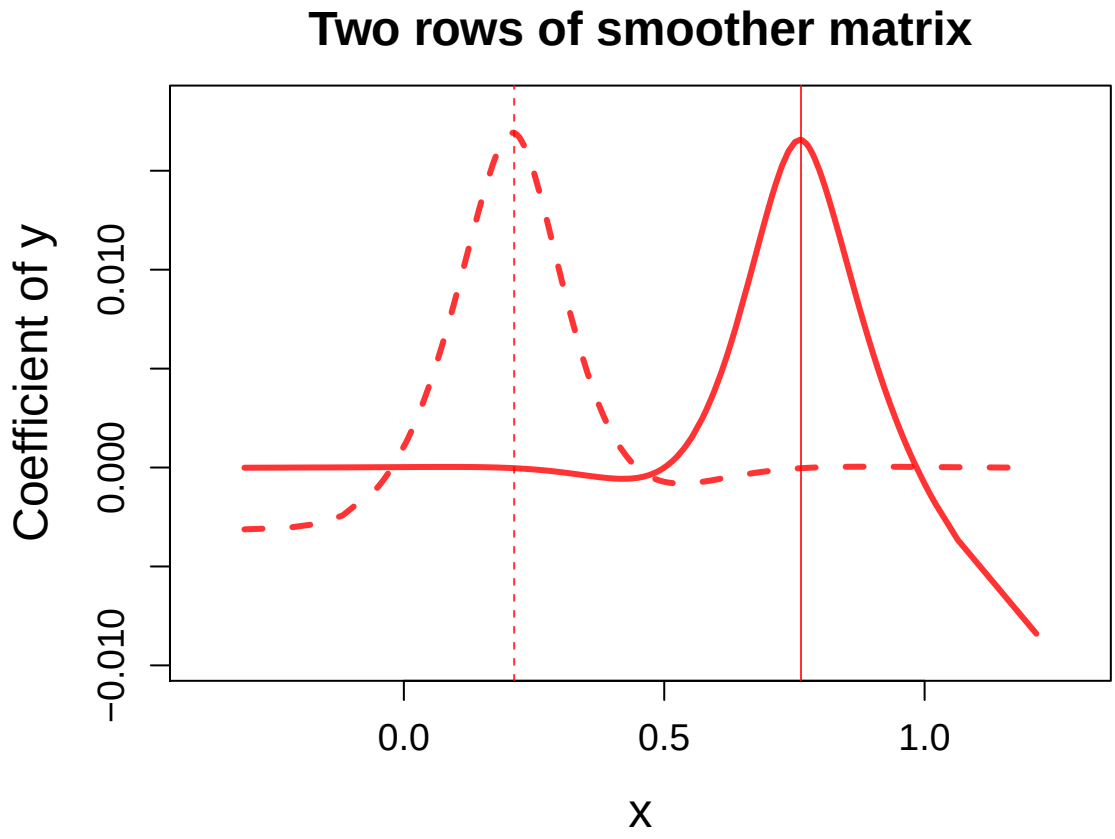

```r
# A function to compute the smoother matrix that corresponds to a loess
smootherMatrixLoess <- function(x,
                                span=NULL,
                                enp.target=NULL,
                                ...) {
  n <- length(x)
  S <- matrix(0, n, n)
  for (i in 1:n) {
    ei = rep(0, n)
    ei[i] <- 1
    # insert the fit into the i'th column of S
    if (is.null(span) & is.null(enp.target))
    {
      S[,i]  <- predict(loess(ei ~ x, ...))
    } else {
      if (is.null(span)) {
        S[,i]  <- predict(loess(ei ~ x,
                                enp.target=enp.target,
                                ...))

      } else {
        S[,i]  <- predict(loess(ei ~ x,
                                span=span,
                                ...))

      }

    }
```

```
  }
  # For loess, the smoother matrix need
  # not be a symmetric matrix
  S
}




S_l <- smootherMatrixLoess(SimulatedData$x, enp.target=7)
S <- S_l

# We now just look at the coefficients for any one of the ys.
#
# Let's arbitrarily pick a couple of rows at random
#set.seed(12231299)
set.seed(844)
row <- sample(1:nrow(S), 2)

# First get the order of the x
xOrder <- order(SimulatedData$x)

# now plot the first row's coefficients ordered by x value
i <- 1
par(mar=c(4,5,3,3))
plot(SimulatedData$x[xOrder], S[row[i],][xOrder], type="l",
     xlim=extendrange(SimulatedData$x), ylim=extendrange(S[row,]),
     xlab="x", ylab="Coefficient of y",
     main="Two rows of smoother matrix",
     lwd=3,
     cex.axis=1.2 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("blue", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("blue", 0.8))

# And another row
i <- 2
lines(SimulatedData$x[xOrder], S[row[i],][xOrder],
      lwd=3, lty=2,
      col=adjustcolor("blue", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("blue", 0.8),
       lty=2)
```



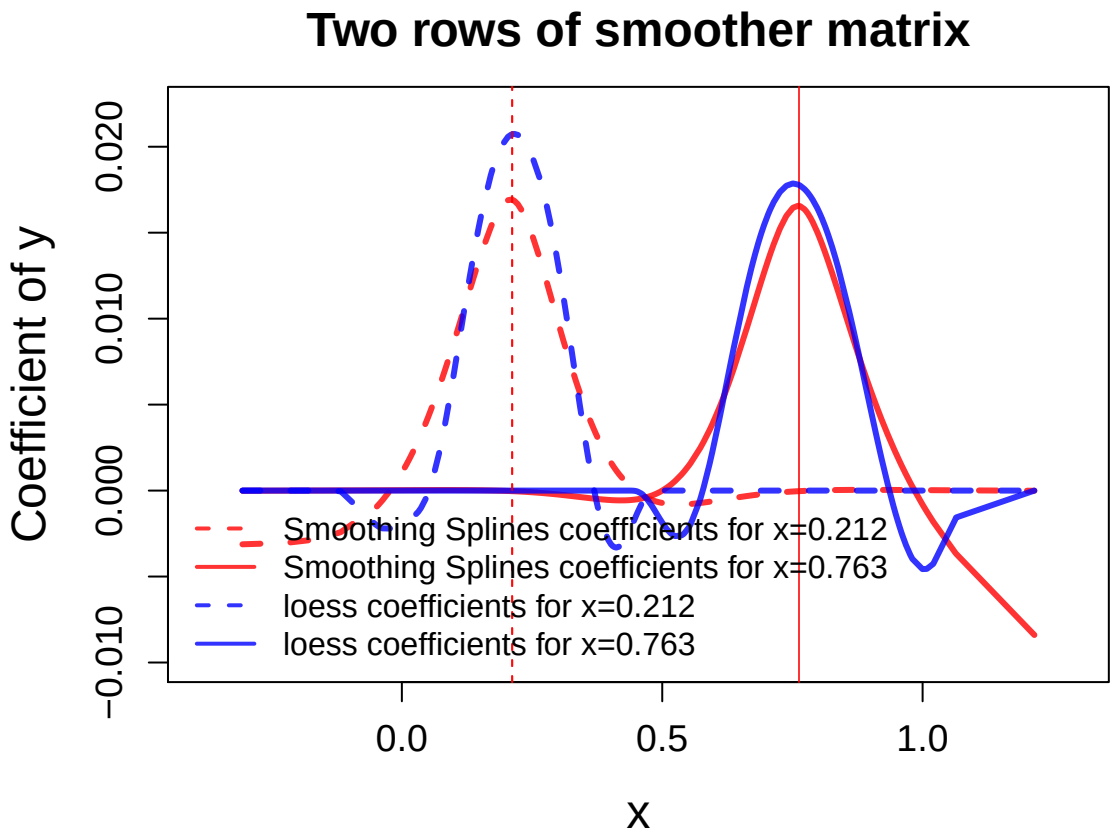

```
# A function to compute the smoother matrix that corresponds to smoothing splines
smootherMatrix <- function(x,df) {
  n <- length(x)
  S <- matrix(0, n, n)
  for (i in 1:n) {
    ei = rep(0, n)
    ei[i] <- 1
    # insert the fit into the i'th column of S
    S[,i]  <- predict(smooth.spline(x, ei, df=df), x)$y
  }
  # To make sure the result is (numerically) symmetric
  S <- (S + t(S))/2
  # and return the symmetric matrix
  S
}


S_s <- smootherMatrix(SimulatedData$x, df=7) # smoothing matrix for smoothing splines
S <- S_s


# First get the order of the x
xOrder <- order(SimulatedData$x)

# now plot the first row's coefficients ordered by x value
i <- 1
par(mar=c(4,5,3,3))
```

```r
plot(SimulatedData$x[xOrder], S[row[i],][xOrder], type="l",
     xlim=extendrange(SimulatedData$x), ylim=extendrange(S[row,]),
     xlab="x", ylab="Coefficient of y",
     main="Two rows of smoother matrix",
     lwd=3,
     cex.axis=1.2 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("red", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("red", 0.8))

# And another row
i <- 2
lines(SimulatedData$x[xOrder], S[row[i],][xOrder],
      lwd=3, lty=2,
      col=adjustcolor("red", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("red", 0.8),
       lty=2)
```




```r
############
## Putting the two plots together
############
S <- S_s # smoothing splines smoothing matrix
# First get the order of the x
xOrder <- order(SimulatedData$x)

# now plot the first row's coefficients ordered by x value
i <- 1
par(mar=c(4,5,3,3))
```

```r
plot(SimulatedData$x[xOrder], S[row[i],][xOrder], type="l",
     xlim=extendrange(SimulatedData$x),
     ylim=extendrange(c(range(S_l[row,]),
                        range(S_s[row,]))),
     xlab="x", ylab="Coefficient of y",
     main="Two rows of smoother matrix",
     lwd=3,
     cex.axis=1.2 , cex.main=1.5, cex.lab=1.5,
     col=adjustcolor("red", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("red", 0.8))

# And another row
i <- 2
lines(SimulatedData$x[xOrder], S[row[i],][xOrder],
      lwd=3, lty=2,
      col=adjustcolor("red", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("red", 0.8),
       lty=2)


S <- S_l # Now, switch the smoothing matrix to add the coefficient
         # values of y from loess

# now plot the first row's coefficients ordered by x value
i <- 1
par(mar=c(4,5,3,3))
lines(SimulatedData$x[xOrder], S[row[i],][xOrder],
      lwd=3, lty=1,
      col=adjustcolor("blue", 0.8))

# And another row
i <- 2
lines(SimulatedData$x[xOrder], S[row[i],][xOrder],
      lwd=3, lty=2,
      col=adjustcolor("blue", 0.8))
abline(v=SimulatedData$x[row[i]], col=adjustcolor("red", 0.8),
       lty=2)
legend("bottomleft", bty = "n",
       legend=c("Smoothing Splines coefficients for x=0.212" ,
                "Smoothing Splines coefficients for x=0.763" ,
                "loess coefficients for x=0.212" ,
                "loess coefficients for x=0.763"),
       col=c(adjustcolor("red", 0.8) ,
             adjustcolor("red", 0.8) ,
             adjustcolor("blue", 0.8),
             adjustcolor("blue", 0.8)),
       lty=c(2,1,2,1), lwd=2
)
```




```
####
#### Singular Value Decomposition of S (loess and Smoothing splines)
#####

svd_S_s <- svd(S_s) # svd for the smoothing splines smoother matrix
sum(svd_S_s$d)
```

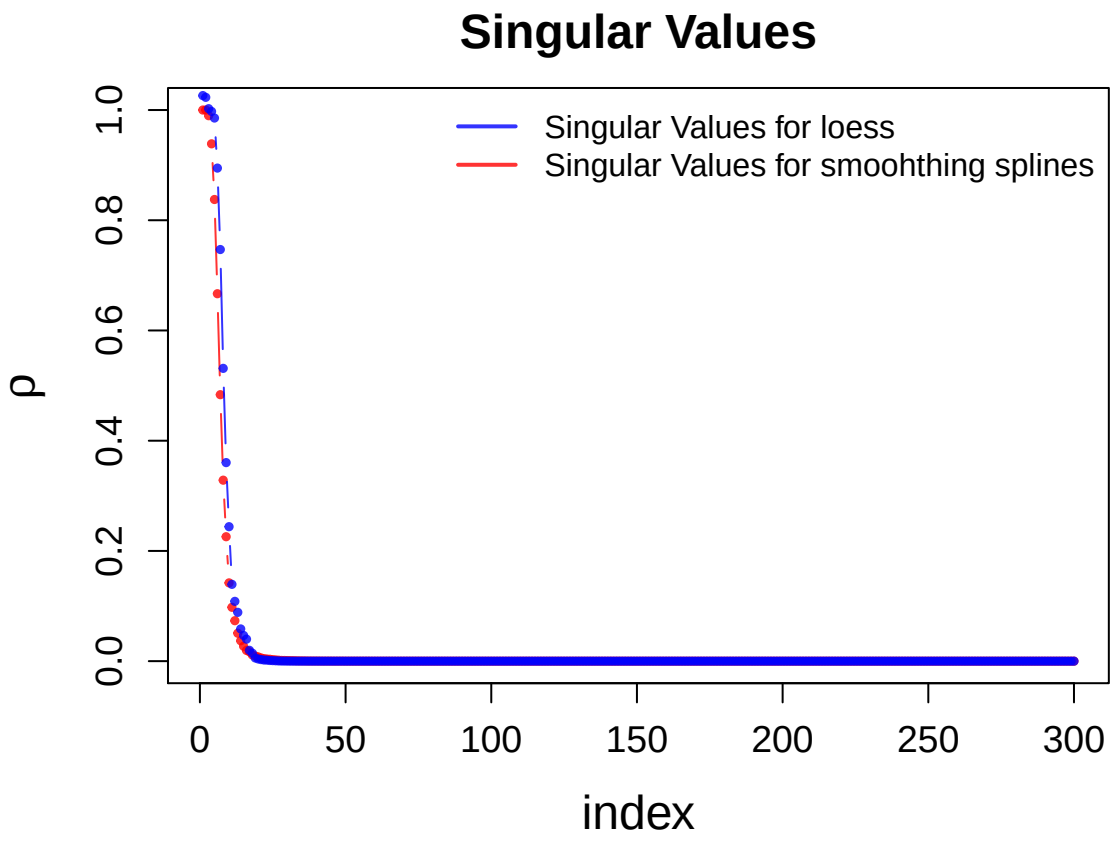
```
## [1] 6.998441
```

```
svd_S_l <- svd(S_l) # svd for loess smoother matrix
sum(svd_S_l$d)
```

```
## [1] 8.347651
```

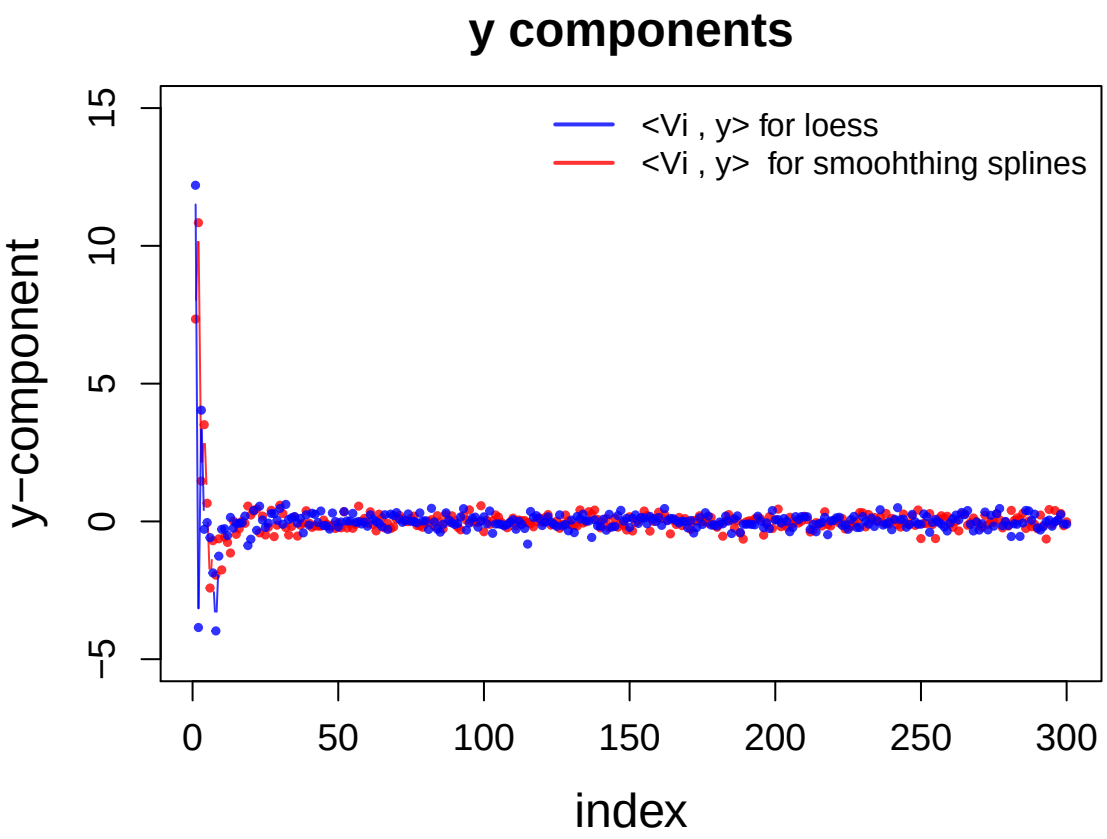
```
####

########
## plot of sigular values
########
par(mar=c(4,5,3,3))
plot(svd_S_s$d, type="b",
     col=adjustcolor("red", 0.8),
     cex=0.5, pch=19,
     cex.axis=1.2 , cex.main=1.5, cex.lab=1.5,
     main="Singular Values", xlab="index", ylab=expression(rho))
lines(svd_S_l$d, type="b",
      col=adjustcolor("blue", 0.8),
      cex=0.5, pch=19)
```

```
legend("topright", bty = "n",
       legend=c("Singular Values for loess" ,
                "Singular Values for smoohthing splines"),
       col=c(adjustcolor("blue", 0.8) ,
             adjustcolor("red", 0.8)),
       lty=c(1,1), lwd=2
)
```



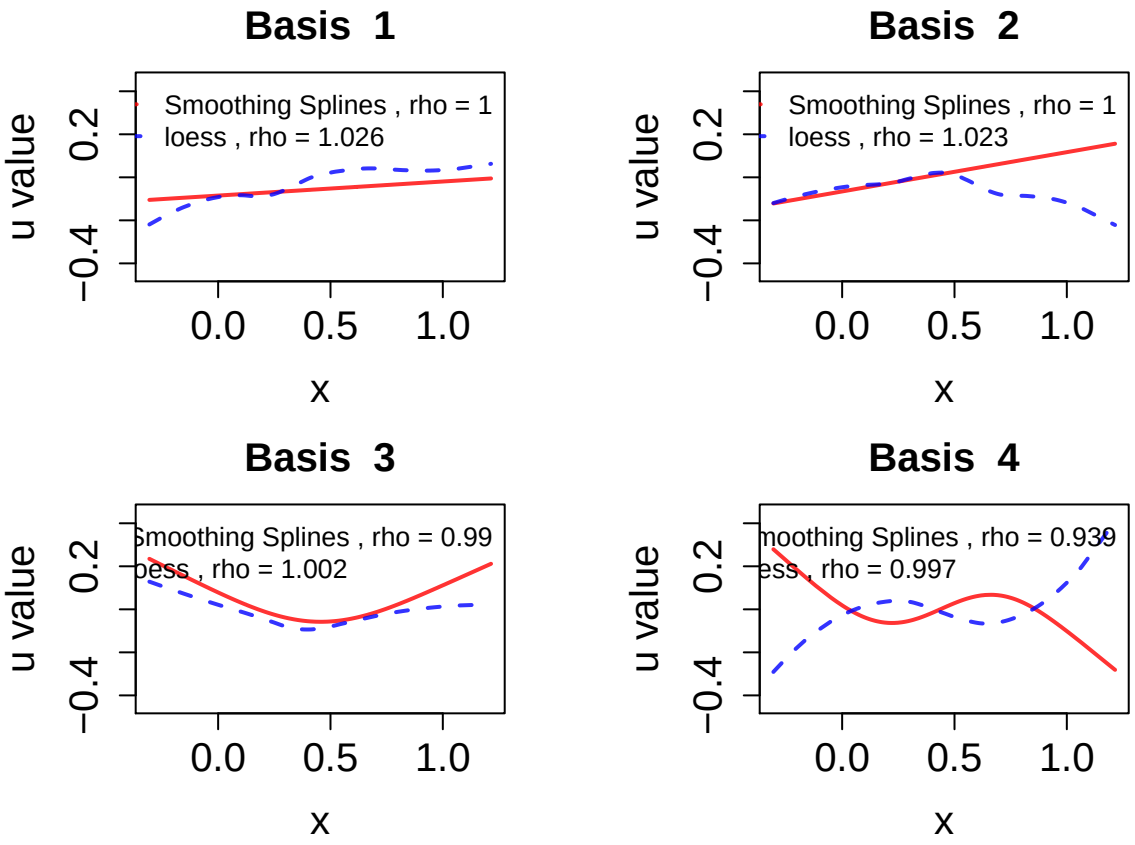

```
########
## plot of y components
########
plot(t(svd_S_s$v) %*% SimulatedData$y,
     col=adjustcolor("red",0.8),
     cex=0.5, pch=19, ylim=c(-5,15),
     cex.axis=1.2 , cex.main=1.5, cex.lab=1.5,
     type="b", main="y components",
     xlab="index", ylab="y-component")
points(t(svd_S_l$v) %*% SimulatedData$y,
       col=adjustcolor("blue",0.8),
       cex=0.5, pch=19,
       type="b")
legend("topright", bty = "n",
       legend=c("<Vi , y> for loess" ,
                "<Vi , y>  for smoohthing splines"),
       col=c(adjustcolor("blue", 0.8) ,
             adjustcolor("red", 0.8)),
       lty=c(1,1), lwd=2
```

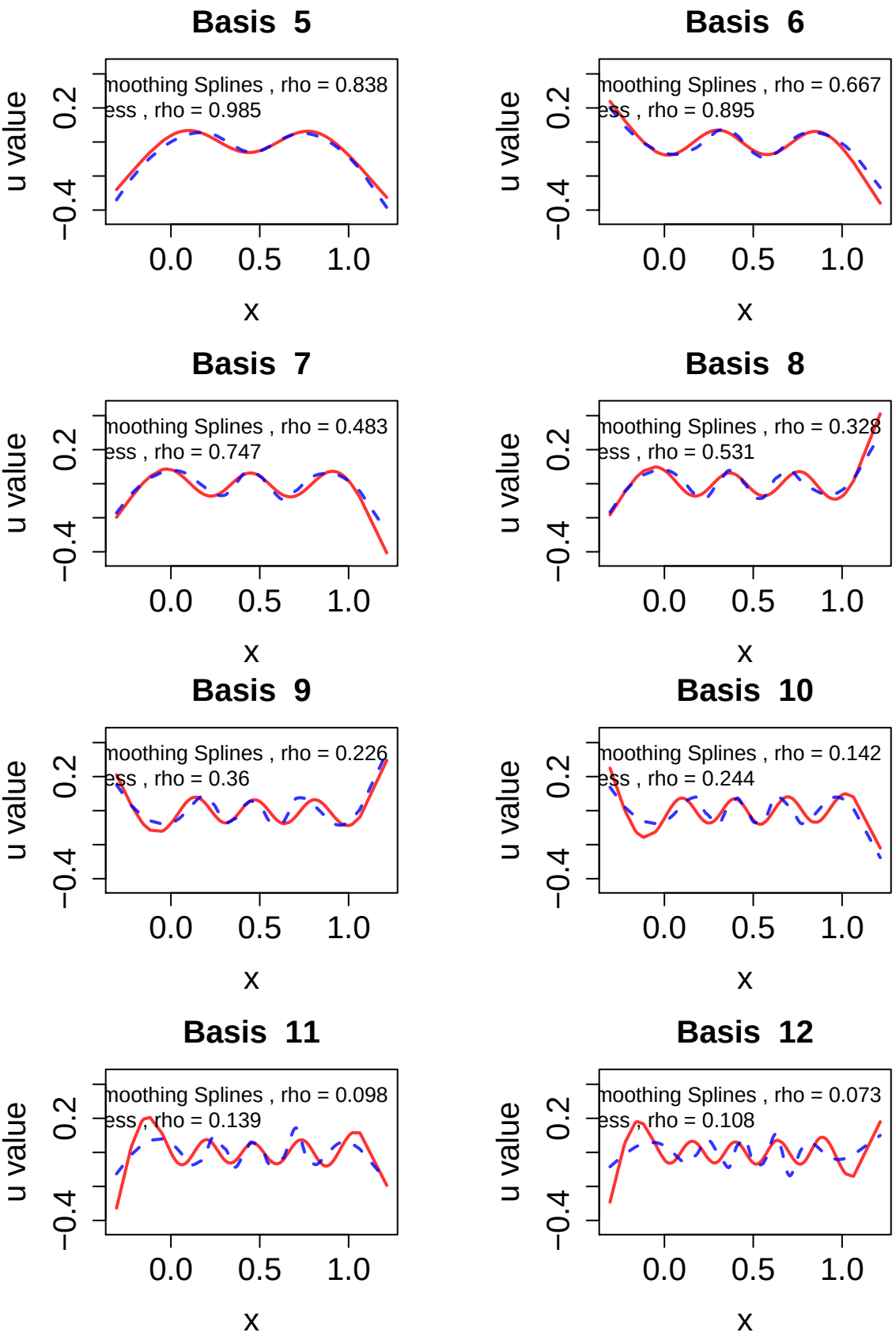
```
)
```




```
####
#### Plotting the basis functions
####
par(mfrow=c(2,2))
k <- 12
ylim <- extendrange(rbind(svd_S_s$u[,1:k],svd_S_l$u[,1:k]))

for (i in 1:k) {
  plot(SimulatedData$x[xOrder],svd_S_s$u[xOrder,i],
       type="l", ylim=ylim, col=adjustcolor("red", 0.8),
       cex.axis=1.5 , cex.main=1.5, cex.lab=1.5, lwd=2,
       main=paste("Basis ", i),
       xlab="x", ylab="u value")

  lines(SimulatedData$x[xOrder],svd_S_l$u[xOrder,i],
        lty=2,col=adjustcolor("blue", 0.8),lwd=2,
  )

  legend("topright", bty = "n",
         legend=c(paste("Smoothing Splines", ", rho =",as.character(round(svd_S_s$d[i],3))) ,
                  paste("loess", ", rho =",as.character(round(svd_S_l$d[i],3)))),
         col=c(adjustcolor("red", 0.8) ,
               adjustcolor("blue", 0.8)),
         lty=c(2,1), lwd=2
  )
```

```
}
```


Basis 1
Smoothing Splines , rho = 1
loess , rho = 1.026
u value
−0.4
0.2
0.0
0.5
1.0
x
Basis 2
Smoothing Splines , rho = 1
loess , rho = 1.023
u value
−0.4
0.2
0.0
0.5
1.0
x
Basis 3
Smoothing Splines , rho = 0.99
oess , rho = 1.002
u value
−0.4
0.2
0.0
0.5
1.0
x
Basis 4
moothing Splines , rho = 0.939
ess , rho = 0.997
u value
−0.4
0.2
0.0
0.5
1.0
x


Basis 5
moothing Splines , rho = 0.838
ess , rho = 0.985
Basis 6
moothing Splines , rho = 0.667
ess , rho = 0.895
Basis 7
moothing Splines , rho = 0.483
ess , rho = 0.747
Basis 8
moothing Splines , rho = 0.328
ess , rho = 0.531
Basis 9
moothing Splines , rho = 0.226
ess , rho = 0.36
Basis 10
moothing Splines , rho = 0.142
ess , rho = 0.244
Basis 11
moothing Splines , rho = 0.098
ess , rho = 0.139
Basis 12
moothing Splines , rho = 0.073
ess , rho = 0.108
u value
x
0.0
0.5
1.0
−0.4
0.2